Predictive Safety Network for Resource-constrained Multi-agent Systems

Meng Guo, Mathias Bürger
Bosch Center for Artificial Intelligence (BCAI), Renningen, Germany
Meng.Guo2, Mathias.Buerger@de.bosch.com

Abstract: Coordinating multiple agents, such as mobile robots, with shared resources, such as common battery charging stations, is a highly relevant but still challenging decision problem. Traditionally, the motion and task planning of multi-agent systems are tackled by either designing ad-hoc decision rules or employing optimization tools. The former requires intensive manual tuning while the latter needs a static and accurate model of the complete system. Both approaches are prone to uncertainties in the robot motion and task execution. In this work, we propose a novel planning framework based on recent advances in deep reinforcement learning. The framework combines a centralized safety policy that acts on direct predictions of future resource levels and a decentralized task policy that optimizes task completions. The safety network is trained using supervised learning without extraneous supervision, while the task policy is trained using concurrent self-play. The whole framework follows a hierarchical structure to avoid the exponential blowup in the state and action space. We demonstrate significant improvements in a practical logistic planning problem for warehouse robots, compared with heuristic solutions, optimization tools and other reinforcement learning methods.

Keywords: Deep Reinforcement Learning, Multi-agent Systems, Motion and Task Planning, Resource Constraints.

1 Introduction

A multi-agent system can be extremely effective for distributing heavy work load across multiple agents. However, multi-agent systems are often more than a collection of individual agents, e.g., some agents are more suitable for certain tasks than others. Proper coordination and collaboration can improve the overall performance significantly. Furthermore, these agents can also be competitive due to the common resources (e.g., charging stations) that are essential for the agent's operation safety (e.g., battery level). Such hard constraints impose direct dependencies among the agents and are particularly difficult to handle. Most multi-agent planning problems even under discrete and deterministic settings are well-known to be NP-hard and suffer from the exponentially-increasing state/action space along with the number of agents, see [1]. Such problems have been mostly tackled by designing heuristic rules and manual tuning. For specific domains such as vehicle routing in [2] and job assignments in [3], there are optimization tools which however require accurate and complete models. Such tools are often designed to plan for a static set of tasks with fixed duration/cost, thus not suitable for mobile robotic tasks that are prone to motion uncertainties. In recent years, machine learning, particularly deep reinforcement learning (DRL), has shown great potentials in handling this type of complexity and uncertainty in various domains, e.g., Atari games in [4] and robot visuomotor control in [5]. However, the application of DRL to relevant multi-agent planning problems is still difficult and often prone to failures as also shown in [6].

The proposed motion and task planning framework consists of two main components: a centralized safety network and decentralized task networks. (I) The safety network oversees the whole system and the current resource level, and predicts the resource level at future times for the chosen action. It also provides a safety policy which the system can follow in order to ensure the resource level remains in the safe set. It is built on the network structure proposed in [7] and trained using supervised learning without extraneous supervision. (II) The task network takes as inputs the observations from

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

each agent's local perspective and outputs the best local action for task completion. Such networks are trained concurrently via parallel self-plays, i.e., the agents are controlled by different seeds of the task policies in parallel simulations of different settings. The complete framework follows a hierarchical structure from [8] to avoid the exponential blowup in the state and action space.

The main contribution of this work is twofold: first, the proposed safety policy acts on direct prediction of the future safety-related features, which is much more intuitive for analyzing safety properties (compared with the traditionally used Q-values); second, learning of the task policy is greatly accelerated by enforcing the safety check from the learned safety network, while in contrary learning both objectives simultaneously often fails due to the difficult manual process of reward shaping. We demonstrate significant improvements in performance over heuristic rules, optimization tools and other RL methods via a logistic planning problem with a fleet of warehouse robots under various operation conditions, where particularly standard RL methods fail to solve the problem at all.

2 Related Work

In the most straightforward form, a multi-agent system can be modeled as a general Markov decision process (MDP), where the system transition is driven by the joint action of all agents and the reward is the collective reward of all agents. However such centralized view quickly becomes impractical as the state and action space grows exponentially with the number of agents. Thus local optimization techniques have been proposed in combination with inter-agent communication to solve the global problem, such as Q-value approximation for partially-observable MDPs (POMDPs) in [9], belief nesting for interactive POMDPs in [10] and factored MDPs in [11, 12].

On the other hand, machine learning approaches, particularly reinforcement learning, have been investigated for multi-agent systems see, e.g., [13, 14, 15, 16]. However, as pointed out in [6, 13], multi-agent domains are particularly difficult for traditional RL methods such as Q-learning by [4] and policy gradient by [17] because (a) the exponential problem size as mentioned earlier; and (b) non-stationary environments due to actions of other agents. Many recent works adapt the traditional methods by incorporating other agents' policies into the decision making, e.g., other agents' actions and payoffs in [16], explicit parameters of other agent's policy in [15], or policy gradients in [6, 13]. Such information exchange can be modeled either explicitly in [18] or implicitly as in [6] for competitive games. However none of the above methods have shown their advantages within practical planning domains as considered in this paper, where these assumptions are hard to verify.

Furthermore, another important technique to address both the exponential complexity and the non-stationary environment is by imposing hierarchical structures on the planning algorithm see [8, 19, 20]. It has shown great potential for domains that involve long-term reasoning but under spare and delayed rewards. For instance, intrinsic motivation is learned in [8, 20] via generating intermediate and simpler goals, while in [21] the complex task is decomposed into smaller tasks via splitting explicitly the reward function. We build on these results by separating the high-level task coordination and resource allocation with the low-level navigation and task execution.

Safety is an important yet difficult aspect in learning as to avoid a set of unsafe states. It has been traditionally handled via assigning a large penalty at unsafe states in [21], which often requires intensive reward shaping and manual definition of unsafe states. A different approach is to *separate* safety policy with task policy as shown in [22, 23]. However, the parameters in the condition to switch from task policy to safety policy in these approaches need to be manually set. The work in [24] and [25] propose similar safety predictor but only for a single agent with safety defined over discrete/sparse variables. Instead in this work, we learn the safety policy via a predictive network that also provides *direct* safety-related features. We build on the idea of predictive network from [7] that replaces sparse scalar reward signal by dense multi-dimensional measurement stream.

3 Model Description

Consider a team of N agents where each agent $n \in \mathcal{N} \triangleq \{1, \cdots, N\}$ is modeled by a conditioned Markov decision process (MDP), denoted by $\mathcal{M}_n = \langle S_n, A_n, C_{\text{-}n}, R_n, T_n \rangle$, where S_n is the state space, A_n is the action space, $C_{\text{-}n} = \langle S_{\text{-}n}, A_{\text{-}n} \rangle$ is the set of state-action pairs of other agents, $R_n : S_n \times A_n \times C_{\text{-}n} \to \mathbb{R}$ is the reward function, and $T_n : S_n \times A_n \times C_{\text{-}n} \times S_n \to [0, 1]$ is the transition probability. Note that both the reward function and the transition function are

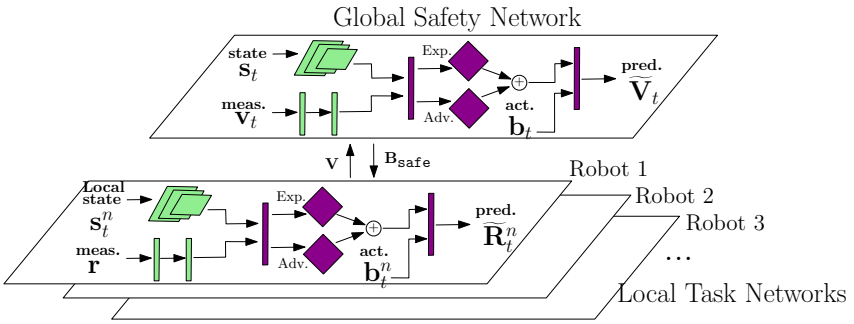

Figure 1: Overall architecture with predictive safety network and local task networks.

conditioned on other agents' states and actions. Both functions might be unknown and change over time. The entire system evolves as an infinite sequence of state-action pairs $\boldsymbol{\tau} = \mathbf{s}_0\mathbf{a}_0\mathbf{s}_1\mathbf{a}_1\cdots$, where $\mathbf{s}_t = (s_0, \cdots, s_N)$ and $\mathbf{a}_t = (a_0, \cdots, a_N)$ are the joint state and action of all agents at time $t \geqslant 0$. The reward obtained by agent n at time t is denoted by $r_{n,t}$. For the ease of notation, we denote by $\mathbf{S} = \times_{n\in\mathcal{N}} S_n$ and $\mathbf{A} = \times_{n\in\mathcal{N}} A_n$. Furthermore, the state variable s_n of each agent $n \in \mathcal{N}$ contains a resource variable $v_n \in \mathbb{R}$, for which the joint resource is denoted by $\mathbf{v} = (v_0, \cdots, v_N)$. The system-wide resource constraint we consider is imposed by

$$\mathbf{v}_t \in \mathcal{V}_{\mathtt{safe}},\ \forall t \geqslant 0, \tag{1}$$

where $\mathcal{V}_{\mathtt{safe}} \subseteq \mathbb{R}^N$ is a static *safety* set of resources, and $\mathbf{v}_t$ is the resource vector at time $t \geqslant 0$.

Our goal is twofold: to synthesize **(a)** a centralized safety policy $\boldsymbol{\pi}_{\mathtt{s}} : \mathbf{S} \times \mathbf{A} \to [0, 1]$ such that the safety constraint above holds if $v_0 \in \mathcal{V}_{\mathtt{safe}}$; and **(b)** a decentralized local task policy $\boldsymbol{\pi}_{\mathtt{p},n} : S_n \times C_{-n} \times A_n \to [0, 1]$ such that the expected discounted total reward $\mathbb{E}_{\pi_{\mathtt{p}},\mathbf{s}_0}\{\sum_n \sum_t \gamma^t r_{n,t}\}$ is maximized, with $0 < \gamma < 1$.

Motivating Example: Consider a fleet of autonomous forklifts that are deployed to complete a continuous stream of transporting tasks, e.g., to load products from one shelf and unload them to another. These robots are battery-powered and thus need to "stay safe" by maintaining a minimum battery level. The whole fleet shares a number of charging stations within the warehouse. The planning objective is to maximize the task completion while keeping all robots safe.

4 Proposed Solution

In this section, we describe all components of the proposed solution as depicted Fig. 1, including the behavior composition, the safety network and finally the task network.

4.1 Hierarchical Composition of Action Primitives

Instead of planning directly on the primitive actions or raw actuation inputs, we follow similar ideas as proposed in [8, 19], we manually define high-level *meta-actions* as agent behaviors B_n. Some behaviors are relevant to resource management (denoted by $B_{\mathtt{safe},n}$) and others to task completion (denoted by $B_{\mathtt{task},n}$), $\forall n \in \mathcal{N}$. The set of all agent behaviors is given by $B = \cup_n\{B_n\}$. Certainly, the design of these behaviors would require domain knowledge. To generate high-level behaviors autonomously as partially discussed in [8] is out of the scope of this paper. For the example above, behaviors relevant to pick-and-drop task completion could be "navigation to pick/drop location", "pick" and "drop". At the same time, to maintain battery level, relevant behaviors could be "navigation to the charge station", "wait to charge" and "charge". Then an execution policy is constructed for each behavior via preferred methods, e.g., via RL [5, 26], or classic control techniques [27, 28]. The focus of this work is not the derivation of such primitive policies but the coordination among them.

4.2 Task-independent Predictive Safety Network

Given the behaviors described above, we construct and train a predictive safety network specifically for the resource constraint in (1). The safety network is generic as it is trained *independently* of the task specifications. Below we describe the safety policy and training method for the network in detail.

4.2.1 Minimum-intervention Safety Policy

The network denoted by $\mathcal{N}_{\boldsymbol{\theta}}$ with parameter $\boldsymbol{\theta}$ takes as inputs the system state $\mathbf{s}_t$ and the resource vector $\mathbf{v}_t$ at time $t \geqslant 0$. It outputs the predicted resource vectors at the chosen future time instants:

$$\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{v}_t) \triangleq (\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}_0}, \widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}_1}, \cdots, \widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}_M}) \tag{2}$$

where $\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}_m}(\mathbf{s}_t, \mathbf{v}_t) = (\widetilde{\mathbf{v}}_{t+\tau_0}, \widetilde{\mathbf{v}}_{t+\tau_1}, \cdots, \widetilde{\mathbf{v}}_{t+\tau_K})$ are the predicted resource vectors at the chosen time shifts $\{\tau_0, \cdots, \tau_K\}$, and $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_M$ are the allowed behaviors at time t. Moreover, the safety policy chooses the current behavior $\mathbf{b}_t$ based on the solution to the following optimization:

$$\begin{aligned} \min_{\{\mathbf{b} \in \mathbf{B}\}} \quad & \Big\{ \sum_n \mathbb{1}_{b_n \in B_{\texttt{safe},n}} \Big\} \\ \textbf{s.t.} \quad & d_{\min}\big(\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}}(\mathbf{s}_t, \mathbf{v}_t),\, \mathcal{V}_{\texttt{unsafe}}\big) > \delta_{\texttt{safe}}, \end{aligned} \tag{3}$$

where the objective is to *minimize* the number of behaviors activated for safety (i.e., belong to $B_{\texttt{safe},n}$), and the constraint is to ensure the minimum Euclidean distance between the predicted future resources $\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}}$ from (2) and the unsafe set $\mathcal{V}_{\texttt{unsafe}}$ is above the a manually-chosen robustness margin $\delta_{\texttt{safe}} > 0$, where $\mathcal{V}_{\texttt{unsafe}} \triangleq \mathbb{R}^N \backslash \mathcal{V}_{\texttt{safe}}$. We define the objective above as a measure of "intervention" and "conservativeness" for the learned safety policy. Smaller intervention means that the safety policy allows more freedom in executing task-related behaviors and thus less conservative. If (3) does not have a solution, then $\mathbf{b}_t$ is chosen among the "safest" behaviors, i.e., the behavior with the maximum intervention. If multiple solutions exist, $\mathbf{b}_t$ is chosen randomly among them.

As depicted in Fig. 1, in our implementation, the state $\mathbf{s}_t$ is represented as an image and thus is processed first by layers of convolutional neural networks (CNNs), while the resource $\mathbf{v}_t$ is followed by a fully-connected layer. These two outputs are then concatenated as the hidden input representation. As proposed in [29], to differentiate more different actions, this hidden input is split into the expectation and advantage branches, which are then summed into the actual predictions. The actual prediction for a chosen behavior $\mathbf{b}_t$ is then simply derived by the one-hot encoding.

4.2.2 Task-independent Safety Training

The safety network is trained using supervised learning without extraneous supervision. In particular, the system starts from initial state $\mathbf{s}_0$ and resource $\mathbf{v}_0$. At step t, the safety network outputs $\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{v}_t)$ by (2). Note that to make the action set $\mathbf{b}_t$ invariant to various task specifications, we replace all task-related behaviors introduced in Sec. 4.1 by a generic behavior "perform task", the cost and duration of which is randomly sampled from the original set of action primitives. The acting behavior $\mathbf{b}_t$ is chosen based on the policy (3) above. Then the system evolves to state $\mathbf{s}_{t+1}$ and resource $\mathbf{v}_{t+1}$. This procedure is repeated until a fixed number (sufficiently larger than the prediction horizon τ_K) of steps or the resource vector becomes unsafe. The set of experience is denoted by $\mathcal{D}_{\texttt{safe}} = \{\langle \mathbf{s}_t, \mathbf{v}_t, \mathbf{b}_t, \mathbf{V}_t \rangle, \forall t\}$, where $\mathbf{V}_t$ are the *recorded* resources at the chosen future instances. Then the network parameter $\boldsymbol{\theta}$ is trained via minimizing the regression loss:

$$L(\boldsymbol{\theta}) = \|\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}^{\mathbf{b}_t}(\mathbf{s}_t, \mathbf{v}_t) - \mathbf{V}_t\|_2, \tag{4}$$

where $\langle \mathbf{s}_t, \mathbf{v}_t, \mathbf{b}_t, \mathbf{V}_t \rangle$ are sampled from $\mathcal{D}_{\texttt{safe}}$. More experiences are collected after more episodes are simulated. Then mini-batches can be drawn and used to update the network periodically. Denote by $\mathcal{N}_{\boldsymbol{\theta}^\star}$ the trained safety network with parameter $\boldsymbol{\theta}^\star$. It is worth pointing out that the prediction from the safety network is only valid if the associated safety policy in Eq. (3) is followed.

4.2.3 Safety Guarantee

Once the training loss has converged to a reasonable value, the safety guarantee can be inferred from two aspects: First, if all episodes in the end of training reach the maximum length, it means that the system under safety policy remains safe for at least the episode length, which is required to be sufficiently larger than the prediction horizon τ_K. Second, if the training loss $L(\boldsymbol{\theta})$ in (4) is sufficiently small, it means the the predicted resource vector is sufficiently close to the actual vector. Moreover, as the constraint in (3) is satisfied always, it means that the distance between the predicted resource vector is far away from the unsafe set, implying that the actual resource vector is also far away from the unsafe set. More precisely, it can be shown that if the final loss with batch-size $\overline{B}$ satisfies $L(\boldsymbol{\theta}) < \overline{B} \times \delta_{\texttt{safe}}$, the safety policy from (3) is ensured to be safe.

Algorithm 1: Predictive Safety Network for Resource-constrained Multi-agent Systems

Input: System simulator for the whole system $\{\mathcal{M}_n\}$.
Output: Learned safety network $\mathcal{N}_{\boldsymbol{\theta}^\star}$ and task network $\mathcal{N}_{\boldsymbol{\beta}^\star}$.

Construct behaviors $\{B_n\}$ and the associated polices from primitive actions.
Collect experience $\mathcal{D}_{\texttt{safe}} = \{\langle \mathbf{s}_t, \mathbf{v}_t, \mathbf{b}_t, \mathbf{V}_t \rangle\}$ under the safety policy in (3).
Train the estimator $\widetilde{\mathbf{V}}_{\boldsymbol{\theta}}$, yielding $\mathcal{N}_{\boldsymbol{\theta}^\star}$. // Sec. 4.2
Collect experience $\mathcal{D}_{\texttt{task}} = \{\langle \mathbf{s}_t, \mathbf{r}_t^n, \mathbf{b}_t^n, \mathbf{R}_t^n \rangle\}$ under the task policy in (5) and the learned $\mathcal{N}_{\boldsymbol{\theta}^\star}$.
Train the estimator $\widetilde{\mathbf{R}}_{\boldsymbol{\beta}}$, yielding $\mathcal{N}_{\boldsymbol{\beta}^\star}$. // Sec. 4.3
for *each step* t **do**
    Observe the current state $(\mathbf{s}_t,\ \mathbf{v}_t,\ \mathbf{r}_t)$; choose behaviors $\{\mathbf{b}^n\}$ via (5) and the learned $\mathcal{N}_{\boldsymbol{\beta}^\star}$.

4.3 Safe-guarded Task Network

Beside maintaining a safe resource level, the multi-agent systems are often assigned tasks to accomplish, e.g., surveillance [30] and collaboration [31]. In this section, we describe the task planning module that assigns these tasks to individual agents in an efficient manner such that more tasks can be accomplished per unit time, under the safety constraints mentioned above.

4.3.1 Task Policy and Safe-guarded Training

Instead of a predefined set of static tasks, we consider a continuous stream of tasks that are assigned to the whole system, e.g., according to online orders. A single task can be any sequence of the task-relevant agent behaviors $B_{\texttt{task},n}$ as introduced in Sec. 4.1. Such sequence can be either directly specified by the user or derived by external planning processes, see [32, 33].

The task network follows a variation of the predictive network from Sec. 4.2.1, denoted by $\mathcal{N}_{\boldsymbol{\beta}}$ with parameter $\boldsymbol{\beta}$. As shown in Fig. 1, even though the task network is identical for all agents, it is applied locally to each agent with only the observable part of the global state $\mathbf{s}_t$ and the reward vector $\mathbf{r}_t$ as inputs and it outputs the predicted reward vectors at future times, denoted by $\widetilde{\mathbf{R}}_{\boldsymbol{\beta}}(\mathbf{s}_t^n,\ \mathbf{r}_t) = (\widetilde{\mathbf{R}}^{\mathbf{b}_0}, \widetilde{\mathbf{R}}^{\mathbf{b}_1}, \cdots, \widetilde{\mathbf{R}}^{\mathbf{b}_M})$, defined similarly as in (2). Note that the action set for task networks only contains the task behaviors. The time shifts here can be different from the the ones for safety network. The task policy of each agent is guarded by the learned safety network as follows:

$$\begin{aligned} \max_{\{\mathbf{b}^n \in \mathbf{B}^n\}} \quad & \{\sum_n \widetilde{\mathbf{R}}^{\mathbf{b}^n}_{\boldsymbol{\beta},t}\} \\ \textbf{s.t.} \quad & d_{\min}\big(\widetilde{\mathbf{V}}^{\mathbf{b}}_{\boldsymbol{\theta}^\star}(\mathbf{s}_t, \mathbf{v}_t),\ \mathcal{V}_{\texttt{unsafe}}\big) > \delta_{\texttt{safe}}, \end{aligned} \tag{5}$$

where $\mathbf{b}^n$ is the local behavior of agent n in the task network, $\mathbf{b}$ is the joint behavior in the safety network, $\widetilde{\mathbf{V}}^{\mathbf{b}}_{\boldsymbol{\theta}^\star}(\mathbf{s}_t, \mathbf{v}_t)$ contains the predicted resources by the learned safety network, and the measure of safety distance is defined in (3). The above constraint ensures that *only* the behaviors that are predicted to be safe by the safety network can be chosen for maximizing the collective rewards by the task network. Thus, training of the task policy is safe-guarded by the learned safety network.

We use the idea of concurrent self-play from [34] to train the task network. Namely, agents are controlled independently in parallel simulations via different copies of the same task network. During training, the local behaviors are chosen for each agent via (5), the whole system evolves accordingly. Experiences for the task networks are gathered locally from the agents' local perspectives. Then these experiences are used to train the task networks via minimizing the prediction error, similar to (4).

4.4 Overall Framework and Discussion

The overall framework is summarized in Algorithm 1. During the training of safety network, a ε-greedy policy should be used to allow the exploration of safety-uncertain behaviors; initial states and resources should be randomized to cover larger part of the state space, especially the unsafe part; the prediction horizons are critical hyper parameters to be tuned: too long shifts would introduce large variances in the prediction while too short shifts would fail to facilitate long-time reasoning.

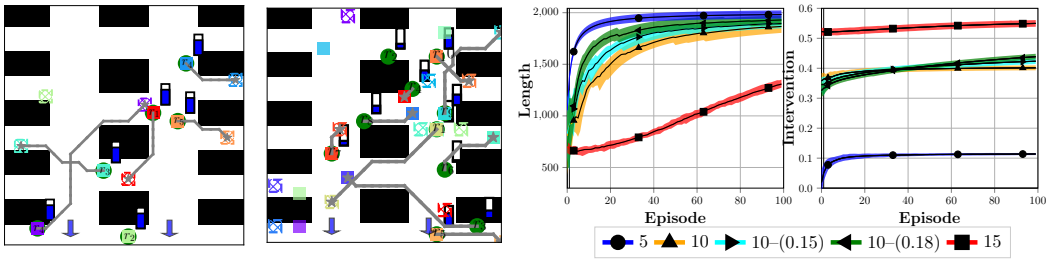

Figure 2: Left two: Snapshots of simulated systems of 5 (left) and 10 (right) robots. Robots are denoted by green circles with battery level indicated by blue bars. Charging stations are marked by blue arrows and shelves are in black. The loading and unloading location of a transportation task are marked in filled and checked squares. Right two: Evolution of episode length (left) and degree of intervention (right) of the safety network for system with 5, 10, 15 robots.

5 Experiments

In this section, we evaluate the proposed solution in a practical indoor logistic systems, as described by the motivating example earlier. All simulation results are obtained on a laptop with 8-core Intel Xeon CPU, using Python and Tensorflow. Implementation details and simulation videos are attached as supplementary files.

5.1 Model Description

As shown in Fig. 2, the warehouse is partitioned into rectangular cells which can be occupied by shelves, robots or charging stations. We consider a fleet of 5, 8, 10, 15 robots with 2 charging stations. The robots are controlled via primitive discrete actions: move into adjacent cells by action "wait" (-0.002), "up" (-0.01), "down" (-0.01), "left" (-0.01), "right" (-0.01); "load" (-0.02) and "unload" (-0.02) objects; and "charge" (0.02) at charging stations. The maximum battery level is set to 1.0 and the battery decrease/increase of performing each action above is given in the respective parentheses. Each action takes one simulation step in terms of duration. We add uncertainties to the model by assuming (a) the robot can drift side-ways with 5% probability when moving to an adjacent cell, and (b) the battery consumption of each action varies by a standard deviation of 10%. Furthermore, the stream of tasks is generated online with randomly-chosen loading and unloading locations, of which the density can be changed by the interval between when new tasks are generated.

The above primitive actions are first composed into high-level behaviors. Particularly, **(a)** "if any charge station is available, go and charge there until full. Otherwise, wait at the current location" is a resource-related behavior when a robot tries to charge. Based on it, we construct a set of $N+1$ resource-related behaviors: $B_{\texttt{safe}} = \{$"n robots should try to charge", where $n = 0, 1, \cdots, N\}$. and **(b)** "navigate to load the object, and then navigate to unload it" is the task-related behavior when a robot executes a plan for a transportation task. Based on it, we construct the set of task-related behaviors: $B_{\texttt{task}} = \{$"robot takes the n_{th} closest task", where $n = 1, 2, 3\}$. All behaviors above are not preemptable, i.e., once a robot continues executing a behavior until it is finished. Note that $B_{\texttt{safe}}$ grows *linearly* with the number of robots and $B_{\texttt{task}}$ is local to each robot with *constant* size.

As input to the safety network, the system state is encoded by an image with 4 channels (robot ids, charging station, shelves, robot battery levels), and the resource vector is the stack of battery level of all robots and additionally the remaining changing level for each charging station. Thus the resource vector has dimension $N+2$. The prediction horizon is set to $[5, 10, 20, 40, 80]$, while the set of actions $B_{\texttt{safe}}$ defined above has size $N+1$. The expectation and advantage branches are passed through individual fully connected layer of size $N_{act} \times N_{\boldsymbol{\tau}} \times \dim(\mathbf{v}_t) = \mathcal{O}(N^2)$. We use 5 workers to gather experience in parallel simulations for the master network and the batch size is set to 500. On the other hand, input to the task network is encoded with 4 channels (the robot under consideration, other robots, task distributions, shelves). The prediction horizon is set to $[2, 4, 10, 20]$ for all robots. The local reward vector as input has dimension N as the accumulated rewards. The set of actions is defined above by $B_{\texttt{task}}$ with size 3. During the concurrent self-play, each robot is controlled by one randomly-chosen worker and the gathered local experiences is saved in the same buffer. The batch size now is set to 300. More implementation details are given in the supplementary file.

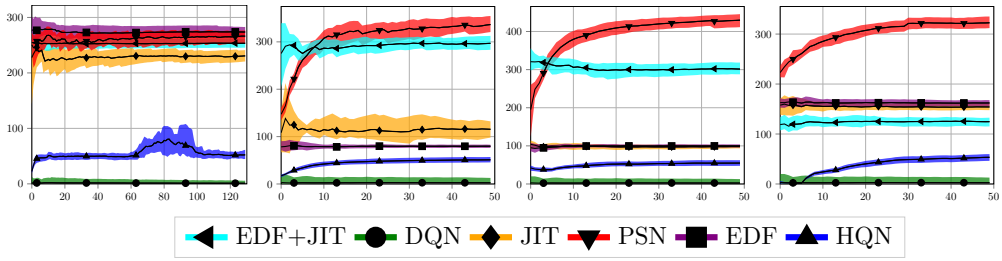

Figure 3: For all subplots, **X**-axis: number of training episodes for task networks, and **Y**-axis: number of accomplished tasks, under different system sizes: 5, 8, 10 and 15 robots (left to right).

5.2 Training Results

The training results of safety networks for system with $5, 10, 15$ robots are illustrated in Fig. 2. To have a reasonable start, we always bootstrap the safety policy with the base policy that any robot should be either charging or waiting to charge if its battery is less than $N/(10 \cdot N_c) - 0.1$. The maximum episode length is set to 2000, which is significantly longer than the prediction horizon. In general, it takes longer for the safety policy to converge with larger number of agents, such that all robots are safe throughout the episode. Specifically, the training of safety networks took around $0.5, 2, 5$ hours for system with $5, 10, 15$ robots, respectively. On the other hand, it also shows that the degree of intervention converges without further increase during training due to the optimization objective in (3). Furthermore, we change the safety margin $\delta_{\texttt{safe}}$ in the constraint of (3) from 0.1 to 0.15 and 0.18 for the system of 10 robots. The learning curves for these cases are also shown in Fig. 2, denoted by $10 - (0.15)$ and $10 - (0.18)$. It shows that with larger safety margin, the safety policy converges faster and the system reaches the maximum length earlier, which however leads to an increased degree of intervention. This illustrates well the trade-off between accelerating the learning of the safety policy and reducing its conservativeness.

The learned safety networks above are used during the training the task networks, as described in Sec. 4.3.1. Due to the safety constraint enforced by the safety network, *almost all* episodes can reach the maximum length, which greatly increase the amount of relevant experiences that can be gathered for a certain amount of episodes. As shown in Fig. 3, our approach denoted by the predictive safety network (PSN) converges quite fast under system sizes of $5, 8, 10, 15$ robots with great performances. Note that we do not modify the warehouse layout such as the location of shelves and charging stations during training or testing of both safety and task networks.

5.3 Alternative Methods

For the purpose of benchmarking, we implemented alternative planning methods to our approach (PSN) for the considered problem: two state-of-the-art deep RL methods for discrete actions: DQN [4] and HQN [8], the widely-used Google OR tools GOR for combinatorial optimization [3] , and three heuristic planners: **(a)** Earliest deadline first (EDF). The robot with the least battery will charge whenever a charging station is available. **(b)** Just in time (JIT). Any robot with battery less than 0.3 will try to charge, i.e., either go to charge or wait to charge. **(c)** EDF+JIT. It combines the above two heuristics and the battery lower-bound is tuned such that all robots remain safe within the episode.

The DQN follows the vanilla implementation [35] with double Q network and multiple workers gathering experience. The HQN uses the same set of high-level behaviors as defined above but learns the safety policy and the task policy *simultaneously*. Regarding the GOR tools, we use particularly the Constrained Programming solver (CP-SAT), which is shown to be much faster than classic mixed integer programming (MIP). Different from the classic *Job Shop* problem, we need to optimize both the start and duration of each agent behavior while satisfying the limited charging stations. To mitigate motion uncertainties and the continuous stream of tasks, we adopt a receding horizon approach [36] that replans periodically, i.e., an optimal plan including the sequence and duration of the behaviors is computed for each robot within a chosen time horizon, which is then updated periodically at a chosen frequency. Lastly, these three heuristic rules defined above are relatively easy to implement and invariant to the team size.

Methods	EDF	JIT	EDF+JIT	DQN	HQN	PSN	GOR
(**5**)Task	**273**	229	253	5	50	252	213
(**5**)Length	**1000**	917	**1000**	101	128	**1000**	**1000**
(**10**)Task	98	97	305	6	58	**453**	65
(**10**)Length	146	228	970	100	120	**1000**	950
(**15**)Task	161	153	124	4	65	**382**	47
(**15**)Length	142	226	968	100	117	**992**	720

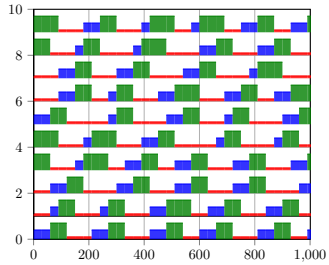

Figure 4: Left table: task performance and length of different methods during test time. Right figure: **X**-axis: time steps, and **Y**-axis: robot IDs, for the 10-robot system under the proposed PSN. Robot behaviors are "wait to charge" (red), "go to charge/charge" (blue) and "task completion" (green).

5.4 Comparison in Performance

Performance is defined as the number of tasks accomplished within one episode. The performance of the above methods are evaluated under the following scenarios of increasing difficulty: a fleet of $5, 8, 10, 15$ robots and always 2 charging stations. During training, each episode lasts for 1000 steps and is terminated once any robot is out of battery. The performance results during training are summarized in Fig. 3 and during test time in Fig. 4. As expected, the DQN method performs extremely poor for all scenarios due to the exponential size in the action space, e.g., for 10 robots, it has 280 Million possible actions and size of network reaches $2GB$. The HQN method struggles to find the balance between accomplishing more tasks and staying safe, *particularly* due to the delayed penalty of "not charging" and in contrast the instant reward of accomplishing a task.

Regarding GOR, for all system sizes, we set the planing horizon to 300 time steps and frequency to 30 steps. The solution time increases significantly: 0.2 seconds for 5 robots, 3.5 hours for 10 robots and more than 20 hours for 15 robots. We limit the planning time to 5 minutes for all cases during test. It can be seen that for 5 robots the performances between GOR and PSN are close, but GOR falls behind when the planning time is limited that only sub-optimal or even *infeasible* plans are found for systems with $10, 15$ robots. The feasibility of an updated plan is not ensured due to the close dependency between behaviors within and outside the planning horizon. Similarly, these heuristic rules perform very well in the 5-robot case as the robots can simply charge whenever they want, i.e., 2 robots per charging station. However, for the cases with 10 or 15 robots, both EDF and JIT fail to keep all robots safe while the performance of EDF+JIT drops dramatically as the lower threshold has to be set very high to keep all robots safe, i.e., 0.6 for 10 robots and 0.9 for 15 robots. One example of the actual high-level schedule for the 10-robot case is shown in Fig. 4. It can be seen that under the PSN method, the robots anticipate well beforehand to charge or wait for charge, which validates the advantage of future predictions from the safety network.

5.5 Generalization to Different Tasks

Lastly, we show that the *same* learned safety network can be applied to the training of different tasks, e.g., different distribution density: sparse, normal and dense, where task interval is set to $16, 8, 4$ time steps. Table 1 shows the results during test: the performance improves accordingly (e.g., almost doubled) when the density is doubled from 'sparse' to 'normal'. However, the performance lags behind from 'normal' to 'dense' because the system is close to the maximum number of tasks it can handle.

Table 1: Performance under different task specifications during test time.

Task Density	**5**-robot	**10**-robot
Sparse-**16**	147 (0.58)	246 (0.55)
Normal-**8**	252 (1.0)	453 (1.0)
Dense-**4**	290 (1.15)	510 (1.13)

6 Conlusion and Future Work

In this work, we present the planning framework for resource-constrained multi-agent systems, which uses a predictive safety network to ensure the resource constraints and task networks to maximize the performance. Future work involves generalization across workspaces and dependent tasks.

References

[1] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[2] W. Cook. Concorde tsp solver. http://www.math.uwaterloo.ca/tsp/concorde/, 2015.

[3] Google. Google or-tools. https://developers.google.com/optimization/, 2018.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[6] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[7] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations (ICLR)*, 2017.

[8] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[9] F. A. Oliehoek, M. T. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[10] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[11] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530, 2002.

[12] C. Guestrin, S. Venkataraman, and D. Koller. Context-specific multiagent coordination and planning with factored mdps. In *AAAI/IAAI*, pages 253–259, 2002.

[13] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[14] S. Sukhbaatar, R. Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[15] G. Tesauro. Extending q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems*, pages 871–878, 2004.

[16] M. L. Littman. Friend-or-foe q-learning in general-sum games. In *ICML*, volume 1, pages 322–328, 2001.

[17] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[18] I. Mordatch and P. Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[19] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

[20] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[21] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5392–5402, 2017.

[22] T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.

[23] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*, 2017.

[24] G. Kahn, A. Villaflor, P. Abbeel, and S. Levine. Composable action-conditioned predictors: Flexible off-policy learning for robot navigation. *arXiv preprint arXiv:1810.07167*, 2018.

[25] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[27] J. K. Salisbury and J. J. Craig. Articulated hands: Force control and kinematic issues. *The International journal of Robotics research*, 1(1):4–17, 1982.

[28] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart. Reciprocal collision avoidance for multiple car-like robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 360–366. IEEE, 2012.

[29] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[30] L. E. Parker. Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous robots*, 12(3):231–255, 2002.

[31] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[32] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[33] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354, 2017.

[35] D. Britz. Deep q-learning implementation. https://github.com/dennybritz/reinforcement-learning/tree/master/DQN, 2018.

[36] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.