The Role of Problem Classification in Online Meta-Cognition

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

Abstract

Agents operating in open environments must be able to adapt their processing to available resources, deadlines, their goal criteria, and their current problem solving contexts. This paper describes the role of meta-cognition in this process; in particular, we define a meta-cognition framework that uses Naive Bayesian classification of the agent's current context in order to represent the meta-level control problem as a Markov Decision Process.

1 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. 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 [4, 5, 11, 13] capabilities to ensure good performance in open environments.

We make the following assumptions in our work. The agents operate in a cooperative environment and can pursue multiple high-level goals or *problems* simultaneously. A *scenario* is a set of problems assigned to multiple agents. 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, e.g., a 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[1] program using Bayesian classification techniques [6] and Markov Decision Processes (MDPs) [9]. 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 should be recalled (rescheduling, replanning, or re-coordination), and if it is, determine the best mode to use.

In the COORDINATORS domain, it is possible for two problems to overlap; they have to be reasoned about and executed in overlapping time intervals, resulting in resource contention. These resources could include processor time for individual modules within an agent as well as availability of other agents. In many cases, processing for both problems may not be feasible. Meta-cognition would then be required to choose which problem to work on. With no problem overlap, meta-cognition could determine the best combination of module problem solving settings and time allocations for each problem independently; however, even this relatively simple sequential presentation of problems poses difficulties 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. In this paper, we present a meta-cognition frame-

[1] http://www.darpa.mil/ipto/programs/coordinators/

work that will allow agents to efficiently handle situations with both overlapping and non-overlapping problems.

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 initial 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.

2 The COORDINATORS Application Domain

COORDINATORS are intelligent agents that address the problem of effective coordination of distributed human activities. They work with each other and assist their human counterparts to dynamically adapt their plans to environmental changes in order to achieve team goals. 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) [3] 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. Each agent has access only to its local view of the task model. Within a problem, tasks are grouped under windows defined by earliest start time (EST) and deadline (DL). Figure 1 shows a simple 2-agent COORDINATORS task structure. The unshaded methods belong to Agent1, and the shaded methods belong to Agent2. The task structure describes the subset of the global C-TÆMS model consisting of those parts of the problem that affect and are affected by the local agent. C-TÆMS models have nodes representing complex actions, called *tasks* and primitive actions, called *methods*. *Get in Position* is an example of a task and *Method A1* is an example of a method in Figure 1. Methods are owned by agents and may have duration distributions, release times (earliest start times) and deadlines. Methods may have uncertainty in their outcomes which is statistically characterized using a discrete model in two dimensions: quality (Q) and duration (D). For example, "Q:60% 10, 40% 8" means that, statistically, the method will complete with a quality of 10 60% of the time and a quality of 8 40% of the time. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to overall problem solving. Method duration describes the amount of time for the agent to execute the modeled action. 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. In this domain, each scenario consists of one or more concurrent problems linked by a sum QAF.

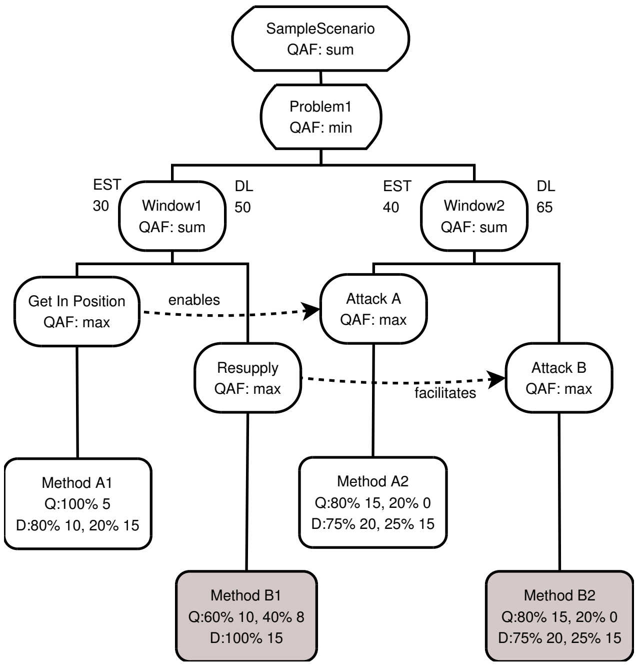

Figure 1. A sample COORDINATORS task

The parameters used to generate scenarios include the number of windows, window tightness, window overlap, number of fallbacks (alternate methods for a given task), NLE loops (chains of agents which affect each other), etc. The complete list of parameters can be found in [2]. In addition to the stochastic nature of method execution, it is possible for new goals to arrive or existing goals to be modified (e.g., modified deadlines or modified performance characteristics of methods) at runtime.

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) which we describe in this paper basically assigns processor time to each of the other modules for each problem in a scenario, as well as determining in which of several operating modes the modules should run. To achieve these goals, our agents use Naive Bayesian Classification to predict the performance of various solver modes applied to incoming problems. They then use this predicted performance to construct a TÆMS task structure representing the available meta-level control decisions. This task structure is converted into an MDP and solved to obtain a policy dictating the optimal allocation of processor time and solver modes for each problem in the scenario. The main contribution of this paper is the use of problem classification and the MDP formalism at the meta-level to handle the inherent uncertainty and sequential nature of our problem domain.

There has been important previous work in meta-cognition. Russell and Wefald [13] 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. Schut and Wooldridge [15] have independently observed that a Markov Decision Process-based model towards decision making is most similar to the bounded optimality model. Russell, Subramanian, and Parr [14] cast the problem of creating resource-bounded rational agents as a search for the best program that an agent can execute. 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. Our approach to meta-level control involves construction of agents similar to these bounded optimal agents. 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, it accounts for computational resources and takes advantage of the Markov Decision model to bound computation and handles multiple inter-dependent meta-level questions. This work extends the meta-level control architecture described in the context of multi-agent systems [11]. 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 based on actual performance of previously seen training problems.

3 Meta-Cognition Framework

Prior to describing our meta-cognition framework, we define several key terms used in the rest of this paper:

scheduling mode: a particular configuration of parameters for a scheduling algorithm. (Our discussion focuses on alternate modes for scheduling; however, the approach discussed is applicable to reasoning about alternate modes for other deliberative actions like coordination and planning, provided suitable performance measures are defined for these actions.)

performance profile: a triple $\{$*solver mode*, *expected quality (EQ)*, and *expected scheduler duration (ED)*$\}$, that describes the expected performance characteristics of applying the various scheduling modes to a problem, in the form of statistical distributions. 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 executing the schedule.

performance type: used to describe problems sharing similar performance profiles. For example, problems for which a particular scheduler mode yields high EQ in a short amount of time (low ED). We create these categories ahead of time; however, in principle, clustering algorithms could be used to "discover" performance types after running some initial experiments.

problem feature: any of several problem characteristics that may affect performance, obtained by examining the TÆMS structure of a problem and used to predict performance type. Examples include number of windows, number of NLEs, etc.

problem class: used to describe problems with a particular configuration of problem features. Problems in the same problem class should fall into the same performance type, but a single performance type may describe problems with widely varying features.

Figure 2 gives a high-level view of the control flow within the MetaMod component. 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 predefined 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). The MetaAlternatives task structure is an abstract representation that captures the agent's end-to-end problem solving choices from the meta-cognitive point of view. Details of the problem classification step are described in Section 3.1 below. The TÆMS task structure is then translated into a Markov Decision Process. This is accomplished by passing the task structure to the MDP sub-component which consists of the MDP Generator and MDP Solver. Details of this sub-component are described in Section 3.3 below. The MDP is evaluated to determine the best action to be taken by MetaMod given the current environmental state (Step 4 in Figure 2). MetaMod will determine the current state of the

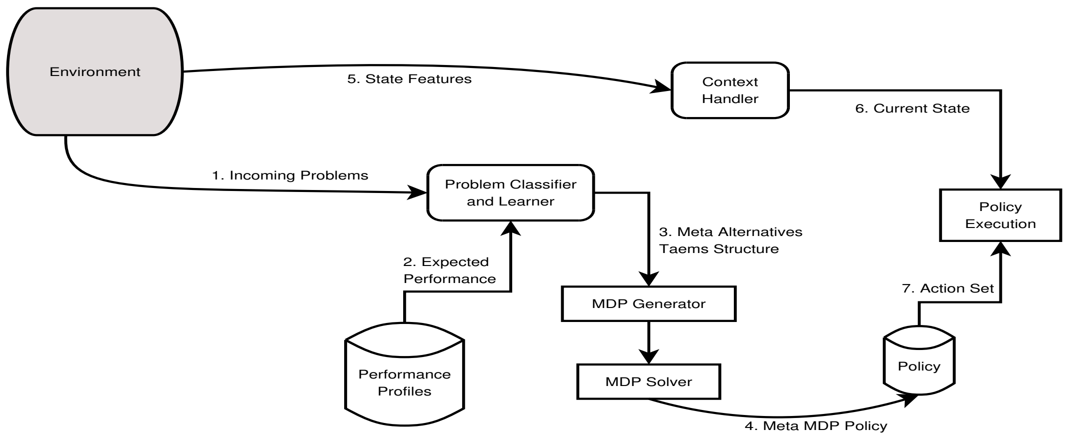

Figure 2. Control-flow in the Meta-Cognition Module (MetaMod)

agent using the Context Handler (Step 5). The best action corresponding to the current state is obtained from the optimal policy by the Policy Execution sub-component (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.

In determining the performance profiles, we wish to avoid making assumptions about the ranges of quantitative performance characteristics 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. Compute a quality upper bound for each problem by assigning to tasks the mean quality of their highest quality methods and propagating quality upwards through the task structure.

2. Represent 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.

3. Multiply 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.

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

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

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}.

3.1 Problem Classification

We define performance types based on the expected performance of the problems in a particular mode, there are NxM possible performance types (N scheduling modes times M types per mode, where M depends on the granularity with which we wish to predict EQ and ED). 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, SchedulerC, etc.). We determine performance types using Naive Bayes classification.[2]This ma-

[2]other classification methods (such as C4.5 [10]) 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.

chine 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:

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

Avg. 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.

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

Avg. problem overlap: This is computed for problems in a similar way to how window overlap is computed.

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

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

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

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 reassigns the performance type corresponding to the scheduling mode we used and uses the problem features and the reassigned 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 S with two problems, P1 and P2, with respective quality upper bounds 100 and 150. We run the Bayesian classifiers (*SchedulerA_Classify*, *SchedulerB_Classify*, and *SchedulerC_Classify*) on each of the problems and obtain the results shown in Table 1. We now look up the performance profile information for each of the performance types that result from the above classification process. Table 2 shows an excerpt of an example performance profile listing (A_9H indicates SchedulerA would produce a schedule achieving 90% of the problem's quality upper bound, but the expected scheduling time would be High). We will continue to use this example in the next section.

Table 1. Example classification results using three different schedulers

	SchedulerA Classify	SchedulerB Classify	SchedulerC Classify
P1	A_9H	B_6L	C_2L
P2	A_6M	B_6L	C_6M

Table 2. Example performance profile information

Performance Type	Scheduler Name	EQ	ED
A_6M	SchedulerA	60%	Medium
A_9H	SchedulerA	90%	High
B_6L	SchedulerB	60%	Low
C_2L	SchedulerC	20%	Low
C_6M	SchedulerC	60%	Medium

3.2 Building the MetaAlternatives Task Structure

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

Suppose our earlier example scenario S consisting of the problems P1 and P2 has been assigned to an agent. For each problem P_i, we construct a subtask ($ModuleAction_i$) that

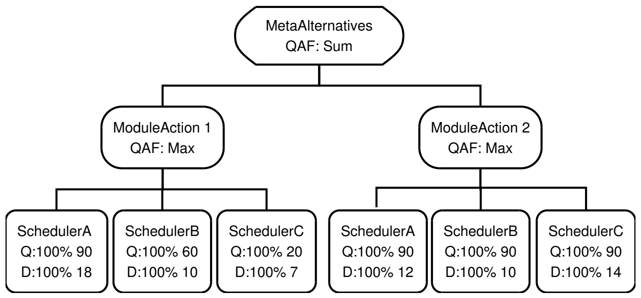

Figure 3. MetaAlternatives before MDP policy execution begins

uses a Max QAF because our agent will accumulate quality only from the schedule that it actually executes. The methods used to accomplish this task correspond to the scheduler modes, for example SchedulerA, SchedulerB, SchedulerC. 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. The resulting task structure is shown in Figure 3. 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.

3.3 Meta-Cognition Decision Process

A Markov Decision Process 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 [16]. 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 *s'* is denoted $P^a(ss')$ and the expected reward associated with that transition is denoted $R^a(ss')$. A rule for choosing actions is called a *policy*. Formally, it is a mapping π 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 π 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 π^* 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 [1]:

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

where $V^*(s')$ is the value of the resulting state s'.

The process of generating an MDP from the MetaAlternatives task structure is based on the TÆMS to MDP translation algorithm in [12]. The resulting MDP 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 [12]. 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.

4 Experiments

In this section, we describe our 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 [8] coordination mechanism. The heuristic scheduler (DTC) can be run in simple (DTC1) versus complex (DTC2) modes, where the schedule search process 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 combinations 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; Window 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 4). The quality value for each problem class is the average of the qualities obtained from the scenarios belonging to that class. The semantics of the sum and min QAFs account for the large difference in schedule quality among problem classes. The solver mode resulting in highest quality varies across problem classes; thus no single solver mode can be considered the best option for all problem classes, motivating the need for the dynamic decision making afforded by the MetaMod component.

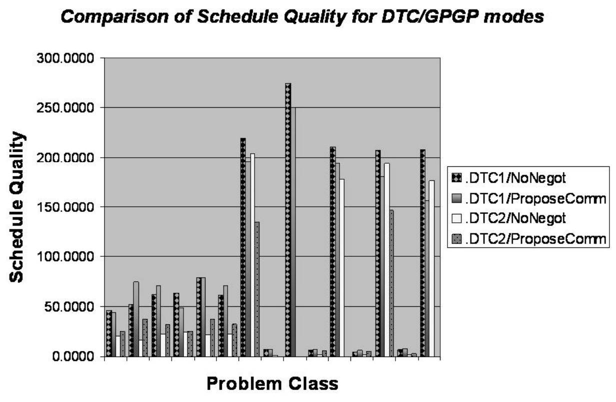

Figure 4. MetaMod is necessary, because no single solver is the best for all problems.

We now describe experiments to evaluate the accuracy of the Naive Bayesian 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

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

Figure 5 shows the classification error averaged over the scenarios belonging to each problem class. On average, the schedule quality predicted by the problem classifier was within 10% of the actual schedule quality in most cases, indicating that MetaMod was accurate in its classification of these scenarios.

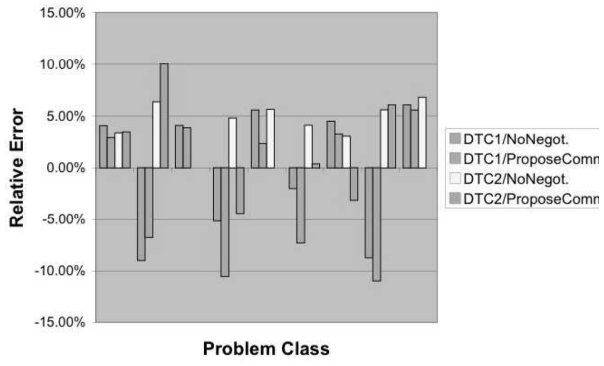

Figure 5. MetaMod's predictions are accurate.

Figure 6 compares the predicted schedule quality of each solver mode alone with the quality of the MDP produced by MetaMod. Since MetaMod reasons about the possibility of a scheduler failing or returning a poor quality schedule, necessitating another scheduler to be called within the agent's remaining deliberation time, the MDP quality is always higher than the predicted quality of any single solver mode.

5 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 in a multi-agent context. This framework leverages a problem classification approach that predicts the expected performance of the new

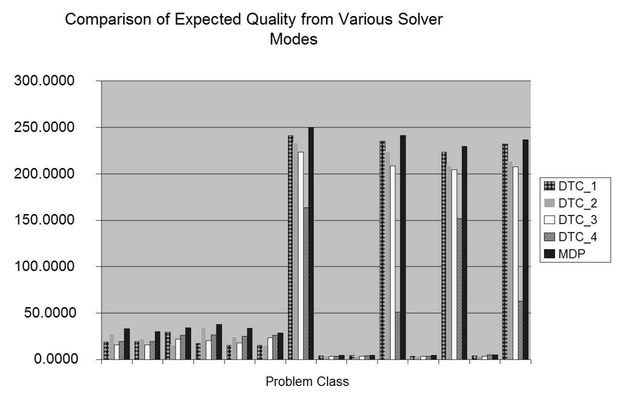

Figure 6. MetaMod's sequential decision making outperforms any single solver over multiple contexts.

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 and have provided 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 work with an MDP-based scheduler/coordination component [7]. We also plan to study coordinated meta-level control to model the interactions between solver modes and constraints of multiple agents. 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.

6 Acknowledgement

We would like to thank Dr. David Musliner, Phani Burra, and V. Mappillai for discussions on the design of the meta-cognition component in the context of the COORDINATORS agent. This material is based upon work supported by the DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA8750–05–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

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

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

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

[4] R. Goldman, D. Musliner, and K. Krebsbach. Managing online self-adaptation in real-time environments. In *Lecture Notes in Computer Science*, volume 2614, pages 6–23. Springer-Verlag, 2003.

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

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

[7] D. J. Musliner, E. H. Durfee, J. Wu, D. A. Dolgov, R. P. Goldman, and M. S. Boddy. Coordinated plan management using multiagent MDPs. In *Working Notes of the AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*, pages 73–80, March 2006.

[8] J. Phelps and J. Rye. GPGP - a domain-independent implementation. In *Working Notes of the AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*, pages 81–88, March 2006.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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 338–345, 1993.

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

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