Solving Large TÆMS Problems Efficiently by Selective Exploration and Decomposition

Jianhui Wu and Edmund H. Durfee
EECS Department, University of Michigan
Ann Arbor, MI 48109 USA
jianhuiw, durfee@umich.edu

ABSTRACT

TÆMS is a hierarchical modeling language capable of representing complex task networks with intra-task uncertainties and inter-task dependencies. The uncertainty and complexity of the application domains represented in TÆMS models often lead to very large state spaces, which push the need to design efficient solution algorithms for TÆMS problems. In this paper, we present a solver that integrates selective state space search techniques with state space decomposition techniques. Our experiments demonstrate that the solver can find an (approximately) optimal solution much faster than prior approaches.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE; I.2.8 [**ARTIFICIAL INTELLIGENCE**]: Control Methods, and Search

General Terms

ALGORITHMS

Keywords

TÆMS model, MDP, selective exploration, informed unrolling, decomposition, mission phasing

1. INTRODUCTION

TÆMS task networks [4] have been successfully used to model complex applications, such as information gathering problems [9]. One way to find an optimal solution to a single-agent problem represented in a TÆMS model is to *unroll* the states implicitly defined in the TÆMS model into a Markov Decision Process (MDP) [9, 5], and then build a policy using an MDP policy formulation algorithm, such as the Bellman backup and value iteration algorithms [8]. However, this solution approach quickly becomes impractical as the size and complexity of the TÆMS model increase. For example, in a moderately complex TÆMS model with $m = 20$ applicable methods where each method has $o = 9$ different possible outcomes and $d = 8$ methods can be carried out over one execution, there are about $(o \times m)^d \approx 1.1 \times 10^{18}$ possible states for the agent to generate and reason over.

In this work, we are interested in a class of challenging problems where the agent has a finite amount of "think time" during which to build and solve an MDP for its TÆMS model, after which the agent executes the policy of the MDP it has solved. Not surprisingly, the time limitations could mean that the agent cannot generate and search the entire state space, in which case the agent will find a policy for only the portion of the state space it does generate. In executing this policy, if the agent reaches a state that is at the edge of its generated state space, the policy does not provide an action choice for this state or any subsequent state. Our goal is to provide mechanisms that integrate selective state space search (overviewed in Section 1.1) with state space decomposition (overviewed in Section 1.2) to help the agent build the best (partial) policy it can in its limited time.

1.1 Selective State Space Search

One of the most commonly used techniques for handling a large state space is to selectively expand and explore the space, which might generate an (approximately) optimal policy while avoiding exhaustive enumeration of all possible states. Barto, Bradtke, and Singh presented a real-time dynamic programming (RTDP) algorithm that performs successive trials on the environment with uncertainties [1]. Each trial starts at the initial state and ends at the goal state. In each trial, value updates are only performed on the states actually visited in that trial. Dean, Kaelbling, Kirman, and Nicholson proposed an *envelope* algorithm, which starts with an initial policy, and a restricted state space (or *envelope*) that contains a path from the starting state to the goal state [2]. It then gradually extends that envelope to include more states, and computes new policies. Given more computational time, the algorithm will compute a more complete partial policy. LAO* is another heuristic search algorithm that is similar to the envelope algorithm but emphasizes the optimality of its solution [3]. It uses an admissible heuristic to direct the search in an analogous way to the well-known A* search algorithm.

Unlike a classical dynamic programming algorithm (e.g., value iteration [8]) that evaluates the complete state space and finds an optimal policy for every state, these heuristic search algorithms can often significantly reduce effort spent on states that will never be reached when following an optimal policy from the initial state. In problems with

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

978-81-904262-7-5 (RPS) ©2007 IFAAMAS

large state spaces, this is an obvious improvement over dynamic programming since the heuristic search might find an (approximately) optimal solution from an initial state by considering many fewer states.

In this work, we design a policy formulation algorithm, named *informed unroller* (IU), by adopting a similar idea of selectively exploring the state space implicitly defined in the TÆMS model. Unlike previous work, TÆMS problems generally lack explicit "goal" states. Instead, they are typically finite-horizon problems and might have tens of thousands of "terminal" states (at the horizon), and so many of the previous "goal-oriented" policy formulation approaches (e.g., the envelope algorithm) are not directly applicable. Furthermore, the TÆMS state space might be so large that even an algorithm that can perfectly expand the space cannot (within "think time" limits) visit every state reachable by the optimal policy. For the aforementioned example, the number of states reachable by an optimal deterministic policy is approximately $o^d \approx 4.3 \times 10^7$, which is much smaller than the total number of states, but it might still overburden the planner when computational time is limited.

To cope with these issues, our IU algorithm builds a policy for its selectively-generated subset of the reachable states, and then generates deterministic plans, which start at the edges of the expanded state space and end at the finite horizon (rather than at a particular goal state), using a fast greedy algorithm. As we will show later, these plans can be used to estimate the values of the edge states for building an (approximately) optimal (partial) policy that leads to high-quality terminal states, and also provide (suboptimal incomplete) policies to follow when execution runs past the edge states.

1.2 State Space Decomposition

A lingering problem, even when unrolling from the initial state can be done selectively, is that the agent will generally expend much more effort in the state space nearer to the initial state, and much less in exploring states that might be reached later. To better balance the state-space expansion over the entire timeframe, our work also exploits problem structure to decompose the state space temporally into more manageable pieces.

Much research work has been devoted to speeding up planning by breaking problems into sequences of subproblems. Porteous, Sebastia and Hoffmann introduced "landmarks", which are facts that must be true at some point in every valid solution plan, and presented algorithms for automating the process of finding and using such landmarks to decompose a given planning task into several smaller subtasks [7]. Parr adopted heuristic methods to decompose the environments of robot navigation problems into weakly-coupled regions [6]. These decomposition techniques break a large complex problem into multiple independent or approximately independent pieces and can often gain significant speedup when computing policies (or plans).

Let us consider the example again, where if we can evenly decompose the TÆMS model into $n = 4$ phases, then the total number of states is reduced to about $n \times (\frac{o \times m}{n})^{\frac{d}{n}} \approx 8.1 \times 10^3$ states. Clearly, a favorable TÆMS decomposition can significantly reduce computational requirements. Unfortunately, making good decomposition decisions is nontrivial in many situations because the outcome of a task might affect the outcomes of other tasks, and the execution time windows of tasks might overlap. Our decomposition technique, named *mission phasing*, aims to address these challenges.

To summarize, we argue that, by breaking a large TÆMS model into multiple sub-models, and focusing exploration on important regions of a (phase's) state space, our TÆMS solver can find an (approximately) optimal policy much faster than prior approaches. The rest of the paper is organized as follows. Section 2 introduces background knowledge about TÆMS models and MDPs. The two main components of our solver—the informed unroller (IU) and mission phasing—are presented in Sections 3 and 4, respectively. We present empirical results in Section 5, where we evaluate our techniques in the Coordinator domain [5]. Finally, Section 6 summarizes our work and outlines future directions.

2. BACKGROUND

2.1 TÆMS Models

In a TÆMS task network [4], leaf nodes represent *methods* (primitive actions). A method might have multiple possible outcomes with different durations and qualities. An internal node in the network is a *task*, which is associated with a *quality accumulation function* (QAF) (such as *sum*, *min*, and *max*) that describes how the quality of the task's subtasks can be used to calculate the quality of the task itself. A node in the TÆMS task network might be constrained by *release time* (earliest possible start time) and *deadline*, and the temporal constraint applies recursively downward: a subtask inherits the most restrictive of its own constraints and those of its parent(s). A method violating its temporal constraint will yield zero quality. The TÆMS task network also supports modeling dependencies among tasks, such as enablement, disablement, facilitation, and hindrance, with *non-local effect* (NLE) links [4]. An NLE indicates a task interrelationship where the execution of some task will have a positive or negative effect on the quality or duration of another task. Given a problem represented in a TÆMS model, the goal is to find a policy maximizing the expected quality at the root node of the model.

Figure 1 shows an example TÆMS model with 42 nodes, which we will use to illustrate our techniques in the subsequent sections. The temporal constraint of a node is represented in the format $[t_a, t_b]$, where t_a and t_b represent the release time and deadline of the node, respectively. The description of the temporal constraint (if there is one) is placed under the node's name. In this example, a single method might have multiple possible outcomes. The distribution on the quality of a method outcome is depicted in the format $Q : a \pm b$ when the method will always succeed, and is depicted in the format $Q : a \pm b\ (p)$ when the method might fail (i.e., yielding zero quality) with probability $1-p$. When the method succeeds, it has probability 0.5, 0.25, and 0.25 of yielding the expected quality a, minimum quality $a-b$, and maximum quality $a+b$, respectively. The duration distribution of the method outcome is depicted in a similar format $D : a' \pm b'$, which says that the method will complete at its expected duration a', minimum duration $a'-b'$, and maximum duration $a'+b'$ with probability 0.5, 0.25, and 0.25, respectively. There is an *enablement* NLE in this example, which indicates that task *Ctask4b* can yield positive quality only when task *Ctask3a* has achieved positive quality first.

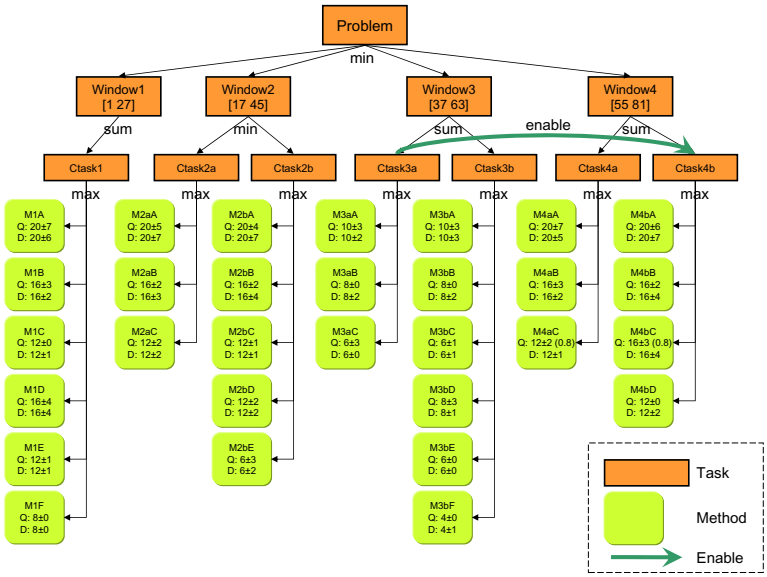

Figure 1: An Example TÆMS Model.

2.2 Markov Decision Processes

Markov Decision Processes (MDPs) provide a good framework to compute optimal policies in uncertain environments, and thus are well suited for TÆMS problems having uncertainties in qualities and durations of method outcomes. We now give a brief introduction to the MDP, and postpone the discussion of the procedure for formulating a TÆMS model into an MDP until Section 3.

An MDP can be defined as a four-tuple $\langle S, A, P, R\rangle$, where S is a finite state space, A is a finite action space, $P = \{p_{s,a,s'}\}$ is the state transition probability where $p_{s,a,s'}$ is the probability that the agent reaches state s' if it executes action a in state s, and $R = \{r_{s,a}\}$ is the reward function where $r_{s,a}$ is the reward that the agent receives if it executes action a in state s. MDPs have been well studied and there exist relatively simple algorithms which build optimal policies mapping states to action choices in polynomial time (in the size of the state space) [8].

3. INFORMED UNROLLER

3.1 Unrolling TÆMS Models into MDPs

In this subsection, we recap the approach for fully unrolling a TÆMS model into an MDP [9, 5], whose solution policy will maximize the expected quality at the root of the TÆMS model, because our subsequent technique extends these foundations. In the TÆMS MDP, a state is defined as $\langle t, \mathbb{M}\rangle$, where t is the current time, and $\mathbb{M}$ is a set of method outcomes $\{o\}$. Each outcome $o = \langle m, \tau, d, q\rangle$ stores the information of its execution method m, start time τ, duration d, and quality q. Such a state representation assures that the conditional probability distribution of future states, given the present state and all past states, depends only upon the current state and not on any past state, i.e., the Markov property holds. When an agent executes a method m_i, which has probability p_i of taking d_i time steps and yielding quality q_i, in state $\langle t, \mathbb{M}\rangle$, the agent will reach the successor state $\langle t + d_i, \mathbb{M} \bigcup \{\langle m_i, t, d_i, q_i\rangle\}\rangle$ with probability p_i.

The unrolling procedure is summarized in Procedure 1. It is similar to a breadth-first search. At each loop, it pops the top state from the queue *openList* (line 4), and finds the set of applicable methods that can be executed in that state by examining their temporal and NLE constraints (line 5). It then generates successor states for each applicable method according to state transition probabilities mentioned above (line 6), updates the MDP (line 8), and puts newly generated, non-terminal successor states (those for times prior to the problem horizon) at the end of *openList* (line 10).

Procedure 1 mdp = **unroll**($model$)

```
1: Initialize empty mdp, initState
2: openList ← {initState}
3: repeat
4:    state ← dequeue(openList)
5:    for all m ∈ applicable-methods(state, model) do
6:       succs ← successor-states(state, m, model)
7:       for all succ ∈ succs do
8:          mdp ← update(state, m, succ, mdp)
9:          if succ is not a terminal state then
10:            enqueue(succ, openList)
11:         end if
12:      end for
13:   end for
14: until openList is empty
15: return mdp
```

The unrolling stops when *openList* is empty. At this point, the TÆMS model is fully unrolled into a finite horizon MDP. Each MDP terminal state captures method outcomes of a possible execution trace of the TÆMS problem, and the reward of the state is the quality of the root node of

the TÆMS model given that execution trace. On the other hand, the internal states in the MDP state space always have zero reward, since all activities will be evaluated at the end of the execution (i.e., in terminal states). Given state transition probabilities and rewards, the Bellman backup algorithm can be used to find an optimal policy in time linear to the number of states [8].

3.2 Informed Unrolling

Many application domains, such as the Coordinator domain [5], are complicated, and often lead to very large state spaces. Full (uninformed) unrolling might not be practical for these complex problems, especially when computational time is limited. A straightforward approach to meet this computational limitation challenge is to stop the unrolling process when runtime reaches a pre-specified limit *tlim*. Thereafter, rewards are computed for edge states of the partially unrolled state space based upon (incomplete) execution traces, and the Bellman backup algorithm is used to compute a policy for the partially explored state space.

However, this straightforward approach suffers from two shortcomings. First, the limited computational time only allows a subset of states to be expanded; the unroller described in Procedure 1 that implements an (uninformed) breadth-first expansion will expand all paths to equal (partial) depth regardless of the likelihood that the agent will traverse the path. Second, the approach does not specify actions to take after the agent executing a policy goes beyond the unrolled state space. That is to say, the agent has no-ops for its policy when it reaches an edge state, which is unlikely to be effective in accruing quality.

To address the first shortcoming mentioned above (the second will be discussed later), our informed-unroller (IU) algorithm prioritizes the queue of states waiting to be unrolled based on an estimate of the likelihood that the state would be encountered when executing an optimal policy from the initial state. In other words, the IU algorithm emphasizes the exploration of the state space that is likely to be reached by the optimal policy, while ignoring other states, to yield a better policy when time limit is reached.

Because the probability of reaching a state is dependent on the policy, the IU algorithm intersperses policy formulation (using the Bellman backup algorithm) with unrolling. It should be noted that, although the Bellman backup algorithm is fast, i.e., its runtime is linear to the number of states, formulating a policy at each state expansion step is generally too costly. The IU algorithm recomputes a policy and reorders states waiting to be expanded less frequently. Specifically, to balance the benefits of unrolling more states and of being selective in the unrolling direction, we choose a heuristic to sort the queue of states waiting to be expanded (currently, when the size of the queue has doubled since the last sorting).

The details of the IU algorithm are shown in Procedure 2, where lines $6-15$ are similar to the aforementioned uninformed unroller. Line 17 evaluates edge states of the partially unrolled state space by building and evaluating greedy plans starting at the edge states; the details will be discussed later (in Procedure 3). Line 18 solves the MDP by using Bellman backup to derive a policy, and line 19 computes the probability of the agent reaching each edge state when executing the derived policy. Edge states waiting to be expanded are sorted in line 20 according to their probabilities of being reached, where the state with the highest reaching probability is placed at the top of the queue, i.e., in a highest-probability-first (HPF) manner.

Procedure 2 mdp = **informed-unroll**($model$, $tlim$)

```
 1: Initialize empty mdp, initState
 2: preSortSize ← 10
 3: K ← 2
 4: openList ← {initState}
 5: repeat
 6:   state ← dequeue(openList)
 7:   for all m ∈ applicable-methods(state, model) do
 8:     succs ← successor-states(state, m, model)
 9:     for all succ ∈ succs do
10:       mdp ← update(state, m, succ, mdp)
11:       if succ is not a terminal state then
12:         enqueue(succ, openList)
13:       end if
14:     end for
15:   end for
16:   if size(openList) ≥ K × preSortSize then
17:     mdp ← eval-edge-states(mdp, model)
18:     policy ← solve(mdp)
19:     prob ← compute-reaching-prob(mdp, policy)
20:     openList ← sort(openList, prob)
21:     preSortSize ← K × preSortSize
22:   end if
23: until openList is empty or runtime reaches tlim
24: return mdp
```

To build an intermediate policy that can properly guide the IU exploration direction, potential activities in the future should be considered when evaluating edge states. An exact evaluation of an edge state requires a full lookahead, and is obviously impractical. Instead, the IU algorithm computes the heuristic value of an edge state by adopting a fast greedy algorithm to build a plan starting at that edge state.[1] Unlike a policy considering all possible eventualities, a plan only represents a particular execution trace. Specifically, a plan is composed of deterministic copies of TÆMS methods. Each deterministic method (deterministic because it has exactly one outcome) corresponds to one actual TÆMS method; its duration is the maximum duration of the TÆMS method and its quality is the expected quality of the TÆMS method.

These greedy plans serve two purposes. First, given an edge state and a greedy plan starting at it, we can predict the (unique) terminal state at the end of the execution of the plan. The IU algorithm uses the quality of this predicted terminal state as the heuristic value of that edge state. Second, these plans can tell the agent what to execute (though maybe suboptimally) after the agent reaches the edge of the unrolled space during execution. Recall that a deterministic method uses the maximum duration of its corresponding TÆMS method, and so the plan composed of deterministic methods is always executable, i.e., the agent can simply wait if a method completes before its maximum duration.

The details of generating a greedy plan are presented in Procedure 3. Given an edge state, the algorithm starts with

[1]In some domains, problems come with some (suboptimal but good) initial solutions, and we can use these solutions to evaluate edge states instead.

an empty plan (line 1); it then gradually augments the plan by myopically adding deterministic methods until reaching the problem horizon (lines 2 – 6).

Procedure 3 [*plan*, *state*] = **greedy-plan**(*state*, *model*)

1: Initialize an empty *plan*
2: **while** *state* is not a terminal state **do**
3: $\overline{method} \leftarrow$ **best-method**(*state*, *model*)
4: *plan* $\leftarrow$ **add**($\overline{method}$, *plan*)
5: *state* $\leftarrow$ **successor-state**(*state*, $\overline{method}$)
6: **end while**
7: **return** *plan*, *state*

The heuristic adopted in determining which deterministic method to insert into the plan is described in the *best-method* algorithm shown in Procedure 4, where argmax stands for the arguments of the maximum, i.e., the set of arguments for which the value of the given expression attains its maximum value. The best-method algorithm begins by checking applicable deterministic methods (line 1). If none of methods can be executed, it returns a *wait* method that will let the agent idle one time step (lines 2 – 4). Otherwise, given a set of applicable methods, the algorithm chooses the method(s) that can lead to the successor state(s) with the highest quality (line 5). If multiple deterministic methods tie, the heuristic picks the method(s) located in an unexplored branch where an unexplored branch refers to a sub-model rooted at a TÆMS node that is directly under a *min* QAF and has zero quality so far (line 6). If tied again, the methods are, in turn, filtered by their deadlines (line 7), success probabilities (i.e., the probability of successfully completing a method while ignoring its temporal constraints) (line 8), and quality densities (i.e., the quality divided by the duration) (line 9).

Procedure 4 $\overline{method}$ = **best-method**(*state*, *model*)

1: $\hat{m} \leftarrow$ **applicable-det-methods**(*state*, *model*)
2: **if** $\hat{m}$ is empty **then**
3: **return** *wait*
4: **end if**
5: $\hat{m} \leftarrow \text{argmax}_{m \in \hat{m}}$ **quality**(**successor-state**(*state*, *m*), *model*)
6: $\hat{m} \leftarrow \text{argmax}_{m \in \hat{m}}$ **is-in-unexplored-branch**(*state*, *m*, *model*)
7: $\hat{m} \leftarrow \text{argmax}_{m \in \hat{m}}$ $-1 \times$**deadline**(*m*)
8: $\hat{m} \leftarrow \text{argmax}_{m \in \hat{m}}$ **method-success-prob**(*m*)
9: $\hat{m} \leftarrow \text{argmax}_{m \in \hat{m}}$ **qual-density**(*m*)
10: **return** **one-of**($\hat{m}$)

We conclude this section by comparing several TÆMS solution algorithms on the example problem shown in Figure 1, including our IU algorithm, the uninformed unroller (which is named as FIFO in the following discussion because it manipulates *openList* in a first-in-first-out manner), HPF-42 algorithm, and HPF-42-BM algorithm. The HPF-42 algorithm intersperses policy formulation with unrolling and sorts states waiting to be expanded in a highest-probability-first manner like the IU algorithm, but it adopts a very fast (but coarse) way of assuming identical heuristic values (i.e., a constant 42) for edge states when computing policies to guide further exploration. The unrolling procedure of the HPF-42-BM algorithm is the same as the HPF-42 algorithm.

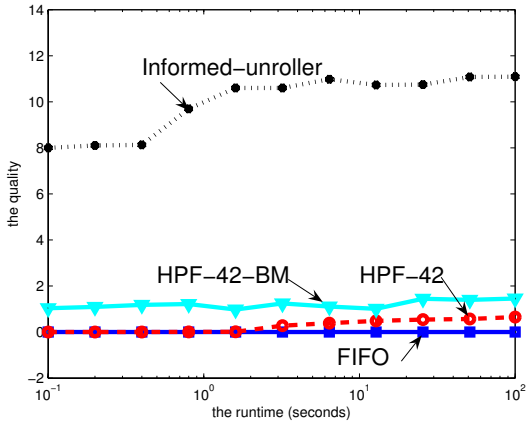

Figure 2: Anytime performance of four TÆMS solvers on the example.

However, unlike the other three algorithms that plan before execution, it assumes that the agent has limited computational capability during execution to adopt the best-method algorithm (described in Procedure 4) to find a method to execute (instead of being idle like HPF-42 and FIFO) when it is beyond the policy boundaries.

Figure 2 shows anytime performance of these four solvers on our example problem.[2] We can see that the IU algorithm strongly dominates the others. It finds an approximately optimal solution policy within 10 seconds.[3] In contrast, even given 100 seconds, the exploration of the FIFO algorithm still cannot reach a state with a positive quality. HPF-42 is slightly better than FIFO because its selective exploration can let it explore deeper. In addition, HPF-42-BM achieves higher quality than HPF-42 since it can decide a method to execute when it is beyond the policy boundaries. However, HPF-42-BM is still much worse than the IU algorithm, because its selective exploration is based upon a very coarse heuristic and often results in a low quality at the *Window2* node of the example TÆMS model.

To better understand behaviors of these solvers, in Figure 3 we show a summary of the explored state spaces when the unrolling time is limited to 50 seconds for each solver. We can see that FIFO unrolls the largest state space, but because of its breadth-first style search, its exploration depth toward the finite horizon is the shallowest. HPF-42 (as well as HPF-42-BM) explores a smaller state space because it spends some of its time in computing policies to guide the further exploration direction, but since HPF-42 prefers to expand states with higher probability of being reached and ignores many states with low or zero reaching probabilities, it explores deeper than FIFO. IU explores the fewest states because it spends much of its computational time in building greedy plans at edge states, but it is smarter in selecting the exploration direction since it adopts a more accurate heuristic in evaluating edge states.

[2]We cannot show the optimal policy because none of the solvers can explore the entire state space within 24 hours.
[3]In the example problem, the "pure" greedy-plan algorithm (without unrolling) yields zero quality.

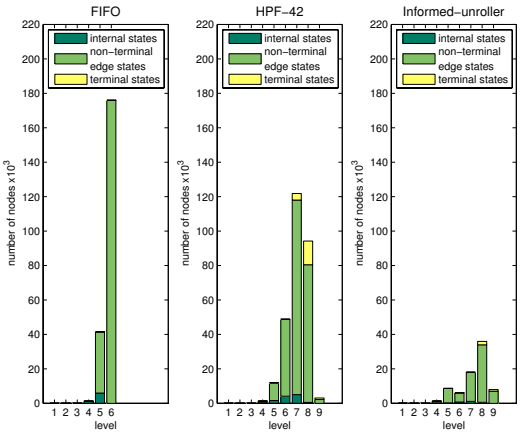

Figure 3: Summary of state spaces explored by the TÆMS solvers on the example ($tlim$ = 50 seconds).

4. MISSION PHASING

As introduced in Section 1.2, implementing decomposition might further improve the policy given limited computational time for a TÆMS problem, but there are typically two challenges in TÆMS model decomposition. First, time windows (temporal constraints) of tasks might overlap, e.g., the release time of a subsequent task can be earlier than the deadline of the current task. Second, there might be dependencies between tasks. In this section, we present our mission phasing algorithm for distributing the unrolling and policy formulation effort more evenly across the temporally-extended state space while addressing these issues. Mission phasing (approximately) decomposes the given TÆMS problem into an ordered list of independent phases to reduce the number of states to be explored, and adopts the IU algorithm as its inner-phase solver to selectively explore the state space in each phase.

It should be emphasized that mission phasing is not for finding the optimal policy. Actually, in some cases, the solution policy from the mission phasing algorithm is suboptimal even given unlimited computational time. The underlying reason is that, due to the nonlinearities in some QAFs (such as min, and max), maximizing solution quality in every phase does not necessarily maximize the solution quality over the original (overall) TÆMS model. However, with cautious decomposition and approximation, computing an optimal policy for each phase and combining them together can often result in an approximately optimal global policy; our technique exploits this property.

The mission phasing algorithm is outlined in Procedure 5. It begins by examining the given TÆMS model to locate nodes that will be used as the root nodes of phase models (line 2). The heuristic we adopt is to choose nodes at the deepest level of the TÆMS model where some nodes have specified release times and/or deadlines (because the decomposition will attempt to adjust these temporal constraints). In cases that the resulting phases are too large, nodes at a deeper level are selected. For each of the selected nodes, the algorithm then constructs a phase corresponding to the sub-model rooted at that node (line 5). After that, phases are sorted according to deadlines of root-nodes of their models

Procedure 5 $phases$ = **mission-phasing**($model$, $tlim$)

1: Initialize empty $phases$
2: $nodes$ = **find-phase-root-nodes**($model$)
3: **for all** $node \in nodes$ **do**
4: Initialize $phase$
5: $phase.model \leftarrow$ **sub-model**($node$, $model$)
6: $phases \leftarrow$ **add**($phase$, $phases$)
7: **end for**
8: $phases \leftarrow$ **sort-phases**($phases$)
9: $phases \leftarrow$ **merge-nle-linked-phases**($phases$)
10: $phases \leftarrow$ **merge-small-phases**($phases$)
11: $phases \leftarrow$ **evenly-split-temporal-overlap**($phases$)
12: $phases \leftarrow$ **solve-phases**($phases$, $tlim$)
13: **return** $phases$

to get ready for merging (line 8). In order to maintain dependencies among tasks, for each NLE, the algorithm merges all phases between the phase containing the NLE's source node and the phase containing the NLE's destination node (line 9). In a highly (NLE) connected problem, this might lead to a large phase. However, we can still rely on the IU algorithm to find a good solution inside that phase. The algorithm also attempts to merge several small phases into a moderate-sized phase, e.g., with less than 10^4 states (line 10). This is because a moderate-sized phase can still be solved efficiently by the IU algorithm while it can maintain (most of) the temporal constraints of the nodes within the merged phase. After merging, the algorithm evenly splits time window overlaps between the neighboring phases (line 11). Finally, the solve-phases procedure described in Procedure 6 is invoked to derive (partial) policies for each phase.

Procedure 6 $phases$ = **solve-phases**($phases$, $tlim$)

1: $schedule \leftarrow$ **deliberation-schedule**($phases$, $tlim$)
2: **for all** $phase \in phases$ **do**
3: $\widetilde{tlim} \leftarrow$ **computational-time**($phase$, $schedule$)
4: $phase.mdp \leftarrow$ **informed-unroll**($phase.model$, $\widetilde{tlim}$)
5: $phase.policy \leftarrow$ **solve**($phase.mdp$)
6: **end for**
7: **return** $phases$

By breaking the problem into multiple phases, we can distribute limited computational time more selectively over the entire timeframe. In this work, we use a simple deliberation scheduling strategy. The computational time allocated to a phase is proportional to the number of methods in that phase. The IU algorithm then follows the deliberation schedule to selectively unroll state spaces of each phase.

Recall that a challenge of the TÆMS model is that QAFs might be nonlinear. Especially, a min QAF returns the minimum quality of subtasks. That is to say, when multiple phases are directly under the same min QAF, if any phase fails, all phases become valueless. To reduce such risks, the mission phasing algorithm adopts a strategy of penalizing every failure state (i.e., every edge state with zero quality) with a large negative reward when the phase is directly under a min QAF. Then the Bellman backup algorithm is used to build a policy based upon the calibrated rewards.

Procedure 5 adopts a simple way of evenly splitting overlapping time intervals of neighboring phases. This is a natural choice when lacking knowledge about phases. However,

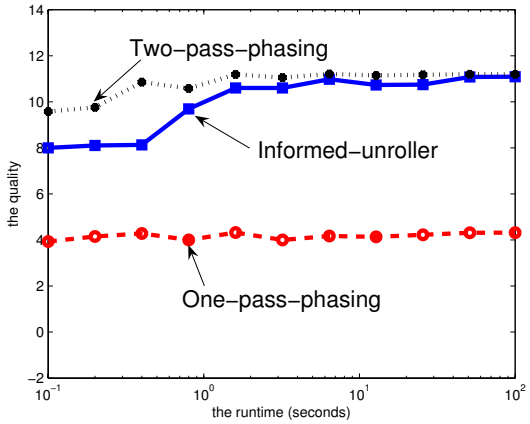

Figure 4: Anytime performance of the IU, and mission phasing algorithms on the example.

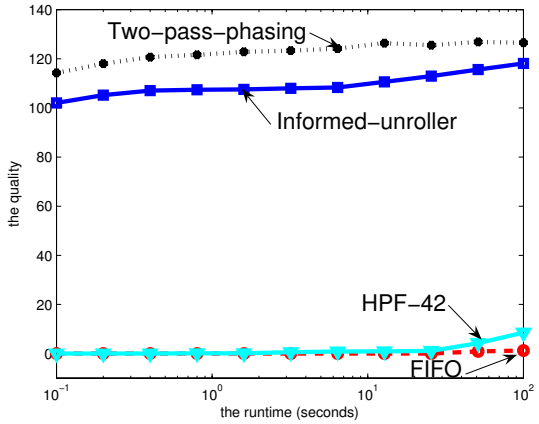

Figure 5: Average anytime performance of four TÆMS solvers on 100 test problems.

it might be too risky in some cases, and we might better distribute the overlapping time by carrying out mission phasing more than once. The two-pass mission phasing algorithm is presented in Procedure 7. It spends 30% of the computational time in the first pass to collect preliminary information about phases, and then uses this knowledge to improve the split in the second pass. When re-splitting the temporal overlap between two phases, the algorithm adopts different heuristics for different types of the QAF at the their parent node (if they have the same parent). For a *sum* QAF, more time steps are distributed to the phase with higher quality density. The distribution for a *max* QAF is to give all time steps to the phase with higher quality. For a *min* QAF, more time steps are allocated to the phase that impairs the overall quality. In other cases, the algorithm evenly splits the overlap in the same way as before.

Procedure 7 $phases$ = **two-pass-phasing**($model$, $tlim$)

1: $tlim' \leftarrow tlim \times 30\%$
2: $phases \leftarrow$ **mission-phasing**($model$, $tlim'$)
3: $phases \leftarrow$ **re-split-temporal-overlap**($phases$)
4: $phases \leftarrow$ **solve-phases**($phases$, $tlim - tlim'$)
5: **return** $phases$

We end this section by comparing the anytime performance of the IU, one-pass mission phasing and two-pass mission phasing algorithms on our example problem. The results are shown in Figure 4. The performance of the one-pass mission phasing algorithm is poor. This is because evenly splitting overlapping time is a bad choice for the problem, which causes the quality of the phase rooted at the *Window2* node to be much lower than the others. In contrast, the two-pass mission phasing algorithm performs much better because it can analyze the results of its first pass and re-split temporal overlaps in a better way, i.e., allocating more time steps for the *Window2* phase. Additionally, we can see that, as expected, the two-pass mission phasing algorithm finds an approximately optimal solution faster than the IU algorithm, which is because the total number of states is much smaller when using phasing.

5. EXPERIMENTAL RESULTS

In this section, we give a preliminary empirical evaluation on the efficacy of our solver in the Coordinator domain [5]. We begin with 16 *3-agent* and 22 *4-agent* Coordination problems (not devised by us), and divide them into 136 single-agent TÆMS problems by ignoring NLEs across agents (while local NLEs are untouched). These single-agent TÆMS models represent initial views of agents. We randomly pick up 100 out of these 136 single-agent problems as our test problems. The complexities of these problems are quite varied. The smallest problem we used has only 26 nodes in its TÆMS model, but the most complex one has 155 nodes. This provides a nice test set to evaluate TÆMS solvers under various conditions.

Figure 5 compares the anytime performance of the four aforementioned TÆMS solvers. Its x-axis indicates the runtime, and its y-axis represents the average quality of the solution policies on the 100 test problems. We can see that these results are similar to the previous results on our example problem. FIFO achieves negligible quality on average because it usually cannot unroll deeply enough to reach states with positive quality even given 100 seconds. HPF-42 returns a zero-quality policy as well when computational time is very limited. As the runtime increases, its solution quality gradually increases. This matches our expectation that HPF-42 can unroll a narrower but deeper state space than FIFO. IU is much better than FIFO and HPF-42. When computational time is limited, its unrolled space cannot (directly) reach states with positive quality either. However, it builds and adopts greedy deterministic plans at the edges of the partially unrolled state space; these plans can often result in good execution traces (to the problem horizon). Similarly to HPF-42, IU can improve its solution when given more computational time. The two-pass mission phasing algorithm uses the IU algorithm as its intra-phase solver, and uses the phasing decomposition technique to further speed up the planning. Its performance is, on average, better than the others on our test problems.

To gain a better understanding of the improvement of the two-pass mission phasing algorithm over the IU algorithm, we now compare their performance in detail when compu-

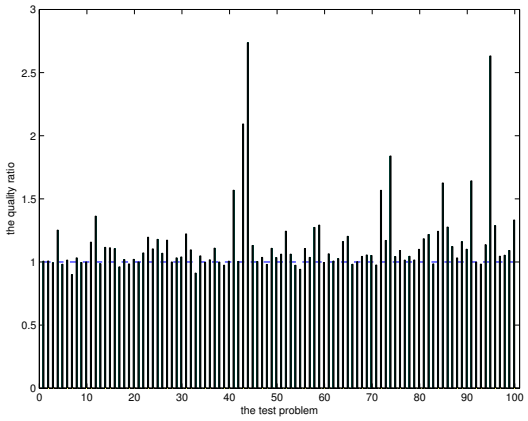

Figure 6: The quality ratio of the two-pass mission phasing solution to the IU solution on 100 test problems ($tlim$ = 1 second).

tational time is limited to 1 second for each test problem. As shown in Figure 6, the two-pass mission phasing algorithm outperforms the IU algorithm in most cases (77%). It achieves a quality 173% higher than the IU algorithm in the best case, and achieves a quality only 9% less than the IU algorithm in the worst case. That is to say, the two-pass mission phasing algorithm is quite an effective and dependable algorithm in (at least) the Coordinator domain despite the approximations and heuristics it uses.

6. CONCLUSIONS AND FUTURE WORK

We have presented an efficient and effective policy formulation solver for problems represented in TÆMS models. Our experiments showed that the solver can find an approximately optimal policy much faster than prior approaches. Our results highlighted the ability of our techniques to cope with very limited "think time"; we are also using these techniques in multi-agent settings so that agents can quickly assess the impact of alternative inter-agent commitments on their local policies.

In the future, we plan to improve our solver in several directions. First, observing the fact that an autonomous agent might have additional computational time when it operates, we would like to design an incremental informed unrolling algorithm which can utilize the available computational time during execution to improve its solution quality. Second, our current mission phasing algorithm copes with NLEs by simply merging phases linked by NLEs, which might result in large phases when tasks are highly dependent. One possible way to avoid this is to use "commitments" in a similar way to coordination decision-making among multiple agents [5]. Once an NLE commitment is made, the phase that includes the NLE source node should try its best to trigger the NLE, and the phase having the NLE destination node can safely assume that the NLE will hold. Not surprisingly, using such commitments well can separate dependent tasks. Finally, we will look into more sophisticated deliberation scheduling algorithms (e.g., [10]) that can better manage the distribution of the limited computational time among multiple intra-phase unrolling procedures. We aim to improve our solver so that it will emphasize the exploration of important regions of the state space through not only the intra-phase heuristic search but by across-phase sophisticated deliberation scheduling.

7. ACKNOWLEDGEMENT

The techniques presented in this work build upon code and concepts developed by a larger research team (of which we are part) centered at Honeywell Labs. We appreciate the contributions to this work of that team (including Dave Musliner, Robert Goldman, John Phelps, and Mark Boddy, among many others).

This material is based upon work supported in part by DARPA/IPTO and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

8. REFERENCES

[1] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72(1-2):81–138, 1995.

[2] T. Dean, L. P. Kaelbling, J. Kirman, and A. E. Nicholson. Planning under time constraints in stochastic domains. *Artif. Intell.*, 76(1-2):35–74, 1995.

[3] E. A. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.*, 129(1-2):35–62, 2001.

[4] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

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

[6] R. Parr. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the 14th Conference in Uncertainty in Artificial Intelligence*, pages 422–430, 1998.

[7] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning*, pages 37–48, 2001.

[8] M. L. Puterman. *Markov Decision Processes.* John Wiley & Sons, New York, 1994.

[9] T. Wagner, A. Raja, and V. Lesser. Modeling Uncertainty and its Implications to Sophisticated Control in TAEMS Agents. *Autonomous Agents and Multi-Agent Systems*, 13(3):235–292, November 2006.

[10] J. Wu and E. H. Durfee. Mathematical programming for deliberation scheduling in time-limited domains. In *Proceedings of the 5th International Joint Conference on Autonomous Agents adn Multiagent Systems*, pages 874–881, 2006.