Meta-level Reasoning in Deliberative Agents

Anita Raja
Department of Software and Information Systems
The University of North Carolina at Charlotte
Charlotte, NC 28223
anraja@uncc.edu

Victor Lesser
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
lesser@cs.umass.edu

Abstract

Deliberative agents operating in open environments must make complex real-time decisions on scheduling and coordination of domain activities. These decisions are made in the context of limited resources and uncertainty about the outcomes of activities. We describe a reinforcement learning based approach for efficient meta-level reasoning. Empirical results showing the effectiveness of meta-level reasoning in a complex domain are provided.

1. Introduction

Open environments are dynamic and uncertain. Complex agents operating in these environments must reason about their local problem-solving actions, coordinate with other agents to complete tasks requiring joint effort, determine a course of action and carry it out. These deliberations may involve computation and delays waiting for arrival of appropriate information. They have to be done in the face of limited resources, uncertainty about action outcomes and in real-time. Furthermore, new tasks can be generated by existing or new agents at any time. These tasks have deadlines where completing the task after the deadline could lead to lower or no utility. This requires meta-level control that interleaves an agent's deliberation with execution of its domain activities.

How does an agent efficiently trade off the use of its limited resources between deliberations about which domain actions to execute and the execution of these domain actions that are the result of previous deliberative actions? This is the meta-level control problem for agents operating in resource-bounded multi-agent environments. The deliberative actions studied here are scheduling and coordination. We enumerate various options for achieving these deliberative actions. In this paper, meta-level control uses a model of the environment and deliberation choices of the agent to determine the behavior of the deliberation level. The deliberation choices vary in their performance characteristics. The advantage of explicit meta-level control is determined by the environment characteristics, available deliberation choices and cost of meta-level control activities.

We show that in a fairly complex domain where there is dynamic decision making on tasks, it is advantageous to have meta-level control with bounded computational overhead and this meta-level control can be dynamically constructed via a reinforcement learning process. We also show that learning meta-level control policies is computationally feasible through the use of a domain-independent abstract representation of the agent state. This abstraction concisely captures critical information necessary for decision making while bounding the cost of meta-level control.

There has been previous work on meta-level control [2, 4, 8] but there is little that is directly related to the meta-level control problem of deliberative multi-agent systems. Hansen and Zilberstein [3] extend previous work on meta-level control of anytime algorithms by using a non-myopic stopping rule. It can recognize whether or not monitoring is cost-effective, and when it is, it can adjust the frequency of monitoring to optimize utility. This work has significant overlap with the foundations of the meta-level control reasoning framework described here. /citehansen96monitoring deals with the single meta-level question of monitoring, considers the sequential effects of choosing to monitor at each point in time and keeps the meta-level control cost low by using a lookup-table for the policy. This work has some of its foundations in Russell and Wefald [7] in that we too do not insist on optimal control for all reasoning. Our analysis is for a multi-agent environment while their work focused on single agent decisions.

The meta-level control work described in this paper is applied to domains that have the following additional complexities: The tasks of the multi-agent system are distributed among the member agents by an external task allocator. Each agent is assigned a set of tasks that are handled exclusively by that agent. The goal is to maximize the util-

ity obtained by the multi-agent system which is the sum of the utilities of the tasks completed by all the agents. The agents are capable of pursuing multiple tasks (goals) concurrently. Some tasks require multiple-agents to coordinate their activities such as one agent doing a subtask for another agent. Each task is represented using a hierarchical task network (HTN) formulation, has its own deadline and an associated utility that is accrued when the task is completed. The task network describes one or more of the possible ways that the task could be achieved. These alternative ways are expressed as an abstraction hierarchy whose leaves are basic action instantiations. These primitive action instantiations are called domain actions. Domain actions can be scheduled and executed and are characterized by their expected quality and duration distributions.

The paper is structured as follows: We first describe the meta-level reasoning process using abstract representation of the system state. We then present the empirical reinforcement learning algorithm used for meta-level control. Finally, we present a performance evaluation comparing the the policies obtained by reinforcement learning to hand-generated heuristic strategies and conclude with a discussion.

2. Meta-Level Control

Meta-level control (MLC) is the process of optimizing an agent's performance by choosing and sequencing domain and deliberative activities. There can be various options for performing each of these activities and these options vary in their performance characteristics. For instance, the simple option for scheduling uses pre-computed information about the task to select the appropriate schedule which fits the criteria. This will support reactive control for highly time constrained situations. The complex option is a soft real-time process of finding an execution path through a hierarchical task network such that the resultant schedule meets certain design criteria, such as real-time deadlines, cost limits, and utility preferences [11]. When certain exogenous or internal events occur, meta-level control determines the best context sensitive activity choice.

The following are events that occur in our domain which require meta-level control. Each such event has an associated decision tree. The set of possible action choices corresponding to each event is also described.

Arrival of a new task: When a new task arrives at the agent, the meta-level control component has to decide whether to reason about it later; drop the task completely; or to do scheduling-related reasoning about an incoming task at arrival time and if so, what type of scheduling - complex or simple. The decision tree describing the various action choices named A1-A11 is shown in Figure 1. Scheduling actions have associated utilities and also costs with respect to scheduling time and decommit costs of previously established commitments if the previous schedule is significantly revised or completely dropped. These costs are diminished or avoided completely if scheduling a new task is postponed to a later convenient time or completely avoided if the task is dropped. The meta-level controller can decide that it does not have enough information to make a good decision and will consequently choose to spend more time in collecting features which will help with the decision making process[A6].

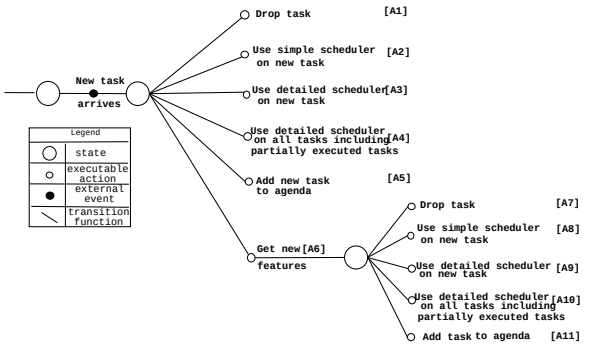

Figure 1. Decision tree when a new task arrives

Invocation of the detailed scheduler: The parameters to the scheduler are scheduling effort, earliest start time and slack amount. Their values are determined based on the current state of the system including characteristics of the existing schedule and the set of new tasks that are being scheduled. The *scheduling effort* parameter determines the amount of computational effort that should be invested by the scheduler. The parameter can be set to either *HIGH*, where a high number of alternative schedules are produced and examined or *LOW*, where pruning occurs at a very early stage and hence few alternative schedules are compared, reducing the computational effort while compromising the accuracy of the schedule. The *earliest start time* parameter determines the earliest starting time for the schedule to begin execution. This parameter is offset by the sum of the time needed to complete any primitive executions whose execution has been interrupted by the meta-level control action and the time spent on scheduling the new task(s). The *slack* parameter determines the amount of flexibility available in the schedule so that unexpected events can be handled by the agent without it detrimentally affecting its expected performance characteristics.

The other events that are reasoned about by the MLC are *When to initiate negotiation with another agent*, *Whether to renegotiate if a coordination event fails* and *Whether*

to reschedule when performance is below expectation. Detailed descriptions are provided in [5].

The MLC in making its decisions does not directly use the information contained in the agent's current state. This would include detailed information regarding the tasks that are not yet scheduled, status of tasks that are partially executed, and the schedule of primitive actions that are to be executed. Instead the MLC uses a set of high-level qualitative features that are computed from the full state information and pre-computed information about the behavior of the tasks that the system can handle. The advantage of this approach is that it simplifies the decision making process and provides the possibility for automatically learning these rules. The following are the features of the high-level state used by the meta-level controller. Most of the features take on qualitative values such as high, medium and low.

F1: Utility goodness of new task: It describes the utility of a newly arrived task based on whether the new task is very valuable, moderately valuable or not valuable in relation to other tasks being performed by the agent.

F2: Deadline tightness of a new task: It describes the tightness of the deadline of a particular task in relation to expected deadlines of other tasks. It determines whether the deadline of the new task is very close, moderately close or far in the future.

F3: Utility goodness of current schedule: It describes the utility of the current schedule normalized by the schedule length and is based on information provided by the scheduler.

F4: Deadline tightness of current schedule: It describes the deadline tightness of the current schedule in relation to expected deadlines of tasks in that environment. If there are multiple tasks with varying deadlines in the schedule, the average tightness of their deadlines is computed.

F5: Arrival of a valuable new task: It provides the probability of a high utility, tight deadline task arriving in the near future by using information on the task characteristics like task type, frequency of arrival and tightness of deadline.

F6: Amount of slack in local schedule: It provides a quick evaluation of the flexibility in the local schedule. Availability of slack means the agent can deal with unanticipated events easily without doing a reschedule. The cost of inserting slack is that the available time in the schedule is not all being used to execute domain actions.

F7: Deviation from expected performance: It uses expected performance characteristics of the schedule and the current amount of slack (F6) to determine by how much actual performance is deviating from expected performance.

F8: Decommitment Cost for a task: This estimates the cost of decommiting from doing a method/task by considering the local and non-local down-stream effects of such a decommit.

F9: Amount of slack in other agent's schedule: This is used to make a quick evaluation of the flexibility in the other agent's schedule if coordination is required.

F10: Relation of slack fragments in local schedule to new task: This determines the feasibility of fitting a new task given the detailed fragmentation of slack in a particular schedule and involves resolving detailed timing and placement issues.

F11: Relation of slack fragments in non-local agent to new task: This determines the feasibility of fitting a new task given the detailed fragmentation of slack in a particular non-local schedule.

We will now describe some of the low-level parameters that determine the high-level features of the system state using a simple example. A typical task is composed of subtasks and primitive actions. Primitive actions can be scheduled and executed and are characterized by their expected utility and duration distributions. These distributions are statistical domain knowledge available to the agent. For instance, the utility distribution of a method if described as $(10\%\ 30\ 90\%\ 45)$, indicates that it achieves a utility value of 30 with probability 0.1 and utility of 45 with probability 0.9.

A multi-agent system M is a collection of n heterogeneous agents. Each agent α has a finite set of tasks T which arrive in a finite interval of time. N_{CT} is the total number of tasks that have arrived at the system from the start to current time CT. Let $t \ \epsilon \ T$ upon arrival has an arrival time AT_t and a deadline DL_t associated with it. A task t can be achieved by one of various alternative ways (plans) $t^j, t^{j+1}, t^{j+2}..., t^k$. A plan t^j to achieve task t is a sequence of executable primitive actions $t^j = \{m_1, m_2, ..., m_n\}$. Each plan t^j has an associated *utility distribution* UD_{t^j} and *duration distribution* DD_{t^j}.

Example: Suppose agent A is one of the agents in the multi-agent system. Agent A at time 44 is executing action m which is on its schedule. Suppose the expected utility goodness (defined below) of the tasks remaining on the schedule is 0.454 with $MEDIUM$ deadline.

Task $T1$ arrives at time 45 and has a deadline of 100. The execution of method m is interrupted by the arrival event and m still needs about 8 time units to complete execution.

$T1^A$ and $T1^B$ are two alternate ways to achieve task $T1$ where

$$UD_{T1^A} = (10\%\ 30\ 90\%\ 45)$$
$$DD_{T1^A} = (25\%\ 22\ 50\%\ 50\ 25\%\ 100)$$
$$UD_{T1^B} = (25\%\ 20\ 75\%\ 30)$$
$$DD_{T1^B} = (50\%\ 32\ 30\%\ 40\ 20\%\ 45)$$

Suppose CT_t is the time required for scheduling a task t if it is chosen for scheduling. R_m is the remaining time required for primitive action m to complete execution. Then the earliest start time EST_t for a task t is the arrival time

AT_t of the task delayed by the sum of R_m, the time required for completing the execution of the action m which is interrupted by a meta-level control event and CT_t, the time required for scheduling the new task.

$$EST_t = AT_t + R_m + CT_t$$

Example: Suppose the time to schedule a task like $T1$ is 5 units on average. AT_t and R_m are 45 and 8 respectively. Then $EST_{T1} = 58$.

The maximum available duration MD_t for a task t is the difference between the deadline of the task and its earliest start time.

$$MD_t = DL_t - EST_t$$

Example:

$$MD_{T1} = 100 - 58 = 42$$

Given a task t and its maximum available duration MD_t, the probability that a plan t^j meets its deadline PDL_{t^j} is the sum of the probabilities of all values in the duration distribution of plan t^j which are less than the maximum available duration of the task.

$$PDL_{t^j} = \sum_{j=1}^{n} \frac{p_j}{100} : ((p_j\%\ x_j) \in DD_{t^j}) \wedge (x_j < MD_t)$$

Example: Since $MD_{T1} = 42$, there is only one duration value in DD_{T1^A} which has a value less than 42 and that value is 22 and occurs 25% of the time.

$$PDL_{T1^A} = \frac{25}{100} = 0.25$$

There are two duration values in DD_{T1^B} which have a value less than 42 and they are 32 and 40 which occur 50% and 30% respectively in the distribution.

$$PDL_{T1^B} = \frac{50\ +\ 30}{100} = 0.8$$

The expected duration ED_{t^j} of a plan t^j, is the expected duration of all values in the duration distribution of plan t^j which are less than the maximum available duration for the task.

$$ED_{t^j} = \frac{\sum_{j=1}^{n} \frac{p_j}{100} * x_j}{PDL_{t^j}} : ((p_j\%\ x_j)\ \in DD_{t^j}) \wedge (x_j < MD_t)$$

Example:

$$ED_{T1^A} = \frac{(\frac{25}{100} * 22\ )}{0.25} = 22$$

$$ED_{T1^B} = \frac{(\frac{50}{100} * 32\ +\ \frac{30}{100} * 40)}{0.8} = 35$$

The expected utility EU_{t^j} of a plan t^j, is the product of the probability that the alternative meets its deadline and the expected utility of all values in the utility distribution of alternative t^j.

$$EU_{t^j} = \sum_{j=1}^{n} PDL_{t^j} * \frac{p_j}{100} * x_j : ((p_j\%\ x_j) \in UD_{t^j})$$

Example:

$$EU_{T1^A} = 0.25 * \frac{10}{100} * 30\ +\ 0.25 * \frac{90}{100} * 45 = 10.875$$

$$EU_{T1^B} = 0.8 * \frac{25}{100} * 20\ +\ 0.8 * \frac{75}{100} * 30 = 22$$

Given the maximum available duration for a task, the preferred alternative ALT_t for a task t is the alternative whose expected utility to expected duration ratio is the highest. ALT_t is the alternative which has the potential obtain the maximum utility in minimum duration within the given deadline.

$$ALT_t = t^j : \max_{j=1}^{n} \frac{EU_{t^j}}{ED_{t^j}}$$

Example: Consider each of $T1$'s alternative plans which were described earlier. Plan $T1^A$'s expected utility to expected duration ratio is $\frac{10.875}{22} = 0.494$ and plan $T1^B$'s expected utility to expected duration ratio is $\frac{22}{35} = 0.629$. So the alternative with the maximum expected utility to expected duration ratio is $T1^B$

$$ALT_{T1} = T1^B$$

The utility goodness UG_t of a task t is the measure which determines how good the expected utility to expected duration ratio of the preferred alternative of a task is in relation to the expected utility to expected duration ratio of the preferred alternatives of all the other tasks which arrive at the system.

The tasks with high utility are the tasks which are in the 66th percentile (top 1/3rd) of the expected utility to expected duration ratio of the preferred alternative of a task.

$$UD_t = \begin{cases} HIGH, & \frac{EU_{ALT_t}}{ED_{ALT_t}} > 66th\ percentile \\ MEDIUM, & \frac{EU_{ALT_t}}{ED_{ALT_t}} > 33rd\ percentile \\ LOW, otherwise \end{cases}$$

Example: The utility goodness of task $T1$ given a deadline of 100 and a maximum available duration of 42 is $\frac{22}{35} = 0.628$ which lies above the 66th percentile. $UG_{T1} = HIGH$. The deadline tightness for this example was computed in a similar fashion and found to be $TIGHT$. The utility goodness of $T1$ is higher than the current scheduled tasks whose utility goodness is only 0.454 and $MEDIUM$ deadline. Hence, Task $T1$ has higher priority.

The other features mentioned earlier are determined in a similar principled fashion. Based on these features and a decision process, agent A determines that the best action

would be to *complete method m, drop the current schedule, detailed schedule $T1$* which means the agent should complete the current method in execution (this is required to maintain consistency of the simulation), then discard the current schedule and reinvoke the scheduler.

We have constructed a number of decision processes based on these features. The most interesting one is based on learning and is described in the next section.

3. Learning a Meta-level Control Policy

We formulate the meta-level control problem in terms of a finite state Markov decision process [1] (MDP) with discounted return. The MDP state is the factored, abstract description of the state features presented in the previous section. Control actions define the action space of the MDP and reward is the utility accrued when a high level task is completed. We use a learning approach based on [9] where the probability transitions and reward function for the MDP can be generated when there is a restricted set of training data. Since we account for real-time control costs, each of our simulation runs takes approximately four minutes and this makes collecting training data a bottle neck.

To implement this approach, we first construct an initial meta-level control policy which randomly chooses an action at each state and collects a set of episodes from a sample of the environment. Each episode is a sequence of alternating states, actions and rewards. As described in [9], we estimated transition probabilities of the form $P(s'|s,a)$, which denotes the probability of a transition to state s', given that the system was in state s and took action a from many such sequences. The transition probability estimate is the ratio of the number of times in all the episodes, that the system was in s and took a and arrived at s' to the number of times in all the episodes, that the system was in s and took a irrespective of the next state. The Markov decision process (MDP) model representing system behavior for a particular environment is obtained from state set, action set, transition probabilities and reward function. In the interest of space, we refer the reader to [6] for further details. The efficiency of the model depends on the extent of exploration performed in the training data with respect to the chosen states and actions. In the final step we determine the optimal policy in the estimated MDP using the Q-value version of the standard value iteration algorithm [10].

To the extent that the estimated MDP is an accurate model of the particular environment, this optimized policy should maximize the reward obtained in future episodes. The algorithm is as follows:

1. Choose an appropriate reward measure for episodes and an appropriate representation for episode states.
2. Build an initial state-based training system that creates an exploratory data set. Despite being exploratory, this system should provide the desired basic functionality.
3. Use these training episodes to build an empirical MDP model on the state space.
4. Compute the optimal meta-level control policy according to this MDP.
5. Reimplement the system using the learned meta-level control policy

4. Experiments

The agents in this domain are in a cooperative environment and have approximate models of the others agents in the multi-agent system. The agents are willing to reveal information to enable the multi-agent system to perform better as a whole. The interaction between 2 agents is studied. The multi-agent aspect of the problem arises when there is task requiring coordination with another agent. The agent rewards in this domain are neither totally positively correlated (team problem) nor are they totally negatively correlated (zero-sum game) but rather is a combination of both.

The meta-level control decisions that are considered in the multi-agent set up are: when to accept, delay or reject a new task, how much effort to put into scheduling when reasoning about a new task, whether to reschedule when actual execution performance deviates from expected performance, whether to negotiate with another agent about a non-local task and whether to renegotiate if a previous negotiation falls through. For all the experiments, the following costs are assumed. The meta-level control actions have an associated cost of 1 time unit; the drop task and delay task actions take 1 time unit also. The decision to negotiate and whether to renegotiate also take 1 unit of time. The call to simple scheduler costs 2 time units and the cost of computation of complex features costs 2 time units, the cost of detailed scheduling tasks with less than five methods is 4 units, with less than ten methods is 12 time units and greater than ten methods is 18 time units.

The task environment generator in the multi-agent setup also randomly creates task networks while varying three critical factors:
complexity of tasks $c \in \{simple(S), complex(C), combo(A)\}$
frequency of arrival $f \in \{high(H), medium(M), low(L)\}$
tightness of deadline $dl \in \{tight(T), medium(M), loose(L)\}$.
Complexity of tasks as described earlier refers to the expected utilities of tasks and the number of alternative paths available to complete the task. A simple task, in the multi-agent setup, has two to four primitive actions. A simple task has an average duration 18 time units and a complex task has an average duration of 25 time units. A complex task has four to six primitive actions.

The frequency of arrival of tasks refers to the number of

tasks that arrive within a finite time horizon. The resource contention among the tasks increases as the task frequency increases. Task arrival is determined by a normal distribution with $\mu = 250$ and $\sigma = 249$. When the frequency of arrival is low, about one to ten tasks arrive at the agent in 500 time unit horizon; medium implies between ten and fifteen tasks; and high implies between fifteen and twenty tasks. The tightness of deadline refers to the parameter defined in the previous section and it is task specific. The resource contention is also proportional to the deadline tightness. If the deadline tightness is set to low, the maximum available duration given to the task is between 120% and 150% of the expected duration of the task; medium, the maximum available duration is between 100% and 120% of the expected duration of the task; and if high, the maximum available duration is between 80% and 100% of the expected duration of the task.

An environment named AMM has a combination(A) of simple and complex tasks arriving at a medium frequency(M) and with medium tightness of deadline(M).

We first tested the effectiveness of meta-level control using the two hand-generated context sensitive heuristic strategies NHS and SHS. The **myopic strategy**, NHS, uses state-dependent heuristics based on high-level features to decide which deliberative (also called control) decision to make. These heuristics are myopic and do not reason explicitly about the arrival of tasks in the near future. A sample NHS heuristic used to make a decision when a new task arrives would be "If new task has high priority; current schedule has low utility goodness, then best action is *drop its current schedule and schedule the new task immediately* independent of the schedule's deadline."

The **non-myopic strategy**, SHS, is a set of rules that use knowledge about environment characteristics to make non-myopic decisions. The knowledge of the task arrival model enables the SHS to make decisions that in the long term minimize the resources spent on redundant deliberative actions. An example SHS heuristic is "If new task has very low utility goodness, loose deadline; low probability of a high priority tasks arriving in the near future, then best action is *schedule new task using simple scheduling*."

The complete list of NHS and SHS heuristic rules and their descriptions can be found in [5] (Chapter 4). The features presented in Section 2 were selected based on what intuitively made sense for meta-level control. We use the following experiments based on the heuristic rules to verify the usefulness of these features. NHS and SHS were compared to base-line approaches which used a random and deterministic strategy respectively.

Performance comparison of the various strategies in environment AMM, over a number of dimensions and averaged over 300 test episodes are described in Table 1. Column 1 is row number; Column 2 describes the various comparison criteria; Columns 3-6 represent each of the four algorithms; Rows 1 and 2 show the average utility gain (AUG) and respective standard deviations (σ) per run; Row 3 shows the percentage of the total 500 units spent on deliberative/control actions(CT); Row 4 is the percentage of tasks rescheduled (RES); Row 5 is the percent of total tasks completed (PTC);Row 6 is percent of tasks delayed on arrival (PTDEL). The results show that the combined utilities of the two agents when using the heuristic strategies is significantly higher than the combined utilities when using the deterministic and random strategies. The utility obtained from using SHS is significantly higher ($p < 0.05$) than NHS and also 14% more tasks are completed using SHS than the NHS. The reason for the improved performance by the heuristic strategies is explained by comparisons of the percent of control time over the same set of environments. As described earlier, control actions do not have associated utility of their own. Domain actions produce utility upon successful execution and the control actions serve as facilitators in choosing the best domain actions given the agent's state information. So resources such as time spent directly on control actions do not directly produce utility. When excessive amounts of resources are spent on control actions, the agent's utility is reduced since resources are bounded and are not available for utility producing domain actions. The trend in other factors such as percent of reschedules, tasks completed and delayed tasks are also affected by the resources spent on control actions.

Row#		SHS	NHS	Deter.	Rand.
1	AUG	111.44	89.84	77.56	45.56
2	σ	2.33	6.54	12.45	15.43
3	CT	9.21%	8.09%	14.28%	7.15%
4	RES	0%	14.28%	19.93%	1.49%
5	PTC	71.32%	56.34%	54.17%	57.78%
6	PTDEL	8.8%	3.98%	0%	59.96%

Table 1. Performance evaluation of four algorithms for two agents in an environment AMM

The heuristic strategies use control activities that optimize their use of available resources (time in this case). The deterministic strategy has higher control costs because it always makes the same control choice (due to the absence of explicit meta-level control), the expensive call to the detailed scheduler, independent of context. The random strategy has low control costs but it does not reason about its choices leading to bad overall performance.

Experimental results describing the behavior of the two interacting agents in 3 environments AMM, AML and ALM averaged over 300 test episodes are presented in Table 2 where Column 1 is the environment type; Column 2 repre-

Environment	RL-3000	SHS	NHS
AMM-UTIL	118.56	111.44	89.84
AMM-CT	8.86%	9.21%	8.09%
AML-UTIL	211.45	207.88	130.68
AML-CT	22.56%	23.21%	40.89%
ALM-UTIL	136.05	113.83	92.56
ALM-CT	10.21%	13.11%	15.88%

Table 2. Utility and Control Time Comparisons over four environments

sents the performance characteristics of the RL policy after 3000 training episodes; Column 3 and 4 represent the performance characteristics of SHS and NHS respectively. Details about the generation of the training episodes can be found in [5] (Chapter 5). In these tests, one agent was fixed to the best policy it was able to learn in a single agent environment. The other agent then learned its meta-level control policy within these conditions.

The results show that the combined utilities of the two agents when using the RL strategy is as good as the SHS strategy which uses environment characteristic information in its decision making process. The ability of the RL algorithm to make non-myopic decisions leads to optimized use of its resources (time) and hence learn policies which significantly outperforms NHS ($p < 0.05$) in these environments and does as well as if not better than SHS. There are twelve features in the abstract representation of the state and each feature can have one of four different values, so the maximum size of the search space is $4^{12} = 2^{24}$, which is about a million states. Of these million states, only about a 100 states on average are visited with high frequency for a specific environment. This is in interesting area of future work where we would like to study the characteristics of the frequently visited states over multiple environments.

5. Discussion

The experimental evaluation leads to the following conclusions: Meta-level control reasoning is advantageous in resource-bounded agents in environments that exhibit non-stationarity, action outcome uncertainty and partial-observability; the high-level features are good indicators of the agent state and facilitate effective meta-level control; the domain independence of these high level features allows the solution approach to be generalized to domains where tasks have associated deadlines and utilities, there are alternate ways of achieving the tasks, uncertainty in execution performance and detailed scheduling and coordination of tasks may be required.

We describe a reinforcement learning approach which equips agents to automatically learn meta-level control policies. The empirical reinforcement learning algorithm used is a modified version of the algorithm developed by [9] for a spoken dialog system. Both problem domains have the bottle neck of collecting training data. The algorithm optimizes the meta-level control policy based on limited training data. The utility of this approach is demonstrated experimentally by showing that the meta-level control policies that are automatically learned by the agent perform as well as the carefully hand-generated heuristic policies.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientifi c, Belmont, MA, 1996.

[2] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments, 1994.

[3] E. A. Hansen and S. Zilberstein. Monitoring anytime algorithms. *SIGART Bulletin*, 7(2):28–33, 1996.

[4] E. J. Horvitz. Reasoning under varying and uncertain resource constraints. In *National Conference on Artificial Intelligence of the American Association for AI (AAAI-88)*, pages 111–116, 1988.

[5] A. Raja. Meta-level control in multi-agent systems. *PhD Thesis, Computer Science Department, University of Massachusetts at Amherst*, September 2003.

[6] A. Raja and V. Lesser. Reasoning about Coordination Costs in Resource-Bounded Multi-Agent Systems. In *In Proceedings of AAAI 2004 Spring Symposium on Bridging the multiagent and multirobotic research gap, Stanford, CA, March 2004, Pages 35-40*, March 2004.

[7] S. Russell and E. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 400–411, 1989.

[8] S. J. Russell, D. Subramanian, and R. Parr. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 338–344, 1993.

[9] S. P. Singh, M. J. Kearns, D. J. Litman, and M. A. Walker. Empirical evaluation of a reinforcement learning spoken dialogue system. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 645–651, 2000.

[10] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.

[11] T. Wagner, A. Garvey, and V. Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also available as UMASS CS TR-97-59.