Cooperative dynamic domain reduction*

Aaron Ma, Michael Ouimet, and Jorge Cortés

[1] Department of Mechanical and Aerospace Engineering, University of California, San Diego, {aam021,cortes}@ucsd.edu
[2] SPAWAR Systems Center Pacific, San Diego, ouimet@spawar.navy.mil

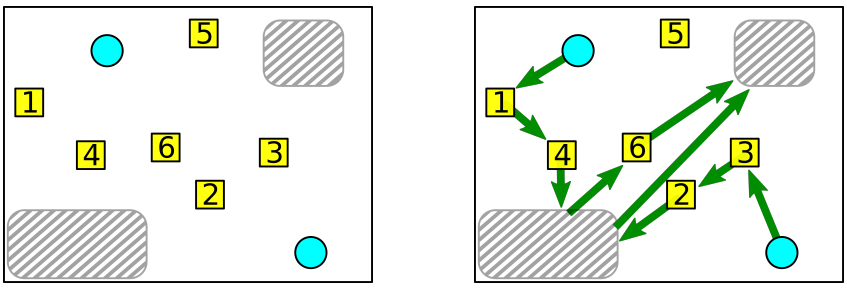
Abstract. Unmanned vehicles (UxVs) are increasingly deployed in a wide range of challenging scenarios, including disaster response, surveillance, and search and rescue. This paper is motivated by scenarios where a heterogeneous swarm of UxVs is tasked with completing a variety of different objectives that possibly require cooperation from vehicles of varying capabilities. Our goal is to develop an approach that enables vehicles to aid each other in the services of these objectives in a distributed and autonomous fashion. To address this problem, we build on *Dynamic domain reduction for multi-agent planning* (DDRP), which is a framework that utilizes model-based hierarchical reinforcement learning and spatial state abstractions crafted for robotic planning. Our strategy to tackle the exponential complexity of reasoning over the joint action space of the multi-agent system is to have agents reason over single-agent trajectories, evaluate the result as a function of the cooperative objectives that can be completed, and use simulated annealing to refine the search for the best set of joint trajectories. The resulting algorithm is termed *Cooperative dynamic domain reduction for multi-agent planning* (CDDRP). Our analysis characterizes the long-term convergence in probability to the optimal set of trajectories. We provide simulations to estimate the performance of CDDRP in the context of swarm deployment.

1 Introduction

UxVs are an outlet for the implementation of state-of-the-art algorithms that pertain to many fields of dynamic systems and machine learning. Recently, particular interest in the autonomous capability of these vehicles is growing. Characterizing multiple UxVs that interact with each other is difficult because of the joint number of possibilities that exist due to the joint state and action spaces. To approach this challenge, we propose DDRP, a hierarchical algorithm that takes slices of the environment and models them as a semi-Markov decision process. DDRP lacks a structure for agents requesting and assisting executing objectives that require more than one agent to complete. The motivation of this paper is to extend the DDRP framework to allow agents to share their requests and to assist others if they deem it beneficial to the entire swarm.

Figure 1 provides an example application scenario of interest. In this example, agents are tasked with building structures. Agents are able to gather resources

* This work was supported by ONR Award N00014-16-1-2836.



(a) Initial conditions (b) Final trajectories

Fig. 1. Example use of CDDRP. Agents are tasked with gathering resources and building structures at build sites. The symbols, ●, 1, ▨, represent agents, resources, and build sites respectively. In this specific case, the build site on the bottom left requires resources 1, 2, and 4, while the build site on the top right requires resources 3 and 6 to begin construction. Figure 1(a) shows the initial conditions and Figure 1(b) illustrates the trajectories resulting from the algorithm proposed in this paper.

that are randomly scattered in the environment, but they are only able to carry two resources at any given time. The agents bring resources to building sites which require certain combination of resources for construction, a process that is expedited when agents cooperate.

Literature review: DDRP [1] is a framework that uses concepts from multi-agent planning and model-based reinforcement learning that can be used by agents as a scalable method to service a variety of objectives in massive environments. Distributed deployment methods satisfy a global task when the agents are limited to local information. These algorithms are an attractive approach for deployment of UxVs because solutions are often robust to failure of individual agents and computational load can be distributed amongst the agents, creating a more scalable option than centralized methods [2–7]. DDRP is inspired by many techniques built for finding a policy in a *Markov decision process* (MDP) [8–11]. If a model is known, it is common practice to search the state and action space of a MDP using a Monte-Carlo tree search (MCTS) [12]. In MCTS, the agent assumes a state and makes an action, observing the reward gathered from the change in state as a result of the action. Taking an action that maximizes the upper confidence bound (UCB) of the expected discounted reward [13] is shown to efficiently search state and action space. One variation of MDPs that we use in the formulation of DDRP, is the *semi-Markov decision process* (SMDP) [14], where agents take multi-time step macro-actions or options instead of actions [8]. DDRP utilizes a hierarchical structure, inspired from hierarchical reinforcement learning algorithms that are utilized when the state space and action space of the environment are large [15, 16]. MDPs have been structured to account for joint actions amongst multiple agents as multi-agent Markov decision processes (MMDP) [17]. We extend DDRP to handle MMDP scenarios with cooperative

objectives using *simulated annealing* (SA) [18–22]. Simulated annealing is an approach for finding a state with optimal value in a Markov chain that borrows the idea of annealing from nature and is capable of handling high-dimensional, nonconvex problems.

Statement of contributions: We strive to enable a swarm of UxVs to cooperate and complete a large variety of objectives in massive environments. In this paper, we use `DDRP` as a baseline framework to handle these conditions and make modifications to allow for cooperation. We also propose an extension to the framework to enable the agents to individually plan their trajectories in a way that converges in probability to a globally optimal joint set of trajectories. Given the overall state of the environment, agents create a set of trajectories in our modified `DDRP`. The agents take the set of trajectories they created in `DDRP` and individually change their planned trajectory with respect to one of two schemes that we explore called *flat* and *weighted*. In both methods, agents use a simulated annealing approach to find their own trajectories. In the flat scheme, agents attempt random trajectories that they found in `DDRP`. Agents can use the weighted scheme to attempt trajectories with respect to the decisions that other agents have made as a means of proposing a joint effort in the completion of cooperative tasks. We call the modifications and extensions to the framework, `CDDRP`. Lastly, we show that the agents are able to plan asynchronously and converge in probability to the set of optimal joint trajectories that yield the greatest discounted reward in a receding time horizon and perform simulations to get empirical results.

2 Preliminaries

We introduce here essential concepts and tools for the rest of the paper, beginning with some notation. We use $\mathbb{Z}$ and $\mathbb{R}$ to denote integers and real numbers, respectively. An objective-oriented approach with the use of tuples is present throughout the paper: for an arbitrary tuple $a = \langle b, c \rangle$, the notation $a.b$ means that b belongs to tuple a. Last, $|\mathcal{Y}|$ indicates the cardinality of a set $\mathcal{Y}$.

2.1 Markov chains and (semi)-Markov decision processes

A Markov chain describes a sequence of states such that the probability of transitioning from one state to another only depends on the current state. We define a Markov chain as a tuple $\langle \mathcal{S}, Pr^s \rangle$ of states $s \in \mathcal{S}$ and probability $Pr^s(s'|s)$ that s transitions to s'. If we assign states of the Markov chain to nodes, and positive state transition probabilities $Pr^s(s'|s) > 0$ to edges, we can create a graph of the Markov chain. A value can be associated to each state s in a Markov chain, $V(s) \in \mathbb{R}$, allowing us to order nodes in the graph vertically with respect to their value, creating an ordered graph as shown in Figure 3. Define $H \in \mathbb{R}$ such that $H \leq V(s)$ for any s. The following properties of a Markov chain are important and can be analyzed by visualization of an ordered graph:

(i) *Strong irreducibility:* there exists a path from s to s' for all s'.
(ii) *Weak reversibility:* if s can be reached by s' at height H, then there exists a path from s' to s at height H as well.

A Markov decision process (MDP) is an extension of Markov chains such that a decision is made at each state influences probability of state transition. A MDP is a tuple $\langle S, A, Pr^s, R\rangle$ that contains a state and action spaces S and A, respectively, the probability of transitioning from one state s to another s' $Pr^s(s'|s,a)$ after taking action $a \in A$, and a reward $R(s'|s,a) \in \mathbb{R}$ that is received for transitioning to s' after taking action a in state s. Amongst several extensions of MDPs are multi-agent Markov decision processes (MMDP) and semi-Markov decision processes (SMDP). A MMDP is a tuple $\langle S, \mathcal{A}, \{A_\alpha\}_{\alpha\in\mathcal{A}}, Pr^s, R\rangle$, similar to a MDP with the addition of agents $\alpha \in \mathcal{A}$, where each agent has an action space $\{A_\alpha\}$. With this modification, the state transition function Pr^s maps the probability of state transition with respect to the joint actions of agents in $\mathcal{A}$. One strategy for multi-agent cooperation is to reason over the joint action space as in MMDPs. This solution does not scale with the number of agents, as the joint action space increases and the problem dimension grows exponentially. Instead, we reason over single-agent SMDPs and utilize the resulting trajectories for cooperation. SMDPs allow for actions that may take multiple time steps to execute. A SMDP is a tuple $\langle S, A, Pr^s, R, Pr^t\rangle$, where the additional term $Pr^t(t|s,a)$ is the probability of action a in state s taking t time to complete. `DDRP` models parts of the environment as a SMDP and uses the upper confidence bound to efficiently search for the optimal policy.

2.2 Simulated annealing

Let the tuple $\langle \mathcal{S}, Pr, V\rangle$ define a Markov chain with value associated to its state. Simulated annealing seeks to determine $\text{argmax}_s V(s)$, which is generally a combinatorial problem. Algorithm 1 outlines the process of simulated annealing.

Algorithm 1: Simulated annealing

```
Initialize ρ, T
for k = 1 to k = N
    Generate new state ρ'
    if V(ρ') > V(ρ) or Random(0,1)
      < e^((V(ρ')−V(ρ))/T):
        ρ = ρ'
    Decrement T
```

As the temperature of the system T is incrementally decreased, the probability that a new solution with lower value is accepted decreases. Simulated

annealing yields strong results [18,19,21,22] and converges to the global optimal if the temperature is decreased sufficiently slowly. The cooling rate

$$T_k = \frac{c}{\log(k)}, \tag{1}$$

is shown [21] to be a necessary and sufficient condition for the algorithm to converge in probability to a set of states that globally optimize V when the underlying Markov chain has both strong irreducible and weak reversible properties.

3 Problem statement

Consider a set of agents $\mathcal{A}$ indexed by α. The agents seek to service a number of different objectives, whose *objective type* is indexed by b. A *sub-objective* $q = \langle w, b \rangle$ contains a waypoint $w \in \mathbb{R}^d$ and an objective type. An *objective* $o = \langle \mathcal{Q}, r \rangle$ consists of a set of sub-objectives $\mathcal{Q}$ and a reward $r \in \mathbb{R}$. We let $\mathcal{O}$ denote the set of all objectives. One possibility is to have objectives that require only one agent to be satisfied, as in [1]. Instead, here we consider objectives with a sub-objective set of cardinality $|\mathcal{Q}| > 0$. In this case, agents need to simultaneously be at specified waypoints and take actions in $q \in o.\mathcal{Q}$ in order to complete objective o. Agents use `DDRP` to generate a set of potential trajectories, termed $\mathcal{V}$, that they may take to service objectives in the environment. We strive to extend the capabilities of `DDRP` to include handling objectives that need two or more agents to complete. Given N agents, we determine a structure that allows the agents to share their trajectories and to distributively determine the joint trajectory that globally maximizes the sum of future discounted rewards.

4 Cooperative dynamic domain reduction planning

In this section we provide an overview `DDRP` and extend the framework to deal with objectives that require more than one agent. We organize this section as follows. First we review basic definitions from `DDRP` introduced in our previous work [1]. As we discuss these definitions, we provide modifications to enable the agents to communicate desire for cooperation from other team members. Next, we present a high-level overview of algorithms used in `DDRP`. This sets the basis to introduce the multi-agent system where agents communicate and search for the joint optimal actions for deployment on large scale environments with cooperative objectives. We call the resulting framework *Cooperative dynamic domain reduction planning* (`CDDRP`).

4.1 Abstractions and definitions

We begin with some core definitions in `DDRP`. First we introduce abstracted regions and actions, the construction of sub-environments, and task trajectories.

Then we give a high level overview of main algorithms in `DDRP`, `SubEnvSearch` and `TaskSearch`, and then finish with a new trajectory selection algorithm on the multi-agent level with some analysis.

Abstracted regions: A *region* is a convex set $x \subseteq \mathbb{R}^d$ such that the union of all regions are disjoint. The state of a region x is an abstraction of the objectives that reside in x. Let $\Phi_x : w \to x$ define the abstraction function that returns the region that q belongs to. We use this function to map where sub-objectives exist, i.e., $\Phi_x(q.w) = x$. Given a region x, let $\mathcal{Q}_x^b = \{q : \Phi_x(q.w) = x, q.b = b\}$ be the set of sub-objectives of objective type b that exist in it. We define the function $\Phi_o : \mathcal{Q}_x^b \to s_x^b$ to describe the abstracted state of the corresponding type of objective in the region. Define the regional state to be $s_x = (s_x^1, s_x^2, \ldots)$.

Abstracted tasks: In [1], a task is a tuple $\tau = \langle s_{x_i}^{b'}, s_{x_i}^b, x_i, x_j, b\rangle$, where x_i is the region that the agent is executing sub-objective of objective type b in, x_j is the next region that the agent plans to travel, $s_{x_i}^b$ is the prior state of x_i, and $s_{x_i}^{b'}$ is the post state of x_i. Here, we augment the notion of task to include the concept of time abstraction. Mapping the time to an interval allows the agents to communicate approximate times to complete coordinated tasks by. If the length of the time intervals is too small, then the number of possible joint actions increases and the problem may become intractable. On the other hand, if the length of the time intervals is too big then the execution time of coordinated tasks become less precise. Let the convex set $\varsigma \subseteq \mathbb{R}$ specify a *time interval*. We specify a *sub-task*, $\mu = \langle x_i, b, \varsigma\rangle$, to be a tuple that contains a region x_i that the agent acts in, a objective type b, and a time interval ς. The modified definition of a task is now $\tau = \langle \mu, s_{x_i}^b, s_{x_i}^{b'}, x_j\rangle$. This modification allows agents to communicate the bare minimum information that is necessary for others to know when they are attempting a coordinated objective. We denote by $\mathcal{T}$ the set of all tasks.

Sub-environments: The DDRP framework takes the environment and generates *sub-environments* ϵ composed of a sequence of abstracted regions and a state encoding proximity and regional states of those regions. We extend the definition of sub-environment to include requirements that the agent agrees to satisfy. We do this by incorporating the requirements into the state of the sub-environment. Let $\overrightarrow{x}$ be a finite sequence of regions in the environment (e.g., $[x_2, x_1, x_3]$). We determine a state of the sub-environment given $\overrightarrow{x}$. To do this, we need to determine if regions are repeated in $\overrightarrow{x}$. Let $\xi(k, \overrightarrow{x})$, return the first index h of $\overrightarrow{x}$ such that $\overrightarrow{x}_h = \overrightarrow{x}_k$. Another necessary component is the time that it takes for the agent to travel between regions. We use $d(x_i x_j) : x_i, x_j \to \mathbb{Z}$ to designate an abstracted amount of time it takes for an agent to move from x_i to x_j, or ∞ if no path exists. We create sub-environments with the constraint that agents may need to satisfy some cooperative tasks. Let the set $\mathcal{U} = \{\mu_1, \mu_2, \ldots\}$ be a set of subtasks that the agent agreed to partake in the sub-environment. We define the *sub-environment state* with the addition of requirements as

$$s = [s_{\overrightarrow{x}_1}, s_{\overrightarrow{x}_2}, \ldots] \times [\xi(1, \overrightarrow{x}), \xi(2, \overrightarrow{x}), \ldots] \times [d(\overrightarrow{x}_1, \overrightarrow{x}_2), d(\overrightarrow{x}_2, \overrightarrow{x}_3), \ldots] \times \{\mu_1, \mu_2, \ldots\}. \quad (2)$$

We denote by S^ϵ the set of all possible sub-environment states. A sub-environment is $\epsilon = \langle \overrightarrow{x}, s, \mathcal{U} \rangle$.

When an agent performs a task in the sub-environment, it expects the action to take multiple time steps to complete, the environment to change states, and to receive some reward. Let Pr^s and Pr^t be the probability distributions for state transition and time of completion for executing a task in a sub-environment, respectively. Also, let r^ϵ designate the expected reward that an agent receives for choosing a task given the state of a sub-environment. Finally, we can represent the process of executing tasks in sub-environments as the *sub-environment SMDP*, $\mathbb{M} = \langle S^\epsilon, \mathcal{T}, Pr^s, r^\epsilon, Pr^t \rangle$. The goal of the agent is to determine a policy $\pi_\epsilon : \epsilon.s \rightarrow \tau$ that yields the greatest rewards in $\mathbb{M}$. The state value under policy π_ϵ is given by

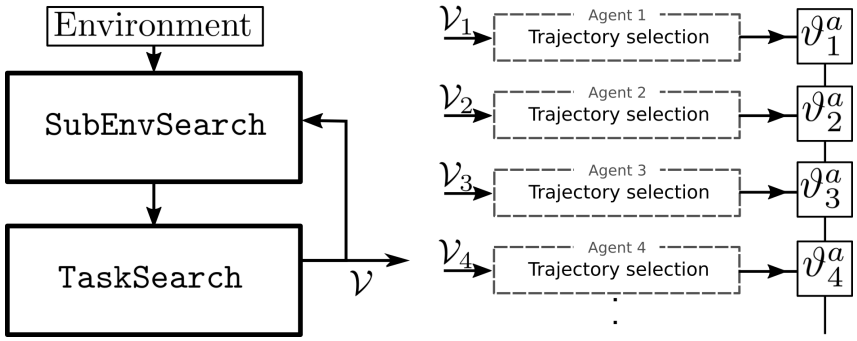
$$V^{\pi_\epsilon}(\epsilon.s) = r^\epsilon + \sum_{t \in \mathbb{R}} Pr^t \gamma^t \sum_{\epsilon.s \in S^\epsilon} Pr^s V^{\pi_\epsilon}(\epsilon.s') \text{ s.t. } 0 \leq \gamma < 1. \tag{3}$$

Task trajectories: Agents cooperate with others by sharing their current and prospective plans for the receding time horizon. In `DDRP`, we call this information a *task trajectory*. This is defined as $\vartheta = \langle [\tau_1, \tau_2, \ldots], [Pr^1, Pr^2, \ldots] \rangle$, where tasks in $[\tau_1, \tau_2, \ldots]$ are executed in order and the completion time of τ_1 is according to the probability density function $Pr^1(t)$, etc. Task trajectories are created with respect to some sub-environment ϵ and are constrained so that a task in $\vartheta.\tau_p.x_j$ must be $\vartheta.\tau_{p+1}.x_i$, making it so that the region that agents travel to next is always the active region of the next task to complete. Here, we redefine the concept of task trajectory to carry information about what cooperative tasks the agent has. Given the cooperative tasks in a trajectory, the agent may have some cooperative tasks that it plans on executing with others, and some cooperative tasks that it might not have found partners for execution. The agent puts all subtasks in the set $\mathcal{U}$. A task trajectory is then defined as $\vartheta = \langle [\tau_1, \tau_2, \ldots], [Pr^1, Pr^2, \ldots], \mathcal{U} \rangle$.

4.2 `DDRP` algorithms and task generation for communication

`DDRP` is a hierarchical framework which has an algorithm called `SubEnvSearch` that generates subsets of the entire environment. The sub-environment that is created is modeled as an SMDP, and `TaskSearch` is used to optimally find a policy for it. In this section, we give a brief description of the algorithmic components `SubEnvSearch` and `TaskSearch`, and introduce some necessary modifications for `CDDRP`. `DDRP` organizes `SubEnvSearch` and `TaskSearch` into a hierarchical structure as shown in Figure 2(a).

Sub-environment search: `SubEnvSearch` uses the environment as an input and creates a sub-environment. Using the set of all regions in the environment, we add one region at a time to a 'sub-environment set' until there are N_ϵ regions. As the sub-environment set is being generated, its value (determined by `TaskSearch`) is evaluated. The agent uses this value to add regions that locally maximize the value of the sub-environment.



(a) Algorithms of `DDRP` (b) `CooperativeTrajectorySearch`

Fig. 2. Workflow of `DDRP` and `CooperativeTrajectorySearch`. `DDRP` in Figure 2(a) is a hierarchical algorithm that dynamically creates sub-environments with sub-environment search. Sub-environments are modeled as a semi-Markov decision process where the agent uses `TaskSearch` to find tasks which yield the greatest expected discounted reward. `CooperativeTrajectorySearch` is a process that is run in parallel with `DDRP`. With respect to the simulated annealing process, agents asynchronously choose active trajectories ϑ_α^a from a set of trajectories $\mathcal{V}_\alpha$ found in `DDRP` as shown in Figure 2(b). The result is a list of active trajectories, $\rho = [\vartheta_1^a, \vartheta_2^a, \ldots]$.

We extend the sub-environment search process to now include cooperative tasks. To do this, first we run `SubEnvSearch` to get a sub-environment from the environment. If a cooperative sub-objective q exist in the sub-environment, then with some probability we include q to the sub-environment sub-task set $\epsilon.\mathcal{U}$ as a requirement for the agent to complete. These cooperative sub-objectives must be compatible with each other. For example, an agent is not able to create a sub-environment with two cooperative sub-objectives at the same time interval. After this, the agent updates the sub-environment state accordingly and sends the created SMDP to `TaskSearch`.

Task search: Given a sub-environment ϵ generated from `SubEnvSearch`, the purpose of `TaskSearch` is to determine the value of taking a task τ. This is done using upper confidence bound tree search (UCT) [13], which is a Monte-Carlo tree search (MCTS) that uses the upper confidence bound of action values to explore the SMDP. The agent takes the sub-environment SMDP and runs `TaskSearch` for many iterations, where the values of taking a task given the state of the sub-environment converges to a real number. When choosing which task to take, the agent chooses the task that maximizes the action value given the state of the sub-environment.

We extend the task search process under the constraint that it must satisfy the cooperative subtasks in the sub-environment. To do this, given the sub-environment from `SubEnvSearch`, the agent must obey the constraints of cooperative subtasks in the sub-environment. At each task selection, the agent chooses from a set of tasks such that it is still feasible to visit regions in $\epsilon.\mathcal{U}$

at the required times. The agent determines the action value for choosing tasks given the sub-environment state and satisfying the constraints given to it and returns a task trajectory. This task trajectory also contains all the cooperative tasks that it was constrained to satisfy as unpaired cooperative tasks.

4.3 Joint DDRP via simulated annealing

In this section we introduce the main contribution of this paper, aside from the modifications to DDRP described above. We specify a value for the joint set of trajectories by determining which objectives can be completed given the planned sub-tasks of all the agents. Then we introduce the algorithm in which agents use simulated annealing to search for the best set of joint trajectories.

Agents execute DDRP yielding a set of trajectories $\mathcal{V}$. The trajectory that an agent currently plans to execute, called the *active trajectory*, is denoted by ϑ^a. Agents share their active trajectories so that they know which cooperative objectives they can collaborate on. The active trajectories are ordered in a list with respect to some arbitrary agent ordering to form the *active trajectory list* $\rho = [\vartheta^a_1, \vartheta^a_2, \ldots]$. In doing so, agents have access to the subtasks of others. The set of all current subtasks is the *collective subtask set*, denoted $\mathcal{U}^{\mathcal{A}} = \{\mu : \mu \in \vartheta.\mathcal{U},\ \forall\ \vartheta \in \rho\}$. We also create a set of subtasks that are planned to be executed in a given time interval, $\mathcal{U}^{\mathcal{A}}_{\varsigma} = \{\mu : \mu \in \mathcal{U}^{\mathcal{A}}, \mu.\varsigma = \varsigma\}$. For the remainder of the paper, we assume that agents have access to $\mathcal{U}^{\mathcal{A}}$ and $\mathcal{U}^{\mathcal{A}}_{\varsigma}$ for all time intervals ς. We say that o is satisfied if all sub-objectives in o exist in a subtask set at ς such that $o.\mathcal{U} \subseteq \mathcal{U}^{\mathcal{A}}_{\varsigma}$. With knowledge about $\mathcal{U}^{\mathcal{A}}_{\varsigma}$ for all time intervals ς, agents are able to determine the minimal time interval in which each objective is expected to be completed. Let the *expected time of completion* for an objective be defined as

$$\varsigma_o(\mathcal{U}^{\mathcal{A}}, \rho) = \underset{\varsigma}{\operatorname{argmin}} \int_{\varsigma} \gamma^t dt \quad \text{s.t. } o.\mathcal{U} \subseteq \mathcal{U}^{\mathcal{A}}_{\varsigma}. \tag{4}$$

or $\varsigma_o = \infty$ if there are no time intervals for which $o.\mathcal{U} \subseteq \mathcal{U}^{\mathcal{A}}_{\varsigma}$. Next, we introduce the *joint task trajectory value* as

$$V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho) = \sum_{o \in \mathcal{O}} \int_{\varsigma_o(\mathcal{U}^{\mathcal{A}}, \rho)} \gamma^t\, o.r\, dt, \tag{5}$$

which corresponds to the cumulative discounted reward of all the agents in the swarm given ρ.

We are now ready to introduce CooperativeTrajectorySearch (cf. Algorithm 2). This strategy takes as input a set of agents, their respective task trajectory sets, and the set of all objectives. The agents initially choose active trajectories that form the active trajectory list and the temperature is initialized to ∞. We use the notation $\rho|\vartheta_\alpha$ to indicate the resulting active trajectory list

that occurs when agent α chooses a new active trajectory $\vartheta_\alpha = \vartheta_\alpha^a$. This operation simply changes the α-index element of ρ to be ϑ_α. An agent at random then chooses to select a trajectory from its trajectory set with probability distribution Pr^ϑ. This distribution follows one of two schemes. The first is called a *flat* scheme, where agent α chooses a trajectory ϑ with respect to the number of trajectories in the set as follows:

$$Pr^\vartheta(\vartheta)^\dagger = \frac{1}{|\mathcal{V}|}. \tag{6}$$

The next method of selecting a trajectory that we explore is called a *weighted* scheme. In this method, agent α chooses a trajectory with probability with respect to the local joint task trajectory values as follows:

$$Pr^\vartheta(\vartheta_\alpha)^\ddagger = \frac{V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho|\vartheta_\alpha)}{\sum\limits_{\vartheta_\alpha^i \in \mathcal{V}_\alpha} V^{\mathcal{A}}(, \mathcal{U}^{\mathcal{A}}, \rho|\vartheta_\alpha^i)} \tag{7}$$

Once the agent has chosen ϑ_α from the distribution, it evaluates the marginal gain of the trajectory given by $V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho|\vartheta_\alpha) - V^{\mathcal{A}}(\mathcal{U}^{\mathcal{A}}, \rho)$. If the marginal gain is positive, or if the simulated annealing acceptance given by line 8 is satisfied, then the agent accepts the new trajectory and notifies all other agents of the change. The temperature decreases by the cooling schedule in (1), and the above process is repeated. `CooperativeTrajectorySearch` is illustrated in Figure 2(b). `DDRP` and `CooperativeTrajectorySearch` are run in parallel, where the trajectory set $\mathcal{V}^{\mathcal{A}}$ for agents is constantly being updated as trajectories are discovered in `DDRP`. We call the resulting framework `CDDRP`.

Algorithm 2: `CooperativeTrajectorySearch`

```
1  CooperativeTrajectorySearch (V, A):
2      ρ = [ϑ_1^0, ϑ_2^0, ...]
3      T = ∞
4      for k ∈ [2, 3, ...]:
5          α ← RandomSelection(A)
6          ϑ ← sample from Pr^ϑ
7          if V^A(ρ|ϑ, Q) > V^A(ρ, Q)
8          or Random(0,1) < e^((V^A(ρ|ϑ,Q) − V^A(ρ,Q))/T)
9              ρ = ρ|ϑ
10         T = c/log(k)
```

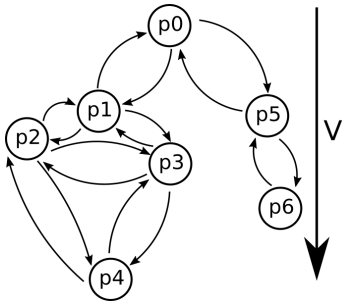


Fig. 3. (Left): Pseudocode description of `CooperativeTrajectorySearch`. (Right): A graph generated from the Markov chain $\langle \rho, Pr, V \rangle$. Nodes represent configurations of task trajectory lists and edges represent positive state transition probabilities $Pr(\rho, \rho')$ between states ρ and ρ'. The graph is ordered such that the y-component of a node increases as node value decreases.

This process can be characterized by a Markov chain. The Markov chain is $\mathbb{C} = \langle \mathcal{P}, Pr^\rho, V^{\mathcal{A}} \rangle$ which contains the set of possible active trajectory lists $\mathcal{P}$,

the probability distribution Pr^{ρ} that encodes the chance of hopping from one active trajectory length to another, and $V^{\mathcal{A}}$. Figure 3 gives an example of the Markov chain that characterizes our system. For the following analysis we define $\mathcal{P}^* = \{\rho : \rho \in \mathcal{P} \text{ s.t. } \text{argmax}_{\rho \in \mathcal{P}} \ V^{\mathcal{A}}(\mathcal{Q}, \rho)\}$ as the set of active trajectory lists that maximize $V^{\mathcal{A}}$ (more than one may exist).

5 Performance of selection schemes: 'flat' vs. 'weighted'

In this section we show experimental results for two trajectory selection schemes and show relative performance differences in the same environments. We perform 3 simulations with the parameters shown in Table 1.

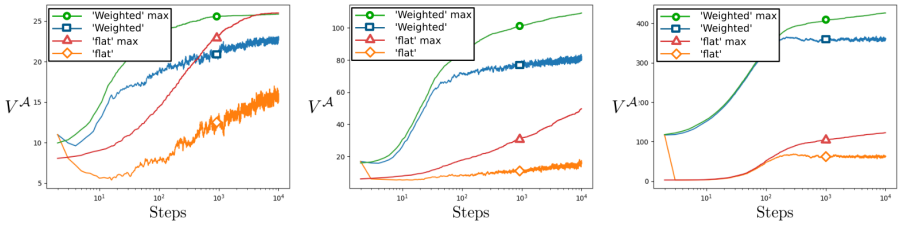
simulation	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V}\rvert$	N_ϵ	c	Δt	total time
1	5	5	5	20	1×10^{-4}	∼1s
2	10	10	5	100	2×10^{-4}	∼2s
3	100	100	3	800	2×10^{-3}	∼20s

Table 1. Parameters used in the simulations.

In these simulations, we assume that the agents have run `DDRP` for some time to generate $\mathcal{V}$ and then use `CooperativeTrajectorySearch` to find ρ. We create the simulation set up with 100 objectives, 50 of which require $2-5$ agents to complete. Rewards for satisfying an objective are randomly picked from a range that scales with the number of agents required to satisfy the objective (which incentivizes collaboration). Varying parameters in the three simulations are number of agents, number of trajectories in their respective trajectory sets, length of the sub-environments they create, and approximated c. The average time per step, denoted by Δt in Table 1, varies because the agent needs to calculate Pr^{ϑ}. Each simulation is run 100 times for both the 'weighted' and 'flat' schemes. We illustrate the results for each simulation in Figure 4. The average time for completion of experiments for cases 1,2 and 3 were 1, 2, and 20 seconds, respectively. In all cases, the maximal value found by the 'weighted' scheme approaches the optimal joint trajectory value sooner than the 'flat' approach. We find that the 'flat' scheme struggles to find joint trajectories with values comparable to the 'weighted' scheme when used for large action spaces. In cases with low action space, such as simulation 1, the maximal value determined from the 'flat' scheme was able to approach the optimal trajectory value. The average value of both schemes is lower than the maximal value found due to the algorithms propensity to escape local maximums.

6 Conclusions

We have extended the dynamic domain reduction framework for multi-agent planning over long time horizons and a variety of objectives to scenarios where



(a) Simulation 1 (b) Simulation 2 (c) Simulation 3

Fig. 4. Results for simulations. Here the joint trajectory discounted reward is shown on the y-axis and the number of steps in log-scale base 10 is shown on the x-axis. We plot the average discounted reward of the current states in each time step of the simulations which are labeled 'weighted' and 'flat'. The lines that correspond to 'weighted' max and 'flat' max indicate the max value that was found in each trial by time step k, averaged over all trials.

some objectives require two or more agents to complete. In order to do this, we have made modified DDRP to allow for necessary information to be shared amongst agents. These include the inclusion of time abstractions and cooperative objectives, and modifications to both trajectories and sub-environments. Building on this framework, we have designed an algorithm based on simulated annealing that allows agents to expedite the exploration of solutions by increasing the chance that they choose tasks that help one another. This is important in the distributed setting in order to reduce the communication needed to find a good solution. Our analysis of the algorithm has shown that, given enough time, the active trajectory list converges in probability to an optimal active trajectory. We do this by showing that the Markov chain that characterizes our multi-agent process satisfies weak reversibility and strong irreducibility properties, and by using a logarithmic cooling schedule. Simulations compare our algorithm with a weighted approach versus our algorithm with a flat approach when it comes to agent select trajectories. In the future, we plan to develop efficient methods for the agents to come up with their trajectories during DDRP, examine the trade-offs in designing how the simulated annealing process can influence the search of trajectories in DDRP, and introduce asynchronous implementations to broaden the utility for real-world scenarios.

References

1. A. Ma, M. Ouimet, and J. Cortés, "Dynamic domain reduction for multi-agent planning," in *International Symposium on Multi-Robot and Multi-Agent Systems*, Los Angeles, CA, 2017, pp. 142–149.
2. B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

3. F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009, electronically available at http://coordinationbook.info.
4. M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*, ser. Applied Mathematics Series. Princeton University Press, 2010.
5. M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 24–39, 2012.
6. J. Das, F. Py, J. B. J. Harvey, J. P. Ryan, A. Gellene, R. Graham, D. A. Caron, K. Rajan, and G. S. Sukhatme, "Data-driven robotic sampling for marine ecosystem monitoring," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1435–1452, 2015.
7. J. Cortés and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 6, pp. 495–503, 2017.
8. R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
9. F. Broz, I. Nourbakhsh, and R. Simmons, "Planning for human-robot interaction using time-state aggregated POMDPs," in *AAAI*, vol. 8, 2008, pp. 1339–1344.
10. M. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
11. R. Howard, *Dynamic programming and Markov processes*. M.I.T. Press, 1960.
12. D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.
13. L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *ECML*, vol. 6. Springer, 2006, pp. 282–293.
14. R. Parr and S. Russell, *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley Berkeley, CA, 1998.
15. A. Bai, S. Srivastava, and S. Russell, "Markovian state and action abstractions for MDPs via hierarchical MCTS," in *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence, IJCAI*, New York, NY, 2016, pp. 3029–3039.
16. A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
17. C. Boutilier, "Sequential optimality and coordination in multiagent systems," in *Proceedings of the 16th international joint conference on Artifical intelligence (IJCAI)*, vol. 1, 1999, pp. 478–485.
18. S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
19. P. Laarhoven and E. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.
20. M. Malek, M. Guruswamy, M. Pandya, and H. Owens, "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 21, no. 1, pp. 59–84, 1989.
21. B. Hajek, "Cooling schedules for optimal annealing," *Mathematics of Operations Research*, vol. 13, no. 2, pp. 311–329, 1988.
22. B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization," *Journal of the Operational Research Society*, vol. 57, no. 10, pp. 1143–1160, 2006.