# Leveraging Problem Classification in Online Meta-Cognition

**Anita Raja, George Alexander, Verghese Mappillai**
Department of Software and Information Systems
The University of North Carolina at Charlotte
Charlotte, NC 28223
{anraja,gralexan,vjmappil}@uncc.edu

**Abstract**

Open environments are characterized by their uncertainty and non-determinism. This poses an inevitable challenge to the construction of agents operating in such environments. The agents need to adapt their processing to available resources, deadlines, the goal criteria specified by the clients as well as their current problem solving context in order to survive. Our research focuses on constructing a meta-cognition framework that will enable agents to adapt to their dynamic environment. This will include deciding which environmental changes to address, how quickly they should be addressed and which of the different planning, scheduling and negotiation modes to use to address these changes. In this paper, we also describe how the classification of environmental changes plays a pivotal role in making the meta-level decisions.

## Introduction

Open environments are dynamic and uncertain. It is paramount for complex agents operating in these environments to adapt to the dynamics and constraints of such environments. The agents have to deliberate about their local problem solving actions and coordinate with other agents to complete problems requiring joint effort. These deliberations have to be performed in the context of bounded resources, uncertainty of outcome and incomplete knowledge about the environment. Deliberations may involve computation and delays waiting for arrival of appropriate information. Furthermore, new problems with deadlines can be generated by existing or new agents at any time. In this paper, we describe our recent efforts in augmenting agents with meta-cognitive capabilities to ensure good performance in open environments. Meta-cognition (Russell & Wefald 1989), (Horvitz 1989), (Goldman, Musliner, & Krebsbach 2003), (Raja & Lesser 2004) of a resource-bounded agent is the ability to efficiently trade-off the use of its limited resources between deliberations and the execution of domain actions.

We make the following assumptions in our work. The agents operate in a cooperative environment and can pursue multiple goals simultaneously. An example of an agent goal would be to appropriately handle a change in the environment, which is modeled as a problem in our framework. A scenario is a set of problems. Each agent has a model of its environment and is aware of the consequences of its actions, and those of collaborating agents, on the environment. These consequences include costs and time-dependent rewards. There are several alternative options for deliberating about a problem in a scenario and these options differ in their performance characteristics, for e.g. quick and dirty, low quality method versus a slow, high quality option. The agent's meta-cognitive capabilities will enable it to reason about which problems need to be processed by the agent, which deliberative actions to apply to the selected problems, when to invoke these deliberative actions and how much time to invest on these deliberative actions.

In this paper, we describe meta-cognition in the context of the DARPA/IPTO Coordinators program using problem classification techniques and Markov Decision Processes (MDPs). This research is part of our efforts to develop efficient decision-making techniques to guide problem solving in complex agents when they are faced with uncertainty, resource bounds and significant amount of interaction with other agents. Meta-cognition in Coordinators is designed to realize the following goals:

1. Given a set of problems, determine which problems to solve and in which mode a solver, i.e. scheduler, planner or coordination algorithm, should operate to maximize expected quality constrained by limited time for cognitive actions.
2. If a new problem arrives during schedule execution, determine if the solver (rescheduling, replanning or recoordination) should be recalled, and if it is, determine the best mode to use.

To achieve these goals, we equip agents with the ability to predict the agent's performance in various solver modes when presented with a novel problem using Naive Bayes Classification. We then use the performance information to model a sequential decision making process using a MDP to determine the optimal allocation of processor time and other resources to each problem in the scenario. At a high-level the meta-cognition module architecture is composed of a problem classification/abstraction component and a decision process component. The problem classification/abstraction component will help control the complexity of the meta-cognition decision process by weeding out superfluous information. The paper is structured as follows: We begin by motivating the need for meta-cognition in the Coordinators application and describe the TÆMS representation used to model the problem solving activities of individual agents. We then present our meta-cognition framework and use example scenarios from the Coordinators domain to describe the process of building of performance profiles, problem classification and action selection. We then present a preliminary evaluation of our approach, followed by a discussion of the generalizability and limitations of the approach and future next steps.

## The Coordinators Application Domain

Coordinators are intelligent agents that address the problem of effective coordination of distributed human activities. They assist their human counterparts to dynamically adapt their plans to environmental changes. The problem solving activities of the Coordinator agent are represented using C-TÆMS which is derived from the TÆMS (Task Analysis, Environment Modeling, and Simulation) (Decker & Lesser 1993) language. C-TÆMS models are hierarchical abstractions of multi-agent problem solving processes that describe alternative ways of accomplishing a desired goal; they represent major problems, the decision points and interactions between problems, but they do not describe the intimate details of each primitive action. Each agent has access only to its subjective view of the task model. It describes the subset of the global C-TÆMS model consisting of those parts of the problem that affect and are are affected by the local agent. C-TÆMS models have nodes representing complex actions, called *tasks* and primitive actions, called *methods*. Methods are owned by agents and may have duration distributions, release times (earliest start times) and deadlines. Methods may have multiple possible outcomes and each outcome are statistically characterized using a discrete model 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. Duration describes the amount of time that the action modeled by the method will take for the agent to execute. C-TÆMS models also capture the dependencies between methods in the form of non-local effects (NLEs). These dependencies can be modeled as hard constraints (Enables NLE) or soft constraints (Facilitates NLE) . Quality Accumulation Functions (QAFs) define how the quality of a task's children can be used to calculate the quality of the task itself. For e.g. in the case of the *sum* QAF the quality of the supertask is equal to the sum of the qualities of its children, regardless of order or which methods are actually invoked. In C-TÆMS the *sync-sum* qaf denotes a form of synchronization across agents and the quality obtained is the sum of the qualities of all the subtaks that start at the same time as the earliest subtask.

A scenario consists of 1-10 concurrent problems linked by a sum QAF. The arrival of a new goal or the modification of an existing goal e.g. modified deadlines or modified performance characteristics of methods are tagged as problems in this application. A window is defined by the task release time and the deadline. Problems are expected to be completed within their time windows to achieve quality. The following are some parameters used to define problems: Number of Windows (6-10), Window Tightness (VeryTight, Tight, Loose). Window Overlap (Small, Medium, Large, Complete), Number of Fallbacks (number of alternate methods an agent has for its tasks), NLE Loops (Number of agents in the chain) etc. The complete list of parameters can be found in (Decker & Garvey 2005).

Each Coordinator agent is composed of three modules to handle planning/scheduling (Taskmod), coordination (Coordmod) and communication with the human (Autonmod). Some of these modules are computation-bound while others are communication bound. The meta-cognition module (Metamod) basically decides how much processor and wall-clock time to assign to each module (Taskmod, Coordmod and Autonmod) for each problem in a scenario. It is possible for two problems to overlap resulting in resource contention. These resources could include processing times of individual modules within an agent as well as availability of other agents. In the worst case, processing for both problems may not be feasible. Metamod would then be required to pick one of the problems to work on. In addition to allocating module times to subproblems, Metamod must choose the appropriate problem solving settings (or modes) for the modules. With no problem overlap, Metamod could determine the best combination of module problem solving settings and time allocations for each problem, treating the problems independently, and using the release times and deadlines of each problem to determine the resource allocation to problem deliberations. However, even this relatively simple sequential presentation of problems poses difficulties for the Metamod because modules may have unpredictable interactions. For instance, better solutions might result in some cases by having modules do some "quick and dirty" processing first and use the results to focus subsequent processing. When problems overlap, Metamod's deliberation problem becomes more complicated. In this paper we present a framework that will allow Metamod to efficiently handle situations with both overlapping and non-overlapping problems.

There has been important previous work in meta-cognition. Russell and Wefald (Russell & Wefald 1989) describe an expected utility based approach to decide whether to continue deliberation or to stop it and choose the current best external action. They introduce myopic schemes such as meta-greedy algorithms, single step and other adaptive assumptions to bound the analysis of computations. We too model the meta-control problem as a decision theoretic problem where the goal is to perform the action with the highest expected utility. Our work can be viewed as a complete implementation of their probabilistic self modeling approach where the agents estimate the probabilities of taking certain actions in certain states. Hansen and Zilberstein (Hansen & Zilberstein 1996) extend previous work on meta-level control of anytime algorithms by using a non-myopic stopping rule. It can recognize whether or not monitoring is cost-effective, and when it is, it can adjust the frequency of monitoring to optimize utility. The monitor-

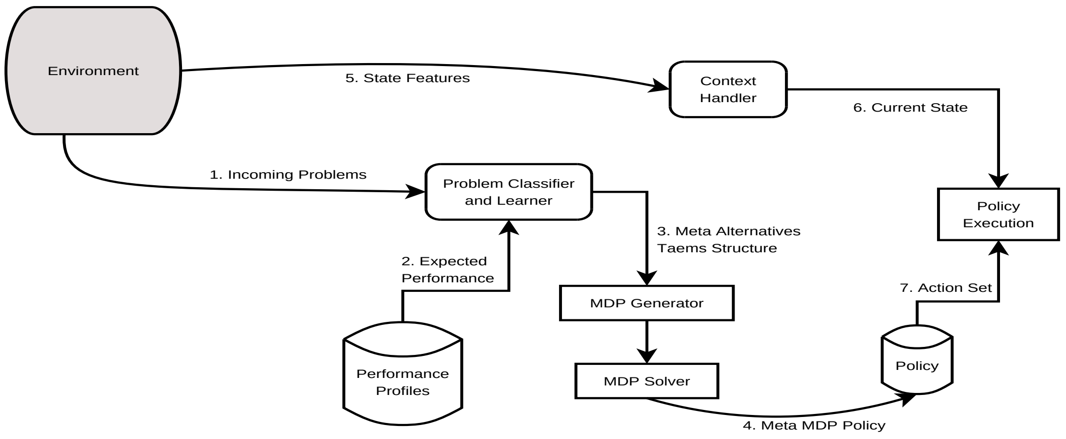

Figure 1: Control-flow in the Meta-Cognition Module (Metamod)

ing work has significant overlap with the foundations of the meta-level control reasoning framework described here. It deals with the single meta-level question of monitoring, considers the sequential effects of choosing to monitor at each point in time and keeps the meta-level control cost low by using a lookup-table for the policy. The work described in this paper handles multiple inter-dependent meta-level questions. It extends the meta-level control architecture described in the context of multi-agent systems (Raja & Lesser 2004). However there is a crucial difference. The Metamod framework in this paper does not make any assumptions about prior knowledge of the performance characteristics of the different deliberation actions on problems. The problem classifier component makes real time predictions about these performance characteristics. (Raja & Lesser 2004) assumes that knowledge about performance profiles of the various problems using the different deliberation components is readily available ahead of time, which is not feasible in the Coordinators domain.

## Meta-Cognition Framework

We now present a high-level description of the control flow within the Metamod component using Figure 1. When a problem modeled as a C-TÆMS task structure arrives at an agent, Metamod parses the structure to obtain problem features in order to classify the scenario into pre-defined performance types (Step 1). The classifier then uses the performance profile information corresponding to these performance types to build a TÆMS task structure of alternative solutions, called *MetaAlternatives task structure* (Steps 2 and 3). We use the TÆMS representation since it concisely captures the agent's problem solving from the meta-cognitive point of view. Details of the problem classification step are described in the *Problem Classification* section below. The TÆMS task structure is then translated into a Markov Decision Process (Puterman 1994). This is accomplished by passing the task structure to the MDP subcomponent which consists of the MDP Generator and MDP Solver. Details of this sub-component are described in the *Action Selection* section below. The MDP is evaluated to determine the best action to be taken by Metamod given the current environmental state (Step 4 in Figure 1). Metamod will determine the current state of the agent using the Context Handler (Step 5). The action corresponding to the current state is obtained from the optimal policy by the Policy Execution (Steps 6 and 7).

We now delve into greater detail of the control flow. When Metamod first receives a scenario, the *Problem Classifier & Learner* sub-component creates performance profiles of the problems in the scenario using various solver modes. A performance profile is a triple *solver mode*, *expected quality (EQ)*, and *expected duration (ED)*, that describes the expected performance characteristics of an entity under the various solver modes [1] in the form of statistical distributions. In the case of schedulers, EQ is the expected quality of the highest-rated schedule and may be continuous. ED is the expected time required to generate the schedule, not the expected duration of the schedule itself. The values of EQ and ED are placed into coarse buckets called *performance types* which are obtained by the conversion process described below. We also store a distribution of values across sub-ranges within a bucket which are used to build the MetaAlternatives task structure as described in the *Building the MetaAlternatives Task Structure* section.

Figure 2 describes an example MetaAlternatives task structure. The MetaAlternatives task structure describes alternate solver modes characterized by quantitative quality and duration distributions obtained from the performance profiles. In this task structure, the high-level task *MetaAlternatives* has a sum QAF meaning it can accrue quality by executing one or both of its subtasks *ModuleAction1* and *ModuleAction2* and adding the qualities. Each ModuleAction subtask has a max QAF, allowing it to accrue quality by successfully completing one or more of its subtasks and taking the maximum of the qualities obtained. In determining the performance profiles, we wish to avoid making assumptions about the ranges of quantitative performance characteristics

[1] Our discussion will focus on alternate modes for scheduling e.g., SchedulerA, SchedulerB, SchedulerC, etc. However the approach discussed is applicable to reasoning about alternate modes for other deliberative actions like coordination and planning

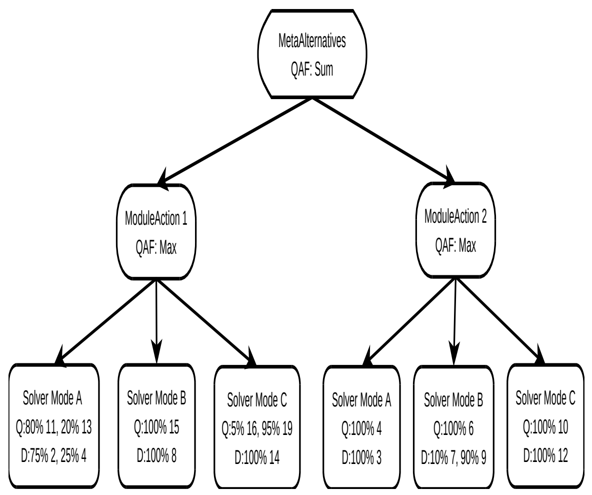

Figure 2: MetaAlternatives before MDP policy execution begins

of previously unseen problems. For example, a schedule with expected quality 100 would be considered high quality if it were part of a group of schedules ranging in quality from 25-110, but it would be low quality if it were part of a group of schedules where quality ranged from 90-250. We achieve this by doing the following:

1. Representing EQ as a ratio of achieved quality vs. quality upper bound to eliminate the need to make assumptions about the range of numerical values of quality in the problems while allowing us to estimate the optimality of various scheduling algorithms applied to a given problem.
2. Multiplying the distribution of EQ ratios associated with a problem's performance type by the quality upper bound computed for that problem in order to compare the expected numerical quality across several problems we may be considering taking action on.

For a given problem we derive an upper bound on the achievable quality in the following way:

1. Each task is assigned the mean of the quality distribution of its highest quality method.
2. Quality propagates upwards through the task structure according to whichever QAFs are present.
3. The final quality assigned to the root task is considered the "upper bound" for the problem.

In our current implementation, we define our performance types based on tenths of the quality upper bound. A topic of future research is determining how changing these divisions, for instance to fifths of the quality upper bound, will affect the performance of Metamod.

Unlike quality, which can vary greatly in numerical value without changing the complexity of a problem (for instance, by multiplying all quality values in a given problem by 100), scheduling time *is* related to problem complexity; and we assume that, for a given scheduling algorithm, it will not vary greatly over our problem space. Let $t_i$ denote the time spent by a particular scheduling algorithm to compute schedule $i$. We may compute $R_t = t_{max} - t_{min}$, the range of values over all schedules in a given training set. Then for the performance profiles, we have

$$ED_i = \begin{cases} low, & t_i < t_{min} + \frac{R_t}{3} \\ medium, & t_{min} + \frac{R_t}{3} \le t_i < t_{min} + \frac{2R_t}{3} \\ high, & t_{min} \frac{2R_t}{3} \le t_i \end{cases}$$

Thus low, medium, and high ED represent the lower, middle, and upper thirds of scheduler runtimes, respectively. To convert these to quantitative values, we simply add the appropriate multiple (.33, .66, or 1.0) of $R_t$ to $t_{min}$.

## Problem Classification

We define our performance types based on the expected performance of the problems in a particular mode. Thus in our current implementation, performance profiles are implicit, and we have NxM possible performance types (N scheduling modes times M types per mode). Also, each problem will correspond to exactly N performance types, one for each scheduling mode. Thus, we would need N classifiers to fully classify the problem (classifiers for SchedulerA, SchedulerB and SchedulerC). The problem features are characteristics obtained by examining the hierarchical structure of a problem that are relevant for differentiating among problems based on differences in performance characteristics. We determine performance types using Naive Bayes classification (Mitchell 1997) [2] This machine learning algorithm infers the type of a new problem based on the performance types of previously learned training problems with similar features.

In the Coordinators application, the MetaMod has to reason about scenarios where each scenario may contain one to ten problems. We use the following problem features for classification:

**F1: Average # of fallbacks per task:** This is the average number of methods available to execute a task.

**F2: Average window tightness:** This is the ratio of the width of a window compared to the expected duration of its tasks. The expected duration of a task is computed as the expected duration of its highest quality method.

**F3: Average window overlap:** Overlap between two windows is expressed as the overlap amount divided by the width of the wider window.

**F4: Average problem overlap:** This is computed for the problem in a similar way to how window overlap is computed.

**F5: Percentage of problems with non-local effects (NLEs):** This is the simple percentage of problems in the scenario that contain NLEs.

[2]other classification methods (such as C4.5 (Quinlan 1993)) could perhaps be used with varying degrees of success in other domains. The advantage of Naive Bayes method is that it is amenable to incremental updates. Also note that, in practice, the independence conditions do not necessarily have to hold for the algorithm to be successful(Domingos & Pazzani 1996).

**F6: Percentage of problems with sync points** This is the percentage of problems in the scenario that contain an explicit synchronization point

**F7: Percentage of problems with Min QAFs** This is the percentage of problems that acquire quality by a Min QAF.

**F8: Percentage of problems with Sum QAFs** This is the percentage of problems that acquire quality by a Sum QAF.

A particular configuration of some or all of these features describes a *problem class*, for example, 2 fallbacks per task, tight windows, small window overlap and small problem overlap describes a problem class. The following is the algorithm used by the Problem Classifier sub-component.

1. Define performance types as mentioned above.
2. Generate X random problems, and solve the problems using all N scheduling modes.
3. For each problem, generate a table listing its problem features, as well as its performance characteristics for each scheduling mode.
4. Based on the performance characteristics for each mode, assign each of the problems N corresponding performance types.
5. Train Bayesian classifiers to associate the features of the training problems with their respective assigned types (In fact, we will need to train a separate classifier for each scheduling mode).
6. When a new problem arrives, Metamod invokes the Bayesian classifiers to determine its expected types, given its problem features. Metamod then looks up these types in the performance profile table, and uses these performance profiles to build the MetaAlternatives task structure, which gets passed to the MDP Generator.
7. Based on the problem's actual performance, Metamod re-assigns the performance type corresponding to the scheduling mode we used and uses the problem features and the re-assigned type as a new training instance for the corresponding Bayesian classifier.

One question we are studying with this approach is to determine how small changes in problem features affect a problem's behavior, and how we can incorporate this information into the classifiers. We are also running empirical studies to determine the size of the initial training set that will ensure good performance by the classifiers.

To further illustrate the problem classification process, consider an example Coordinators scenario with three problems, P1, P2, and P3. We run the Bayesian classifiers (*SchedulerA_Classify*, *SchedulerB_Classify*, and *SchedulerC_Classify*) on each of the three problems and obtain the results shown in Figure 3.

We assume that performance profile information of the different problem scenarios using the various solver modes have been obtained offline ahead of time. We now look up

| | SchedulerA Classify | SchedulerB Classify | SchedulerC Classify |
|---|---|---|---|
| P1 | A_25 | B_12 | C_02 |
| P2 | A_14 | B_12 | C_15 |
| P3 | A_24 | B_17 | C_11 |

Figure 3: Classification Results using 3 Different Schedulers

| Scheduler Name | Performance Type | EQ | ED |
|---|---|---|---|
| SchedulerA | A_14 | 60% | Medium |
| SchedulerA | A_24 | 80% | Medium |
| SchedulerA | A_25 | 90% | High |
| SchedulerB | B_12 | 60% | Low |
| SchedulerB | B_17 | 60% | High |
| SchedulerC | C_02 | 20% | Low |
| SchedulerC | C_11 | 60% | Low |
| SchedulerC | C_15 | 60% | Medium |

Figure 4: Performance Profile Information

the performance profile information for each of the performance types that result from the above classification process. Figure 4 describes the predicted performance characteristics of each type using the relevant solver modes.

## Building the MetaAlternatives Task Structure

This section describes how we build the MetaAlternatives TÆMS task structure, once we have classified the incoming problem.

Suppose the above scenario S consisting of 3 problems P1, P2 and P3 has been assigned to an agent. For each problem i, we construct a subtask (ModuleActioni) that uses a Max QAF because it accumulates quality only from the schedule that it actually executes. the top-level task accrues the maximum of the qualities accrued by any of the child tasks. The methods used to accomplish this task correspond to the scheduler modes, for example SolverMode A, SolverMode B, SolverMode C, etc. The task structure for scenario S has a root task called MetaAlternatives with two subtasks, one for each problem, combined together by a Sum QAF, meaning quality accrued is the sum of the qualities accrued by the individual problems. Figure 2 is the MetaAlternative task structure for a scenario having 2 problems. The MetaAlternative task structure is sent to the Markov Decision Process (MDP) sub-component as described in the next section, and the meta-level control policy is then computed.

Now suppose the MDP policy indicates that we should solve scenario S using SolverMode A, and we have already begun executing this when a new scenario arrives. The changed scenario is called $S'$. We classify $S'$ and evaluate the resulting MDP. If the value of the new MDP differs significantly from the value of the original MDP, we drop the original policy and begin rescheduling according to the new MDP.

## Performance Profile Learning and Improving Metamod Problem Classification

A perfect Metamod would be able to classify an incoming problem with 100% accuracy, and the expected performance indicated by the performance profiles would always be the actual performance. However, when Metamod falls short of this ideal, we would like it to learn from its mistakes.

To illustrate the performance profile learning, we refer back to the example described in Figure 3. Suppose the meta-control MDP indicates that we should use SchedulerA in all three cases. We run SchedulerA on all 3 problems and we obtain the following actual results:

| | Quality | Duration |
|---|---|---|
| Problem 1 | 80% | Medium |
| Problem 2 | 60% | Medium |
| Problem 3 | 60% | Medium |

From Figure 4, we see that the classifier for SchedulerA correctly predicted the type of Problem 2 but was wrong about the other problems. The actual performance results indicate that Problem 1 was really of type A_24 and Problem 3 was really of type A_14. We do not know whether the classifiers for SchedulerB or SchedulerC correctly classified these problems, because we only got actual results from the scheduling mode specified by the meta-control MDP, which in this case is SchedulerA. In order to improve the accuracy of the SchedulerA classifier, we can next use the actual results as training instances, i.e:

`Train_SchedulerA_Classifier(`*Problem 1 features, A_24*`)`
`Train_SchedulerA_Classifier(`*Problem 2 features, A_14*`)`
`Train_SchedulerA_Classifier(`*Problem 3 features, A_14*`)`

An alternate learning method is that instead of learning performance profiles after every simulation, we would do batch learning by retaining information about problems and execution results to use in updating the performance profiles and classification algorithm and doing the updates in stages. We would store the following information: problem features, scheduler mode used, quality achieved, quality upper bound, time spent scheduling. Then after determining the true performance type of the problem based on our actual results, the problem features and performance type would be used to update the appropriate classifier.

## Action Selection

As described earlier, the MetaAlternatives task structure gets sent to the MDP sub-component. The MDP generator in the MDP subcomponent generates the MDP corresponding to the MetaAlternatives task structure and a policy is created by the MDP Solver. This MDP generation is based on the TÆMS to MDP translation algorithm (Raja, Lesser, & Wagner 2000).

A Markov Decision Process (Puterman 1994) is a probabilistic model of a sequential decision problem, where states can be perceived exactly, and the current state and action selected determine a probability distribution on future states (Sutton & Barto 1998). Specifically, the outcome of applying an action to a state depends only on the current action and state (and not on preceding actions or states). Formally a MDP is defined via its state set *S*, action set *A*, transition probability matrices *P*, and reward matrices *R*. On executing action *a* in state *s* the probability of transitioning to state $\acute{s}$ is denoted $P^a(s\acute{s})$ and the expected reward associated with that transition is denoted $R^a(s\acute{s})$. A rule for choosing actions is called a *policy*. Formally it is a mapping $\pi$ from the set of states *S* to the set of actions *A*. If an agent follows a fixed policy, then over many trials, it will receive an average total reward known as the *value* of the policy. In addition to computing the value of a policy averaged over all trials, we can also compute the value of a policy when it is executed starting in a particular states s. This is denoted $V^\pi(s)$ and it is the expected cumulative reward of executing policy $\pi$ starting in state s. This can be written as

$$V^\pi(s) = E[r_{t+1} + r_{t+2}...|s_t = s, \pi]$$

where $r_t$ is the reward received at time t, $s_t$ is the state at time t, and the expectation is taken over the stochastic results of the agent's actions.

For any MDP, there exists one or more optimal policies which we will denote by $\pi*$ that maximize the expected value of the policy. All of these policies share the same optimal value function, written as $V^*$ The optimal value function satisfies the Bellman equations (Bertsekas & Tsitsiklis 1996):

$$V^*(s) = \max_a \Sigma_{\acute{s}} P(\acute{s}|s,a)[R(\acute{s}|s,a) + V^*(\acute{s})]$$

where $V^*(\acute{s})$ is the value of the resulting state $\acute{s}$.

The MDP generated from the MetaAlternatives task structure is defined as follows: state in the MDP representation is a vector which represents the TÆMS methods that have been executed in order to reach that state along with their execution characteristics (quality and duration). The MDP action set is the set of TÆMS methods (executable leaf nodes). MDP actions have outcomes and each outcome is characterized by a 2-tuple consisting of discrete quality and duration values obtained from the expected performance distribution of the MDP action. The transition probabilities are obtained from the probability distributions of the corresponding MDP action as described in (Raja, Lesser, & Wagner 2000). The rewards are computed by applying a complex criteria evaluation function of the quality, cost and duration values obtained by the terminal state. The output from the MDP Solver will be an optimal policy that solves the MDP . Once the optimal policy is obtained, Metamod will determine the current state of the agent using the Context Handler and the action corresponding to the current state is obtained from the optimal policy. When the action completes execution, Metamod will be notified; it will then recompute the current state and determine the current best action. This process continues until a terminal state is reached in the MDP or a new problem arrives that requires Metamod's attention.

## Preliminary Results

In this section, we describe our preliminary efforts towards evaluating the meta-level control mechanisms described in this paper. The experiments were performed in a distributed simulation environment called GMASS which was developed by Global Infotech Inc. (GITI) for the purpose of evaluating technologies built for the Coordinators program.

Agents are initiated in GMASS with a local view of problem solving and have access to the initial schedule of actions. The simulation time is mapped to the wall clock time by a parameter that is controlled by the evaluator. Each scenario also has a certain amount of simulation ticks allocated for deliberation. Each agent would use this initial allocation of deliberation time to determine its course of domain-level actions and begins execution of domain activities at the end of the deliberation period.

The solvers that we used for this evaluation are four variations of the current implementation of the DTC/GPGP (Phelps & Rye 2006) coordination mechanism. This implementation of the GPGP protocol makes use of an abstraction-based domain-independent context management technique. The heuristic scheduler (DTC) can be run in simple (DTC1) versus complex (DTC2) modes, where the schedule search process is defined by parameters and is more extensive in the latter case. Similarly the coordination component (GPGP) can be run in NoNegotiation versus ProposeCommitments mode, where GPGP in the NoNegotiation mode optimistically assumes commitments in the current schedule can be established as needed at run time. In the ProposeCommitment mode, GPGP executes a negotiation algorithm to establish commitments during the deliberation process. The four solvers used in the evaluation are the cross product of the various modes of DTC and GPGP and are called DTC1/NoNegotiation, DTC1/ProposeCommitments, DTC2/NoNegotiation and DTC2/ProposeCommitments.

Combinations of the following parameters were used to generate 100 problem scenarios: Number of problems = 1 to 3; Overlap = Small to Medium; Number of fallbacks = 0 to 5 and QAFs at the highest level were sum and min. Each scenario allocated 100 ticks of deliberation time to the agent. We trained our system on 75 of the 100 scenarios and used the remaining 25 scenarios as test cases. The classification process took 10-15ms on average for each problem.

We first compare the performance of the four solvers on the training scenarios. Figure 5 compares the qualities obtained by running the scenarios through each of the 4 solvers. The quality value for each problem class is the average of the qualities obtained from the scenarios belonging to that class. We can see that the best solver modes, i.e. solver mode resulting in highest quality, varies with problem classes. This shows that no single solver mode can be considered the best option for all problem classes, thus motivating the need for the dynamic decision making afforded by the Metamod component.

We now describe experiments to evaluate the accuracy of the Naive Bayes classifier in predicting the performance characteristics of new scenarios. The classifier for each solver mode was trained using the 75 training instances. The quality of the scenarios from the test set as predicted by the classifiers were then recorded (PredictedQuality). In addition, the quality upper bound of each scenario in the test set was computed (TaskQualityUpperBound) and the actual quality obtained by running the solver on each scenario was also recorded (ActualQuality). The classification error is defined as follows:

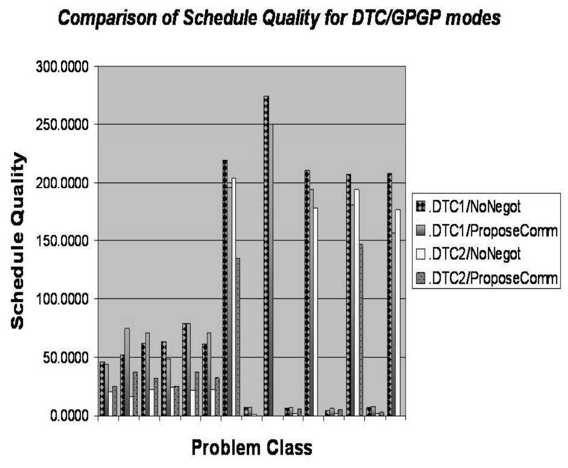

Figure 5: Comparison of four solver modes

$$\frac{(PredictedQuality - ActualQuality)}{TaskQualityUpperBound}$$

Figure 6 shows the classification error averaged over the scenarios belonging to each problem class. It can be observed that the predicted expected quality values by the problem classification component were within 10% of actual quality values in most cases. This shows that MetaMod predicts qualities of these scenarios with a high rate of accuracy.

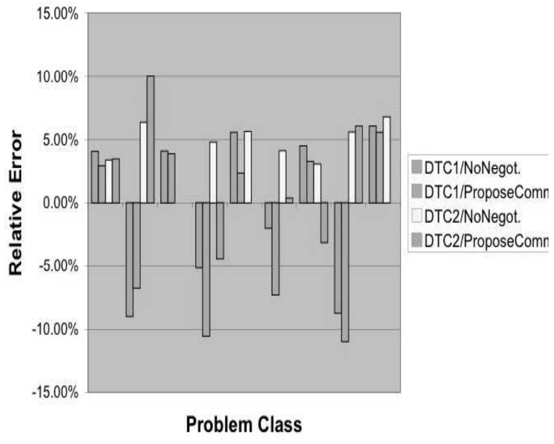

Figure 6: Accuracy of MetaMod's predictions

We are currently testing the effectiveness of meta-cognition for various problem classes by comparing the results to approaches that don't use meta-cognition. In other words, we would compare the performance of the agent that uses Metamod to the performance of the agent when it uses deterministic approaches to handle new problems, for e.g. when the solver with the best expected quality (independent of duration) is always chosen or the solver with the lowest duration is always chosen for every new problem. The goal of this empirical study is to identify the scenarios for which meta-cognition is advantageous and scenarios where meta-cognition does not produce any obvious advantage. This would give us a deeper understanding of the relationship between meta-cognition and the scenario characteristics in the context of the Coordinators program.

## Conclusion and Future Work

We have presented a single agent meta-cognition framework that can make decisions about whether to process a new problem and if so, how to do it and how much time to allocate for processing the problem. This framework leverages a problem classification approach that predicts the expected performance of the new problem based on the performance characteristics of problems encountered before. This approach makes no a priori assumptions about the nature of the new problem and our hypothesis is that the effectiveness of the approach improves incrementally with experience. We have currently implemented the Metamod framework, including the problem classifier sub-component and the MDP sub-component and have provided preliminary results using various modes of the DTC/GPGP solvers in the context of the Coordinators program.

Our next step is to extend the MetaMod reasoning process to include other deliberative solvers (including one based on MDPs ) in its decision process. This would give us greater understanding about the role of MetaMod in switching control from one deliberative solver to another as problem solving proceeds. We also plan to implement an online learning process such that runtime performance information can be part of a feedback loop that will assist Metamod improve its performance in realtime as it gains more experience. We also plan to study the role of meta-cognition from a multi-agent perspective. Specifically we are considering the advantages and costs of having meta-cognition modules in the multiple Coordinator agents collaborate to enhance problem solving We hope that the problem abstraction and learning techniques developed in this project will eventually contribute to a generalized framework for meta-cognition in agents operating in resource-bounded multi-agent environments.

## Acknowledgement

We would like to thank Dr. David Musliner for discussions on the design of the meta-cognition component in the context of the Coordinators agent. We also thank the two anonymous reviewers for their helpful comments. We also acknowledge the contributions of John Stamper in the initial design and implementation of the MDP sub-component. This material is based upon work supported by the DARPA/IPTO COORDINATORs program and the Air Force Research Laboratory under Contract No. FA87550-05-c-0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

## References

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Decker, K., and Garvey, A. 2005. Scenario generation: Automatically generating domain-independent coordination scenarios. Unpublished.

Decker, K. S., and Lesser, V. R. 1993. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management* 2(4):215–234.

Domingos, P., and Pazzani, M. J. 1996. Beyond independence: conditions for the optimality of the simple bayesian classifier. In *Proceedings of the Thirteenth International Conference on Machine Learning*, 105–112. Morgan Kaufmann.

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

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

Horvitz, E. 1989. Rational metareasoning and compilation for optimizing decisions under bounded resources. In *Proceedings of Computational Intelligence*.

Mitchell, T. M. 1997. *Machine Learning*. WCB/McGraw-Hill.

Phelps, J., and Rye, J. 2006. Gpgp - a domain-independent implementation. In *Working Notes of the AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*.

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

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc.

Raja, A., and Lesser, V. 2004. Meta-level Reasoning in Deliberative Agents. *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)* 141–147.

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

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

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