On Opportunistic Techniques for Solving Decentralized Markov Decision Processes with Temporal Constraints

Janusz Marecki and Milind Tambe
Computer Science Department
University of Southern California
941 W 37th Place, Los Angeles, CA 90089
{marecki, tambe}@usc.edu

ABSTRACT

Decentralized Markov Decision Processes (DEC-MDPs) are a popular model of agent-coordination problems in domains with uncertainty and time constraints but very difficult to solve. In this paper, we improve a state-of-the-art heuristic solution method for DEC-MDPs, called OC-DEC-MDP, that has recently been shown to scale up to larger DEC-MDPs. Our heuristic solution method, called Value Function Propagation (VFP), combines two orthogonal improvements of OC-DEC-MDP. First, it speeds up OC-DEC-MDP by an order of magnitude by maintaining and manipulating a value function for each state (as a function of time) rather than a separate value for each pair of state and time interval. Furthermore, it achieves better solution qualities than OC-DEC-MDP because, as our analytical results show, it does not overestimate the expected total reward like OC-DEC- MDP. We test both improvements independently in a crisis-management domain as well as for other types of domains. Our experimental results demonstrate a significant speedup of VFP over OC-DEC-MDP as well as higher solution qualities in a variety of situations.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence - Multi-agent Systems

General Terms

Algorithms, Theory

Keywords

Multi-agent systems, Decentralized Markov Decision Process, Temporal Constraints, Locally Optimal Solution

1. INTRODUCTION

The development of algorithms for effective coordination of multiple agents acting as a team in uncertain and time critical domains has recently become a very active research field with potential applications ranging from coordination of agents during a hostage rescue mission [12] to the coordination of Autonomous Mars Exploration Rovers [2]. Because of the uncertain and dynamic characteristics of such domains, decision-theoretic models have received a lot of attention in recent years, mainly thanks to their expressiveness and the ability to reason about the utility of actions over time.

Key decision-theoretic models that have become popular in the literature include Decentralized Markov Decision Processes (DEC-MDPs) and Decentralized, Partially Observable Markov Decision Processes (DEC-POMDPs). Unfortunately, solving these models optimally has been proven to be NEXP-complete [3], hence more tractable subclasses of these models have been the subject of intensive research. In particular, Network Distributed POMDP [14] which assume that not all the agents interact with each other, Transition Independent DEC-MDP [2] which assume that transition function is decomposable into local transition functions or DEC-MDP with Event Driven Interactions [1] which assume that interactions between agents happen at fixed time points constitute good examples of such subclasses. Although globally optimal algorithms for these subclasses have demonstrated promising results, domains on which these algorithms run are still small and time horizons are limited to only a few time ticks.

To remedy that, locally optimal algorithms have been proposed [13] [4] [5]. In particular, Opportunity Cost DEC-MDP [4] [5], referred to as OC-DEC-MDP, is particularly notable, as it has been shown to scale up to domains with hundreds of tasks and double digit time horizons. Additionally, OC-DEC-MDP is unique in its ability to address both temporal constraints and uncertain method execution durations, which is an important factor for real-world domains. OC-DEC-MDP is able to scale up to such domains mainly because instead of searching for the globally optimal solution, it carries out a series of policy iterations; in each iteration it performs a value iteration that reuses the data computed during the previous policy iteration. However, OC-DEC-MDP is still slow, especially as the time horizon and the number of methods approach large values. The reason for high runtimes of OC-DEC-MDP for such domains is a consequence of its huge state space, i.e., OC-DEC-MDP introduces a separate state for each possible pair of method and method execution interval. Furthermore, OC-DEC-MDP overestimates the reward that a method expects to receive for enabling the execution of future methods. This reward, also referred to as the *opportunity cost*, plays a crucial role in agent decision making, and as we show later, its overestimation leads to highly suboptimal policies.

In this context, we present VFP (= **V**alue **F**unction **P** ropagation), an efficient solution technique for the DEC-MDP model with temporal constraints and uncertain method execution durations, that builds on the success of OC-DEC-MDP. VFP introduces our two orthogonal ideas: First, similarly to [7] [9] and [10], we maintain

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

and manipulate a value function over time for each method rather than a separate value for each pair of method and time interval. Such representation allows us to group the time points for which the value function changes at the same rate (= its slope is constant), which results in fast, functional propagation of value functions. Second, we prove (both theoretically and empirically) that OC-DEC- MDP overestimates the opportunity cost, and to remedy that, we introduce a set of heuristics, that correct the opportunity cost overestimation problem.

This paper is organized as follows: In section 2 we motivate this research by introducing a civilian rescue domain where a team of fire- brigades must coordinate in order to rescue civilians trapped in a burning building. In section 3 we provide a detailed description of our DEC-MDP model with Temporal Constraints and in section 4 we discuss how one could solve the problems encoded in our model using globally optimal and locally optimal solvers. Sections 5 and 6 discuss the two orthogonal improvements to the state-of-the-art OC-DEC-MDP algorithm that our VFP algorithm implements. Finally, in section 7 we demonstrate empirically the impact of our two orthogonal improvements, i.e., we show that: (i) The new heuristics correct the opportunity cost overestimation problem leading to higher quality policies, and (ii) By allowing for a systematic tradeoff of solution quality for time, the VFP algorithm runs much faster than the OC-DEC-MDP algorithm

2. MOTIVATING EXAMPLE

We are interested in domains where multiple agents must coordinate their plans over time, despite uncertainty in plan execution duration and outcome. One example domain is large-scale disaster, like a fire in a skyscraper. Because there can be hundreds of civilians scattered across numerous floors, multiple rescue teams have to be dispatched, and radio communication channels can quickly get saturated and useless. In particular, small teams of fire-brigades must be sent on separate missions to rescue the civilians trapped in dozens of different locations.

Picture a small *mission plan* from Figure (1), where three fire-brigades have been assigned a task to rescue the civilians trapped at site B, accessed from site A (e.g. an office accessed from the floor)[1]. General fire fighting procedures involve both: (i) putting out the flames, and (ii) ventilating the site to let the toxic, high temperature gases escape, with the restriction that ventilation should not be performed too fast in order to prevent the fire from spreading. The team estimates that the civilians have 20 minutes before the fire at site B becomes unbearable, and that the fire at site A has to be put out in order to open the access to site B. As has happened in the past in large scale disasters, communication often breaks down; and hence we assume in this domain that there is no communication between the fire-brigades 1,2 and 3 (denoted as FB1, FB2 and FB3). Consequently, FB2 does not know if it is already safe to ventilate site A, FB1 does not know if it is already safe to enter site A and start fighting fire at site B, etc. We assign the reward 50 for evacuating the civilians from site B, and a smaller reward 20 for the successful ventilation of site A, since the civilians themselves might succeed in breaking out from site B.

One can clearly see the dilemma, that FB2 faces: It can only estimate the durations of the "Fight fire at site A" methods to be executed by FB1 and FB3, and at the same time FB2 knows that time is running out for civilians. If FB2 ventilates site A too early, the fire will spread out of control, whereas if FB2 waits with the ventilation method for too long, fire at site B will become unbearable for the civilians. In general, agents have to perform a sequence of such difficult decisions; in particular, decision process of FB2 involves first choosing when to start ventilating site A, and then (depending on the time it took to ventilate site A), choosing when to start evacuating the civilians from site B. Such sequence of decisions constitutes the policy of an agent, and it must be found fast because time is running out.

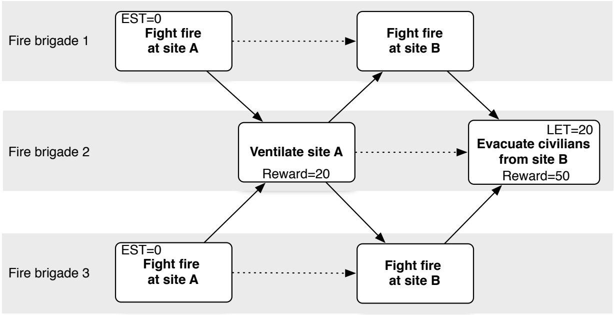


Figure 1: Civilian rescue domain and a mission plan. Dotted arrows represent implicit precedence constraints within an agent.

3. MODEL DESCRIPTION

We encode our decision problems in a model which we refer to as Decentralized MDP with Temporal Constraints [2]. Each instance of our decision problems can be described as a tuple $\langle M, A, C, P, R\rangle$ where $M = \{m_i\}_{i=1}^{|M|}$ is the set of methods, and $A = \{A_k\}_{k=1}^{|A|}$ is the set of agents. Agents cannot communicate during mission execution. Each agent A_k is assigned to a set M_k of methods, such that $\bigcup_{k=1}^{|A|} M_k = M$ and $\forall_{i,j;i\neq j} M_i \cap M_j = \emptyset$. Also, each method of agent A_k can be executed only once, and agent A_k can execute only one method at a time. Method execution times are uncertain and $P = \{p_i\}_{i=1}^{|M|}$ is the set of distributions of method execution durations. In particular, $p_i(t)$ is the probability that the execution of method m_i consumes time t. C is a set of temporal constraints in the system. Methods are partially ordered and each method has fixed time windows inside which it can be executed, i.e., $C = C_\prec \cup C_{[\,]}$ where $C_\prec$ is the set of predecessor constraints and $C_{[\,]}$ is the set of time window constraints. For $c \in C_\prec$, $c = \langle m_i, m_j\rangle$ means that method m_i precedes method m_j i.e., execution of m_j cannot start before m_i terminates. In particular, apart from multi-agent predecessor constraints, for an agent A_k, all its methods form a chain linked by predecessor constraints. We assume, that the graph $G = \langle M, C_\prec\rangle$ is acyclic, does not have disconnected nodes (the problem cannot be decomposed into independent subproblems), and its source and sink vertices identify the source and sink methods of the system. For $c \in C_{[\,]}$, $c = \langle m_i, EST, LET\rangle$ means that execution of m_i can only start after the Earliest Starting Time EST and must finish before the Latest End Time LET; we allow methods to have multiple disjoint time window constraints. Although distributions p_i can extend to infinite time horizons, given the time window constraints, the planning horizon $\Delta = \max_{\langle m,\tau,\tau'\rangle \in C_{[\,]}} \tau'$ is considered as the mission deadline. Finally, $R = \{r_i\}_{i=1}^{|M|}$ is the set of non-negative rewards, i.e., r_i is obtained upon successful execution of m_i.

Since there is no communication allowed, an agent can only estimate the probabilities that its methods have already been enabled

[1] We explain the EST and LET notation in section 3

[2] One could also use the OC-DEC-MDP framework, which models both time and resource constraints

by other agents. Consequently, if $m_j \in M_k$ is the next method to be executed by the agent A_k (we also say, that agent A_k is at method m_j) and the current time is $t \in [0, \Delta]$, the agent has to make a decision whether to **E**xecute the method m_j (denoted as E), or to **W**ait (denoted as W). In case agent A_k decides to wait, it remains idle for an arbitrary small time ϵ, and resumes operation at the same place (= about to execute method m_j) at time $t + \epsilon$. In case agent A_k decides to Execute the next method, two outcomes are possible:

Success: The agent A_k receives reward r_j so long as the following conditions hold: (i) All the methods $\{m_i | \langle m_i, m_j \rangle \in C_{\prec}\}$ that directly enable method m_j have already been completed, (ii) Execution of method m_j started in some time window of method m_j, i.e., $\exists_{\langle m_j, \tau, \tau' \rangle \in C_{[\,]}}$ such that $t \in [\tau, \tau']$, and (iii) Execution of method m_j finished inside the same time window, i.e., agent A_k completed method m_j in time less than or equal to $\tau' - t$. The agent executes method m_j until it detects that that the method is completed. The agent then moves to its next method (if such method exists).

Failure: Agent A_k can detect if any of the above-mentioned conditions does not hold, in which case it stops its execution. Although methods $m_k \in \{m | \langle m_j, m \rangle \in C_{\prec}\}$ will never become enabled, other agents may still continue their execution.

The policy π_k of an agent A_k is a function $\pi_k : M_k \times [0, \Delta] \rightarrow \{W, E\}$, and $\pi_k(\langle m, t \rangle) = a$ means, that if A_k is at method m at time t, it will choose to perform the action a. Because of multiple time windows of method execution, π_k cannot simply be stored as a set of pairs (M_k, t_k), where t_k is a point in time when the agent switches from action W to action E (Figure 2) .

A joint team policy $\pi = [\pi_k]_{k=1}^{|A|}$ is considered to be optimal (denoted as π^*), if it maximizes the sum of expected rewards for all the agents.

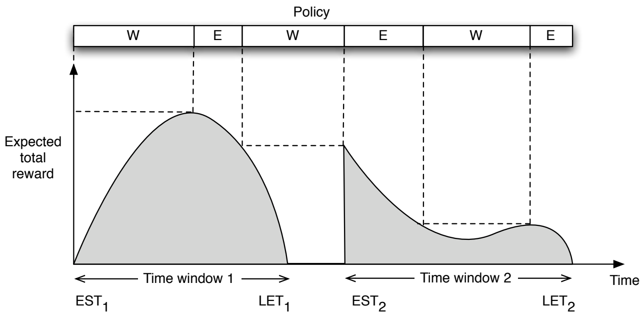


Figure 2: Agent policy for a method

4. SOLUTION TECHNIQUES

4.1 Optimal Algorithms

Optimal joint policy π^* is usually found by using the Bellman update principle, i.e., in order to determine the optimal policy for method m_j, optimal policies for methods $m_k \in \{m | \langle m_j, m \rangle \in C_{\prec}\}$ are used. Unfortunately, for our model, the optimal policy for method m_j also depends on policies for methods $m_i \in \{m | \langle m, m_j \rangle \in C_{\prec}\}$. This double dependency results from the fact, that the expected reward for starting the execution of method m_j at time t also depends on the probability that method m_j will be enabled by time t. Consequently, if time is discretized and method execution order is not given, the problem complexity is in the NEXP-complete class [3]. This complexity could be reduced if the Coverage Set Algorithm (CSA) [1] was used. However, CSA complexity is double exponential in number of method enabling time points, which ranges from 0 to Δ for our domains.

4.2 Locally Optimal Algorithms

Following the limited applicability of globally optimal algorithms for DEC-MDPs with Temporal Constraints, locally optimal algorithms appear more promising. Specially, the OC-DEC-MDP algorithm [4] is particularly significant, as it has shown to easily scale up to domains with hundreds of methods. The idea of the OC-DEC-MDP algorithm is to start with the *earliest starting time* policy π^0 (according to which an agent will start executing the method m as soon as m has a non-zero chance of being already enabled), and then improve it iteratively, until no further improvement is possible. At each iteration, the algorithm starts with some policy π, which uniquely determines the probabilities $P_{i,[\tau,\tau']}$ $EST \leq \tau < \tau' \leq LET$ that method m_i will be performed in the time interval $[\tau, \tau']$. It then performs two steps:

Step 1: It propagates from sink methods to source methods the values $V_{i,[\tau,\tau']}$, that represent the expected utility for executing method m_i in the time interval $[\tau, \tau']$. This propagation uses the probabilities $P_{i,[\tau,\tau']}$ from previous algorithm iteration. We call this step a *value propagation phase*.

Step 2: Given the values $V_{i,[\tau,\tau']}$ from Step 1, the algorithm chooses the most profitable method execution intervals which are stored in a new policy π'. It then propagates the new probabilities $P_{i,[\tau,\tau']}$ from source methods to sink methods. We call this step a *probability propagation phase*. If policy π' does not improve π, the algorithm terminates.

There are two shortcomings of the OC-DEC-MDP algorithm that we address in this paper. First, each of OC-DEC-MDP states is a pair $\langle m_j, [\tau, \tau'] \rangle$, where $[\tau, \tau']$ is a time interval in which method m_j can be executed. While such state representation is beneficial, in that the problem can be solved with a standard value iteration algorithm, it blurs the intuitive mapping from time t to the expected total reward for starting the execution of m_j at time t. Consequently, if some method m_i enables method m_j, and the values $V_{j,[\tau,\tau']} \forall_{\tau,\tau' \in [0,\Delta]}$ are known, the operation that calculates the values $V_{i,[\tau,\tau']} \forall \tau, \tau' \in [0, \Delta]$ (during the value propagation phase), runs in time $O(I^2)$, where I is the number of time intervals [3]. Since the runtime of the whole algorithm is proportional to the runtime of this operation, especially for big time horizons Δ, the OC- DEC-MDP algorithm runs slow.

Second, while OC-DEC-MDP emphasizes on precise calculation of values $V_{j,[\tau,\tau']}$, it fails to address a critical issue that determines how the values $V_{j,[\tau,\tau']}$ are split given that the method m_j has multiple enabling methods. As we show later, OC-DEC-MDP splits $V_{j,[\tau,\tau']}$ into parts that may overestimate $V_{j,[\tau,\tau']}$ when summed up again. As a result, methods that precede the method m_j overestimate the value for enabling m_j which, as we show later, can have disastrous consequences. In the next two sections, we address both of these shortcomings.

5. VALUE FUNCTION PROPAGATION (VFP)

The general scheme of the VFP algorithm is identical to the OC-DEC-MDP algorithm, in that it performs a series of policy improvement iterations, each one involving a Value and Probability Propagation Phase. However, instead of propagating separate values, VFP maintains and propagates the whole functions, we therefore refer to these phases as the *value function* propagation phase and the *probability function* propagation phase. To this end, for

[3] Similarly for the probability propagation phase

each method $m_i \in M$, we define three new functions (related to each other in Equation 3):

- **Value Function**, denoted as $v_i(t)$, that maps time $t \in [0, \Delta]$ to the expected total reward for starting the execution of method m_i at time t.
- **Opportunity Cost Function**, denoted as $V_i(t)$, that maps time $t \in [0, \Delta]$ to the expected total reward for starting the execution of method m_i at time t assuming that m_i is enabled.
- **Probability Function**, denoted as $P_i(t)$, that maps time $t \in [0, \Delta]$ to the probability that method m_i will be completed before time t.

Such functional representation allows us to easily read the current policy, i.e., if an agent A_k is at method m_i at time t, then it will wait as long as value function $v_i(t)$ will be greater in the future (see Figure 2). Formally:

$$\pi_k(\langle m_i, t\rangle) = \begin{cases} W & \text{if } \exists_{t'>t} \text{ such that } v_i(t) < v_i(t') \\ E & \text{otherwise.} \end{cases}$$

We now develop an analytical technique for performing the value function and probability function propagation phases.

5.1 Value Function Propagation Phase

Suppose, that we are performing a value function propagation phase during which the value functions are propagated from the sink methods to the source methods. At any time during this phase we encounter a situation shown in Figure 3, where opportunity cost functions $[V_{j_n}]_{n=0}^N$ of methods $[m_{j_n}]_{n=0}^N$ are known, and the opportunity cost V_{i_0} of method m_{i_0} is to be derived. Suppose the current time is t and agent A is at method m_{i_0}. If $\langle m_{i_0}, \tau, \tau'\rangle \in C_{[\,]}$ and $t \in [\tau, \tau']$ then the agent can start executing the method m_{i_0}. With probability $p_{i_0}(t')$ the agent will need t' time units to complete the method and as long as $t' < \tau' - t$, the method m_{i_0} will become enabled, and the agent will obtain the reward r_{i_0}. At this point, the agent can also expect to obtain $V_{j_n,i_0}(t+t')$ $n = 1, ..., N$ from the methods that m_{i_0} enables. Formally, the function V_{i_0} is derived by: [4]

$$V_{i_0}(t) \;=\; \begin{cases} \int_0^{\tau'-t} p_{i_0}(t')(r_{i_0} + \sum_{n=0}^N V_{j_n,i_0}(t+t'))dt' \\ \qquad \text{if } \exists_{\langle m_{i_0}\tau,\tau'\rangle \in C_{[\,]}} \text{ such that } t \in [\tau, \tau'] \\ 0 \qquad \text{otherwise} \end{cases} \tag{1}$$

Assume for now, that V_{j_n,i_0} represents a *full* opportunity cost that the agent receives, postponing the discussion on different techniques for splitting the opportunity cost V_{j_0} into $[V_{j_0,i_k}]_{k=0}^K$ until section 6. We now show how to derive V_{j_0,i_0} (derivation of V_{j_n,i_0} for $n \neq 0$ follows the same scheme).

Let $\overline{V}_{j_0,i_0}(t)$ be the opportunity cost of starting the execution of method m_{j_0} at time t assuming that method m_{i_0} has been completed. It can be derived by multiplying V_{j_0} by the probability functions of all methods other than m_{i_0} that enable m_{j_0}. Formally:

$$\overline{V}_{j_0,i_0}(t) = V_{j_0}(t) \cdot \prod_{k=1}^K P_{i_k}(t).$$

Note, that this derivation is only approximate because in general the probability functions $[P_{i_k}]_{k=1}^K$ do not have to represent independent random variables (similar approximation was used in [4], [5]). Observe now, that $\overline{V}_{j_0,i_0}$ does *not* have to be monotonically decreasing, i.e., delaying the execution of the method m_{i_0} can sometimes be profitable. Consequently the opportunity cost $V_{j_0,i_0}(t)$ of enabling method m_{i_0} at time t must be greater than or equal to $\overline{V}_{j_0,i_0}$. Furthermore, V_{j_0,i_0} should be non-increasing. Formally:

$$V_{j_0,i_0} = \min_{f \in F} f \tag{2}$$

Where $F = \{f \mid f \geq \overline{V}_{j_0,i_0} \text{ and } f(t) \geq f(t')\ \forall_{t<t'}\}$.

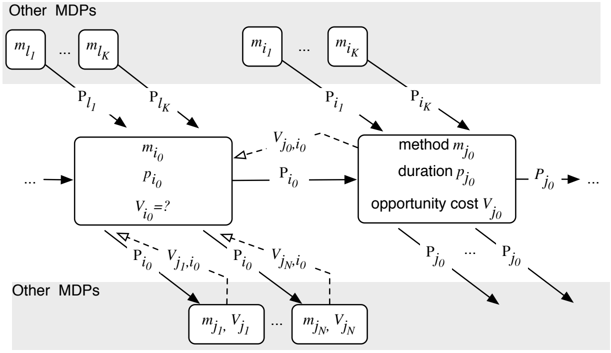


Figure 3: Fragment of an MDP of agent A_k. Probability functions propagate forward (left to right) whereas value functions propagate backward (right to left).

Knowing the opportunity cost V_{i_0}, we can then easily derive the value function v_{i_0}. If agent A is about to start the execution of m_{i_0} it means, that A must have completed its part of the mission plan up to the method m_{i_0}. Since A does not know if other agents have completed methods $[m_{l_k}]_{k=1}^{k=K}$, in order to derive v_{i_0}, it has to multiply V_{i_0} by the probability functions of all methods of other agents that enable m_{i_0}. Formally:

$$v_{i_0}(t) \;=\; V_{i_0}(t) \cdot \prod_{k=1}^K P_{l_k}(t) \tag{3}$$

Where the dependency of $[P_{l_k}]_{k=1}^K$ is also ignored.

We have consequently shown a general scheme how to propagate the value functions: Knowing $[v_{j_n}]_{n=0}^N$ and $[V_{j_n}]_{n=0}^N$ of methods $[m_{j_n}]_{n=0}^N$ we can derive v_{i_0} and V_{i_0} of method m_{i_0}. In general, the value function propagation scheme starts with sink nodes. It then visits at each time a method m, such that all the methods that m enables have already been marked as visited. The value function propagation phase terminates when all the source methods have been marked as visited.

5.2 Reading the Policy

In order to determine the policy of agent A_k for the method m_{j_0} we must identify the set Z_{j_0} of intervals of inactivity $[z, z'] \subset [0, ..., \Delta]$, such that:

$$\forall_{t \in [z,z']}\ \pi_k(\langle m_{j_0}, t\rangle) = W.$$

One can easily identify the intervals of Z_{j_0} by looking at the time intervals in which the value function v_{j_0} does not decrease monotonically (Figure 2).

[4] Note, that for $t \in [\tau, \tau']$, if $h(t) := r_{i_0} + \sum_{n=0}^N V_{j_n,i_0}(\tau' - t)$ then V_{i_0} is a convolution of p and h: $V_{i_0}(t) = (p_{i_0} * h)(\tau' - t)$.

5.3 Probability Function Propagation Phase

Assume now, that value functions and opportunity cost values have all been propagated from sink methods to source nodes and the sets Z_j for all methods $m_j \in M$ have been identified. Since value function propagation phase was using probabilities $P_i(t)$ for methods $m_i \in M$ and times $t \in [0, \Delta]$ found at previous algorithm iteration, we now have to find new values $P_i(t)$, in order to prepare the algorithm for its next iteration. We now show how in the general case (Figure 3) propagate the probability functions forward through one method, i.e., we assume that the probability functions $[P_{i_k}]_{k=0}^{K}$ of methods $[m_{i_k}]_{k=0}^{K}$ are known, and the probability function P_{j_0} of method m_{j_0} must be derived. Let p_{j_0} be the probability distribution function of method m_{j_0} execution duration, and Z_{j_0} be the set of intervals of inactivity for method m_{j_0}, found during the last value function propagation phase. If we ignore the dependency of $[P_{i_k}]_{k=0}^{K}$ then the probability $\overline{P}_{j_0}(t)$ that the execution of method m_{j_0} *starts* before time t is given by:

$$\overline{P}_{j_0}(t) = \begin{cases} \prod_{k=0}^{K} P_{i_k}(\tau) & \text{if } \exists(\tau, \tau') \in Z_{j_0} \text{ s.t. } t \in (\tau, \tau') \\ \prod_{k=0}^{K} P_{i_k}(t) & \text{otherwise.} \end{cases}$$

Because inside the interval of inactivity (τ, τ'), the probability that the execution of method m_{j_0} has started does not increase.
Given $\overline{P}_{j_0}(t)$, the probability $P_{j_0}(t)$ that method m_{j_0} will be *completed* by time t is derived by:

$$P_{j_0}(t) = \int_0^t \int_0^{t'} \frac{\partial \overline{P}_{j_0}}{\partial t}(t'') \cdot p_{j_0}(t' - t'') dt'' \;\; dt' \qquad (4)$$

Also written compactly as $\frac{\partial P_{j_0}}{\partial t} = p_{j_0} * \frac{\partial \overline{P}_{j_0}}{\partial t}$.
We have consequently shown how to propagate the probability functions $[P_{i_k}]_{k=0}^{K}$ of methods $[m_{i_k}]_{k=0}^{K}$ to obtain the probability function P_{j_0} of method m_{j_0}. The general, the probability function propagation phase starts with source methods m_{s_i} for which we know that $P_{s_i} = 1$ since they are enabled by default. We then visit at each time a method m such that all the methods that enable m have already been marked as visited. The probability function propagation phase terminates when all the sink methods have been marked as visited.

5.4 The Algorithm

Similarly to the OC-DEC-MDP algorithm, VFP starts the policy improvement iterations with the earliest starting time policy π^0. Then at each iteration it: (i) Propagates the value functions $[v_i]_{i=1}^{|M|}$ using the old probability functions $[P_i]_{i=1}^{|M|}$ from previous algorithm iteration and establishes the new sets $[Z_i]_{i=1}^{|M|}$ of method inactivity intervals, and (ii) propagates the new probability functions $[P'_i]_{i=1}^{|M|}$ using the newly established sets $[Z_i]_{i=1}^{|M|}$. These new functions $[P'_i]_{i=1}^{|M|}$ are then used in the next iteration of the algorithm. Similarly to OC-DEC-MDP, VFP terminates if a new policy does not improve the policy from the previous algorithm iteration within some parameter ϵ.

5.5 Implementation of Function Operations

So far, we have derived the functional operations for value function and probability function propagation without choosing any function representation. In general, our functional operations can handle continuous time, and one has freedom to choose a desired function approximation technique, such as piecewise linear [7], piecewise constant [9], or piecewise gamma [11] approximation. However, since one of our goals is to compare VFP with the existing OC-DEC- MDP algorithm, that works only for discrete time, we also discretize time, and choose to approximate value functions and probability functions with piecewise linear (PWL) functions.
When the VFP algorithm propagates the value functions and probability functions, it constantly carries out operations represented by equations (1) and (4) and we have already shown that these operations are equivalent to computing the convolution $f(t) = (p*h)(t)$. If time is discretized, functions $p(t)$ and $h(t)$ are discrete; however, $h(t)$ can be nicely approximated with a PWL function $\widehat{h}(t)$, which is exactly what VFP does. As a result, instead of performing $O(\Delta^2)$ multiplications to compute $f(t)$, VFP only needs to perform $O(k \cdot \Delta)$ multiplications to compute $f(t)$, where k is the number of linear segments of $\widehat{h}(t)$ (note, that since $h(t)$ is monotonic, $\widehat{h}(t)$ is usually close to $h(t)$ with $k \ll \Delta$). Since P_i values are in range $[0, 1]$ and V_i values are in range $[0, \sum_{m_i \in M} r_i]$, we suggest to approximate $V_i(t)$ with $\widehat{V}_i(t)$ within error ϵ_V, and $P_i(t)$ with $\widehat{P}_i(t)$ within error ϵ_P. We now prove that the overall approximation error accumulated during the value function propagation phase can be expressed in terms of ϵ_P and ϵ_V:

THEOREM 1. *Let $C_\prec$ be a set of precedence constraints of a DEC-MDP with Temporal Constraints, and ϵ_P and ϵ_V be the probability function and value function approximation errors respectively. The overall error $\epsilon_\pi = \max_V sup_{t \in [0,\Delta]} |V(t) - \widehat{V}(t)|$ of value function propagation phase is then bounded by:*
$|C_\prec| \left(\epsilon_V + ((1 + \epsilon_P)^{|C_\prec|} - 1) \sum_{m_i \in M} r_i \right)$.

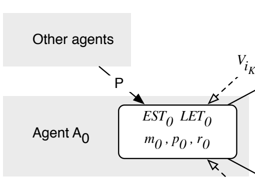
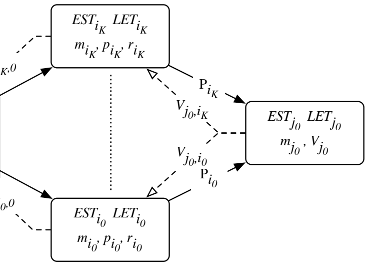
PROOF. *In order to establish the bound for ϵ_π, we first prove by induction on the size of $C_\prec$, that the overall error of probability function propagation phase, $\epsilon_{\pi(P)} = \max_P sup_{t \in [0,\Delta]} |P(t) - \widehat{P}(t)|$ is bounded by $(1 + \epsilon_P)^{|C_\prec|} - 1$.*
Induction base: *If $n = 1$ only two methods are present, and we will perform the operation identified by Equation (4) only once, introducing the error $\epsilon_{\pi(P)} = \epsilon_P = (1 + \epsilon_P)^{|C_\prec|} - 1$.*
Induction step: *Suppose, that $\epsilon_{\pi(P)}$ for $|C_\prec| = n$ is bounded by $(1 + \epsilon_P)^n - 1$, and we want to prove that this statement holds for $|C_\prec| = n$. Let $G = \langle M, C_\prec \rangle$ be a graph with at most $n + 1$ edges, and $\overline{G} = \langle M, \overline{C}_\prec \rangle$ be a subgraph of G, such that $\overline{C}_\prec = C_\prec - \{\langle m_i, m_j \rangle\}$, where $m_j \in M$ is a sink node in G. From the induction assumption we have, that $\overline{C}_\prec$ introduces the probability propagation phase error bounded by $(1 + \epsilon_P)^n - 1$. We now add back the link $\{\langle m_i, m_j \rangle\}$ to $\overline{C}_\prec$, which affects the error of only one probability function, namely P_j, by a factor of $(1 + \epsilon_P)$. Since probability propagation phase error in $\overline{C}_\prec$ was bounded by $(1 + \epsilon_P)^n - 1$, in $C_\prec = \overline{C}_\prec \cup \{\langle m_i, m_j \rangle\}$ it can be at most $((1 + \epsilon_P)^n - 1)(1 + \epsilon_P) < (1 + \epsilon_P)^{n+1} - 1$. Thus, if opportunity cost functions are not overestimated, they are bounded by $\sum_{m_i \in M} r_i$ and the error of a single value function propagation operation will be at most*

$$\int_0^\Delta p(t)(\epsilon_V + ((1+\epsilon_P)^{|C_\prec|} - 1) \sum_{m_i \in M} r_i) \, dt < \epsilon_V + ((1+\epsilon_P)^{|C_\prec|} - 1) \sum_{m_i \in M} r_i.$$

Since the number of value function propagation operations is $|C_\prec|$, the total error ϵ_π of the value function propagation phase is bounded by: $|C_\prec| \left(\epsilon_V + ((1 + \epsilon_P)^{|C_\prec|} - 1) \sum_{m_i \in M} r_i \right)$. □

6. SPLITTING THE OPPORTUNITY COST FUNCTIONS

In section 5 we left out the discussion about how the opportunity cost function V_{j_0} of method m_{j_0} is split into opportunity cost functions $[V_{j_0,i_k}]_{k=0}^{K}$ sent back to methods $[m_{i_k}]_{k=0}^{K}$, that directly enable method m_{j_0}. So far, we have taken the same approach as in [4] and [5] in that the opportunity cost function V_{j_0,i_k} that the method m_{i_k} sends back to the method m_{j_0} is a minimal, non-increasing function that dominates function $\overline{V}_{j_0,i_k}(t) = (V_{j_0} \cdot \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{i_{k'}})(t)$. We refer to this approach, as heuristic $H_{\langle 1,1 \rangle}$. Before we prove that this heuristic overestimates the opportunity cost, we discuss three problems that might occur when splitting the opportunity cost functions: (i) overestimation, (ii) underestimation and (iii) starvation. Consider the situation in Figure (4) when value function propagation for methods $[m_{i_k}]_{k=0}^{K}$ is performed. For each $k = 0, ..., K$, Equation (1) derives the opportunity cost function V_{i_k} from immediate reward r_k and opportunity cost function V_{j_0,i_k}. If m_0 is the only methods that precedes method m_k, then $\overline{V}_{i_k,0} = V_{i_k}$ is propagated to method m_0, and consequently the opportunity cost for completing the method m_0 at time t is equal to $\sum_{k=0}^{K} V_{i_k,0}(t)$. If this cost is overestimated, then an agent A_0 at method m_0 will have too much incentive to finish the execution of m_0 at time t (e.g. instead of postponing the execution to the start time of the next time window). Consequently, although the probability $P(t)$ that m_0 will be enabled by other agents by time t is low, agent A_0 might still find the expected utility of starting the execution of m_0 at time t higher than the expected utility of doing it later. As a result, it will choose at time t to start executing method m_0 instead of waiting, which can have disastrous consequences. Similarly, if $\sum_{k=0}^{K} V_{i_k,0}(t)$ is underestimated, agent A_0 might loose interest in enabling the future methods $[m_{i_k}]_{k=0}^{K}$ and just focus on maximizing the chance of obtaining its immediate reward r_0. Since this chance is increased when agent A_0 waits[5], it will consider at time t to be more profitable to wait, instead of starting the execution of m_0, which can have similarly disastrous consequences. Finally, if V_{j_0} is split in a way, that for some k, $V_{j_0,i_k} = 0$, it is the method m_{i_k} that underestimates the opportunity cost of enabling method m_{j_0}, and the similar reasoning applies. We call such problem a starvation of method m_k. That short discussion shows the importance of splitting the opportunity cost function V_{j_0} in such a way, that overestimation, underestimation, and starvation problem is avoided. We now prove that:

THEOREM 2. *Heuristic $H_{\langle 1,1 \rangle}$ can overestimate the opportunity cost.*

PROOF. *We prove the theorem by showing a case where the overestimation occurs. For the mission plan from Figure (4), let $H_{\langle 1,1 \rangle}$ split V_{j_0} into $[\overline{V}_{j_0,i_k} = V_{j_0} \cdot \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{i_{k'}}]_{k=0}^{K}$ sent to methods $[m_{i_k}]_{k=0}^{K}$ respectively. Also, assume that methods $[m_{i_k}]_{k=0}^{K}$ provide no local reward and have the same time windows, i.e., $r_{i_k} = 0; EST_{i_k} = 0, LET_{i_k} = \Delta$ for $k = 0, ..., K$. To prove the overestimation of opportunity cost, we must identify $t_0 \in [0, ..., \Delta]$ such that the opportunity cost $\sum_{k=0}^{K} V_{i_k}(t)$ for methods $[m_{i_k}]_{k=0}^{K}$ at time $t \in [0, .., \Delta]$ is greater than the opportunity cost $V_{j_0}(t)$. From Equation (1) we have:*

$$V_{i_k}(t) = \int_0^{\Delta - t} p_{i_k}(t') V_{j_0,i_k}(t + t') dt'$$

Summing over all methods $[m_{i_k}]_{k=0}^{K}$ we obtain:

$$\begin{aligned} \sum_{k=0}^{K} V_{i_k}(t) &= \sum_{k=0}^{K} \int_0^{\Delta - t} p_{i_k}(t') V_{j_0,i_k}(t + t') dt' \qquad (5) \\ &\geq \sum_{k=0}^{K} \int_0^{\Delta - t} p_{i_k}(t') \overline{V}_{j_0,i_k}(t + t') dt' \\ &= \sum_{k=0}^{K} \int_0^{\Delta - t} p_{i_k}(t') V_{j_0}(t + t') \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{i_{k'}}(t + t') dt' \end{aligned}$$

Let $c \in (0, 1]$ be a constant and $t_0 \in [0, \Delta]$ be such that $\forall_{t > t_0}$ and $\forall_{k=0,..,K}$ we have $\prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{i_{k'}}(t) > c$. Then:

$$\sum_{k=0}^{K} V_{i_k}(t_0) > \sum_{k=0}^{K} \int_0^{\Delta - t_0} p_{i_k}(t') V_{j_0}(t_0 + t') \cdot c \ dt'$$

Because $P_{j_{k'}}$ is non-decreasing. Now, suppose there exists $t_1 \in (t_0, \Delta]$, such that $\sum_{k=0}^{K} \int_0^{t_1 - t_0} p_{i_k}(t') dt' > \frac{V_{j_0}(t_0)}{c \cdot V_{j_0}(t_1)}$. Since decreasing the upper limit of the integral over positive function also decreases the integral, we have:

$$\sum_{k=0}^{K} V_{i_k}(t_0) > c \sum_{k=0}^{K} \int_{t_0}^{t_1} p_{i_k}(t' - t_0) V_{j_0}(t') dt'$$

And since $V_{j_0}(t')$ is non-increasing we have:

$$\begin{aligned} \sum_{k=0}^{K} V_{i_k}(t_0) &> c \cdot V_{j_0}(t_1) \sum_{k=0}^{K} \int_{t_0}^{t_1} p_{i_k}(t' - t_0) dt' \qquad (6) \\ &= c \cdot V_{j_0}(t_1) \sum_{k=0}^{K} \int_0^{t_1 - t_0} p_{i_k}(t') dt' \\ &> c \cdot V_{j_0}(t_1) \frac{V_j(t_0)}{c \cdot V_j(t_1)} = V_j(t_0) \end{aligned}$$

Consequently, the opportunity cost $\sum_{k=0}^{K} V_{i_k}(t_0)$ of starting the execution of methods $[m_{i_k}]_{k=0}^{K}$ at time $t \in [0, .., \Delta]$ is greater than the opportunity cost $V_{j_0}(t_0)$ which proves the theorem. We demonstrate in the experimental section that the overestimation of the opportunity cost is easily observable in practice. □

To remedy the problem of opportunity cost overestimation, we propose three alternative heuristics that split the opportunity cost functions:

- **Heuristic** $H_{\langle 1,0 \rangle}$: Only one method, m_{i_k} gets the full expected reward for enabling method m_{j_0}, i.e., $\overline{V}_{j_0,i_{k'}}(t) = 0$ for $k' \in \{0, ..., K\} \backslash \{k\}$ and $\overline{V}_{j_0,i_k}(t) = (V_{j_0} \cdot \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{i_{k'}})(t)$.
- **Heuristic** $H_{\langle 1/2,1/2 \rangle}$: Each method $[m_{i_k}]_{k=0}^{K}$ gets the full opportunity cost for enabling method m_{j_0} divided by the number K of methods enabling the method m_{j_0}, i.e., $\overline{V}_{j_0,i_k}(t) = \frac{1}{K}(V_{j_0} \cdot \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{i_{k'}})(t)$ for $k \in \{0, ..., K\}$.
- **Heuristic** $\widehat{H}_{\langle 1,1 \rangle}$: This is a normalized version of the $H_{\langle 1,1 \rangle}$ heuristic in that each method $[m_{i_k}]_{k=0}^{K}$ initially gets the full

[5] Assuming $LET_0 \gg t$

opportunity cost for enabling the method m_{j_0}. To avoid opportunity cost overestimation, we normalize the split functions when their sum exceeds the opportunity cost function to be split. Formally:

$$\overline{V}_{j_0,i_k}(t) = \begin{cases} V^{H_{\langle 1,1\rangle}}_{j_0,i_k}(t) & \text{if } \sum_{k=0}^{K} V^{H_{\langle 1,1\rangle}}_{j_0,i_k}(t) < V_{j_0}(t) \\ V_{j_0}(t)\frac{V^{H_{\langle 1,1\rangle}}_{j_0,i_k}(t)}{\sum_{k=0}^{K} V^{H_{\langle 1,1\rangle}}_{j_0,i_k}(t)} & \text{otherwise} \end{cases}$$

Where $V^{H_{\langle 1,1\rangle}}_{j_0,i_k}(t) = (V_{j_0} \cdot \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{j_{k'}})(t)$.

For the new heuristics, we now prove, that:

THEOREM 3. *Heuristics $H_{\langle 1,0\rangle}$, $H_{\langle 1/2,1/2\rangle}$ and $\widehat{H}_{\langle 1,1\rangle}$ do not overestimate the opportunity cost.*

PROOF. *When heuristic $H_{\langle 1,0\rangle}$ is used to split the opportunity cost function V_{j_0}, only one method (e.g. m_{i_k}) gets the opportunity cost for enabling method m_{j_0}. Thus:*

$$\sum_{k'=0}^{K} V_{i'_k}(t) = \int_0^{\Delta - t} p_{i_k}(t') V_{j_0,i_k}(t+t')dt' \qquad (7)$$

And since V_{j_0} is non-increasing

$$\leq \int_0^{\Delta - t} p_{i_k}(t') V_{j_0}(t+t') \cdot \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{j_{k'}}(t+t')dt'$$

$$\leq \int_0^{\Delta - t} p_{i_k}(t') V_{j_0}(t+t')dt' \leq V_{j_0}(t)$$

The last inequality is also a consequence of the fact that V_{j_0} is non-increasing.
For heuristic $H_{\langle 1/2,1/2\rangle}$ we similarly have:

$$\sum_{k=0}^{K} V_{i_k}(t) \leq \sum_{k=0}^{K} \int_0^{\Delta - t} p_{i_k}(t') \frac{1}{K} V_{j_0}(t+t') \prod_{\substack{k' \in \{0,\dots,K\} \\ k' \neq k}} P_{j_{k'}}(t+t')dt'$$

$$\leq \frac{1}{K} \sum_{k=0}^{K} \int_0^{\Delta - t} p_{i_k}(t') V_{j_0}(t+t')dt'$$

$$\leq \frac{1}{K} \cdot K \cdot V_{j_0}(t) = V_{j_0}(t).$$

For heuristic $\widehat{H}_{\langle 1,1\rangle}$, the opportunity cost function V_{j_0} is by definition split in such manner, that $\sum_{k=0}^{K} V_{i_k}(t) \leq V_{j_0}(t)$. Consequently, we have proved, that our new heuristics $H_{\langle 1,0\rangle}$, $H_{\langle 1/2,1/2\rangle}$ and $\widehat{H}_{\langle 1,1\rangle}$ avoid the overestimation of the opportunity cost. □

The reason why we have introduced all three new heuristics is the following: Since $H_{\langle 1,1\rangle}$ overestimates the opportunity cost, one has to choose which method m_{i_k} will receive the reward from enabling the method m_{j_0}, which is exactly what the heuristic $H_{\langle 1,0\rangle}$ does. However, heuristic $H_{\langle 1,0\rangle}$ leaves $K - 1$ methods that precede the method m_{j_0} without any reward which leads to starvation. Starvation can be avoided if opportunity cost functions are split using heuristic $H_{\langle 1/2,1/2\rangle}$, that provides reward to all enabling methods. However, the sum of split opportunity cost functions for the $H_{\langle 1/2,1/2\rangle}$ heuristic can be smaller than the non-zero opportunity cost function for the $H_{\langle 1,0\rangle}$ heuristic, which is clearly undesirable. This is why we have proposed the $\widehat{H}_{\langle 1,1\rangle}$ heuristic, which by definition avoids the overestimation, underestimation and starvation problems.

7. EXPERIMENTAL EVALUATION

Since the VFP algorithm that we introduced provides two orthogonal improvements over the OC-DEC-MDP algorithm, the experimental evaluation we performed consisted of two parts: In part 1, we tested empirically the quality of solutions that an locally optimal solver (either OC-DEC-MDP or VFP) finds, given it uses different opportunity cost function splitting heuristic, and in part 2, we compared the runtimes of the VFP and OC-DEC- MDP algorithms for a variety of mission plan configurations.

Part 1: We first ran the VFP algorithm on a generic mission plan configuration from Figure 4 where only methods m_{j_0}, m_{i_1}, m_{i_2} and m_0 were present. Time windows of all methods were set to 400, duration p_{j_0} of method m_{j_0} was uniform, i.e., $p_{j_0}(t) = \frac{1}{400}$ and durations p_{i_1}, p_{i_2} of methods m_{i_1}, m_{i_2} were normal distributions, i.e., $p_{i_1} = N(\mu = 250, \sigma = 20)$, and $p_{i_2} = N(\mu = 200, \sigma = 100)$. We assumed that only method m_{j_0} provided reward, i.e. $r_{j_0} = 10$ was the reward for finishing the execution of method m_{j_0} before time $t = 400$. We show our results in Figure (5) where the x-axis of each of the graphs represents time whereas the y-axis represents the opportunity cost. The first graph confirms, that when the opportunity cost function V_{j_0} was split into opportunity cost functions V_{i_1} and V_{i_2} using the $H_{\langle 1,1\rangle}$ heuristic, the function $V_{i_1} + V_{i_2}$ was not always below the V_{j_0} function. In particular, $V_{i_1}(280) + V_{i_2}(280)$ exceeded $V_{j_0}(280)$ by 69%. When heuristics $H_{\langle 1,0\rangle}$, $H_{\langle 1/2,1/2\rangle}$ and $\widehat{H}_{\langle 1,1\rangle}$ were used (graphs 2,3 and 4), the function $V_{i_1} + V_{i_2}$ was always below V_{j_0}.

We then shifted our attention to the civilian rescue domain introduced in Figure 1 for which we sampled all action execution durations from the normal distribution $N = (\mu = 5, \sigma = 2))$. To obtain the baseline for the heuristic performance, we implemented a globally optimal solver, that found a true expected total reward for this domain (Figure (7a)). We then compared this reward with a expected total reward found by a locally optimal solver guided by each of the discussed heuristics. Figure (7a), which plots on the y-axis the expected total reward of a policy complements our previous results: $H_{\langle 1,1\rangle}$ heuristic overestimated the expected total reward by 280% whereas the other heuristics were able to guide the locally optimal solver close to a true expected total reward.

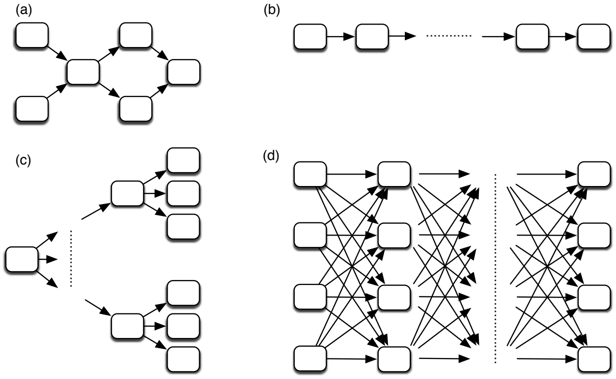


Figure 6: Mission plan configurations: (a) civilian rescue domain, (b) chain of n methods, (c) tree of n methods with branching factor $= 3$ and (d) square mesh of n methods.

Part 2: We then chose $H_{\langle 1,1\rangle}$ to split the opportunity cost functions and conducted a series of experiments aimed at testing the scalability of VFP for various mission plan configurations, using the performance of the OC-DEC-MDP algorithm as a benchmark. We began the VFP scalability tests with a configuration from Figure (6a) associated with the civilian rescue domain, for which method execution durations were extended to normal distributions $N(\mu = 30, \sigma = 5)$, and the deadline was extended to $\Delta = 200$.

We decided to test the runtime of the VFP algorithm running with

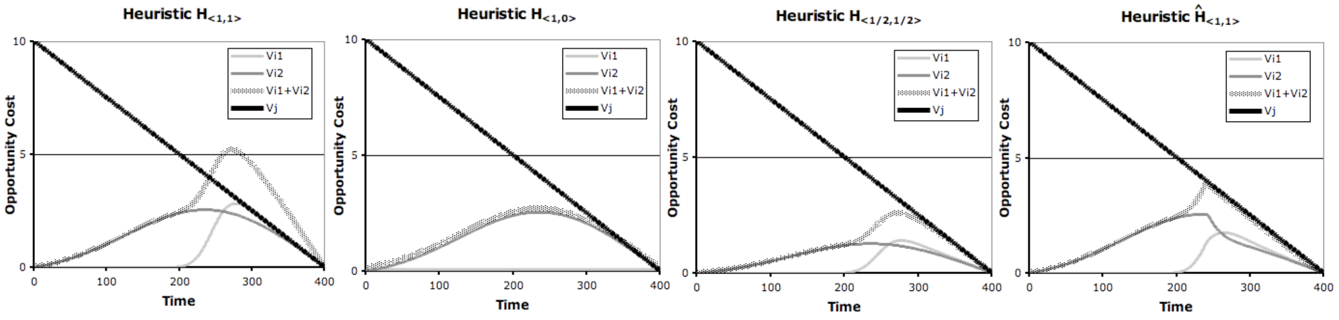


Figure 5: Visualization of heuristics for opportunity costs splitting.

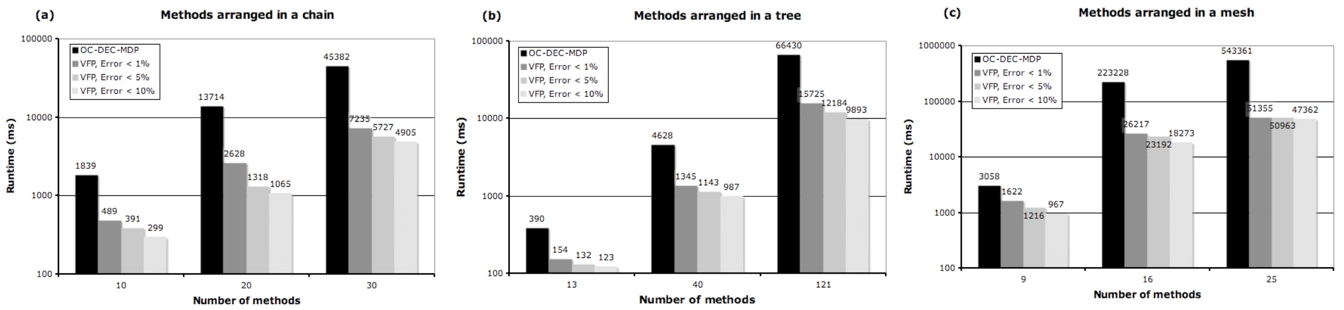


Figure 8: Scalability experiments for OC-DEC-MDP and VFP for different network configurations.

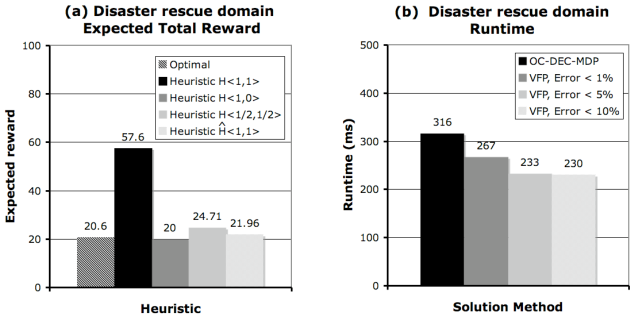


Figure 7: VFP performance in the civilian rescue domain.

three different levels of accuracy, i.e., different approximation parameters ϵ_P and ϵ_V were chosen, such that the cumulative error of the solution found by VFP stayed within 1%, 5% and 10% of the solution found by the OC- DEC-MDP algorithm. We then run both algorithms for a total of 100 policy improvement iterations. Figure (7b) shows the performance of the VFP algorithm in the civilian rescue domain (y-axis shows the runtime in milliseconds). As we see, for this small domain, VFP runs 15% faster than OC-DEC-MDP when computing the policy with an error of less than 1%. For comparison, the globally optimal solved did not terminate within the first three hours of its runtime which shows the strength of the opportunistic solvers, like OC-DEC-MDP.

We next decided to test how VFP performs in a more difficult domain, i.e., with methods forming a long chain (Figure (6b)). We tested chains of 10, 20 and 30 methods, increasing at the same time method time windows to 350, 700 and 1050 to ensure that later methods can be reached. We show the results in Figure (8a), where we vary on the x-axis the number of methods and plot on the y-axis the algorithm runtime (notice the *logarithmic* scale). As we observe, scaling up the domain reveals the high performance of VFP: Within 1% error, it runs up to 6 times faster than OC-DEC-MDP.

We then tested how VFP scales up, given that the methods are arranged into a tree (Figure (6c)). In particular, we considered trees with branching factor of 3, and depth of 2, 3 and 4, increasing at the same time the time horizon from 200 to 300, and then to 400. We show the results in Figure (8b). Although the speedups are smaller than in case of a chain, the VFP algorithm still runs up to 4 times faster than OC-DEC-MDP when computing the policy with an error of less than 1%.

We finally tested how VFP handles the domains with methods arranged into a $n \times n$ mesh, i.e., $C_{\prec} = \{\langle m_{i,j}, m_{k,j+1}\rangle\}$ for $i = 1, ..., n;\ k = 1, ..., n; j = 1, ..., n-1$. In particular, we consider meshes of 3×3, 4×4, and 5×5 methods. For such configurations we have to greatly increase the time horizon since the probabilities of enabling the final methods by a particular time decrease exponentially. We therefore vary the time horizons from 3000 to 4000, and then to 5000. We show the results in Figure (8c) where, especially for larger meshes, the VFP algorithm runs up to one order of magnitude faster than OC-DEC-MDP while finding a policy that is within less than 1% from the policy found by OC- DEC-MDP.

8. CONCLUSIONS

Decentralized Markov Decision Process (DEC-MDP) has been very popular for modeling of agent-coordination problems, it is very difficult to solve, especially for the real-world domains. In this paper, we improved a state-of-the-art heuristic solution method for DEC-MDPs, called OC-DEC-MDP, that has recently been shown to scale up to large DEC-MDPs. Our heuristic solution method, called Value Function Propagation (VFP), provided two orthogonal improvements of OC-DEC-MDP: (i) It speeded up OC-DEC-MDP by an order of magnitude by maintaining and manipulating a value function for each method rather than a separate value for each pair of method and time interval, and (ii) it achieved better solution qualities than OC-DEC-MDP because it corrected the overestimation of the opportunity cost of OC-DEC-MDP.

In terms of related work, we have extensively discussed the OC-DEC-MDP algorithm [4]. Furthermore, as discussed in Section 4, there are globally optimal algorithms for solving DEC-MDPs with temporal constraints [1] [12]. Unfortunately, they fail to scale up to large-scale domains at present time. Beyond OC-DEC-MDP, there are other locally optimal algorithms for DEC-MDPs and DEC-POMDPs [8] [13], [14], yet, they have traditionally not dealt with uncertain execution times and temporal constraints. Finally, value function techniques have been studied in context of single agent MDPs [7] [9]. However, similarly to [6], they fail to address the lack of global state knowledge, which is a fundamental issue in decentralized planning.

Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA875005C0030. The authors also want to thank Sven Koenig and anonymous reviewers for their valuable comments.

9. REFERENCES

[1] R. Becker, V. Lesser, and S. Zilberstein. Decentralized MDPs with Event-Driven Interactions. In *AAMAS*, pages 302–309, 2004.

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *AAMAS*, pages 41–48, 2003.

[3] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *UAI*, pages 32–37, 2000.

[4] A. Beynier and A. Mouaddib. A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *AAMAS*, pages 963–969, 2005.

[5] A. Beynier and A. Mouaddib. An iterative algorithm for solving constrained decentralized Markov decision processes. In *AAAI*, pages 1089–1094, 2006.

[6] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, pages 478–485, 1999.

[7] J. Boyan and M. Littman. Exact solutions to time-dependent MDPs. In *NIPS*, pages 1026–1032, 2000.

[8] C. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems, 2003.

[9] L. Li and M. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI*, pages 1175–1180, 2005.

[10] Y. Liu and S. Koenig. Risk-sensitive planning with one-switch utility functions: Value iteration. In *AAAI*, pages 993–999, 2005.

[11] J. Marecki, S. Koenig, and M.Tambe. A fast analytical algorithm for solving Markov decision processes with real-valued resources. In *IJCAI*, 2007. To Appear.

[12] D. Musliner, E. Durfee, J. Wu, D. Dolgov, R. Goldman, and M. Boddy. Coordinated plan management using multiagent MDPs. In *AAAI Spring Symposium*, 2006.

[13] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, pages 705–711, 2003.

[14] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synergy of distributed constraint optimization and POMDPs. In *IJCAI*, pages 1758–1760, 2005.