An Examination of Criticality-Sensitive Approaches to Coordination

Pedro Szekely, Rajiv T. Maheswaran, Robert Neches, Craig M. Rogers and **Romeo Sanchez**
University of Southern California, Information Sciences Institute
4676 Admiralty Way, Suite 1001, Marina Del Rey, CA 90292

Marcel Becker and **Stephen Fitzpatrick**
Kestrel Institute
3260 Hillview Avenue, Palo Alto, CA 94304

Gergely Gati, David Hanak, Gabor Karsai and **Chris van Buskirk**
Vanderbilt University, Institute for Software Integrated Systems,
Box 1829, Station B, Nashville, TN 37235

Abstract

In this work, we address the problem of coordinating the distributed execution of plans and schedules by multiple agents subject to a number of different execution uncertainties. The coordination of multi-agent teams in uncertain, dynamic domains is a challenging problem requiring the fusion of techniques from many disciplines. We describe an approach based on the dynamic and selective use of a family of different problem-solving strategies that combine stochastic state estimation with repair-based and heuristic-guided planning and scheduling techniques. This approach is implemented as a cognitive problem-solving architecture that combines (i) a deliberative scheduler, which performs partially-centralized solution repair, (ii) an opportunistic scheduler, which locally optimizes resource utilization for plan enhancement, and (iii) a downgrader, which proactively guides the execution into regions of higher likelihood of success. This paper characterizes the complexity of the problem through examples and experiments, and discusses the advantages and effectiveness of the implemented solution.

Introduction

This work addresses the collaborative and coordinated execution of activities of a multi-agent team. Joint operations in military settings, project/personnel management in global enterprise settings, and multiple-rover/UAV missions in science-discovery or search-and-rescue settings are some examples of complex and dynamic execution environments where effective and efficient coordination is crucial to the successful achievement of a set of pre-defined mission goals. The complexity of the problem is due to (i) uncertainties associated with the execution environment, which causes an explosion on the number of system states for which courses of action need to be determined, (ii) dynamism, meaning that parameters that define the initial models of uncertainty and constraints may change in the execution phase, thus requiring online reasoning and decisions (iii) partial observability of global goals and other agents' plans, where agents' local policies may not align towards a coherent global strategy, and (iv) distributed decision-making authority, which may cause conflicts and delays on the selection of the appropriate course of action.

A centralized problem-solver eliminates the issues of partial observability and conflicting authority, but puts a high computational burden on a single agent. When bounded rationality is considered, this agent cannot make decisions in a timely manner. While we do not address communication failures or delay in this study, centralization will suffer when these extensions are added. Traditional AI methods that can model the distributed nature of the problem such as SAT or DCR techniques (DCSP, DCOP) cannot currently handle uncertainty without cumbersome encodings. Standard OR methods such as mathematical programming or decision-theoretical approaches are built to handle uncertainty yet they cannot be immediately utilized for our problem without addressing the partial-observability and multi-agent decision-making issues. Developing techniques that handle all the aspects of these settings is an emerging area of research. This paper takes a step towards that goal while also addressing the reasoning limitations that are brought about by scale, dynamism and bounded rationality which require manageable state spaces and quick reasoning.

We combine various elements of these techniques to construct a hybrid-scheme with multiple reasoning components. To address uncertainty, each agent constructs local estimates of the likelihood of success for its potential activities, and propagates them to relevant agents whose activities may be affected. These likelihoods are utilized to calculate the importance of various activities. We construct a local state as the union of non-probabilistic classification and a low-

dimensional probabilistic state vector denoted as a *profile*. The reasoning is performed by three components: (1) the deliberative scheduler, (2) the opportunistic scheduler, and (3) the downgrader. The deliberative scheduler performs *dynamic partial-centralization* of the relevant agents when a problem is detected in some substructure of the team reward. It then extracts potential solutions and proposes modifications to remedy the problem. Due to the complexity of this task and the nature of the anticipated failure, this process occurs over several decision epochs with decision windows that begins slightly in the future. The opportunistic scheduler performs schedule modifications in the near-term. Using only local information, it is capable of making decisions on a faster time-scale, and attempts to increase the quality and probability of success by utilizing free resources. The downgrader complements the previous two components by freeing resources both in the near-term and the future based on the evolution of the system up to the present.

Model

We model our problem with CTAEMS, which is an adaptation of the TAEMS formulation (Decker & Lesser 1993) representing distributed multi-agent coordination problems. In this formulation, each agent has the potential to execute a set of activities or *methods* over a finite time horizon broken up into decision epochs. Once an agent starts a method, it can either abort it or wait for its completion in order to start another method. Each method produces some quality and takes a duration. These quantities are drawn from probability distributions. Failure occurs when a method yields zero quality. A failure is modeled when quality zero is listed in its probability distribution with non-zero probability. Otherwise, it is an unmodeled failure. Each method belongs to a *window* that prescribes the earliest time at which it may begin and the latest time at which it may complete in order to obtain positive quality.

The qualities achieved by the executed methods are aggregated through a tree of quality accumulation functions (QAFs) to yield the reward for the multi-agent team. This tree consists of a hierarchy of nodes, where each leaf node represents a method associated to a single agent that can execute it. The non-leaf nodes are *tasks* with associated QAFs that define their quality as a function of the qualities of the children. The children of a task can be methods or tasks. This hierarchy defines how the quality of the root node, the team reward, is computed from the individual methods that agents execute.

The QAFs include: (i) MIN, which yields the minimum value of qualities of the children, used to model situations where all children must succeed in order for the parent to accumulate quality. (ii) MAX, which yields the maximum value of qualities of the children, used to model situations where at least one of the children must achieve positive quality. (iii) SUM, which adds the qualities of the children, also used to model situations where some children must succeed. (iv) SyncSUM, which adds the qualities of the children whose start time is the earliest start time of all children. This encourages child nodes to align their execution to begin at the same time.

An additional complexity is the possibility of non-local effects (NLEs) between nodes. An NLE is an *enabling* coupling between nodes, a directed link between a source and a target. Targets started before all sources have accumulated quality will fail and accumulate zero quality regardless of the their probability distributions. If the target is a task, the NLE applies to all descendant nodes.

The challenge for the multi-agent team is to make the appropriate choices of methods to execute that yield the highest quality at the root of the CTAEMS hierarchy. Effective policies must react dynamically to the uncertainties in the durations and qualities of methods in an initial schedule and create new schedules as the sample path unfolds. Reacting to the state of the system is made more difficult by the fact that agents cannot see the entire CTAEMS hierarchy. Agents have a *subjective view* consisting of the methods they can execute and their direct ancestry to the root node along with nodes that are one-step removed due to NLEs. This models the partial-observability inherent in multi-agent systems where an agent cannot monitor the entire system and often is not aware of certain parts of the team. This aspect occurs to a greater extent in large-scale settings.

Example

In order to illustrate our model, and part of the complexity of the problem, we discuss an example in this section. The CTAEMS structure for our example can be seen in Figure 1.

This problem consists of three phases (or *windows*), denoted as *setup*, *deploy* and *engage* modeled using SUM QAFs. Each window has two different MAX tasks, each with two children methods. The problem has two agents, *alpha* and *bravo*. Each method belongs either to plan *A* or plan *B*. The root, S_1, is a MIN QAF, requiring each phase (window) to produce quality in order for the overall problem to succeed. Figure 1 shows the *enabling* relationships among different methods. For example, method $p^{1a}A$ enables method $p^{2a}A$, which implies that $p^{1a}A$ must execute successfully before method $p^{2a}A$ is worth starting.

The layout of Figure 2 illustrates some of the complexities of the problem. For simplicity, each method has been labeled with the information regarding the schedule where it participates, and the agent who executes it. Bars correspond to the probability distribution for the durations of each method, here having three outcomes. Plan *A* is the initial schedule, which leads to a higher accumulated quality but it is somewhat risky, since its activities may take longer. On the other hand, plan *B* produces less quality but it is safer. Note that uncertainty in the method durations can lead to failures. Furthermore, *enabling* dependencies increase the probability of failure for future phases of the problem. We can see an example of such failing conditions in Figure 3. Here, Plan *A* gets delayed given that the *setup* phase activities take their maximum duration. Consequently, agent $alpha$ cannot complete its method by the deadline during the *deploy* phase. The failure in the *deploy* phase along with other *enabling* conditions preclude the activities in the final *engage* phase of the problem which prevents the team from accumulating quality.

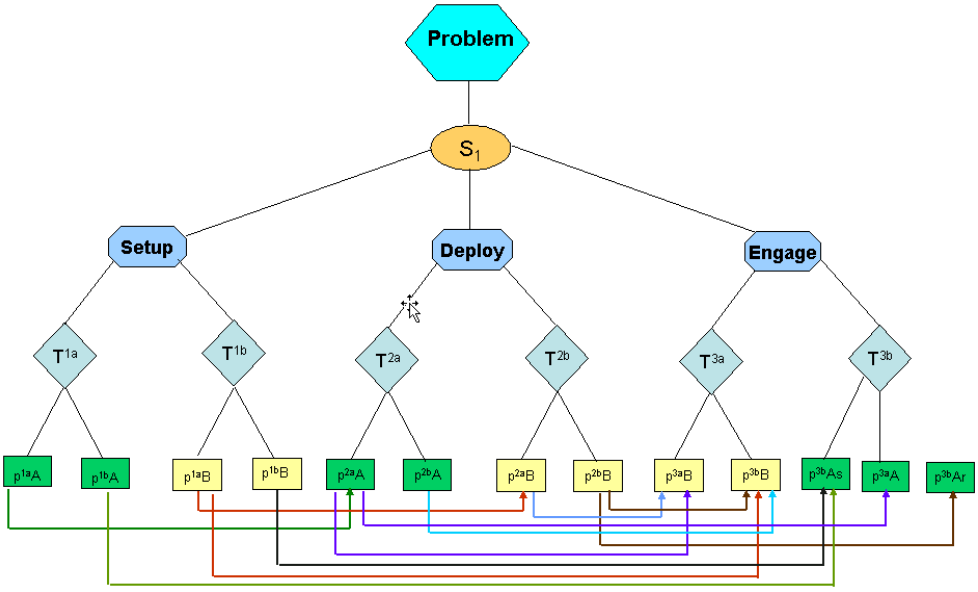

Figure 1: CTAEMS Example Structure

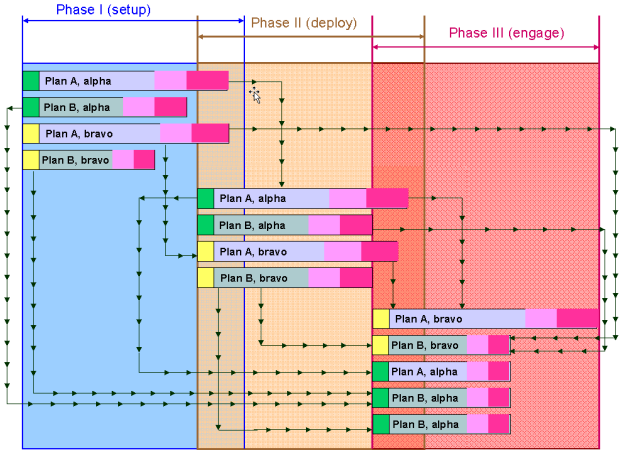

Figure 2: Time Uncertainty in Example

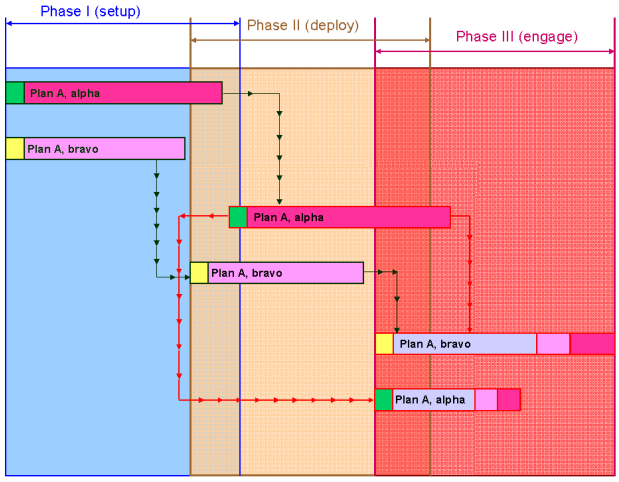

Figure 3: Failing Execution Trace

Our system can predict failures, and trigger timely schedule repairs. Determining when to act in uncertain and distributed domains is inherently complex. The general idea is to detect potential problems in the schedule early, in order to have enough lead time to schedule and execute the enablers of methods needed to repair problems later on. Consider again Figure 3. Once the method in the *setup* window hits its maximum duration, the probability that the *alpha* agent completes its method on time during the *deploy* phase decreases (i.e, only two outcomes will allow completion before the deadline). Note that this has a cascading effect, since the probabilities of success for the activities in the *engage* phase depend on the success of activities in the *deploy* phase. A solution to our example is to anticipate the potential failure in plan *A* and switch to plan *B* during the second phase as illustrated in Figure 4.

In the following sections, we present our approach for dynamically detecting and repairing potential failures on the schedule based on the probability of success of each method, and its importance with respect to the success of the overall plan.

Approach

Our approach involves the selective combination of different problem solving strategies, including partially-centralized solution repair (*deliberative scheduler*), locally optimized resource allocation (*opportunistic scheduler*) and proactive re-prioritization (*downgrader*). These higher-level reasoning components are triggered and supported by a *state manager* which holds *schedules* and *profiles* and an *execution controller* which interacts with the environment and trans-

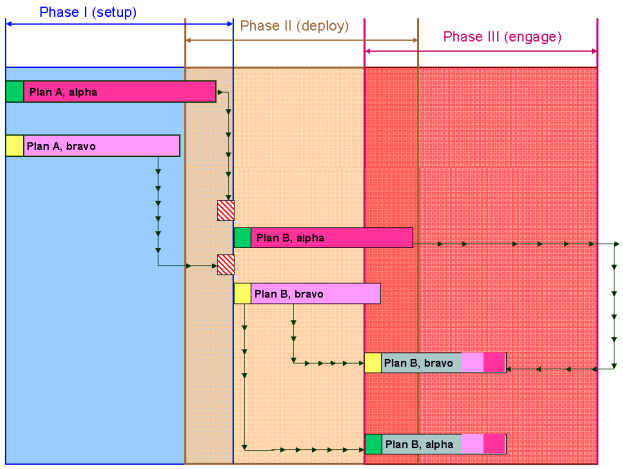

Figure 4: Execution Trace of Plan B

lates our strategies into actions. The architecture of the CSC coordinator can be seen in Figure 5.

State Manager and Execution Controller

State Manager The purpose of the state manager is to give the reasoning components the information they need to act. There are essentially two types of such information: *probabilities* and *importance*, denoted by p and α respectively in Figure 5. Both types of information are kept about the *current schedule* and about *potential schedules* that the agents may engage. The probabilities about the current schedule enable agents to determine whether they should change the current schedule. The importance allows them to determine the marginal contribution that any individual method has on the success probability of the current schedule. The potential probabilities allow agents to reason about the probability improvements that could be achieved by engaging new options. The potential importance allows agents to determine whether methods can still contribute to the overall quality of the schedule.

The probability and importance of current and potential schedules are global quantities that, in general, depend on the local state of all agents. We have developed distributed algorithms that compute approximations of these values based on local information and approximations received from other agents. The state manager uses these algorithms and the protocols to share local information with other agents.

Schedule The system stores the current schedule in a distributed manner as part of the state graphs maintained by the State Manager instance on each agent. Every *method* has an associated *scheduled start window* and a *priority*. At the beginning of a simulation, these are determined based on the *initial schedule* contained in the initial subjective view of an agent. At runtime, the *downgrader*, the *opportunistic* and *deliberative schedulers* alter the current schedule by modifying the start windows and priorities of scheduled methods or by adding new methods to the schedule. The first two schedulers operate only on local methods while the deliberative scheduler can create a set of schedule changes that are installed on remote agents. To reduce the potential conflicts during remote schedule updates running in parallel, a locking mechanism is in place that serializes the distributed installation of schedules.

Profiles Profiles are designed to reason about uncertainty, and contain (i) a pair representing probability and busy-cycles of a CTAEMS node and (ii) an importance value. These measures are used to evaluate the current and potential schedules. For the current schedule, probabilities for methods are calculated using information about the distributions of durations and their execution windows. In addition, we factor in the probabilities of enabling methods. The busy-cycles are a resource consumption measure also calculated from the duration distributions. These pairs are propagated throughout the system via nearest neighbor communication. Using these probabilities, we can calculate the importance of each node to the overall system performance based on the importance values in the local view of each node, again in a distributed manner. The potential profiles consist of a single pair for the methods denoting the likelihood and costs independent of the current schedule and a set of pairs for higher nodes, which represent potential solutions for achieving positive quality at that node. We use these profiles to obtain potential importance, which determines if a method or node can still contribute to helping the team goal succeed. As in the case of the scheduled profiles and importance, the potential measures are propagated in a distributed manner through communication to relevant neighbors.

Execution Controller The execution controller manages the flow of data between the agent and the underlying execution or simulation environment, including inter-agent messaging. It converts execution environment events, such as method start and stop, into a platform-neutral representation for portability. It runs at a higher priority level than the rest of the agent and avoids priority inversions when accessing shared data, allowing it to continue to execute scheduled methods even when the rest of the agent lags behind the pace of execution environment events.

The execution controller contains its own distributed window-sliding scheduler. A scheduled method's execution will be delayed until its enabling conditions are met, even if some enabling conditions depend upon the state of methods residing in other agents; the necessary inter-agent communications will take place autonomously between execution controller instances. A method will be dropped from the schedule when one or more of its enabling conditions has failed. When a set of methods is to be initiated synchronously on a set of agents, the execution controllers will communicate to ensure that synchronized execution takes place in the execution environment once all enabling conditions have been met.

Higher-Level Reasoning Components

Deliberative Scheduler The CTAEMS task structure provides a hierarchical representation for the entire family of possible schedules for a particular input scenario or prob-

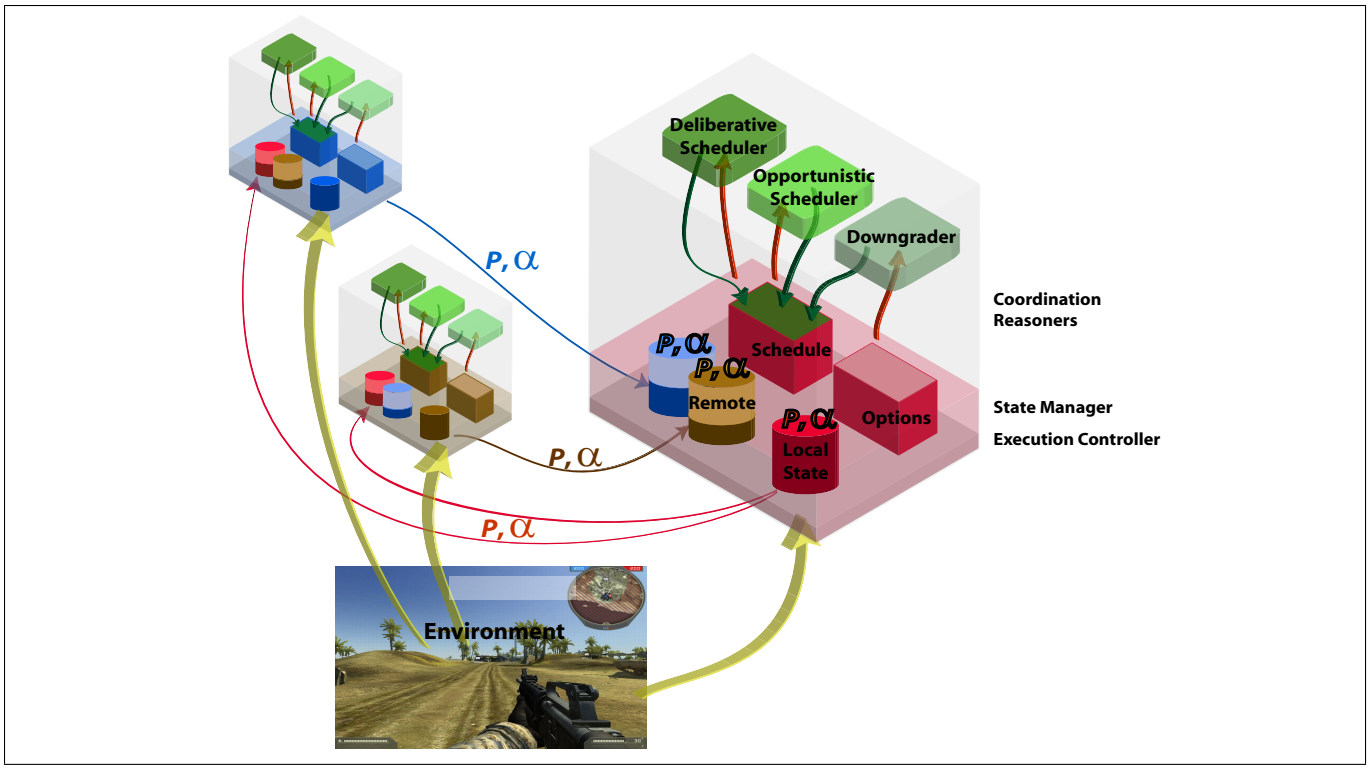

Figure 5: CSC Architecture

lem. The scheduler component is responsible for traversing this hierarchy and selecting the set of *methods*, or activities, that should be executed by the different agents with the goal of maximizing global task quality. Method execution is constrained by *feasibility intervals,* scenario specific restrictions on earliest method start and latest method finish times; *single threadedness,* each agent can only execute one method at a time; *method interactions*, dependencies on the order of execution of certain methods.

The scheduler is implemented as a configurable, heuristic-guided search engine that traverses the task hierarchy defined in the CTAEMS problem description provided as input, selects a set of methods to execute, and assigns start-time windows for methods in this set. The search tree expanded by the search engine follows the topology of the task structure defined in the CTAEMS structure: starting from a pre-defined root node, the algorithm recursively expands the search tree for the children of the selected node. The order in which the children of a particular node are selected, the strategy used to expand the sub-tree, the allocation of agent's time for method execution, and the metric used to select the results for different types of nodes is parameterizable. The cognitive problem solving mechanism uses several different pre-configured search strategies: fast repair-based strategies that only try to fill existing gaps in agent's availability, load balance techniques that try to guarantee minimum amounts of positive quality for certain tasks, robustness enhancement strategies that schedules redundant methods to increase probability of positive quality achievement, and schedule compression techniques combined with some other strategy to reduce agent's idle time and increase expected quality.

The different agents participating in the execution do not have a complete view of the task network. Each agent *owns* a subset of all the methods and tasks, and can only see the constraints and nodes directly affecting the owned node. In response to scheduling decisions, or execution events, each agent perform propagation on its local constraint network. This will cause modifications to both owned and visible nodes. It then sends to other remote agents the changes on its owned nodes. When receiving this information, remote agents update their local views or the other agent's node, and perform constraint propagation to keep the state of their owned nodes consistent. The updated state of owned nodes for each agent is then sent to other agents, until there are not more modifications in the state of owned nodes. If, during propagation, an agent detects conflicts involving one of its owned nodes, this agent is responsible for resolving the conflict. At this point, the agent may then trigger more complex problem-solving components that would require the creating of a cluster of agent to solve the problem in a partially centralized fashion.

Opportunistic Scheduler The opportunistic scheduler is instantiated whenever there is a gap where an agent is not performing a high priority method. At this point, the agent estimates the window of available resource, namely execution time before the next high priority method is scheduled to begin. The agent then may choose to begin executing a

method that is not part of the current schedule. The goal is to choose the best method to execute given the resource restrictions, without harming the existing schedule. Using local knowledge of the reward function and methods scheduled to be executed, the agent dismisses methods that may cause harm should they get aborted by higher priority methods introduced by the deliberative scheduler. Then, the opportunistic scheduler utilizes the probabilities and importance measures in the profile to determine the method with the greatest likelihood of helping the team. This method is inserted with medium priority such that any (existing or future) alterations to the schedule by the deliberative scheduler are not affected. Every agent runs an opportunistic scheduler to maximize the use of its execution capabilities.

Downgrader The downgrader utilizes the potential importance to determine whether methods in the schedule can still contribute to the team goal. The priorities of methods are reduced if it is determined that they no longer can contribute, freeing agent resources for other scheduler components that schedule methods at higher priority. Downgrading gives methods installed by the deliberative scheduler a higher likelihood of succeeding by removing unimportant methods that start earlier, and gives the opportunistic scheduler more chances to be instantiated. The preceding effects serve to enhance the robustness of the system.

Experiments

We have described in previous sections the main components of our system to handle distributed scheduling problems under uncertainty, dynamism, temporal constraints and partial observability. One remaining challenge is to produce a set of experiments that could meaningfully test our system, despite the complexity and quantity of interactions in the problem. Our aim is to introduce a set of problems that share some systematic structure in order to present some preliminary evaluation of our work. In particular, we have designed our problems taking into account two important factors: NLE chains and failure rate. Each factor has been categorized into four subclasses: empty, low, medium and high. Each subclass relates to the number of NLE chains or failures introduced into a particular problem. The entire problem space is then partitioned as a cross product of the factors and their subclasses. Finally, each subspace of problems is built using an increasing degree of window overlap. NLEs and failures are introduced randomly for each problem, all of which have four agents for the purpose of this evaluation.

Our testing platform includes a cluster of distributed 32-bit Intel Linux systems with at least 1.5GB of memory. One of them serves as the simulation controller and the rest are assigned to agents.

Figure 6 displays a plot with our solutions. The plot shows the difference in quality between our base system and our higher-level reasoning enhancements. The data points represent average results over ten trials per problem, under different random seeds. Each problem is provided with an initial schedule, that may fail in accordance to a failure probability distribution. Negative data points illustrate those problems in which our system produces a reduction in quality over the base system. However, we can see that for most of the problems our system consistently returns better quality solutions, anticipating and recovering from failures, and generating alternative schedules.

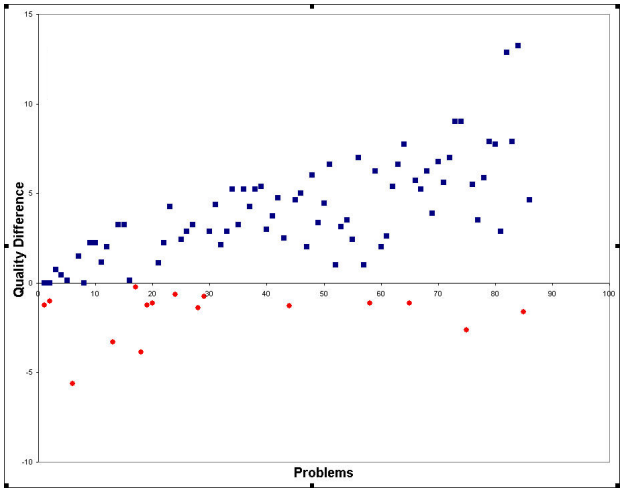

Figure 6: Quality Improvement Over Base System

Related Work

The coordination of multi-agent systems in dynamic, stochastic, temporally-constrained and partially observable domains is a challenging problem. The most general decision-theoretic models for this problem have been proved to be extremely complex (NEXP-complete) (Shen, Becker, & Lesser 2006). In order to lower complexity, some models restrict the types of interactions allowed in the system, or the features they are able to address. One of such models is DEC-MDP (Goldman & Zilberstein 2003; Beynier & Mouaddib 2005). The decentralized MDP model usually bases its computation on local policies, with very little or no communication among agents (Beynier & Mouaddib 2005). Although, our approach also computes local estimates of the probability of success for each activity, such estimates are propagated in a distributed manner to the interacting agents.

Another way of modeling the multi-agent coordination problem is through traditional AI methods. Distributed Constraint Optimization Problems (DCOP) have been adapted to capture the locality of interactions among agents with a small number of neighbors (Cox, Durfee, & Bartold 2005), but it fails to capture the uncertainty factor in the general problem. Network Distributed POMDPs have been proposed to address these issues (Nair *et al.* 2005). Unfortunately, they solve only small problems.

Decision-theoretic planning can also be used to model our problem (Littman, Goldsmith, & Mundhenk 1998). In this model, execution monitoring of the system and replanning are very important. Conditional plans are generated in order to deal with contingencies during execution. Replanning is invoked when an agent identifies unsatisfied,

or likely to fail conditions. However, the requirements for a rich model for actions and time are generally problematic for planning techniques based on MDP, POMDP, SAT, CSP, planning graphs or state space encodings (Bresina *et al.* 2002). Finally, scalability is also an issue given the size of the problem and the number of potential contingencies. Our approach tries to alleviate these problems by being proactive, and focusing on activities with high probability of failure. Some other techniques that follow similar reasoning are *Just-In-Case* (JIC) contingency scheduling (Drummond, Bresina, & Swanson 1994) and Mahinur (Onder & Pollack 1999).

Conclusion

Preliminary evaluation indicates that the full CSC system significantly outperforms the baseline both in terms of the average quality of schedule execution (see Figure 6) and in terms of avoidance of schedule execution failures. The preliminary ablation studies show no significant difference in the scores obtained using the deliberative scheduler only, the opportunistic scheduler only or the deliberative and opportunistic scheduler together. The deliberative and opportunistic schedulers use very different approaches, so we are surprised to see no significant difference in their scores. We speculate that the structure of the problems is such that both approaches identify similar solutions. In order to verify this hypothesis, we are now conducting experiments to analyze the detailed behavior of the system on specific problem instances. We are also working on a range of evaluation tools ranging from problem generators, visualization tools and centralized algorithms to compute benchmark scores.

Acknowledgments

The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S.Government is authorized to reproduce and distribute reports 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 any of the above organizations or any person connected with them.

References

Beynier, A., and Mouaddib, A. 2005. A polynomial algorithm for decentralized markov decision processes with temporal constraints. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi Agent Systems(AAMAS-05)*.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for ai. In *Proceedings of UAI*.

Cox, J.; Durfee, E.; and Bartold, T. 2005. A distributed framework for solving the multiagent plan coordination problem. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi Agent Systems(AAMAS-05)*, 821–827.

Decker, K., and Lesser, V. 1993. Quantitative modeling of complex computational task environments. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, 217–224.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, 1098–1104.

Goldman, C., and Zilberstein, S. 2003. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems(AAMAS-03)*, 137–144.

Littman, M.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *Artificial Intelligence Research (JAIR)* 9:1–36.

Nair, R.; Varakantham, P.; M.'Tambe; and Yokoo, M. 2005. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi Agent Systems(AAMAS-05)*.

Onder, N., and Pollack, M. 1999. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, 577–584.

Shen, J.; Becker, R.; and Lesser, V. 2006. Agent Interaction in Distributed MDPs and its Implications on Complexity. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*.