Bounded Suboptimal Search with Learned Heuristics for Multi-Agent Systems

Markus Spies,* Marco Todescato,* Hannes Becker, Patrick Kesper, Nicolai Waniek, Meng Guo
Bosch Center for Artificial Intelligence (BCAI), Renningen, Germany
{markus.spies2, marco.todescato, hannes.becker,
patrick.kesper, nicolai.waniek, meng.guo2}@de.bosch.com
(*) equal contribution

Abstract

A wide range of discrete planning problems can be solved optimally using graph search algorithms. However, optimal search quickly becomes infeasible with increased complexity of a problem. In such a case, heuristics that guide the planning process towards the goal state can increase performance considerably. Unfortunately, heuristics are often unavailable or need manual and time-consuming engineering. Building upon recent results on applying deep learning to learn generalized reactive policies, we propose to learn heuristics by *imitation learning*. After learning heuristics based on optimal examples, they are used to guide a classical search algorithm to solve unseen tasks. However, directly applying learned heuristics in search algorithms such as A^* breaks optimality guarantees, since learned heuristics are not necessarily admissible. Therefore, we (i) propose a novel method that utilizes learned heuristics to guide Focal Search A^*, a variant of A^* with guarantees on bounded suboptimality; (ii) compare the complexity and performance of jointly learning individual policies for multiple robots with an approach that learns one policy for all robots; (iii) thoroughly examine how learned policies generalize to previously unseen environments and demonstrate considerably improved performance in a simulated complex dynamic coverage problem.

1 Introduction

Intelligent robotic systems have to select from a diverse set of actions to perform complex operations. For instance, they must plan sequences of optimal actions to complete specified tasks such as following trajectories, navigation within clustered workspace, or coordination among different robots. Unfortunately, planning is known to be a hard computational problem that often relies on heavily engineered solutions, for instance heuristic-based rules which are tailored to specific environments (Ghallab, Nau, and Traverso 2004; LaValle 2006; Latombe 2012). These solutions typically lack generalization capabilities and performance guarantees for new environments. Conversely, *intelligent* robotic systems will be deployed in previously unseen, complex environments and thus need capabilities to plan dynamically while reacting to unforeseen situations.

Here, we propose a step towards bridging this gap by exploiting recent advancements in deep learning which yields intelligent behavior without hand-crafted solutions. Specifically, rather than synthesizing an entire plan, we learn policies that output action probabilities given an observation of the current system state. We build upon recent results where a reactive policy was learned from expert demonstrations using a deep neural network (Groshev et al. 2017). The learned policy generalizes to some degree to novel environments, which is, however, limited and typically not optimal for every situation. Nevertheless the learned policy can be used as a *guiding heuristic* in classical planners such as A^*. Still, directly applying learned policies, as previously proposed by Groshev et al. (2017), breaks optimality or even completeness guarantees, because the learned heuristic is not ensured to be admissible (Russell and Norvig 2016).

In this paper, we therefore present a novel combination of learned policies with ω-optimal A^* focal search (A^*_ω), introduced by Pearl and Kim (1982). Our algorithm exploits guidance from learned policies and ensures not only completeness but also bounds on suboptimality. We model the generalized policy with a deep convolutional neural network that takes observations of the environment states as input and outputs next actions as well as the predicted remaining plan length. Furthermore, we propose two different heuristics that are derived from the outputs of the learned network, one that directly predicts a value function, and a second one that estimates the path likelihood of nodes that are expanded during search.

In summary, the contibution of this paper is (i) we propose a novel combination of learned heuristics with A^*_ω search. This combination guarantees bounds on suboptimality while exploiting guidance from learned policies; (ii) we present a general framework for learning control policies and extend previous work to multi-agent systems; (iii) we introduce a novel problem domain based on the real world application of autonomous valet parking. Extensive experimental evaluations demonstrate the advances or our approach over existing methods in this domain.

2 Related work

Graph search has been used extensively in the past decades to solve planning problems, resulting in search algorithms such as A^* (Hart, Nilsson, and Raphael 1968) that are optimal and complete if admissible guiding heuristics are used. For problems with a large number of states, trade-offs are commonly made between solution quality and planning efficiency. For instance, A^* variants with bounded relaxation have been proposed that allow suboptimal results (Arya et al. 2004; Helmert and Domshlak 2009; Pearl and Kim 1982; Cohen et al. 2018). In particular, A^*_ω by Pearl and Kim (1982) uses an additional heuristic $h_F(n)$ that does not need to be admissible but guides planning into promising regions. Cohen et al. (2018) present an *anytime*-version of A^*_ω that tightens the bound iteratively over the planning time.

Finding good heuristics can be hard and in many cases involves tedious manual work. Moreover, good heuristics are often subject to the environments for which they were designed, i.e., they do not generalize well to novel environments. Consequently, machine learning has been suggested and successfully applied to learn and improve such heuristics. For instance, Samadi, Felner, and Schaeffer (2008) combine several existing heuristics using artificial neural networks and Arfaee, Zilles, and Holte (2011) propose the method of iterative deepening, which starts planning with weak heuristics and iteratively improves them. In contrast, we focus on the combination of bounded suboptimal planning with learned policies.

Over the past years, deep neural networks (DNN) have been used with extraordinary success in a plethora of different domains, e.g., image classification (Krizhevsky, Sutskever, and Hinton 2012), natural language processing (Sutskever, Vinyals, and Le 2014), or control (Mnih et al. 2015). This, together with DNNs theoretically being universal function approximators (Cybenko 1989), has motivated their usage in the planning domain. One of the first examples is the work by Ernandes and Gori (2004), in which DNNs were used to learn heuristic value functions for guided planning. Whereas they apply *likely admissible* heuristics for which the admissibility requirement is relaxed in a probabilistic sense, our method computes solutions with a hard suboptimality bound.

Besides learning value functions as heuristics, Groshev et al. (2017) proposes to directly learn policies, i.e., mappings from state to actions by imitation learning. Imitation learning and, in particular, behavioral cloning are supervised learning techniques in which a model learns the correct control policy based on sequences of observation-action pairs that are provided by an expert demonstrator. As shown in Groshev et al. (2017), the learned policy is able to generalize to situations it has never seen during training. Behavioral cloning was also successfully used in applications such as path following with obstacle avoidance (Tamar et al. 2016) or focused robot skills (Mülling et al. 2013). Others used actor-critic methods to directly learn policies in multi-agent scenarios for the purpose of collective construction (Sartoretti et al. 2018).

Our approach is closely related and builds upon the work by Groshev et al. (2017). We extend their work to learn multi-agent policies and with novel methods that combine the learned policies with bounded suboptimal planners. This allows us to retain the aforementioned guarantees. Furthermore, this outperforms the previous method in the proposed application domain.

Our application domain is connected to coverage problems, traditionally tackled by choosing frontier cells for the robots to explore (Burgard et al. 2005) or by designing local potential functions that the robots minimize (Cortes et al. 2002; Lee, Diaz-Mercado, and Egerstedt 2015). However, these approaches consider static environments, whereas the main focus here is on dynamic coverage.

3 Bounded Suboptimal Search with Learned Heuristics

In this part, we start with the preliminaries of anytime focal A^* search. Then we formally state the system model and the planning objective. The proposed planning framework consists of mainly two parts: first, we describe the supervised imitation learning scheme to learn a generalized policy based on observation-action pairs of an optimal expert demonstrator. Second, we show how this learned policy can be fused with anytime focal A^* search algorithms to ensure bounded suboptimality under different heuristics.

3.1 On Anytime Focal A^* Search

A^* is a search algorithm that computes minimum-cost paths from a start node s to a goal node g on graphs with *non-negative* edge-cost (Hart, Nilsson, and Raphael 1968). During search, the algorithm maintains an open list $\mathcal{N}$ of nodes and always expands the node with minimal *f-value*. This value is computed by a function $f(n)$, defined as $f(n) = b(n) + h(n)$, where $b(n)$ is the currently best-known cost from the start node s to node n, and $h(n)$ is a heuristic[1] that estimates the cost from node n to the goal node g. After node g was chosen for expansion, the algorithm guarantees to return the minimum-cost path if the heuristic is admissible, i.e., if it always *under*-estimates the true minimal cost from a given node to the goal. In the special case of $h(n) = 0, \forall n$, A^* defaults to Dijkstra's algorithm (Dijkstra 1959). The efficiency of A^* strongly depends on the nature of the heuristic $h(n)$. For example, if a perfect oracle heuristic is available, A^* only expands the nodes on a minimum-cost path (Russell and Norvig 2016). In practice, however, it is usually hard to find admissible heuristics that are tight lower bounds for the true cost.

Several A^* variants trade solution quality for planning efficiency via bounded relaxations. For these algorithms, guarantees on the solution quality can be given. In particular, A^*_ω by Pearl and Kim (1982) guarantees solutions that are not worse than $\omega \cdot c_{\text{opt}}$, where c_{opt} is the cost of the optimal solution and $\omega \geq 1$ is a design parameter called *focal value*. In contrast to A^*, focal search does not always choose $n_{\min} = \text{argmin}_n f(n)$, but maintains a *focal set* with all nodes for which $f(n) < \omega \cdot f_{\min}$, with $f_{\min} = \min_{n \in \mathcal{N}} f(n)$. When choosing only nodes from the focal set for expansion in every step, the above bound on the resulting cost is guaranteed. Still, the performance highly depends on *which* node within the focal set the algorithm chooses. For this, A^*_ω uses an *additional* heuristic $h_F(n)$, selecting nodes that are 'more promising' to expand. It is important to note that this heuristic $h_F(n)$ does not need to be admissible.

[1]In our experiments we set h to be the steps-to-go for the car for vanilla $A^*$and for all variants of Focal A^*.

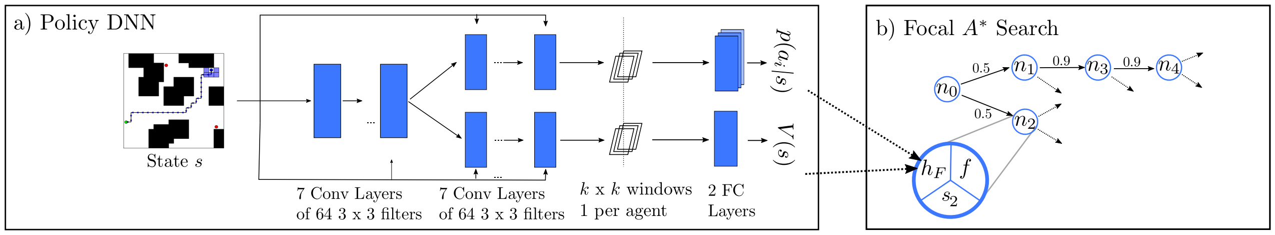

Figure 1: *a)* Policy network structure. It consists of one path for predicting action probabilities $p(a_i|s)$ for each robot, one for predicting the plan length for all the robots. *b)* Focal A$^*_\omega$ search. Example search path with action probabilities depicted on the edges. The close-up shows information attached to each node: the state s_i, the f-value $f(n)$ and the focal heuristic $h_F(n)$.

Instead of specifying values of ω, Cohen et al. (2018) present an *anytime* version of A$^*_\omega$ in which the search starts with high ω and tightens the bound iteratively. Whenever A$^*_\omega$ returns a solution path P, the value $\bar{\omega} = c(P)/f_{\text{min}}$ is a valid suboptimality bound, since the minimum f-value depends on an admissible heuristic and always under-estimates the cost of the optimal solution. (This is not true when using classical A* with non-admissible heuristics, since then the f-value itself depends on a non-admissible heuristic). Also, by design of A$^*_\omega$, it always holds that $\bar{\omega} \leq \omega$. Cohen et al. (2018) propose to continue search after a valid solution has been found, e.g. by choosing $\omega_{\text{new}} = \bar{\omega} - \epsilon$, where $\epsilon > 0$ is a small step size, which guarantees decreasing values of ω sizes over time. Importantly, the anytime version can efficiently re-use the existing search tree when continuing with decreased values of ω.

3.2 Problem Description

We consider a fully-observable state space S on which a set of N agents operate jointly. The agents perform joint actions $a = (a^1, \ldots, a^N) \in A$ that factor into one action a^i for each agent i. Furthermore, we assume a known, deterministic dynamics model of the environment $s_{t+1} = f(s_t, a_t)$ that maps a state $s_t \in S$ and action $a_t \in A$ at time step t to the state s_{t+1} at time step $t + 1$. Function $c(s, a) \in \mathbb{R}^+$ defines a cost for each state-action pair. In addition, we have an initial configuration s_{init}, a set of invalid states $\{s\}_{\text{invalid}}$ which the system is not allowed to reach, as well as a set of goal states $\{s\}_{\text{goal}}$. A control strategy $\pi : S \to A$ is a mapping from state $s \in S$ to action $a \in A$. The objective is to construct a control strategy that drives the system from the initial state to any goal state while avoiding the invalid states. We refer to a *problem instance* as the tuple $(S, A, f, c, s_{\text{init}}, \{s\}_{\text{invalid}}, \{s\}_{\text{goal}})$. While we assume that the state space S and action set A remain invariant for all possible problem instances, the initial, invalid, and goal states can change per instance.

Given a problem instance and a control strategy π, the system path of length T is given by $P = s_{\text{init}} a_0 s_1 a_1 \cdots a_{T-1} s_T$, where $a_t = \pi(s_t)$. A system path is called *valid* if there exists $T > 0$ such that $s_T \in \{s\}_{\text{goal}}$ and $s_t \notin \{s\}_{\text{invalid}}, \forall t = 0, \cdots, T$. In addition, each valid path has an associated cost of $C(P) = \sum_{t=0}^{T-1} c(s_t, a_t)$. Then, *optimal planning* is concerned with finding the control strategy π^* that minimizes the cost of the resulting path P^*.

3.3 Multi-Agent Policy via Imitation Learning

As mentioned earlier, we first train a DNN to imitate expert behaviors. The network takes an observation of the current state of a problem instance as input and generates distributions over the set of actions A, i.e., $p(a|s)$ for each robot. Moreover, the same network also learns a *value function* $V(s)$ which estimates the minimum cost to reach the goal states for state $s \in S$.

The structure of our proposed DNN, shown in Figure 1, is based on the *single-agent policies* presented by Groshev et al. (2017) . The network consists of two predictive paths, one for the reactive policy and one for the value function. Each predictive path is composed of a series of 14 convolutional layers, consisting of 64 3×3 filters, followed by 2 fully connected layers. The first 7 convolutional layers are shared between the two paths. Moreover, each convolutional layer receives skip connections from the input. In the *multi-agent* case, we extract a window of fixed $k \times k$ size around *each* robot's position before entering the linear layers, whereby the network becomes invariant to the number of agents. This may appear as only a small extension of Groshev et al. (2017), where only a single window is extracted. However, as we will show in our experiments, this makes a big difference in performance compared to learning policies for each agent individually. The final action probabilities are computed by a softmax activation function and all other activation functions are linear rectifiers.

We generate training data with an optimal planner, i.e., A*, to solve sampled problem instances. Whenever the optimal planner could not solve an instance, we discard it from the training set. This yields training data that consists of optimal state-action pairs as well as state-value-function pairs. For training, we use standard *cross entropy* as loss function for action probabilities and ℓ_1-norm for value predictions.

The whole setup might raise the question why learning to imitate an optimal planner and not directly use the optimal planner. One important reason is that in many applications the optimal planner can not produce a reasonable plan within given timing limits. Therefore, we propose to learn a guiding heuristic *off-line* which helps to speed up *on-line* planning, as described in the next section.

3.4 Combine Learned Policy with Focal Search

The straightforward approach to use the DNN model learned as described in the previous section is to fully trust the network and act according to the learned policy, e.g., always act according to the most likely action. However, this approach does not perform well due to the limited generalization ability of the network, as also shown in the quantitative numerical evaluation in Section 4. A more robust way is to use the learned policy to assist classical planners such as A*. Groshev et al. (2017) propose to use the learned value function *directly* as planning heuristic. It considerably improves performance for some instances but invalidates optimality guarantees of the solution.

We solve this issue by combining the learned policy with focal A^*_ω, introduced in Section 3.1. Specifically, we propose two novel approaches to design the focal heuristic h_F: (i) Use the estimated value function from the learned network as h_F. We refer to this algorithm as $A^*_{\omega,\text{len}}$. (ii) Use the path probability estimated by the learned policy as h_F. This algorithm is referred to as $A^*_{\omega,\log}$. The motivation behind this focal heuristic is that it guides the search along paths that the learned action probabilities suggest.

The log-likelihood of a path $P = s_0 a_0 s_1 a_1 \cdots a_{T-1} s_T$ is given by $\log p(P) = \sum_{t=0}^{T-1} \log p(a_t|s_t)$, where $p(a_t|s_t)$ is the probability of choosing action a_t at state s_t. During search, we would like to estimate the likelihood of the path through the node that is being evaluated. However, we only know the actions preceding the current time step $t_k > 0$. To avoid bias towards the beginning of the path, we therefore assume a uniform policy for all succeeding steps up until a fixed horizon, i.e.,

$$\log p(a_0, .., a_{t_k}, .., a_T) \approx \sum_{i=0}^{t_k} \log p(a_i) + \sum_{i=t_k+1}^{T} \log \frac{1}{|A|} ,$$

where we set T to the maximum number of steps of any node in the tree, and $|A|$ is the cardinality of the set of actions.

Figure 1*b* depicts an example where n_2 and n_4 are currently in the open list. Using h_F with log-likelihood estimation would prefer n_2 over n_4 if we were only adding along the currently expanded path, since $\log(0.5) < \log(0.5 \cdot 0.9 \cdot 0.9)$. Normalizing to a common path length with the proposed method, however, prefers n_4 because now the log-likelihood of n_2 changes to $\log(0.5 \cdot 0.5 \cdot 0.5)$.

Since we only modify the focal heuristic h_F in A^*_ω, i.e., not $h(n)$, we retain the guarantees on bounded suboptimality. At the same time, the A^*_ω search process is greatly accelerated via using the learned heuristic as guidance.

4 Experimental Results

In the following, we present experimental results of applying the proposed approach to the use case of *Autonomous Valet Parking* (AVP). The resulting dynamic coverage problem is computationally complex and cannot be solved optimally using A* in acceptable time, which motivates the proposed use of learned heuristics with focal search.

4.1 Autonomous Valet Parking

An AVP system parks cars or picks up parked cars autonomously in public parking decks. Most existing solutions require extensive infrastructures such as a large number of laser scanners, tracks, and shuttles to move cars. Such solutions are not only expensive, but also customized for specific environment structures. These problems could be mitigated by replacing static infrastructure with mobile robots that are equipped with sensors (mobile sensor units). Besides autonomously parking the car, a critical safety requirement is that the area in front of the car has to be monitored continuously to ensure no collisions with obstacles or humans. Even though robot localization and navigation techniques are quite mature (Thrun, Burgard, and Fox 2005), the dynamic coverage problem still has to be solved, i.e., where to position which mobile sensor unit for optimal execution time with minimal energy consumption. Instead of hand-crafted solutions, we propose to apply the proposed method to solve this dynamic coverage problem for multi-robot systems.

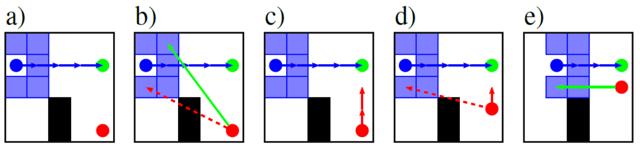

Figure 2: Execution of an optimal plan in a single-robot AVP example. a) The car (blue circle) has a fixed desired path τ_c (blue arrows) to reach its goal (green circle). b) Each cell of the car's monitor region (shaded blue area) has to be observed by at least one robot. While the robot (red circle) can observe some cells (green solid arrow), the line-of-sight to other cells is occluded (dashed red arrow) by an obstacle (black area). c) The optimal strategy for the robot is to move two cells upwards, because the line-of-sight is still occluded in position (d). e) The monitor region is fully observable and the car can drive to the next cell along its path.

Modeling AVP: We model the AVP as a problem instance, described in Section 3.2, as follows. Assume a grid world with dimensions $D_x, D_y \in \mathbb{N}$, where each cell $cell(x, y)$, with $(x, y) \in \mathcal{W} \triangleq \{1, \ldots, D_x\} \times \{1, \ldots, D_y\}$, is either occupied by an obstacle or free. The desired path of the car $\tau_c = s^c_1 s^c_2 \cdots s^c_L, s^c_i \in \mathcal{W}$, is assumed to be provided by an external planner. We have $R \in \mathbb{N}$ robots available to guide the car along its path. The current state space is given by $S = (s^r_1, s^r_2, \ldots s^r_R, s^c)$, where $s^r_i \in \mathcal{W}$ is the position of robot i and $s^c \in \tau_c$ is the position of the car. Note that the car is assumed to only move along its path τ_c. The robots can perform actions $a^r_i \in \{up, down, left, right, none\}$, i.e., move to an adjacent cell within the grid world or stay still. Goal states are the terminal states of car paths, and invalid states are the states where robots or the car collide into each other or into obstacles. Lastly, the car is only allowed to move to the next cell in its path when every cell in a monitor region is visible to at least one robot. This monitor region is defined as the set of cells surrounding the front of the car. As a cost function $c(a, s)$, we choose 1 for every time step until a goal state is reached, and in addition a cost of 1 for every robot action. The robots are equipped with 360° field-of-view laser scanners with infinite depth. In practice, we use a ray tracer and test the center of each cell to examine

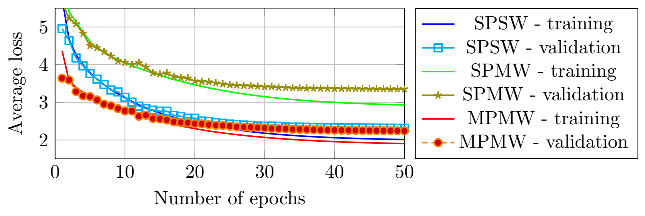

Figure 3: Learning curves during training, over training and validation set, as function of the number of training epochs.

whether a cell is visible for a robot or if the line-of-sight is occluded. Figure 2 depicts an example of a car path, the monitor region, the line-of-sight test, as well as execution of an optimal plan for the AVP problem.

4.2 Dataset Creation and Policy Learning

Training and Validation Dataset We generated a diverse set of problem instances (grid worlds of size 20×20) for the AVP domain as described in Section 4.1 by randomly sampling numbers and shapes of obstacles, initial positions for the car and the robot(s), as well as the goal position for the car. In a first step, we computed the desired path for the car, ignoring the robots, using a python implementation of standard A* planner with Euclidean distance as heuristic h. In a second step, an optimal planner based on A* computed optimal action sequences for the robot(s) that minimize the objective function. In this case the heuristic h is the steps-to-go for the car. Since A* is complete and the state space of the problem instances is finite, the optimal planner eventually returns for all instances, although in some complex cases only after *hours* of computing. Either it returns a valid solution or it fails when no solution is possible, in which case we discard this instance. Finally, we added all intermediate states along the optimal path together with their respective optimal actions into the dataset.

Learning the Models For training, the network receives an observation of the current system state which contains the static map, the car path together with its current position along it, as well as the robots' current positions. Specifically, a 3-layer tensor of dimension $\mathcal{W} \times 3$ from the current state of a grid world is created. The first layer contains a binary map of static obstacles, i.e., $cell(x, y)$ of this layer is 1 if there is an obstacle, and 0 otherwise. The second layer consists of the car position and path, encoded as $cell(x^c_{s^c}, y^c_{s^c}) = {}^{s^c}/{}_L$, $s^c \in \{1, \dots, L\}$. Hence this also embeds the car goal position. The third layer contains the robot positions, encoded as $cell(x^r_i, y^r_i) = i$, $i \in \{1, \dots, R\}$. The network outputs are then trained against the ground truth actions and value functions from the dataset, as described in Section 3.3.

4.3 Experiments and Results

We evaluated the ability of different policies to learn and, more importantly, generalize to new problem instances. Also, we examined the need to learn *true* multi-robot policies versus the possibility of applying single-robot policies to multi-robot scenarios. For this, we used the following two experiments: (1) A comparison of the accuracy and performance between learning single vs multi-robot policies. (2) A comparison between the different strategies for solving the entire problem instances.

	SPSW 1 rob.	SPSW 2 rob.	SPMW 2 rob.	MPMW 2 rob.
mean acc.	0.84	0.045	0.25	0.64
std acc.	0.37	0.21	0.43	0.48
mean len. err.	0.88	17.67	1.26	0.81
std len. err.	3.83	8.40	4.53	3.00

Table 1: Performance of learned policies on the validation set. Comparison in terms of accuracy as the percentage of correctly predicted actions, and plan length prediction error. Mean and std values are computed on a per trial basis by forward simulating each world in the validation set.

Learning single vs multi-robot models: In this experiment we learned three different DNN-models for single and multi-robot problem instances. To distinguish among the models, we denote them as follows:

- SPSW – Single-robot Policy over Single-robot Worlds: outputs action probabilities and a plan prediction for one single robot and is learned over problem instances with one single robot.
- SPMW – Single-robot Policy over Multi-robot Worlds: outputs action probabilities and plan prediction for one single robot but is learned over problem instances with two robots. In particular, the network selects only one $k \times k$ window at the position of one of the robots, for which it outputs the action probability. This policy can also be seen as the *decentralized* case, where each robot observes the full state but only controls its own actions.
- MPMW – Multi-robot Policy over Multi-robot Worlds: outputs action probabilities and plan prediction for two robots and is learned over problem instances with two robots. In contrast to SPMW, this reflects the *centralized* case, where one policy controls all robots.

For this, we created two different test and validation datasets according to Section 4.2. The first contains problem instances where one single robot is in charge of escorting one car. The second contains instances with two robots escorting one car. To examine performance over training, we performed validation (with fixed weights) after each training epoch. All reported timing information in the experimental section refer to a server with 2.10 GHz Intel(R) Xeon(R) E5-2695 v4 CPUs and a GTX 1080 TI graphics card.

The learning curves in Figure 3 show the average loss, over the corresponding validation dataset, for the learned policies as function of the number of training epochs. The curves demonstrate the ability of each policy to consistently learn without over-fitting their corresponding training set. More importantly, Table 1 compares accuracies of action choice, and plan length predictions when the learned

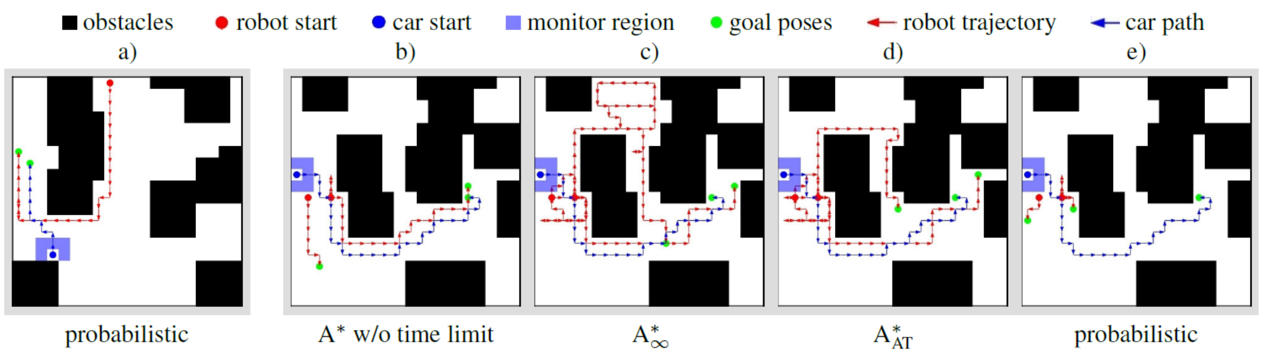

Figure 4: Examples of solved problem instances. Each image depicts a full run to the goal state. a) Single-robot problem instance, solved perfectly just by applying the learned policy. b)-e) Multi-robot problem instances. b) Optimal solution computed by A* without time limit. A* could not find a solution in 5 min. c) First solution computed by $A^*_{\infty,\log}$ after 2.5 s, d) Refined solution by A^*_{AT} after five minutes. e) Directly applying the policy MPMW did not solve the problem.

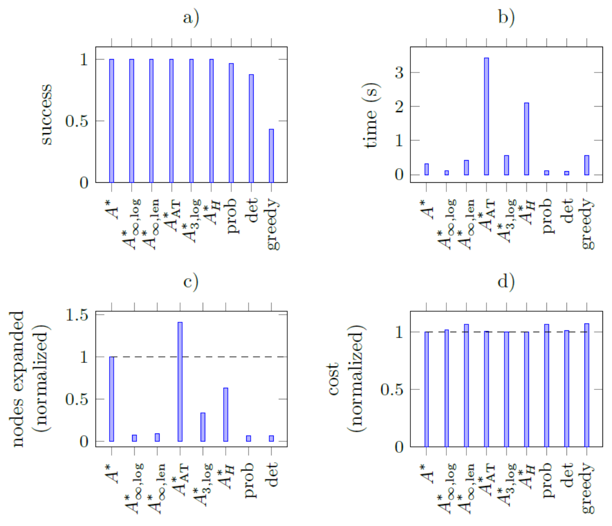

Figure 5: Comparing different policies solving 100 problem instances with *one* robot using SPSW policies. a) Success rate. b) Computation time. c) Nodes expanded during search. d) Cost of the final solution, normalized to A*.

networks were applied to multi-robot problem instances. Single-robot policies from SPSW and SPMW were applied to each robot, while MPMW already outputs control actions for all robots. For completeness also the values of SPSW applied to single robot dataset are shown in the table in the left column. The figure shows that merely applying single-robot policies, even if learned over multi-robot scenarios, leads to poor performance. Although learning single policies over multi-robot scenarios (SPMW) leads to improved performances, it is still not sufficient to reach acceptable performance as indicated by the mean accuracy in Table 1. In other words, the results demonstrate that multi-robot coordination must be learned jointly.

Application of learned heuristics: In this experiment we took the learned policies SPSW, SPMW and MPMW from above and used them as learned heuristics in combination with different control strategies. Each combination was tested on 100 randomly selected worlds from the respective validation sets. However, each problem instance was limited to a maximal planning time of 5 min (arbitrarily chosen). If this time was exceeded, the control strategies failed. Moreover, whenever the system reached an invalid state, the episode was terminated. In particular, we evaluated the following control strategies:

- **A***: Classical planner without learned policy. Heuristic function h is the steps-to-go for the car.
- **Focal A*, first solution** ($A^*_{\infty,\log}$): Focal search with $\omega = \infty$. This means that the search fully trusts the focal heuristic and returns after the first valid solution has been found. Its suboptimality bound can be computed after a solution is found (see Section 3.1). Using the learned policy to compute h_F with log-likelihood estimation.
- **Focal A*, first solution** ($A^*_{\infty,\text{len}}$): As before, focal search with $\omega = \infty$. Here the learned value function is used as focal heuristic h_F.
- **Focal A*** ($A^*_{3,\log}$): Focal search with $\omega = 3$ and learned log-likelihood estimation as h_F.
- **Anytime Focal A*** (A^*_{AT}): Focal search in anytime version, log-likelihood as h_F.
- **Groshev et al.** (A^*_H): Directly using the learned value function as heuristic in A* as proposed in Groshev et al. (2017)
- **Greedy**: Hand-crafted method to provide baseline benchmarks. The strategy employs available robots in the order of their relative distances to the monitored cells. Additional robots are acquired if the already deployed ones are insufficient to cover the monitor region.
- **Deterministic** (det): The deterministic control strategy maps a state to the action with the highest probability.
- **Probabilistic** (prob): Choose actions by a weighted random selection based on the action probability given by the network. Here, we allow repeatedly choosing an action

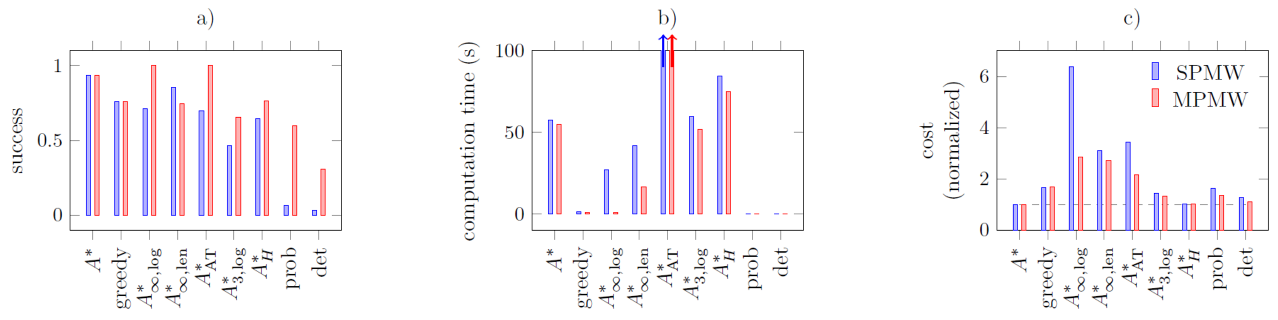

Figure 6: Comparing policies for problem instances with *two* robots. In all panels, blue and red bars correspond to using the SPMW or MPMW model, respectively. a) Success rates. b) Average planning time for all methods. c) Average cost of the returned environments, normalized to the optimal A*, only problem instances for which A*succeeded are considered.

until either a valid action is selected or until a maximum number of 5 trials is reached.

Figure 4 shows examples of a) one- and b)-e) two-robots problem instances from the validation set, solved using different control strategies. Figure 4*a* depicts the solution of directly applying the learned policy (SPSW) as a probabilistic strategy in a one-robot instance. In this comparably easy example the policy solved the problem almost optimally, which shows that the DNN models have learned meaningful behavior.

This is in contrast to more complex scenarios with two robots, as depicted in Figure 4*b-e*. Panel *b* shows the ground truth solution, computed by A* within 18 min, and resulting in a cost of 72. In the optimal solution, one of the robots escorts the car close to its goal pose. At the end of its path the car 'parks' close to a wall, which requires both robots to cooperate to observe the whole monitor region. Therefore, the second robot has to move to a location where it can monitor parts of the region that cannot be observed by the first robot.

Here, directly applying the learned (MPMW) model did *not* lead to successful behavior (as shown in Figure 4*e*). Also, classical A* could *not* find a solution within 5 min, while focal search guided by the learned heuristics found a first solution already after 1 s with a resulting cost of 211. In this example, the cost of the returned path is three times the optimal solution, which by design is less than the planner's guarantee of $\omega = 6$. Continuing focal search within the 5 min planning time refines the solution, as depicted in Figure 4*d*, resulting in $\omega = 5$ and $C(P) = 156$.

The qualitative results illustrated in the example above also show up in quantitative analysis. Figure 5 shows the average results of 100 single-robot problem instances using SPSW strategies. The success rates of all A* based methods are 1, while directly applying the learned probabilistic policies result in a success rate around 0.96. However, A$^*_{\infty,\log}$ uses an order of magnitude less nodes to find a valid solution compared to A*. The difference in computation time is not as big, since the network queries (of about 5 ms per query) increase computation time whenever the learned policy is used. Figure 5*d* shows the cost of the solutions, normalized over the solution of A*. In the single robot case, all strategies found solutions that are close to the optimal.

The planning difficulty increases *drastically* when two robots are deployed due to the much larger state space. Figure 6 shows results for SPMW (blue) and MPMW (red), averaged over 100 two-robot problems. In these complex scenarios, A* has a success rate of only 0.9 when planning time is limited to 5 min. Also, directly applying the policies did not perform well and showed success rates of at most 0.6 when MPMW was applied using the probabilistic strategy. Directly using the learned heuristic in an optimal planner (A^*_H), as proposed in Groshev et al. (2017), returns a success rate of about 0.8. Our proposed method A$^*_{\infty,\log}$ using the learned MPMW heuristics as guidance returned a solution in all of the 100 validation scenarios. Moreover, our method found a first solution already within 1 s on average, with average $\omega = 7.2$ and a real suboptimality factor of 3.1. Continuing the search for the remainder of the 5 min planning time window using A$^*_{\text{AT}}$ further reduced the value of ω to 4.2, with an actual suboptimality factor of 2.0 on average.

Comparing the results for SPMW and MPMW in Figure 6 suggests that jointly learned multi-robot policies significantly outperform single-robot policies. Also note that, in general, focal A* using log likelihood estimation as heuristic (A$^*_{\infty,\log}$) surpasses value function estimation (A$^*_{\infty,\text{len}}$). In all plots, whenever we show normalized values, we only compare planning instances for which A* found a solution within the allowed planning time.

The results of our experiments show that directly applying learned heuristics does not perform well in complex scenarios. Using them as guiding heuristics improves performance considerably, though. Moreover, using (anytime) focal search has the benefit of getting first solutions fast that can be refined over time, always having a guaranteed upper bound on the suboptimality factor.

5 Conclusion

In this work, we present novel methods to combine learned policies with classical planning methods to ensure bounded sub-optimality. Exhaustive experiments demonstrate that our technique clearly improves performances over previous methods, especially over directly applying the learned policies. Even though highly hand-crafted heuristics could still provide better performance, the proposed method requires much less manual engineering and, more importantly, is easily adaptable to different problem domains. Finally, we achieve guarantees for the qualities of our estimated solutions with respect to optimal solutions by combining an existing focal search algorithm with our learned heuristics.

Since the network already produces a policy and a value function, further work in reinforcement learning seems to be an obvious path to follow in future work. Specifically, we are interested in learning policies in even more complex tasks, such as problem instances with more than two agents for which optimal ground truth is not available.

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artif. Intell.* 175(16-17):2075–2098.

Arya, V.; Garg, N.; Khandekar, R.; Meyerson, A.; Munagala, K.; and Pandit, V. 2004. Local search heuristics for k-median and facility location problems. *SIAM Journal on computing* 33(3):544–562.

Burgard, W.; Moors, M.; Stachniss, C.; and Schneider, F. E. 2005. Coordinated multi-robot exploration. *IEEE Transactions on robotics* 21(3):376–386.

Cohen, L.; Greco, M.; Ma, H.; Hernandez, C.; Felner, A.; Kumar, T. K. S.; and Koenig, S. 2018. Anytime focal search with applications. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 1434–1441. International Joint Conferences on Artificial Intelligence Organization.

Cortes, J.; Martinez, S.; Karatas, T.; and Bullo, F. 2002. Coverage control for mobile sensing networks. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, 1327–1332. IEEE.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2(4):303–314.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1(1):269–271.

Ernandes, M., and Gori, M. 2004. Likely-admissible and sub-symbolic heuristics. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 613–617. Citeseer.

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

Groshev, E.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2017. Learning generalized reactive policies using deep neural networks. *CoRR* abs/1708.07280.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *ICAPS*, 162–169.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Latombe, J.-C. 2012. *Robot motion planning*, volume 124. Springer Science & Business Media.

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

Lee, S. G.; Diaz-Mercado, Y.; and Egerstedt, M. 2015. Multirobot control using time-varying density functions. *IEEE Transactions on Robotics* 31(2):489–493.

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

Mülling, K.; Kober, J.; Kroemer, O.; and Peters, J. 2013. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research* 32(3):263–279.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4(4):392–399.

Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *AAAI*, 357–362.

Sartoretti, G.; Wu, Y.; Paivine, W.; Kumar, S.; Koenig, S.; and Choset, H. 2018. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Proceedings of the International Symposium on Distributed Autonomous Robotics Systems (DARS)*, (in print).

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2154–2162.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.