Planning with Continuous Resources for Agent Teams

Janusz Marecki
IBM T.J. Watson Research Center
1101 Kirchawan Road, Route 134
Yorktown Heights, NY 10598
marecki@us.ibm.com

Milind Tambe
University of Southern California
3737 Watt Way, Powell Hall of Engineering 410
Los Angeles, CA 90089-0781
tambe@usc.edu

ABSTRACT

Many problems of multiagent planning under uncertainty require distributed reasoning with continuous resources and resource limits. Decentralized Markov Decision Problems (Dec-MDPs) are well-suited to address such problems, but unfortunately, prior Dec-MDP approaches either discretize resources at the expense of speed and quality guarantees, or avoid discretization only by limiting agents' action choices or interactions (e.g. assumption of transition independence). To address these shortcomings, this paper proposes M-DPFP, a novel algorithm for planning with continuous resources for agent teams, with three key features: (i) it maintains the agent team interaction graph to identify and prune the suboptimal policies and to allow the agents to be transition dependent, (ii) it operates in a continuous space of probability functions to provide the error bound on the solution quality and finally (iii) it focuses the search for policies on the most relevant parts of this search space to allow for a systematic trade-off of solution quality for speed. Our experiments show that M-DPFP finds high quality solutions and exhibits superior performance when compared with a discretization-based approach. We also show that M-DPFP is applicable to solving problems that are beyond the scope of existing approaches.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Theory

Keywords

Multi-agent systems, Decentralized Markov Decision Process, Continuous Resources

1. INTRODUCTION

Effective coordination of agents acting as a team in an uncertain environment has recently become a very active area of research with potential applications ranging from coordination of unmanned planetary exploration rovers [2] to coordination of agents during a hostage rescue mission [13]. Unfortunately, finding optimal policies for agents acting in such domains is a difficult problem, especially when agents' actions are contingent on the availability of continuous resources such as time or energy. A predominant approach for solving such planning problems is to discretize resource levels that results in a finite number number of candidate policies to consider. It is then possible to find the optimal policies by casting the planning problem as decentralized MDPs or POMDPs [1], [2], [4], [14], [15], [16]. However, discretization of continuous resource levels leads to policies that are defined for only a handful of all possible resource levels and hence, invalidates formal solution quality guarantees.

To preserve the formal solution quality guarantees the candidate algorithms must *not* discretize resources and rather operate directly on the continuous state-spaces spanning continuous resources. However, extending existing single-agent discretization-free algorithms [7], [8], [9], [11], [12] to multi-agent settings is a non-trivial task, mainly due to the lack of global state knowledge in multi-agent systems. To date, only few algorithms have overcome this problem, yet, at a cost of introducing new restrictive assumptions: Generalized Semi MDP [18] assumes unlimited resources, Value Function Propagation [10]—which only finds locally optimal policies—assumes that agents already know the ordering of their actions and must only decide when to start them and finally, Transition Independent Decentralized Hybrid MDP [3] assumes that agents are transition independent (i.e., the outcomes of agent actions do not affect the action choice of other agents).

To remedy these shortcomings, this paper introduces **M**ultiagent **D**ynamic **P**robability **F**unction **P**ropagation (M-DPFP), a new algorithm for planning with continuous resources for transition dependent agent teams. Precisely, M-DPFP solves the planning problems modeled as Continuous Resource Decentralized MDPs (CR-DEC-MDPs [10]), a framework that extends an existing framework [5] for planning with discrete resources for agent teams by allowing resources to take real-values. In solving CR-DEC-MDPs near-optimally M-DPFP relies on a novel combination of three key ideas: (i) it first constructs the agent team interaction graph to identify and prune the suboptimal policies early on, based on the analysis of suboptimal action sequences, (ii) it formulates the planning problem in the dual space of cumulative distribution functions [11], to avoid the discretization of resource levels and finally (iii) it groups similar regions of the dual search space (regions that correspond to similar agent observations) and focuses the search for the optimal solution to the dual problem on the states of that dual search space that are most likely to be encountered, to allow for a systematic trade-off of solution quality for speed. We begin by introducing our planning problems of interest and recalling the CR-DEC-MDP framework used to represent them formally.

Cite as: Planning with Continuous Resources for Agent Teams, Janusz Marecki and Milind Tambe, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

2. BACKGROUND

2.1 Problem Statement

Of general interest to the multiagent planning community are decision theoretic problems that assume agents with methods to be executed, as commonly proposed in the literature [1], [5], [10] and inspired by the DARPA Coordinators effort [13], which in turn is based on the popular GPGP paradigm [6]. Formally, such planning problems assume a team of N agents deployed on a mission to perform methods from the set $M = \{m_1, ..., m_K\}$. Each agent n is assigned to a set M_n of methods, such that $\{M_n\}_{n=1}^{N}$ is a partitioning of M. Furthermore, each agent can be executing only one method at a time and each method can be executed only once. We assume a single, continuous resource (e.g., time): The initial resource level (e.g. starting time) of agent n is $l_{n,0}$ and the probability that the execution of method m_k will require x amount of resource is $p_k(x)$ where p_k is a continuous probability density function. The outcome of the execution of methods is contingent on the satisfiability of resource precedence and resource limit constraints defined as follows:

Resource precedence constraints grouped in set $C_{\prec}$ impose a partial ordering of method execution. Precisely, a constraint $\langle i, j\rangle \in C_{\prec}$ between methods m_i and m_j (of possibly different agents) imposes two necessary (but not sufficient) conditions for method m_j to be executed successfully: (i) method m_i must be executed successfully and (ii) if the execution of method m_i finished with l amount of resource left, the execution of method m_j must start with *less* than l resource left (e.g., if time is a resource, the execution of m_j cannot start before the execution of m_i is finished). We often say that method m_j is *enabled* at resource level l if all methods m_i such that $\langle i, j\rangle \in C_{\prec}$ have been successfully finished with more than l resources left (e.g. if time is a resource, method m_j is enabled at time t if the execution of all methods m_i such that $\langle i, j\rangle \in C_{\prec}$ has been successfully finished before time t). Agents learn the status of resource precedence constraints postfactum: Given constraint $\langle i, j\rangle \in C_{\prec}$, the agent that owns method m_j will start executing it without knowing whether m_i has been completed (by some other agent). Only after it finishes the execution of m_j (successfully or unsuccessfully) it will learn if m_i has been executed successfully or unsuccessfully.[1]

Resource limit constraints grouped in set $C_{[]}$ restrict agent resource levels during method execution. Precisely, a resource limit constraint $\langle i, l', l''\rangle \in C_{[]}$ for a method m_i imposes that resource levels of an agent executing method m_i must stay within range $[l', l'']$ (e.g., if time is a resource, m_i can only be executed in time interval $[l', l'']$). We distinguish $\Delta = \max\{l'' : \langle i, l', l''\rangle \in C_{[]}\}$ to be the maximum resource level for a given planning problem (e.g. if time is a resource, Δ is the mission deadline).

A reward r_i for the execution of method m_i is earned when: (i) resource limit constraints are not violated during the execution of method m_i and (ii) method m_i is enabled when its execution starts. As long as condition (i) holds, the execution of method m_i proceeds normally, regardless of whether condition (ii) holds or not. Yet, as soon as condition (i) stops to hold (agent resource level drops too low) the execution of method m_i is interrupted, m_i is considered to be executed unsuccessfully and reward r_i is not earned. The problem is to find a team policy that maximizes the expectation over the sum of rewards earned by the agents.

An example of such a planning problem is shown in Fig.1. Here, resource = time, agent team consists of agents 1 and 2 assigned to methods m_1, m_2, m_3 and m_4, m_5, m_6 respectively and resource precedence constraints (dashed arrows) read: "method m_4 will fail if method m_1 has not been completed before m_4 starts" and "m_2 will fail if m_5 has not been completed before m_2 starts". (Resource limit constraints are not shown.) Agents need to coordinate their efforts to maximize the expectation over the sum of their local rewards. For example, if $r_1 \ll r_3 \ll r_4$, agent 1 might still want to consider first executing method m_1 (rather than m_3) to enable agent 2 to execute its method m_4 before time Δ, to earn reward r_4.

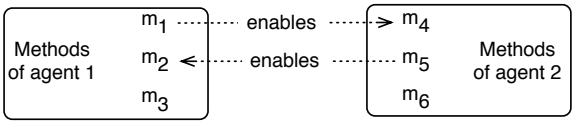

Figure 1: Example domain

2.2 CR-DEC-MDP Framework

The CR-DEC-MDP framework [10] for modeling the planning problems discussed above is defined as follows:

DEFINITION 1. **Execution event** *$e = \langle i, l_1, l_2, q\rangle$ represents the execution of method $m_i \in M_n$ with outcome $q \in \{0, 1\}$ (0 = unsuccessful; 1 = successful), during which the resource level of agent n dropped from l_1 to l_2.*

DEFINITION 2. *CR-DEC-MDP for a team of N agents is a set $\{\mathcal{MDP}_n\}_{n\in N}$ where $\mathcal{MDP}_n = \langle \mathcal{S}_n, \mathcal{A}_n, \mathcal{P}_n, \mathcal{R}_n\rangle$ is a Markov decision processes of agent n defined as follows:*

- $\mathcal{S}_n$ is the set of states of agent n and $s = (e_{n,0}, e_1, ..., e_k) \in \mathcal{S}_n$ is a sequence of execution events. Execution event $e_{n,0} = \langle 0, l_{n,0}, l_{n,0}, 0\rangle$ is used solely to encode the initial resource level $l_{n,0}$ of agent n whereas execution events $e_1, ..., e_k$ correspond to methods that the agent has actually executed. The agent's starting state is then simply $s_{n,0} = (e_{n,0})$.
- $\mathcal{A}_n$ is the set of actions of agent n and $A(s) \subset \mathcal{A}_n$ is the set of methods that the agent can start executing in state s. Because methods can be executed only once, $A(s) = M_n \setminus \{m_i : \langle i, l_1, l_2, q\rangle \in s\}$. Additionally, the agent can choose in state s to remain idle and allow its resource level to drop by the controllable amount δ.
- $\mathcal{P}_n$ is the state-to-state transition function of agent n that depends not only on the current state of agent n, but also on the current state of other agents. When agent n is in state $s = (..., \langle i, l_1, l_2, q\rangle)$ it can either choose to delay the execution of its next action until its resource level drops by a controllable amount δ, which results in a transition to state $s' = (..., \langle i, l_1, l_2, q\rangle, \langle 0, l_2, l_2-\delta, 0\rangle)$ or to execute a method $m_j \in A(s)$. In the latter case the agent will transition to state $s' = (..., \langle i, l_1, l_2, q\rangle, \langle j, l_2, l_2 - x, q_j\rangle)$ with probability $p_j(x)$ where x is the amount of resource consumed during the execution of method m_j. The result q_j of the execution of method m_j is contingent upon the satisfiability of resource limit constraints and resource precedence constraints. Precisely, $q_j = 1$ if and only if (i) the resource levels of the agent remain within the admissible range during the execution of method m_j, i.e., there exists a resource limit constraint $\langle j, l', l''\rangle \in C_{[]}$ such that $l' \geq l_2 \geq l_2 - x \geq l''$ and (ii) method m_j is enabled when it is started, i.e., the execution of all methods $m_k \in M$ such that $\langle k, j\rangle \in C_{\prec}$ has finished successfully with at least l_2 resources left. (In other words, condition (ii) states that for all methods $m_k \in M_{n'}$

[1] In this paper we focus on time as a resource, as temporal reasoning is clearly one of the most important paradigms in multi-agent planning. However, our techniques also apply to other resources such as energy, temperature, space etc.

such that $\langle k, j\rangle \in C_{\prec}$, the current state of agent n' contains event $e = \langle k, l', l'', 1\rangle$ such that $l'' \geq l_2$.)
Finally, $\mathcal{P}_n$ must be modified accordingly to reflect that the execution of method m_j is automatically interrupted when the agent resource level drops too low, i.e., below value l'' for some admissible resource range constraint $\langle j, l', l''\rangle \in C_{[]}$. Thus, the probability of interruption, i.e., the probability that the agent will utilize $x \geq l_2 - l''$ amount of resource for the execution of method m_j is $1 - \int_0^{l_2 - l''} p_j(y)dy$.

- $\mathcal{R}_n$ is the reward function of agent n. Upon transitioning to state $s' = (..., \langle j, l_2, l_2 - x, q_j\rangle)$ the agent receives reward $q_j \cdot r_j$.

Given local policies π_n of agents $n = 1, .., N$ the **joint policy** is a vector $\pi = (\pi_1, ..., \pi_n)$. The **value of policy** π is defined as $V(\pi) := \sum_{n=1}^{N} V^{\pi_n}(s_{n,0})$ where $V^{\pi_n}(s_{n,0})$ is the total expected reward of policy π_n of agent n followed from state $s_{n,0}$. The optimal policy $\pi^* = (\pi_1^*, ..., \pi_N^*)$ then maximizes $V(\pi)$.
For explanation purposes, in the following we use time as a resource. It then follows that time values range from 0 to Δ, execution of methods causes time to *increase*, resource limit constraints are method execution time windows and resource precedence constraints specify that the execution of one method can only start after another methods has been successfully executed.

2.3 DPFP Algorithm

The M-DPFP algorithm builds upon the highly efficient DPFP algorithm [11] for solving single agent continuous resource MDPs. DPFP's main idea is to perform a forward search for policies in a search space referred to as the dual space of cumulative distribution functions. The use of that search space allows DPFP to estimate the likelihood of reaching different regions of the state space and then utilize these likelihoods to focus the policy generation effort on more relevant regions of the state space, while still providing guarantees on the solution quality. Precisely, DPFP assumes the following: Φ is the set of all possible sequences ϕ of actions that the agent can execute from its starting state; A_ϕ is the set of actions that the agent can execute after completing the actions from ϕ; $F_\phi^\pi(t)$ is the probability that actions from ϕ will be completed *before* time t and $F_\phi^\pi(a)(t)$ is the probability that actions from ϕ will be completed and the next action $a \in A_\phi$ will be started *before* time t, when the agent follows policy π from its starting state. (F_ϕ^π, $F_\phi^\pi(s)$ are cumulative distribution functions over $t \in [0, \Delta]$.) $F^\pi = \{F_\phi^\pi, F_\phi^\pi(a) : \phi \in \Phi; a \in A_\phi\}$ is then the *solution to the dual problem* that corresponds to policy π and F^* is the optimal solution (that corresponds to the optimal policy π^*). The correspondence is reflexive: A deterministic policy $\pi_\phi(t)$ for the action sequence ϕ completed at time t can be extracted from F^π by comparing the rate of increase of functions $\{F_\phi^\pi(a)\}_{a \in A_\phi}$, i.e., $\pi_\phi(t) = \arg\max_{a \in A_\phi}\{\lim_{\epsilon \to 0^+} F_\phi^\pi(a)(t+\epsilon) - F_\phi^\pi(a)(t)\}$.
DPFP does not exactly find the optimal policy π^*; instead, it finds an approximate policy $\widehat{\pi}^*$ that is arbitrarily close to π^*. Precisely, it first projects with granularity κ the search space X onto its finite counterpart $\widehat{X}$: The projection consists in converting each solution $F \in X$ to a solution $\widehat{F} \in \widehat{X}$. The conversion itself consists in approximating each cumulative distribution function $F_\phi \in F$ with a function $\widehat{F}_\phi \in \widehat{F}$ given by: $\widehat{F}_\phi(t) = \lfloor F_\phi(t)/\kappa \rfloor * \kappa$. (Notice that $\widehat{F}_\phi \in \widehat{F}$ is a stair function with stairs of fixed height κ.) The immediate effect of the projection is that, assuming that the starting state of the agent is known, $\widehat{X}$ contains only a finite number of elements. In other words, DPFP can then iterate over different elements $\widehat{F} \in \widehat{X}$ (i.e., iterate over all corresponding policies $\widehat{\pi}$) to find the best approximate policy $\widehat{\pi}^*$. Precisely, for each solution $\widehat{F}$ found during that iteration DPFP calculates the value of $\widehat{F}$ (the value of a policy that corresponds to $\widehat{F}$) with the formula $V(\widehat{F}) = \sum_{\phi \in \Phi} \widehat{F}_\phi(\Delta) \cdot r_\phi$ where r_ϕ is the reward for executing the last action from sequence ϕ. In particular, DPFP identifies the optimal solution $\widehat{F}^* \in \widehat{X}$ that maximizes $V(\widehat{F})$ and then uses $\widehat{F}^*$ to extract the corresponding policy $\widehat{\pi}^*$. Although $\widehat{\pi}^* \neq \pi^*$, the error $|V(\widehat{F}^*) - V(F^*)|$ of $\widehat{\pi}^*$ is provably proportional to κ.
The key contributor to the superior efficiency of DPFP is that it exploits the loss of probability mass of cumulative distribution functions. In essence, when the probability of reaching certain regions of the state-space is below a given threshold, the expected quality loss for executing suboptimal actions in these regions can be bounded, and DPFP can tradeoff this quality loss for efficiency.

3. M-DPFP APPROACH

We now show that DPFP's idea to use the dual space of cumulative distribution functions for planning with continuous resources in single agent systems is applicable to solving the planning problems modeled as CR-DEC-MDPs. In essence, planning with continuous resources in single or multiagent systems is similar in that it requires the planner to reason about agents whose actions are conditioned upon resource availability. However, problems modeled as CR-DEC-MDPs are fundamentally more complex, as they involve multiple agents, each with its local state knowledge, required to act as a team to perform a set of tasks. In particular, agents must act despite resource precedence constraints and without being informed at every instance about the status of execution of other team members; their only information about other agents inferred from the success or failure of execution of dependent tasks.
To address the challenges that these requirements pose we develop M-DPFP, a new algorithm for solving CR-DEC-MDPs. M-DPFP borrows from DPFP in that it too operates on the dual space of cumulative distribution functions. Yet, M-DPFP fundamentally differs from DPFP in that M-DPFP's search for policies in the dual space respects the interactions between the agents. We explain the key novelties of M-DPFP in three steps: First, we introduce the *agent team interaction graph* to capture the interactions between the agents and to filter them accordingly to prune out the suboptimal policies. Next, we revisit the dual space of cumulative distribution functions: we extend it to multiple agents and formulate the *multiagent dual problem*. Finally, we show how to use the agent team interaction graph to build the joint policies and find the optimal solution to the multiagent dual problem.

3.1 Agent Team Interaction Graph

At a basic level, the agent team interaction graph: (i) provides a compact representation of the interactions between the agents, (ii) filters out the agent interactions associated with suboptimal joint policies and finally, (iii) arranges subsequent agent decisions in a priority list, to allow the centralized planner to construct the joint policies one-agent-decision-at-a-time, to find the solution to the multiagent dual problem. The agent team interaction graph is defined as follows:

DEFINITION 3. **Action sequence** *is a vector $\phi = (m_1, ..., m_k)$ of methods executed by an agent. Also, Φ_n is a set of all possible action sequences ϕ of agent n and $\Phi = \bigcup_n \Phi_n$ is a set of all possible action sequences of all agents.*[2]

[2]For convenience, $(\phi|m)$ denotes the action sequence that is a concatenation of action sequence ϕ with method m and $(..., m)$ denotes the action sequence that terminates with method m.

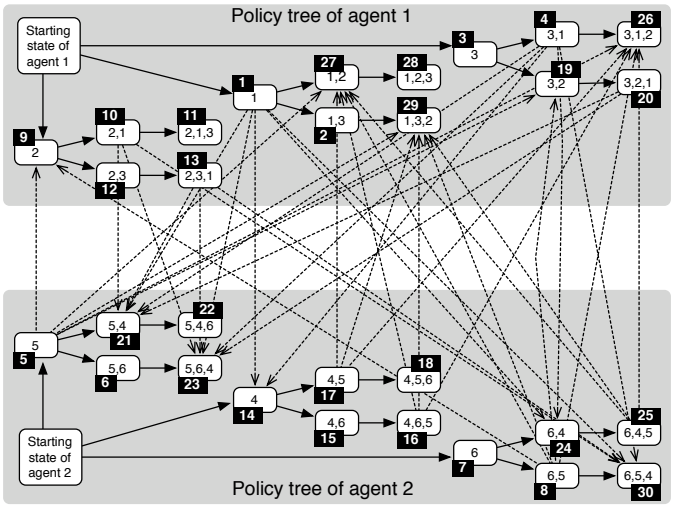

Figure 2: Agent Team Interaction Graph: Solid arrows represents agent actions, white boxes represent action sequences, dashed arrows represent agent interactions and black labels denote positions on the action sequences list $\mathcal{L}$.

DEFINITION 4. **Agent team interaction graph** *$G = (\Phi, C)$ consists of a set Φ of nodes and a set $C = C_l \cup C_{nl}$ of local and non-local arcs. Local arcs $C_l = \{(\phi, (\phi|m)) : \phi \in \Phi_n \wedge m \in M_n \wedge m \notin \phi\}$ link the action sequences of the same agent and represent the progression of its actions whereas the non-local arcs $C_{nl} = \{((\phi_i|m_i), (\phi_j|m_j)) : \phi_i \in \Phi_i \wedge m_i \in M_i \wedge \phi_j \in \Phi_j \wedge m_j \in M_j \wedge \langle i, j\rangle \in C_\prec\}$ link the action sequences of different agents and represent the agent interactions.*

The first function of the agent team interaction graph is to represent the agent interactions compactly, i.e., with time and method execution outcomes abstracted out. Consider the graph in Fig.2 for the domain in Fig.1. Here, agent actions (solid arrows) link the agent action sequences (white boxes). For example, agent 1 can perform action "execute method m_1" to transition from (m_2) to (m_2, m_1). Agent interactions (dashed arrows) are the non-local effects of agent actions resulting from the satisfiability of the resource precedence constraints. For example, resource precedence constraint $\langle 1, 4\rangle$ = "m_4 will fail if m_1 has not been completed before m_4 starts" translates into multiple non-local effects: Each non-local effect points from the action sequence $\phi = (..., m_1)$ to the action sequence $\phi' = (..., m_4)$ because the success or failure of the execution of method m_1 when agent 1 executes methods from sequence ϕ affects the outcome of execution of method m_4 when agent 2 executes methods from sequence ϕ'.

The second function of the agent team interaction graph is to speed up the search for policies (explained later) by filtering out the agent interactions (dashed arrows) that can only take place if the agents are executing suboptimal policies. In essence, the benefit of filtering is that, with some dashed arrows removed, parts of the agent team interaction graph that belong to different agents will hopefully become disconnected and hence, corresponding parts of their local policies could be optimized independently. For example, consider a policy according to which agent 1 starts by executing method m_2 and agent 2 starts by executing method m_4. This policy is clearly suboptimal because, even if agents wait with the execution of their methods, either m_2 or m_4 must fail. In the latter case, agent 1 waits with the execution of m_2 whereas agent 2 immediately executes method m_4 (which fails) and then m_5. An alternative policy that dominates the previous policy is when agent 2 immediately executes method m_5, to enable method m_2 earlier. Going farther with the identification of suboptimal policies, if agent 1 starts by executing method m_2 and agent 2 executes methods m_6, m_4, m_5 (in that order) the resulting behavior is also suboptimal as either m_2 or m_4 must fail (for the same reasons as just discussed). In general, for methods $m_{1A}, m_{1B} \in M_1$; $m_{2A}, m_{2B} \in M_2$ and precedence constraints $\langle m_{1A}, m_{2A}\rangle$; $\langle m_{2B}, m_{1B}\rangle \in C_\prec$, the filtering of agent interactions produced by suboptimal policies involves removing from graph G all dashed arrows $((\phi|m_{2B})(\phi'|m_{1B}))$ where $m_{2A} \in \phi$ and $m_{1A} \notin \phi'$ and all dashed arrows $((\phi''|m_{1A})(\phi'''|m_{2A}))$ where $m_{1B} \in \phi''$ and $m_{2B} \notin \phi'''$. (The filtering has already been done in Fig.2, e.g., a dashed arrow from (m_4, m_5) to (m_2) has been removed.) The benefit of filtering is that some parts of agents policies become independent and can be optimized independently. For example, agent 1's policies for sequences $(m_2, ...)$ are no longer dependent on agent 2's policies for sequences $(m_4, ...)$.

The final function of the agent team interaction graph is to arrange the agent actions in a priority list, to allow the centralized planner to build a joint policy in a *forward* fashion (and to iterate over these policies to find the optimal policy as explained in Section 3.3). Naturally, the planner starts constructing a joint policy by choosing the methods (e.g. m_2 and m_6) that agents execute from their starting states. The planner must then determine the times at which the execution of m_2 and m_6 should start. m_6 should be started as soon as possible because m_6 is always enabled. However, to determine when the execution of m_2 should start, the planner must first calculate the probability of m_2 being enabled by m_5 over time. (If the execution of m_2 starts too early, m_2 might still not be enabled by m_5; if it starts too late, m_2 might not be completed before time Δ.) Notice, that these probabilities are not yet known, as the planner has not yet constructed a policy involving the execution of m_5. Thus, prior to determining when the execution of m_2 should start, the planner must fix the policy for action sequence (m_6), i.e., choose when and what method (either m_4 or m_5) to start executing once the execution of m_6 finishes. Thus, to ensure that the planner can build a joint policy in a forward fashion, it must construct this policy from smaller building blocks (referred to as *policies for action sequences*), added in a specific order $\mathcal{L}$ defined as follows:

DEFINITION 5. **Action sequences list** *$\mathcal{L}$ is a topological sorting of nodes of the agent team interaction graph G according to which the action sequence ϕ appears on list $\mathcal{L}$ before the action sequence ϕ' if there exists a path in G from node ϕ to node ϕ'.*

Note, that $\mathcal{L}$ is not unique as there can be many topological sorting orderings of nodes of graph G with one such ordering shown in Fig.2).

3.2 Multiagent Dual Problem

A fundamental difference between single-agent and multi-agent systems is that in single-agent systems an agent knows exactly the state of the world whereas in multi-agent systems each agent can only estimate the state of the world from its local observation history. As such, DPFP's centralized planner did not have to consider observation histories in determining agent's optimal actions. In contrast, in CR-DEC-MDPs, the next action to be executed by an agent depends on this agent's *observation histories*, i.e., the times and the outcomes of execution of agent action. For example, if agent 2 in Fig.1 starts the execution of m_4 at time t and finishes this execution successfully, it learns that agent 1 has executed m_1 successfully before time t — this observation history allows agent 2 to make a more informed decision as to what method to execute next, as it has

a better understanding of the execution progress of agent 1. It is because of the need to keep track of agent's observations that the dual problem formulation, proposed in [11] for single agent planning problems, must be fundamentally revised.

A key insight in M-DPFP's formulation of the *multiagent* dual problem is that certain observation histories can be combined for the planning purposes and that the error that this abstraction entails is controllable. Precisely, M-DPFP's approach in combining observation histories consists in removing from them the times, and only preserving the outcomes of method execution. For example, two observation histories: "m_1 started at time t_1 executed successfully at time t_2 followed by m_2 started at time t_3 executed unsuccessfully at time $\mathbf{t}$" and "m_1 started at time t_4 executed successfully at time t_5 followed by m_2 started at time t_6 executed unsuccessfully at time $\mathbf{t}$" are indistinguishable for M-DPFP and both read "m_1 executed successfully followed by m_2 executed unsuccessfully at time $\mathbf{t}$". Keeping in mind this abstraction scheme, we now develop M-DPFP's formulation of the multiagent dual problem. (The error that this abstraction entails is bounded in Section 3.3.)

For an action sequence $\phi = (m_1...,m_k) \in \Phi_n$ of agent n let $Q = (q_1,...,q_k)$ be a corresponding sequence of outcomes of method execution and $A_{\phi,Q} \subset \mathcal{A}_n$ be the set of methods that agent n can execute upon finishing the execution of methods from ϕ with outcomes Q. (Pair ϕ, Q is referred to as an *action-outcome sequence*.) Also, let $F^{\pi_n}_{\phi,Q}(t)$ be the probability that agent n has completed the execution of methods from sequence ϕ with outcomes Q *before* time t when following a policy π_n. Upon completing at time t the execution of the last method of ϕ the agent chooses a method $m \in A_{\phi,Q}$ to be executed next, waits $\delta^{\pi_n}_{\phi,Q}(m)(t)$ amount of time and then starts the execution of m. Thus, let $F^{\pi_n}_{\phi,Q}(m)(t)$ be the probability that the agent has completed the execution of methods from sequence ϕ with outcomes Q *before* time t and chose m to be executed next (in the future) and $\widetilde{F}^{\pi_n}_{\phi,Q}(m)(t)$ be the probability that the agent has completed the execution of methods from sequence ϕ with outcomes Q and *actually started* the execution of method m before time t. Notice, that $F^{\pi_n}_{\phi,Q}$, $F^{\pi_n}_{\phi,Q}(m)$, $\widetilde{F}^{\pi_n}_{\phi,Q}(m)$ are cumulative distribution functions over $t \in [0,\Delta]$. In addition, for times t when the agent does not wait with the execution of method m (when $\delta^{\pi_n}_{\phi,Q}(m)(t) = 0$) it holds that $\widetilde{F}^{\pi_n}_{\phi,Q}(m)(t) = F^{\pi_n}_{\phi,Q}(m)(t)$; otherwise, the agents is waiting and $\widetilde{F}^{\pi_n}_{\phi,Q}(m)(t)$ remains constant as shown in Fig.3. In this context, a solution to the multiagent dual problem is a set of functions $F^{\pi} := \{F^{\pi_n}_{\phi,Q}; F^{\pi_n}_{\phi,Q}(m); \delta^{\pi_n}_{\phi,Q}(m)$ for all $\phi \in \cup_n \Phi_n; Q \in \{0,1\}^{|\phi|}; m \in A_{\phi,Q}\}$. The optimal solution to the multiagent dual problem is denoted as F^*.

We now formalize the multiagent dual problem. First, observe that rewards r_i are earned upon the successful execution of methods m_i from sequences $\phi = (...,m_i)$ before time Δ[3] and thus, the value $V(\pi)$ of the joint policy $\pi = (\pi_1,...,\pi_n)$ is:

$$V(\pi) = \sum_{n=1}^{N} V^{\pi_n}(s_{n,0}) = \sum_{\substack{\phi=(...,m_i)\in\cup_n\Phi_n \\ Q=(...,q_i)\in\{0,1\}^{|\phi|}}} F^{\pi}_{\phi,Q}(\Delta)\cdot q_i \cdot r_i \quad (1)$$

The optimal solution F^* to the dual multiagent problem must then maximize the right-hand-side of Equation (1). Furthermore, F^* must be an element of a set $X = \{F : (2),(3),(4)\}$ where constraints (2), (3), (4) are defined for agents $1 \le n \le N$, sequences $\phi \in \Phi_n$ and outcomes $Q \in \{0,1\}^{|\phi|}$ as follows:

$$F_{(),()}(m_{n,0})(t) = 1 \quad (2)$$

$$F_{\phi,Q}(t) = \sum_{m\in A_{\phi,Q}} F_{\phi,Q}(m)(t) \quad (3)$$

$$F_{(\phi|m),(Q|q)}(t) = \int_0^t \widetilde{F}_{\phi,Q}(m)(t')\cdot p_m(t-t') \cdot E_{(\phi|m),(Q|q)}(t')dt' \quad (4)$$

Constraint (2) ensures that there is a method $m_{n,0} \in M_n$ that agent n will execute first. (For $m \in M_n$ such that $m \neq m_{n,0}$ we automatically have $F_{(),()}(m)(t) = 0$.) Constraint (3) can be seen as the conservation of probability mass flow through the action-outcome sequence ϕ, Q. Applicable only if $A_{\phi,Q} \neq \emptyset$ it ensures that the cumulative distribution function $F_{\phi,Q}$ is split into cumulative distribution functions $F_{\phi,Q}(m)$ for methods $m \in A_{\phi,Q}$. Constraint (4) ensures the correct loss of probability mass from functions $\widetilde{F}_{\phi,Q}(m)$ to functions $F_{(\phi|m),(Q|q)}$ caused by the execution of method m. The constraint uses function $p_m(t)$ (the duration of execution of method m, in relative time, given by the model) and function $E_{(\phi|m),(Q|q)}(t')$ (explained in Section 3.3), which for $q = 1$ specifies the probability that method m has been enabled before time t' and for $q = 0$ specifies the probability that method m has *not* been enabled before time t', for a given action-outcome sequence ϕ, Q. In this context, constraint (4) reads that the execution of method m will finish *successfully* at time t if it started at time t' with m being enabled and takes $t - t'$ time units to finish. Consequently, we have explained all the necessary conditions for F to belong to the set X.

Observe, that there can be many deterministic policies π^* implied by F^* found by M-DPFP. In essence, F^* cannot be solely used to determine a unique π^* as F^* does not contain enough information about π^*— some of this information is irreversibly lost in M-DPFP's abstraction of observation histories. To alleviate this shortcoming, M-DPFP provides a technique for constructing a deterministic policy $\widehat{\pi}^*$ from F^* that guarantees a a near-optimal behavior of the agent team (the error of $\widehat{\pi}^*$ is bounded in Section 3.3). $\widehat{\pi}^*$ is defined as follows: When agent n transitions at time $\mathbf{t}$ to state $s = (e_1,...,e_k) \in \mathcal{S}_n$ where $e_i = \langle m_i, *, *, q_i\rangle$ for $1 \le i < k$ and $e_k = \langle m_k, *, \mathbf{t}, q_k\rangle$ ($*$ is an arbitrary time earlier than t), the agent will wait $\delta^*_{\phi,Q}(m)(t)$ units of time and then execute method $\pi^*_{\phi,Q}(t) := m$, where m is given by:

$$m = \arg\max_{m'\in A_{\phi,Q}} \{\lim_{\epsilon\to 0} F^{\pi^*_n}_{\phi,Q}(m')(t+\epsilon) - F^{\pi^*_n}_{\phi,Q}(m')(t)\} \quad (5)$$

We finally show how to derive $\delta^*_{\phi,Q}(m)$ used by $\widehat{\pi}^*$. Suppose that the agent finished executing methods from action sequence ϕ with outcomes Q at time t and, according to policy $\widehat{\pi}^*$, chose method $m \in A_{\phi,Q}$ to be executed next. The calculation of the optimal time $t' := t + \delta^*_{\phi,Q}(m)(t)$ at which the agent should start executing m takes into account the probability $E_{(\phi|m),(Q|1)}(t')$ that m will be enabled before time t' and the probability $P_m(t') := \int_0^{\Delta-t'} p_m(x)dx$ that the execution of m, if started at time t', will finish before time Δ. Precisely, let $G_{\phi,Q}(m)(t) := E_{(\phi|m),(Q|1)}(t)\cdot P_m(t)$ be the probability that method m will be executed successfully if started at time t. The agent then starts the execution of method m at times that correspond to the local maxima of $G_{\phi,Q}(m)$:

$$\delta^*_{\phi,Q}(m)(t) = \begin{cases} t'-t & \text{if } \exists t' > t : G_{\phi,Q}(m)(t') > G_{\phi,Q}(m)(t) \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Functions $\delta^*_{\phi,Q}(m)$ and $F^*_{\phi,Q}(m)$ can then be used to determine functions $\widetilde{F}^*_{\phi,Q}(m)$ as demonstrated in Fig.3.

[3]For expository purposes we assume that agents start the execution of their methods at time 0 and that method execution time windows are $[0,\Delta]$— extensions to different time windows are straightforward. Also, we simplify the notation by dropping n from $F^{\pi_n}_{\phi,Q}(\Delta)$.

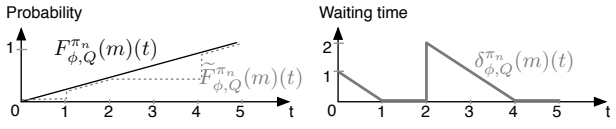

Figure 3: Using the agent waiting time function $\delta^{\pi_n}_{\phi,Q}(m)$ and function $F^{\pi_n}_{\phi,Q}(m)$ to derive function $\widetilde{F}^{\pi_n}_{\phi,Q}(m)$.

3.3 Solving the Multiagent Dual Problem

In general, the multiagent dual problem is difficult to solve optimally because, when method execution duration distributions are continuous, the set X where F^* is to be found is infinite. Yet, even if action duration distributions are continuous, the multiagent dual problem can be solved near-optimally with guarantees on solution quality. Essentially, our algorithm builds upon the idea in [11] to restrict the search for F^* to finite number of elements in X by pruning from X the elements F that correspond to reaching regions of the state-space with very low probability (the expected quality loss for executing suboptimal actions in these regions can be bounded). Similarly to the DPFP algorithm [11], our algorithm searches for F^* in set $\widehat{X} \subset X$ where $\widehat{X}$ differs from X in that values of functions F in $\widehat{X}$ are restricted to integer multiples of a given $\kappa \in \Re^+$. Informally, κ creates a step function approximation of F. Formally, $\widehat{X} = \{F : (2), (3), (7), (8), (9)\}$ where

$$F'_{(\phi|m),(Q|q)}(t) = \int_0^t \widetilde{F}_{\phi,Q}(m)(t') \cdot p_m(t-t') \cdot E_{(\phi|m),(Q|q)}(t')dt' \quad (7)$$

$$F_{\phi,Q}(t) = \lfloor F'_{\phi,Q}(t)/\kappa \rfloor \cdot \kappa \quad (8)$$

$$F_{\phi,Q}(m)(t) = \kappa \cdot k \text{ where } k \in N \quad (9)$$

We refer to the problem of finding the optimal solution $\widehat{F^*} \in \widehat{X}$ (i.e., the solution that maximizes the right-hand-side of Equation 1) as the *restricted multiagent dual problem*. Similarly to DPFP, M-DPFP solves the restricted problem by iterating over all elements of $\widehat{X}$ (notice, that $\widehat{X}$ is finite). However, the fundamental difference between M-DPFP and DPFP is that M-DPFP solves the *multiagent* dual problem and must thus construct candidate solutions (elements of $\widehat{X}$) in a way that respects the agent interactions. Precisely, M-DPFP's solutions must be build in a specific way, i.e., adding policies for action sequences in an ordering $\mathcal{L}$ discussed in Section 3.1. We first show this process on an example and later bound the total error of M-DPFP.

Fig.4 shows M-DPFP in action for the domain in Fig.1 and the agent team interaction graph in Fig.2. Assume $\kappa = 0.2$. M-DPFP's iteration over all elements in $\widehat{X}$ begins with the iteration over all the combinations of methods that agents can execute first — for each combination, M-DPFP separately calls a recursive function FINDSPLITTING(1), to iterate over all policies where the given combination of methods is executed first. We now focus on one such call, where agent 1 first executes method m_2 and agent 2 first executes method m_6.[4]

At a basic level, the goal of FINDSPLITTING(p) is to iterate over policies for action sequences that are *on or after* position p on list $\mathcal{L}$, assuming that policies for action sequences that are before position p on list $\mathcal{L}$ are fixed. In particular, our example in Fig.4 demonstrates how FINDSPLITTING(7) works for the action sequence (m_6) with label 7 on the agent team interaction graph in Fig.2. FINDSPLITTING(7) then works as follows: it first uses constraints (3), (7) to derive $F'_{(m_6)(1)}$ (solid gray line for action-outcome sequence $(m_6)(1)$) from $F_{()()}(m_6)$ and $E_{(m_6),(1)}$ (Note, that $E_{(m_6),(1)}(t) = 1$ because method m_6 is always enabled) and then uses constraint (8) to approximate $F'_{(m_6)(1)}$ with a step function $F_{(m_6)(1)}$ (solid black line for for action-outcome sequence $(m_6)(1)$)[5]. At this point M-DPFP knows the probability $F_{(m_6)(1)}(t)$ that m_6 will be executed successfully *before* time t but does not know what method will be chosen to be executed next, i.e., does not know the probabilities $F_{(m_6)(1)}(m_4)(t)$, $F_{(m_6)(1)}(m_5)(t)$ (dotted black lines for action-outcome sequence $(m_6)(1)$). Thus, to iterate over all the elements in $\widehat{X}$, M-DPFP must iterate over all $|A_{(m_6)(1)}|^{F_{(m_6)(1)}(\Delta)/\kappa} = 16$ pairs $\{F_{(m_6)(1)}(m_4), F_{(m_6)(1)}(m_5)\}$ (called different *splittings* of $F_{(m_6)(1)}$). A splitting determines the policy (see Equation 5): For the specific splitting shown, if the execution of m_6 finishes successfully at times t_1, t_2, t_4, the agent will choose to execute m_4 next whereas if the execution of m_6 finishes successfully at time t_3, the agent will choose to execute m_5 next. M-DPFP then extrapolates this point-based policies onto time interval $[0, \Delta]$ as follows: If $\widehat{\pi}^*_{(m_6),(1)}(t_2) = m_4$, $\widehat{\pi}^*_{(m_6),(1)}(t_3) = m_5$, and $\widehat{\pi}^*_{(m_6),(1)}(t)$ is undefined for $t \in (t_2, t_3)$, M-DPFP puts $\widehat{\pi}^*_{(m_6),(1)}(t) := m_4$ for all $t \in (t_1, t_2)$.

At this point, FINDSPLITTING(7) still needs to determine how long agent 2 should wait before starting the execution of methods m_4 or m_5, i.e, it needs to determine $\delta_{(m_6)(1)}(m_4)$ and $\delta_{(m_6)(1)}(m_5)$. Since method m_5 is always enabled, it holds that $E_{(m_5),(1)}(t) = 1$ and thus, $G_{(m_5),(1)}$ is monotonically decreasing, which implies that $\delta_{(m_6)(1)}(m_5)(t) = 0$. Now, to determine $\delta_{(m_6)(1)}(m_4)$, M-DPFP must first find $E_{(m_4),(1)}(t)$ for all $t \in [0, \Delta]$ (Notice, that $E_{(m_4),(1)}(t)$ cannot be equal to 1 for all $t \in [0, \Delta]$ because it takes a non-zero amount of time to execute method m_1 that enables method m_4.) In our implementation of M-DPFP we estimate $E_{(m_4),(1)}(t)$ using a Monte Carlo sampling technique. In essence, a partially constructed policy (fixed for the action sequences on positions 1—6 on list $\mathcal{L}$) allows us to sample the CR-DEC-MDP from the starting state and, given enough samples, accurately estimate the probability that method m_1 will be completed before time t according to the partially constructed policy. We then use Equation 6 to derive $\delta_{(m_6)(1)}(m_4)(t)$. Having fixed a policy for the action sequence (m_6), FINDSPLITTING(7) executes a recursive call to FINDSPLITTING(8) to iterate over all the policies for the action sequence (m_6, m_5). Here, the algorithm behaves similarly as for sequence (m_6). However, since agent 2's only option here is to execute method m_4, the only policy to be considered is $F_{(m_6,m_5)(11)}(m_4) = F_{(m_6,m_5)(11)}$. (Similarly for the outcome sequences $(10), (01), (00)$.) Upon fixing the policy for the action sequence (m_6, m_5) M-DPFP moves forward, i.e., FINDSPLITTING(8) executes a recursive call to FINDSPLITTING(9) etc. and this recursive policy construction process continues until FINDSPLITTING(30) is called at which point the complete joint policy has been constructed.

M-DPFP intertwines policy construction with policy evaluation. Specifically, when FINDSPLITTING(P) is called, the joint policy is *incomplete* (only policies for action sequences on positions $1, ..., p-1$ on list $\mathcal{L}$ are known), yet, it already yields the expected reward of $\sum_{\phi=(...,m_i)\in\Phi:pos(\phi,\mathcal{L})<p;Q=(...,q_i)\in\{0,1\}^{|\phi|}} F_{\phi,Q}(\Delta) \cdot r_i \cdot q_i$. To maximize the expected reward for the *complete* policy, M-DPFP must iterate over all possible policies for action sequences on positions $p, ..., |\mathcal{L}|$ on list $\mathcal{L}$, to find the set of functions $\{F_{\phi,Q} : \phi \in$

[4]For this particular call to FINDSPLITTING(1), constraint 2 sets $F_{()()}(m_2)(t) = 1$ and $F_{()()}(m_6)(t) = 1$, which then implies that $F_{(m,...)(*)}(t) = 0$ for $m \notin \{m_2, m_6\}$. For expository purposes, in the description below we do not stress if $F_{(*)(*)}(t) = 0$.

[5]The same operations need to be performed for the action-outcome sequence $(m_6)(0)$ in Fig.4.

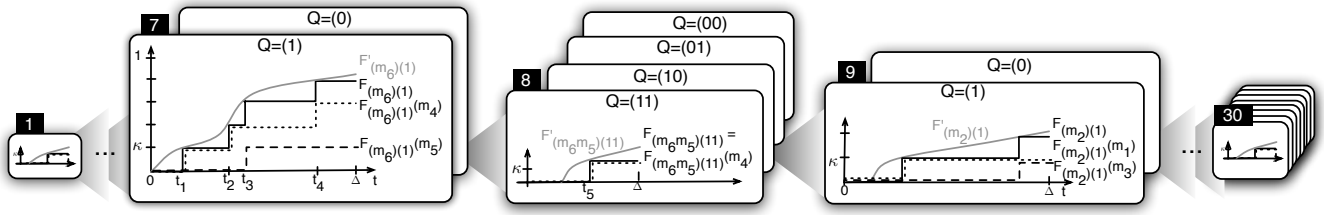

Figure 4: Search for the optimal solution to the restricted multiagent dual problem

$\Phi; pos(\phi, \mathcal{L}) \geq p; Q \in \{0,1\}^{|\phi|}\}$ that maximizes the objective from Equation 1. The algorithm then backtracks, i.e., it considers another incomplete policy—by picking another policy for the action sequence on position $p-1$ on list $\mathcal{L}$—and repeats the policy completion/evaluation process. M-DPFP terminates once all the incomplete policies, i.e., all the policies for the action sequence on position 1 on list $\mathcal{L}$ have been considered.

The total error of M-DPFP (the error of projection of X onto $\widehat{X}$ combined with the error of abstraction of observation histories) can be bounded as follows: We first borrow from [11] that the bound on the error of projection of X onto $\widehat{X}$ is:

$$\max_{t\in[0,\Delta]} |F^*_{\phi,Q}(t) - \widehat{F}^*_{\phi,Q}(t)| \leq \kappa|\phi|$$

for all $\phi \in \cup_n \Phi_n$ and $Q \in \{0,1\}^{|\phi|}$ and hence (refer to Equation 3), $\max_{t\in[0,\Delta]} |F^*_{\phi,Q}(m)(t) - \widehat{F}^*_{\phi,Q}(m)(t)| \leq \kappa|\phi|$. Now, in order to bound the error of M-DPFP's abstraction of observation histories, assume temporary that $\widehat{F}^*$ is the optimal solution to the *non-restricted* multiagent dual problem. In such case, whenever $\widehat{F}^*_{\phi,Q}(m)$ increases for a given ϕ, Q (e.g. at time $t \in [0,\Delta]$), there is a probability κ chance that the agent will choose method $m \in A_{\phi,Q}$ to be executed next. For example, there is a probability κ chance that agent 2 in Fig.4 will finish executing methods (m_6, m_5) with outcomes $(1,1)$ at time t_5 and choose m_4 to be executed next. In this context, M-DPFP's abstraction of observation histories implies that the entire probability mass κ of a stair $\widehat{F}^*_{\phi,Q}(t)$ must be assigned to a unique method $m \in A_{\phi,Q}$ to be executed next. In contrast, without the abstraction, this probability mass κ can be split and assigned to various methods $m' \in A_{\phi,Q}$ that the agent can execute next, where the choice of m' depends on the actual times when methods from ϕ have been completed (that M-DPFP's abstraction ignores). Thus, there is at most probability κ chance that the agent will *not* choose at time t the optimal method to be executed next. Now, relaxing our assumption that $\widehat{F}^*$ is the optimal solution to the *non-restricted* multiagent dual problem, the probability that the agent will *not* choose at time t the optimal method to be executed next is at most $\kappa|\phi| + \kappa$. The total error of M-DPFP is thus:

$$\epsilon \leq \sum_{\substack{\phi=(\ldots,m_i)\in\Phi \\ Q\in\{0,1\}^{|\phi|}}} \kappa(|\phi|+1)r_i \qquad (10)$$

4. EXPERIMENTS

We implemented M-DPFP and evaluated it on a variety of CR-DEC-MDPs. Since M-DPFP is the only technique available that provides solution quality guarantees when solving arbitrary CR-DEC-MDPs, we first examined M-DPFP's performance alone, when applied to the decision problem in Fig.1. Here, we fixed the method execution durations to Normal distributions $\mathcal{N}(\mu = 3, \sigma = 1)$, set the deadline to $\Delta = 10$ and run M-DPFP with accuracy settings $\kappa = 0.25, 0.2, 0.15, 0.1$. In this context, we used the formula from Equation 10 to established the upper bound on the error

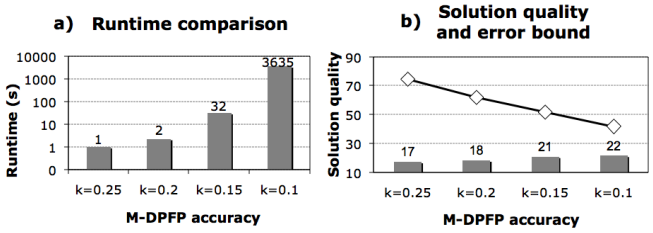

Figure 5: Error bounded solutions of M-DPFP when planning with continuous time.

of solutions returned by M-DPFP. As can be seen (Fig.5), as M-DPFP's accuracy increases (i.e., parameter κ decreases), the quality of M-DPFP's solution (gray bars in Fig.5(b)) increases, while at the same time, the theoretical error bound (calculated in terms of κ and shown in Fig.5(b) as a solid line) decreases and approaches the solution quality returned by M-DPFP (the error converges to 0). In particular, M-DPFP with $\kappa = 0.1$ needs 3635 seconds to find a solution of quality 22 that is guaranteed to be *at least* 50% of the optimal quality. This result is particularly significant, as there is currently no alternative algorithm for solving problems such as in Fig.1 that returns error bounded solutions.

Indeed, the only way to compare M-DPFP with other globally optimal algorithms is to restrict our attention to CR-DEC-MDPs with method execution durations sampled from discrete probability density functions. In essence, for such distributions the planning problems automatically become discrete (the number of CR-DEC-MDP states is finite) and can be modeled as Decentralized POMDPs. In this context, we implemented the SPIDER algorithm [17] constrained to 2 agents (where it can be transition dependent) and compared it to M-DPFP. (Notice that M-DPFP is not specialized for and hence may not work well for discrete distributions.) The result of this comparison is shown in Tables 1 and 2. Here, we gradually increase the domain size (from 4 to 12 methods), measure the runtimes and compare the quality of solutions found by SPIDER and M-DPFP when run with various accuracy settings ($\kappa = 0.25, 0.2, 0.15$). Across all the domains, the time horizon is set to $\Delta = 10$ time ticks and method execution durations are sampled from a discrete version of the Normal distributions $\mathcal{N}(\mu = 3, \sigma = 1)$. Note the non-apparent scale of these domains. For example, there are $841,807$ distinct belief states when modeling our domain with 8 methods using Decentralized POMDPs.

When run on the domain with 4 methods, both SPIDER and M-DPFP found solutions with exactly the same quality (Table 2, row 2), yet M-DPFP completed this task up to two orders of magnitude faster (Table 1, row 2). In particular, M-DPFP with $\kappa = 0.25$ terminated after only $0.2s$ whereas SPIDER needed $21s$ to finish. We hypothesize that the surprisingly high quality of M-DPFP solutions for this domain is a consequence of the existence of a solution

where all the methods have a 100% chance of being completed before the deadline. Indeed, only when we consider bigger domains (with $6+$ methods) where there is no solution that guarantees that all the methods will be completed before the deadline, M-DPFP's approximation starts contributing to the loss of solution quality (Table 2, rows 3—6). However, for these domains, M-DPFP can find high quality solutions much faster than SPIDER. In particular, for a domain with 6 methods, M-DPFP with $\kappa = 0.2$ finds a solution of quality 23 (less than 25% off an optimal solution) in just under $33s$ whereas SPIDER needs over $995s$ to terminate with an optimal solution (Table 1, row 3). Finally, as we scale-up the domain size to double-digit method numbers, we observe that, SPIDER fails to terminate with a solution whereas M-DPFP continuous to find solutions of high quality. We hence conclude that, although M-DPFP is designed primarily for continuous time problems, it still demonstrates superior performance in terms of its ability to quickly find high-quality solutions when time is discretized.

Number of methods	M-DPFP $\kappa = 0.25$	M-DPFP $\kappa = 0.2$	M-DPFP $\kappa = 0.15$	SPIDER
4	0.2	0.5	6	21
6	19	33	591	995
8	17	1122	6031	26190
10	30	1281	n/a	n/a
12	29	4162	n/a	n/a

Table 1: Runtimes (in seconds) of M-DPFP and SPIDER

Number of methods	M-DPFP $\kappa = 0.25$	M-DPFP $\kappa = 0.2$	M-DPFP $\kappa = 0.15$	SPIDER
4	28	28	28	28
6	16	23	25	31
8	18	27	29	37
10	46	46	n/a	n/a
12	42	47	n/a	n/a

Table 2: Solution quality of M-DPFP and SPIDER

5. CONCLUSIONS

Many real world planning problems require distributed reasoning with continuous resources and resource limits. Unfortunately, prior approaches have failed to address this problem by either discretizing resources, which automatically removes quality guarantees, or finding only locally optimal solutions to a substantially restricted version of the problem. To address these shortcomings, we introduced M-DPFP, a new algorithm for planning with continuous resources in a multiagent setting. The key idea of M-DPFP is first, to represent the planning problem as continuous resource, decentralized MDP, then, to build the agent team interaction graph to quickly identify and prune suboptimal policies and finally, to perform a forward search for policies in a continuous space of probability functions. Our experiments revealed that M-DPFP not only provides solutions with quality guarantees when resources are continuous, but also, that it finds high quality solutions and exhibits superior performance when resources are discrete.

In terms of related work, algorithms for solving decentralized MDPs or POMDPs [1], [2], [4], [14], [15], [16] all assume that resources are discrete and if that is not the case, fail to provide solution quality guarantees. On the other hand, existing planners that handle continuous resources properly [7], [8], [9], [11], [12] are tailored to single agent systems as they fail to address the lack of global state knowledge, a fundamental issue in decentralized planning. The remaining techniques [3], [10] have been very successful in leveraging planning with continuous resources to the multi-agent world. Yet, [3] assumes that agents are transition independent whereas [10] imposes a fixed ordering of agent actions and only finds locally optimal solutions.

6. ACKNOWLEDGMENTS

This research was partially supported by the United States Department of Homeland Security through the Center for Risk and Economic Analysis of Terrorism Events (CREATE) under grant number 2007-ST-061-000001. However, any opinions, findings, and conclusions or recommendations in this document are those of the authors and do not necessarily reflect views of the United States Department of Homeland Security.

7. REFERENCES

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

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-Independent Dec-MDPs. In *AAMAS*, 2003.

[3] E. Benazera. Solving decentralized continuous Markov decision problems with structured reward. In *KI*, 2007.

[4] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of MDPs. In *UAI*, 2000.

[5] A. Beynier and A. Mouaddib. An iterative algorithm for solving constrained Dec-MDPs. In *AAAI*, 2006.

[6] K. Decker and V. Lesser. Designing a Family of Coordination Algorithms. *ICMAS-95*, January 1995.

[7] Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous MDPs. In *UAI*, 2004.

[8] L. Li and M. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI*, 2005.

[9] J. Marecki, S. Koenig, and M.Tambe. A fast analytical algorithm for solving MDPs with real-valued resources. In *IJCAI*, 2007.

[10] J. Marecki and M. Tambe. On opportunistic techniques for solving Dec-MDPs with temporal constraints. In *AAMAS*, 2007.

[11] J. Marecki and M. Tambe. Towards faster planning with continuous resources in stochastic domains. In *AAAI*, 2008.

[12] Mausam, E. Benazera, R. I. Brafman, N. Meuleau, and E. A. Hansen. Planning with continuous resources in stochastic domains. In *IJCAI*, 2005.

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

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

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

[16] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *UAI*, 2000.

[17] P. Varakantham, J. Marecki, M. Tambe, and M. Yokoo. Letting loose a spider on a network of pomdps: Generating quality guaranteed policies. In *AAMAS*, 2007.

[18] H. Younes and R. Simmons. Solving generalized semi-MDPs using continuous phase-type distributions. In *AAAI*, 2004.