Solving Continuous-Time Transition-Independent DEC-MDP with Temporal Constraints

Zhengyu Yin[1], Kanna Rajan[2], and Milind Tambe[1]
[1]University of Southern California, Los Angeles, CA 90089, USA
{zhengyuy, tambe}@usc.edu
[2]Monterey Bay Aquarium Research Institute, Moss Landing, CA 95039, USA
kanna.rajan@mbari.org

ABSTRACT

Despite the impact of DEC-MDPs over the past decade, scaling to large problem domains has been difficult to achieve. The scale-up problem is exacerbated in DEC-MDPs with continuous states, which are critical in domains involving time; the latest algorithm (M-DPFP) does not scale-up beyond two agents and a handful of unordered tasks per agent.

This paper is focused on meeting this challenge in continuous resource DEC-MDPs with two predominant contributions. First, it introduces a novel continuous time model for multi-agent planning problems that exploits transition independence in domains with graphical agent dependencies and temporal constraints. More importantly, it presents a new, iterative, locally optimal algorithm called SPAC that is a combination of the following key ideas: (1) defining a novel *augmented* CT-MDP such that solving this single-agent continuous time MDP provably provides an automatic best response to neighboring agents' policies; (2) fast convolution to efficiently generate such augmented MDPs; (3) new enhanced lazy approximation algorithm to solve these augmented MDPs; (4) intelligent seeding of initial policies in the iterative process; (5) exploiting graph structure of reward dependencies to exploit local interactions for scalability. Our experiments show SPAC not only finds solutions substantially faster than M-DPFP with comparable quality, but also scales well to large teams of agents.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Theory

Keywords

Multiagent systems, Decentralized Markov Decision Process, Continuous Time

1. INTRODUCTION

Since the introduction of decentralized Markov Decision Processes (DEC-MDPs) to the field of multiagent systems over a decade ago, there has been significant progress in improving their efficiency [1, 8, 16]. Yet given the NEXP-complete complexity of DEC-MDPs [4] scale-up has been difficult. This challenge is further exacerbated in many real-world domains where we wish to apply DEC-MDPs: these involve continuous resources such as time or energy, and actions may involve uncertainty in their resource consumption (e.g. duration or energy consumption). And these domains often require that we deploy teams of agents, e.g. large numbers of autonomous underwater vehicles for scientific observations in the ocean [6], unmanned aerial vehicles for surveillance or large autonomous mobile sensor webs deployed for disaster response.

The state-of-the-art in continuous time planning for DEC-MDPs often fails to meet this challenge. One of the latest algorithms, M-DPFP [12], is an attempt to find a global optimal for continuous time DEC-MDPs, where agents must coordinate over an unordered set of tasks. Unfortunately M-DPFP cannot scale-up beyond two agents and a handful of tasks. Other attempts in planning with continuous time for DEC-MDPs [5, 11] have successfully solved problems with much larger number of tasks, but require that the task ordering be supplied ahead of time without a significant number of agents. There has been some success in scale-up in discrete state planning, in models such as ND-POMDPs [13] that exploit transition independence [1] and a network structure; for example ND-POMDPs have shown results for up to a dozen agents. While we build on some of the key ideas in ND-POMDPs, their discrete state approximation of continuous time domains can lead to significant degradation in solution quality (coarse-grained discretization) or very large inefficiencies (fine-grained discretization) [12].

This paper presents two key contributions, to meet the challenges of continuous time DEC-MDPs. First, we introduce a novel continuous time model (MCT-MDP) for multiagent planning problems that exploits transition independence in domains with graphical agent dependencies and temporal constraints. This model is motivated by domains such as ones dicussed in Section 2. More importantly, we present a new iterative locally optimal algorithm called SPAC that is a combination of the following key ideas: (1) defining an augmented CT-MDP such that solving this single-agent continuous time MDP provably provides a best response to neighboring agents' policies; (2) fast convolution to efficiently generate such augmented MDPs; (3) a new enhanced Lazy Approximation algorithm to solve these augmented MDPs; (4) intelligent seeding of initial policies in the iterative process; (5) exploiting graph structure of reward dependencies for scale-up. Our experiments show

The Sixth Annual Workshop on Multiagent Sequential Decision-Making in Uncertain Domains (*MSDM-2011*), held in conjunction with *AAMAS-2011* on May 3, 2011 in Taipei, Taiwan.

SPAC not only finds solutions substantially faster than M-DPFP with comparable quality, but also scales well to large team of agents – it can solve a 1000-agent problem with 5 unordered tasks per agent in 8 minutes.

2. MOTIVATING PROBLEM

An important domain that motivates our work is with the use of Autonomous Underwater and Surface Vehicles (AUVs and ASVs). These are untethered mobile robots used for collecting data and returning targeted water samples from within dynamic and unstructured coastal features [14]. To scale the existing on-board automated planning techniques to observe and sample large scale spatio-temporal fields we need to consider *teams of robots* working in a coordinated fashion to sample both the spatial extent and rapid temporal changes within different water columns.

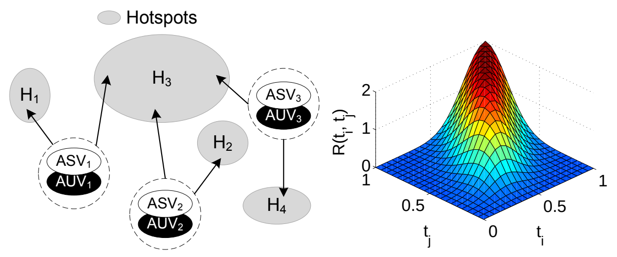
For example, we consider a 6-agent scenario shown in Figure 1. A team of three pairs of AUV/ASV is assigned to collect sensor data and water samples in pre-identified biological hotspots which characterize a Harmful Algal Bloom. Depending on the size of the hotspot, multiple agent pairs may be required to sample the area. For example, in Figure 1a, H_1, H_2, and H_4 are three small hotspots that require only one pair of AUV/ASV each, while H_3 is a large area that requires all three pairs to perform the sampling task simultaneously. While the ASVs remain on the surface to take surface samples, their corresponding AUVs travel below the surface periodically taking underwater samples cotemporally. Actions such as traveling between hotspots, performing a sampling task, surfacing to get a GPS fix etc. may have uncertain durations with known distributions. Note that in this problem, one agent's action will not affect other agents' physical states and possible actions. In DEC-MDP research, this type of transition independence has been motivated earlier in many domains [1, 13].

Each single task has a reward associated with it; but the reward obtained may be based on the joint actions of multiple agents in the team. In particular, there are several types of temporal constraints within different sets of tasks. For example, we require the underwater samples to be taken simultaneously with the surface samples, which is modeled as a joint reward function. An example function is shown in Figure 1b, where the x and y axes represent the remaining time when the two tasks are started and the z value represents the corresponding joint reward. Similarly, we have temporal constraints between pairs of agents as well as for large hotspots, which can also be modeled as joint reward functions. We may also have precedence constraints that require one water sample to be taken before another, etc. While this scenario is illustrated with six agents, in general we can imagine a much larger team.

Our goal in this domain is to compute an optimal plan that maximizes the total expected reward for the potentially large agent teams, taking into account the uncertainties in the ocean domain (location and travel time) and potentially the complex set of continuous temporal constraints. Since a number of mobile sensors may be required for a large scientific mission, it is particularly important to design an efficient planner that can scale to a large number of agents.

3. MCT-MDP

As our motivating domain illustrates, one agent's rewards



(a) (b)

Figure 1: (a) AUV/ASV teams for ocean sampling; (b) An example reward function for a pair of simultaneous tasks.

may be dependent on other agent's state and actions, but its transitions are not. In DEC-MDP research, these types of transition-independent problems have been motivated earlier in many domains with discrete states [2, 13]. To efficiently model time as part of an agent's local state, we formulate these types of planning problems as continuous-time transition-independent DEC-MDPs with soft temporal constraints. We first define a single-agent problem and then a decentralized one.

Definition 1. *A single-agent continuous resource MDP (CT-MDP) $\mathcal{M}$ is represented by a tuple $\langle S, \Delta, A, T, \sigma, R\rangle$, where,*

- *S is a finite set of discrete states. s^0 denotes the initial state.*
- *$\Delta = [0, t^*]$ is a one-dimensional continuous state representing the remaining time. The agent initially has t^* remaining time. $S \times \Delta$ defines the hybrid state space of the agent. We say the agent is at hybrid state $\langle s, t\rangle$, if its discrete state is s and remaining time is t.*
- *A is a finite set of actions. In addition, the agent can wait for an arbitrary amount of time. Wait action is denoted by ϕ (Wait is critical in multiagent settings, e.g. one agent may need to wait to perform its task as it may be dependent on another agent finishing its task).*
- *$T : S \times A \times S \mapsto \mathbb{R}$ is the transition function of discrete states. $T(s, a, s')$ is the probability of entering s' when action a is taken at state s.*
- *$\sigma : \Delta \times S \times A \times S \mapsto \mathbb{R}$ is the relative transition density function of the continuous states. $\sigma(t|s, a, s')$ is the probability density function (PDF) of the duration of taking action a at s and reaching s'. We assume there is a minimum duration for all non-wait actions.*
- *$R : \Delta \times S \times A \mapsto \mathbb{R}$ is the individual reward function. $R(t|s, a)$ is the immediate reward obtained when the agent takes action a at hybrid state $\langle s, t\rangle$.*

Definition 2. *The policy π is a mapping from $\langle s, t\rangle$ to an action a. Denote such mapping by $\pi(s, t) = a$. We define the policy function $\alpha_\pi(t|s, a)$:*

$$\alpha_\pi(t|s,a) = \begin{cases} 1, & \text{if } \pi(s,t) = a, \\ 0, & \text{otherwise.} \end{cases}$$

Definition 3. *An event $e = \langle s, a\rangle$, is a pair of a discrete state and a non-wait action. The set of all events $E = S \times A$. In this paper, we use e and $\langle s, a\rangle$ interchangeably.*

Definition 4. *Given policy π, $f_\pi(t|e)$ is the PDF that event e happens with remaining time t.*

The value of a policy π is the non-discounted expected reward of R: $V(\pi) = \sum_{e\in E}\int_{t=0}^{t^*} R(t|e) f_\pi(t|e)dt$.

Definition 5. *An n-agent transition-independent multi-agent CT-MDP (MCT-MDP) is defined by $\langle\{\mathcal{M}_i\}, \Omega, \mathcal{R}\rangle$.*

- *$\mathcal{M}_i = \langle S_i, \Delta_i = [0, t_i^*], A_i, T_i, \sigma_i, R_i\rangle$ is the individual CT-MDP of agent i.*
- *Ω is a set of reward-dependent joint events. A joint event is denoted by $\mathbf{e_c}$, where $\mathbf{c} = \{c_1, \ldots, c_{|\mathbf{c}|}\}$ is a sub-group of agents and $\mathbf{e_c} = \{e_{c_1}, \ldots, e_{c_{|\mathbf{c}|}}\}$ is a joint event defined on $\mathbf{c}$.*
- *For every $\mathbf{e_c} \in \Omega$, $\mathcal{R}(t_{c_1}, \ldots, t_{c_{|\mathbf{c}|}}|\mathbf{e_c})$, defines the joint reward received if for every agent $c_i \in \mathbf{c}$, its event e_{c_i} happens at t_{c_i}.*

Ω determines the graphical reward dependencies among agents, i.e. two agents are reward dependent if and only if there exists a joint event in Ω which comprises both agents' individual events. $\mathcal{R}$ captures the temporal constraint on dependent events by rewards. For example, $\mathcal{R}(t_1, t_2|e_1, e_2)$ is the reward function between e_1 of agent 1 and e_2 of agent 2. The global value of a joint policy $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_n\}$ is the sum of non-discounted expected reward of both individual rewards R_i and joint rewards $\mathcal{R}$. Thus,

$$GV(\boldsymbol{\pi}) = \sum_{i=1}^{n} V_i(\pi_i) + \sum_{\mathbf{e_c}\in\Omega} JV(\boldsymbol{\pi}_\mathbf{c}|\mathbf{e_c}),$$

$$JV(\boldsymbol{\pi}_\mathbf{c}|\mathbf{e_c}) = \int\int\ldots\int_{t_{c_i}=0,\forall c_i\in\mathbf{c}}^{t^*_{c_i}} \mathcal{R}(t_{c_1}, \ldots, t_{c_{|\mathbf{c}|}}|\mathbf{e_c}) \left(\prod_{c_i\in\mathbf{c}} f_{\pi_{c_i}}(t_{c_i}|e_{c_i})dt_{c_i}\right).$$

Here $\boldsymbol{\pi}_\mathbf{c} = \{\pi_{c_1}, \ldots, \pi_{c_{|\mathbf{c}|}}\}$ denotes the policies on sub-group $\mathbf{c}$, where π_{c_i} is the policy for agent c_i. In this paper, we will represent $R(t|e)$ by piecewise constant functions and $\mathcal{R}(t_{c_1}, \ldots, t_{c_{|\mathbf{c}|}}|\mathbf{e_c})$ by piecewise hyper-rectangular constant functions (values are constant in each hyperrectangles).

4. SPAC: LOCALLY OPTIMAL ALGORITHM

From an initial joint policy, SPAC (Scalable Planning for Agent teams under Continuous temporal constraints) obtains a locally optimal solution by iteratively finding the best response of one agent to its neighboring agents' policies. For finding a best response, a naive approach is to enumerate and evaluate all policies of the best-responding agent given fixed policies of its neighboring agents. However, this is infeasible in an MCT-MDP because first, there are infinite number of states given continuous remaining time; second, there are infinite number of decision choices since an agent can wait any amount of time. To address this difficulty, we present a novel *augmented* CT-MDP (defined below) to efficiently compute the best response of an agent. SPAC optimizes the joint policy as follows: (1) create the set of augmented CT-MDPs for each agent i with respect to other agents' policies π_{-i}; (2) find new optimal policy π_i^* by solving the augmented CT-MDP, and update the gain $dV_i = \tilde{V}_i(\pi_i^*) - \tilde{V}_i(\pi_i)$; (3) terminate if the maximum gain is smaller than a given threshold η, otherwise find a set of mutually reward independent agents with high overall gain greedily, update their policies from π_i to π_i^*, and repeat the process from the first step. The pseudo code is shown in Algorithm 1. Next we will discuss the two sub-routines in Algorithm 1: (1) quickly create the augmented CT-MDP for an agent given its neighbors' policies (line 4); (2) solve it efficiently using piecewise constant approximations inspired by [9] (line 5).

Algorithm 1: Pseudo code of SPAC

```
1  $\boldsymbol{\pi}$ = Find_Initial_Policy();
2  while $\max_i dV_i > \eta$ do
3      forall the $i$ in $\{1, \ldots, n\}$ do
4          $\tilde{\mathcal{M}}_i$ = Create_Augmented_CTMDP(i, $\pi_{-i}$);
5          $\pi_i^*$ = Solve_Augmented_CTMDP($\tilde{\mathcal{M}}_i$);
6          $dV_i = \tilde{V}_i(\pi_i^*) - \tilde{V}_i(\pi_i)$;
7      end
8      AvailableSet = $\{1, \ldots, n\}$;
9      while AvailableSet not empty do
10         $i^* = \arg\max_{i\in \text{AvailableSet}} dV_i$;
11         $\pi_{i^*} = \pi_{i^*}^*$, $dV_{i^*} = 0$;
12         AvailableSet.remove($i^* \cup$ Neighbors($i^*$));
13     end
14 end
```

4.1 Fast Creation of Augmented CT-MDP

Definition 6. *Given a MCT-MDP $\langle\{\mathcal{M}_i\}, \Omega, \mathcal{R}\}\rangle$ and a joint policy $\boldsymbol{\pi}$, $\tilde{\mathcal{M}}_i$, the augmented CT-MDP of agent i, is the same as $\mathcal{M}_i$ except for the reward function, $\tilde{R}_i$. $\tilde{R}_i(t_i|e_i)$ is the augmented reward function that sums up agent i's individual reward and all related joint rewards of an event e_i,*

$$\tilde{R}_i(t_i|e_i) = R_i(t_i|e_i) + \sum_{\mathbf{e_c}\in\Omega\wedge e_i\in\mathbf{e_c}} \int\int\ldots\int_{t_{c_j}=0,\forall c_j\neq i}^{t^*_{c_j}} \mathcal{R}(t_{c_1}, \ldots, t_i, \ldots, t_{c_{|\mathbf{c}|}}|\mathbf{e_c}) \left(\prod_{c_j\neq i} f_{\pi_{c_j}}(t_{c_j}|e_{c_j})dt_{c_j}\right). \quad (1)$$

The following Proposition shows that the optimal policy of the augmented CT-MDP for agent i given $\boldsymbol{\pi}_{-i}$ is also the best response to $\boldsymbol{\pi}_{-i}$, i.e. it maximizes the global value function.

Proposition 4.1. *Let π_i^* be the optimal policy of the augmented CT-MDP $\tilde{\mathcal{M}}_i$ with respect to $\boldsymbol{\pi}_{-i}$. Then for agent i's any policy π_i, we have, $GV(\pi_i^*, \boldsymbol{\pi}_{-i}) \geq GV(\pi_i, \boldsymbol{\pi}_{-i})$.*

Proof. Let $\tilde{V}_i(\pi_i)$ be the expected value of $\tilde{\mathcal{M}}_i$ with policy π_i. After expanding and rearranging terms, we have,

$$\tilde{V}_i(\pi_i) = \sum_{e_i \in E_i} \int_{t_i=0}^{t_i^*} \tilde{R}_i(t_i|e_i) f_{\pi_i}(t_i|e_i) dt_i$$
$$= \sum_{e_i \in E_i} \int_{t_i=0}^{t_i^*} R_i(t_i|e_i) f_{\pi_i}(t_i|e_i) dt_i + \sum_{e_i \in E_i} \sum_{\mathbf{e_c} \in \Omega \wedge e_i \in \mathbf{e_c}} \int\int$$
$$\dots \int_{t_{c_j}=0, \forall c_j \in \mathbf{c}}^{t^*_{c_j}} \mathcal{R}(t_{c_1}, \dots, t_{c_{|\mathbf{c}|}})|\mathbf{e_c}) \left(\prod_{c_i \in \mathbf{c}} f_{\pi_{c_i}}(t_{c_i}|e_{c_i}) dt_{c_i} \right)$$
$$= V_i(\pi_i) + \sum_{e_i \in E_i} \sum_{\mathbf{e_c} \in \Omega \wedge e_i \in \mathbf{c_c}} JV(\pi_i, \boldsymbol{\pi}_{\mathbf{c}_{-i}}|\mathbf{e_c}).$$

Since for any π_i, $\tilde{V}_i(\pi_i^*) \geq \tilde{V}_i(\pi_i)$, we have,

$$GV(\pi_i^*, \boldsymbol{\pi}_{-i}) - GV(\pi_i, \boldsymbol{\pi}_{-i}) = V_i(\pi_i^*) - V_i(\pi_i)$$
$$+ \sum_{e_i \in E_i} \sum_{\mathbf{e_c} \in \Omega \wedge e_i \in \mathbf{e_c}} [JV(\pi_i^*, \boldsymbol{\pi}_{\mathbf{c}_{-i}}|\mathbf{e_c}) - JV(\pi_i, \boldsymbol{\pi}_{\mathbf{c}_{-i}}|\mathbf{e_c})]$$
$$= \tilde{V}_i(\pi_i^*) - \tilde{V}_i(\pi_i) \geq 0.$$

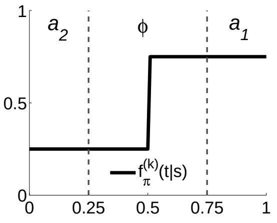
Proposition 4.1 shows augmented CT-MDPs are useful, however we still need $f_\pi(t|e)$ to obtain $\tilde{R}$ via Equation (1). To obtain $f_\pi(t|e)$ efficiently, we apply dynamic programming on decision steps:

Definition 7. *If an agent takes a non-wait action, we say it takes one decision step. An agent is initially at decision step* 0 *and is at decision step k after taking k non-wait actions.*

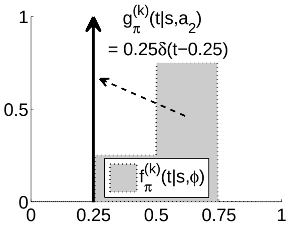
Considering the wait action as one decision step may result in an infinite number of decision steps since the agent can wait for an arbitrarily small amount of time. Hence in SPAC, we exclude the wait action from a decision step and treat it differently from any other actions (see below). Recall we assume all non-wait actions have a minimum duration. Thus, the maximum number of decision steps K can be bounded by the maximum number of non-wait actions that can be performed within the time limit t^*.

Then we can define $f_\pi^{(k)}(t|e)$ as the PDF of the remaining time when event e happens at decision step k. By definition we have, $f_\pi(t|e) = \sum_{k=0}^{K} f_\pi^{(k)}(t|e)$. In addition, we define $f_\pi^{(k)}(t|s)$ as the PDF of the remaining time when the agent enters s at decision step k. As the basis, we know at decision step 0, the agent is at s^0 with remaining time t^*, i.e. $f_\pi^{(0)}(t|s) = 0$ except $f_\pi^{(0)}(t|s^0) = \delta(t - t^*)$, where $\delta(t)$ is the Dirac delta function. From decision step 0, $f_\pi^{(k)}(t|e)$ and $f_\pi^{(k)}(t|s)$ must be obtained using PDF propagation. To address the significant computational difficulties in this propagation due to the continuous functions and the wait action, our key ideas are to (a) approximate the continuous functions f and σ as piecewise constant (PWC) functions and (b) use Dirac delta functions to represent infinite probability density values due to the wait action. We first demonstrate these ideas using an example, followed by our formal definition.

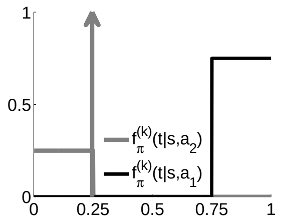
Suppose there are two actions a_1 and a_2 available at s, both taking the agent to s' with probability 1. $f_\pi^{(k)}(t|s)$ is given by Figure 2a, where the x-axis is the remaining time and y-axis is the probability density (In SPAC, $f_\pi^{(k)}(t|s)$ is always a PWC function as shown later). The policy for s is to take a_2 if the remaining time $t \in (0, 0.25]$, wait if $t \in (0.25, 0.75]$, and take a_1 if $t \in (0.75, 1]$. We will show the procedures to obtain $f_\pi^{(k)}(t|s, a_1)$, $f_\pi^{(k)}(t|s, a_2)$, and $f_\pi^{(k+1)}(t|s')$. Getting $f_\pi^{(k)}(t|s, a_1)$ is straightforward as shown by the black line in Figure 2c. Getting $f_\pi^{(k)}(t|s, a_2)$ is more difficult because of the wait action. If the agent enters s with $t \in (0.25, 0.75]$, it will wait until the remaining time drops to 0.25 and then perform a_2, implying that the non-wait action (a_2) at 0.25 will inherit the probability mass related to the wait action. This exact point of remaining time (0.25) will have an infinite value in terms of a probability density function. To address this, we use Dirac delta functions in addition to the PWC function to properly represent all the infinite values, e.g. represented by a Dirac component at 0.25 in the corresponding PDF as shown in Figure 2b. The PDF of performing a_2 is the combination of a PWC function and a Dirac delta function as shown by the grey line in Figure 2c. Knowing the PDF of the durations of both a_1 and a_2, we can calculate the PDF of entering s' at step $k+1$. The new PDF is a piecewise linear (PWL) function (dashed line in Figure 2d), and has to be approximated by a PWC function (solid line in Figure 2d) before the next iteration.



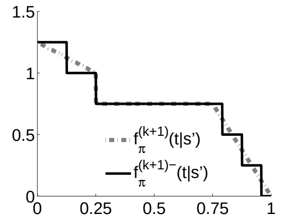
(a) Approximated PDF



(b) Compute $g_\pi^{(k)}(t|s, a)$



(c) PDF of taking actions



(d) New approximated PDF

Figure 2: Probability density function update.

Formally, knowing the f functions for decision step k, we can apply the following equations to compute the f functions for decision step $k+1$.

$$f_\pi^{(k)}(t|s, a) = f_\pi^{(k)}(t|s)\alpha_\pi(t|s, a) + g_\pi^{(k)}(t|s, a), \qquad (2)$$

$$f_\pi^{(k+1)}(t|s') = \sum_{s \in S} \sum_{a \in A} T(s, a, s') \int_{t'=t}^{t^*} f_\pi^{(k)}(t'|s, a) \cdot \sigma(t' - t|s, a, s') dt' \qquad (3)$$

Equation (2) captures the two possibilities that the agent will take action a at $\langle s, t\rangle$: $f_\pi^{(k)}(t|s)\alpha_\pi(t|s, a)$ is the PDF that the agent enters $\langle s, t\rangle$ and the policy of $\langle s, t\rangle$ is to take action a; $g_\pi^{(k)}(t|s, a)$ represents the PDF that the agent enters $\langle s, t'\rangle$ with remaining time $t' > t$ and the policy indicates the agent should wait until the remaining time drops to t. To formally define $g_\pi^{(k)}(t|s,a)$, we suppose given policy π and discrete state s, the agent should wait at L intervals $\{(t_{2l}, t_{2l+1}]\}$ where $t_{2l-1} < t_{2l} < t_{2l+1}$ and $l = 0, \dots, L-1$ (policy functions are always PWC when value functions are PWC, more details in the next subsection). Then $g_\pi^{(k)}(t|s,a)$ can be written as the following,

$$g_\pi^{(k)}(t|s,a) = \sum_{l=0}^{L-1} \beta_{a,l}\delta(t - t_{2l})$$

$$\beta_{a,l} = \begin{cases} \int_{t'=t_{2l}}^{t_{2l+1}} f_\pi^{(k)}(t'|s)dt', & \text{if } \alpha_\pi(t_{2l}|s,a) = 1, \\ 0, & \text{if } \alpha_\pi(t_{2l}|s,a) = 0. \end{cases}$$

Here $\beta_{a,l}$ is the probability (not probability density) that the agent enters s at interval $(t_{2l}, t_{2l+1}]$ and takes the non-wait action a at t_{2l}. In PDF, this probability corresponds to a Dirac delta function at t_{2l} with a magnitude equal to $\beta_{a,l}$.

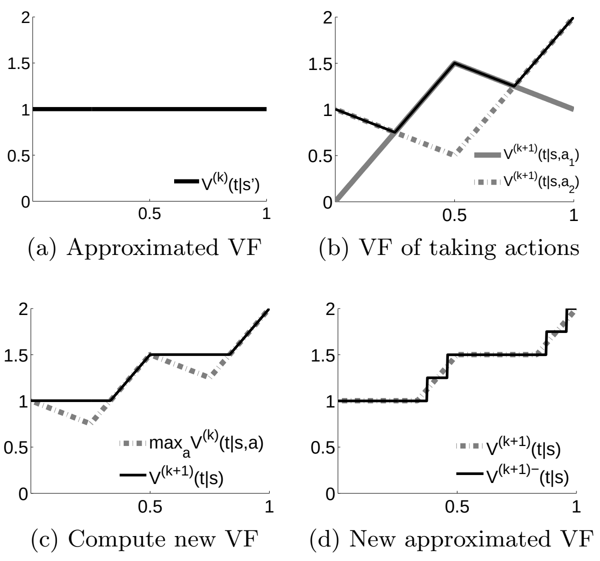
Since $f_\pi^{(k)}(t|s)$ is a PWC function, $f_\pi^{(k)}(t|s,a)$ obtained from Equation (2) is then the combination of a PWC function and a Dirac delta function. Equation (3) contains a convolution step between $f(t)$ and the duration functions $\sigma(t)$. Since duration functions are approximated by PWC functions, $f_\pi^{(k+1)}(t|s')$ computed by Equation (3) is a PWL function. At this point, we need to approximate $f(t)$ by a PWC function $f^-(t)$ so that it can be used in the next iteration. Similar to [9], the approximated PDF $f^-(t)$ is chosen such that the L^∞ distance $\|f(t) - f^-(t)\|_\infty$ can be controlled by a given error bound $\epsilon_f > 0$. In our experiments, we use fixed $\epsilon_f = 0.01$.

At this point, $f_\pi(t|e)$ can be obtained from summing up all $f_\pi^{(k)}(t|e)$ over $0 \le k \le K$. Recall $\mathcal{R}(t_{c_1}, \dots, t_{c_{|\mathbf{c}|}}|\mathbf{e_c})$ are piecewise hyper-rectangular constant functions, then the integral in Equation (1) returns a PWC function (see appendix). Since $R_i(t_i|e_i)$ are also PWC, then augmented reward functions $\tilde{R}_i$ are guaranteed to be PWC and can be directly used in solving the augmented CT-MDP as shown below.

4.2 Post-Creation: Solving Augmented CT-MDPs

To solve an augmented CT-MDP, we enhance Lazy Approximation [9] to explicitly address the wait action. In particular, since the agent with remaining time t can wait until any $t' \le t$ with no cost, the optimal value for $\langle s, t\rangle$ is the maximum over the values of taking all possible actions $a \in A$ at all possible continuous states $t' \le t$.

To first provide an intuitive explanation, we again demonstrate the overall process in Figure 3. Again suppose there are two actions a_1 and a_2 available at s, both taking the agent to s' with probability 1. The value function of s' at step k is as given by Figure 3a. Figure 3b shows the value functions of taking a_1 and a_2 at s. The maximum of the two functions at t is the optimal value of taking an immediate action with t remaining time. Since the agent can let remaining time drop to any level $t' < t$, the actual optimal value at t is the maximum of all values for all $t' < t$ as shown by the solid line in Figure 3c. The new value function is however piecewise linear which then needs to be approximated by a PWC function before the next iteration. The solid line in Figure 3d shows the new value function after approximation. Formally, we have the following modified



(a) Approximated VF (b) VF of taking actions (c) Compute new VF (d) New approximated VF

Figure 3: Enhanced Lazy Approximation.

Bellman update,

$$V^{(k+1)}(t|s) = \max_{0 \le t' \le t} \max_{a \in A} V^{(k+1)}(t|s,a), \tag{4}$$

$$V^{(k+1)}(t|s,a) = R(t|s,a) + \sum_{s'} T(s,a,s') \int_{t'=0}^{t} V^{(k)}(t'|s') \cdot \sigma(t-t'|s,a,s')dt', \tag{5}$$

where $V^{(k)}(t'|s')$ is the optimal value of s' with remaining time t' and k more decision steps. We approximate the optimal value functions $V(t)$ by PWC functions. In Equation (5), $\int_{t'=0}^{t} V(t')\sigma(t-t')dt'$ convolutes two PWC functions, V and σ, and returns a PWL function. Recall from the previous subsection, the augmented reward functions $\tilde{R}(t|s,a)$ are PWC functions. Therefore $V^{(k+1)}(t|s,a)$ obtained from Equation (5) is a PWL function. Then $\max_{a\in A} V^{(k+1)}(t|s,a)$ is also a PWL function, and therefore $V^{(k+1)}(t|s)$ obtained by Equation (4) is a PWL function. Similar to the last step in the previous subsection, we then approximate $V(t)$ by a PWC function $V^-(t)$ so that the L^∞ distance $\|V(t) - V^-(t)\|_\infty$ can be controlled by a given error bound $\epsilon_v > 0$. ϵ_v is a key parameter our experiments test for efficiency tradeoffs.

4.3 Finding Initial Policies

An appropriate initial policy is important in SPAC to reach a high-quality solution. One solution to obtaining such an initial policy is to just repeat the planning process many times with different random initial policies and select the best solution. The problem is that the algorithm will spend an equal amount of time optimizing every random starting policy, even though some of them might be dead-ends. Since the number of pieces in value functions is inversely proportional to ϵ_v, the runtime of SPAC is expected to be inversely proportional to ϵ_v. Although a smaller ϵ_v may provide a better final solution, our experiments show the difference is diminishing as ϵ_v approaches 0. Based on such observations, we can speed up SPAC by applying the following hybrid er-

Algorithm 2: Prune Initial Policy using Hybrid Error Bounds

```
1  while not Terminate do
2      $\boldsymbol{\pi}$ = Random_Policy();
3      $\boldsymbol{\pi}_h^*$ = Solve($\boldsymbol{\pi}$, $\epsilon_v^h$);
4      if $V(\boldsymbol{\pi}_h^*) + max\_gain > best\_solution$ then
5          $\boldsymbol{\pi}_l^*$ = Solve($\boldsymbol{\pi}_h^*$, $\epsilon_v^l$);
6          if $V(\boldsymbol{\pi}_l^*) > best\_solution$ then
7              best_solution = $V(\boldsymbol{\pi}_l^*)$ ;
8          end
9          if $V(\boldsymbol{\pi}_l^*) - V(\boldsymbol{\pi}_h^*) > max\_gain$ then
10             max_gain = $V(\boldsymbol{\pi}_l^*) - V(\boldsymbol{\pi}_h^*)$ ;
11         end
12     end
13 end
```

ror bound heuristic. The key idea is to quickly prune initial policies by solving it with a large error bound ϵ_v^h. Only when the returned policy has a relatively high quality, do we refine it with a lower error bound ϵ_v^l to get the final policy. The details are described in Algorithm 2.

5. EMPIRICAL VALIDATION

We create a set of multiagent task allocation problems motivated by the real world domain described in Section 2. Each agent is assigned a disjoint set of tasks, corresponding to a disjoint set of actions a_i. Each task i needs an uncertain amount of time to complete, whose distribution is denoted by σ_i. Completing task i before the deadline gains the team a reward of r_i. We assume all the agents start execution at time 0 and share the same deadline of 1.0. For simplicity, we assume all tasks can be started from the beginning and must be completed before the deadline. An agent's discrete state can be represented by the set of tasks it has completed. We consider three types of binary temporal constraints (we omit the discrete state terms in $\mathcal{R}$ below for better readability).

- **Precedence:** The team gets a positive reward r_{ij} if task j is started after task i is completed. Let $P_i(t) = \int_{t'=0}^{t} \sigma_i(t')dt'$ be the probability that task i can be completed in t amount of time. Then the constraint can be modeled by the corresponding joint component reward function,
$$\mathcal{R}(t_i, t_j | a_i, a_j) = r_{ij} P_j(t_j) P_i(t_i - t_j).$$
- **Simultaneity:** The team receives a positive reward $r_{ij} > 0$ if the starting times of task i and task j are close enough.
$$\mathcal{R}(t_i, t_j | a_i, a_j) = \begin{cases} r_{ij} P_i(t_i) P_j(t_j), & \text{if } |t_i - t_j| < \eta, \\ 0, & \text{otherwise.} \end{cases}$$
- **Exclusivity:** The team receives a negative reward r_{ij} if the execution intervals of task i and task j overlap.
$$\mathcal{R}(t_i, t_j | a_i, a_j) = \begin{cases} r_{ij}\left(1 - P_j(t_j - t_i)\right), & \text{if } t_i \le t_j, \\ r_{ij}\left(1 - P_i(t_i - t_j)\right), & \text{if } t_i > t_j. \end{cases}$$

We first compare SPAC with M-DPFP, the only existing multiagent planner for unordered tasks and continuous resources. We test 2-agent problems with task values chosen uniformly randomly between 0 and 10. The task duration is fixed to a normal distribution $\mathcal{N}(0.3, 0.01)$. A total of two precedence constraints are added randomly. We vary the number of tasks assigned to each agent. For each setting, we create one problem instance and compare the runtime and final solution quality of SPAC and M-DPFP. For M-DPFP we use approximation parameter $\kappa = 0.25$ and $\kappa = 0.2$ from [12]. For SPAC, we use $\epsilon_v = 1$ and $\epsilon_v = 0.1$ to allow tradeoffs between quality and runtime. Table 1 shows the *total* runtime of running SPAC 10 times with different random initial policies and the best solution among them. The first number of an entry is the runtime in seconds and the number in parentheses is the solution quality. NA indicates the algorithm fails to solve the problem within an hour. SPAC is seen to run substantially faster than M-DPFP and provides a comparable solution quality. For example, in the problem where each agent has 4 tasks, M-DPFP with $\kappa = 0.2$ finds a solution with quality of 27.7 in 247.6 seconds while SPAC with $\epsilon_v = 0.1$ finds a better solution with quality of 34.7 in 0.6 seconds (412-fold speedup).

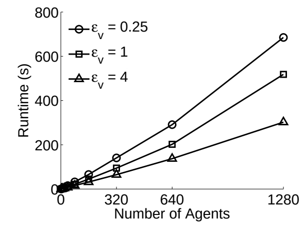
#Tks	SPAC $\epsilon_v = 1$	SPAC $\epsilon_v = 0.1$	M-DPFP $\kappa = 0.25$	M-DPFP $\kappa = 0.2$
3	0.1 (24.8)	0.1 (26.0)	0.4 (14.7)	2.7 (16.4)
4	0.4 (34.5)	0.6 (34.7)	21.7 (24.9)	247.6 (27.7)
5	0.4 (34.6)	0.7 (34.9)	63.2 (36.2)	NA
6	0.6 (36.9)	0.8 (38.2)	NA	NA

Table 1: Runtime in seconds & (quality): SPAC vs M-DPFP

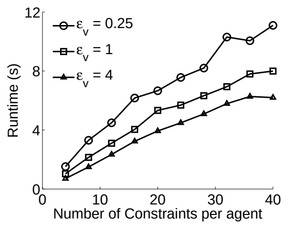
Next, we conduct experiments to show SPAC's scalability. We assign every agent in the team a disjoint set of tasks with an equal size. The task durations are chosen randomly from a set of pre-generated uniform distributions with mean in $[0, 0.5]$ and variance of 0.01. Task values are chosen uniformly randomly between 0 and 10. The temporal constraints are randomly assigned between tasks, with joint rewards chosen uniformly randomly between 0 and 20 (-20 for penalties).

First we fix the number of tasks per agent to 5, the number of constraints per agent to 8, and vary the number of agents from 5 to 1280 with different error bounds $\epsilon_v = 0.25, 1, 4$. For each setting we create 100 random problems and report the average runtime results in Figure 4a. SPAC scales almost linearly with respect to the number of agents for every ϵ_v setting. Second, we fix the number of agents to 10, the number of tasks per agent to 5, and vary the number of constraints per agent. Again Figure 4b shows the average results over 100 random problems. As we can see, the runtime is roughly linear to the number of constraints per agent. Since we consider only soft constraints among agents, adding constraints to a problem does not reduce the state space but requires more computation in creating augmented reward functions.

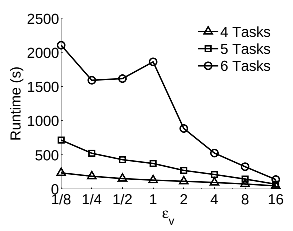
Next, we test on large scale problems where there are 1000 agents with 8 constraints per agent. We consider 4, 5, and 6 tasks per agent with varying ϵ_v from 0.125 to 16. Figure 4c and Figure 4d show the runtime and solution quality results of the average over 10 random problems respectively. The x-axes in both figures are in log-scale. These figures show that using smaller value of ϵ_v helps find better solutions at the cost of increasing runtime (however, the benefit of



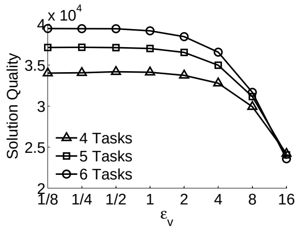
(a) Scale to large number of agents.



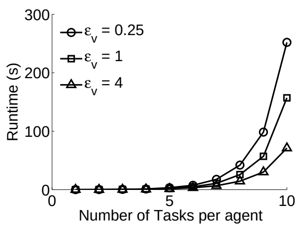
(b) Scale to large number of constraints.



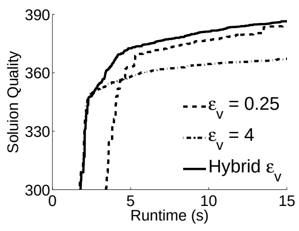
(c) Runtime of varying ϵ_v in 1000-agent problems.



(d) Solution quality of varying ϵ_v in 1000-agent problems.



(e) Scale to large number of tasks.



(f) Effectiveness of hybrid error bounds.

Figure 4: Results of SPAC.

decreasing $\epsilon_v \leq 0.5$ is negligible). SPAC with $\epsilon_v = 0.5$ is seen to solve a 1000 agent problem with 5 tasks per agent in 8 minutes – SPAC scales beyond capabilities of current algorithms.

Third, Figure 4e shows that unfortunately SPAC does not scale well with respect to the number of tasks per agent — this is a result from a test of 10-agent problems with 8 constraints per agent and varying number of tasks per agent from 1 to 10. Problems with unordered tasks cause the state space to grow exponentially in the number of tasks assigned to it, e.g. 1024 discrete states result if the agent has 10 unordered tasks compared to only 11 if the 10 tasks are fully ordered.

Finally, to study the tradeoff between solution quality and runtime of SPAC, we let the algorithm solve for as many initial policies as possible within a given time limit and we plot the solution quality over runtime, where the solution quality at a particular runtime point is the best solution the algorithm can find until that time. We compare three different approaches on random generated 10-agent problems with 5 tasks and 8 constraints per agent. The first approach uses $\epsilon_v = 0.25$ all the time, the second one uses $\epsilon_v = 4$ all the time, and the last one uses hybrid error bounds described in Section 4.3 with $\epsilon_v^h = 4$ and $\epsilon_v^l = 0.25$. As we can see in Figure 4f, all three approaches converge quickly because with more and more starting policies explored, finding a better solution becomes significantly harder. Comparing to $\epsilon_v = 0.25$, $\epsilon_v = 4$ tends to find a solution much quicker but converges to a lower solution quality. The hybrid error bound heuristic is clearly better than both, as it can find a low quality solution as quick as $\epsilon_v = 4$ and converges to a better solution than $\epsilon_v = 0.25$.

To test SPAC's solution quality – in the absence of an efficient global optimal solver – we run SPAC with $\epsilon_v = 0.25$ for 5, 10, and 20 seconds and compare with best solution found in running SPAC with $\epsilon_v = 0.1$ for 2 hours. As Table 2 shows, we can find good solutions with quality of at least 92% of the best by running SPAC for 5 seconds, and of at least 95% of the best by running SPAC for 20 seconds. Thus, SPAC can find reasonable solutions at a cheap computational cost.

Id	1	2	3	4	5
5s	92.9%	92.0%	96.2%	92.4%	96.8%
10s	93.7%	96.4%	96.2%	97.9%	98.4%
20s	95.6%	96.4%	96.2%	97.9%	99.2%

Table 2: SPAC: comparing to best solution found in 2 hours.

6. CONCLUSION AND RELATED WORK

To address the complexity of the general class of continuous state DEC-MDPs, this paper presents two contributions. First, it introduces a novel model (MCT-MDP) for multi-agent planning problems that exploits transition independence in domains with graphical agent dependencies and continuous temporal constraints. More importantly, it presents a new, iterative, locally optimal algorithm called SPAC that is based on the following key ideas: (1) defining an augmented CT-MDP such that solving this single-agent continuous state MDP provably provides a best response to neighboring agents' policies; (2) fast convolution to efficiently generate such augmented MDPs; (3) a new enhanced Lazy Approximation algorithm to solve these augmented MDPs; (4) exploiting graph structure of reward dependencies for scalability. Our experiments show SPAC not only finds solutions substantially faster than M-DPFP with comparable quality, but also scales well to large teams of agents – it can solve a 1000-agent problem with 5 unordered tasks per agent in 8 minutes.

As for related work, there have been many algorithms proposed for solving discrete state DEC-MDPs and DEC-POMDPs such as [1, 8]. However, these techniques usually cannot be easily applied to continuous state problems. On the other hand, there are algorithms for solving hybrid state MDPs such as [9, 10], which however, only solve for single-agent problems. Algorithms such as [3, 5, 11, 12] have been successful in solving multi-agent planning problems with continuous states. Unfortunately, [5, 11] consider only a restricted problem where a fixed ordering of agent actions is given. The exponential complexity of M-DPFP [12] limits its applicability to only small problems. The goal-oriented joint reward structure in [3] cannot represent the temporal constraints in our problems. Furthermore, none have been shown to scale to large numbers of agents. The MCT-MDP model introduced in this paper is a continuous state generalization of the TI-DEC-MDP model introduced by Becker et al. [1]. Unfortunately the coverage set algorithm (CSA) introduced in [1] cannot directly solve the continuous state model given that there are an infinite number of policies for an individual agent. We notice there exist efficient discrete state graphical models of agent interactions including IDID [7] and IDMG [15]. Integrating such graphical representations to our model can be an interesting future research topic.

7. REFERENCES

[1] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-independent decentralized Markov Decision Processes. In *AAMAS*, 2003.

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition independent decentralized Markov Decision Processes. *J. Artif. Int. Res.*, 22:423–455, 2004.

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

[4] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Math. Oper. Res.*, 27:819–840, 2002.

[5] A. Beynier and A. Mouaddib. A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *AAMAS*, 2005.

[6] T. B. Curtin and J. G. Bellingham. Guest editorial - autonomous ocean-sampling networks. *Oceanic Engineering, IEEE Journal of*, 27, 2001.

[7] P. Doshi, Y. Zeng, and Q. Chen. Graphical models for interactive POMDPs: representations and solutions. *JAAMAS*, 2009.

[8] A. Kumar and S. Zilberstein. Point-based backup for decentralized POMDPs: Complexity and new algorithms. In *AAMAS*, 2010.

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

[10] J. Marecki, S. Koenig, and M. Tambe. A fast analytical algorithm for solving Markov decision processes with continuous resources. In *IJCAI*, 2007.

[11] J. Marecki and M. Tambe. On opportunistic techniques for solving decentralized Markov decision processes with temporal constraints. In *AAMAS*, 2007.

[12] J. Marecki and M. Tambe. Planning with continuous resources for agent teams. In *AAMAS*, 2009.

[13] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, 2005.

[14] F. Py, K. Rajan, and C. McGann. A systematic agent framework for situated autonomous systems. In *AAMAS*, Toronto, Canada, May 2010.

[15] M. T. J. Spaan and F. S. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *AAMAS*, 2008.

[16] F. Wu, S. Zilberstein, and X. Chen. Point-based policy generation for decentralized POMDPs. In *AAMAS*, 2010.

APPENDIX

A. PIECEWISE HYPER-RECTANGULAR CONSTANT FUNCTIONS

DEFINITION 8. *We define a k-dimensional hyper-rectangle $H = \{(x_1, \ldots, x_k) \mid \alpha_i \le x_i \le \beta_i, \forall i = 1, \ldots, k\}$. A k-dimensional function $f(x_1, \ldots, x_k)$ defined on hyper-rectangle H_f, is called piecewise hyper-rectangular constant if and only if there are finite number of hyper-rectangles $H_1, \ldots, H_n$ such that $H_i \cap H_j = \emptyset$ for any $i \neq j$, $\bigcup_{i=1}^n H_i = H_f$, and $f(x_1, \ldots, x_k) = c_i$ is constant for any i and $(x_1, \ldots, x_k) \in H_i$.*

PROPOSITION A.1. *Suppose $f(x_1, \ldots, x_k)$ defined on hyper-rectangle $\{(x_1, \ldots, x_k) \mid \alpha_i \le x_i \le \beta_i, \forall i = 1, \ldots, k\}$ is piecewise hyper-rectangular constant, then for any $i = 1, \ldots, k$, and for any j and any function $g_j(x)$ defined on interval $[\alpha_j, \beta_j)$, the following integral is a piecewise constant function,*

$$f_i(x) = \int\int \ldots \int_{x_j=\alpha_j, \forall j \neq i}^{\beta_j} f(x_1, \ldots, x, \ldots, x_k) \prod_{j \neq i} g_j(x_j) dx_j.$$

PROOF. Because f is piecewise hyper-rectangular constant, for any dimension $i = 1, \ldots, k$, there must exist a finite number of intervals $[\alpha_i = a_1, a_2), \ldots, [a_{m-1}, a_m = \beta_i)$, such that for any point $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)$ where $x_j \in [\alpha_j, \beta_j]$, the function of fixing all dimensions except for i, $f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k)$ is constant within each interval. For example, the number of those intervals should never be greater than the total number of hyper-rectangles that defines the piecewise hyper-rectangular function. Then consider any two different points in the same interval, $x, y \in [a_l, a_{l+1}]$ and $x \neq y$, we have,

$$\begin{aligned} & f_i(x) - f_i(y) \\ =& \int\int \ldots \int_{x_j=\alpha_j, \forall j \neq i}^{\beta_j} [f(x_1, \ldots, x, \ldots, x_k) \\ & - f(x_1, \ldots, y, \ldots, x_k)] \left(\prod_{j \neq i} g_j(x_j) dx_j \right) \\ =& 0. \end{aligned}$$

This implies $f_i(x)$ is constant within each intervals, and therefore is piecewise constant. □