Distributed Scheduling for Multi-Agent Teamwork in Uncertain Domains: Criticality-Sensitive Coordination

Rajiv T. Maheswaran, Craig M. Rogers, Romeo Sanchez,
Pedro Szekely and Po-An Chen
University of Southern California - Information Sciences Institute
4676 Admiralty Way, Suite 1001, Marina Del Rey, CA 90292
{maheswar, rogers, rsanchez, pszekely, pchen}@isi.edu

Abstract

We consider a team of agents that are required to coordinate their actions in order to maximize a global objective. Our domains are characterized by uncertainty, dynamism, and distributed information. Determining appropriate actions becomes quite difficult, especially as the number of agents and the coupling between them increases. This paper discusses four contributions toward the goal of coordinating agents under uncertainty in large-scale settings: (i) an approach based on identifying the *criticality* of various activities with respect to their effect on the team reward; (ii) an architecture and implemented coordinator agent that can execute this approach in a distributed manner; (iii) metrics for evaluating system performance in such settings, and (iv) a proof of concept of our approach on both focused and randomly-generated experimental domains.

1 Introduction

This work addresses the collaborative and coordinated execution of activities of a multi-agent team in domains with uncertainty. Joint operations in military settings, project/personnel management in global enterprise settings,

and multiple-rover/UAV missions in science-discovery/search-and-rescue settings are some examples of complex and dynamic execution environments where effective and efficient coordination is crucial. Characteristics of problems in these domains include: (i) *uncertainty* associated with the execution environment, causing an explosion on the number of system states for which actions need to be determined, (ii) *dynamism*, meaning that parameters that define the initial models of uncertainty and constraints may change in the execution phase, requiring online reasoning and decision-making, (iii) *making decisions over time*, leading to either to a further explosion in the state space (by including time as an extra state dimension) or action space (by differentiating identical actions at various times), and (iv) *partial observability* of global goals and other agents' policies, where agents' local policies may not align towards a coherent global strategy. Discovering an approach that will scale with computationally-bounded agents making decisions in real-time is a challenging task.

A centralized problem-solver eliminates the issues of partial observability and coordination, 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 further 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 within our computational bounds. 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 limitations that are brought about by scale, dynamism and bounded rationality that require manageable state spaces and quick reasoning.

In this paper, we describe our approach to coordination which is based on the notion of *criticality*. Our higher level reasoning has three components: an opportunistic scheduler (which performs local optimization), a deliberative scheduler (which performs partially-centralized repairs), and a downgrader (which proactively guides the execution into better regions of operations). Each component utilizes low-dimensional metrics that capture *criticality* with respect to the decisions assigned to it. In practical systems, decision-making

agents need to manage distributed information, communication and execution. The effects of these real-world details can be lost in purely theoretical simulations. To accurately evaluate our approach, we have constructed an architecture to implement our coordination reasoning in functional agents capable of running on distributed machines. Furthermore, there is no standard metric evaluating performance in large-scale settings with uncertainty. Determining an optimal solution is intractable due to the large number of potential sample paths. Here, we discuss a tractable option for setting an upper bar for performance and propose a benchmark approach for a lower bar. The upper bar is calculated by the performance of a prescient central agent. The suggested benchmark approach is a fully-local policy that therefore scales. Finally, we test the validity of our ideas on both, a focused test that isolates the capabilities of the approach, and on diverse randomly generated set of scenarios. While not universally dominant over the performance evaluation standards introduced (in general, a difficult claim to prove), we begin to discover the regions of operation where we exceed the benchmark.

2 Model

We model our problem with CTAEMS, which is an adaptation of the TAEMS formulation [5] representing distributed multi-agent coordination problems. A scenario is run over a finite horizon of decision epochs indexed by $t \in \{1, \cdots T\}$. Each agent $a \in A$ has the potential to execute a set of activities or *methods*. Let (a, m) denote the m^{th} method available to agent a where $m \in \{1, \cdots, N^a\}$ and N^a is the total number of methods available to agent a. Each method is assigned uniquely to a single agent, i.e., given $(a_i, m_i), (a_j, m_j)$, if $a_i \neq a_j$, then (a_i, m_i) and (a_j, m_j) must refer to different methods. Each method (a, m) produces some quality $q^{a,m}$ and takes a duration $\delta^{a,m}$ from a set of possible qualities an durations, respectively.

$$\delta^{a,m} \in \{\delta_1^{a,m}, \cdots, \delta_{N^{a,m,\delta}}^{a,m}\}$$
$$q^{a,m} \in \{q_1^{a,m}, \cdots, q_{N^{a,m,q}}^{a,m}\}$$

The likelihood of obtaining a particular outcome is represented by probability distributions

$$P^{a,m,\delta} \in \{p_1^{a,m,\delta}, \cdots, p_{N^{a,m,\delta}}^{a,m,\delta}\}$$
$$P^{a,m,q} \in \{p_1^{a,m,q}, \cdots, p_{N^{a,m,q}}^{a,m,q}\}$$

Let $q^{a,m}(t)$ denote the quality accrued by a method at time t. Once an agent starts a method, it can either abort it or wait for its completion in order to start another method. In addition, each method belongs to a *window* that prescribes the earliest time at which it may begin (referred to as the *release*, denoted as $r^{a,m}$), and the latest time at which it may complete in order to obtain positive quality (referred to as the deadline, denoted as $d^{a,m}$). Thus, if the start time of method (a, m) is $s^{a,m}$ and the end time is $e^{a,m}$, we have $q^{a,m}(t) = 0\ \forall t$, if $s^{a,m} < r^{a,m}$ or $e^{a,m} > d^{a,m}$. If a method has $q^{a,m}(d^{a,m}) = 0$, then it is considered a *failed* method or *failure*. This occurs if $q^{a,m}(e^{a,m}) = 0$, if it completes before the deadline ($\delta^{a,m} \leq d^{a,m}$), or if the method is aborted.

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.

Formally, we index a task by $T \in \{1, \cdots, N^T\} =: \mathcal{T}$ where N^T is the total number of tasks. If $\mathcal{N} = \mathcal{T} \cup \mathcal{M}$ is the set of all *nodes* where $\mathcal{M}$ is the set of all methods, let $C(n) \subset \mathcal{N}$ be the set of children for node n . We have $C(n) = \emptyset$, if and only if $n \in \mathcal{M}$. Furthermore, if node n has no parent, $n \notin C(\tilde{n}),\ \forall \tilde{n} \in \mathcal{N}$, then it is the root node of a tree. Here, we consider cases where the task structure is a single tree and only a single node (task) may be a root. Let $Q^n(t)$ denote the quality accrued at time t at node n. If $n \in \mathcal{M}$, then $Q^n(t) = q^n(t)$. If $n \in \mathcal{T}$, then

$$Q^n(t) = f : \{Q^{\tilde{n}}(t) : \tilde{n} \in C(n)\} \rightarrow I\!R$$

where f is the quality accumulation function. The quality accumulated at the root node at time T is the reward to the team.

The QAFs include: (i) Min, which yields the minimum value of qualities of the children, used to model situations where all children must succeed 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.

An additional complexity is the possibility of non-local effects (NLEs) between nodes. An NLE is an *enabling* condition between nodes represented by a directed link between a source and a target. Target methods started before all their sources have accumulated quality will fail (accumulate zero quality). The start time for a node n (denoted as s^n), that is a *task* is the minimum start time of its children: $s^n = \min\{s^{\tilde{n}}\}_{\tilde{n} \in C(n)}$. If the target is a task, the NLE applies to all descendant nodes. Thus, if $\mathcal{E} = \{(n_s, n_t)\}$ is a set of source and target nodes that capture all enables NLEs, then $Q^{n_t} = 0 \ \forall t$ if $Q^{n_s}(s^{n_t} - 1) = 0$ for any n_s that is part of a pair $(n_s, \tilde{n_t})$ where $\tilde{n}_t = n_t$ or n_t is a descendent of $\tilde{n}_t$.

Dynamism is added through the capability of the environment to change or use different probabilities, releases, or deadlines than those stated at $t = 1$. These changes can occur at any time and may or may not be announced to the agents. For example, an unmodeled failure can occur, when the environment returns a failure for a method whose prescribed distribution contained no failure outcomes. While the model described here is only a subset of TAEMS, it is sufficiently rich to create problems of great complexity.

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. The agent team is equipped with an initial static schedule (i.e., a list of methods to execute and associated start times). This is typically a bad strategy as it does not react to failures or varying execution times which may invalidate or cripple future activities. Effective policies must react dynamically not only to the uncertainties in the durations and qualities of methods in the schedule, but also to the dynamism of the environment, and generate 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 immediate NLE source or target nodes of any nodes in the ancestry. This models the partial-observability inherent in multi-agent systems, where an agent cannot monitor the entire system and often is not aware of the states, actions or even the existence of other agents in the team.

3 Approach

Our approach consists of embedding the notion of criticality in three higher-level reasoning components: a locally optimized resource allocation (*opportunistic scheduler*), a partially-centralized solution repair (*deliberative scheduler*) and proactive re-prioritization (*downgrader*). To construct a functional agent capable of utilizing these strategies, we developed an architecture for our Criticality-Sensitive Coordinator (CSC) agent that is displayed in Figure 1. The higher-level reasoning components are triggered and supported by a *state manager* and an *execution controller*. We first discuss these support modules that provide the infrastructure for information, communication and execution. Then, we describe the higher-level reasoning components.

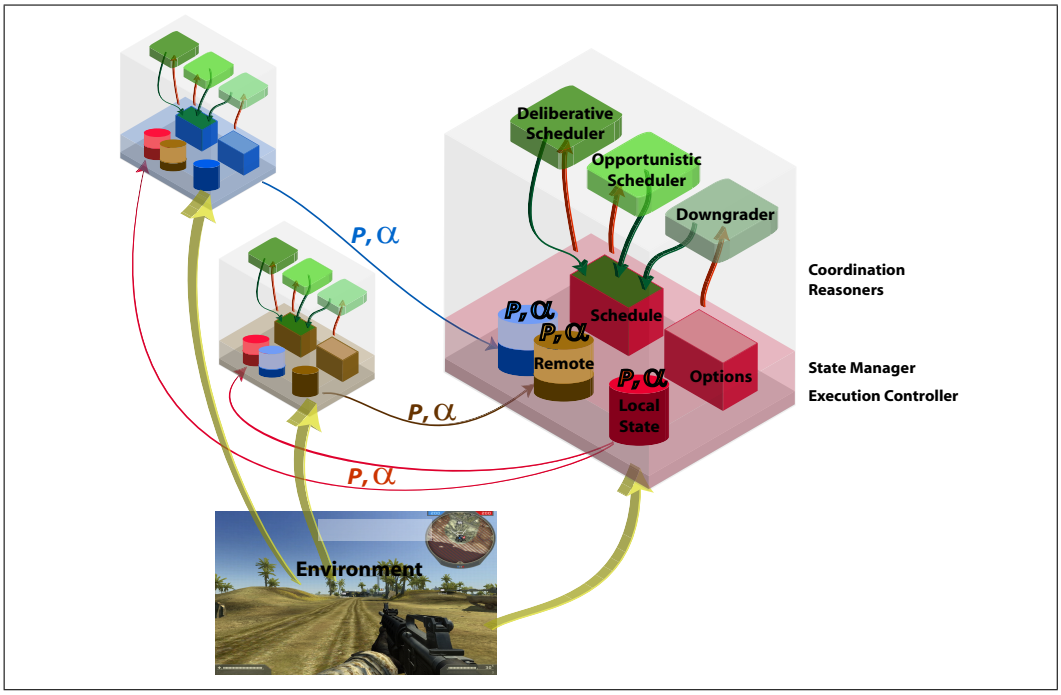

Figure 1: CSC Architecture

3.1 State Manager and Execution Controller

3.1.1 State Manager

The purpose of the state manager is to give the reasoning components the information they need to act. It also performs the communication necessary to keep this information up-to-date. There are essentially two types of such information: *probabilities* and *importance*, denoted by p and α respectively in Figure 1. Both types of information are kept about the *current schedule* and about *potential schedules* that the agents may engage. Probabilies and importance are stored in a structure known as a *profile*. These quantities in various combinations are the input that capture criticality in the higher-level reasoning components.

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 protocols to share local information with other agents.

3.1.2 Schedule

The system stores the current schedule in a distributed manner in the state managers of each agent. Every *method* has an associated *scheduled start window* and a *priority*. At the beginning of a run, these are determined based on an *initial schedule* contained in the initial subjective view of an agent. At runtime, the opportunistic scheduler, deliberative scheduler and the downgrader alter the current schedule by modifying the start windows and priorities of scheduled methods, or by adding new methods to the schedule. The deliberative scheduler can create a set of schedule changes that are installed on remote agents while the other two operate only on local methods. 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.

3.1.3 Profiles

Profiles are designed to reason about uncertainty. A profile contains: (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. Probabilities reflect the likelihood that methods will achieve positive quality before their deadlines. 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 their likelihood and costs independent of the current schedule and a set of pairs for higher nodes, that 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.

3.1.4 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 starting execution of a method, into a platform-neutral representation for portability. It runs at a higher priority level than the rest of the agent, avoiding priority inversions when accessing shared data. This allows it to continue to execute scheduled methods even when the rest of the agent lags behind the pace of execution environment events. This is critical in large-scale problem domains with un-

certainty where reasoning components may be overrun due to computational and communication-handling burdens.

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.

3.2 Higher-Level Reasoning Components

3.2.1 Opportunistic Scheduler

The opportunistic scheduler is instantiated whenever there is a gap where an agent is idle. 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 (e.g. starting a method under a SyncSum may damage a coordinated start by other methods). Then, the opportunistic scheduler utilizes the scheduled probabilities and importance measures in the profile to calculate a criticality factor that determines the method with the greatest likelihood of helping the team. This method is inserted with a 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.

3.2.2 Deliberative Scheduler

The deliberative scheduler is triggered when the scheduled probability of nodes falls below a threshold determined based on the position of the node

in the CTAEMS task structure. Thus, scheduled probability and task structure combine to form the criticality factor for the deliberative scheduler. Because these probabilities are calculated for methods and nodes throughout the structure, the deliberative scheduler can (and often is) called to fix problems predicted to occur in the future. This is an important capability given that repairing a failure at a particular time can necessitate scheduling methods before that time due to the enables NLEs. When a problem at a node is detected, the deliberative scheduler accesses the profile to obtain probabilities of potential schedules. Potential probabilities of the problem node are constructed using combinations of potential probabilities of children nodes to create a Pareto frontier of options. These options, along with the transitive closure of the Pareto frontiers of potential probabilities for nodes that enable something in the subtree of the problem node, are sent to a scheduler. The scheduler checks the feasibility of the various options and assigns start windows to the appropriate methods to fix the problem node.

3.2.3 Downgrader

The downgrader utilizes the potential importance as its measure of criticality 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 can no longer contribute to boost the probability of the root task, freeing agent resources for other scheduler components that schedule methods at higher priority. These methods are not removed from the schedule given that they may boost the total quality. 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.

4 Metrics for Performance

Evaluating performance of a system dealing with uncertainty at a large-scale is especially challenging. The notion of optimality is difficult because finding an optimal solution is computationally intractable as the number of sample paths that the system can take is immense. Furthermore, with dynamic model changes, finding an optimal solution, becomes even more

daunting and potentially impossible if the space of changes is uncountably large or undefined. Here, we discuss two possible methods to evaluate system performance for the static problem with uncertainty and scale, though the ideas may be extended to domains with dynamism.

To obtain an upper bound, for each run of a scenario, we construct a *prescient* solver that knows the duration and quality outcomes of the problem *a priori*. With this information in hand, the prescient solver can compose a schedule of methods with no failure that can run without repair and maximize the quality of the root. The quality of this solution will be higher than the expected quality of an optimal causal solution, even if it were computational tractable. We have devised a pseudo-boolean encoding that takes the outcome draws, and yields the maximum achievable quality for any trial of a scenario.

In addition to an upper bound, we propose the need for a lower bound to evaluate systems. One should be able to determine how much improvement a particular reasoning strategy has added beyond a basic implementation that suits the environment's restrictions. The quality of simply executing an initial schedule is generally too low for a benchmark, as it may be easy to improve with simple reasoning. We propose that a benchmark system for large-scale multi-agent systems in uncertain domains be one where agents make the best local decision at all time without any communication. This system is scalable to any degree as there are no effects of communication. Computing a local decision with the given bounded rationality must be feasible for any worthwhile investigation. This is similar to the opportunistic scheduler, without the measure of importance. Thus, we propose the *Benchmark Opportunistic Scheduler* and *Prescient Optimal Solution* as metrics for lower and upper bars of performance, respectively, for large-scale systems with uncertainty.

5 Experiments

In order to test our system and its various reasoning components against the metrics proposed earlier, we conducted an extensive sets of experiments. Here we describe results for (i) a constructed scenario intended to isolate and test various reasoning components to verify the validity of our concepts (hereby referred to as the *backup problem*), and also (ii) a large set of randomly generated examples.

Intuitively, the backup problem has a set of primary agents, each working on a single problem over time. A set of of backup agents are available to buffer or repair the activities of the primary agents and are capable of working on multiple problems (though not simultaneously). The challenge is to find the best way to assign backup agents to methods that leads to the greatest number of problems being completed successfully. The CTAEMS formulation of the backup problem is parameterized, but for simplicity we discuss a concrete set of instances. We begin with a root node with 20 children nodes referred to as *problems*. Each problem node is a Min QAF, and has identical structure to other problem nodes except for the probability distributions and agent ownership of the methods. Each problem has W children, referred to as *windows*. Due to the Min QAF of the problem, all windows must be successfully completed to successfully complete the problem. The window nodes (Sum QAFs) determine the release and deadlines for all the the methods under it. The interval between release and deadline is of length 40 and each window overlaps with the adjacent windows by 8, i.e. the release and deadlines of the windows are: $\{[0, 40], [32, 72], [64, 104], ...\}$. There is a Max QAF node under each window with four methods as children, i.e. at least one of the children must succeed to successfully complete a window. One method is the *primary* method and is part of the initial schedule. All primaries are assigned to a primary agent that handles all primary methods for a single problem. Thus, there are 20 primary agents. There are 3 additional *backup methods* whose owner is determined by drawing randomly from a pool of 10 backup agents. The primary methods have quality outcomes $q^{0,m} \in \{0, 1\}$ with success probability $P(q^{0,m} = 1) \in [0.5, 1.0]$ and have durations $\delta^{0,m} \in \{7+d_s, 8+d_s, 9+d_s\}$ with probabilities $p^{0,m,\delta} \in \{0.25, 0.50, 0.25\}$ and $d_s \in [-4 \quad 4]$. Primary methods are released at the beginning of the window and have deadlines at the end of the window. The backup methods have quality outcomes $q^{0,m} \in \{0, 1\}$ with success probability $P[q^{0,m} = 1] \in [0.8, 0.9]$ and have durations $\delta^{0,m} \in \{23, 24, 25\}$ with probabilities $p^{0,m,\delta} \in \{0.25, 0.50, 0.25\}$. The release of each backup method is the beginning of the window plus a shift drawn randomly from $\{1, \cdots, 7\}$. The deadline of each backup method is the end of the window. We ran the backup problem with windows ranging from $W \in \{1, \cdots, 10\}$ with our system run on a distributed cluster of 8 machines. In the 10-window case, this is a system with 30 agents, 800 methods, and over 1000 nodes which is quite large for settings with uncertainty. For this example, we wanted to isolate the opportunistic scheduler and the downgrader, so the deliberative scheduler was not used in testing. The performance of

our system with respect to the separately simulated benchmarks is shown in Figure 2.

What we can extract is that the Benchmark Opportunistic Scheduler (BOS) does outperform the Initial Schedule Execution. Since it is a simple strategy that improves quality, it serves as a better baseline for comparison. In the backup problem experiment, the Prescient Optimal Solution (POS) always yielded the theoretical maximum. This may imply that using POS as an upper bound may be too strong or perhaps not as useful. It could also be an artifact of our construct. In our experiments, CSC is shown to have outperformed BOS. The reason for this is that the opportunistic scheduler uses global information to better allocate backups, and the downgrader frees up resources for these repairs to occur. This, by no means suggests that this phenomenon is universal.

To further test our system, we randomly generated 86 scenarios that were designed on a categorization of our scenarios based on NLE chains and failure rates. We have four subclasses (empty, low, medium and high) where 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 with scenarios using increasing degrees of window overlap. The results of four agents running 8 trials per scenario for both the distributed implementation of CSC and BOS are shown in Figure 3. CSC outperforms the benchmark in 73% of the scenarios.

6 Related Work

The coordination of multi-agent systems in dynamic, distributed, stochastic, temporally-constrained and partially observable domains is a challenging problem. One way to control cooperative multi-agent systems under such conditions is through Decentralized MDPs (DEC-MDP or DEC-POMDP). Unfortunately, the most general decision-theoretic models for this problem have been proved to be extremely complex (NEXP-complete) [12, 1, 11].

In order to lower complexity, some models restrict the types of interactions allowed in the system, the amount of communication among agents, or the general features they are able to address. One of such models is the Opportunity Cost DEC-MDP (OC-DEC-MDP) [2]. This model bases its computation in local policies, taking into account the loss in value produced

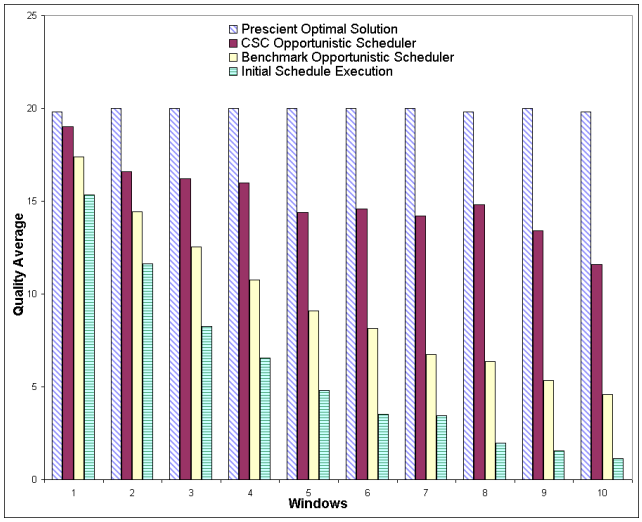

Figure 2: Backup Problem

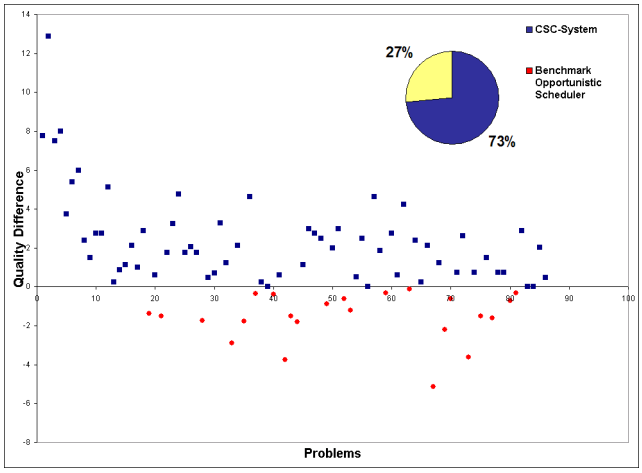

Figure 3: CSC vs.Benchmark

by its local computation. The approach also enforces temporal constraints among the activities to eliminate the communication among agents. 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. Other approaches allow the agents to communicate to exchange local policies. The DEC-POMDP with communication (DEC-POMDP-Com) [7] presents a first greedy meta-level approach to agent communication, unfortunately the number of agents considered by the framework is very small.

Another way of modeling the multi-agent coordination problem is through traditional AI methods. The Distributed Constraint Optimization Problem (DCOP) framework captures the locality of interactions among agents with a small number of neighbors [4], it fails to capture the uncertainty in the general problem. Network Distributed POMDPs have been proposed to address these issues [9]. Unfortunately, they solve only small problems.

Decision-theoretic planning can also be used to model our problem [8]. 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 [3]. 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 [6] and Mahinur [10], but they focus in a centralized, single-agent solution to the problem.

7 Conclusion

In this paper, we presented a criticality-sensitive approach to coordinating multi-agent systems operating in uncertain and larg-scale domains. The distributed implementation of our ideas has shown in many cases to improve upon a proposed benchmark system. Interestingly, preliminary ablation studies (not discussed here) show no significant difference in the qualities obtained using the deliberative scheduler only, the opportunistic scheduler only or the

deliberative and opportunistic scheduler together. This is surprising as the deliberative and opportunistic schedulers use very different approaches. 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, additional visualization tools and centralized algorithms to compute alternate performance metrics. Identifying the regimes in which each scheduler is dominant has not yielded any insight, thus far, into why the winning scheduler was the best choice. These investigations reflect our understanding that, while a model such as TAEMS can be stated succinctly, the uncertainty and scale combine very rapidly to form problems that are challenging to solve and understand.

8 Acknowledgments

The authors would like to thank Marcel Becker, Stephen Fitzpatrick, Gergely Gati, David Hanak, Jing Jin, Gabor Karsai, Bob Neches, Nader Noori, Kevin Smyth and Chris van Buskirk for their contributions to the design, implementation, debugging and testing of CSC.

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

[1] D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[2] A. Beynier and A. Mouaddib. 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), 2005.

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

[4] J. Cox, E. Durfee, and T. Bartold. 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)*, pages 821–827, 2005.

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

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

[7] C. Goldman and S. Zilberstein. 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)*, pages 137–144, 2003.

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

[9] R. Nair, P. Varakantham, M.'Tambe, and M. Yokoo. 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)*, 2005.

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

[11] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Artificial Intelligence Research (JAIR)*, pages 389–423, 2002.

[12] J. Shen, R. Becker, and V. Lesser. 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*, 2006.