Predicting Possible Conflicts in Hierarchical planning for Multi-Agent Systems

Toshiharu Sugawara[1] Satoshi Kurihara[2] Toshio Hirotsu[3] Kensuke Fukuda[4] and Toshihiro Takada[1]

[1]NTT Communication Science Laboratories
[4]NTT Network Innovation Laboratories
2–4 Keihanna City,Soraku-gun
Kyoto 619-0237, Japan
{sugawara,fukuda,takada}@t.ecl.net

[2]Inst. of Scientific and Industrial Research
Osaka University
kurihara@ist.osaka-u.ac.jp

[3]Dept. of Information and Computer Sciences
Toyohashi University of Technology
hirotsu@entia.org

ABSTRACT

This paper proposes a learning method to select the most appropriate abstract plans during hierarchical planning in the context of multi-agent systems (MAS). In hierarchical planning, a plan is first created at the most abstract level, and is then refined to a more concrete plan, level by level. Thus, selecting an appropriate plan at the abstract level is very important because the selected plan restricts the scope of lower concrete-level plans. This restriction can enable agents to create plans efficiently, but if all the plans under the selected plan contain serious and difficult-to-resolve conflicts with other agents' plans, the resulting plan does not work well or is of low quality. We propose a method in which, from the conflict pattern among agents' plans, an agent learns which abstract plans will cause conflicts with less probability or which conflicts are easy to resolve, thus inducing probabilistically higher-utility concrete plans after conflict resolution. We also show some experimental results to evaluate our method, with the results suggesting structures of resources where tasks are executed.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Distributed Artificial IntelligenceMultiagent systems

General Terms

Theory

Keywords

Planning, Coordination, Cooperation, Learning

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

1. INTRODUCTION

Our project focuses on applications using ubiquitous networks/computers with many types of sensors and a number of effectors/robots[15]. In this application, a number of agents are located in sensors, effectors, robots, PCs and other appliances. They predict persons' activities from sensed data and assist these activities through a system embedded in the environment. To develop this kind of assistance, an agent has to construct a series of actions (or programs). Planning in AI involves a process for generating a plan that is a course of actions designed to achieve a given goal from a specific initial state. Thus, a plan can be considered an abstract form of programming. Planning for agents' actions is a key issue in our application domain [13].

Agents should generate their own plans within a reasonable time because people will complete their intended actions without waiting. Additionally, the processes for conflict detection among resources in agents' plans and their resolutions are necessary in the context of multi-agent systems (MAS). This MAS issue reflects the fact that a number of persons will act in the environment in parallel. For efficient planning, we adopt hierarchical planning[3, 7, 11] using the decision-theoretic planning (DTP) approach[5]. Moreover, both descriptive information and utility are used for plan generation. An abstract plan is first created, and its individual action is refined, level by level, into a sequence of concrete actions (refinement) based on description information such as pre- and post-conditions. Because there is usually a number of refinements for each abstract action, the planner in an agent selects the appropriate one to maximize the value of the utility $U(p)$, where p is the finally generated plan, that is, the sequence of primitive actions.

However, the utility-based selection of actions/plans does not always lead to good results in MASs. If many possible conflicts are found, additional processes for avoiding the conflicts (*conflict resolution*) are invoked and some extra actions (such as waiting for synchronization and detouring) are added to the plan as a result of this process. Thus, the second- or third-best plans that cause no conflict might be better for overall performances from the global multi-agent viewpoint. Furthermore, the determination of which actions easily cause conflicts depends on the environments where agents are deployed. Thus, it is impossible to provide this kind of utility for all possible situations in any envi-

ronment. Agents are therefore required to adaptively select refinements of their abstract plans by learning from past experience, to some extent.

In particular, appropriate selections of actions at the abstract levels are essential in hierarchical planning, since an abstract plan restricts the subsequent refinements at more concrete levels. If the most appropriate refinement existed under another abstract plan, it would never be selected; in this case, the agent has to cancel the resulting plan and return to the abstract level, or give up the effort to generate an acceptable plan.

In this research, we introduce the patterns of conflicts at a certain abstract level called the *screening level*. Then we introduce an additional negative utility, called *conflict discount*, that cumulatively predicts, from screening-level plans and conflict patterns, the cost of resolutions and the quality/performance of the resulting plans with the possibility of conflicts in the subsequent refinement processes. The conflict discount is initially given as 0, and then it's learned by some statistical learning method to estimate more appropriate refinement at the screening level. We assume that the initial utility results in appropriate plans for the single-agent cases. This may also lead to satisficing plans, but it's not optimal in the MAS context, and thus agents learn the conflict discount for the environment to select appropriate screening-level plans when combined with the original utility.

This paper is organized as follows: First, we discuss how to apply the case described above to our application systems. We then explain the process of conflict detection and resolution. Following that, we introduce conflict patterns to classify situations of conflicts with other agents' plans. Finally, we present the experimental results obtained in a simulated laboratory and state our concluding remarks.

2. BACKGROUND

2.1 Hierarchical Planning in a MAS Context

In hierarchical planning, plans are generated based on an abstract hierarchy of the domain model, which includes tasks and resources (Fig. 1). Initial states and goals are first described in the most abstract (or uppermost) model, and a number of task sequences are generated to achieve these goals in this model. (An example of the task hierarchy established in accordance with the model hierarchy is shown in Fig. 2.) This plan generation is usually based on the descriptive information represented in the corresponding model. One of the sequences is selected according to a particular planning strategy (the utility is used in the case of DTP), and then each task in the sequence is further refined, that is, the sub-task sequences in the less-abstract model for achieving the task are generated. These sequences are called *refinements* of the task.

This refine-and-select process is subsequently iterated until all tasks in the sequences have been refined to primitive tasks that are expressed in the lowest model and that directly correspond to executable forms of programs and procedures. Higher-layer (more abstract) models are simpler and thus do not contain complete information, but they are appropriate for understanding the global and long-term picture of activities. Naturally, the lower-layer models are more informative and complicated, but they are used for detailed descriptions of local and sectional plans.

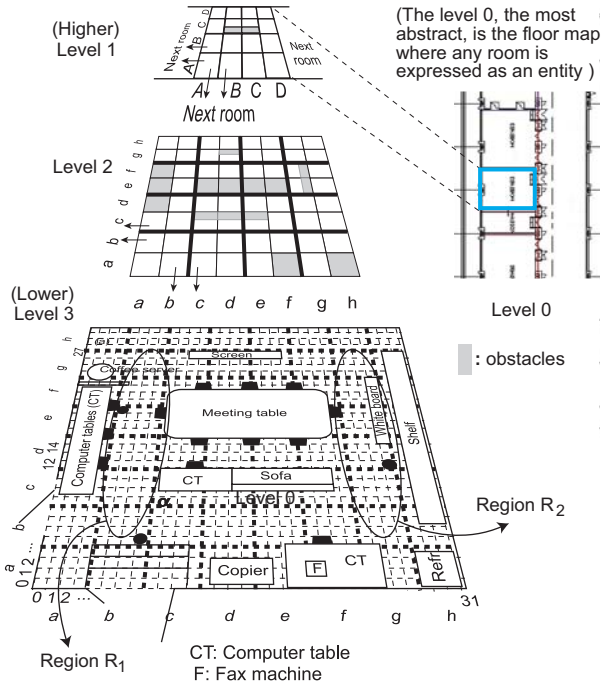

Figure 1: Example of Hierarchical Description.

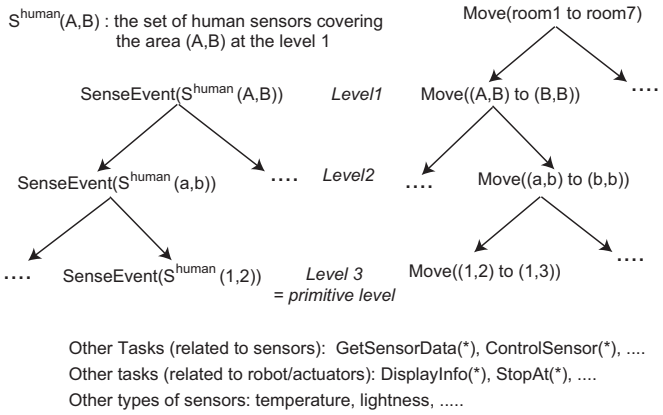

Figure 2: Hierarchical Task Structure.

An agent selects the plan that may lead to the highest utility (plan selection). However, the utility value is determined from the primitive task/plan, so the utility of a non-primitive task or plan is expressed as a range calculated by the possible lower-level refinements. It has been reported that agents should choose the plan that contains the highest utility and expand it to the next layer for effective planning[5].

2.2 Main Issue

The actual conflicts are also determined when all tasks are expanded to primitive tasks, since the required amount of resources and the time needed for plan execution are precisely identified at this level. An agent can partly investigate the possibility of conflicts at an abstract level and resolve

them. For example, if a certain room is roughly modeled as a single object at the abstract level (such as level 0 in Fig. 1) and two agents have plans to work in the room at the same time, they can resolve this possible conflict by one agent's replanning to work another time. However, this predicted conflict may not occur after all when all plans are expanded as primitive plans because they can work at different places in the room. Thus conflict resolution at an abstract level is redundant. A similar situation arises when the planner takes into account other used/consumed resources such as power current, sensors, effectors, robots, displays, etc,, because a single entity of the resource description at non-primitive levels corresponds to the set of resources grouped according to a certain perspective, as shown in Fig. 2. In general, the process of conflict detection and resolution in abstract layers is simple because its domain model and the related operators are simple, but the resulting plans may not be appropriate. In applications where real-time performance is stipulated, agents preferably can predict whether the conflicts will vanish or the resolutions will be easy or difficult during the remainder of the planning.

Normal utility functions do not usually take into account possible conflicts with other agents. As a result, they can create acceptable plans when there is no interference between agents' plans, but, otherwise, they might not. It is important, therefore, to provide another utility for plan selection when there is the possibility of conflicts arising. However, what conflicts and which tasks easily cause them reflect the locations of scarce or heavily used resources, so the outcome depends heavily on the environment. This type of information cannot be provided *a priori* in the design time. Furthermore, agents have to predict possible conflicts in a real-time manner; therefore, agents are required to learn an additional utility for the MAS context.

2.3 Related Work

An abstract task is considered to be a subroutine or a subfunction to be learned; accordingly, our work is related to hierarchical reinforcement learning (RL) such as [6, 1, 4, 8, 10, 14], although our approach is not RL. For example, in [14], a frequently used task sequence is identified as a single operation in order to improve learning efficiency. In the MAXQ approach[4], a task is divided into subroutines that are individually learned by RL methods, improving each subroutine. Our approach is to select an appropriate subroutine for each situation, but MAXQ also needs to do this. In MAXQ any conflict discount is assumed to be already learnt at lower levels. However, in a multi-agent setting, it is naturally difficult to define the task hierarchy over all agents simultaneously.

In the MAS context, some hierarchical planning and coordination issues were also discussed. For example, [2] proposed methods to choose the most appropriate abstract task/plan using summary information that is summed up from the primitive tasks and plans in a bottom-up fashion. [12] discussed the RL method to learn coordinated activities using local resource information. The main difference from these studies is that our major problem is to lower the cost of conflict resolution and to improve the quality of the plans resulting after conflict resolution.

3. LEARNING SITUATIONS AND AN ASSUMPTION

3.1 Application Environment

We aim to achieve effective planning for ubiquitous computing/network systems. In this type of application, a number of people enter the environment, identify their objectives, and receive assistance in their activities. In such a case, a number of agents are already working based on their plans. Furthermore, the system must have their plans in order to detect possible interference among agents. It is, however, costly to collect and analyze all primitive plans.

The application system of our proposed architecture shares only plans described in a certain abstract-level model; the *manager agent* holds the plans being executed at this level. This level is called the *screening level*. We assume that these plans are simple enough but not too simple to be analyzed for conflict detection. These plans also provide details on overall and long-term activities of the activated agents. The manager agent maintains all screening-level plans being executed. When an agent starts to create its plan for this environment, it requests the manager agent to investigate the possible conflicts between this new plan and other executed current plans at the screening level. If conflicts are detected, the agent that has the new plan directly requests the primitive plans from the partner agent to accurately analyze the possible conflicts. This system architecture allows the agents to identify a plan that has no conflicts and to use the normal utility for planning. If conflicts are predicted, agents will use the utility with the conflict discount. Thus our aim is to learn to predict conflicts at the screening level for this purpose. We think that this architecture can lower the cost of predicting conflicts and their resolutions. Additionally, we focus on plans that frequently appear, that is, actions that are the repeatedly observed series of sensor data that suggest a person's daily activities[9].

We introduce the assumption that all possible conflicts are detected in the screening-level analysis by the manager agent. Some conflicts will vanish in the lower-level planning. However, this assumption means that if no conflicts with other plans are detected, no conflicts will actually occur in the subsequent planning as long as the agent does not change its screening-level plan.

In our proposed system, some agents are already running in the system. When another agent joins them, it creates the plan for its goal, starting from the most abstract level. Then, once a number of screening-level plans have been generated, these plans are sent to the manager agent. The manager agent investigates possible conflicts using all screening-level plans being executed. It then returns conflict information with other screening-level plans to the agents. This information states (1) if the agent is not expected to have any conflict, its utility, or (2) if the agent may have conflicts, other agents' plans expected to have conflicts, the locations of these conflicts in the plans, and its utility with the conflict discount. Note that the other plans are already executed; therefore, in the current implementation, the executed plans are not modified (at least, the plans that once were recognized as executed should be preferred). Often this restricts the quality of the resulting plan. Our aim, however, is to select the most appropriate screening-level plan. If all of the generated plans at the screening-level appear to result in high-discount conflicts, the agent can backtrack and se-

lect another plan at the upper level; the agent still creates a relatively simple plan, so this cost is not so high.

3.2 Concept of Conflict Discount

Let $U(p)$ (or $U(t)$) be the initially given utility for a primitive plan p (or a primitive task t). $U(p)$ for a non-primitive plan (or task) is the range which cumulatively indicates possible lower-primitive plans/tasks. We introduce the *conflict discount*, $cd(a(p), e, a(p'))$ for an environment e, the screening-level plan $a(p)$ of p, and another agent's screening-level plan $a(p')$. When no conflicts are predicted, the agent can use only $U(p)$ for selecting plans at this level. When conflicts with another agent's plan, say $a(p')$, are likely to occur, it uses the modified utility

$$U(p) - cd(a(p), e, a(p'))$$

to consider the probability of the conflicts. We assume that $a(p')$ contains scheduling data of the plan, that is, the planned execution time and duration of each task (these data are determined because this plan is already scheduled).

We try to handle conflicts among three or more agents. Here, we define

$$U(p) - cd(a(p), e, a(p'), a(p''), \dots)$$

when conflicts with $p', p'', \dots$ are predicted at different locations in the plan p.

Our proposed method aims at adjusting cd by adopting a statistical method for frequently appearing situations. The conflict discount is conceptually defined as follows:

$$cd(a(p), e, a(p'), a(p''), \dots) \quad (1)$$
$$= U(pp) - U(pp_m) + \textit{cost-of-conflict-resolution},$$

where pp is the primitive plan before conflict resolution and pp_m is the primitive plan that is modified for resolving conflicts with other plans. The term *cost-of-conflict-resolution* is calculated by combining the cost of applied conflict resolution rules and the number of messages; then $cd(p)$ is redefined by cumulatively adding this value; how to calculate $cd(p)$ is described below. We assume that $U(p)$ is the length of the primitive plan in our example below.

3.3 Conflict detection and resolution

A number of resolution methods, shown in Table 1, are applied to resolve conflicts. Thus, the agents involved must negotiate which agent (or all agents involved) should commit to modifying their plans and then decide what methods should be applied.

These resolution methods are defined as rules and applied under a certain policy. The resulting plans usually have extra cost for the resolutions. In this paper, we do not care what kind of policy is used; our only concern is the cost of resolution and the quality of the resulting plan.

Table 1: Examples of methods of resolution.

Method	Description
Synchronization	Stop until another agent performing a task that requires a needed resource finishes the task and releases the resource. Wait for a primitive task or use of some resource by another agent until the task finishes or the agent releases the resource. This method may insert a number for "wait for a tick" for synchronization.
Waiting	Stop until other agents finish tasks that create pre-conditions of the local task. This method may insert a number for "wait for a tick" for synchronization.
Replacement	Replace tasks whose post-conditions do not affect tasks in other agents or whose pre-conditions are not affected by tasks in other agents. This method may replace the conflicting task with others, but these other tasks usually have lower utility (or incur extra cost).
Reordering	Reorder tasks to avoid negative relationships.
Insertion	Insert tasks whose post-conditions recover the pre-conditions of the task. This method adds some tasks, so the utility of the resulting plan decreases.
Commission	Entrust the task to other agents. This form of resolution is preferable when, for example, a conflict can only be resolved by other agents, or if another agent can do the task at lower cost. This method can eliminate some tasks, though some communications, not only for detecting the sharable tasks of the plans but also for committing them to another agent, take place.

4. ADJUSTING CONFLICT DISCOUNT

4.1 Conflict Detection at the screening level

At the screening level, the manager agent detects the possible conflicts, according to resource and task information described in the model, by identifying the possibilities of whether multiple plans will use the same resources, such as sensors and places (which are squares in Fig. 1) in the ubiquitous sensor network environment. An example of a conflicting situation is shown in Fig. 3, where there are two executed plans that will conflict with the new plan.

An actual example is illustrated in Fig. 4, where the screening level is two; the manager agent suggests that task t_l = *move(cd → dd)* in the new plan may have some conflict with task t'_n = *move(cd → bd)* in the plan presently being executed. This conflict detection can be traced according to the used resource data (in this example, it is possible that some squares in the area *(c,d)* may be occupied by two agents simultaneously during a certain time interval).

The manager agent also takes into account time relationships between tasks in the plans because conflicts occur only when more than two agents use some resources simultaneously. The required duration (that is, start to end time) of the task in the executed plans is already determined, but the one in the new plan is not. Thus, the manager agent uses the expected average duration of each screening-level task. This value is initially given as the expected required duration in the model; for example, *move(cd to bd)* takes four ticks if we assume that the robot or person can move to the next small square in four ticks[1]. The expected duration

[1]However, the average is about five ticks, specifically 5.7

is then statistically adjusted according to actually generated primitive plans.

4.2 Conflict Pattern and Prediction

Let us define a conflict pattern (CP) of a plan with other agents' plans as the relationship between detected conflicts, with consideration given to time relativity. First, we focus on each task at the screening level that is identified to have some conflict. For example, in the previous example of Fig. 4, the CP was expressed as follows:

$$CP_1(t_l) = (t_l, (t'_n, (\max(s'_n - s_l, 0), \min(e_l - s_l, e'_n - s_l)))),$$

where the final term is the relative time interval during which the expected conflict may occur. For efficient learning, in this paper we introduce simplified expressions of time relativity: anterior half[ah], posterior half[ph] and overlap[ol]. Thus, this example is,

$$CP_1(t_l) = (t_l, (t'_n, r'_l)),$$

where r'_l = *ah, ph* or *ol*. The second term (t'_n, r'_l) in the right-hand side of this equation is called *conflict data*.

The task may have more than two conflicts with other plans, such as the situation shown in Fig. 3, so a CP may have two or more conflicting data; in this example situation the CP is:

$$CP_2(t_l) = (t_l, (t'_{n+1}, r'_l), (t''_{m-1}, r''_l))$$

where r'_l, r''_l = *ah, ph* or *ol*.

Then, the agent iteratively adjusts the conflict discount, *cd*, for a certain conflict pattern, *CP*, with the following equation:

$$cd'_m(CP) = \lambda * cd'_{m-1}(CP) + (1 - \lambda) * \textit{differential-cost},$$

where $0 < \lambda < 1$ and *differential-cost* means the differential utilities of the original plan and the plan resulting after conflict resolution. For example, if the partner agent takes route (1) in Fig. 4 and this conflict is resolved by using "wait for two ticks" to wait until the partner agent passes by, two of the "wait" tasks are inserted. In this case the *differential-cost* is 2. This value differs if the partner agent takes the other route (2) in Fig. 4; in this case, no conflict will actually occur and *differential-cost* is 0.

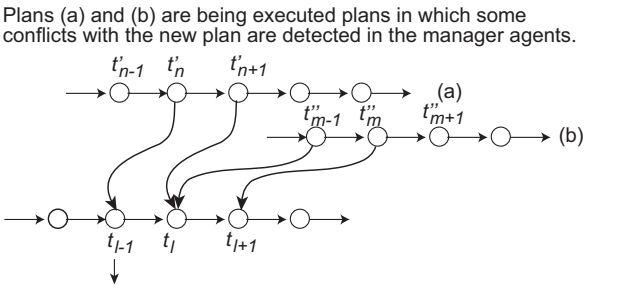

Figure 3: Example Situation of conflicts between plans.

ticks, for *(c,d)*, after the statistical analysis of actual plan data.

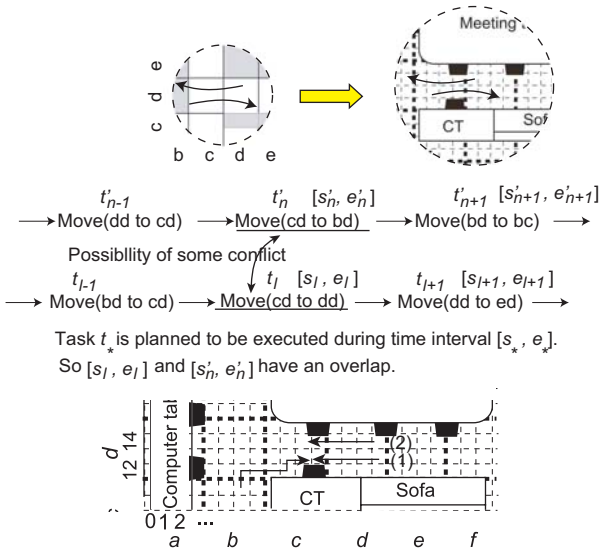

Figure 4: Example of a Detected Conflict.

The process of *cd* calculation is, like the process of conflict resolution, an iteration of the procedures for (P1) searching the first task t in the new plan, which may have conflicts in the execution order, and for (P2) predicting the conflict discount cd' for only this specific situation. This specific situation corresponds to the conflict pattern. In procedure P2, the additional cost for avoiding the conflicts is predicted, and thus the start times of subsequent tasks may be delayed for this amount of time. Because of this delay, a number of conflicts may disappear, although some new ones may be detected in the remaining part of the plan. In particular, another conflict may occur with the current task t. If so, the agent must also calculate the cd' for this new conflict.

4.3 Window Size

CPs in the previous subsection have limited ranges of tasks; they only focus on a single task that may have conflicts with other agents. However, the type of conflict and its resolution cost may depend on other conflicts occurring in neighboring tasks, and thus it is often essential to take into account conflict occurrences between neighboring tasks. For this purpose, we introduce the window size of the CP; the general form of CP with a window size of 3 is

$$\begin{aligned} CP(t_{l-1}) = [&(t_{l-1}, (t'_{n-1}, r'_{l-1}), \dots), \\ &(t_l, (t'_n, r'_l), \dots), \\ &(t_{l+1}, (t'_{n+1}, r'_{l+1}), \dots)] \end{aligned}$$

since this pattern includes information on conflicts occurring on the right after two tasks. Furthermore, conflict data with the same plan are aligned in the same column; for example, the CP for t_{l_1} in Fig. 3 is:

$$\begin{aligned} CP(t_{l-1}) = [\ &(t_{l-1},\ (t'_n, r'_{l-1}),\ ()), \\ &(t_l,\ \ (t'_{n+1}, r'_l),\ (t''_{m-1}, r''_l)), \\ &(t_{l+1},\ (),\ \ \ \ \ \ (t''_m, r''_{l+1}))], \end{aligned}$$

where the first and second columns of conflict data correspond to plans (a) and (b) in Fig. 3, respectively.

Each column corresponds to a single plan of another agent, so a column of conflict data can replace another column. To avoid multiple expressions for the same situation, we normalize this expression. First, we look at the first line and select the column whose first element is not empty. If there are multiple columns, those columns are sorted according to the alphabetical order of the first elements, then the second elements, and so on (thus, they are sorted by the task name). Next, we focus on the columns whose second element is not empty but whose first element is empty and apply the same sorting methods.

We define conflict discount cd from eq. (1) as follows:

$$cd(p) = \sum_{t \in p} cd'(CP(t)) + \#\text{-}of\text{-}conflict\text{-}plans$$

where p is a screening-level plan (so t is a screening-level task) and we assume that $cd'(CP(t)) = 0$ if task t is identified as having no conflict with other plans. We must note that the values of $cd'(CP(t))$ in this formula are derived as the iterated process of conflict prediction described in Section 4.2. In the following experiments, we assume that the cost of receiving and analyzing a primitive plan from another agent is 1 (of course, this value is system-dependent). Thus, the second term on the right-hand side is the number of plans having conflicts with the new plan, as the approximate value of the cost of conflict resolution.

5. EXPERIMENTS

We experimentally calculated the values cd' using our simulated laboratory room, which is based on our actual laboratory rooms (see Fig. 1). In this experiment, the screening level is two. Agent A_1 randomly selects the starting point in area R_1 and the goal in area R_2 and then tries to generate a new plan for this movement. There is another agent B that already has an approved plan whose start and goal are also randomly selected in R_1 and R_2. Conflicts often do not occur at the screening level, since two agents may take different routes such as north and south of a meeting table; therefore, this experiment is iterated until conflicts are detected between A_1's task *move(cd → dd)* and A_2's task, which is *move(cd → dd)* (same direction) or *move(cd → bd)* (opposite direction). We especially focus on area *(c,d)* because its route is slightly narrow due to the chairs and computer tables in the meeting room.

The screening-level plan is then actually expanded to a primitive plan, and we investigate the conflict discount of the plan resulting after conflict resolution. Note that this cost may be 0 if the agent can find another route of the same length or if no conflicts actually occur. Then, we iterate this experiment a few hundred times to calculate cd'.

The task *move(cd → dd)* usually takes four to six ticks in this environment. Consequently, we define in this experiment that if the manager agent finds the possibility of a conflict: (1) within the first two ticks, the relative time relationship is *ah*; (2) within the last two ticks, the relative time relationship is *ph*; and (3) otherwise, the relative time relationship is *ol* as shown in Fig. 5. Consequently, we estimate the values of cd' for the following conflict patterns:

$$CP_1 = (move(cd \rightarrow dd), ((move(cd \rightarrow dd), r)))$$
$$CP_2 = (move(cd \rightarrow dd), ((move(cd \rightarrow bd), r))),$$

where r is *ah, ph,* or *ol*.

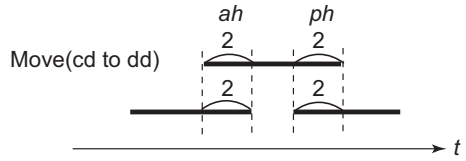

Figure 5: Relative Time Relationships.

Figure 6 shows the estimated value of cd'_m ($\lambda = 0.98$) and the average of additional costs when $r = ol$. Their expected cd' values are approximately 0.27 (opposite direction) and 0.55 (same direction). The graph is not shown here, but when $r = ph$, the expected values are approximately 0.29 (opposite direction) and 0 (same direction). In these cases, the additional cost is reasonably small, because the two-square-width path is wide enough for two agents to pass through the area, but agent A_1 sometimes has to take a detour to avoid conflicts. The value is higher when agents move in the same direction because in this situation, it's often observed that agent A_1 has to wait for one or two ticks until another agent passes by. In the cases of the opposite direction, the agent can often take another route through the narrow area without incurring any additional cost.

These experimental results suggest that the values of cd'_m depend on the resource structure of routes, especially area *(c,d)* in Fig. 1. For example, in the case when two agents move in the same direction and the relative time relationship is *ph*, these agents hardly interfere with each other at all. If they move in the opposite direction, they will meet near a narrow path in *(c,d)*, but in many cases, the new agent can re-route to avoid a collision because the width of the narrow path is 2.

Next, we investigate the cd' values in the situation where there are two agents that have approved plans (and thus these plans have no conflicts), where they move in the same direction *move(cd → dd)*. The conflict pattern of this situation can be expressed as follows:

$$CP_3 = (move(cd \rightarrow bd), ((move(cd \rightarrow dd), ol), (move(cd \rightarrow dd), ol))).$$

The result of the estimated conflict discount is 6.17 (plus 2 for receiving and analyzing the other agents' plans), which is quite different from the previous cases, as Fig. 7 shows. The two agents have plans under execution that have no conflicts, and thus they usually occupy the narrow route in area *(c,d)*. Therefore, the agent creating the new plan always has to move aside, wait for several ticks until the agents pass by, and move back to the original route. If the agent has a new plan that is predicted to have a conflict with this conflict pattern in the screening level, the new agent can select after learning another route such as a route taking it north of the meeting table or a route taking it south of the sofa, as in Fig. 1. When one of the relative time relationships in CP_3 is *ph*, the estimated cd'_m is approximately 0.98, because the two agents move a slight distance away. Of course, if this kind of conflict occurs at another position (such as *(d,d)* and *(e,d)*), the agent does not have to change its screening-level plan because these areas have enough room for three agents to pass by each other.

Finally, we mention four important points. First, when a

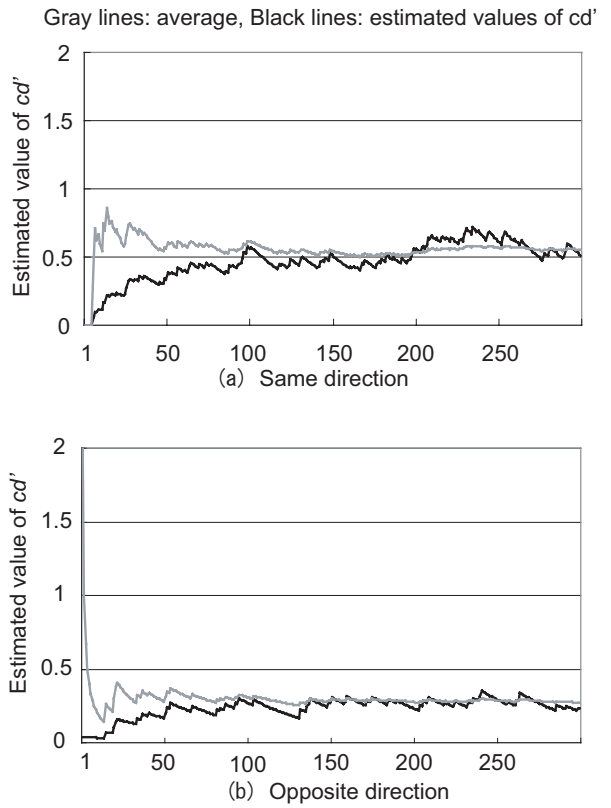

Figure 6: Estimated cd' and average values.

conflict among more than four agents is detected at *(c,d)*, the agent that has the new plan can often use the result of the previous experiment. If the conflict pattern contains the CP_3 as sub-pattern, its cd' value becomes higher than 6.17, so the agent can decide to adopt another route. Second, although start and goal positions are selected randomly in our experiments, these might not be random in actual situations because agents (including persons) usually have fixed start and goal points. As a result, these features of the experimental results are expected to be more noticeable. Third, it is essential to choose an appropriate screening level. For example, if level 1 in Fig. 1 is the screening level, this method does not work well. This issue will be discussed elsewhere. Finally, we introduce three relative time relationships *ah, ol* and *ph*, but these relationships are not always necessary, as with the case of *(e,d)* and *(f,d)*, because they are not non-scarce resources if the number of agents is not so large.

6. CONCLUSION

This paper proposed a method to predict, at some abstract level called the screening level, the cost of possible conflict resolution and the quality of the resulting plan in order to generate better primitive (concrete) plans. We are now interested in ubiquitous/sensor network applications, in which a number of agents located in sensors, effectors, robots, PCs and other appliances predict persons' activities from sensed data and assist in these activities. To develop agents' activities, real-time (and thus very efficient) planning is required. In our framework, an agent called the

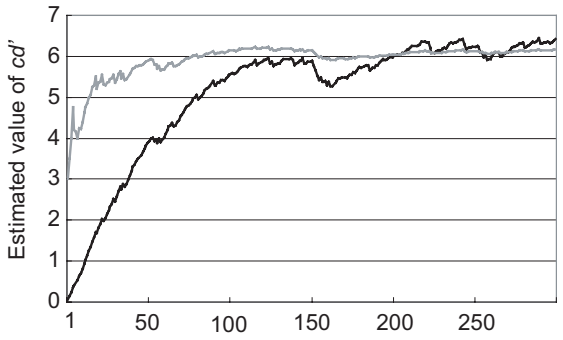

Figure 7: Estimated cd' and average values.

manager maintains the plans presently being executed at the screening level and predicts the possible conflicts between these plans and the newly proposed plan. Then, if necessary, a detailed analysis for primitive plans is done by individual agents. Furthermore, selecting appropriate abstract plans is very important because the selected plan restricts the scope of more concrete plans. Finally, we also showed experimental results on the expected additional cost of the plans after conflict resolution.

We plan to add another dimension to the proposed framework: determining what resolution method is most appropriate for each situation. We will also evaluate our method by incorporating the total cost of planning and execution in our experimental environment.

7. ACKNOWLEDGMENTS

The authors would like to thank Prof. Sachiyo Arai, Faculty of Engineering, Chiba University, Japan and anonymous referees for their valuable comments on an earlier version of this paper.

8. REFERENCES

[1] D. Chapman and L. P. Kaelbling. Input Generalization in Delayed Reinforcement Learning: An Algorithm And Performance Comparisons. In *IJCAI-91*, pages 726–731, 1991.

[2] B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proc. of AAAI-99*, pages 495–502, 1999.

[3] D. D. Corkill. Hierarchical planning in a distributed environment. In *Proc. of IJCAI-79*, pages 168–175, 1979.

[4] T. G. Dietterich. The MAXQ Method for Hierarchical Reinforcement Learning. In *Proc. of ICML-98*, pages 118–126, 1998.

[5] R. Goldwin and R. Simmons. Search Control of Plan Generation in Decision-Theoretic Planners. In *AIPS 1998*, pages 94–101, 1998.

[6] B. Hengst. Model Approximation for HEXQ Hierarchical Reinforcement Learning. In *ECML 2004*, pages 144–155, 2004.

[7] J. H. K. Erol and D. S. Nau. HTN planning: Complexity and expressivity. In *Proc. of AAAI-94*, pages 1123–1128, 1994.

[8] L. P. Kaelbling. Hierarchical Learning in Stochastic Domains: Preliminary Results. In *Proc. of Int. Conf. on Machine Learning*, pages 167–173, 1993.

[9] S. Kurihara, K. Fukuda, T. Hirotsu, S. Aoyagi, T. Takada, and T. Sugawara. Multi-agent Framework for Human-Environment Interaction in Ubiquitous Environment. In *Proc. of the Workshop on Agents for Ubiquitous Computing (UbiAgents2004)*, pages 5 – 12, 2004.

[10] R. Parr and S. Russell. Reinforcement Learning with Hierarchies of Machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049, 1998.

[11] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115 – 135, 1974.

[12] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proc. of AAAI-94*, pages 426–431, 1994.

[13] T. Sugawara, S. Kurihara, K. Fukuda, T. Hirotsu, S. Aoyagi, and T. Takada. Reusing Coordination and Negotiation Strategies in Multi-Agent Systems for Ubiquitous Network Environment. In *Proc. of AAMAS2004*, pages 496 – 503, 2004.

[14] R. S. Sutton, D. Precup, and S. Singh. Intra-Option Learning about Temporary Abstract Actions. In *Proc. of Int. Conf. on Machine Learning (ICML)*, pages 556–564, 1998.

[15] T. Takada, S. Kurihara, T. Hirotsu, and T. Sugawara. Proximity Mining: Finding Proximity using Sensor Data History. In *Proc. of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA2003)*, pages 129 – 138, 2003.