A Problem Decomposition Scheme for Distributed Problem Solving

Yasuhiko KITAMURA, Shoji TATSUMI, and Takaaki OKUMOTO
Faculty of Engineering, Osaka City University
3-3-138 Sugimoto, Sumiyoshi-ku, OSAKA 558, JAPAN
{kitamura, tatsumi, okumoto}@info.osaka-cu.ac.jp

S. Misbah DEEN
DAKE Centre, Keele University
Keele, Staffordshire, ST5 5BG, UK
misbah@cs.keele.ac.uk

Abstract

To deal with large-scale and/or geographically distributed problems, a number of expert systems should be combined through communication networks, and distributed problem solving (DPS) techniques are required to make such a group of expert systems works effectively. In DPS, if a given problem can be decomposed and allocated properly, the performance will be increased because of high parallelism and small communication overhead. Most of DPS techniques so far have been using such problem decomposition knowledge in explicit or implicit ways, and such knowledge is assumed to be given a priori. In this paper, we proposed an automated problem decomposition scheme based on knowledge hierarchy which can be applied to problem domains where problem decomposition knowledge is not given a priori.

1 Introduction

To deal with large-scale and/or geographically distributed problems, a number of expert systems with knowledge bases should be combined through communication networks, and distributed problem solving (DPS) techniques are required to make such a group of expert systems works effectively.

In DPS, problem solvers called *agents* are distributed and a given problem to be solved needs to be decomposed and allocated to all or some of them. For example, Contract Net Protocol [8] is a communication protocol to assign problems to appropriate agents based on negotiations among agents. In this framework, problems are assumed to be decomposed a priori by users or designers in an explicit manner. Functionally/Accurate Cooperative(FA/C) DPS system [6] is another canonical example of DPS technique, but problems are also assumed to be assigned to agents implicitly before problem solving starts.

There has been little work that addresses automated problem decomposition [2] although it shows a crucial aspect for DPS. If a given problem can be decomposed and allocated properly, the performance will be increased because of high parallelism and small communication overhead among agents.

In this paper, we proposed an automated problem decomposition scheme based on knowledge hierarchy which can be applied to problem domains where problem decomposition knowledge is not given a priori. In our scheme, when an agent receives a problem, referring the knowledge hierarchy, it autonomously decomposes the problem and allocates the decomposed subproblems to available agents. Moreover, for cases where there are several alternatives to decompose a problem, we propose several selection strategies to reduce them to one.

In section 2, we give a rigid formulation of DPS based on the state space graph representation. In section 3, we present a distributed inference scheme called diffusing search [4] which is the base of our proposed scheme. When the given problem is a large-scale one, without problem decomposition, agents with this inference scheme have to search a huge search space generating a big amount of communication which is almost intractable. To cope with this drawback, we propose a problem decomposition scheme using an example of travel planning in section 4.

2 Formulation of DPS

In this section, we present a formulation of DPS based on the state space graph representation [1].

2.1 Problem and Solution

Non-AI problem solving can be depicted as a deterministic process where the sequence of operators for problem solving has been determined beforehand unambiguously, for example, in a form of program. The process thus simply follows the sequence of operators and there are no alternative operators to choose. On the other hand, AI problem solving can be depicted as a non-deterministic process. The process is required to choose an operator out of multiple alternatives and the choice

may not be always correct. When it makes a wrong choice, it may lead to getting stuck and a way to escape from that is to get back to the point of wrong choice and to choose another one out of other alternatives. Therefore, AI problem solving can be formalized as a search process which finds a path on a graph as follows.

Definition 2.1 (Problem and solution) *A* problem p *is given as 4-tuple* $< S, O, s_I, G >$ *where* S *is a non-empty set of* states, $O(\subseteq S \times S)$ *is a set of* operators, *each of them specifies a state conversion,* $s_I(\in S)$ *is the* initial state, $G(\subseteq S)$ *is a set of* goal states. *The tuple* $< S, O >$ *can be viewed as a graph called* state space graph. *A problem is said to be* finite *if the state space graph is finite. A* solution *is given as any path from the initial state to a goal state in the state space graph. In other words, it is a sequence of operators that converts the initial state into a goal state.* □

2.2 Agent, Community, and Knowledge Distribution

DPS deals with problem solving by multiple agents. For example, let us assume a system, which offers international travel information, consisting of multiple agents. Each agent offers information of only one country but it cooperatively works with other agents when it is asked to find an international travel plan. As this example, the knowledge, namely a set of operators, to be required for problem solving is distributed among agents. The local knowledge of each agent therefore can be depicted as a graph which represents the range of state conversions by the agent. More formally, we can define them as follows.

Definition 2.2 (Knowledge) *The local knowledge of agent* a *is denoted by* $K_a =< S_a, O_a >$ *where* S_a *and* $O_a(\subseteq S_a \times S_a)$ *are a set of local states and local operators of agent* a *respectively. A set of agents is called* community *and the knowledge of community* C *is defined as* $K_C =< S_C, O_C >$, *where* $S_C = \bigcup_{a \in C} S_a$ *and* $O_C = \bigcup_{a \in C} O_a$, *in words, that is the sum of local knowledge of agents in the community.* □

We here define several concepts about knowledge distribution in a community: *coverage*, *joint state*, and *redundancy*, that characterize a community, namely a DPS system.

Coverage As a relation between a problem and knowledge of agent or community, we define a concept *coverage*.

Definition 2.3 (Coverage) *For a problem* $p =< S, O, s_I, G >$ *and a community* C, *we say the knowledge of agent* $a(\in C)$, $K_a =< S_a, O_a >$, covers *the problem if* $O \subseteq O_a$. *Likewise, we say the knowledge of community* C, $K_C =< S_C, O_C >$, *covers the problem if* $O \subseteq O_C$. □

Joint State We then introduce a concept, a *joint state*, that is a state shared by multiple agents and that reflects relations between knowledge of an agents and that of another.

Definition 2.4 (Joint state) *For knowledge of agent* a, $K_a =< S_a, O_a >$, *and that of* $b(\neq a)$, $K_b =< S_b, O_b >$, *if there exists a state* s *which satisfies* $s \in S_a \cap S_b$, *then* s *is called a* joint state *of agent* a *and* b. □

Definition 2.5 (Joint knowledge) *Joint knowledge of agent* a, JK_a, *is denoted as* $< S_a, C >$ *where* C *is a community, a set of agents. This joint knowledge is said to be* complete *if*

$$\forall s \in S_a, \forall b \in C - \{a\} : s \in S_a \cap S_b \Leftrightarrow (s, b) \in JK_a.$$ □

In words, a complete joint knowledge means the agent knows all the joint states and all the agents which share them.

Redundancy Finally, we define a concept *redundancy*, which classifies overlap of knowledge among agents, as follows.

Definition 2.6 (Redundancy) *For a community* C, *if it satisfies* $\forall a, b \in C : a \neq b \Rightarrow O_a \cap O_b = \emptyset$, *the community is said to be* non-redundant, *and if it satisfies* $\forall a, b \in C : O_a = O_b$, *the community is said to be* completely redundant. *Otherwise, the community is said to be* partially redundant. □

3 Diffusing Search

The diffusing search [4] is an inference method for DPS based on distributed search. The search proceeds as in a uniform manner and the completeness of search is guaranteed.

3.1 Diffusing Search System

Before we describe the diffusing search algorithm, we clarify the definition and assumptions on systems on which the algorithm is executed.

Definition 3.1 (Diffusing search system) *A diffusing search system is a community composed of a finite number of agents. One of agents is called the* root agent a_{root} *which submits the initial problem.* [1] *Each agent can communicate with any other agents only by exchanging messages and no message are lost during communication.* □

Generally speaking, the characteristics, the capability of producing solutions or the efficiency of a diffusing search system vary depending on its knowledge distribution. In this section, we assume a diffusing search system which satisfies the following assumptions.

Assumption 3.2 *Problems are finite and covered by the diffusing search system.* □

Assumption 3.3 *Each agent has complete joint knowledge.* □

Assumption 3.4 *The diffusing search system is non-redundant.* □

[1] A user can be the root agent.

3.2 Diffusing Search Algorithm

The diffusing search algorithm is based on distributed search and can be viewed as a combination of intra-agent *local search* and inter-agent *global search*. A local search is executed by a single agent to find partial solution paths inside its local knowledge and the global search specifies message exchange among agents to control local searches to obtain a global solution path. So far these two search algorithms have been investigated independently in AI [7] and distributed algorithm [3] respectively. We here describe how to integrate these algorithms as the diffusing search algorithm.

3.2.1 Local Search

The local search is based on classical search algorithm [7] and executed by a single agent. A classical search algorithm is initiated by giving the initial state and terminated with $<$ `success` $>$ if it finds a goal state or with $<$ `failure` $>$ if it fails. In the diffusing search algorithm, the local search can be terminated with a joint state which is a candidate of intermediate state on a path to goal state when the agent cannot find the goal state by itself, so we extend the classical algorithm to the one which returns one more result $<$ `continue`;$JS>$ where JS is a set of joint states. The algorithm of an agent a is as follows.

- Database
  - $OP(\subseteq S_a)$. A set of open states that are candidates to be expanded.
  - $CL(\subseteq S_a)$. A set of closed states that have been expanded.
  - $PN(\subseteq S_a \times S_a)$. A set of pointers between two states. $(s_i, s_j) \in PN$ means a path from s_i to s_j has been found.
  - $JS(\subseteq S_a)$. A set of joint states that have been found.
- Function
  - $SS : 2^{OP} \rightarrow OP$. A function to choose a state from OP to expand.
- Algorithm: `local_search`(s_{LI})

Step1 Put s_{LI} in OP. Set $JS = \emptyset$.

Step2 If $OP = \emptyset$ then

(a) If $JS = \emptyset$ then end with $<$ `failure` $>$ else end with $<$ `continue`;$JS>$.

Step3 $s \leftarrow SS(OP)$. Add s in CL. Remove s from OP.

Step4 If $s \in G$ then end with $<$ `success` $>$.

Step5 For each of $s' \in O_a(s) \cap O(s)$,

(a) If $s' \notin CL \cup OP$ then add s' in OP and (s, s') in PN.

Step6 If $JK_a(s) \neq \emptyset$ then add s in JS.

Step7 Go to Step2.

This algorithm is initiated by giving s_{LI} as the local initial state and terminate with a *local search tree*, denoted by $< CL, PN >$ which has the local initial state s_{LI} as the root node.

3.2.2 Global Search

The global search algorithm specifies message passing among agents to invoke local searches to obtain a global solution. We show below the global search algorithm of agent a as an extension of a conventional distributed search algorithm [3].

- Message
  - <`search`;s>. To request a problem $< s, G >$.
  - <`found`>. To notify that the agent has found a goal state.
- Database
  - $CJ(\subseteq S_a)$. A set of joint states that have been found.
  - $RQ(\subseteq S_a \times C)$. A set of pair lists of a local initial states and an agent which requested it. $(s, b) \in RQ$ means that agent b requested a problem $< s, G >$ to agent a.
- Algorithm

<Root agent>

State1 When initializing

Step1 Send $<$ `search`;$s_I>$ to agent a_I which has the initial state s_I in its local knowledge.

State2 When receive $<$ `found` $>$.

Step1 Halt.

<General agents>

State1 When receive $<$ `search`;$s>$ from a_p

Step1 If $s \in CJ$ then wait else add (s, a_p) in RQ.

Step2 Execute `local_search`(s).

(a) If it ends with $<$ `success` $>$ then send $<$ `found` $>$ to all agents.

(b) If it ends with $<$ `continue`;$JS>$ then for each of $s_c \in JS$, if $s_c \notin CJ$ then add s_c in CJ and send $<$ `search`;$s_c>$ to all $b \in CK_a(s_c)$.

(c) If it ends with $<$ `failure` $>$ then wait.

State2 When receive $<$ `found` $>$

Step1 Halt.

4 A Problem Decomposition Scheme Based on Knowledge Hierarchy

In this section we describe a problem decomposition scheme using travel planning problem as an example.

4.1 Travel Planning

Travel planning problem described here is to find a travel route that is spreading very wide area like an international travel route. The knowledge required to solve this problem is very large, various according to the means of transportation, and geographically distributed. It is almost impossible to maintain this knowledge in a single centralized knowledge base and it is reasonable to solve this problem by cooperations of distributed knowledge bases, *agents*, each of which maintains a part of the knowledge according to the means of transportation (bus, train, ship, aircraft, etc) or the domain it covers (inter-country, country, county, town, etc).

This travel planning problem can be viewed as finding a path from a starting place to a goal place on a state space

graph where places and transportations between two places are represented as states and operators respectively. We, therefore, can use the diffusing search scheme to find such a path from the knowledge that is distributed among agents, but actually we would face a big amount of communication overhead among agents if we use the diffusing search algorithm naively because the state space graph would be very big and be maintained by many agents.

To reduce the communication, we here propose a problem decomposition scheme by using knowledge hierarchy and further improve the decomposition by using several selection strategies. By this scheme, an agent does not need to submit messages for all the joint states that are connected from the local initial state, but submits a message for a single selected joint state. Only when the selected joint state does not produce a solution, a succeeding message is submitted for another alternative joint state.

4.2 Knowledge Hierarchy

For our world-wide travel planning problem, we extend our representation of a state s into a list consisting of hierarchical *domains* such as $[s^1, s^2, \ldots, s^n]$. We say s^i is domain name at level-i.

For example, we can represent the Heathrow airport as $[\texttt{UK}, \texttt{London}, \texttt{Heathrow}]$, each of which represents the domain name of country, county or state, and town respectively instead of just `Heathrow` as before.

By this extension, we can classify problems according to their level and domain. When a problem $p =< [s_I^1, s_I^2, \ldots, s_I^n], [s_G^1, s_G^2, \ldots, s_G^n] >$ is given, we say p is a problem of domain $[s_I^1, \ldots, s_I^{i-1}]$ at level-i if the following conditions are satisfied. [2]

$$\begin{array}{ll} s_I^j = s_G^j & \text{if } j < i \\ s_I^j \neq s_G^j & \text{if } j = i \end{array}$$

For example, $< [\texttt{UK}, \texttt{Staffs}, \texttt{Keele}], [\texttt{UK}, \texttt{Staffs}, \texttt{Newcastle}] >$ is a problem of domain $[\texttt{UK}, \texttt{Staffs}]$ at level-3. $< [\texttt{UK}, \texttt{London}, \texttt{Heathrow}], [\texttt{Japan}, \texttt{Chiba}, \texttt{Narita}] >$ is a problem of domain `world` at level-1. [3]

We can classify operators as well as problems. We say an operator $o = ([s_p^1, s_p^2, \ldots, s_p^n], [s_q^1, s_q^2, \ldots, s_q^n])$ is an operator of domain $[s_p^1, \ldots, s_p^{i-1}]$ at level-i if it satisfies the following condition.

$$\begin{array}{ll} s_p^j = s_q^j & \text{if } j < i \\ s_p^j \neq s_q^j & \text{if } j = i \end{array}$$

We here define the following concepts.

Definition 4.1 (Separation and Hierarchy) *If every operator of an agent is a level-i operator of the same domain, then the agent is said to be* separated *and called an agent in the domain at level-i. If all the agents in a community are separated, then the community is said to be* hierarchical. □

[2] We assume a problem has only one goal state.

[3] We call the domain name of level-0 `world` as a special case.

When a community is hierarchical, for an agent at level-i, agents and domains at level-$j(< i)$ are called *upper agents* and *upper domains* respectively, and agents and domains at level-$j(> i)$ are called *lower agents* and *lower domains* respectively. Likewise agents and domains at level-$j(= i)$ are called *peer agents* and *peer domains*.

Joint states in an agent also can be categorized according to the level of agent with which they are shared. For a level-i agent, its *upper joint states* are shared with level-$j(< i)$ agents, its *peer joint states* are shared with level-$j(= i)$ agents, and its *lower joint states* are shared with level-$j(> i)$ agents. In general, an upper joint state of agent a to agent b is identical with a lower joint state of agent b to agent a, but as an exception, we introduce *indirect upper joint states*. For an indirect upper joint state, there exists no corresponding lower joint state in the upper agent but a path from it to a lower joint state in the other peer or lower agent at least.

Example 4.2 *We show an example of hierarchical community in Figure 1. In this example, there are five separated agents; agent* `BA` *in world domain at level 1, agents* `BR` *and* `NE` *in* $[\texttt{UK}]$ *domain at level 2, and agent* `PMT` *in* $[\texttt{UK}, \texttt{Staffs}]$ *domain and* `LUG` *in* $[\texttt{UK}, \texttt{London}]$ *domain at level 3. For agent* `NE`*, agent* `BA` *is an upper agent and the upper joint state is* $[\texttt{UK}, \texttt{London}, \texttt{Heathrow}]$*, agent* `BR` *is a peer agent and the peer joint state is* $[\texttt{UK}, \texttt{WestMidland}, \texttt{Birmingham}]$*, agent* `PMT` *is a lower agent and the lower joint state is* $[\texttt{UK}, \texttt{Staffs}, \texttt{Hanley}]$*, and agent* `LUG` *is also a lower agent and the lower joint state is* $[\texttt{UK}, \texttt{London}, \texttt{Heathrow}]$*. Notice for* `LUG`*,* `BA` *can be an upper agent (whose joint state is* $[\texttt{UK}, \texttt{London}, \texttt{Heathrow}]$*) even though the difference between the levels is 2. An example of indirect upper joint state is* $[\texttt{UK}, \texttt{London}, \texttt{Euston}]$ *of* `BR` *to* $[\texttt{UK}, \texttt{London}, \texttt{Heathrow}]$ *of* `BA`*. There is a path connects them in* `LUG`*.*

4.3 Problem Decomposition

When an agent receives a problem, it can solve it if it has a path to the goal state in its local knowledge, or it can decompose it, for example, a problem $< s_I, s_G >$ into $< s_I, s_J >$ and $< s_J, s_G >$ if it has a joint state s_J. In this sense, to decompose a problem can be viewed as to find a joint state.

As we have shown in the previous section, there are three types of joint states; upper, peer, and lower joint states. When a problem $< s_I, s_G >=< [s_I^1, \ldots, s_I^j, \ldots, s_I^n], [s_G^1, \ldots, s_G^j, \ldots, s_G^n] >$ of level-l_p is given to an agent at level-l_a, by observing the level relation between the agent and the problem, we can select one type out of three by using the following rules.

1. If $l_p < l_a$, then choose an upper joint state s_J that is identical with $[s_I^1, \ldots, s_I^{l_a - 1}, s_J^{l_a}, \ldots, s_J^n]$.
2. If there is a lower joint state s_J that is identical with $[s_G^1, \ldots, s_G^{l_a + 1}, s_J^{l_a + 2}, \ldots, s_J^n]$, then choose it.
3. Otherwise, choose a peer joint state s_J that is identical with $[s_I^1, \ldots, s_I^{l_a - 1}, s_J^{l_a}, \ldots, s_J^n]$.

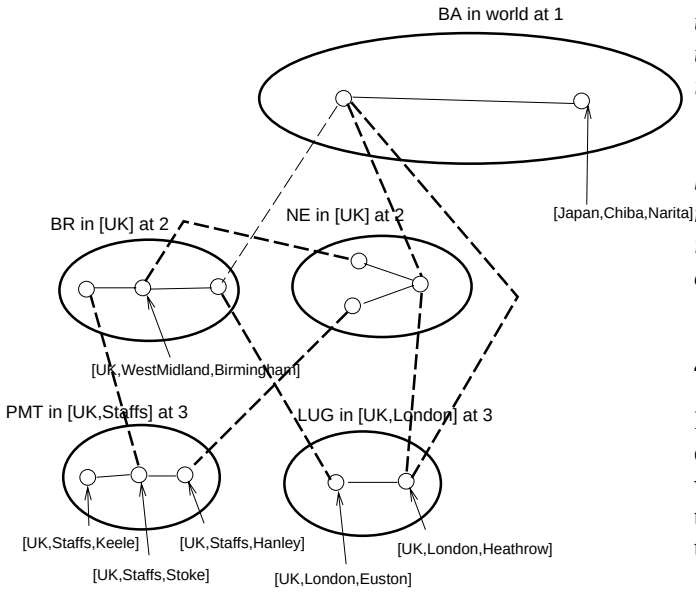


Figure 1: Joint states.

When there are multiple joint states to be selected, we can use one of several strategies described in next section for a further selection to a single joint state.

When a joint state s_J is selected for a problem $< s_I, s_G >$, the problem is decomposed into $< s_I, s_J >$ and $< s_J, s_G >$, and the former one is solved by the agent and the latter one is assigned to the corresponding agent with which the joint state is shared. However, there is an exception when the selected joint state is an indirect upper joint state s_{IJ}. In this case, the problem is decomposed into three partial problems such as $< s_I, s_{IJ} >$, $< s_{IJ}, s_J >$ and $< s_J, s_G >$, and the second and the third ones are assigned to the appropriate agents respectively.

Example 4.3 *Let us assume a problem* $<$ [UK, Staffs, Keele], [Japan, Chiba, Narita] $>$ *is given to a hierarchical community as shown in Figure 1. The problem is sent to* PMT *agent at first, as it has the initial state* [UK, Staffs, Keele] *in its local knowledge. Since the level of the problem is higher than that of the agent, this problem is decomposed and assigned to an upper agent. There are two joint state candidates to select. Let us see each case.*

Case1 ([UK, Staffs, Hanley]*): the problem is decomposed into* $<$ [UK, Staffs, Keele], [UK, Staffs, Hanley] $>$ *and* $<$ [UK, Staffs, Hanley], [Japan, Chiba, Narita] $>$ *and the first one is solved by* PMT *and the second one is assigned to* NE*. Then,* NE *further decomposes the partial problem into* $<$ [UK, Staffs, Hanley], [UK, London, Heathrow] $>$ *and* $<$ [UK, London, Heathrow], [Japan, Chiba, Narita] $>$ *and the latter one is assigned to* BA*.*

Case2 ([UK, Staffs, Stoke]*): the problem is decomposed into* $<$ [UK, Staffs, Keele], [UK, Staffs, Stoke] $>$ *and* $<$ [UK, Staffs, Stoke], [Japan, Chiba, Narita] $>$ *and the first one is solved by* PMT *and the second one is assigned to* BR*. Then,* BR *further decomposes the partial problem into* $<$ [UK, Staffs, Stoke], [UK, London, Euston] $>$*,* $<$ [UK, Staffs, Euston], [UK, London, Heathrow] $>$*, and* $<$ [UK, London, Heathrow], [Japan, Chiba, Narita] $>$ *because the upper joint state* [UK, London, Euston] *is indirect. As the level of second partial problem is 3 and its domain is* [UK, London]*, it is assigned to* LUG*, and the third one is assigned to* BA *as in case 1.* □

4.4 Selection Strategies

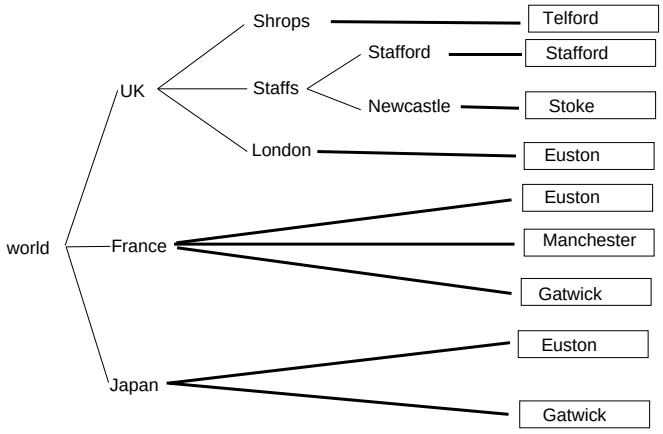
In the previous section, we described how to reduce the candidates of joint states by using knowledge hierarchy. However, we still may have several candidates and need to reduce further to one candidate. In this section, we present several strategies to reduce them to one.

By Using Local Search When no information to select a joint state is available, the agent can select the first joint state obtained by using the local search of the diffusing search algorithm.

By Using Predefined Knowledge A user or system designer can specify how to decompose problems beforehand. A format of the knowledge representation may be like a tree shown in Figure 2, that has the similar structure as the knowledge hierarchies. A leaf nodes is a description of a joint state, shown in Figure 2(b), which includes agent name that share the joint state, and additional parameters such as approximate travel time, travel cost, number of transfer, etc. For example, when the goal state of problem is [UK, Staffs, Newcastle], the joint state to be selected is Stoke by referring this discrimination tree. When the goal state of problem is [UK, London, Somewhere] (Somewhere in London), the joint state is Euston. When there are multiple leaf nodes for a specified domain, one joint state can be selected by referring parameters. This extends the flexibility of the selection.

By Learning The information contained in the discrimination tree given by users or designers is static and could be incorrect or become out-of-date. By using leaning technique, we can keep the information up-to-date or build such information from scratch when there is no information given beforehand. Once an agent finds a solution, it records the result in the discrimination tree for future problem solving.

By Inquiring Information obtained by learning is limited because it is a passive method only based on the results the agent has obtained even though the other agents may have more or better information. Therefore, by inquiring other agents, the agent can select a better joint state. On the other hand, this scheme has to pay communication cost, and may cause chain reactions of inquiring.



(a) Discrimination Tree.

[Joint State, Agent Name, Parameters]

(b) Leaf Node Description.

Figure 2: A discrimination tree.

By Informing If agents inform each other, in other words, if they exchange their information of discrimination trees each other, they can get more information to select a better joint state. This method is useful also when a new agent joins in the community, the agent can advertise what it can do by informing other agents. However, informing as well as inquiring causes to increase the communication overhead [5].

4.5 Multiple Selection

Though we have discussed how to select a single joint state to decompose a problem, when it is difficult to reduce the candidates to one and when it seems multiple candidates have almost the same possibility to lead a solution, it is possible to select multiple joint states and to assign multiple partial problems in parallel. Moreover, in the early stage of problem solving when information in the discrimination tree is small, to assign multiple problems in parallel is a way to get more information than to assign a single one. For example, in Example 4.3, there are two joint state candidates, $[\texttt{UK}, \texttt{Staffs}, \texttt{Hanley}]$ and $[\texttt{UK}, \texttt{Staffs}, \texttt{Stoke}]$, for PMT agent, and it is possible to assign the former problem to NE and the latter problem to BR.

However, multiple assignments increase communication and may generate redundant process. In the above example, both of NE and BR request the same problem $< [\texttt{UK}, \texttt{London}, \texttt{Heathrow}], [\texttt{Japan}, \texttt{Chiba}, \texttt{Narita}] >$ to BA. The issue of how many partial problem should be assigned depends on the quality of expected solutions and the congestion of the communication channel, and we need a further study for this issue including techniques to reduce redundant processes.

5 Conclusion

Problem decomposition is an important process for effective large-scale problem solving. In the conventional DPS techniques, the problem solving knowledge is assumed to be given a priori. In this paper, we presented a problem decomposition scheme for cases that such knowledge is not given. In our scheme, agents autonomously decompose and allocate problems by using the knowledge hierarchy. We also presented several strategies for the cases where there are several alternatives for problem decomposition. For our future study, we are planning to build a prototype to evaluate our proposed scheme.

References

[1] Ranan B. Banerji. *Artficial intelligence: a theoretical approach*. Elsevier North Holland, Inc., 1980.

[2] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*. Morgan Kaufman Publishers, Inc., San Mateo, California, 1988.

[3] To-Yat Cheung. Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Transactions on Software Enginerring*, SE-9(4):504–512, July 1983.

[4] Yasuhiko Kitamura and Takaaki Okumoto. Diffusing inference: An inference method for distributed problem solving. In S. M. Deen, editor, *Cooperating Knowledge Based Systems 1990*, pages 79–94. Springer-Verlag, 1991.

[5] Yasuhiko Kitamura, Baochuang Zheng, Shoji Tatsumi, Takaaki Okumoto, and S. Misbah Deen. A cooperative search scheme for dynamic problems. In *Proceedings 1993 IEEE International Conference on Systems, Man and Cybernetics, Vol.5*, pages 120–125, 1993.

[6] Victor R. Lesser. A retrospective view of fa/c distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1347–1362, November/December 1991.

[7] Judea Pearl. *Heuristics*. Addison-Wesley, Reading, Massachusetts, 1984.

[8] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.