2007

A Framework for Meta-level Control in Multi-Agent Systems

Anita Raja
University of North Carolina at Charlotte

Victor Lesser
University of Massachusetts - Amherst

A Framework for Meta-level Control in Multi-Agent Systems

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

Sophisticated agents operating in open environments must make decisions that efficiently trade off the use of their limited resources between dynamic deliberative actions and domain actions. This is the meta-level control problem for agents operating in resource-bounded multi-agent environments. Control activities involve decisions on when to invoke and the amount to effort to put into scheduling and coordination of domain activities. The focus of this paper is how to make effective meta-level control decisions. We show that meta-level control with bounded computational overhead allows complex agents to solve problems more efficiently than current approaches in dynamic open multi-agent environments. The meta-level control approach that we present is based on the decision-theoretic use of an abstract representation of the agent state. This abstraction concisely captures critical information necessary for decision making while bounding the cost of meta-level control and is appropriate for use in automatically learning the meta-level control policies.

Keywords: Multi-Agent Systems, Bounded Rationality, Meta-level Control Architecture

1 Introduction

Open environments are dynamic and uncertain. Complex agents operating in these environments must reason about their deliberations in real-time. Deliberative control actions include scheduling domain-level actions and coordinating with other agents to complete tasks requiring joint effort. These deliberations may involve computation and delays waiting for arrival of appropriate information. Furthermore, new tasks can be generated by agents at any time. Tasks have deadlines, where completing the task after the deadline could lead to lower or no utility. We define meta-level control as the ability of complex agents operating in open environments to sequence domain and deliberative actions to optimize expected performance. Meta-level control supports decisions on when to accept, delay, or reject a new task; when it is appropriate to negotiate with another agent; whether to renegotiate when a negotiation task fails; how much effort to put into scheduling when reasoning about a new task; and whether to reschedule when actual execution performance deviates from expected performance. These decisions influence each other and affect the amount of resources available for future computations. The intent of this paper is to show that a meta-level reasoning component with bounded and small computation overhead can be constructed such that it significantly improves the overall performance of agents in a cooperative multi-agent system. Further, we show that appropriately abstracting the agent state is key to the development of the meta-level control component and that such abstraction can be the basis for automatically learning meta-level control policies.

A problem with most single-agent and multi-agent systems [4, 19, 23, 29, 35, 58] is that they do not explicitly reason about the cost of deliberative computation because they assume all deliberative computations are always done and always done in the same way. Thus, most systems, have no way to trade off the resources used for deliberative actions and domain actions. An agent is not performing rationally if it fails to account for all the costs involved in achieving a desired goal. Failure to account for all costs could potentially lead to agents taking actions that are without operational significance [43]. Taking the entire cost of computation into account leads to what Simon calls procedural rationality, Good refers to as type II rationality [13] and what Russell and Wefald refer to as bounded rationality [37]. An agent exhibits such bounded rationality if it maximizes its expected utility given its computational and other resource limits. If significant resources are expended on making this meta-level control decision, then ”meta-meta”-level decisions have to be made on whether to spend these resources on meta-level control. However, if the meta-level reasoning process has a small computational overhead, there is no need for explicit meta-meta level reasoning. In this work, we avoid infinite regress of the meta-level control problem by ensuring that the meta-level reasoning process has a small and bounded computational overhead.

Meta-level control can be viewed as a sequential decision problem. The essence of sequential decision problems is that decisions that are made now can have both immediate and long-term effects; the best current action choice depends on the types of future situations the agent will face and the action choices that have to be made at those future decision points. For instance, the resource-bounds of the agent cause the current meta-level action choices to affect the resources available to future action choices. Effective meta-level control also needs to use past performance information to make predictions about the future to make non-myopic decisions at each decision making point. This is in contrast to myopic decision making which tries to optimize only the next state of the system. Some of the characteristics of the meta-level problem in this work that make it difficult are the complexity of the information that characterizes the state of the agent and other agents it interacts with; variety of responses with differing costs and parameters available to the situation; deadlines associated with these tasks; high degree of uncertainty caused by the non-deterministic arrival of tasks and outcomes of primitive domain actions; consequence of decisions are often not observable immediately and may have significant down-stream effects. To our knowledge, meta-level control for such a complex agent environment has not been studied.

In our work, each agent will have its own meta-level control component and the meta-level control policy will be computed offline. The problem environment is cooperative as each agent has its individual goals to achieve and some of these goals require cooperation of other agents. The agents are trying to maximize the sum of the utilities attainable by the multi-agent system as a whole. The following assumptions are made in this paper:

- The agents are cooperative and will prefer alternatives that increase social utility/quality even if it is at the cost of decreasing local utility.

- An agent may concurrently pursue multiple high-level goals and completing a goal derives utility for the system or agent. The high-level goals are generated either by sensing internal event triggers or by receiving requests for assistance from other agents.

- The high-level goals must often be completed by a certain time in order to achieve any utility.
- It is not necessary for all high-level goals to be completed in order for an agent to derive utility from its activities, and partial satisfaction of a high-level goal is sometimes permissible while trading-off the amount of utility derived for decrease in resource usage.
- The overall objective of the system or agent is to maximize the utility generated over some finite time horizon. Although a fixed horizon is used in the experiments, this information is not provided to the agents. This was deliberately done to equip the agents to operate in domains and environments with indefinite horizons (an unknown finite horizon).

To equip the agents with the ability to perform this type of meta-level control, we augment the classic agent architecture [36] with meta-level control that reasons about the deliberative actions (also called control actions) and alternative ways of performing them. Figure 1 describes the meta-level control agent architecture. The arrival of percepts trigger the meta-level control layer to determine the tasks which the agent desires to pursue. The agent's control layer determines how these chosen tasks will be processed and mapped into action sequences.

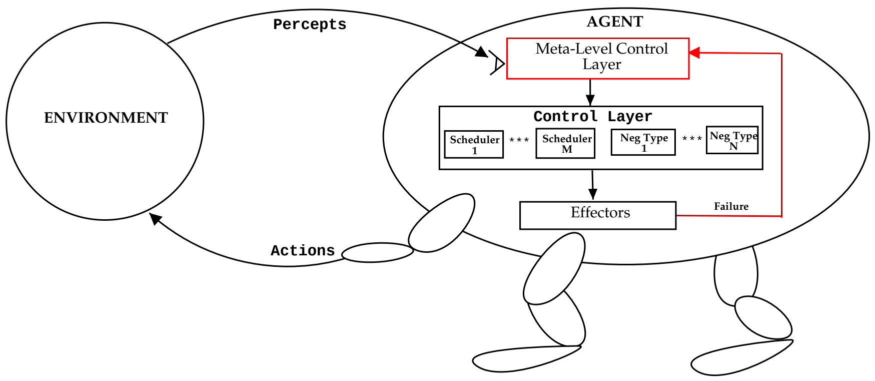


Figure 1: Meta-level control architecture for a bounded rational agent

The three classes of deliberative actions discussed in this paper are: information gathering actions, planning/scheduling actions and coordination actions. The first type of deliberative action is information gathering which can be of two kinds: *gathering information about the environment*; and *determination of complex state features*. The environmental information gathered by an agent including information about the state of other agents, is used by its meta-level controller to determine the relevant control actions. These external information gathering actions do not use significant local processor time but they delay the meta-level deliberation process because of the end-to-end delay of getting information from other agents. However, information gathering that involves determination of complex state features of the agent can involve a significant amount of local computation. These features, for instance, can compute detailed timing, placement and priority information about the primitive actions which have to be executed to complete the agent's tasks. The agent must make explicit meta-level control decisions on whether to gather complex features and determine which complex features are appropriate.

The second type of deliberative action involves planning and scheduling. Planning is the process in which the agent uses beliefs about actions and their consequences to search for solutions to one or more high-level tasks (goals) over the space of possible plans. It determines which domain actions should be taken to achieve the tasks. Scheduling is the process of deciding when and where each of these actions should be performed. In this paper, planning is integrated with scheduling. The agent's scheduling decisions involve choosing which subset of these high-level goals to pursue and how to go about achieving them. The meta-level control decision is to decide whether to invoke a scheduler, which scheduler to invoke, and how much resources to invest in the scheduling process. Finally, the third type of deliberative action, coordination, is the process by which a group of agents achieve their tasks in a shared environment. In this research, coordination is the inter-agent negotiation process that establishes commitments on finish times of tasks or methods done by one agent in the context of constraints of another agent's activities. The meta-level control decisions on coordination involve choosing the tasks that require coordination, deciding whether to coordinate with another agent and how much effort to spend on coordination. We make the simplifying assumption that results of coordination are binding and that other agents will not decommit from their commitments at later stages. We believe that as we build more advanced agents operating as a group in less predictable real-time environments, reasoning about agent activities from a meta-level perspective will be crucial for effective agent operation.

The paper is structured as follows: We first present the meta-level agent architecture which can support reasoning about costs at all levels of the decision making process; various meta-level decisions that need to be made; and the state information necessary to make these decisions. A description of high-level features that capture the state information concisely while bounding the size of the state space is also provided. We then describe a formal model of the problem with an emphasis on the sequential decision making process that is involved. We discuss the difficulty in using this formal model for this complex problem which motivates our approximate solution method which capitalizes on the ability to model and use an abstract representation of the state. We then describe two strategies based on hand-generated heuristics: the Naive Heuristic Strategy and the Sophisticated Heuristic Strategy. They differ in the amount of environmental information available as part of the system state. These strategies use the high-level features that will be provided to the meta-level learning strategy. Snapshots of the meta-level reasoning process for specific exogenous events are also presented. The performance of the hand-generated strategies provide a sanity check on the effectiveness of the state features to allow for effective meta-level control. Based on the positive results of the previous section, we show that a reinforcement learning strategy based on the abstract state features can be used to learn meta-level control policies within a reasonable number of learning episodes. These policies are shown to be as effective as those that are hand-generated. We then conclude the paper with a review of the important ideas presented in the paper and experimental results and briefly discuss future work.

2 An Agent Architecture with Meta-Level Control

Meta-level control is the process of optimizing an agent's performance by choosing and sequencing domain and control activities. In this section we present a classic agent architecture augmented with a meta-level control component. This includes a description of the interaction among the various components in the architecture and the agent's ability to reason about control costs as first class entities. A high-level representation of the state which captures the critical information while bounding the computation required to process the state is also described.

We will describe the role of meta-level control in the agent architecture by concentrating on the control flow among the various components (see Figure 2). In this architecture, the control components such as the schedulers, negotiation components and execution subsystem interact with the meta-level control (MLC) component. The MLC is invoked when certain exogenous or internal events occur (e.g. the request by another agent to perform a task for it). Both the meta-level and control components are involved in the agent decision making process. There are a number of data structures which help keep track of the agent's state. The NewTask List contains the tasks which have just arrived at the agent from the environment. The Agenda List is the set of tasks which have arrived at the agent but the reasoning about how to achieve the tasks has been delayed. They have not been scheduled yet. The Schedule List is the set of high-level tasks chosen to be scheduled and executed. The Execution List is the set of primitive actions which have been scheduled to achieve the high-level tasks and maybe in execution or yet to be executed. Examples of the decision making process corresponding to particular agent states are provided later in the section.

The meta-level is invoked when a new task arrives at the agent, even if the agent is in the midst of executing another task. The execution subsystem is invoked whenever the agent has to act upon the environment. These actions may or may not have immediate rewards. When an action completes execution, the execution subsystem sends the execution characteristics to the meta-level controller which is also the monitoring subsystem.

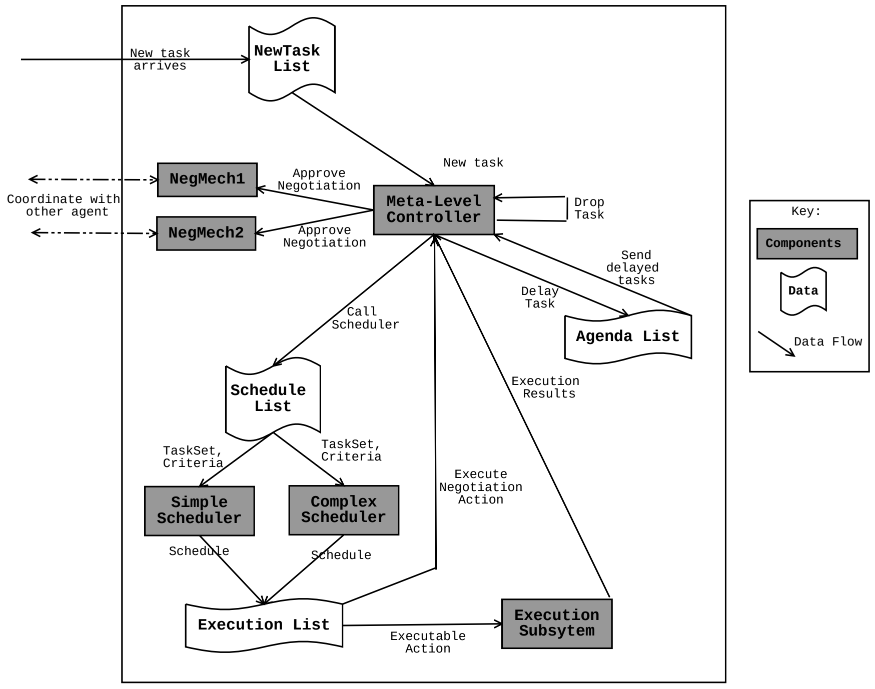


Figure 2: Control flow in meta-level control agent architecture

The control layer may consist a number of schedulers and negotiation protocols. For the purposes of our discussion, we consider two schedulers, simple and complex, and two negotiation protocols that differ in their performance profiles.

Simple Scheduler: The simple scheduler is invoked by the meta-level controller and receives the task structure and goal criteria as input. It selects the most appropriate schedule for the current context from a set of pre-computed task schedules. This choice does not take into account other tasks that could be simultaneously scheduled with this task. When an agent has to schedule a task but doesn't have the resources or time to call the complex domain-level scheduler, the pre-computed information about the possible schedules of the task structure can be used to provide a reasonable but often non-optimal schedule. The agent gathers knowledge about all tasks that it is capable of executing by performing off-line analysis on each task. This off-line process constructs potential schedules in the form of linear sequences of primitive actions. Each sequence has associated performance characteristics such as expected quality distribution, expected duration distribution, and expected duration uncertainty for achieving the high level tasks. These performance characteristics are discovered by systematically searching over the space of objective criteria. The task abstraction hides the details of these schedules and provides only the high level information necessary to make meta-level choices.

Complex Scheduler: The domain level scheduler depicted in the architecture is an extended version of the Design-to-Criteria (DTC) scheduler [54]. Design-to-Criteria (DTC) scheduling is the 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. Casting the language into an action-selecting-sequencing problem, the process is to select a subset of primitive actions from a set of candidate actions, and sequence them, so that the end result is an end-to-end schedule of an agent's activities that meets situation specific design criteria. If the meta-level action is to invoke the complex scheduler, the scheduler component receives the task structure, objective criteria and a set of scheduler parameters as input and outputs a satisficing schedule as a sequence of primitive actions. The complex scheduler, in contrast to the simple scheduler, can reason about and schedule multiple tasks simultaneously. A detailed description of the scheduler parameters are provided later on in this section.

Negotiation Protocols: There are two types of negotiation protocols [57]: **NegMech1** and **NegMech2**. The choice of the exact negotiation protocol will depend on the relative gain of doing the associated task and the likelihood of the other agent doing the task. NegMech1 is a single-shot negotiation protocol that works in an all or nothing mode. A single proposal is sent out and single response is received. It is inexpensive but has a lower probability of success than the other negotiation protocol. NegMech2 is a multi-step negotiation protocol which tries to achieve a commitment by a sequence of proposals and counter-proposals until a consensus is reached or time runs out. It is more expensive than the single-shot protocol because of the computation and communication overhead. It, however, has a higher probability of success.

We will now discuss how the architecture equips the agent with the capability to adapt to changing conditions in an unpredictable environment. This architecture accounts for computational and execution cost at all three levels of the decision hierarchy: domain, control and meta-level control activities. The cost of domain activities is modeled directly in the task structures which describe the tasks. Domain activities are reasoned about by control activities like scheduling and coordination. Performance profiles of the various control activities are used to compute their costs and are reasoned about by the meta-level controller. Meta-level control activities in this architecture are modeled as activities with small yet non-negligible costs which are incurred by the computation of state features which facilitate the decision-making process. These costs are accounted for by the agent, whenever events trigger meta-level activity. The state features and their functionality are described in greater detail below. This MLC architecture is an open architecture in that the modules belonging to the various layers can be replaced by modules with better performance characteristics and the advantages of the architecture described below will still hold true [34].

There are five types of event triggers that require meta-level decision making in our framework.

1. Arrival of a new task from other agents or the external environment.
2. Presence of a task in the current task set that requires negotiation with a non-local agent[1].
3. Failure of a negotiation to reach a commitment.

[1]Another agent in the multi-agent system that owns the task or method that enables a task or method in the local agent of concern. These tasks and methods are called non-local tasks and non-local methods respectively.

4. Domain action completes execution requiring a check to see if there is a significant deviation of online schedule performance from expected performance.

5. Decision to schedule a new set of tasks or to reschedule existing tasks.[2]

These particular event triggers were chosen because they occur frequently in the domain described in this paper. It is our view that most meta-level decisions in other multi-agent applications [34] could be mapped into one of these five event trigger classes.

In order to illustrate the meta-level control decision making process, we describe a simple scenario consisting of two rovers *RoverA* and *RoverB*. Rovers are unmanned vehicles equipped with cameras and a variety of scientific sensors for the purpose of planetary surface exploration. The discussion here will focus on the various meta-level questions that will have to be addressed by *RoverB*. Figure 3 describes *Assist Sample Collection*, also called task *S0* which is performed by *RoverA*; *Analyze Rock*, also called task *T0*, and *Explore Terrain*, also called task *T1*, which are the tasks performed by *RoverB*; as well as the non-local enables relationship that exists between *RoverA* and *RoverB*.

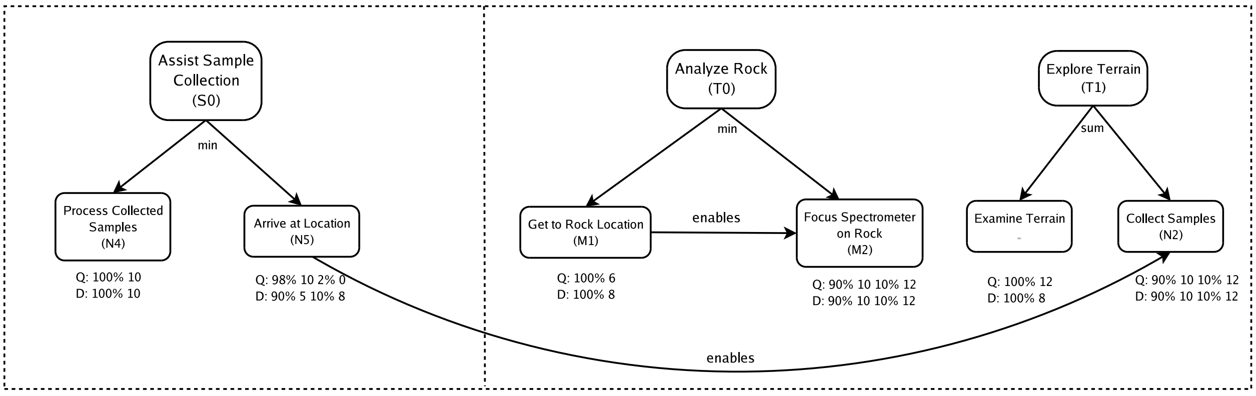


Figure 3: Task *Assist Sample Collection* belongs to RoverA; Tasks *Analyze Rock* and *Explore Terrain* belong to *RoverB*

In this example, each top-level task, as described in the TÆMS task description language [7], is decomposed into two executable primitive actions. In order to achieve the task *Analyze Rock*, *RoverB* must execute both primitive actions *Get To Rock Location* and *Focus Spectrometer on Rock* in sequence. All primitive actions in TÆMS called *methods*, are statistically characterized in two dimensions: quality and duration. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to overall problem solving. Thus, different applications have different notions of what corresponds to model quality. The sequence is denoted by the enables arrow between the two actions and the min quality attribution factor (which denotes a conjunction operator) states that the minimum of the qualities of the two actions will be attributed to the *Analyze Rock* task. To achieve the task *Explore Terrain*, *RoverB* can execute one or both primitive actions *Examine Terrain* and *Collect Samples* within the task deadline and the quality accrued for the task will be cumulative (denoted by the *sum* function).

RoverA is equipped with a storage compartment while *RoverB* is not. The *Collect Samples* method requires *RoverA* and *RoverB* to coordinate: *RoverB* has the ability to pick up the soil sample and put it in RoverA's storage compartment. This relationship between the two agents is denoted by the non-local

[2]This meta-level control decision is triggered as a consequence of one of the four decisions mentioned above.

enables from *RoverA*'s *Arrive at Location (N5)* method to *RoverB*'s *Collect Samples* method. Utility and duration distributions for each primitive action are provided.

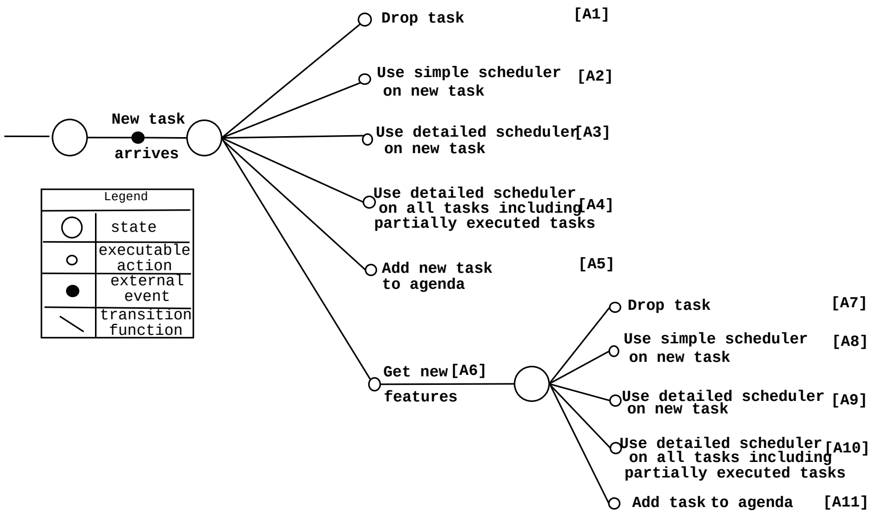
To illustrate the trade-offs involved in the meta-level decision making process, we frame the meta-level questions for the **Arrival of new task** event trigger in the context of the rover example. We then describe the cost/benefit trade-offs. In the interests of space, we refer the readers to [33] for the cost/benefit trade-offs for the other four meta-level event triggers as well as detailed time-line execution trace of a sample run of the example. **Event Trigger:** Arrival of a new task from the environment.
Meta-Level Question: Should *RoverB* schedule a new task immediately at the time it became known or postpone scheduling to sometime in the future or drop that particular instance of the task.
Benefit: If the new task has low expected utility, its deadline is very close and there is a high probability of a high utility task arriving in the future, then it should be discarded. This means *RoverB* chooses not to expend its limited resources on a low priority task and instead will wait for a future high priority task. If the incoming task has very high priority, in other words, the expected task utility is very high and it has a relatively close deadline, then *RoverB* should override its current schedule and schedule the new task immediately. If the current schedule has average utility that is significantly higher than the new task and the average deadline of the current schedule is significantly closer than that of the new task, then reasoning about the new task should be postponed till later.
Cost: There is the cost associated with this meta-level control decision. This is a small, fixed cost as described in the next section. Additionally, if the new task is scheduled immediately regardless of its expected utility or deadline, then its possible that the opportunity cost[3] of scheduling that task can be very high. Scheduling the new task has an associated cost in time units in addition to costs for dropping established commitments if the previous schedule is significantly revised or completely dropped. These costs can be diminished or avoided completely if the decision about the new task is postponed to later or completely avoided if the task is dropped.

We now describe how the MLC handles the five events and their corresponding set of possible action choices. Each of the external events and corresponding meta-level decisions has an associated decision tree. The external action triggers a state change. The response actions, execution of domain action or complex feature computation, are also modeled in the decision tree. We use the rover example to illustrate the state representation for the *Arrival of a new task* and *Presence of task requiring coordination in current task set* event triggers. We refer the reader to [33] for details relating to the rover example for the remaining event triggers.

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 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-A8 is shown in Figure 4. Scheduling actions have 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 by adding it to the agenda of unscheduled tasks [A5] or completely avoided if the task is dropped [A1]. If a task is of high priority relative to other tasks in execution or on the agenda, the meta-level controller might decide to use the complex scheduler to schedule the task [A3]. If the new task is of high priority and the currently executing schedule is also of high priority, the meta-level controller could decide to reschedule all the tasks using the detailed scheduler [A4]. If there are tight constraints on scheduling the task, the simple scheduler could be invoked [A2]. The

[3]Resources that could have been invested in future high priority tasks are instead invested in lower priority tasks leading to lower overall utility gains.

meta-level controller could also determine 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]. The meta-level controller can hence choose to spend more resources to make a better informed decision. After getting the additional state information, the meta-level control will choose from one of the five possible choices described earlier (A7-A11).[4]



Drop task [A1]
Use simple scheduler on new task [A2]
Use detailed scheduler on new task [A3]
Use detailed scheduler on all tasks including partially executed tasks [A4]
Add new task to agenda [A5]
New task arrives
Get new features [A6]
Drop task [A7]
Use simple scheduler on new task [A8]
Use detailed scheduler on new task [A9]
Use detailed scheduler on all tasks including partially executed tasks [A10]
Add task to agenda [A11]
Legend
state
executable action
external event
transition function

Figure 4: Decision tree when a new task arrives

To illustrate this control process, instances of the state of agent *RoverB* and the corresponding decision choice made by the meta-level controller are provided. These are hand-generated rules specifying the required type of meta-level control decisions. These will be represented as rules in the heuristic strategies and learned automatically in the reinforcement learning strategy.

An example of the above described decision process occurs when the *RoverB* is in State *S1*. It represents the situation at time 2. *RoverB* is in a wait state doing nothing when a new task *Analyze Rock*, which arrives at time 1 with a deadline of 40, is added to the NewTaskList. *RoverB's* meta-level controller is invoked. All the other lists are empty and *RoverB* has not executed any task and has accrued zero utility. Based on its current state, *RoverB's* meta-level control decision is to *Call the Detailed Scheduler*.

State S1:

CurrentTime : 2
NewTaskList : AnalyzeRock$< 1, 40 >$; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList : ϕ
InformationGathered : ϕ
Utility of current schedule : 0.0; Duration of current schedule : 0.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 0.0
Meta-Level Control Decision : **Call Detailed Scheduler**

[4]The cost of computing complex features for the experiments described in this work is assumed to be low when compared to the cost of scheduling actions. This was done to test the effectiveness of these features on all the decision choices that succeed them. The cost of the computing complex features can be significantly higher than the cost of other control actions in certain domains. In those domains, it might be appropriate to reduce the number of options available after the information gathering action. For instance, if the cost of simple scheduling is 2 units and the cost of computing complex features is 4 units, it might be sensible to always execute the more expensive complex scheduling instead of simple scheduling, after computing complex features.

Here is another instance of the meta-level control decision process where *RoverB* is in state *S5* and it is time 16. A new task *Explore Terrain* arrives at time 15 with a deadline of 80. The new task is added to the NewTask List and *RoverB's* meta-level controller is invoked. *RoverB* is in the midst of executing method *Focus Spectrometer on Rock*, which has executed for 2 time units. The current schedule has gained 6.0 utility points and *RoverB* has gained a total of 6.0 utility points also. Based on its current state, *RoverB's* meta-level control decision is to *Delay Explore Terrain task* and to add it to the Agenda List instead. *RoverB* will continue execution of method *Focus Spectrometer on Rock*. When execution of this method is completed and if the NewTask List is empty, *RoverB* will automatically make meta-level control decision on all the tasks in the Agenda List.

State S5:
CurrentTime :16
NewTaskList : $ExploreTerrain < 15, 80 >$; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList : $\{FocusSpectrometeronRock^{exe}\}$
InformationGathered : ϕ
Utility of current schedule : 6.0; Duration of current schedule : 8.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;
Total Utility accrued : 6.0
MLC Decision : **Add New task to agenda**

Event Trigger 2: Presence of task requiring coordination in current task set: Suppose there is a subtask or method in the currently scheduled task set which either requires a non-local method to enable it or should be sub-contracted out to another agent. The local agent has to decide whether it is worthwhile to even initiate negotiation and if so, which negotiation protocol to use. The decision tree associated with this meta-level decision is described in Figure 5. This decision is made using the *MetaNeg* information described below.

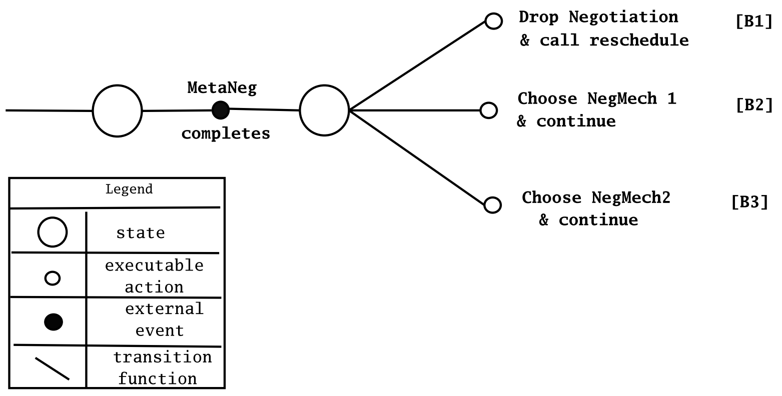


Figure 5: Decision tree on whether to negotiate and effort

Coordination actions are split into an external information gathering phase and a negotiating phase, with the outcome of the former enabling the latter. The negotiation phase can be achieved by choosing from a family of negotiation protocols [57]. The information gathering phase facilitates the negotiation phase and is modeled as a **MetaNeg** method in the task structure (see Figure 6) and the negotiation methods are modeled as individual primitive actions. Thus, reasoning about the costs of negotiation is done explicitly, just as it is done for regular domain-level activities.

The **MetaNeg** method belongs to a special class of domain actions which request an external agent for a certain set of information that does not require any significant use of local processor time. It queries the other agent and returns information such as expected utility of other agent's schedule, expected finish time of other agent's schedule, and amount of slack in the other agent's schedule. This information assists the meta-level controller in its decision making process.

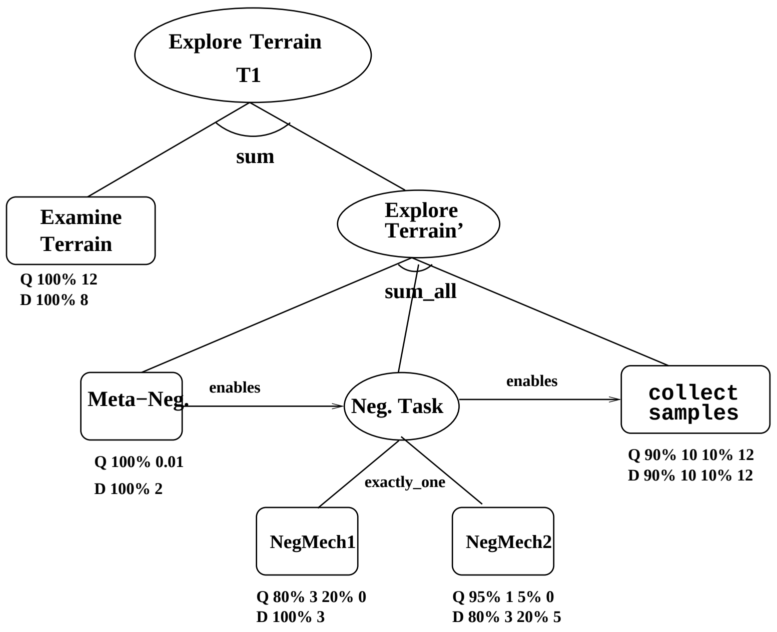


Figure 6: Task ExploreTerrain modified to include meta-negotiation action

The following is an instance where *RoverB* is in a state named *S11* at time 33. The Information Gathering Action (MetaNeg) has completed execution. The following information is returned by the information gathering action: Agent *RoverA* is executing high utility tasks, has deadlines which are far off and has a high amount of slack. Based on this information. *RoverB*'s meta-level controller is invoked and it uses the above information to decide that *RoverB* should negotiate with *RoverA* using the NegMech2 protocol about the finish time of *RoverA's* method *Arrive at Location.*

State S11:

CurrentTime :33
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{NegMech2, ExamineTerrain, CollectSamples\}$
InformationGathered : $< HIGH, HIGH, HIGH$
Utility of current schedule : 0.0; Duration of current schedule : 1.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;
Total Utility accrued : 18.0
MLC Decision : **Choose NegMech2 and continue**

Event Trigger 3: Negotiation process fails to reach a commitment: Suppose there is a subtask or method in the currently scheduled task set which has been negotiated about with a non-local agent and suppose the negotiation fails. The local agent should decide whether to renegotiate and if so, which protocol should it use. Figure 7 describes the associated decision tree.

Event Trigger 4: Domain action completes execution: When a primitive action is completed, the meta-level controller checks to see if the real-time performance of the current schedule is as expected. If the actual performance deviates from expected performance by more than the available slack time, then a reschedule

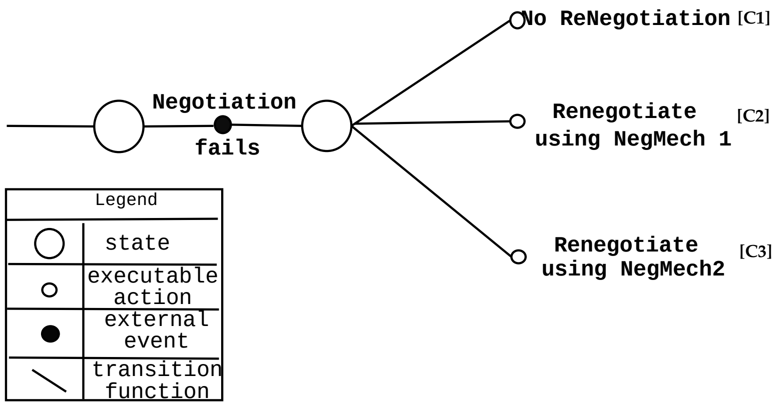


Figure 7: Decision tree on whether to renegotiate upon failure of previous negotiation

may be initiated. A decision to reschedule helps in two ways: it would preclude the agent from reaching a bad state in which too many resources are spent on a schedule with bad performance characteristics; and it would allow for meta-level activities to be processed without the detrimental effects such processing would have on domain activities if slack is minimal. Hansen's work [14] on meta-level control of anytime algorithms using a non-myopic stopping rule is described in Section 5. It finds an intermediate strategy between continuous monitoring and not monitoring at all. It can recognize whether or not monitoring is cost-effective, and when it is, it can adjust the frequency of monitoring to optimize utility. Thus, the decision to reschedule in this paper can be viewed as a non-myopic stopping rule within Hansen's work. The decision tree associated with this meta-level decision is described in Figure 8.

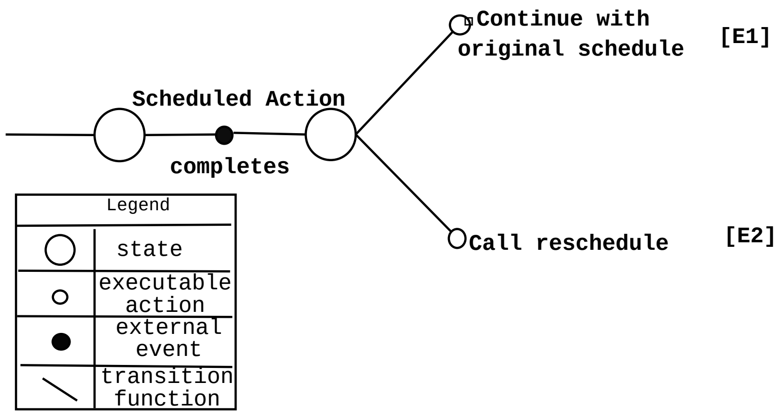


Figure 8: Decision tree when a domain action completes execution

Event Trigger 5: Invocation of the detailed scheduler: The parameters to the planner/scheduler are scheduling effort (E) and slack amount (S). They are determined based on the current state of the agent including characteristics of the existing schedule and the set of new tasks that are being scheduled. The *effort* parameter determines the amount of computational effort that should be invested by the planner/scheduler. The parameter can be set to either *HIGH*, where a high number of alternative plans/schedules are produced and examined or *LOW*, where pruning occurs at a very early stage and hence few alternative plans/schedules are compared, reducing the computational effort while compromising the optimality of the schedule. The effort is proportional to the expected utility and complexity (in terms of number of possible alternative plans) of the task. Although, the effort can be any discrete value, two qualitative values are used in the current implementation of the agent. These two values were sufficient to show the importance of varying the effort based on problem solving context. Depending on the problem domain, one could increase and decrease the number of feature values and the decision process will handle them appropriately.

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 amount of slack to be inserted depends on three factors, the amount of uncertainty in the schedule, the importance of the currently scheduled tasks and the expected amount of meta-level control activity that will occur during the duration of the schedule. The scheduler determines the amount of uncertainty in the schedules it builds and automatically inserts slack to handle highly uncertain primitive actions. The meta-level control component uses information about the arrival of future tasks to suggest slack amounts to the scheduler. Three slack values of 10%, 30% and 50% of the total available time are used in the current implementation of the agent. These values, like in the case of the effort, can be varied as needed. The decision tree describing the various action choices for this meta-level decision is shown in Figure 9. Each of the choices in the decision tree are combinations of possible effort and slack values.

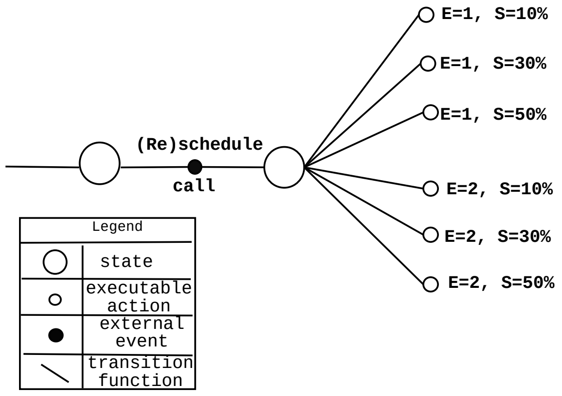


Figure 9: Decision tree for invoking the scheduler

In the introduction, we presented meta-level control as a sequential decision making process. In the next sub-section, we describe a formal model of the meta-level control problem which frames it as a sequential decision making process.

2.1 A Formal Model of Meta-level Control Decision Problem

1. Let *S* be the set of states of the agent and $s_i \in S$ is a particular state of the agent. Since this is a finite horizon problem, $i = 0, 1, 2, 3 \ldots, n$

2. *A* is the set of possible control actions and $a \in A$ is the action taken by the agent in state s_i.

   Control actions do not directly affect the utility achieved by the agent since they affect only the agent's internal state. These actions consume time and have only indirect effects on the external world.

   Control actions are followed by the execution of utility achieving domain actions. These domain actions are directly the result of control actions in the current and preceding states. These domain actions are not explicitly represented in this model since they are encased by the control actions.

3. A policy π is a description of the behavior of the system. A stationary meta-level control policy $\pi : S \rightarrow A$ specifies, for each state, a control action to be taken. The policy is defined for a specific

environment.

An environment is defined by three distributions describing arriving task type, task arrival rate and task deadline tightness.

4. $\pi(s_i, a)$ is the probability of an agent taking an action a in state s_i under policy π.

5. s_j is the new state reached after executing control action a followed by the execution of corresponding domain actions that follow a.

6. $R(s_i, a, s_j)$ is the reward obtained in state s_j as a consequence taking control action a in state s_i and then executing the domain actions that follow a.

   The reward is the cumulative value of the domain actions which are completed between the state transitions. Since the values achieved by the tasks have associated uncertainties, the reward function is represented as a distribution.

7. $U_\pi(s_i)$ is the utility of state s_i under policy π.

8. $P(s_j|s_i, a)$ is the probability that agent is in state s_j as a result of taking action a in state s_i.

The above model defines a finite Markov decision process [2].

According to decision theory, an optimal action is one which maximizes the agent's expected utility. For a finite-horizon problem, this is given by

$$E[U_\pi(s_i)] = E_\pi\{\sum_{j=1}^{n}\gamma^j \; R(s_i, a, s_j)\}$$

. $\gamma\epsilon[0, 1)$ is a discount-rate parameter which determines the present value of future utility gains.

This can be computed as follows

$$E[U_\pi(s_i)] = \sum_a \pi(s_i, a)\sum_{j=1}^{n} P(s_j|s_i, a))[R(s_i, a, s_j) + \gamma \; E[U_\pi(s_j)]]$$

The meta-level control problem for an individual agent is to find a best meta-level control policy π^* which maximizes the expected return for all states. This optimal policy can be determined by using dynamic programming [2] or reinforcement learning [48] methods. Reinforcement learning methods like Q-learning will implicitly determine the transition probability model and reward function defined previously. If the model were to be used as defined above for a sophisticated agent architecture, finding the optimal meta-level control policy would be computationally intractable because of the size of the state space. The quantitative values of the agent's state features contributes to the explosion of the state space. We feel that independent of the approach used to formulate meta-level control, the ability to appropriately abstract the agent state is key to the development of an effective meta-level control component.

In this paper, we describe a computationally feasible approach to handle the above described complexity. The following are the salient features of our approach:

1. We identify state features that are effective approximations of the system state. This helps bound the size of the state set making the problem tractable.

2. We test the effectiveness of this approximate representation of system state using two classes of hand-generated heuristics for meta-level control. These heuristics use the approximate system state in their decision making. We show that these strategies that leverage meta-level control perform better than strategies without any meta-level control.

3. We then use these features to define the state of a meta-level Markov Decision Process for various environments and use reinforcement learning techniques to estimate the probability transition and reward model. The resulting MDP is evaluated and a non-myopic meta-level control policy for agents operating in complex environments is determined.

2.2 Agent State

The meta-level controller uses the current state of the agent to make appropriate decisions. In Section 2.1, we discussed that the use of quantitative feature values would make computing the optimal meta-level control policy intractable. Consequently in this work, a distinction is made between the current state of the agent (also called real state) and the approximate representation of the state and the use of qualitative values which capture only the critical information about the current state.

The real state of the agent has also the detailed information related to the agent's decision making and execution. It accounts for every task which has to be reasoned about by the agent, the execution characteristics of each of these tasks, and information about the environment such as types of tasks arriving at the agent, frequency of arrival of tasks and the deadline tightness of each of these tasks. The real state is continuous and complex. This leads to a combinatorial explosion in the real state space even for simple scenarios. The complexity of the real state is addressed by defining an abstract representation of the state which captures the important qualitative state information relevant to the meta-level control decision making process. There are eleven 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 $4^{11} = 2^{22}$, which is about a million states. Framing this problem in the MDP framework would result in a search space of million states out of which only states in the order of a few thousand are actually encountered because the feature values are not independent of each other and act as constraints on each other and the specifics of the environmental dynamics. For instance when the utility goodness of the current schedule is HIGH and the deadline tightness of the current schedule is TIGHT, the amount of slack in local schedule is usually LOW. It never takes on the value HIGH.

The following are some characteristics of the system state features, the environment and its dynamics.

1. The status of tasks currently being processed and those which need future processing, e.g. NewTaskList, AgendaList, ScheduleList, ExecutionList.

2. Environmental model, e.g. Probability of arrival of specific types of tasks and their deadline tightness.

3. Internal influences on action choice, e.g. Slack in the schedules.

4. External influences on action choice, e.g. Utility of tasks of other agents, Deadline Tightness of tasks belonging to other agents, Slack in schedules of other agents

5. Real time performance characteristics, e.g. Deviation from expected performance, Cost of decommitting from existing tasks

These "real" features are used to construct the abstract state representation that will permit effective meta-level control for the domain described in this paper.

FeatureID	Feature	Complexity
F1	Utility goodness of new task	Simple
F2	Deadline tightness of a new task	Simple
F3	Utility goodness of current schedule	Simple
F4	Deadline tightness of current schedule	Simple
F5	Arrival of a valuable new task	Simple
F6	Amount of slack in local schedule	Simple
F7	Amount of slack in other agent's schedule	Simple
F8	Deviation from expected performance	Simple
F9	Decommitment Cost for a task	Complex
F10	Relation of slack fragments in local schedule to new task	Complex
F11	Relation of slack fragments in non-local agent to new task	Complex

Table 1: Table of proposed state features, their description and category

2.3 Abstract Representation of the State

The overhead of meta-level control activities is accounted for by the cost of state feature computation. The eleven features, which are of two categories - simple features where the reference values are readily available by simple lookups and complex features which involve computation to determine their values. Simple features help the agent make informed decisions on executable actions or whether to obtain more complex features to make the decisions. An example of a simple feature would be the availability of slack in the current schedule. If there is a lot of slack or too little slack, the decision to accept the new task or drop the new task respectively is made. However, if there is a moderate amount of slack, the agent might choose to obtain a more complex feature, namely computing the relation of slack fragments which is described below.

Complex features usually involve computations that take time that is sufficiently long that, if not accounted for, will lead to incorrect meta-level decisions. The computation of the complex features is cumbersome since they involve determining detailed timing, placement and priority [5] characteristics and provide the meta-level controller with information to make more accurate action choices. For instance, instead of having a feature which gives a general description of the slack distribution in the current schedule i.e. there is a lot of slack in the beginning or end of the schedule, there is a feature which examines the exact characteristics of the new task and makes a determination whether the available slack distribution will likely allow for a new task to be included in the schedule. The agents make explicit meta-level control decisions based on whether to gather complex features and determine which complex features are appropriate.

In Section 2.1, a formal definition of the meta-level control problem was presented. The abstract representation of the state defined in this section will be the states in the Markov Decision process model. The control actions defined in section 2 will be the actions in the MDP model. The probability transition function and the reward function will be determined by estimating them from data gathered in previous system runs.

Table 1 enumerates the features of the abstract representation of the state used by the meta-level controller.

F1: Utility goodness of new task: It is a simple feature which 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

[5] Priority accounts for quality and deadline.

other tasks being performed by the agent. The assigned feature values are HIGH, MEDIUM and LOW respectively.

F2: Deadline tightness of a new task: It is a simple feature which describes the tightness of the deadline of a particular task in relation to expected deadlines of other tasks. It determines whether the new task's deadline is very close, moderately close or far in the future. The assigned feature values are TIGHT, MEDIUM, LOOSE respectively.

F3: Utility goodness of current schedule: It is a simple feature describes the utility of the current schedule normalized by the schedule length and is based on information provided by the scheduler. This feature determines whether the current schedule is very valuable, moderately valuable or not valuable with respect to other tasks and schedules. The assigned feature values are HIGH, MEDIUM and LOW respectively.

F4: Deadline tightness of current schedule: It is a simple feature which 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. It determines whether the schedule's deadline is very close, moderately close or far in the future. The assigned feature values are TIGHT, MEDIUM, LOOSE respectively.

F5: Arrival of a valuable new task: It is a simple feature which 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. It can take on the values of HIGH, MEDIUM, LOW.

F6: Amount of slack in local schedule: It is a simple feature which 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. This feature can take on the values of HIGH, MEDIUM, LOW.

F7: Amount of slack in other agent's schedule: This is a simple feature used to make a quick evaluation of the flexibility in the other agent's schedule. This is used when an agent is considering coordinating with the other agent to complete a task. This feature can take on the values of HIGH, MEDIUM, LOW.

F8: Deviation from expected performance: This is a simple feature which 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. The feature can take on the values of HIGH, MEDIUM, LOW.

F9: Decommitment Cost for a task: This is a complex feature which estimates the cost of decommiting from doing a method/task by considering the local and non-local down-stream effects of such a decommit. This feature can take on the values of HIGH, MEDIUM, LOW.

F10: Relation of slack fragments in local schedule to new task: This is a complex feature which determines the feasibility of fitting a new task given the detailed fragmentation of slack in a particular schedule. It involves resolving detailed timing and placement issues. This feature can take on the values of HIGH, MEDIUM, LOW.

F11: Relation of slack fragments in non-local agent to new task: This is a complex feature which determines the feasibility of fitting a new task given the detailed fragmentation of slack in a particular non-local schedule. This feature can take on the values of HIGH, MEDIUM, LOW.

Each of the state features takes on qualitative values. The quantitative values such as utility of 80 versus utility of 60 are classified into these qualitative buckets (high versus medium utility) in a principled way as shown later in this section. As will be seen in the experimental results in later sections, these qualitative measures provide information that can be exploited to make effective meta-level control decisions.

2.4 Computation of State Features

As described above, our goal is to bound the size of the state space. This is done by making the high level state features time independent and also by eliminating the specifics of tasks and their performance characteristics from the state. We adopt a qualitative representation for the state feature values and consider the characteristics of task sets instead of individual tasks.

The following describes the mechanism that exploits knowledge about the agent tasks and environmental characteristics to determine the high-level features of the agent state.

- *The 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_Ψ is the total number of tasks that have arrived at the system from the start to current time Ψ.*

- *A task t ϵ 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) $t^1, t^2, t^3...t^k$.*

- *A plan t^j to achieve task t is an executable sequence of 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: $ExamineTerrain^A$ and $ExamineTerrain^B$ are two alternate plans to achieve task $ExamineTerrain$.
  $(25\%$ 22 50% 50 25% $100)$ is the duration distribution of $ExamineTerrain^A$, which means that plan $ExamineTerrain^A$ takes 22 units of time 25% of the time, 50 time units 50% of the time and 100 time units 25% of the time. Also $ExamineTerrain^A$ has a utility distribution of $(10\%$ 30 90% $45)$. $ExamineTerrain^B$ has a duration distribution $(50\%$ 32 30% 40 20% $45)$ and utility distribution of $(25\%$ 20 75% $30)$.

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

- Ψ_t *is the time required for scheduling a task t if it is chosen for scheduling.*

  Example : Ψ_t is 2 units[6] if simple scheduling is chosen. If detailed scheduling is chosen, the cost is 4 units if the scheduling set has less than 5 primitive actions to evaluate, 12 units if the scheduling set has between 5 and 10 primitive actions to evaluate and 18 units if the scheduling set has more than 10 primitive actions.

  System execution is single threaded allowing for one primitive action at the most to be in execution at any time. If a meta-level action is required when a primitive action m is executing, the execution is interrupted and control is turned over to the meta-level controller. When the meta-level control action is completed, execution of m is always resumed[7].

[6]The number of time units is the statistical average of the durations obtained from offline simulations of executing the task using the particular deliberative mode. It is described in terms of simulation ticks.

[7]This is an artifact of the simulation environment. In order for the simulator to keep accurate records of utility accumulated, it requires that executing actions should be fully completed.

- *Υ_m is the remaining time required for primitive action m to complete execution.*
- *The earliest start time EST_t for a task t is the arrival time AT_t of the task delayed by the sum of Υ_m, the time required for completing the execution of the action m which is interrupted by a meta-level control event and Ψ_t, the time required for scheduling the new task.*

$$EST_t = AT_t + \Upsilon_m + \Psi_t$$

- *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: Suppose $ExamineTerrain$ arrives at time 45 and has a deadline of 100. Also suppose the execution of method m is interrupted by the arrival of $ExamineTerrain$ and m still needs about 8 time units to complete execution. Suppose the time spent on scheduling $ExamineTerrain$ is 5 units. Then the maximum available duration for task $ExamineTerrain$ is $100 - 45 - 8 - 5 = 42$ time units. The meta-level controller is aware that the entire range of the maximum available duration is not always available solely for the execution of this task. When the maximum available duration ranges of a number of tasks overlap, the maximum duration available for a particular task is effectively reduced.

- *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 task's maximum available duration.*

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

  Example: Suppose the maximum available duration for task $ExamineTerrain$ is 42. There is only one duration value in $DD_{ExamineTerrain^A}$ which has a value less than 42 and that value is 22 and occurs 25% of the time.

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

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

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

- *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.*

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

  Example: For the above constraint where the maximum available duration for task $ExamineTerrain$ is 42

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

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

- *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.*

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

  Example: When the maximum available duration for task $ExamineTerrain$ is 42,

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

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

- *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 to 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: Suppose the maximum available duration for task $ExamineTerrain$ is 42. Consider each of $ExamineTerrain$'s alternative plans which were described earlier. Plan $ExamineTerrain^A$'s expected utility to expected duration ratio is $\frac{10.875}{22} = 0.494$ and plan $ExamineTerrain^B$'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[8] is $ExamineTerrain^B$.

  $$ALT_{ExamineTerrain} = ExamineTerrain^B$$

  .

- *The utility goodness UD_t of a task t (feature F1 in Table 1) is the measure which determines how good a task's preferred alternative is in relation to 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 task's preferred alternative.

  $$UD_t = \begin{cases} HIGH, \; \frac{EU_{ALT_t}}{ED_{ALT_t}} \; is \; above \; the \; 66th \; percentile \\ MEDIUM, \; \frac{EU_{ALT_t}}{ED_{ALT_t}} \; is \; between \; the \; 66th \; and \; 33rd \; percentile \\ LOW, otherwise \end{cases}$$

  Example: The utility goodness of task $ExamineTerrain$ given a deadline of 100 and a maximum available duration of 42 is $\frac{22}{35} = 0.628$ which lies above the 66th percentile. $UD_{ExamineTerrain} = HIGH$

[8]The assumption here is that there may be other tasks that can use the available time. If we add a model of opportunity cost, this definition can be modified. This is an area of future work.

- *The deadline tightness TD_t of a task t (feature F2 in Table 1) measures the flexibility of the maximum available duration. It determines how much unexpected meta-level activities and similar delays affect the maximum available duration.* Suppose a meta-level activity on average has an expected duration of C_{ML}. The *expected amount of time required for handling unexpected meta-level activities* Ω_t, during the execution of task t, is computed as follows:

  $$\Omega_t = C_{ML} * \frac{N_\Psi}{\Psi} * MD_t$$

  Example: Suppose the average time per meta-level activity is 2 units, 4 tasks have arrived at the agent and the current time is 60. MD_t is 42 as determined previously. $\Omega_{ExamineTerrain} = 2*\frac{4}{60}*42 = 5.6$ The amount of time expected to be spent on future meta-level activities is 5.6 units.

  In order to determine if a given deadline is tight, the *proposed maximum available duration*, MD_t^X for a proposed scenario X is computed. It is the maximum available duration which also accounts for the anticipated meta-level costs of future activities.

  $$MD_t^X = MD_t - \Omega_t$$

  Example: From the previous example, the $MD^X_{ExamineTerrain} = 42 - 5.6 = 36.4$

  The related parameters $PDL^X_{ALT_t}$, $ED^X_{ALT_t}$, $EU^X_{ALT_t}$ and the expected utility to expected duration ratio $\frac{EU^X_{ALT_t}}{ED^X_{ALT_t}}$ for the proposed scenario are also recomputed with respect to the redefined MD_t^X.

  $$PDL^X_{ExamineTerrain^B} = 0.5, \quad ED^X_{ExamineTerrain^B} = 32$$

  $$EU^X_{ExamineTerrain^B} = 11.25, \quad UD^X_{ExamineTerrain} = \frac{EU^X_{ExamineTerrain^B}}{ED^X_{ExamineTerrain^B}} = 0.3451$$

  The expected utility to expected duration ratio now falls below the 33rd percentile,

  $$UD^X_{ExamineTerrain} = LOW$$

  $$TD_t = \begin{cases} TIGHT, (UD_t = HIGH) \wedge (UD_t^X \neq HIGH \\ LOOSE, (UD_t = HIGH) \wedge (UD_t^X = HIGH) \\ MEDIUM, \forall \; other \; values \; of \; UD_t, \; UD_t^X \end{cases}$$

  Example: Since $(UD_{ExamineTerrain} = HIGH) \wedge (UD^X_{ExamineTerrain} = LOW)$, the time spent on unexpected meta-level control activities is detrimental to task t's utility gain, which in turn means its deadline is tight.

  $$TD_{ExamineTerrain} = TIGHT$$

- *The high priority task set for an agent α $HPTS_\alpha$ is the set of tasks whose utility goodness is HIGH and deadline tightness is TIGHT.*

  $$HPTS_\alpha = \{T_k\} : (UG_k = HIGH) \wedge (TD_k = TIGHT)$$

  Example:

  $$HPTS_A = \{ExamineTerrain\}$$

- *The arrival rate of high priority tasks for an agent α, ART_α (feature F5 in Table 1), is the ratio of the number of high priority tasks that arrive at the system to the total number of tasks n that have arrived at the system.*
$$ART_\alpha = \frac{|T_k|}{n} : T_k \, \epsilon \, HPTS_\alpha$$
- *The probability of a high priority task arriving in the near future PHT_α depends on the arrival rate of high priority tasks.* The intuition behind this relation is that the characteristics of tasks that arrived in the past can be used to predict the characteristics of tasks that will arrive in the near future. The assumption made by the system that the past information can be used to predict the future is a valid assumption since the environment is stationary for a finite-horizon.

  For instance, if ART_α is less than 0.04 (arrival rate is less than 4%), then PHT_α is also low.
$$PHT_\alpha = \begin{cases} LOW, & ART_\alpha < 0.04 \\ MEDIUM, & 0.04 <= ART_\alpha < 0.10 \\ HIGH, & ART_\alpha >= 0.10 \end{cases}$$
- *The slack in the schedule $SLACK_{scur}$ (used to compute features F6 and F7 in Table 1), is the total amount of flexibility that should be inserted in the schedule so that unexpected meta-level activities and uncertainty in method execution durations of all the tasks being scheduled can be accommodated without expensive rescheduling control actions.* The slack is defined using a simple slack distribution strategy, where the duration of each method in the schedule is extended by equal fractions of the total slack.
$$SLACK_{scur} = \sum_{\forall t \epsilon scur} \Omega_t$$

The following sections will test the hypothesis that using this state information, the best sequence of control and domain actions can be determined for each environment. The action sequence can either be determined by a heuristic hand-generated rules as described in the next section or can be learned automatically as described in Section 4.

3 Heuristic Strategies

We have discussed the reasons for the intractability of computing a meta-level control policy using the real system state in Section 2.1. We also defined an approximate representation of the state that will control the complexity of the meta-level control problem. In this section we validate the effectiveness of the approximate state representation using hand-generated heuristics for meta-level control. These hand-generated heuristics will then be used as a baseline for evaluating the performance of a reinforcement learning approach that we propose to use to learn the meta-level control policies for different environments.

We address the three following questions: Does meta-level control lead to better performance in rational agents situated in the domain described in this paper? Is it possible to construct a hand-generated meta-level control policy based on the high-level state features described earlier for specific environments. Does this hand-generated policy outperform a deterministic meta-level control policy?

Two heuristic strategies, the Naive Heuristic Strategy and the Sophisticated Heuristic Strategy, that use context sensitive rules for meta-level control are described. Both strategies use high-level state features and they serve as a test-bed for the effectiveness of the state features for efficient meta-level control. The Naive

Heuristic Strategy (NHS) uses state-dependent hand-generated heuristics to determine the best course of meta-level control action. The current state information will allow the meta-level controller to dynamically adjust its decisions. The heuristics, however, are myopic and do not reason explicitly about the arrival of tasks in the near future. The Sophisticated Heuristic Strategy (SHS), on the other hand, is a set of hand-generated rules that use knowledge about task arrival models to predict the environment characteristics. The agent's environment is typically characterized by the expected utilities of the tasks, their deadline tightness and frequency of arrival. In this work, the information on the three parameters is available to the SHS. Though not implemented in the context of this paper, this information can be learned by the SHS by gathering statistics over multiple runs. The meta-level controller can make non-myopic decisions by including information about its environment in its reasoning process.

In the interests of space, we describe a simple example to differentiate the decision making process between the two strategies and provide details of the SHS heuristics for the *Arrival of New Task* event trigger. We refer the reader to [33] where details of the NHS heuristic rules as well the SHS heuristics for all five event triggers are described in detail. Table 2 and Table 3 describe SHS rules required to support each of the actions on the new task in Figure 4 for the *Arrival of new task* event trigger. For instance the first row in the Table 2 describes the following rule: If new task has LOW utility goodness and TIGHT deadline; HIGH probability of high priority tasks arriving in the near future, then best action is *Drop Task (A1)*. When a feature is not specifically addressed in a rule, it is assumed that the feature can take on any of its domain values. So in the above example the Current Schedule Utility Goodness (CSUG) state feature can be HIGH, MEDIUM or LOW and the rule would still hold true. We now describe an example that the decision making process using NHS and SHS rules respectively. Consider the scenario where a new task arrives at the agent at time t. The agent has to decide whether to delay the reasoning about the task until later; never execute the task (drop task); execute the task immediately at arrival time by calling for a reschedule action or add the new task to the agenda. Now suppose an agent has to make a meta-level decision in the following context: the utility of the new task is MEDIUM; the deadline tightness of the new task is TIGHT; utility of current task set is MEDIUM; and the deadline of the existing task is MEDIUM; and the probability of a high priority task (HIGH utility, TIGHT deadline) arriving in the near future is HIGH. An agent equipped with NHS rules would make the myopic decision to *Call the Detailed Scheduler on All Lists (A4)* as it does not have access to the information about the probability of arrival of a high priority task in the near future. This decision implies that the agent is willing to incur the cost of interrupting the current schedule and reschedules the current task and new task expecting that this would produce a higher quality schedule. A second agent equipped with SHS rules has access to the knowledge that there is a high probability of a high priority task. The SHS rule in the third row of Table 2 prescribes that the meta-level control decision is *Drop Task*. This implies that the new task will be dropped and the agent will continue execution of the current schedule until the schedule completes execution or another meta-level decision is triggered. Given that the high probability of arrival of a high priority task arriving in the near future will trigger a reschedule using both NHS and SHS rules, the SHS decision would be the correct non-myopic choice in most environments as it would avoid the cost of the first reschedule prescribed by the NHS rules.

3.1 Single-agent Experiments

This sub-section provides performance comparisons of four different strategies to single-agent meta-level control: Naive Heuristic Strategy (NHS); Sophisticated Heuristic Strategy (SHS); Deterministic Strategy; and Random Strategy in different types of problem environments. Specifically, we empirically validate the advantage of using the heuristic strategies that dynamically adjust to context. We also show that knowledge

ID	NTUG	NTDL	CSUG	CSDL	P	MLC Decision
1	L	T	*	*	H	Drop Task (A1)
2	L	*	H	T	*	Drop Task (A1)
3	H/M	M/T	H/M	M/T	H	Drop Task (A1)
4	H	T	L	*	*	Simple Scheduler (A2)
5	L	*	-	-	L	Simple Scheduler (A2)
6	L	*	*	*	*	Simple Scheduler (A2)
7	H	M	*	*	M/L	Detailed Scheduler (A3)
8	H	T	L	T	L	Detailed Scheduler (A3)
9	H	T	L	T	L	Detailed Scheduler on All Lists (A4)
10	H	LS	H	M/LS	L/M	Add New Task to Agenda (A5)
11	M/L	M/LS	H	*	L	Add New Task to Agenda (A5)

Table 2: SHS rules for *Arrival of New Task* event trigger (Actions A1-A5). The column headers are ID = Heuristic Rule Number; NTUG = New Task Utility Goodness state feature; NTDL = New Task Deadline state feature; CSUG = Current Schedule Utility Goodness state feature; CSDL = Current Schedule Deadline state feature; P = Probability of Arrival of High Priority Tasks in the near future state feature; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

about the type of problem environment, specifically knowledge about the future, is advantageous.

We define a deterministic strategy as one that uses a fixed choice of meta-level action. When a new task arrives, this strategy always chooses to perform complex scheduling on the new task along with the tasks in the current schedule and tasks in the agenda. The scheduler is invoked with a fixed effort level of high and fixed slack amount of 10% of the total schedule duration. The deterministic strategy also does not automatically reschedule upon execution failure. The random strategy randomly chooses its actions for each of the three single-agent meta-level control decisions: when to accept, delay or reject a new task, how much effort to put into scheduling when reasoning about a new task and whether to reschedule when actual execution performance deviates from expected performance. The following costs are assumed for the experiments. 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. These actions are designed to be very quick, yet they are actions with costs. Hence they are assigned a duration of 1 simulation tick each. The call to the simple scheduler costs 2 time units[9] 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 [10]. The duration of the control action is proportional to the effort level. Actions requiring higher effort levels will require longer durations.

The agents in the experimental test-bed were implemented using the Java Agent Framework (JAF) framework [53] and situated in the Multi-Agent Survivability Simulator (MASS) environment [18]. In order to randomly generate different types of problem environments, we implemented a task environment generator that varied the following environment parameters.

[9]As defined earlier, the number of time units is the statistical average of the durations obtained from offline simulations of executing the task using the particular deliberative mode.

[10]The non-linear increase in processing time is due to the design of the heuristic complex scheduler. The Design-to-Criteria scheduler is shown to work well for tasks with fewer than 10 primitive actions[54].

ID	NTUG	NTDL	CSUG	CSDL	CSSL	AGUG	AGDL	DC	P	MLC Decision
12	*	*	M/H	LS/M	*	M/H	LS/M	*	L	Get More Features (A6)
13	*	*	M	M	L	M	M	L/M	*	Drop Task (A7)
14	*	*	M/H	M	*	M/H	M	*	*	Simple Scheduler (A8)
15	M/H	M	M/H	M	H	*	*	*	*	Detailed Scheduler (A9)
16	H	*	H	M	H	H	*	M	*	All Lists Detailed Scheduler (A10)
17	H	LS	H	M	M	*	*	*	*	Add New Task to Agenda (A11)

Table 3: SHS rules for *Arrival of New Task* event trigger (Actions A6-A11). The column headers are ID = Heuristic Rule Number; NTUG = New Task Utility Goodness; NTDL = New Task Deadline; CSUG = Current Schedule Utility Goodness; CSDL = Current Schedule Deadline; CSSL = Slack in Current Schedule; AGUG = Utility Goodness of tasks in Agenda; UGDL = Deadline Tightness of tasks in Agenda; DC = Decommitment Cost; P = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

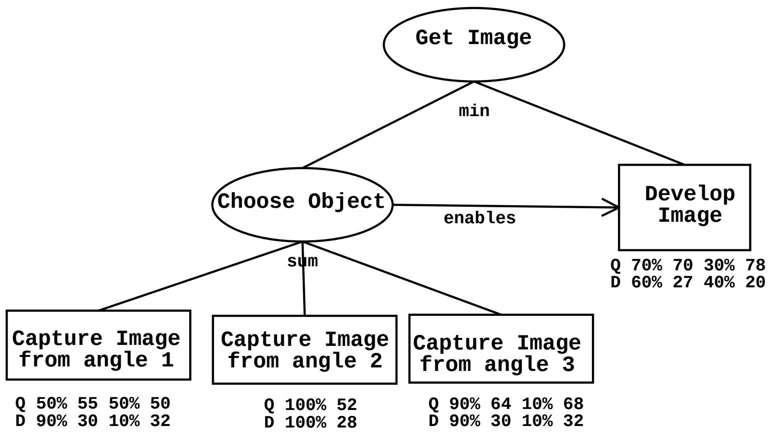


Figure 10: A complex task in a single agent environment

1. complexity of tasks $c \,\epsilon\, \{simple(S), complex(C), combination(A)\}$
2. frequency of arrival $f \,\epsilon\, \{high(H), medium(M), low(L)\}$
3. tightness of deadline $dl \,\epsilon\, \{tight(T), medium(M), loose(L)\}$.

Complexity of tasks refers to the expected utilities of tasks and the number of alternative plans available to complete the task. Typically, complex tasks have higher expected utility, higher expected durations and a greater number of alternatives than simple tasks. A simple task has two primitive actions and its structure and number of possible alternatives is similar to the AnalyzeRock task (Figure 3) described in Section 2. The utility distribution and duration distribution of a simple task is within a 5% range of the corresponding distributions of AnalyzeRock. A complex task also has structure similar to that of GetImage task described in Figure 10. It has between four and six primitive actions. The utility distribution and duration distribution of a complex task is within a 5% range of the corresponding distributions of GetImage. The combination value means that 50% of the tasks are simple and 50% are complex tasks.

The frequency of arrival of tasks refers to the number of tasks that arrive within a finite time horizon. The

Row#		SHS	NHS	Deter.	Rand.
1	AUG	205.49	192.10	121.90	89.97
2	σ	7.0	12.5	12.55	19.114
3	CT	20.37%	23.92%	39.27%	11.77%
4	RES	0%	14.53%	0%	50.56%
5	PTC	41.08%	39.64%	30.52%	21.56%
6	PTDEL	43.78%	49.0%	0%	11.49%

Table 4: Performance evaluation of four algorithms over a single environment AMM with a combination of tasks, medium frequency of arrival and medium deadline tightness. 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 control actions(CT); row 4 is percent of tasks rescheduled (RES); Row 5 is the percent of total tasks completed (PTC); Row 6 is percent of tasks delayed on arrival (PTDEL)

resource contention among the tasks increases as the task frequency increases. Task arrival is determined by a normal distribution with $\mu = 250$ and $\sigma^2 = 35$. When the frequency of arrival is low, about one to ten tasks arrive at the agent in 500 time unit horizon; when the frequency is medium, between ten and fifteen tasks arrive at the agent; and when the arrival frequency is high, fifteen to twenty arrive on average at the agent. 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; if the deadline tightness is set to medium, the maximum available duration given to the tasks is between 100% and 120% of the expected duration of the task; and if the deadline tightness is set to high, the maximum available duration is between 80% and 100% of the expected duration of the task. Environments are named based on values of these three criteria in the order mentioned above. For instance, environment AMM is one that has a combination of simple and complex tasks (A), with medium frequency of arrival (M) and medium deadline tightness (M).

The experimental results described in Table 4 show the performance of the various strategies in the environment, AMM, which, as mentioned before, contains a combination of simple and complex tasks. The frequency of task arrival in this environment is medium and ranges between 10 and 15 tasks in the 500 time unit interval. The deadline tightness is also medium. Simple tasks have a minimum duration of 8 time units and a maximum duration of 20 time units while complex tasks take a minimum duration of 50 time units and a maximum duration of 120 time units. Also the utility gained by completing a single task can range between 6 and 24 while the utility gained from a complex task is between 70 and 80. Each strategy was evaluated over 300 runs and each run has an associated task arrival model, lasts 500 time units and has an average of 15 meta-level control decision points per run.

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 of the table describe the average utility gained (AUG) by each of the strategies and the corresponding standard deviations. The heuristic strategies (SHS and NHS) significantly ($p < 0.05$) outperform the deterministic and random strategies with respect to utility gain. This shows that heuristic strategies that dynamically adjust their decisions to environmental state perform significantly better than strategies that do not take state information into account in their decision making process.

SHS has about a 10% improvement in utility gain than NHS. Detailed analysis of the data shows that

NHS assigns incorrect amounts of slack in the schedule which is required to handle unexpected meta-level activities. This leads to frequent reschedule calls and an increase in time spent on control actions. The SHS is able to allocate accurate amounts of slack because it has access to the task arrival model information and is able to avoid unnecessary control actions (particularly reschedules). This shows that knowledge about the future allows an agent to make better meta-level decisions on time allocations.

Row 3 shows the percent of the 500 time units for each run that was spent on control actions (CT) and row 4 shows the percent of tasks that were rescheduled (RES) per run in the midst of their execution. For the above mentioned reason, NHS has a significant number of reschedules resulting in time being spent on control actions instead of being spent the utility deriving domain actions. Row 3 shows that the duration spent on control actions by NHS is significantly ($p < 0.05$) higher than that of SHS. The deterministic strategy does not automatically reschedule but invests a lot of time on control actions since the fixed strategy is time-intensive. The random strategy spends the least amount of time on control (11.77%) because it attempts relatively few tasks (there is a high probability of a task being dropped randomly upon arrival).

Row 5 is the percent of total tasks completed (PTC). This was found to be less than 50% for this environment. This is because this environment is fairly dynamic (in terms of frequency of occurrence of exogenous events) and has tight constraints (the deadlines of task are of medium tightness) that limit the number of tasks that can be successfully completed

Row 6 is percent of tasks delayed on arrival (PTDEL). Here too about 45% of the tasks are delayed in case of the heuristic strategies signifying there is significant overlap among the tasks in terms of resource usage. In other words, new tasks often arrive at the agent when the agent is busy with other tasks.

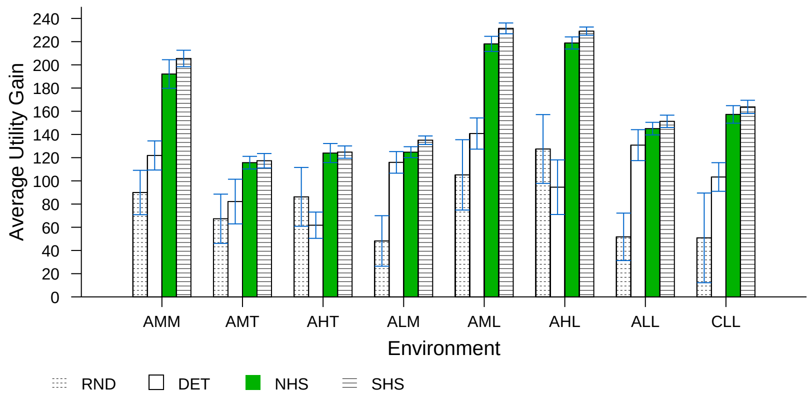


Figure 11: Average utility comparison between heuristic strategies and baseline strategies over 8 different environments. The error bars are one standard deviation above and below each mean

We ran experiments for all 27 environments that were generated by the enivronment generator. The results showed that meta-level control was advantageous in eight of these environments. Figure 11 shows the utility comparisons over these eight environments. The heuristic strategies (SHS and NHS), as in the case of environment (AMM) described previously, significantly outperform ($p<0.05$) the baseline strategies (Deterministic and Random) over all eight types of environments. In the discussion of the results, we emphasize the characteristics of the environments that justify the cost of explicit meta-level control.

Table 5 provides the detailed information on the performance comparison. Columns 2-5 show the average utility gained by each of the four algorithms for that environment. Column 6 named p1 shows the statistical significance (p-value) of SHS with respect to NHS. Column 7 named p2 shows the statistical

Environment	SHS	NHS	Deter.	Rand.	p1	p2	p3
AMM	205.49	192.10	121.90	89.97	0.032	0.0001	0.0001
AMT	117.34	115.69	82.17	67.33	0.4391	0.0001	0.0001
AHT	124.80	123.96	61.77	86.20	0.6906	0.0001	0.0001
ALM	135.05	124.74	115.93	48.21	0.004	0.0001	0.0001
AML	231.44	218.07	140.80	105.16	0.0045	0.0001	0.0001
AHL	229.07	218.86	94.55	127.47	0.0024	0.0001	0.0001
ALL	151.31	145.03	130.80	51.76	0.2596	0.0001	0.0001
CLL	163.77	157.27	103.33	50.86	0.0643	0.0001	0.0001

Table 5: Utility comparisons over a number of environments. Columns 2-5 show the average utility gained by each of the four algorithms for that environment. Column 6 named p1 shows the statistical significance (p-value) of SHS with respect to NHS. Column 7 named p2 shows the statistical significance of SHS with respect to the deterministic algorithm. Column 8 named p3 shows the statistical significance of NHS with respect to the deterministic algorithm.

significance of SHS with respect to the deterministic algorithm. Column 8 named p3 shows the statistical significance of NHS with respect to the deterministic algorithm. It can be observed from the table that the SHS strategy is significantly better than the NHS ($p<0.05$) in some environments (ALM, AML, AHL). All three environments can be characterized as medium constrained environments. In environment ALM, the arrival frequency is loosely constrained while the deadline tightness is MEDIUM. On detailed analysis of the data, it was found that there were extended periods in which no tasks arrived at the agent and then there would be burst of task arrivals. So information on the nature of future tasks allowed the agent to make better decisions during those periods of resource contentions (caused by the medium deadlines). In the other two environments, the deadline tightness was LOOSE while the arrival frequency was either MEDIUM and HIGH. Since the tasks have loose deadlines, they can be processed whenever resources are available without detrimentally affecting the utility. However since the arrival frequency is medium to tightly constrained, there is a very high probability of overlapping tasks contending for resources. The arrival model information will allow the agent to dynamically adjust its decisions on tasks and use the bounded resources in an efficient way. It can be deduced that the arrival model information available to the SHS is advantageous only in environments that are neither tightly constrained or loosely constrained.

The reason for the improved performance by the heuristic strategies when compared to the deterministic and random strategies is found in Figure 12 which shows the percent of control time comparisons over the same set of environments. As described in 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 heuristic strategies use control activities that optimize their use of available resources (time in this case). The deterministic strategy on the other hand always makes the same control choice, the expensive call to the detailed scheduler, independent of context. Hence the deterministic strategy has higher control costs, than the heuristic strategies and has less resources (time) to execute domain actions and accrue utility. The random strategy has low control costs but it doesn't reason about its choices leading to bad overall

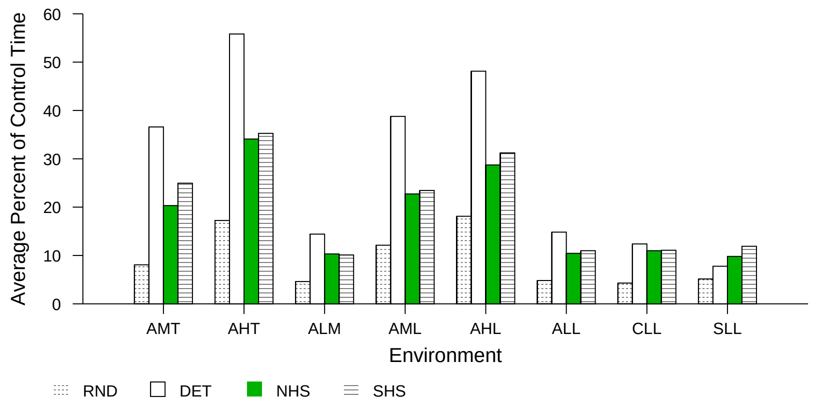


Figure 12: Average percent control time comparison between heuristic strategies and baseline strategies over 8 different environments

Environment#	SHS	NHS	Deter.	Rand.
AMM	20.37%	23.92%	39.27%	11.77%
AMT	24.95%	20.32%	36.59%	8.07%
AHT	35.26%	34.09%	55.82%	17.24%
ALM	10.11%	10.32%	14.42%	4.61%
AML	23.45%	22.73%	38.77%	12.12%
AHL	31.23%	28.73%	48.12%	18.11%
ALL	10.99%	10.44%	14.83%	4.82%
CLL	11.08%	10.99%	12.39%	4.29%

Table 6: Control time comparisons over a number of environments; Column 1 is the environment type; Columns 2-5 represents the % of total time spent on control actions by each of the four algorithms for that environment;

performance. Table 6 provides the details about the percent of total available time per episode (500 units) that was spent on control actions.

Table 7 compares the percentage of tasks that were successfully completed by the four algorithms in different environments. In tightly constrained environments like those with tight task deadlines (AMT, AHT), the number of tasks completed is relatively low because often there aren't enough resources to execute the task and process all the external events also. In loosely constrained environments like ALM, CLL and ALL, task arrival is few and far between allowing the agent to complete one task successfully and to move on to the next task.

3.2 Multi-Agent Experiments

An agent in a multi-agent setting not only makes decisions on the three events described in the single agent setup, but is extended to make decisions on tasks that cross agent boundaries. We provide performance comparisons of the four different strategies to meta-level control: Naive Heuristic Strategy (NHS); Sophisticated Heuristic Strategy (SHS); Deterministic Strategy; and Random Strategy within a multi-agent context.

Environment#	SHS	NHS	Deter.	Rand.
AMM	41.08%	39.64%	30.52%	21.56%
AMT	28.60%	27.51%	21.7%	19.87%
AHT	22.08%	21.28%	12.47%	13.97%
ALM	56.86%	56.66%	55.09%	22.15%
AML	45.11%	39.61%	21.14%	21.27%
AHL	33.80%	28.75%	22.0%	19.04%
ALL	65.12%	63.39%	52.84%	23.48%
CLL	76.95%	71.78%	28.11%	24.51%

Table 7: Comparison of percent of tasks completed over a number of environments; Column 1 is the environment type; Columns 2-5 represents the % of total time spent on control actions by each of the four algorithms for that environment;

The experimental setup is similar to the single agent set up except for the fact that two more decisions are added to the decision process, whether to negotiate with another agent about a non-local task and whether to renegotiate if a previous negotiation falls through. These two new decisions specifically require coordination with another agent for completing the task. The following are the heuristics for the two additional meta-level decisions. The NHS is a myopic variant of these heuristics. Table 8 describes SHS rules required to support each of the actions in Figure 5 for the *Presence of task requiring negotiation* event trigger. The event occurs when the *MetaNeg* information gathering action completes execution. Table 9 describes SHS rules required to support each of the actions in Figure 7 for the *Failure of negotiation* event trigger.

ID	NTUG	NTDL	NLUG	NLDL	NLS	P(HPT)	MLC Decision
1	L	*	H	*	L	H/M	Drop Negotiation and Reschedule (B1)
2	H	*	L	*	H	L	Choose NegMech1 (B2)
3	M	*	L/M	*	H	L	Choose NegMech1 (B2)
4	H	M/LS	L	*	H	L/M	Choose NegMech2 (B3)

Table 8: SHS rules for *Presence of Task requiring Negotiation* event trigger. The column headers are ID = Heuristic Rule Number; NTUG = New Task Utility Goodness; NTDL = New Task Deadline;NLUG = Utility Goodness of Non-Local agent's task set; NLDL = Deadline of Non-Local task set; NLS = Slack in Non-Local schedule; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

Experimental results describing the behavior of two interacting agents is presented in Figure 13 and Table 10. Performance comparison of the various strategies in an environment, AMM, over a number of dimensions are provided. 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 than NHS and also 14% more tasks are completed using SHS than the NHS. These preliminary results are encouraging since in this specific environment, the performance of the multi-agent system supports the hypothesis of this paper.

We have now presented two context sensitive heuristic strategies: the Naive Heuristic strategy (NHS) that uses myopic information to make meta-level control action choices; and the Sophisticated Heuristic

ID	NTUG	NTDL	NLUG	NLDL	NLS	P(HPT)	MLC Decision
1	*	T	*	*	*	*	No ReNegotiation (C1)
2	M	*	L/M	*	H	L	Renegotiate using NegMech1 (C2)
3	H	LS/M	L	*	H	L/M	Renegotiate using NegMech2 (C3)

Table 9: SHS rules for *Failure of Negotiation* event trigger. The column headers are ID = Heuristic Rule Number; NTUG = New Task Utility Goodness; NTDL = New Task Deadline;NLUG = Utility Goodness of Non-Local agent's task set; NLDL = Deadline of Non-Local task set; NLS = Slack in Non-Local schedule; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

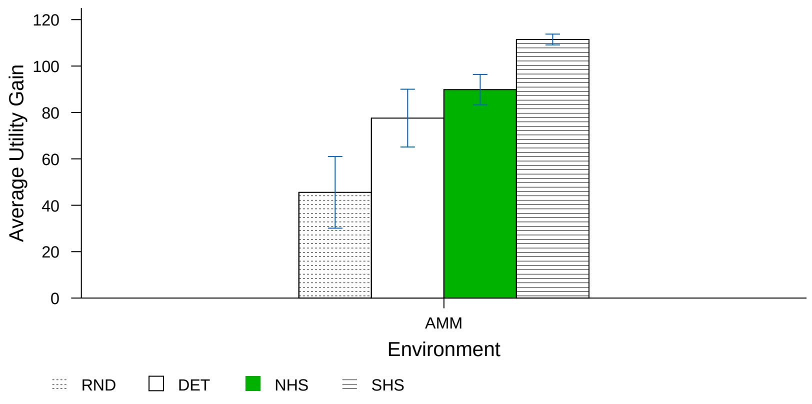


Figure 13: Average utility comparison between heuristic strategies and baseline strategies in a multi-agent environment. The error bars are one standard deviation above and below each mean

strategy (SHS) that uses current state information and predictive information about the future to make non-myopic action choices. A description of the decision rules used in each of these strategies is provided. The experimental evaluation described in this section lead to the following conclusions : Meta-level control reasoning is advantageous in resource-bounded agents in different types of environments; the high-level features are good indicators of the agent state and facilitate effective meta-level control; the heuristic strategies establish the positive effects of meta-level control in resource-bounded agents because they outperform deterministic and random strategies; and predictive information about future arrival tasks is useful in some environments and not in others.

4 Reinforcement Learning Strategy

Can an agent automatically learn meta-level control policies for specific environments based on the high-level state information described in Section 2? Does this learned policy outperform the corresponding hand-crafted policy for that environment as described in Section 3? These are the two questions addressed in this section.

The high-level goal of this paper is to create agents which can maximize the social utility by successfully completing their goals. These agents also necessarily have limited computation, and detailed models of the

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 10: Performance evaluation of four algorithms for two agents in a environment AMM with a combination of tasks, medium frequency of arrival and medium deadline tightness. 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 control actions(CT); row 4 is percent of tasks rescheduled (RES); Row 5 is the percent of total tasks completed (PTC); Row 6 is percent of tasks delayed on arrival (PTDEL)

task environments are not readily available. Reinforcement learning is useful for learning the utility of these control activities and decision strategies in such contexts. Section 4.1 describes the construction of a Markov decision process-based [32] meta-level controller which uses reinforcement learning techniques to approximate an optimal policy for allocating computational resources. This approach to meta-level control implicitly deals with opportunity cost as a result of the long-term effects of the meta-level decisions on utility. Sections 4.3 describes the complexities of the issues faced by multi-agent reinforcement learning agents. Experimental results describing the performance of the learned polices in both the single-agent and multi-agent cases are provided.

4.1 Reinforcement Learning

Reinforcement Learning [1, 21, 49, 50, 48, 55, 56] is a mathematical framework used by agents to learn how to map situations to actions so as to maximize a numerical reward signal. Supervised Learning (commonly used in research in machine learning, statistical pattern recognition and artificial neural networks) is learning from examples provided by a knowledgeable external supervisor. Reinforcement Learning is different from supervised learning in that the agent does not learn what actions to take from a "supervisor". Instead the usual approach taken by reinforcement learning agents involves discovering which actions yield the most reward by trying them out, associating expected reward values with different agent states, and using reward values to choose actions.

Two key features of reinforcement learning are the exploration-exploitation trade-off and credit assignment. A reinforcement learning agent, to maximize its reward, must prefer (exploit) actions which it has tried in the past and found to be effective in producing rewards. But to discover such actions, the agent has to try (explore) actions it has not selected before. The agent should be able to explore the action space to make better action selections in the future while at the same time progressively favor those actions that appear best.

The temporal credit assignment problem involves distributing rewards over a sequence of state-action pairs that lead up to that reward. When actions are not rewarded immediately but receive a large positive or negative reward some time later, it is called delayed reinforcement. Reinforcement learning algorithms typically use a scheme for assigning the appropriate credit to all preceding state-action pairs after receiving a delayed reinforcement.

The learning approach adopted for the meta-level control problem is based on the algorithm developed in [46] where reinforcement learning is used in the design of a spoken dialogue system. Their problem is similar to the meta-level control problem in that it is also a sequential decision making problems and there is a bottle neck associated with collecting training data. As described in the experimental setup in Section 3, each episode lasting 500 simulation time clicks takes about 180 real-world seconds. It takes about 150 real-world hours to obtain data from 3000 training episodes making data collection quite expensive.

As discussed previously, our MDP-based meta-level controller (MLC) uses a set of high-level qualitative features that is constructed to abstract the real state information as much as possible without losing critical information. The appropriate actions to take in each state are defined and the reward function is determined by the utilities accrued by each completed domain task. The meta-level control policy is a mapping from each state to an action. 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 is implemented. Each episode is a sequence of alternating states, actions and rewards. As described in [46], the transition probabilities of the form $P(s'|s,a)$ are estimated, 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 MDP model representing system behavior for a particular environment is obtained from state set, action set, transition probabilities and reward function. Confidence in the accuracy 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 the optimal policy in the estimated MDP is determined using the Q-value version of the standard value iteration algorithm [48]. $\gamma \epsilon [0,1)$ is a discount-rate parameter which determines the present value of future utility gains. In the experimental section, we vary the discount rate to determine its effect on the meta-level control decisions. The expected cumulative reward (or Q-value) Q(s,a) of taking action a from state s is calculated in terms of the Q-values of successor states via the following recursive equation [48]:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

When the value iteration algorithm converges[11], an optimal meta-level control policy (according to the estimated model) is obtained by selecting the action with the maximum Q-value at each state. The optimality of the policy depends on the accuracy with which the estimated MDP represents the particular environment.

4.2 Single-Agent Experiments

The experimental setup is as described in the previous sections. The training data for the RL strategy consisted of 3000 episodes with each episode executing for 500 time steps . The training data as mentioned earlier is exploratory in that at each decision point one action from a set of allowable actions is chosen at random. After the 3000 episodes were completed, the estimated transition probabilities and reward function were determined. The meta-level control policy was determined using the Q-value version of value iteration as described above in the algorithm. The policy was then used on a test run consisting of 300 simulation test episodes.

[11]The algorithm iteratively updates the estimate of Q(s,a) based on the current Q-values of neighboring states and stops when the update yields a difference that is below a threshold.

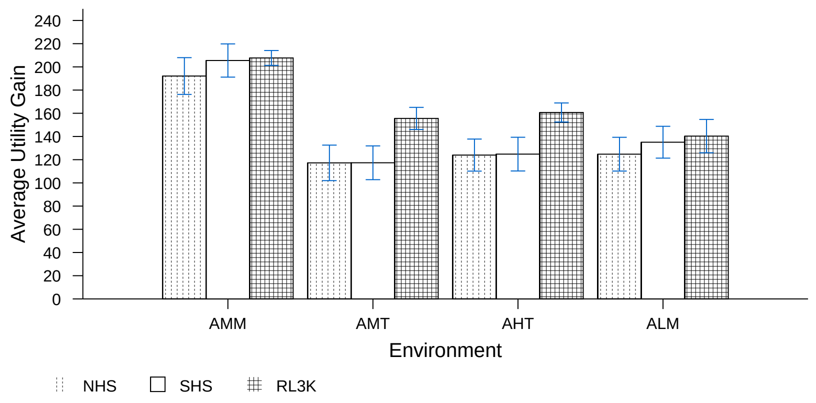


Figure 14: Utility comparison of learning method to heuristic strategies for four different environments. The error bars are one standard deviation above and below each mean

The results described in Figure 14 show the utility accrued by the reinforcement learning, SHS and NHS strategies for four environments AMM, AHT, AMT and ALM. The data collection bottleneck described previously limited the number of environments considered to four. The four environments were chosen to represent problem classes where interesting behavior of meta-level control could occur: medium constrained environments (AMM, ALM) and tightly constrained environments (AHT, AMT). The following performance results are established experimentally:

1. In two of the environments, AMT and AHT, the RL strategy using the policy based on 3000 training episodes performed significantly better ($p < 0.05$) than the SHS with respect to utility and had significantly lower control duration.
2. In the two other environments, AMM and ALM, the RL strategy using the policy based on 3000 training episodes performed as well as (no significant difference at $p < 0.05$) than the SHS with respect to utility and had significantly lower control duration.
3. In loosely constrained environments like ALM, agents have enough resources to complete tasks successfully within the deadlines without too much contention of resources.

Figure 15 describes the percent of total time spent on control actions. The RL method spends significantly less time on control actions ($p < 0.05$) than the heuristic strategies in all four environments. The RL optimizes its actions in a non-myopic fashion since it can learn a more accurate model of the sequential decision making process than the heuristic strategies.

Learning Curve Saturation: Figure 16 describes the effect of increasing training data on the performance of the learned policies. After every 1000 episodes, the cumulative transition probabilities and reward function were estimated and the corresponding policy was computed. This policy was then applied to 300 test episodes and the average results were computed. The performance of the agent improves with added training but the improvement does not increase proportionately with the training size. This seems to indicate that increased training data will not necessarily guarantee a monotonic improvement in performance and that the performance improvement will flatten out after a certain amount of training. This threshold is determined for each specific environment experimentally in this paper. 3000 seemed to be a good threshold

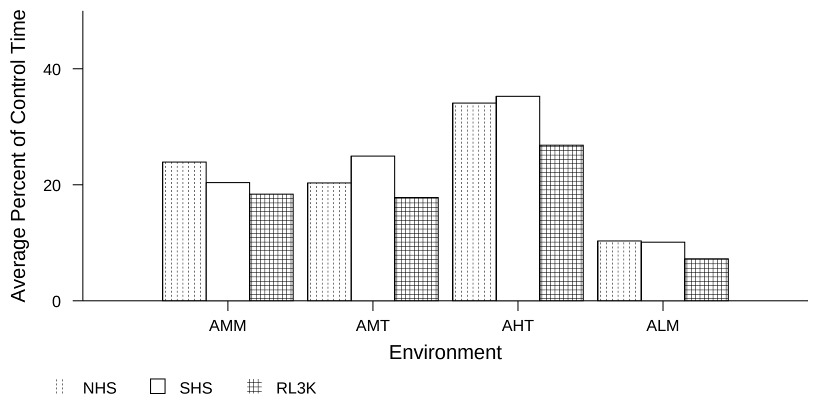


Figure 15: Control time comparison of learning method to heuristic strategies for four different environments

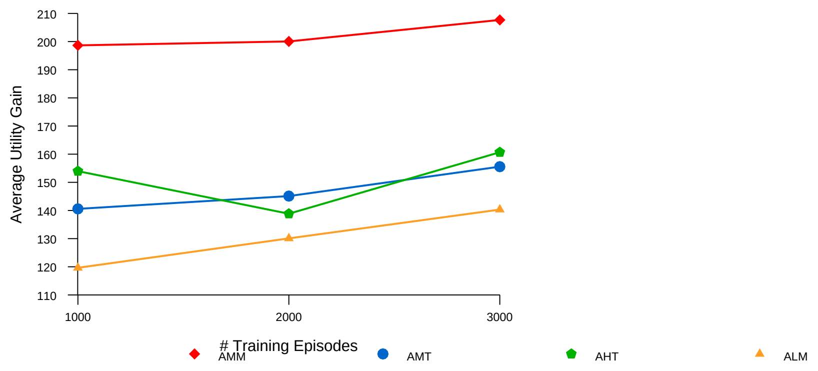


Figure 16: Relation of average utility to increasing training data

for training size for the four environments described. The dip in the curve for AHT at episode 2000 is a local minima which occurred because of the tightly constrained environment. On deeper analysis of the data, we found that tasks in episode 2000 had extremely tight deadlines[12] and hence considerably fewer number of tasks could be completed. Figure 17 describes the relation of the percent of control durations to increasing training size.

Table 11 describes the actual values of the measures described in the preceding discussion.

Significance of discounting: γ in the dynamic programming formulation denotes the discount factor. The discount factor determines how much value is given to future rewards. When γ is set to 1.0, the agent gives a lot of importance to the long term effects of its current decision. When γ is set to 0.0, the agent does a one-step look ahead and is very myopic in its decision making. Figure 18 describes the utility gained by the agent after 3000 training episodes. Three meta-level control policies with γ set to 1.0, 0.5 and 0.0 are computed. These polices are then used to evaluate 300 test episodes and the average utilities over these 300 episodes are computed. Table 12 describes the values of the utility gained and the corresponding percent of control time for the three different polices. Column 1 is the type of environment, Column 2 describes

[12]This is determined by the simulation environment.

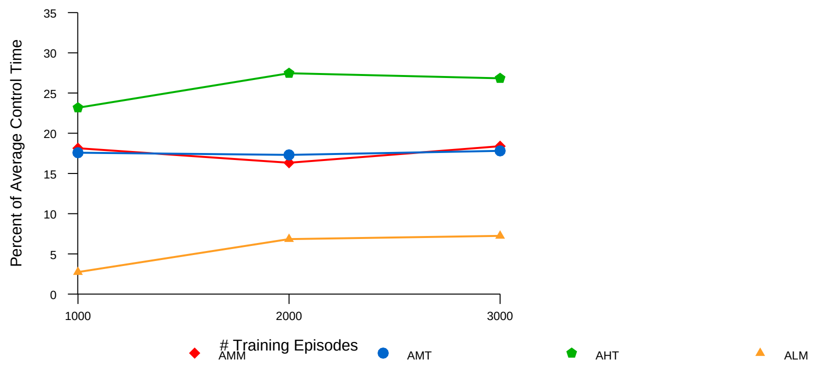


Figure 17: Relation of control time durations to increasing training data

Environment	RL-3000	RL-2000	RL-1000	SHS	NHS
AMM-UTIL	207.69	200.05	198.65	205.49	192.10
AMM-CT	18.39%	16.33%	18.14%	20.37%	23.92%
AMT-UTIL	155.56	145.11	140.57	117.34	117.25
AMT-CT	17.81%	17.31%	17.58%	24.95%	20.32%
AHT-UTIL	160.68	138.84	153.97	124.80	123.96
AHT-CT	26.83%	27.46%	23.17%	35.26%	34.09%
ALM-UTIL	140.32	130.09	119.64	135.05	124.74
ALM-CT	7.24%	6.84%	2.74%	10.11%	10.32%

Table 11: Utility and control time comparisons over four environments; Column 1 is the environment type; Column 2, 3 and 4 represent the performance characteristics of the RL policy after 3000, 2000 and 1000 training episodes respectively; Column 4 and 5 represent the performance characteristics of SHS and NHS respectively;

the performance characteristics when the agent has a completely myopic view (γ=0.0), Column 3 describes the performance characteristics and control time when the agent has a partially myopic view (γ=0.5) and Column 4 describes the performance characteristics when the agent gives a lot of priority to long term effects of its decisions.

In medium constrained environments such as AMM and ALM, the average utility gained using the policy with γ set to 1.0 is significantly better ($p<0.05$) than the partially myopic policy with γ=0.5 and the myopic policy with γ=0.0. In tightly constrained environments such as AMT and AHT, the difference in performance of the non-myopic policy, the partially myopic policy and the myopic policy was not significant at the 0.05 level. These environments are so tightly constrained and are too dynamic to be able to effectively predict the future events and act on that information.

4.3 Multi-Agent 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

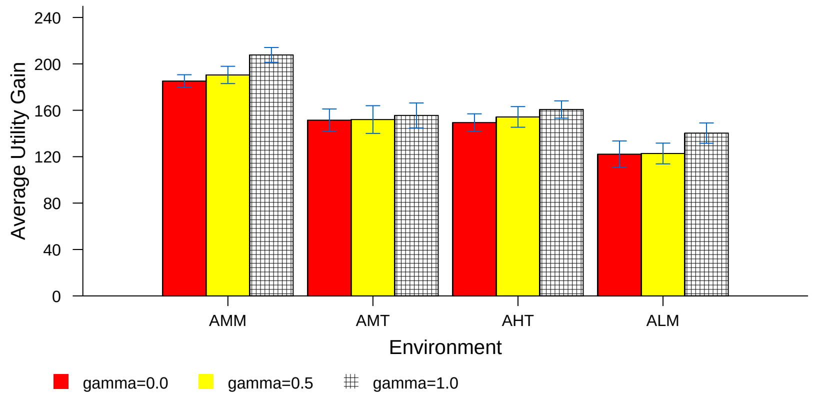


Figure 18: Utility gains with varying discount rate (γ = 0.0, 0.5, 1.0). The error bars are one standard deviation above and below each mean

Environment	γ=0.0	γ=0.5	γ=1.0
AMM-UTIL	185.19	190.46	207.69
AMM-CT	19.56%	18.63%	18.38%
AMT-UTIL	151.48	151.99	155.56
AMT-CT	17.95%	17.90%	17.80%
AHT-UTIL	149.40	154.26	155.56
AHT-CT	26.42%	26.23%	17.81%
ALM-UTIL	122.16	122.74	140.32
ALM-CT	6.88%	6.90%	7.24%

Table 12: Comparison of utility gain and percent of control time for four different environments while varying the discount rate (γ = 0.0, 0.5, 1.0)

system to perform better as a whole. In this section, we look at multi-agent scenarios similar to the 2-rover scenario described in Section 2. The multi-agent aspect of the problem arises only 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). Multi-agent reinforcement learning has been recognized to be much more challenging than single-agent learning, since the number of parameters to be learned increases dramatically with the number of agents. In addition, since agents carry out actions in parallel, the environment is usually non-stationary and often non-Markovian as well [27]. The experiments describe results on the convergence rates of the policies of the two agents in simple scenarios.

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 costs described in Section 3.1 are assumed. Additionally, the decision to negotiate and whether to renegotiate is assumed to take 1 unit of time. Transmission delay is 1 unit of time. The

Environment	RL-3000	SHS	NHS
AMM-UTIL	118.56	111.44	89.84
AMM-CT	8.86%	9.21%	8.09%

Table 13: Utility and control time comparison for a multi-agent environment; Column 1 is the environment type; Column 2 represents the performance characteristics of the RL policy after 3000 training episodes; Column 3 and 4 represent the performance characteristics of SHS and NHS respectively;

amount of time for the the other agent to respond is not fixed but based on the duration of the meta-level decisions at the other agent.

The task environment generator in the multi-agent setup also randomly creates task structures while varying the complexity of tasks, frequency of arrival and tightness of deadline as described in Section 3.1 for two agents instead of one. Experimental results describing the behavior of two interacting agents is presented in Figure 19 and Table 13. Agent *RoverB's* was fixed to the best policy it was able to learn in the single agent environment. Agent *RoverA* then learned its meta-level control policy within these conditions. We did not address the deep issues involved in Multi-agent Reinforcement Learning involving concurrent learning [61] by agents. Performance comparison of the heuristic strategies to the RL strategy in a single environment, AMM, is provided. 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 RL strategy also learns policies that significantly outperform the NHS strategy in this environment. The performance of the multi-agent system supports the hypothesis of this paper.

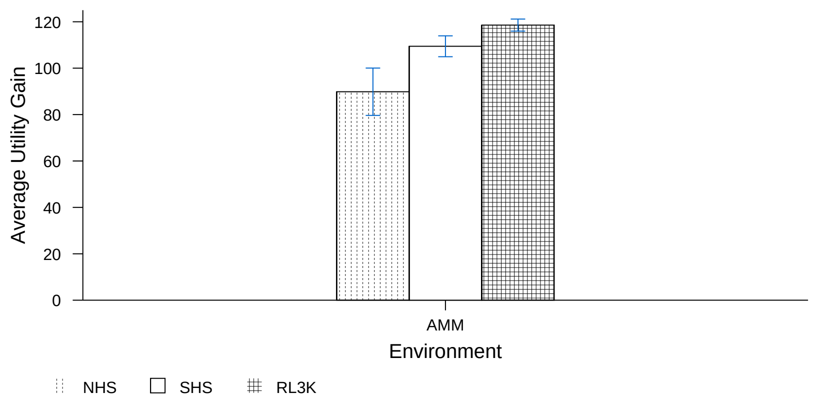


Figure 19: Average utility comparison between heuristic strategies and RL strategy (3000 training episodes) in a multi-agent environment. The error bars are one standard deviation above and below each mean

We have described 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 [46] 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. The sequential effects of the problem domain was verified by showing that varying the value of future rewards significantly affects the agent's performance.

5 Related Work

There has been enormous amount of work on intelligent agent control e.g. [3, 10, 28, 54, 60]. These systems describe flexible and goal-directed mechanisms capable of recognizing and adapting to environmental dynamics and resource bounds. The emphasis in these works is to build an adaptive control layer which reasons about domain-level costs. They do not, however, explicitly reason about the control costs. The meta-level control architecture described in this paper reasons explicitly about control costs and includes reasoning about costs at all levels of computation.

We will first discuss two well-known agent architectures that have some form of meta-level control. The Procedural Reasoning System (PRS) [11] is a hybrid system, where beliefs are expressed in first-order logic and desires represent system behaviors instead of fixed goals. It is an architecture for embedded systems that need to deliberate in real-time. A PRS agent consists of a database of the system's current beliefs, a set of current goals, a library of plans (called knowledge areas or KAs) and an intention structure. The KAs describe sequences of actions and tests that can be performed to meet a goal or react to a situation. The intention structure consists of a partially ordered set of those plans chosen for execution. An interpreter works with these components to select an appropriate KA based on beliefs and goals, place that plan in the intention structure and execute it. Meta-level KAs are functionally similar to the meta-level control layer in the agent architecture presented in this paper. The meta-level KAs are used to decide among multiple applicable domain KAs in a particular situation, reason about failure to satisfy goals, and manage the flow of control among intentions (including determining when to continue applying meta-level KAs versus executing the current domain-level plan). KAs are interruptible when external events cause changes to the database, thus allowing rapid response to changing environmental situations. PRS can be configured to respond to world events within a bounded amount of time though there is no explicit end-to-end reasoning nor performance guarantees. PRS is not concerned with cost of meta-level reasoning explicitly and thus differs significantly from the work presented here.

Hayes-Roth [16] describes an opportunistic control model that can support different control modes expected of an intelligent agent. The control model handles multiple goals, limited resources, and dynamic environments. She argues that in dynamic environments, it is often necessary to make decisions that may not be optimal, but are rather satisfactory under the current conditions. The meta-level control work described in this paper similarly computes approximate solutions rather than optimal solutions. The system she develops that solves problems closest to the complexity to the problems we are interested in is Guardian [17]. It is an experimental intelligent agent based on a blackboard architecture for monitoring patients in a surgical ICU. The agent consists of a manager that filters and processes inputs, a satisficing control cycle to bound the amount of time spent doing meta-level reasoning, and an anytime diagnosis component. Large amounts of input data arrive at the agent periodically. Much of this is low level data that just confirms current patterns, but occasionally important or unexpected information arrives. The input manager dynamically builds and modifies filters to sending new important information to be processed by the reasoning component while not overburdening it with needless detail as problem-solving progresses. High-level control takes the form of plans that are dynamically created at runtime by control knowledge sources. They emphasize that such dynamic construction is necessary because of the changing requirements of the filters in different problem-solving situations.

Guardian has an agenda based control mechanism. Its satisficing control cycle chooses the best action to perform by processing actions most likely to be rated highly first. As soon as an action is found that is good enough or the time limit for control reasoning has run out, the best action found so far is recommended. This time limit is set dynamically by control plans that can adjust the sequence and type of knowledge that is used in a specific situation; control input filters to separate out important data; and adjust the satisficing control cycle to quickly determine how to respond to it. Guardian, however, is not equipped with an overall planning mechanism to guide its real-time behavior. It does not reason about long-term effects of choices explicitly. Though Guardian has some ability to dynamically balance the amount of computation to invest in control versus domain activities, it does it in a qualitative and implicit manner, rather than the more quantitative and non-myopic approach taken in our work.

More generally, flexible, autonomous systems in complex environments generally require the ability to reason about resource allocation to computation at any point in time. Doyle's 'rational psychology' project [9] is based on the idea that computations, or state changes, are also actions to be reasoned about. He used the idea of bounded rationality in the context of beliefs, intentions and learning. Horvitz [20] also studied rational choice of computation in the context of designing intelligent systems.

The basic idea of bounded rationality arises in the work of Simon with his definition of procedural rationality [43]. Simon's work has addressed the implications of bounded rationality in the areas of psychology, economics and artificial intelligence [45]. He argues that people find satisfactory solutions to problems rather than optimal solutions because people do not have unlimited processing power. In the area of agent design, he has considered how the nature of the environment can determine how simple an agent's control algorithm can be and still produce rational behavior. In the area of problem-solving, Simon and Kadane [44] propose that search algorithms for finding solutions to problems given in terms of goals are making a trade-off between computation and solution quality. A solution that satisfies the goals of a problem is a minimally acceptable solution. Good's type II rationality [13] is closely related to Simon's ideas on bounded rationality. Type II rationality, which is rationality that takes into account resource limits, is a concept that has its roots in mathematics and philosophy rather than psychology. Good creates a set of normative principles for rational behavior that take computational limits into account. He also considers explicit meta-level control and how to make decisions given perfect information about the duration and value of each possible computation.

In order to make the trade-offs necessary for effective meta-level control, the meta-level controller needs some method for predicting the effect of more computation on the quality of a plan. One method for doing this is to use a performance profile. The idea comes from the study of anytime algorithms. Anytime algorithms can be interrupted at any point to return a plan that improves with more computation [6]. The performance profile gives the expected improvement in a plan as a function of computation time. An alternative to using performance profiles is to use the performance of the planner on the current problem to predict the future. Nakakuki and Sadeh use the initial performance of a simulated annealing algorithm on a machine shop scheduling problem to predict the outcome for a particular run [30]. They have found that poor initial performance on a particular run of the algorithm is correlated with poor final performance. This observation is used to terminate unpromising runs early and restart the algorithm at another random initial state.

Anytime algorithms can be combined to solve complex problems. Zilberstein and Russell [59] look at methods for combining anytime algorithms and performing meta-level control based on multiple performance profiles. Combining anytime algorithms produces new planning algorithms that are also characterized by a performance profile. Compilation techniques described in [60], can be used to compile programs consisting of both anytime and traditional algorithms[13]. Hansen and Zilberstein [14] extend previous work on meta-level control of anytime algorithms by using a non-myopic stopping rule. It finds an intermediate strategy between continuous monitoring and not monitoring at all. 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 in this paper. It deals with the single meta-level question of monitoring and considers the sequential effects of choosing to monitor at each point in time. It keeps the meta-level control cost low by using a lookup-table for the policy.

Harada and Russell [15] describe initial work where the computational process is explicitly modeled. It provides initial ideas for using search as the model of computation in the Tetris domain. They propose the use of Markov Decision Processes and reinforcement learning as their solution approach. This work was not pursued further[14]. The methodology in this research was developed independently of their effort. It was built for a complex domain where the meta-level decisions have down-stream effects. The domain is characterized by uncertainty in action durations and utility accrued. Russell and Wefald [39] define meta-level control as the ability of an agent to choose between executing a computational action which changes the internal state of an agent and a physical action which changes the environment. They show that the agent will continue to deliberate only if it is possible that the computation will change the agent's current choice of physical action. They describe an ideal control algorithm as one that will continue to perform the computation with the highest expected net value until no computation has positive expected value. When there is no computation left, the external action that is preferred according to the internal state resulting from the last computation is executed. They view the meta-level control problem as one of calculating the expected values of various computations. Since the computations can be arbitrarily long, they approximate the expected value computation using simplifying assumptions. Particularly, they use the agent's own utility estimation function to estimate the expected value of computations. They also make the analysis tractable by making myopic assumptions such as the meta-greedy and single step assumptions which could lead to underestimation of some computations. From their model, it is clear that the knowledge necessary to assign values to computations resides in the probability distribution for the future utility estimates of the top-level actions (external actions). They assume that these probability distributions can be obtained by gathering statistics on past computations. The approach we take in our work is a constructive one that Russell calls meta-level rationality. By approximating the correct meta-level decisions, the agents attempt to produce high expected utility within the resource limits. However, the agents provide no guarantees about their optimality. In our model we choose between an external action and sequence of computational actions in a single episode. We reason about and execute sequences of such sequences that consist of both external actions and computational actions that can occur in a single episode. Like Russell and Wefald [39], our goal is to choose the sequence of control and domain (external) actions that would maximize performance in the long run.

Goldman et al [12] develop computationally feasible heuristics that make greedy deliberation scheduling decisions quickly in the context of SA-CIRCA, a self-adaptive control architecture. They model the meta-level the deliberation scheduling problem as a MDP. Their work, however, does not handle the problem of trading off deliberation versus domain activities. In their framework, the execution subsystem does not compete for resources with the deliberation system and the choices involve only deliberative activities. Schut and Wooldridge [41] have independently observed that a Markov Decision Process-based model towards decision making is most similar to the bounded optimality model. The abstract representation of states in our

[13]The performance profile of a traditional algorithm is presumably a single step function.
[14]Personal communication with second author.

MDP for meta-level control allows us to bound the complexity of the problem. Schut and Wooldridge [41] provide a useful and in-depth comparison of continuous deliberation scheduling [3], discrete deliberation scheduling [39] and bounded optimality [38] methods for single-agent meta-level control. Russell, Subramanian and Parr [38] cast the problem of creating resource-bounded rational agents as a search for the best program that an agent can execute. This definition of rationality does not depend on the method used to create a program or the method it uses to do computation but only on the behaviors that result from running the program. In searching the space of programs, the agents, called bounded-optimal agents, can be optimal for a given class of programs or they can approach optimal performance with learning, again given a limited class of possible programs. The computation of bounded optimal agents can still be very hard and additional assumptions are made in this work to tackle the complexity. Our approach to meta-level control involves construction of agents similar to these bounded optimal agents [38]. We too do not assume complete accessibility to the environment, which makes our approach applicable to a wide range of problems and delivers an execution model which makes it relevant to real-world applications. While our model has targeted only finite horizon problems (episodic environments), our model accounts for computational resources and takes advantage of the Markov Decision model to bound computation.

Algorithms for sequential Reinforcement Learning (RL) tasks have been studied mainly within a single agent context [1, 50, 55]. Some of the later work described below have applied RL methods such as Qlearning to multi-agent settings. In many of these studies, the agents learn about either simple dependent tasks or independent tasks. Sen et al. [42] describe 2-agent block pushing experiments, where the agents try to make the block follow a line by independently applying forces to it. Tan [52] reports on grid-world predator-prey experiments with multi-agent RL, focusing on the sharing of sensory information, policies, and experience among the agents. Unfortunately, just slightly harder prisoner's dilemma problems [40] have uncovered discouraging results. The standard Q-learning algorithms are not guaranteed to converge in non-stationary environments where all agents are learning simultaneously. The agents had to keep detailed accounts of their entire history and interaction patterns, in addition to implementing long exploration schedules to achieve convergence.

Crites and Barto [**?**] apply multi-agent RL algorithms to elevator dispatching, where each elevator car is controlled by a separate agent. The agents don't communicate with each other and an agent treats the other agents as a part of the environment. The problem is complicated by the fact that their states that are not fully-observable and they are non-stationary due to changing passenger arrival rates. Littman and Boyan [25] describe a distributed RL algorithm for packet routing, using a single, centralized Q-function, where each state entry in the Q-function is assigned to a node in the network which is responsible for storing and updating the value of that entry. In our work, the entire Q-function, not just a single entry, is stored by each agent. Littman [26] experiments with Q-learning agents that try to learn a mixed strategy that is optimal against the worst possible opponent in a zero-sum 2-player game.

Lagoudakis and Littman [24] describe a RL-based approach for dynamically selecting the right algorithm for a given instance based on instance features while minimizing overall execution time. This problem has several interesting overlaps with the meta-level control problem although they only reason about a single problem instance at any point in time. The sequential nature of the decision process in our work complicates the reasoning process. Other multi-agent learning research has used purely heuristic algorithms for complex real-world problems such as learning coordination strategies [47] and communication strategies [22] with varying success.

The meta-level control architecture described in this paper differs from the above mentioned works in that it uses RL to make meta-level control decisions in a complex sequential decision making, cooperative multi-agent environment. It emphasizes the necessity for alternative ways of performing computations and it dynamically reasons about the cost of computation based on the current context.

Many researchers in AI have addressed the need for abstractions to solve large-scale planning problems. Abstraction is the process by which a system simplifies its decision making process by choosing only the information relevant to decision making process and ignoring the irrelevant information. In the RL literature, temporal abstraction and hierarchical control have been used to combat the curse of dimensionality in a principled way. The aim of hierarchical RL is to discover and exploit hierarchical structure within a Markov decision problem. The options formalism of Sutton, Precup and Singh [51] describes closed-loop policies for taking action over a period of time. They show that options can be used interchangeably with primitive actions in both planning methods and learning methods. The foundation of the theory of options is provided by the existing theory of Semi-Markov Decision Processes (SMDPs) and associated learning methods. Parr and Russell [31] developed an approach to RL in which the policies considered are constrained by hierarchies of partially specified machines. This allows for the use of prior knowledge to reduce the search space. The SMDP -based framework allows knowledge to be transferred across problems and for component solutions to be recombined to solve larger and more complicated problems. The MAXQ framework of Dietterich [8] relies on creating a hierarchy of SMDPs whose solutions can be learned simultaneously. He shows that hierarchical RL using the MAXQ framework can be much faster and more compact than flat RL. He also shows that recursively optimal policies can be decomposed into recursively optimal policies of individual subtasks and these subtask policies can be re-used wherever the same subtask arises.

These works emphasize the importance and advantages of abstraction in RL. The meta-level control work however is different from these works because it uses abstract representation of the state based on the similarity of states. In other words, A number of the agent's real states are represented by a single abstract state because of their similarity of their feature values (excluding time) which is different from the temporal abstractions described in the above three works.

6 Conclusions

This paper explores the issue of meta-level control in complex agents situated in social and dynamic environments. As discussed in the introduction, complex agents can concurrently perform several different goals of varying worth and deadlines, dynamically choose alternate ways to achieve these goals and make choices on how much effort to spend on deliberative actions. Deliberations about the tasks may involve resource-intensive computation. Also, the control decisions made by the agent may have down-stream effects on the availability of resources and processing available to future tasks. Meta-level control is the ability of an agent to optimize its long-term performance by choosing and sequencing its deliberation and execution actions appropriately. It reasons about the cost of computation at all levels as a first-class entity.

This paper establishes the following hypothesis: *Meta-level control with bounded computational overhead allows complex agents to solve problems more efficiently in dynamic open multi-agent environments. Meta-level control is computationally feasible through the use of an abstract representation of the agent state. This abstraction concisely captures critical information necessary for decision making while bounding the cost of meta-level control and is appropriate for use in automatically learning the meta-level control policies.*

Main Results

A meta-level agent architecture for bounded-rational agents which supports alternative approaches for deliberative computation is described. The meta-level control has limited and bounded computational overhead and supports reasoning about costs of planning, scheduling and negotiation as first-class entities. Accounting for costs of reasoning at all levels is necessary for guaranteeing the performance characteristics of real-time systems. An experimental testbed to evaluate the agent performance was set up using the MASS simulation environment where the architecture described was fully implemented. Tasks of varying complexity were used to study the performance of the architecture using various policies for meta-level control. A deterministic policy was used as a base-line for evaluation. An agent while deciding to trade off deliberation versus execution action is in effect reasoning on whether to retain control of its resources or to decide to perform a task to gain the associated utility while at the same time giving up control of the required resources. One of the interesting contributions of this work is the way it exploits knowledge of the tasks from the task structures. The state features are computed using thresholds which are specific to the task being analyzed.

This paper establishes that meta-level control in resource-bounded rational agents is beneficial using empirical evidence. Two context sensitive hand-generated heuristic strategies are defined: the Naive Heuristic strategy (NHS) that uses myopic information to make meta-level control action choices; and the Sophisticated Heuristic strategy (SHS) that uses current state information and predictive information about the future to make non-myopic action choices. The heuristic strategies significantly outperform ($p<0.05$) deterministic and random strategies confirming the importance of meta-level control. We also experimentally show that a few abstract features which accurately capture the state information and task arrival model enable the meta-level control component to make computationally-bound decisions which significantly improve agent performance.

An observation made from the experiments is that the cost of control actions in terms of resources used is an important factor in determining the need for meta-level control. Meta-level control is advantageous in environments where the control costs are high enough so that the resources available for domain actions are significantly constrained. When the cost of control actions becomes significantly inexpensive in a non-stationary environment, the hand-generated rules have to be rewritten to account for this fact. The learning method, on the other hand, can automatically construct a policy offline which adapts to the new costs.

This work also provides insight into the usefulness of reinforcement algorithms in complex multi-agent sequential decision-making problems. A reinforcement learning approach which equips agents to automatically learn meta-level control policies is described. This empirical algorithm is a modified version of the algorithm developed by [46] 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 consisting of 3000 runs. 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 if not better than the carefully hand-generated heuristic policies at the $p<0.05$ level. One surprising and useful result was that the agents were able to learn useful meta-level control policies with a small amount of training (3000 episodes). The sequential effects of the problem domain were verified by showing that varying the value of future rewards significantly affects the agent's performance.

Applying this work

This paper shows that meta-level control can be effective in real-time environments, characterized by uncertainty and limited computational resources. In these environments, computational commodities such as time, memory, or information can be traded for gains in the value of computed results. It also shows that efficient and inexpensive meta-level control which reasons about the costs and benefits of alternative computations leads to improved agent performance in resource-bounded environments. This is a flexible, run-time approach which seeks to optimize rather than satisfice solution quality.

This work also shows that a meta-level control policy can be learned in a non-deterministic, inaccessible and model-free environment. In an inaccessible environment, an agent must maintain some internal state to try to keep track of the environment, since it is not possible for states to be identified just based on percepts. The learning strategy allows for meta-level control in uncertain environments whose model is not available. The empirical reinforcement learning algorithm allows the agent to construct a partial model of the environment and use the information to define effective action policies.

Additionally, the paper has identified scenarios in which predictive information about future task arrivals has limited utility. If the environment is characterized by high frequency of arrival of tasks with tight deadlines, then the meta-level controller will constantly have to reevaluate its decisions every time a new task arrives. These decisions are valid when made within a myopic context because of the dynamic environment. Hence predictive information about the future does not necessarily improve performance. If the environment is characterized by low frequency of arrival of tasks and the tasks have loose deadlines, then the environment is loosely constrained. In such environments, the downstream effects of decisions is minimal, since the tasks are spaced out enough so that there is minimal contention of resources by multiple tasks. This means predictive information about the future does not provide any additional performance advantage.

We plan to extend this work by introducing more complex features that will make the reasoning process more robust. And finally, we plan to reason about coordination, organizational adaptation and communication as control actions [34] to achieve our overall goal of introducing efficient meta-level control in cooperative multi-agent systems.

7 Acknowledgments

We would like to thank Professor Shlomo Zilberstein for his help in constructing the model described in Section 2.1 and Professor Andy Barto for his valuable comments on the Reinforcement Learning Algorithm. We also thank the two anonymous reviewers and Dr. Claudia Goldman for their detailed comments.

”Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number #F30602-99-2-0525 P00005. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory or the U.S. Government.”

References

[1] A. Barto, R. Sutton, and C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[2] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[3] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments, *Artificial Intelligence*, 67(2):245-286, 1994.

[4] C. Boutlier. Sequential Optimality and Coordination in Multiagent Systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[5] R. Crites and A. Barto, ”Improving Elevator Performance Using Reinforcement Learning”, Multi-ag In *Advances in Neural Information Processing Systems*, pages 8: 1017–1023”, 1996.

[6] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 49–54, Saint Paul, Minnesota, USA, 1988. AAAI Press/MIT Press.

[7] K. Decker. Taems: A framework for environment centered analysis and design of coordination mechanisms. In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.

[8] T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[9] J. Doyle. What is rational psychology? toward a modern mental philosophy. *AI Magazine*, 4(3):50–53, 1983.

[10] A. Garvey and V. Lesser. Issues in design-to-time real-time scheduling. In *AAAI Fall 1996 Symposium on Flexible Computation*, November 1996.

[11] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, WA*, pages (2) 677–682, 1987.

[12] Goldman, R.; Musliner, D.; and Krebsbach, K. Managing online self-adaptation in real-time environments. In *LNCS*, volume 2614. SV. 6–23, 2003.

[13] I. J. Good. Twenty-seven principles of rationality. In V. P. Godambe and D. A. Sprott, editors, *Foundations of statistical inference*, pages 108–141. Holt Rinehart Wilson, Toronto, 1971.

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

[15] D. Harada and S. Russell. Extended abstract: Learning search strategies. In *Proc. AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information, Stanford, CA, 1999.*, 1999.

[16] B. Hayes-Roth. Opportunistic control of action in intelligent agents. In *Proceedings of IEEE Transactions on Systems, Man and Cybernetics*, pages SMC–23(6):1575–1587, 1993.

[17] B. Hayes-Roth, S. Uckun, J.E. Larsson, D. Gaba, J. Barr, and J. Chien. Guardian: A prototype intelligent agent for intensive-care monitoring. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1503–1511, 1994.

[18] B. Horling, V. Lesser, and R. Vincent. Multi-agent system simulation framework. In *16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation*, EPFL, Lausanne, Switzerland, August 2000.

[19] B. Horling, V. Lesser, R. Vincent, and T. Wagner. The Soft Real-Time Agent Control Architecture. *Autonomous Agents and Multi-Agent Systems*, 12(1):35–92, 2006.

[20] E. 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.

[21] L. Kaelbling. *Learning in Embedded Systems*. PhD thesis, Stanford University, 1990.

[22] M. Kinney and C. Tsatsoulis. Learning communication strategies in multiagent systems, In *Applied Intelligence*, pages 9(1):71-91, 1998.

[23] K. Kuwabara. Meta-level Control of Coordination Protocols. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS96),* pages 104–111, 1996.

[24] M. Lagoudakis and M. Littman. Reinforcement learning for algorithm selection. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000),* page 1081, 2000.

[25] M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. Technical Report CS-93-165, 1993.

[26] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94),* pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann.

[27] M. Mataric. Reinforcement learning in the multi-robot domain, In *Autonomous Robots,* pages 4(1):73-83, 1997.

[28] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul. The Challenges of Real-Time AI. In *IEEE Computer,* pages 28(1):58–66, 1995.

[29] D. Musliner. Plan Execution in Mission-Critical Domains. In *Working Notes of the AAAI Fall Symposium on Plan Execution - Problems and Issues,* 1996.

[30] Y. Nakakuki and N. Sadeh. Increasing the efficiency of simulated annealing search by learning to recognize (un)promising runs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* pages 1316–1322, 1994.

[31] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems,* volume 10. The MIT Press, 1997.

[32] M. L. Puterman. *Markov decision processes - discrete stochastic dynamic programming.Games as a Framework for Multi-Agent Reinforcement Learning*. John Wiley and Sons, Inc., New York, 1994.

[33] A. Raja. *Meta-level Control in Multi-Agent Systems*. PhD thesis, University of Massachusetts at Amherst, Amherst, Massachusetts, June 2003.

[34] A. Raja, G. Alexander, and V. Mappillai. Leveraging Problem Classification in Online Meta-Cognition. In *Proceedings of AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management, Stanford,* pages 97–104, March 2006.

[35] A. Raja, V. Lesser, and T. Wagner. Toward Robust Agent Control in Open Environments. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 84–91, Barcelona, Catalonia, Spain, July, 2000. ACM Press.

[36] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[37] S. Russell and E. Wefald. *Do the right thing: studies in limited rationality*. MIT press, 1992.

[38] 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.

[39] 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.

[40] T. Sandholm and R. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma, *Biosystems Journal*, 37:147-166, 1995.

[41] M. Schut and M. Wooldridge. The control of reasoning in resource-bounded agents. *Knowledge Engineering Review*, 16(3):215–240, 2001.

[42] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, Seattle, WA, 1994.

[43] H. Simon. *From substantive to procedural rationality.* In Method and Appraisal in Economics (S. J. Latsis, Ed.) Cambridge University Press, pages 129–148, 1976.

[44] H. Simon and J. Kadane. Optimal problem solving search: All-or-none solutions. Artificial Intelligence, 6:235-247,1974.

[45] H. Simon. *Models of Bounded Rationality, Volume 1.* The MIT Press, Cambridge, Massachusetts, 1982.

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

[47] T. Sugawara and V. Lesser. On-line learning of coordination plans. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 335–345,371–377, 1993.

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

[49] R. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[50] R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[51] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[52] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

[53] R. Vincent, B. Horling, and V. Lesser. An agent infrastructure to build and evaluate multi-agent systems: The java agent framework and multi-agent system simulator. In *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems.*, volume 1887. Wagner and Rana (eds.), Springer,, January 2001.

[54] 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.

[55] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge, England, 1989.

[56] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[57] X. Zhang and V. Lesser. Multi-linked negotiation in multi-agent system. *Proceedings of the First International Joint Conference on Autonomous Agents And MultiAgent Systems (AAMAS 2002)*, pages 1207–1214, 2002.

[58] S. Zilberstein and A. Mouaddib. Reactive control of dynamic progressive processing. In *IJCAI*, pages 1268–1273, 1999.

[59] S. Zilberstein and S. J. Russell. Efficient resource-bounded reasoning in AT-RALPH. In James Hendler, editor, *Proceedings of the First International Conference of Artificial Intelligence Planning Systems (AIPS 92)*, pages 260–268, College Park, Maryland, USA, 1992. Morgan Kaufmann.

[60] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.

[61] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. *International Conference in Machine Learning*, 929-936, 2003.