Strengthening Schedules Through Uncertainty Analysis

Laura M. Hiatt[†], Terry L. Zimmerman[‡], Stephen F. Smith[‡], Reid Simmons[†‡]
[†]Computer Science Department
[‡]Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{lahiatt, wizim, sfs, reids}@cs.cmu.edu

Abstract

In this paper, we describe an approach to scheduling under uncertainty that achieves scalability through a coupling of deterministic and probabilistic reasoning. Our specific focus is a class of over-subscribed scheduling problems where the goal is to maximize the reward earned by a team of agents in a distributed execution environment. There is uncertainty in both the duration and outcomes of executed activities. To ensure scalability, our solution approach takes as its starting point an initial deterministic schedule for the agents, computed using expected duration reasoning. This initial agent schedule is probabilistically analyzed to find likely points of failure, and then selectively strengthened based on this analysis.

For each scheduled activity, the probability of failing and the impact that failure would have on the schedule's overall reward are calculated and used to focus schedule strengthening actions. Such actions generally entail fundamental trade-offs; for example, modifications that increase the certainty that a high-reward activity succeeds may decrease the schedule slack available to accommodate uncertainty during execution. We describe a principled approach to handling these trade-offs based on the schedule's "expected reward," using it as a metric to ensure that all schedule modifications are ultimately beneficial. Finally, we present experimental results obtained using a multi-agent simulation environment, which confirm that executing schedules strengthened in this way result in significantly higher rewards than are achieved by executing the corresponding initial schedules.

1 Introduction

The general problem of scheduling for uncertain domains remains a difficult challenge. Research that focuses on constructing schedules (or scheduling policies) by reasoning with explicit models of uncertainty has produced some promising mechanisms for coping with specific types of domain uncertainties (e.g., [McKay *et al.*, 2000; Beck and Wilson, 2007]), but in general these approaches have scaling difficulties [Bryce *et al.*, 2008]. Research in robust scheduling has alternatively emphasized the use of expected models and deterministic scheduling techniques, with the goal of constructing a flexible schedule (or a set of schedules) that can absorb deviations at execution time [Policella *et al.*, 2006]. These approaches are much more scalable, but either result in overly conservative schedules at the expense of performance (e.g., [Morris *et al.*, 2001]) or ignore the potential leverage that can be provided by an explicit uncertainty model.

In this paper, we adopt a composite approach that couples the strengths of the two above tactics. Like [Policella *et al.*, 2006], we assume as our starting point a partial-order schedule, which is constructed (based on deterministic modeling assumptions) to optimize the performance objective at hand, but retains temporal flexibility where permitted by domain constraints as a basic hedge against uncertainty. Our main contribution is the act of layering an uncertainty analysis on top of the deterministic schedule, taking advantage of the known uncertainty model while avoiding the computational overhead of probabilistic scheduling. We use this analysis of the schedule to identify its weak points and strengthen them, increasing the expected quality of schedule execution.

The general problem that motivates our work is that of coordinating the activities of a team of collaborative agents as they execute a joint schedule in an uncertain environment. In the class of problems we focus on, various activities in the joint schedule contribute differentially to the reward (referred to in this paper as quality) that is achieved by executing the schedule. All activities have both durational and outcome uncertainty, and are subject to deadlines. Lower quality and shorter duration backup activities are often available as alternatives if time is short. Overall, the challenge is one of constructing a robust, quality-maximizing schedule.

Some ways to increase schedule robustness in this context are to: (1) protect against activity failure by adding redundant, backup activities; (2) increase schedule slack by replacing activities with shorter backup activities; or (3) remove activities altogether, potentially increasing slack and the likelihood that other methods meet their deadlines. However, there are basic trade-offs in trying to fortify the schedule in these ways. The addition of new activities will reduce overall temporal slack in the agents' schedules, possibly increasing the risk that other activities will fail to execute within their

specified time windows; similarly, removing activities from the timeline ensure that they earn zero quality, potentially decreasing quality earned overall.

We argue that reasoning about the uncertainty associated with scheduled activities provides a basis for effectively strengthening deterministic schedules. In this paper, a failure probability analysis is developed for targeting the most profitable activities to bolster, and an analysis of overall expected quality is defined to determine when individual strengthening actions are beneficial. These offline analyses serve to strengthen the initial schedules in a manageable amount of time before they are distributed to agents for the execution phase. On a reference set of multi-agent problems generated for the DARPA Coordinators program[1], we demonstrate, in simulated execution experiments, that schedules strengthened in this manner are significantly more robust and earn more quality during execution than the original schedules.

2 Related Work

At a high level, there are two main approaches that have been applied to multi-agent problems with domain uncertainty: reasoning with probabilities during the scheduling process; and building deterministic schedules that are robust in the face of executional uncertainty.

2.1 Probabilistic Planning

[Xuan and Lesser, 1999; Wagner *et al.*, 2006] consider a domain similar to ours, and use contingency planning (i.e. multiple plan branches based on execution results) in their scheduling process and to manage the uncertainty between agents. Contingency planning, however, incurs a significant computational overhead for their approach. In contrast, conformant planners such as [Smith and Weld, 1998; Onder *et al.*, 2006] deal with uncertainty by developing single-path plans that are robust regardless of action outcomes. Such plans, however, often cannot be found even in classical planning domains and are ill-suited for our problem of interest where the goal is to maximize a metric like quality.

Policy-based planners such as MDPs have also been applied in such domains and recently extended for temporal reasoning and multiple agents [Younes and Simmons, 2004; Mausam and Weld, 2006]. There has also been some work in heuristic policy generation that attemps to overcome scalability limitations [Musliner *et al.*, 2007]. However, the Coordinators program found the problems we are considering to be beyond the capabilities of an MDP problem solver that it developed to establish problem benchmarks.

2.2 Robust Scheduling

Partial-order schedules [Policella *et al.*, 2006] flexibly allow activity execution intervals to drift within a range of feasible start and end times. This allows the plan to absorb deviations in execution while still maintaining a valid plan. Plans can also be probabilistically verified [Fox *et al.*, 2006] to increase robustness. Alternately, a plan can be configured to be dynamically controllable [Morris *et al.*, 2001], which guarantees that a solution strategy exists for an uncertain temporal
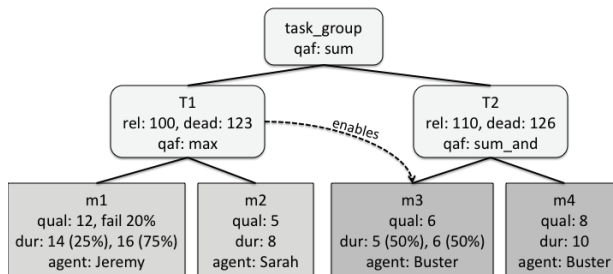
Figure 1: A simple 3-agent *C_TAEMS* problem.

planning problem. The above approaches in general, however, are either overly conservative or ignore the extra leverage that explicitly considering uncertainty can provide.

3 Domain

We focus on a multi-agent scheduling problem that is concerned with the collaborative execution of a joint mission by a team of agents in a highly dynamic environment. Missions are formulated as a network of activities in a version of the *TAEMS* language (*T*ask *A*nalysis, *E*nvironment *M*odeling and *S*imulation) [Decker, 1996] called *C_TAEMS* [Boddy *et al.*, 2005]. A simple example of such a hierarchical activity network is shown in Figure 1. The root of the tree is the overall mission task, called the "taskgroup" (TG), and agents cooperate to accrue as much quality at this level as possible. On successive levels, interior nodes constitute aggregate activities that can be decomposed into sets of sub-activities and/or primitive activities (leaf nodes), which are directly executable in the world. Hereafter we refer to interior nodes as "tasks" and leaf nodes as "methods"; the term "activity" will be used to refer to either type of node.

Each method can be executed only by a specified agent (*agent: Name* in Figure 1) and each agent can execute at most one activity at a time. Method durations are specified as discrete probability distributions, and the exact durations are known only after they have been executed. Method qualities, (*qual: X*), are deterministic. Methods may also have a likelihood of failing (*fail: X%*), which results in zero quality.

Each task in the activity network has a specified *quality accumulation function* (*qaf*) defining when and how a task accumulates quality. For example, a *sum* task accrues quality when one child accrues positive quality, and its final value is the total earned quality of all children. Other *qafs* are *max*, which earns the maximum quality of its executed children, and *sum_and*, which earns the sum of the qualities of its children once they *all* execute with positive quality. For simplicity we omit discussion of additional *qafs* here.

A common structural feature of these task networks is to have *max* tasks as the direct parents of several leaf node methods. These sibling methods are typically of different durations and qualities, and often belong to different agents. For example, a search and rescue domain might have "Restore Power" modeled as a *max* task, with the option of fully restoring power to a town (higher quality and longer duration) or only restoring power to priority buildings (lower quality and

shorter duration). Hereafter we refer to *max* tasks that are parents of methods as "max-leafs." They play a key role in the schedule strengthening strategies described below.

Inter-dependencies between activities can be modeled via non-local effects (NLEs). NLEs express causal preconditions: for example, the *enables* NLE in Figure 1 stipulates that the target activity m_3 cannot be executed until T_1 accumulates positive quality. *C_TAEMS* includes other types of NLEs but for simplicity we omit discussion of these here.

Additional constraints in the network include a release (earliest start) time and a deadline (denoted by *rel: X* and *dead: X* respectively in Figure 1) that can be specified for any node. Each descendant of a task inherits these constraints from its ancestors, and its effective execution window is defined by the tightest of these constraints. An executed activity earns zero quality if any of these constraints are violated.

During execution, agents receive a copy of their portion of the initial schedule and limited local views of the task hierarchy. Agents send commands to a simulator to begin method execution, and receive back execution results based on the stochastics of the problem description as they become known. Agents also communicate relevant status information to other agents asynchronously.

4 Approach

While deterministic planners have the benefit of being more scalable than probabilistic schedulers, they make strong and often inaccurate assumptions on how execution will unfold. In oversubscribed domains such as ours, if the timeline is tightly packed and even one method exceeds its scheduled duration, there is a high risk that a later activity will violate one of its constraints (e.g. a deadline).

Rescheduling during execution can effectively respond to such unexpected dynamics, as long as dependencies allow sufficient time for one or more agent schedulers to recover from problems that arise. Our interest is in making deterministic schedules more robust in order to minimize the number of unrecoverable problems. To do so, we augment deterministic schedulers with a schedule strengthening step, which is focused by probabilistic analysis. In this paper we use a centralized analysis to target areas of an initial schedule that are likely to earn less than their projected quality, and fortify these areas to increase the schedule's expected quality, making the schedule more robust during distributed execution.

4.1 Core Scheduler

Our scheduling approach is rooted in an incremental flexible-times scheduling framework [Smith *et al.*, 2007]. We adopt a partial-order schedule (*POS*) representation where the execution intervals associated with scheduled activities are allowed to float within imposed time and activity sequencing constraints. Its underlying implementation as a Simple Temporal Network (STN) model provides efficient updating and consistency enforcement mechanisms.

To produce an initial deterministic schedule, we compute and assume expected duration values for methods in the input problem's activity network and focus on producing a *POS* for the set of agents that maximizes overall problem quality. Given the complexity of solving this problem optimally

and our interest in scalability, we adopt a heuristic approach. Specifically, a schedule is developed via an iterative two-phase process. In the first phase a relaxed version of the problem is solved optimally to determine the set of "contributor" methods, or methods that would maximize overall quality if there were no capacity constraints. In the second phase, an attempt is made to sequentially allocate these methods to agent timelines. If a contributor being allocated depends on one or more enabling activities that are not already scheduled, these are scheduled first. Should a method fail to be inserted at any point (due to capacity constraints), the first step is re-invoked with the problematic method excluded. More detail on this can be found in [Smith *et al.*, 2007].

4.2 Schedule Strengthening Tactics

To date, we have investigated three strengthening tactics:

1. "Backfilling" - Scheduling redundant methods under a max-leaf task to improve the task's likelihood of earning quality (recall that *max* tasks accrue only the quality of the highest quality child that executes successfully).

2. "Swapping" - Replacing a method with a sibling method that has either a lower failure likelihood or a shorter duration (to increase schedule slack).

3. "Pruning" - Unscheduling a method from the timeline to increase schedule slack, thereby increasing the likelihood that remaining methods earn quality.

All three tactics have an associated trade-off with their use: backfilling decreases schedule slack, potentially increasing the risk that other methods will miss their deadline; swapping can lower quality when it replaces methods with lower-quality siblings; and pruning can also lead to quality loss since activities go unexecuted. Although this limits the usefullness of these tactics separately, their different properties can also have synergistic effects (e.g. swapping can increase schedule slack, making room for backfilling).

When performing schedule robustification on an initial centralized schedule, we first identify parts of the schedule that need strengthening, such as a method likely to miss its deadline, and try to use a tactic to strengthen it. We have previously shown that targeting areas of the schedule in this way is a very effective way to strengthen schedules [Hiatt *et al.*, 2008]. Next we calculate the change in overall expected quality of the schedule, and commit to actions that increase the schedule's expected quality, ensuring that the strengthening is helpful overall. The next sections describe this process.

4.3 Probabilistic Analysis of Deterministic Schedules

We developed an algorithm that performs a *P*robabilistic *A*nalysis of *D*eterministic *S*chedules (PADS). This algorithm propagates probability information through the task hierarchy, yielding a *probability profile* for each activity a which includes its finish time distribution $FT(a)$, its probability of earning quality $PQ(a)$, and its expected quality $EQ(a)$. Probability profiles are stored with each activity, and are readily updated if the information associated with upstream activities changes.

Methods can earn zero quality for three reasons: (1) their execution results in a failure outcome; (2) one of their enablers earns zero quality; or (3) they miss their deadline. As the second and third conditions depend on the probability profiles of upstream activities such as predecessors and enablers, there is a natural order in which the PADS algorithm calculates probability profiles: only after a method's predecessor's and enablers' probability profiles are known can its own profile can be determined.

A method m's finish time distribution $FT(m)$ is a function of its release time, its predecessor's end time distribution, its enablers' end time distributions, and its duration distribution. When $FT(m)$ is known, $PQ(m)$ can be calculated: starting with a value of 1, it is discounted if the method has a failure outcome, if the method has an enabler that might earn zero quality, and if there is a possible finish time past the method's deadline. Finally, the method's expected quality is calculated, $EQ(m) = PQ(m) \cdot qual(m)$, and its probability profile is passed on to downstream activities, as well as up to its parent.

Tasks combine their children's probability profiles according to their *qaf*. The simplest example is a *sum* task. A *sum* task's probability of earning quality equals the probability that at least one child does, $1 - \prod_i (1 - PQ(child_i))$, and its finish time distribution is the distribution of the time that the first child earns quality. Its expected quality is the sum of the expected qualities of its children.

A final useful number is expected quality loss, which acts as a measure of the importance of an activity and is used to prioritize activities for strengthening. The expected quality loss of an activity, $EQL(a)$, compares the schedule's expected quality given the activity earns its full scheduled positive quality, $EQ(TG \mid PQ(a) = 1)$, with the schedule's expected quality should that activity earn zero quality, $EQ(TG \mid PQ(a) = 0)$. We calculate these values by setting the $PQ(a)$ to 1 and 0, respectively, and propagating this through the task tree. Then, $EQL(a) = EQ(TG \mid PQ(a) = 1) - EQ(TG|PQ(a) = 0)$. While loss of an important activity would greatly reduce taskgroup quality, an unimportant activity that earns no quality may have little impact.

4.4 Probabilistic Schedule Strengthening

Schedule strengthening employs the above probabilistic framework to focus improvement on the most vulnerable portions of the initial schedule.

As mentioned above, we consider three tactics when robustifying schedules: backfilling, swapping and pruning. The backfilling step begins by sorting max-leaf tasks with $PQ(t) < 1$ by their expected quality loss. Each task is sequentially considered for backfilling until a single valid backfill is found. A backfill is valid if (1) it is successful (i.e. a child method is successfully scheduled) and (2) it raises the schedule's expected quality. When a valid backfill is found the backfilling step commits to this scheduling action and returns (without trying to backfill further). The backfill step will also return if all max-leaf tasks have been considered but no valid backfill is found.

For the swapping and pruning steps, chains of methods are first identified where methods are scheduled consecutively, threatening to cause a method to miss its deadline. Meth-

ods within the chain are sorted by expected quality loss, as are the chains themselves. The swapping (pruning) step iterates through the chains, attempting to swap (prune) each method in the current chain in turn. Swapping also considers exchanging methods which have a failure likelihood. As with backfilling, the swapping (pruning) step returns either when a valid action (i.e. a successful swap or prune that raises the schedule's expected quality) is found and committed to, or when all possible swaps (prunes) have been ruled out. Note that for all three tactics, a step may or may not result in an actual backfill, swap or prune; if none of the three steps result in a modification, we know that no more steps are possible.

Given these three independent steps, the full exhaustive algorithm that finds the best possible combination of these actions considers each possible order of backfilling, swapping and pruning steps, finds the schedule expected quality that results from each ordering, and returns the sequence of steps which results in the highest expected quality. This algorithm, however, can take an unmanageable amount of time as it is exponential in the number of possible strengthening actions.

Therefore, we instead use a variant of a round-robin strategy, which performs one backfilling step, one swapping step and one pruning step, repeating until no more steps are possible. Two additional heuristics are incorporated. The first prevents swapping methods for lower-quality methods while backfilling the parent max-leaf task (and preserving a higher quality option) is still possible. The second prevents pruning methods, and thus entire branches of the taskgroup, when a backfill or swap is still possible and would better preserve the existing task structure.

We also experimented with a greedy heuristic variant which performs one 'trial' step of each tactic, and commits to the step which raises the expected quality the most, repeating until no more steps are possible. It did not, however, perform as well as the round-robin strategy in our preliminary testing.

5 Experiments and Results

In our experiments we used a test suite of 20 problems, taken from the Phase II evaluation test problem suite of the DARPA Coordinators program. Fifteen of the problems involved 20 agents, 3 involved 10 agents, and 2 involved 25 agents. Five of the problems had under 200 methods, 8 had between 200 and 300 methods, 5 between 300 and 400 methods, and 2 had greater than 400 methods.

5.1 Strengthening Multi-Agent Schedules

We compared the effectiveness of different strengthening strategies by strengthening initial schedules for the 20 problems in 6 ways: pruning only; swapping only; backfilling only; the round-robin strategy; the greedy strategy; and the exhaustive strategy. We used a 3.6 GHz Intel computer with 2GB RAM. For 8 of the 20 problems the exhaustive algorithm did not finish by 36 hours; for these cases we considered the run time to be 36 hours and the quality earned to be the best found when it was stopped. The resulting average expected quality after schedule strengthening, as well as the average running time for each of the algorithms, is shown in Table 1.

Clearly the round-robin strategy outperforms each of the individual strategies it employs at a modest run-time cost. It

Table 1: A comparison of strengthening strategies

	avg expected quality (% of max)	avg running time (in seconds)
initial	74.5	–
prune	77.2	6.7
swap	85.1	32.9
backfill	92.8	68.5
round-robin	99.5	262.0
greedy	98.5	354.5
exhaustive	100	32624.3

is interesting to see how the round-robin strategy outperforms the greedy strategy. We attribute this to the round-robin strategy prioritizing the tactics in order of effectiveness (i.e. backfilling, swapping, then pruning). Round-robin is almost as effective as the exhaustive approach and, with a run-time that is over two orders of magnitude faster, is the preferred choice.

5.2 Executing Strengthened Schedules

We performed an experiment in a multi-agent simulation environment to compare quality earned when executing initial deterministic schedules with quality earned when executing strengthened versions of the schedules. Agents are given a limited, local view of the task hierarchy and their portion of the initial schedule and are responsible for managing the execution of their own schedule. They communicate method start commands with a simulator and receive back execution results (based on the stochastics of the problem) as they become known, and share status information asynchronously. All agents and the simulator run on their own processor and communicate via message passing. We assume communication is perfect and sufficient bandwidth is available.

As we are interested in comparing relative robustness between schedules, we constrain the amount of rescheduling that is allowed during execution. As mentioned above, the flexible-times representation allows methods to float in response to some dynamics. Beyond that the only rescheduling permitted is the removal of a method that can no longer be executed due to, for example, loss of an enabling activity or projected violation of a deadline (detected as a conflict in the underlying STN). Thus the relative quality earned at the end of simulation by each different schedule indicates their relative robustness. We ran simulations comparing three classes of schedules: (1) the schedule generated by the deterministic, centralized planner; (2) the schedule generated by backfilling (the best individual tactic) only; and (3) the schedule generated by round-robin schedule strengthening.

For each problem in our test suite we ran 25 simulations, each with its own stochastically chosen durations and outcomes, but purposefully seeded such that the same set of durations and outcomes were executed by each of the conditions. We numbered the problems from 1 to 20; in general, as the problem numbers increase the problems become larger and have more uncertainty.

Figure 2 shows the average quality earned by the three sets of schedules for each of the 20 problems. The quality

is expressed as the fraction of the best known possible quality earned during execution. We define the best known possible quality for a given simulation to be the quality produced by the deterministic scheduler on an omniscient version of the problem, where the actual method durations and outcomes generated by the simulator are given as inputs.
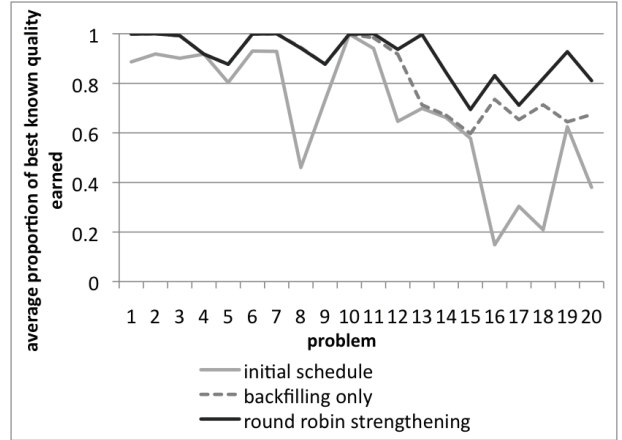

Figure 2: Average quality for each problem earned by the different strengthening strategies, expressed as the proportion of the best known quality earned

For problems 1 through 6, round-robin strengthening identified only backfill steps to be of value, and as a result produced results identical to the backfilling only strategy. Problems 7 through 10, while not identical, are on average very close for these conditions. For individual simulations of these problems, we see that round-robin strengthening usually performs slightly worse than backfilling alone, but occasionally far outperforms it. This interesting behavior arises because round-robin can hedge against method failure by swapping in a lower quality, shorter duration method for a method with a small chance of earning zero quality. While for many runs this tactic sacrifices a few quality points when the original method succeeds, on average round-robin avoids the big quality loss that occurs when the method fails.

Note also the interesting performance of round-robin strengthening for the smaller, more certain problems: it is already near the best maximum known quality it could earn. For the larger and more uncertain (higher numbered) problems, however, further improvement is clearly possible. We discuss this point further below.

As the figure shows, round-robin schedule strengthening outperforms backfilling, which in turn outperforms the initial schedule condition. The differences between the conditions are significant with $p < 0.01$, when tested for significance with a repeated measures ANOVA.

6 Future Work and Conclusions

In this paper, we have described an approach which probabilistically analyzes deterministic schedules to strengthen them in a targeted way. Our hybrid approach is motivated by the desire to scalably exploit knowledge about activity

duration and outcome uncertainty in large-scale, multi-agent scheduling contexts. We use backfilling, swapping and pruning steps in a round-robin strategy to strengthen identified weak portions of an initial deterministic schedule and increase the schedule's expected quality, taking only a modest amount of time. On reference multi-agent scheduling problems, schedules fortified in this way are shown to be significantly more robust and earn on average 31% more quality during execution than their corresponding initial schedules.

As mentioned above, for larger, less certain problems, we would like to improve further on the quality earned by schedule strengthening. We are developing a distributed version of PADS that enables agents to strengthen their schedules as execution unfolds. This run-time approach has the additional benefit of hedging against an agent's scheduler failing during execution as the last schedule produced will likely be more effective in the face of unexpected outcomes.

The distribution of PADS raises interesting issues. The calculation of expected quality loss cannot be performed in the same way as agents do not have access to the whole tree; instead it is necessary to estimate this value. The strengthening tactics must also be altered to allow for coordination so that, for example, multiple agents do not backfill the same max-leaf task at once. Finally, there are off-setting impacts on the overhead incurred by PADS: updating probabilities takes less time for a single update as an agent is only concerned with its local view; however, communications increase as agents need to transmit changes to others with which they have dependencies.

Acknowledgments

References

[Beck and Wilson, 2007] J. C. Beck and N. Wilson. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, 28:183–232, 2007.

[Boddy *et al.*, 2005] Mark Boddy, Bryan Horling, John Phelps, Robert Goldman, Regis Vincent, C. Long, and Bob Kohout. C_TAEMS language specification v. 1.06. October 2005.

[Bryce *et al.*, 2008] Daniel Bryce, Mausam, and Sungwook Yoon. Workshop on a reality check for planning and scheduling under uncertainty. http://www.ai.sri.com/b̃ryce/ICAPS08-workshop.html, April 2008.

[Decker, 1996] Keith Decker. TAEMS: A framework for environment centered analysis & design of coordination mechanisms. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 16, pages 429–448. Wiley Inter-Science, 1996.

[Fox *et al.*, 2006] Maria Fox, Richard Howey, and Derek Long. Exploration of the robustness of plans. In *Proceedings of AAAI-06*, 2006.

[Hiatt *et al.*, 2008] Laura M. Hiatt, Terry L. Zimmerman, Stephen F. Smith, and Reid Simmons. Reasoning about executional uncertainty to strengthen schedules. In *Proceedings of the Workshop on A Reality Check for Planning and Scheduling Under Uncertainty (held in conjunction with ICAPS-08)*, 2008.

[Mausam and Weld, 2006] Mausam and Daniel S. Weld. Probabilisitic temporal planning with uncertain durations. In *Proceedings of AAAI-06*, Boston, MA, July 2006.

[McKay *et al.*, 2000] K.N. McKay, T.E. Morton, P. Ramnath, and J. Wang. Aversion dynamics' scheduling when the system changes. *Journal of Scheduling*, 3(2), 2000.

[Morris *et al.*, 2001] Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of IJCAI*, pages 494–502, 2001.

[Musliner *et al.*, 2007] David J. Musliner, Jim Carciofini, Edmund H. Durfee, Jianhui Wu, Robert P. Goldman, and Mark S. Boddy. Flexibly integrating deliberation and execution in decision-theoretic agents. In *Proceedings of the 3rd Workshop on Planning and Plan Execution for Real-World Systems (held in conjunction with ICAPS-07)*, September 2007.

[Onder *et al.*, 2006] Nulifer Onder, Garrett C. Whelan, and Li Li. Engineering a conformant probabilistic planner. *Journal of Artificial Intelligence Research*, 25:1–15, 2006.

[Policella *et al.*, 2006] Nicola Policella, Amedeo Cesta, Angela Oddi, and Stephen F. Smith. From precedence constraint posting to partial order schedules. a CSP approach to robust scheduling. *AI Communications*, 20(3):163–180, 2006.

[Smith and Weld, 1998] Daniel E. Smith and Daniel S. Weld. Conformant graphplan. In *Proceedings of AAAI-98*, Madison, WI, 1998.

[Smith *et al.*, 2007] Stephen F. Smith, Anthony Gallagher, Terry Zimmerman, Laura Barbulescu, and Zachary Rubinstein. Distributed management of flexible times schedules. In *Proceedings of AAMAS-07*, May 2007.

[Wagner *et al.*, 2006] Thomas Wagner, Anita Raga, and Victor Lesser. Modeling uncertainty and its implications to sophisticated control in TAEMS agents. *Autonomous Agents and Multi-Agent Systems*, 13(3):235–292, 2006.

[Xuan and Lesser, 1999] Ping Xuan and Victor Lesser. Incorporating uncertainty in agent commitments. In *Proceedings of ATAL-99*, 1999.

[Younes and Simmons, 2004] Håkan L. S. Younes and Reid G. Simmons. Solving generalized semi-markov decision processes using continuous phase-type distributions. In *Proceedings of AAAI-04*, San Jose, CA, 2004.