Dynamic domain reduction for multi-agent planning

Aaron Ma Michael Ouimet Jorge Cortés

***Abstract*— We consider a scenario where a swarm of arbitrary unmanned vehicles (UxVs) are used to satisfy a multitude of diverse, spatially distributed objectives. The UxVs strive to determine an efficient schedule of tasks to service the objectives while operating as a swarm. We focus on developing autonomous high-level planning, where low-level controls are leveraged from previous work in distributed motion, target tracking, localization, and communication algorithms. We take a Markov decision processes (MDP) approach to develop a multi-agent framework that can extend to multi-objective optimization and human-interaction for swarm robotics. Utilizing state and action abstractions, we introduce a hierarchical algorithm, *Dynamic domain reduction for multi-agent planning*, to enable multi-agent planning for large multi-objective environments. Simulated results show significant improvement over using a standard *Monte Carlo tree search* in an environment with large state and action spaces.**

I. Introduction

Recent technology has enabled the deployment of UxVs under human control in a wide range of scenarios. UxVs have many purposes in society, such as intelligence, surveillance and reconnaissance, disaster response, exploration, and surveying for agriculture. In many scenarios, people control these unmanned vehicles. More often than not, multiple people are required to control and guide UxVs. Reducing UxV dependence on human effort enhances their capability in scenarios where communication is expensive or nonexistent as agents can make smart and safe choices on their own. In this paper we design a framework for enabling multi-agent autonomy within a swarm in order to satisfy arbitrary objectives. Planning presents a challenge because as the size of the swarm, environment, and objectives increase, determining optimal actions becomes computationally expensive.

Literature review: Recent algorithms for decentralized methods of multi-agent deployment and path planning enable agents to use local information to satisfy some global task. A variety of decentralized methods can be used for deployment of mobile sensors with the ability to monitor regions, see e.g., [1]–[3] and references therein. We gather motivation from these decentralized methods because they enable centralized goals to be realized with decentralized computation and autonomy.

In general, these decentralized methods provide approaches for low-level autonomy in multi-agent systems, so we look to systems research for high-level planning and scheduling algorithms. In the traveling salesperson problem (TSP) [4], an agent must visit a set of nodes and return to the origin node while traversing the minimal distance possible. An example inspired from nature models an ant colony system to develop a cooperative learning algorithm [5] as a decentralized approach to the TSP. Orienteering [6] is a variant of TSP, where an agent strives to gain the most reward by visiting nodes constrained by some maximum distance, closely fits the desired framework for this problem. Although TSP/Orienteering have been well studied, the frameworks are not directly suitable for multi-objective optimization, or human-interaction extensions. The *planning domain definition language* [7] is a language and framework developed for artificial intelligence planning and can solve a broad class of problems, though it cannot easily handle probabilistic actions or the large state spaces induced by multi-agent planning.

In machine learning research, Markov decision processes (MDP), Semi Markov decision processes (SMDP) and Partially-Observable MDP (POMDP) are standard frameworks for temporal planning under uncertainty [8]–[11]. The MDP framework is conducive to constructing a policy for optimally executing actions in an environment given an infinite time horizon [12]. The focal point in machine learning research is addressing the challenges imposed by the curse of dimensionality. Many works attempt to lower the state space through abstracting the state space [13], history in POMDPs [14], and action space [15]. Other research attempts to alleviate the curse of dimensionality through hierarchical methods. The work [16] introduces an algorithm that allows an agent to simultaneously optimize hierarchical levels, learn abstracted actions, and abstract the state space with a chosen abstraction function. The decentralized POMDP is a framework that incorporates joint decision making and collaboration of multiple agents under uncertain and high-dimensional environments. *Masked Monte Carlo Search* is an algorithm proposed in [17] that determines joint abstracted actions in a centralized way for multiple agents that plan their trajectories in a decentralized POMDP. In contrast, our approach performs task allocation among the agents in a distributed fashion and does not, at the moment, handle tasks that cannot be performed by agents individually.

Recent efforts have focused on bridging the gap between high-level mission planning and low-level control algorithms. Belief states are used to contract the expanding history and curse of dimensionality found in POMDPs. Inspired by Rapidly-Exploring Randomized Trees [18], the Belief Roadmap [19] allows an agent to find minimum cost paths efficiently by finding a trajectory through belief spaces. Sim-

A. Ma and J. Cortés are with the Department of Mechanical and Aerospace Engineering, University of California, San Diego. {aam021,cortes}@ucsd.edu. M. Ouimet is with SPAWAR Systems Center Pacific, San Diego. ouimet@spawar.navy.mil

ilarly, the algorithm in [20] creates Gaussian belief states and exploits feedback controllers to reduce POMDPs to MDPs for tractability in order to find a trajectory. The approach we introduce here provides a higher-level framework for multi-agent planning where these algorithms can be incorporated at a lower path planning level.

Statement of contributions: Our goal is to synthesize a multi-agent algorithm that enables agents to abstract and plan over large, complex environments taking advantage of the benefits resulting from coordinating their actions. We determine meaningful ways to represent the environment and develop an algorithm that reduces the computational burden on an agent to determine a plan. We introduce methods of generalizing positions of agents, and objectives with respect to proximity. We introduce sub-environments, which are subsets of the environment with respect to proximity-based generalizations. We use high-level actions, with the help of low-level controllers, designed to reduce the action space and to plan in the sub-environments. The main contribution of the paper is an algorithm for splitting the work of an agent between dynamically constructing and evaluating sub-environments and learning how to best act in that sub-environment. Finally, we introduce modifications that enable multi-agent deployment by allowing agents to interact with other agents' plans. We illustrate the effectiveness of dynamically constructing sub-environments for planning in environments with large state spaces through simulation and compare against Monte Carlo tree search techniques.

Organization: The paper is organized as follows. Section II presents preliminaries on Markov decision processes. Section III introduces the problem of interest. Section IV describes our approach to abstract states and actions with respect to spatial proximity, and Section V builds on this to design our dynamic domain reduction algorithm. We gather our conclusions and ideas for future work in Section VI.

Notation: We use $\mathbb{Z}$, $\mathbb{Z}_{\geq 1}$, $\mathbb{R}$, and $\mathbb{R}_{>0}$ to denote integers, positive integers, real numbers, and positive real numbers respectively. We use $|Y|$ to denote the cardinality of any arbitrary set Y. We employ object-oriented notation throughout the paper; $b.c$, means that c belongs to b, for arbitrary objects b and c. For reference, Table I presents a list of the most commonly used symbols throughout the paper.

II. Preliminaries

We follow the exposition from [16] to describe a Markov decision process as a tuple $\langle S, A, Pr^s, R\rangle$. S and A are the state and action spaces, respectively. $Pr^s(s'|a,s)$ is a transition function that returns the probability that state $s \in S$ becomes state s' after taking action $a \in A$. $R(s,a,s')$ is the reward that an agent gets after taking action a from state s to reach state s'. A policy is a feedback control that maps a state to an action, $\pi : s \rightarrow a,\ \forall s$. The value of a state under a given policy is

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} Pr^s(s'|\pi(s),s) V^\pi(s'),$$

where $\gamma \in (0,1)$ is a discount factor, which makes the value function converge with infinite steps. The solution to the MDP is an optimal policy that maximizes the value function, $\pi^*(s) = \text{argmax}_\pi\ V^\pi(s)$ for all s. The value of taking an action at a given state under a given policy is

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} Pr^s(s'|a,s) V^\pi(s')$$

Usual methods for obtaining π^* require a tree search of the possible states that can be reached by taking a series of actions. The rate at which the tree of states grow is the branching factor, which is a challenge for solving MDPs with large state spaces and actions with low-likelihood probabilistic state transitions. A technique often used to decrease the state space is called state abstraction, where a collection of states are clustered into one in a meaningful way. Generally, an abstraction function $\phi_s : s \rightarrow s_\phi$, is used to determine an abstracted state. Similarly, actions can be abstracted with $\phi_a : a \rightarrow a_\phi$. Abstracting actions is used to decrease the action space, which can make π^* easier to calculate. In MDPs, actions take one time step per action, however abstracted actions may take a probabilistic amount of time to complete, $Pr^t(t|a_\phi, s)$. When considering the problem using abstracted actions $a_\phi \in A_\phi$ in $\langle S, A_\phi, Pr^s, R\rangle$, the process becomes a semi-Markov Decision Process (SMDP), which allows for probabilistic time per abstracted action. The loss of precision in the abstracted actions means that an optimal policy for an SMDP with abstracted modifications may not be optimal with respect to the original MDP. Determining π^* often involves constructing a tree of reachable MDP/SMDP states determined through simulating actions from an initial state. Dynamic programming is commonly used for approximating π^* by using a Monte Carlo tree search to explore the MDP for the initial state. The action with the maximum upper confidence bound (UCB) [21] of the approximated expected value for taking the action at a given state,

$$\text{argmax}_{a \in A}\ \Big\{ Q(s,a) + C\sqrt{\frac{\ln(N_s)}{N_{s,a}}} \Big\},$$

is often used to efficiently explore the MDP. Here, N_s is the number of times a state has been visited, $N_{s,a}$ is the number of times that action a has been taken at state s and C is a constant. Taking this action allows for a combination of both exploring the reachable states from an initial state and receiving high expected reward.

III. Problem statement

Consider a set of agents, $\alpha \in \mathcal{A}$. For simplicity of exposition we assume agents are able to communicate with each other and have access to the location of each other. We leave scenarios with constrained communication for future treatment of our algorithm and framework. An agent, α, occupies a point in $\mathbb{Z}^d$, and has computational, communication, and mobile capabilities. Let $o \in \mathbb{Z}^d$ be a waypoint that an agent must visit in order to serve an objective. Every

waypoint belongs to an objective, $\mathcal{O}_b = \{o_1, \ldots, o_{|\mathcal{O}_b|}\}$. Agents are able to satisfy objectives when waypoints are visited such that $o \in \mathcal{O}_b$ is removed from $\mathcal{O}_b$. An agent receives a reward, $r^o \in \mathbb{R}$, for servicing a sub-objective, $o \in \mathcal{O}_b$. Define the set of objectives to be $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_{|\mathcal{O}|}\}$, and assume agents are only able to service one $\mathcal{O}_b \in \mathcal{O}$ at a time. We consider the environment to be $\mathcal{E} = \mathcal{O} \times \mathcal{A}$, which contains information about all objectives and agents. The state size of $\mathcal{E}$ increases exponentially with respect to $|\mathcal{O}|$, $|\mathcal{O}_b|$, and $|\mathcal{A}|$. We strive to design a decentralized algorithm that allows agents in $\mathcal{A}$ to individually compute a policy, π^*, that optimally services objectives in $\mathcal{O}$ in scenarios where $\mathcal{E}$ is very large. We determine abstraction strategies that reduce the stochastic branching factor in order to find a good policy in the environment. To tackle this, the paper is structured as follows: we begin by spatially abstracting objectives in an environment into regions and we design high-level actions that the agents take. We dynamically create and search subsets of the environment to reduce dimensions of the state that agents reason over. Then we structure a plan of high-level actions with respect to the subset of the environment. Next, we make modifications necessary for multi-agent planning and finally introduce the *Dynamic domain reduction for multi-agent planning* (DDRMAP) algorithm to approximate the optimal policy.

IV. Abstractions

In order to leverage dynamic programming solutions to approximate a good policy, we reduce the state space and action space. We begin by introducing methods of abstraction for states and actions with respect to spatial proximity.

A. Single-agent abstractions

The number of unique states grows exponentially with respect to the number of dimensions that exist ($|\mathcal{O}|, |\mathcal{O}_b|$ for all b, and $|\mathcal{A}|$). First, we cluster waypoints and agent locations with respect to regions in the Cartesian product, which we call region abstraction, then we construct abstracted actions that agents are allowed to execute with respect to the region abstractions.

Region abstraction: We define a region to be a convex set $x \subseteq \mathbb{R}^d$ and consider a set of $\mathcal{X}$ all regions such that all $x \in \mathcal{X}$ are disjoint. For simplicity, we consider regions to be equal in size, shape, and orientation, so that the set of regions creates a tessellation of the environment. We leave non-regular/repeating regions as future extensions to our formulation. Furthermore, let the set of waypoints of objective, $\mathcal{O}_b$, in region x_i be denoted $\mathcal{O}_b^{x_i}$ such that $\mathcal{O}_b^{x_i} \subseteq x_i$. We use an abstraction function, $\phi : \mathcal{O}_b^{x_i} \to s_i^b$, to get the **abstracted objective state**, s_i^b, which enables us to generalize the states of an objective in a region. In general, the abstraction function is designed by a human user that distinguishes importance of an objective in a region. We define a **regional state**, $\mathcal{S}_{x_i} = (s_i^1, \ldots, s_i^{|\mathcal{O}|})$, to be the Cartesian product of s_i^b for all objectives, $\mathcal{O}_b \in \mathcal{O}$, with respect to x_i.

Action abstraction: We assume that low-level feedback controllers are available, allowing for servicing of waypoints. We describe how we utilize low-level controllers as components of a task to reason over. We define a **task** to be $\tau = \langle s_i^b, s_i'^b, x_i, x_j, \mathcal{O}_b \rangle$, where s_i^b, and $s_i'^b$ are abstracted objective states of x_i, x_j is a target region, and $\mathcal{O}_b$ is a target objective. We assume that low-level controllers exists that can satisfy the following requirements.

- Objective transition: low-level controllers executing τ, will drive the state transition, $\tau.s_i^b \to \tau.s_i'^b$.
- Regional transition: low-level controller executing, τ, drives the agent's location to x_j after the objective transition is complete.

Candidates of low-level controllers include policies determined by using [16] and [21] after setting up the region as a MDP, modified traveling salesperson [6], or path planning based algorithms interfaced with the dynamics of the agent. Agents that start a task are required to continue working on the task until requirements are met. Because the tasks are dependent on abstracted objectives states, the agent completes a task in a probabilistic time, given by $Pr^t(t|\tau)$, that is determined heuristically. An agent can only choose to start a task if the task is in the feasible task set. The **feasible task set**, $\tau \in \Gamma$, includes all possible tasks that are feasible to an agent. We consider τ to be feasible if the following are possible:

- $\tau.s_i^b$ is the abstracted objective state of $\tau.\mathcal{O}_b$ at region $\tau.x_i$ at the time of starting τ.
- $\tau.s_i'^b$ exists, and can be transitioned to from $\tau.s_i^b$.
- Agent resides in $\tau.x_i$ at the time of starting τ.
- $\tau.x_j$ exists, and the agent can transition to it from $\tau.x_i$.

Sub-environments: In order to further alleviate the curse of dimensionality, we introduce sub-environments, a subset of the environment, in an effort to only utilize relevant regions. Let the size of the sub-environment be $N_\epsilon \in \mathbb{Z}_{\geq 1}$. Let $\overrightarrow{x} = [\overrightarrow{x}_1, \ldots, \overrightarrow{x}_{N_\epsilon}]$ be an ordered list of regions, such that $\overrightarrow{x}_k \in \mathcal{X}$. We consider $\mathcal{S}_{\overrightarrow{x}_1} = \mathcal{S}_{x_i}$ if $\overrightarrow{x}_1 = x_i$. In order to simulate the sub-environment, the agent must know if there are repeated regions in $\overrightarrow{x}$. Let $\xi(k, \overrightarrow{x})$, return the first index of $\overrightarrow{x}$, h, such that $\overrightarrow{x}_h = \overrightarrow{x}_k$, or 0 if there is no match. Define the **repeated region list** to be $\xi_{\overrightarrow{x}} = [\xi(1, \overrightarrow{x}), \ldots, \xi(N_\epsilon, \overrightarrow{x})]$. Let $\phi_t(x_i, x_j) : x_i, x_j \to \mathbb{Z}$, be an abstracted amount of time it takes for an agent to move from x_i to x_j, or ∞ if no path exists. Let $S = [\mathcal{S}_{\overrightarrow{x}_1}, \ldots, \mathcal{S}_{\overrightarrow{x}_{N_\epsilon}}] \times \xi_{\overrightarrow{x}} \times [\phi_t(\overrightarrow{x}_1, \overrightarrow{x}_2), \ldots, \phi_t(\overrightarrow{x}_{N_\epsilon - 1}, \overrightarrow{x}_{N_\epsilon})]$, be the abstracted state for a chosen $\overrightarrow{x}$. A **sub-environment**, $\epsilon = \langle \overrightarrow{x}, S \rangle$, is valid if and only if the following are satisfied.

- No elements of $[\phi_t(\overrightarrow{x}_1, \overrightarrow{x}_2)), \ldots, \phi_t(\overrightarrow{x}_{N_\epsilon - 1}, \overrightarrow{x}_{N_\epsilon})]$ are ∞.
- The agent begins in the initial region, $\overrightarrow{x}_1$.

In general, we allow a sub-environment to contain any region that is reachable in finite time. However, in practice, we only allow agents to choose sub-environments that they can traverse within some reasonable time in order to reduce the number of possible sub-environments and save onboard

memory.

Task trajectory: We also define an ordered list of tasks that the agents execute with respect to a sub-environment ϵ. Let $\overrightarrow{\tau} = [\overrightarrow{\tau}_1, \ldots, \overrightarrow{\tau}_{N_\epsilon - 1}]$ be an ordered list of feasible tasks such that $\overrightarrow{\tau}_k.x_i = \epsilon.\overrightarrow{x}_k$, and $\overrightarrow{\tau}_k.x_j = \epsilon.\overrightarrow{x}_{k+1}$ for all $k \in \{1, \ldots, N_\epsilon - 1\}$. Agents generate this ordered list of tasks assuming that they will execute each of them in order. The probability distribution on the time of completing the k^{th} task in $\overrightarrow{\tau}$ (after completing all previous tasks in $\overrightarrow{\tau}$) is given by $\overrightarrow{Pr^t}_k$. We define $\overrightarrow{Pr^t} = [\overrightarrow{Pr^t}_1, \ldots, \overrightarrow{Pr^t}_{N_\epsilon - 1}]$ to be the ordered list of probability distributions. We construct the **task trajectory** to be $\vartheta = \left\langle \overrightarrow{\tau}, \overrightarrow{Pr^t} \right\rangle$, which is used to determine the finite time reward for a sub-environment.

Abstracted rewards: The exact reward that an agent would receive when acting on the environment, $R(\mathcal{E}', a, \mathcal{E})$ is a function of $\mathcal{O}$ and $\mathcal{O}'$, the objective set before and after being serviced respectively. Because we have abstracted states and actions, we must determine the reward an agent receives for completing τ as a probabilistic function that is determined heuristically. Let $Pr^r(r|\tau)$ be the probability that an agent receives r reward after completing τ. We determine the total reward an agent receives for executing a trajectory as

$$r^\epsilon = \sum_{k=1}^{N_\epsilon - 1} \sum_{t=0}^{\infty} \sum_{r \in \mathbb{R}_{>0}} \overrightarrow{Pr^t}_k(t) Pr^r(r|\overrightarrow{\tau}_k)\gamma^t r. \tag{1}$$

We strive to determine the optimal policy $\pi^{*\epsilon} : \epsilon \to \vartheta$ that satisfies

$$\pi^{*\epsilon} = \underset{\pi^\epsilon}{\text{argmax}}\ r^\epsilon, \tag{2}$$

for any valid ϵ. We approximate this optimal policy with Algorithm 1 introduced in Section V.

B. Multi-agent abstractions

Due to the large environment induced by the action coupling of multi-agent joint planning, determining the optimal policy is computationally infeasible. To reduce computational burden on an agent, we restrict the number of coupled interactions. In this section, we modify the sub-environment, task trajectory, and rewards to allow for multi-agent coupled interactions. The following equations and algorithms are taken from the perspective of an arbitrary agent α in the swarm, where other agents are indexed with $\beta \in \mathcal{A}$.

Sub-environment modifications: Agents may choose to interact other agents in the **interaction set**, $\beta \in \mathcal{I}_\alpha \subseteq \mathcal{A}$, while executing $\overrightarrow{\tau}_\alpha$. The interaction set is constructed as a parameter of the sub-environment and indicates to the agent which tasks should be avoided based on the other agents' trajectories. Let $\mathcal{N}$ be the maximum number of agents that an agent can interact (hence $|\mathcal{I}_\alpha| \leq \mathcal{N}$). The interaction set is constructed by adding another agent and their interaction set, $\mathcal{I}_\alpha = \mathcal{I}_\alpha \cup \{\beta\} \cup \mathcal{I}_\beta$. If $|\mathcal{I}_\alpha \cup \{\beta\} \cup \mathcal{I}_\beta| > \mathcal{N}$, then we consider β to be an invalid candidate for interaction. Adding β's interaction set is necessary because tasks that affect ϑ_β, may also affect all agents in $\mathcal{I}_\beta$. Constraining the maximum interaction set size reduces the large state size that occurs when agents' actions are coupled. In order to avoid interacting with agents not in the interaction set, we create a set of waypoints that are off-limits when creating a trajectory.

We define a **claimed regional objective** as $\theta = \langle \mathcal{O}_b, x_i \rangle$. The agent creates a set of claimed region objectives, $\Theta_\alpha = \{\theta_1, \ldots, \theta_{N_\epsilon - 1}\}$ that contains a claimed region objective for every task in an agent's trajectory and describes a waypoint in $\mathcal{O}_b$ in a region that the agent is planning to service. We define the **global claimed objective set** to be $\Theta^{\mathcal{A}} = \left\{\Theta_1, \ldots, \Theta_{|\mathcal{A}|}\right\}$, which contains the claimed region objective set for all agents. Lastly, let $\Theta'_\alpha = \Theta^{\mathcal{A}} \setminus \left\{\bigcup_{\beta \in \mathcal{I}_\alpha} \Theta_\beta\right\}$, which contains the complete set of claimed objectives an agent must avoid when planning a trajectory. The agent uses Θ'_α to modify its perception of the environment. As shown in the following function, the agent sets the state of claimed objectives in Θ'_α to 0, removing appropriate tasks from the feasible task set.

$$s^b_{\Theta'_\alpha, \overrightarrow{x}_k} = \begin{cases} 0 & \text{if } \langle \mathcal{O}_b, \epsilon_\alpha.\overrightarrow{x}_k \rangle \in \Theta'_\alpha, \\ s^b_{\overrightarrow{x}_k} & \text{otherwise.} \end{cases} \tag{3}$$

Let $\overrightarrow{\mathcal{S}}_{\Theta'_\alpha, \overrightarrow{x}} = \overrightarrow{\mathcal{S}}_{\Theta'_\alpha, \overrightarrow{x}_1} \times \ldots \times \overrightarrow{\mathcal{S}}_{\Theta'_\alpha, \overrightarrow{x}_{N_\epsilon}}$, where $\overrightarrow{\mathcal{S}}_{\Theta'_\alpha, \overrightarrow{x}_k} = (s^{\mathcal{O}_1}_{\Theta'_\alpha, \overrightarrow{x}_k}, \ldots, s^{\mathcal{O}_{|\mathcal{O}|}}_{\Theta'_\alpha, \overrightarrow{x}_k})$. In addition to the modified sub-environment state, we include the partial trajectories of other agents being interacted with. Consider β's trajectory, ϑ_β, and an arbitrary ϵ_α. Let $\vartheta^p_{\beta,k} = \left\langle \vartheta_\beta.\overrightarrow{\tau}_k.s^b_i,\ \vartheta_\beta.\overrightarrow{\tau}_k.\mathcal{O}_b \right\rangle$. The **partial trajectory**, $\vartheta^p_\beta = [\vartheta^p_{\beta,1}, \ldots, \vartheta^p_{\beta,|\vartheta_\beta|}]$ describes β's trajectory with respect to $\epsilon_\alpha.\overrightarrow{x}$. Let $\xi(k, \epsilon_\alpha, \epsilon_\beta)$ return the first index of $\epsilon_\alpha.\overrightarrow{x}$, h, such that $\epsilon_\beta.\overrightarrow{x}_h = \epsilon_\alpha.\overrightarrow{x}_k$, or 0 if there is no match. Each agent in the interaction set creates a matrix, Ξ, of elements, $\xi(k, \epsilon_\alpha, \epsilon_\beta)$, for $k \in \{1, \ldots, N_\epsilon\}$ and $\beta \in \{1, \ldots, |\mathcal{I}_\alpha|\}$. We finally determine the **complete multi-agent state**, $S = \overrightarrow{\mathcal{S}}_{\Theta'_\alpha, \overrightarrow{x}} \times \left\{\langle \vartheta^p_1, \Xi_1 \rangle, \ldots, \langle \vartheta^p_{|\mathcal{I}_\alpha|}, \Xi_{|\mathcal{I}_\alpha|} \rangle\right\} \times [\phi_t(\overrightarrow{x}_1, \overrightarrow{x}_2), \ldots, \phi_t(\overrightarrow{x}_{N_\epsilon - 1}, \overrightarrow{x}_{N_\epsilon})]$. With modifications to the sub-environment abstracted state, we redefine the **sub-environment** as $\epsilon_\alpha = \langle \overrightarrow{x}, S, \mathcal{I}_\alpha \rangle$.

Multi-agent abstracted actions: We consider the effect that an agent has on another agent $\beta \in \mathcal{A}$ when executing a task that affects s^b_i in β's trajectory. Some tasks will be completed sooner with two or more agents working on them, for instance. For all $\beta \in \mathcal{I}_\alpha$, let t_β be the time that β begins a task that transitions $s^b_i \to s'^b_i$. If agent β does not contain such a task in its trajectory, then $t_\beta = \infty$. Let $T^{\mathcal{A}}_{\mathcal{I}_\alpha} = [t_1, \ldots, t_{|\mathcal{I}_\alpha|}]$. We now define the probability that τ is completed at exactly time t if other agents work on transitioning $s^b_i \to s'^b_i$, as $\Pr^t_{\mathcal{I}_\alpha}(t|\tau, T^{\mathcal{A}}_{\mathcal{I}_\alpha})$. In Section IV-A, we defined $\overrightarrow{Pr^t}$ to be a set of Pr^t that returned the probability that τ will be completed at time t given previous tasks are also completed. We redefine here $\overrightarrow{Pr^t} = [\overrightarrow{Pr^t}_{1,\mathcal{I}_\alpha}, \ldots, \overrightarrow{Pr^t}_{N_\epsilon - 1, \mathcal{I}_\alpha}]$, which is the probability time set of a modified by other agents trajectories. Furthermore, if an agent chooses a task that modifies agent, a's trajectory, we define the probability time set to be $\overrightarrow{Pr^{t\prime}} = [\overrightarrow{Pr^{t\prime}}_{1,\mathcal{I}_\alpha}, \ldots, \overrightarrow{Pr^{t\prime}}_{N_\epsilon - 1, \mathcal{I}_\alpha}]$. With these modifications, we redefine the **task trajectory** to be $\vartheta =$

$\left\langle \overrightarrow{\tau}, \overrightarrow{Pr^t} \right\rangle$.

Multi-agent reward abstractions: We modify (2) so that the objective is for an agent to approximate the optimal policy, while taking into account agents that it may interact with. We redefine the total reward as

$$r^{\epsilon}_{\alpha} = \sum_{k=0}^{N_\epsilon - 1} \sum_{t=0}^{\infty} \sum_{r \in \mathbb{R}_{>0}} \overrightarrow{Pr^t}_{k,\mathcal{I}_\alpha}(t) Pr^r(r|\overrightarrow{\tau}_k)\gamma^t r, \quad (4)$$

where $\Pr^t_{\mathcal{I}_\alpha}$ replaces Pr^t. Furthermore, if the agent's trajectory is being interacted with from another agent, we use the reward r^{ϑ}_{β}. We introduce the **interaction reward function**, which returns a reward based on whether the agent executes a task that interacts with one or more other agents as

$$r^{\phi}(\epsilon, \tau) = \begin{cases} \sum\limits_{\beta \in \mathcal{I}_\alpha} (r^{\epsilon'}_{\beta} - r^{\epsilon}_{\beta}) & \text{if } \tau \in \vartheta^{p}_{\beta} \text{ for any } \beta, \\ \sum\limits_{r \in \mathbb{R}_{>0}} Pr^r(r|\tau) r & \text{otherwise.} \end{cases} \quad (5)$$

In this equation, we quantify the effect that an interacting task has on an existing task. If a task helps another agent trajectory in an impactful way, the agent may choose a task that aids the global expected reward amongst the agents. Finally, we rewrite (2) with the inclusion of (5),

$$\max_{\pi^{*\epsilon}} \sum_{k=0}^{N_\epsilon - 1} \sum_{t=0}^{\infty} \overrightarrow{Pr^t}_k(t) \gamma^t r^{\phi}(\epsilon, \tau) \quad (6)$$
$$s.t.\ \ \pi^{*\epsilon} : \epsilon \to \vartheta.$$

We approximate this optimal policy with Algorithm 1 introduced in Section V.

V. Dynamic domain reduction for MA planning

In this section we introduce our algorithm to approximate the optimal policy $\pi^{*\epsilon}$. Our algorithm consists of three main functions, `DDRMAP`, `TaskSearch`, and `SubEnvSearch`[1]. Algorithm 1 presents a formal description of the deployment algorithm in the multi-agent case. The deployment algorithm for the case of a single agent simply corresponds to taking $\mathcal{N} = 0$ (implying that $\mathcal{I}$ and Θ' are empty) in Algorithm 1.

A. Algorithm and function overview

First, we describe necessary variables and parameters, then we discuss each of the main functions and the role of the support functions. In our algorithm, we allow agents to switch trajectories based on a polling time, $T \in \mathbb{Z}_{\geq 1}$, as a design parameter. We consider the size of the sub-environments, N_ϵ and the polling time, T, to be global constants in the swarm. Let N, $N_{\mathcal{O}_b}$, and $N^k_{\mathbb{E}}$ be global variables in the swarm, which mature as the agent accrues more data. The variables Q, N, and $N_{\mathcal{O}_b}$ correspond, respectively, to the expected discounted value of choosing a task when the state is $\epsilon.S$ under the current policy, the number of times the agent has visited $\epsilon.S$, and the number of times the agent has simulated choosing $\mathcal{O}_b$ in $\epsilon.S$. Let ϵ_k be a sub-environment of length $k < N_\epsilon$. Let $Q^k_{\mathbb{E}}$ and $N^k_{\mathbb{E}}$ be the estimation of Q, given that $\epsilon_k \in \epsilon$, and the number of times the agent has simulated $\epsilon_k.S$ respectively. Let $\mathbb{W}_\epsilon$ and $\mathbb{W}_{\epsilon_k}$, be sets of ϵ, and ϵ_k respectively that contain possible sub-environment subsets of $\mathcal{E}$ at some future time.

[1]pseudocode of functions denoted with † is omitted for space reasons

Algorithm 1 : Dynamic domain reduction for MA planning

$T \in \mathbb{Z}_{\geq 1}$

`DDRMAP` ():
- $t = T$, $\mathbb{W}_{\epsilon_k} \to \epsilon_0$, $\mathbb{W}_\epsilon \to \emptyset$
- **repeat** forever:
  - $\mathcal{E} \leftarrow$ current environment
  - $\mathcal{E}' \leftarrow$ `EstimateEnv`† $(\mathcal{E}, \epsilon, \vartheta, t)$
  - $\Theta^{\mathcal{A}} \leftarrow \bigcup \Theta_\beta,\ \forall \beta \in \mathcal{A}$
  - $\mathbb{W}_{\epsilon_k}, \mathbb{W}_\epsilon \leftarrow$ `ValidSubEnvSets`† $(\mathbb{W}_{\epsilon_k}, \mathbb{W}_\epsilon, \mathcal{E})$
  - **while** run time $<$ step time:
    - $\epsilon_k \leftarrow \max\limits_{\epsilon_k \in \mathbb{W}_{\epsilon_k}} \left\{ Q^k_{\mathbb{E}}[\epsilon_k.S] + \frac{c\,\sigma^k_{\mathbb{E}}[\epsilon_k.S]}{\sqrt{N^k_{\mathbb{E}}[\epsilon_k.S]}} \right\}$
    - $\epsilon \leftarrow$ `SubEnvSearch` $(\epsilon_k, \mathcal{E}', \Theta^{\mathcal{A}})$
    - $r \leftarrow$ `TaskSearch` $(\epsilon, 0)$
    - $\mathbb{W}_\epsilon$ .add(ϵ, ϑ, r)
  - **if** $t \leq 0$:
    - $\vartheta \leftarrow$ `MaxTrajectory` $(\mathbb{W}_\epsilon)$
    - $k \leftarrow 0$, $t \leftarrow T$, $\mathbb{W}_{\epsilon_k} \leftarrow \epsilon_0$, $\mathbb{W}_\epsilon \leftarrow \emptyset$
    - $\Theta \leftarrow$ `CreateClaimedObjSet` (ϵ, ϑ)
  - **if** $\overrightarrow{\tau}_k$ is satisfied:
    - $k = k + 1$
  - `Step`† $(\overrightarrow{\tau}_k)$
  - $t = t - 1$

`TaskSearch` (ϵ, d)
- $\Gamma \leftarrow$ `GenerateFeasibleTaskSet`† (ϵ)
- $\tau \leftarrow \max\limits_{\tau \in \Gamma} \left\{ Q[\epsilon.S][\tau.\mathcal{O}_b] + 2C_p \sqrt{\frac{lnN[\epsilon.S]}{N_{\mathcal{O}_b}[\epsilon.S][\tau.\mathcal{O}_b]}} \right\}$
- $t \leftarrow$ `Sample`† $Pr^t(t|\tau, \epsilon)$
- **if** $d + t > T$, **return** 0
- $\epsilon' \leftarrow$ `EvolveSubEnv`† (ϵ)
- $r = \gamma^t$(`GetReward` (τ, ϵ) + `TaskSearch` $(\epsilon', d + t)$)
- `TaskValueUpdate` $(\epsilon.S, \tau.\mathcal{O}_b, r)$
- `SubEnvValueUpdate` (ϵ, r)
- **return** r

`SubEnvSearch` $(\epsilon_k, \mathcal{E}')$
- $\mathbb{W}^+_{\epsilon_k} \leftarrow$ `GetFeasibleSubEnvironments`† $(\epsilon_k, \mathcal{E}')$
- $\mathbb{W}_{\epsilon_k}$.add$(\mathbb{W}^+_{\epsilon_k})$
- $\mathbb{W}_{\epsilon_k}$.remove(ϵ_k)
- $\epsilon^*_{k+1} = \max\limits_{\epsilon_{k+1} \in \mathbb{W}^+_{\epsilon_k}} Q^k_{\mathbb{E}}[\epsilon_{k+1}]$
- **if** $|\epsilon^*_{k+1}| = |N_\epsilon|$, **return** `InitSubEnv` $(\epsilon^*_{k+1}, \mathcal{E}')$
- **return** `SubEnvSearch` $(\epsilon^*_{k+1}, \mathcal{E}')$

The main function, `DDRMAP`, interfaces the real environment to the simulated environment and learning algorithms. At every time step, the agent observes the current environment. The agent chooses a new trajectory when $t = 0$, so the agent estimates what the environment will look like, $\mathcal{E} \to \mathcal{E}'$, using `EstimateEnv`†. `ValidSubEnvSets`† takes the current $\mathbb{W}_{\epsilon_k}$ and $\mathbb{W}_\epsilon$ and removes all sub-environments which become non-feasible in $\mathcal{E}$ in t given the current state of the environment. Until the allocated time runs out, the agent finds a suitable sub-environment, ϵ, in `SubEnvSearch`. The chosen ϵ is used in `TaskSearch`, which returns an expected reward and trajectory, which are added to $\mathbb{W}_\epsilon$. This

process repeats for an allocated amount of time per step. Every T time steps, the agent chooses the feasible trajectory with the maximum expected reward with `MaxTrajectory`. Variables are reset and the set of claimed regional objectives Θ is created with `CreateClaimedObjSet`. Θ is available to the rest of the agents once created. The agent uses the new trajectory to act on the environment with `Step`†, and switches to a task when the previous one is completed. `TaskSearch` takes a given sub-environment and uses a process similar to the UCT algorithm [21], which utilizes the upper confidence bound when searching the state space of a MDP, to approximate an optimal trajectory for that sub-environment. Initially, the agent determines what tasks are feasible given ϵ by using `GenerateFeasibleTaskSet`†, which makes sure that state transitions of a particular objective are possible. The agent chooses a task to take by using the upper confidence bound on Q, and pulls t from the probability time of completion $Pr^t(t|\tau)$. We evolve $\epsilon \rightarrow \epsilon'$, in order to correctly model the simulation. We get the reward of taking τ by using `GetReward`, which executes the process in (5) by estimating the total effect that an agent has on another's trajectory if they intersect, else the amount of reward an agent gets on its own. This process is recursive and continues until $d + t \geq T$, where $d + t$ is the simulated time steps.

We employ `SubEnvSearch` to find the value of the sub-environment ϵ to be used by `TaskSearch`. `SubEnvSearch` is a algorithm that efficiently returns a sub-environment by searching the set of possible sub-environments available to the agent. It takes the sub-environment in $\mathbb{W}_{\epsilon_k}$ with the highest upper confidence bound. The set of feasible extensions of the sub-environment, $\mathbb{W}^+_{\epsilon_k}$, are determined with `GetFeasibleSubEnvironments`†, and is done by adding a region to $\overrightarrow{x}$. $\mathbb{W}^+_{\epsilon_k}$ is added to $\mathbb{W}_{\epsilon_k}$ in order to keep track of possible partially explored sub-environments, and ϵ_k is removed since it is currently being explored. The algorithm recursively looks for the best expected ϵ_k until $k = |\epsilon|$, where the state is initialized in `InitSubEnv` by using `GetModSubEnv`† according to Equation (3).

B. Simulations and results

Here we discuss the results of two simulations of *Dynamic domain reduction for multi-agent planning*. In the first simulation, displayed in Figure 1, we consider an environment with a single agent and compare the performance of three strategies. The first algorithm is a *Monte Carlo tree search* (MCTS) [22], with the abstracted states as the regions we have defined and abstracted states as tasks. The state an agent uses in the MCTS algorithm is the Cartesian product of all region objective states. The second algorithm we test is a version of our algorithm that only uses policies learned offline. We call this algorithm *Dynamic domain reduction planning: offline* (DDRP-O). DDRP-O skips `TaskSearch` and focuses on search for sub-environments and approximating their values with pre-learned policies. The final algorithm

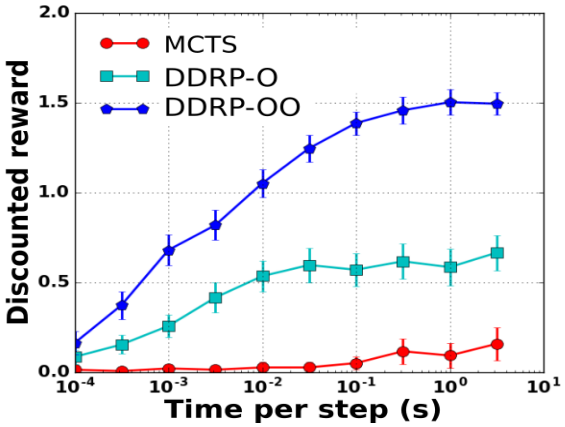

Fig. 1. Empirical results of single-agent simulation.

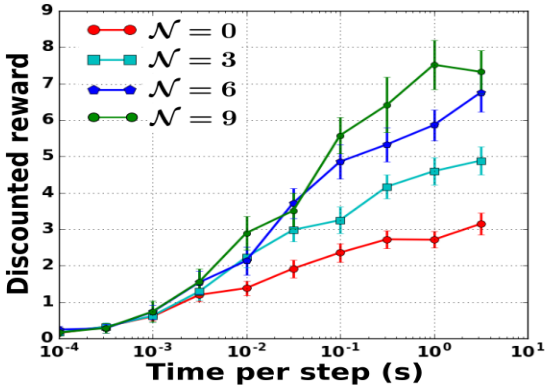

Fig. 2. Empirical results of multi-agent simulation.

we test is called, *Dynamic domain reduction planning: offline+online* (DDRP-OO), which includes `TaskSearch` in order to refine policies and to further explore a sub-environment's states. In the second simulation, displayed in Figure 2, we test the main algorithm, *Dynamic domain reduction for multi-agent planning* with 10 agents using the maximum interaction number, $\mathcal{N}$, as a test parameter.

We perform 100 trials per data point in each of the two simulations. At the beginning of each trial, the agent is given an allocated time to think, as shown by the x-axis. The y-axis shows the expected discounted reward that the agent finds during its allotted time. Data is plotted to show the average and confidence intervals of the expected discounted reward of the agent(s) found in the allotted time. In both simulations, the agent(s) start at location $(50, 50)$ in an environment that is a 100×100 grid. The sub-environment size for both tests is $N_\epsilon = 4$.

In the first simulation, the environment has $|\mathcal{O}| = 10$, where objectives have a random number of waypoints ($\mathcal{O}_b| \leq 15$) placed uniformly randomly. For each of the algorithms (MCTS, DDRP-O, DDRP-OO), the agents are given 10 seconds to train on randomized environments. We chose this

environment because of the massive state space and action space in order to illustrate the strength of Dynamic domain reduction for multi-agent planningin breaking it down into components that have been previously seen. Figure 1 shows that DDRP-O performs well with only 10 seconds of training time on randomized environments because it is able to search for previously found sub-environments. DDRP-OO performs the best because it is able to continue learning sub-environments, and prioritize sub-environments that exist in the environment at hand. The MCTS performs poorly for the chosen environment because of its large state space and branching factor. Offline learning does not help in this situation because, given the state space, the probability of finding the same environment it has previously visited is very low.

The second simulation uses an environment similar to the first simulation, except with 10 agents and $|\mathcal{O}| = 3$, where objectives have a random number of waypoints ($|\mathcal{O}_b| \leq 5$) that are placed randomly. In this simulation we do not train the agents before trials, and *Dynamic domain reduction for multi-agent planning* is used with varying $\mathcal{N}$ where agents asynchronously choose trajectories. In Figure 2, we can see the benefit of allowing agents to interact with each other. When agents are able to take coupled actions, the expected potential discounted reward is greater, a feature that becomes more marked as agents are given more time to think.

VI. Conclusions

We have presented a framework for high-level multi-agent planning leading to the design of the *Dynamic domain reduction for multi-agent planning* algorithm . We have alleviated the curse of dimensionality by searching the environment for sub-environments with the greatest expected reward. We have allowed for multi-agent interaction by allowing for the inclusion of other agents' state in the sub-environment search. Finally we tested in environments with parameters that enable our algorithm to perform well. The biggest limitation of our algorithm is related to the spatial distribution of objectives. The algorithm performs poorly if the environment is set up such that objectives cannot be split well into regions. Otherwise, the hierarchical approach of simultaneously searching for, and creating sub-environments and macro-actions, helps alleviate the curse of dimensionality. In the future, we plan on using this framework in order to fully enable multi-agent, multi-objective optimization, and to explore constraints on battery life and connectivity of the network of agents. To enable multi-agent cooperation, we plan on adding elements of predicting whether other agents will aid in the completion of an objective. We also plan on enabling human-swarm interactions by estimating human intentions through multi-objective preferences.

Acknowledgments

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

References

[1] 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.

[2] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*, ser. Applied Mathematics Series. Princeton University Press, 2010.

[3] W. Ren and R. W. Beard, *Distributed Consensus in Multi-Vehicle Cooperative Control*, ser. Communications and Control Engineering. Springer, 2008.

[4] S. Lin, "Computer solutions of the traveling salesman problem," *The Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, Dec 1965.

[5] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, Apr 1997.

[6] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation algorithms for orienteering and discounted-reward TSP," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 653–670, 2007.

[7] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL-The planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep. TR-98-003, 1998.

[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] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[13] E. A. Hansen and Z. Feng, "Dynamic programming for pomdps using a factored state representation," in *AIPS*, 2000, pp. 130–139.

[14] A. K. McCallum and D. Ballard, "Reinforcement learning with selective perception and hidden state," Ph.D. dissertation, University of Rochester. Dept. of Computer Science, 1996.

[15] G. Theocharous and L. P. Kaelbling, "Approximate planning in POMDPs with macro-actions," in *Advances in Neural Information Processing Systems*, 2004, pp. 775–782.

[16] 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*, 2016, pp. 3029–3039.

[17] S. Omidshafiei, A. A. Agha-mohammadi, C. Amato, and J. P. How, "Decentralized control of partially observable Markov decision processes using belief space macro-actions," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5962–5969.

[18] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Workshop on Algorithmic Foundations of Robotics*, Dartmouth, NH, Mar. 2000, pp. 293–308.

[19] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance," in *Robotics Research*. Springer, 2010, pp. 293–305.

[20] A. A. Agha-mohammadi, S. Chakravorty, and N. M. Amato, "FIRM: Feedback controller-based information-state roadmap-a framework for motion planning under uncertainty," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4284–4291.

[21] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *ECML*, vol. 6. Springer, 2006, pp. 282–293.

[22] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

$\mathbb{Z}$ ..... integer
$\mathbb{Z}_{\geq 1}$ ..... positive integer
$\mathbb{R}$ ..... real number
$\mathbb{R}_{>0}$ ..... positive real number
$|Y|$ ..... cardinality of set Y
s ..... state
S ..... state space
A ..... action space
Pr^s ..... transition function
r, R ..... reward, reward function
π ..... policy
π^* ..... optimal policy
V^π ..... value of a state given policy, π
γ ..... discount factor
α, β ..... agent indices
$\mathcal{A}$ ..... set of agents
o ..... waypoint
$\mathcal{O}_b$ ..... objective
$\mathcal{O}$ ..... set of objectives
$\mathcal{E}$ ..... environment
x ..... region
$\mathcal{X}$ ..... set of regions
$\mathcal{O}_b^{x_i}$ ..... set of waypoints of an objective in a region
s_i^b ..... abstracted objective state
$\mathcal{S}_{x_i}$ ..... regional state
τ ..... task
Γ ..... set of feasible tasks
$\overrightarrow{x}$ ..... ordered list of regions
$\xi_{\overrightarrow{x}}$ ..... Repeated region list
$\phi_t(x_i, x_j)$ ..... time abstraction function
ϵ ..... sub-environment
ϵ_k ..... partial sub-environment
N_ϵ ..... size of a sub-environment
$\overrightarrow{\tau}$ ..... ordered list of tasks
$\overrightarrow{Pr^t}$ ..... ordered list of probability distributions
ϑ ..... task trajectory
$\mathcal{I}_\alpha$ ..... interaction set of agent α
$\mathcal{N}$ ..... max size of interaction set
θ ..... claimed regional objective
Θ_α ..... claimed objective set of agent α
$\Theta^{\mathcal{A}}$ ..... global claimed objective set
ϑ_β^p ..... partial trajectory
Ξ ..... interaction matrix
Q ..... expected discounted reward
N ..... number of simulations in $\epsilon.S$
$N_{\mathcal{O}_b}$ ..... number of simulations of an objective in $\epsilon.S$
$Q_{\mathbb{E}}^k$ ..... estimate of Q when selecting sub-environment
$N_{\mathbb{E}}^k$ .number of simulations of sub-environment of length k
t ..... time
T ..... max polling time
$\mathbb{W}_\epsilon$ ..... set of searched feasible sub-environments
$\mathbb{W}_{\epsilon_k}$ ..... set of searched feasible partial sub-environments

TABLE I
LIST OF SYMBOLS