Predictive Safety Network for Resource-constrained Multi-agent Systems — Supplementary File —

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

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

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

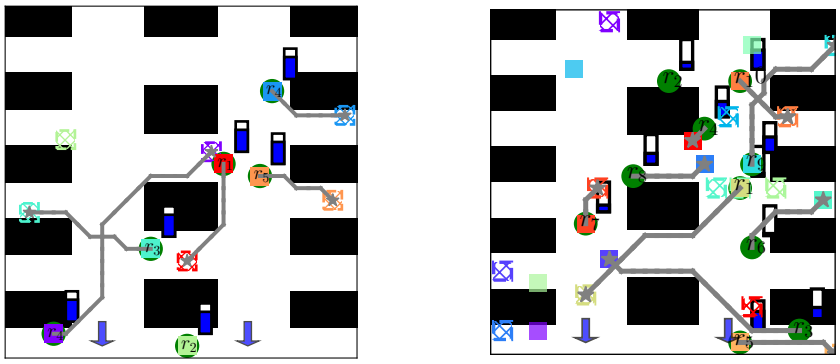

Figure 1: Snapshots of simulated systems of 5 (left) and 10 (right) robots. Robots are denoted by green circles with battery level indicated by blue bars. Charging stations are marked by blue arrows and shelves are in black. The loading and unloading location of a transportation task are marked in filled and checked squares.

1 Experiments

In this section, we evaluate the proposed solution in a practical planning problem of indoor logistic systems. It follows the motivating example referred earlier. In the sequel, we first present the detailed system model and then the setup of our safety and task networks. Then we compare the performance of our approach to other planning algorithms under different problem sizes and varying task difficulties. We also evaluate the ability of the learned safety network to generalize across different task specifications. All simulation results are obtained on a laptop with 8-core Intel Xeon CPU, using Python and Tensorflow. Simulation videos for different cases via different methods are attached as the supplementary file.

1.1 Model Description

As shown in Fig. 1, 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. The density of this stream can also be changed by the interval between when new tasks are generated. Note that inter-robot collision avoidance is not addressed here and assumed to be handled by a low-level reactive navigation scheme, see [1].

1.1.1 Implementation Details

As mentioned in Sec. 4.1, we first compose the above primitive actions into high-level behaviors. To begin with, the point-to-point navigation behavior is easily learned via DQN or finding stochastic shortest paths. Then we group and sequence this navigation behavior with manipulation actions into other behaviors: **(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 $n = 0, 1, \cdots, N\}$. For example, when five robots try to charge, the robot with the *least* amount of battery will be assigned first to the available charging station; 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 $n = 1, 2, 3\}$. Namely, the available tasks around the robot are ordered by their relative distances the robot. The robot then choose among the closest three tasks. Always choosing the closest task is not collectively optimal as sometimes the other robots are much

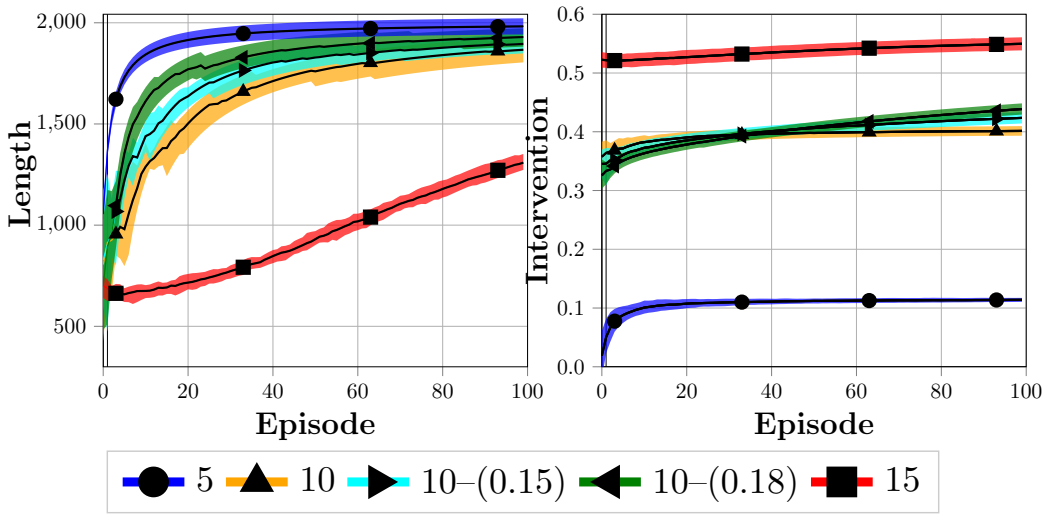

Figure 2: Evolution of episode length (left) and degree of intervention (right) along with the training episodes of the safety network for system with 5, 10, 15 robots and different safety margins.

closer. Note that $B_{\texttt{safe}}$ grows *linearly* with the number of robots and $B_{\texttt{task}}$ is local to each robot with *constant* size.

The general structure of the safety network is described in Sec. 4.2: the system state is encoded by an image with 4 channels (robots with ids, charging station, shelves, robots with 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 predication horizon is set to $[5, 10, 20, 40, 80]$ for the safety network. The safety margin is set to 0.1 in (3). The exploration rate ε is 0.2 initially and decays gradually with time. The state is processed by two convolution layers with 64 filters and a kernel size of 5. The resource vector is followed by a fully connected layer of size 512 and the concatenated hidden state is followed by a fully connected layer of size 1024. The set of actions is defined above by $B_{\texttt{safe}}$ with size $N + 1$. The expectation and advantage branches are passed through individual fully connected layer of size $N_{act} \times N_{\boldsymbol{\tau}} \times \dim(\mathbf{v}_t) = \mathcal{O}(N^2)$. The actual prediction for each robot is derived by summing these two branches. We use 5 workers to gather experience in parallel simulations for the master network. Then the master network is trained via mini-batches of size 500 drawn randomly from the experience buffer, which is then used to update the worker network periodically.

The local task network we use has a similar structure as described above. Since it is a local network for each robot, the system state 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. The CNN layers are the same as the safety network. The fully-connected layer that summarizes the expectation and advantage branches have now size $\mathcal{O}(N)$. We also use 5 workers to gather experiences for the master task network. During the current self-play, each robot is controlled by one randomly-chosen worker and the gathered local experiences is saved in the same buffer. Then the master network is trained via mini-batches of size 300 drew randomly from the buffer and the workers are updated periodically by copying the master network.

1.2 Results

1.2.1 Training of Safety Network

In this section, we present the training results of safety networks for system with $5, 10, 15$ robots, which is 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 $N/(10 \cdot N_c) - 0.1$. Given the maximum episode length 2000, 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

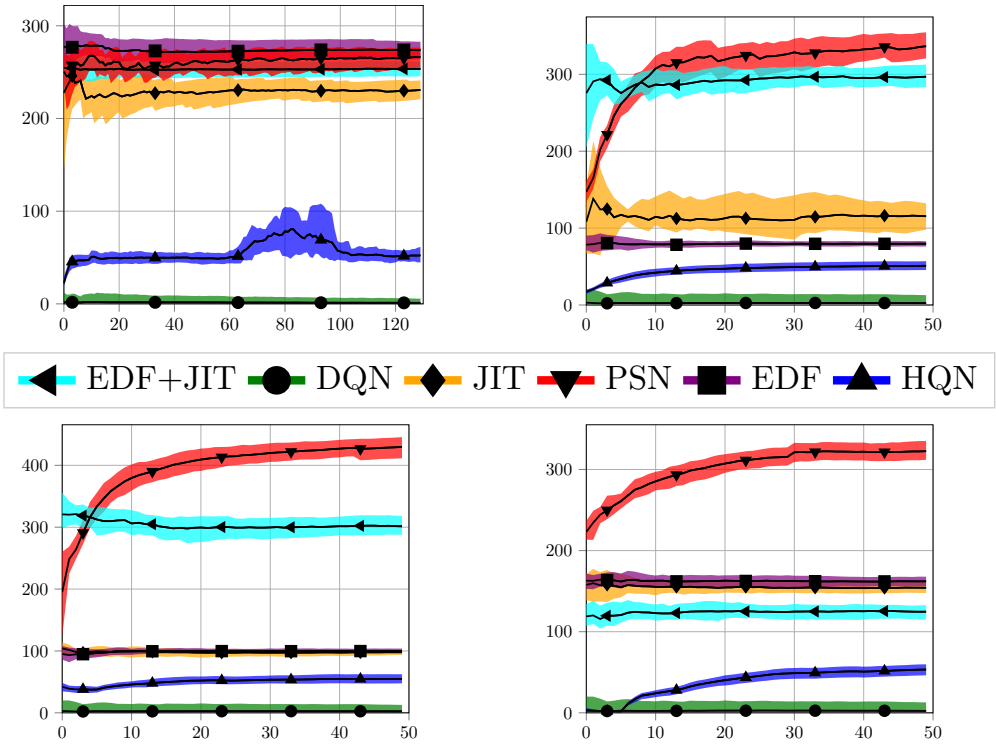

Figure 3: For all subplots, **X**-axis: number of training episodes for task networks, and **Y**-axis: performance as the number of accomplished tasks, under different system sizes: 5 (top left), 8 (top right), 10 (bottom left) and 15 (bottom right) robots.

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 Figure 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.

1.2.2 Training of Task Networks

The learned safety networks above are used during the training the task networks, as described in Sec. 4.3. 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 different 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 staions during training or testing of both safety and task networks.

1.2.3 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 [2] and HQN [3], the widely-used Google OR tools GOR [4] for combinatorial optimization, 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 [5] with double Q network and multiple workers gathering experience. The HQN uses the same set of high-level behaviors as defined above but learns

Table 1: Task performance and length of different methods during test time.

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

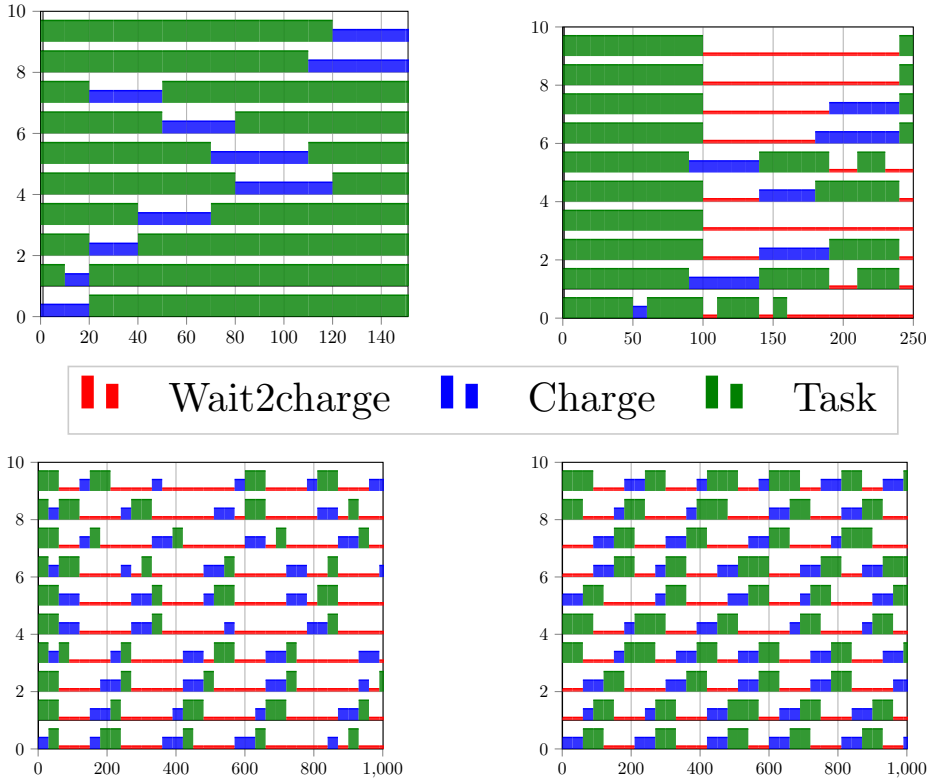

Figure 4: For all subplots, **X**-axis: time steps, and **Y**-axis: robot IDs, for the 10-robot system under the heuristic planners EDF (top left), JIT (top right), EDF+JIT (bottom left) and the proposed PSN (bottom right). Three high-level robot behaviors are "wait to charge", "go to charge/charge" and "task completion".

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). In order to avoid the exponential blowup, we model the scheduling problem on the same level as our approach PSN, i.e., to schedule the behaviors of "wait", "charge" and "performing task" for each agent. Different from the classic *Job Shop* problem, we need to optimize both the start and duration of each agent behavior while at the same time considering the limited charging stations. To mitigate motion uncertainties and the continuous stream of tasks, we adopt a receding horizon approach [6], 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.

1.2.4 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

Table 2: Performance under different task specifications

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)

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 Table 1. 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 $7^{10} \approx 282$ Million possible actions and size of network reaches $2GB$. The HQN method struggles to find the balance between accomplishing more tasks and staying safe, especially due to the very 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. Both parameters have shown great impact on the performance and feasibility of the resulting problem. The solution time to find the optimal solution 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 also due to the close dependency between any 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 seen very high to keep all robots safe, i.e., 0.6 for 10 robots and 0.9 for 15 robots. Lastly, it is interesting to see that when the shared resource is very limited, increasing the number of robots from 10 to 15 will not necessarily lead to performance improvement due to reduced time of task execution. For detailed comparisons, simulation videos for all three cases are attached as the supplementary file.

1.2.5 Comparison in Schedule

In Figure 4, we compare one example of the actual high-level schedule generated by these heuristic planners with our approach for the 10-robot case. It can be seen that the EDF method utilizes the charging stations fully but the robots do not wait for charging to reserve battery. Under the JIT method, all robots decide to charge at almost the same time and thus have to wait in turn for the charging station. In both cases, due to the large number of robots, some robots run out of battery during task execution or waiting at around 200 time steps. Under the EDF+JIT method, even through mostly all robots remain safe, the performance is much lower compared with our PSN method. Particularly, the utilization factor of each robot (measured as the ratio of time for task execution) is around 0.17 for EDF+JIT and around 0.33 for PSN. It can be seen from the schedule 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.

1.2.6 Generalization to Different Tasks

We demonstrate here that the *same* learned safety network can be applied to the training of different task specifications, e.g., different distribution density of tasks: sparse, normal and dense. As described earlier, task density is the maximum interval between when new tasks are generated. Such interval is set to $16, 8, 4$ time steps for the sparse, normal and dense condition, respectively. All task locations are randomly chosen during training and test time. The performance results during test time are shown in Table 2, which shows that the performance improves accordingly (e.g., almost doubled) when the density is doubled from 'sparse' to 'normal'. However, when the density is changed from 'normal' to 'dense', the performance lags behind because the system is already close to its operation limit during normal density, in terms of maximum number of tasks it can handle. Note that for our approach PSN, the *same* safety network is used during the training of different task networks.

References

[1] 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.

[2] 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.

[3] 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.

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

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

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