Value Function Learning via Prolonged Backward Heuristic Search

Anonymous submission

Abstract

In practical applications like autonomous robots, we often need to solve similar problems repeatedly (e.g. replanning). Existing methods, that improve search performance based on learning from experience in similar previously solved problems, train the heuristics by imitating oracle data. However, such methods rather focus on generating the data with appropriate distribution (e.g. by aggregating data online) rather than the computational complexity of generating it. Computational complexity becomes especially limiting for high-dimensional problems. Here, we present a search-inspired method for systematic model exploration that allows us to efficiently generate data and use all explored states for learning the value function – that can then be employed as heuristic. Our method helps with data distribution as the search typically explores many more states besides the optimal path. The coverage can be improved even further with the Prolonged Search algorithm, which does not stop when a goal is reached, but rather keeps the search running until an extended region around the optimal path is explored. This, in turn, improves both the efficiency and robustness of successive planning. To address the negative effects of using an ML heuristic, we bound it with other heuristics to prevent (significant) overestimating the cost-to-go and ensure bounds on optimality even for non-iid or out-of-domain data. Our approach outperforms existing methods on benchmark problems and shows promising directions for developing efficient and robust search-based planning systems.

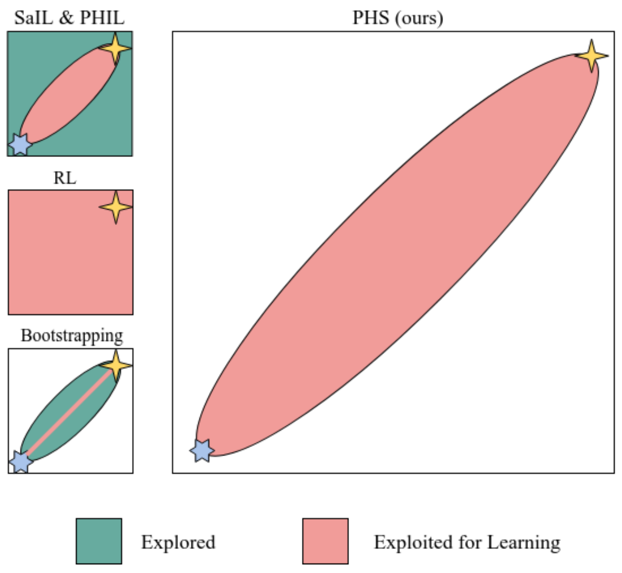

Figure 1: State-space explored and data used for learning in different methods including RL methods (e.g., ϵ-greedy exploration, SaIL (Bhardwaj, Choudhury, and Scherer 2017) and PHIL (Pándy et al. 2022), Bootstrapping (Groshev et al. 2018), DeepCube (Agostinelli et al. 2019) and Prolonged Backward Heuristic Search (PHS).

Introduction

Many real-world applications (e.g. autonomous robots, logistics, etc.) are using search-based planning algorithms. Considering that most of them require continuous replanning to adapt to dynamic environments, the speed of search algorithms becomes the major limitation for practical use, in particular for robots and safety-critical tasks. Therefore, many solutions end up with a conservative compromise of having a global-but-slow search-based planner generating a coarse plan, followed by a fast-but-local optimization-based planner. Having fast search-based planners would unlock many capabilities and enable real-time adaptation without the need for such compromises.

It is well known that the efficiency of search-based planning (i.e., A* Search (Hart, Nilsson, and Raphael 1968)) largely depends on the quality of the heuristic function for estimation of the cost-to-go (Helmert, Röger et al. 2008). Ideally, if we knew the exact cost-to-go (oracle), we could find the optimal solution with minimum effort (practically traversing greedy). If the robot constantly operates in similar environments, learning the cost-to-go function from previous search experience seems a reasonable approach. That would be also desirable as planning and learning methods have complementary strengths. It is an issue for learning approaches to provide any guarantees on performance, have safe exploration or learn long-term rewards. In these aspects, planning algorithms can provide valuable support. On the other hand, planning algorithms are rather slow in high-dimensional spaces which can be improved if planning is properly guided. The effective synergy of planning and learning provided exceptional results so far including seminal achievement of super-human performance in the game of Go (Silver et al. 2017).

Interaction of planning and learning has a long history (Bellman 1952; Korf 1990; Barto, Bradtke, and Singh 1995), with several modern directions including End-to-end learning approximations of planning algorithms (i.e. inspired by Value Iteration algorithm (Tamar et al. 2016), MCTS (Guez et al. 2018), MPC (Amos et al. 2018)), planning to guide exploration in Reinforcement Learning (Weber et al. 2017; Lowrey et al. 2018) and learning to guide planning (Kim, Kaelbling, and Lozano-Perez 2017; Bhardwaj, Choudhury, and Scherer 2017; Zhang, Huh, and Lee 2018; Ichter, Harrison, and Pavone 21.05.2018 - 25.05.2018) as well as model learning and Model-Based Reinforcement Learning in general.

(Whitehead 1991), inspired by the idea that RL can be viewed as an online search where an agent explores unknown environments instead of a simulated model, extend blind exploration (like in blind search e.g. Dijkstra's algorithm) with some human-cooperative mechanisms.

This work is in the direction of learning the value function in order to guide heuristic search-based planning but focused in particular on the method for the exploration of the model. The most similar approaches to the one presented in this work appeared in (Groshev et al. 2018) and in (Choudhury et al. 2018; Pándy et al. 2022). As it can be seen in Figure 1, the main drawback of the other methods is that they explore unnecessarily large portion of state space for computing the oracle data. In particular, in (Groshev et al. 2018) the authors used initial heuristics for path planning and used only nodes from the shortest path for the learning of improved value function (heuristics). In (Choudhury et al. 2018) and (Pándy et al. 2022) the authors make the assumption that the exploration algorithm can query the oracle for the exact cost-to-go value. For the oracle, they used backward Dijkstra's algorithm to compute heuristics and then the online mixed policy (Ross and Bagnell 2014) for selecting which states are useful for learning. However, Dijkstra's algorithm explores the whole search space which is unnecessary and extremely limiting for higher-dimensional problems. We relax this assumption and iteratively explore the state-space and compute oracle values.

Similarly, DeepCube (Agostinelli et al. 2019) used random backward scrambling to generate data for learning. A similar method was used also for domain-independent planners (Ferber et al. 2020). Although they utilize all explored states, the random backward rollout is de-facto brute-force exploration, and far from a systematic approach. It can explore the same states multiple times and does not cover neighboring regions. Additionally, it is not guided to the start and might not include any trajectory for a given start. Exploration is also important in Reinforcement Learning, in particular exploration-exploitation in online learning. However, our work is more related to offline learning in Sim2Real setting (Höfer et al. 2021), where the agent is acting in the simulation to learn the policy. RL agents learn based on all encountered data but explore the environment quite hectically. In particular, the search can be efficiently exploited there with reinitialization to desired states to systematically explore the state-space.

There are several recent approaches related to learning search and not directly to the exploration. For example, Neural A* Search (Yonetani et al. 2021) used a trainable encoder and differentiable A*, which allows training of the complete algorithm using backpropagation. Also, Neuro-algorithmic Policy (NAP) (Vlastelica, Rolinek, and Martius 2021) that is performing the planning on raw image inputs. These methods are extremely inefficient in terms of exploration and may run Dijskta's algorithm even at more iterations than e.g., (Choudhury et al. 2018) and (Pándy et al. 2022).

The main contribution of our work is a novel approach for efficient and systematic exploration of the models based on *backward* and *prolonged* heuristic search. This ensures that only interesting states are explored and all explored states are used for value function learning.

Premise: For learning of the Value function it is more beneficial to explore states in the neighborhood of the optimal path (policy) than elsewhere, as the agent (the planner) should spend most of the time in the neighborhood of the optimal path.

Having explored the neighboring regions around the optimal path helps to get back to the optimal path if the planner (or the agent) deviates and has better coverage for the A* search, which always looks for neighboring states. Inspired by this idea, we prolonged the search even after the optimal path was found to explore a wider region around the optimal path. Additionally, the direction of the search is reversed, such that the search starts from the goal node. In this way, all expanded nodes lead to the goal state and therefore can be used in the dataset for learning.

The contributions can be summarized as:

- search-inspired exploration method - PHS;
- asymmetric loss function for close-to-admissible value function learning;
- bounded ML heuristics for guarantees on sub-optimality.

Preliminaries

Search-Based Planing

We consider the problem of planning based on the graph search, as shown in the Algorithm 1. Starting from the *initial node* n_I (i.e., representing the initial state), chosen as the first *current node* n. At each iteration, successor nodes are generated in the function `Expand` by expanding the *current node* n using a transition model $\mathcal{M}$ to all reachable neighboring states. Each reachable collision-free state is represented with one *child node*. All collision-free *child nodes* $\mathbf{n}'$ are processed and are added to the OPEN list, if they are not in there already. If the *child node* is already in the OPEN list, and the new *child node* has a lower cost, the parent of that node is updated, otherwise, it is ignored. From the OPEN list, at every iteration, the node with the lowest cost is chosen to be the next *current node* (in the function `Select`), and the procedure is repeated until the goal is reached, the whole graph is explored or the computation time limit for planning is reached. At the end of the planning, if successful, the path is reconstructed starting from the goal n_G.

Algorithm 1: `Search`: search-based planning.

```
input  : $n_\mathrm{I}, n_\mathrm{G}, \mathcal{O}, \mathcal{M}, h(\cdot)$
1  begin
2      OPEN $\leftarrow n \leftarrow n_\mathrm{I}$                // initialization
3      CLOSED $\leftarrow \varnothing$
4      while $n \neq n_\mathrm{G}$ and OPEN $\neq \varnothing$ and
         CLOSED.$size() \leq N_\mathrm{max}$ do
5          $n \leftarrow$ Select(OPEN)
6          OPEN $\leftarrow$ OPEN $\setminus n$
7          CLOSED $\leftarrow$ CLOSED $\cup\ n$
8          $(\mathbf{n}', \mathbf{n}'_\mathrm{C}) \leftarrow$ Expand$(n, \mathcal{O}, \mathcal{M}, h(n))$
9          CLOSED $\leftarrow$ CLOSED $\cup\ \mathbf{n}'_\mathrm{C}$
10         foreach $n' \in \mathbf{n}'$ do
11             OPEN $\leftarrow$ Update(OPEN, $n'$)
12     return GetPath$(n_\mathrm{G})$    // reconstruct the path
```

Algorithm 2: Value function learning framework.

```
input  : $k_\mathrm{sc}, k_\mathrm{pr}, \mathcal{M}, h_\mathrm{adm}(\cdot), \mathcal{L}(\cdot)$
output: $h_\mathrm{ML}$       // Trained ML heuristic fuction
1  begin
2      $\mathcal{D} \leftarrow \varnothing$                          // Dataset
3      foreach $k \in [1, k_\mathrm{sc}]$ do
4          $\langle n_\mathrm{I}, n_\mathrm{G}, \mathcal{O}\rangle \leftarrow$ NewScenario()  // Curriculum
           // Search-based Exploration
5          $\mathbf{n} \leftarrow$ PHS$(n_\mathrm{I}, n_\mathrm{G}, \mathcal{O}, k_\mathrm{pr}, \mathcal{M}, h_\mathrm{adm})$
           // Extracting data from the search
6          foreach $n \in \mathbf{n}$ do
7              $\mathcal{D} \leftarrow \mathcal{D} \cup (n, n_\mathrm{G}, \mathcal{O}, n.g())$ // Data points
8      return Train$(\mathcal{D}, \mathcal{L})$
```

Learning Optimal Search

In this work, we focus on the Imitation learning approach for learning optimal search. In contrast to previous works in the field, we do not assume that we have access to the oracle, representing the exact cost-to-go but we consider the computation of that as part of the problem. So we compute the exact state-cost iteratively as er explore the environment.

The problem of Learning to search can be split into four subproblems:

- Curriculum design of example scenarios and scenario-variations;
- Scenario Exploration for exact state-cost data generation;
- Supervised Learning to imitate the exact state-cost;
- Using learned heuristic function in the search;

In this work, we assume having the fixed problem instances - curriculum similar to (Bhardwaj, Choudhury, and Scherer 2017) and focus on efficiently exploring them and learning a generalized value function.

Method

The presented approach for learning value function uses an existing admissible heuristic function (h_adm) and a known model $\mathcal{M}$ of the system to generate dataset $\mathcal{D}$ of exact state-cost (n, h^*) data points, as shown in the Algorithm 2. The dataset is used for supervised learning of the value function.

Algorithm 3: `PHS`: Prolonged Backward Heuristic Search-based exploration.

```
input  : $n_\mathrm{I}, n_\mathrm{G}, \mathcal{O}, k_\mathrm{pr}, \mathcal{M}, h(\cdot)$
output: $\mathbf{n}$                              // explored nodes
1  begin
2      OPEN $\leftarrow n \leftarrow n_\mathrm{G}$                // initialization
3      CLOSED $\leftarrow \varnothing$
4      while OPEN $\neq \varnothing$ and CLOSED.$size() \leq N_\mathrm{max}$ do
5          $n \leftarrow$ Select(OPEN)
6          OPEN $\leftarrow$ OPEN $\setminus n$
7          CLOSED $\leftarrow$ CLOSED $\cup\ n$
8          $(\mathbf{n}', \mathbf{n}'_\mathrm{C}) \leftarrow$ Expand$(n, \mathcal{O}, \mathcal{M}^{-1}, h(n))$
9          CLOSED $\leftarrow$ CLOSED $\cup\ \mathbf{n}'_\mathrm{C}$
10         foreach $n' \in \mathbf{n}'$ do
11             OPEN $\leftarrow$ Update(OPEN, $n'$)
12         if $n = n_\mathrm{G}$ then
13             $N_\mathrm{max} = k_\mathrm{pr} \cdot$ CLOSED.$size()$     // prolong.
14     return OPEN $\cup$ CLOSED    // all explored nodes
```

The learned value function is then used as a heuristic function h_ML in the search, bounded by the admissible heuristic to provide guarantees on sub-optimality. The dataset $\mathcal{D}$ consists of data points that carry information about the scenario (obstacles $\mathcal{O}$, initial n_I, and goal state n_G) and current state n together with the corresponding cost-to-go (i.e. shortest path from the current state to the goal state h^*). Using our `PHS` for exploration allows that, for each node n in the OPEN and CLOSED lists, the corresponding data point can be stored in dataset $\mathcal{D}$ This approach enables theoretically inexhaustible data generation from different scenarios and initial conditions, therefore using computational resources offline to have faster planning online, when necessary.

Prolonged Heuristic Search for Dataset generation

For the generation of dataset $\mathcal{D}$, planning algorithms can be used to generate data points with the exact cost-to-go. In the vanilla Shortest Path Planning problem (SP), the goal is to find only one collision-free path (i.e. from the initial to the goal state), so planning is stopped when the goal state is reached. As the goal state is reached only from one node (and each node has only one parent), the exact cost-to-go can be computed only for nodes on the optimal path. Contrary to SP, when generating the dataset, the objective is to generate as many different data points as possible. One approach is to use Backward Dynamic Programming (Dijkstra's algorithm), however, this would explore the whole search space which is not practical in higher dimensional problems. For this purpose, we propose a novel exploration approach Prolonged Backward Heuristic Search (`PHS`), as it is shown in Algorithm 3. `PHS` is a systematic and focused approach for generating the dataset $\mathcal{D}$. In `PHS`, the search is performed *backward*, from n_G so all explored nodes can be used in the dataset, as the exact cost to n_G is already computed. Additionally, as the region of interest is also the neighborhood of the optimal path, the search is not terminated when the initial node n_I is reached (as in the SP problem), but rather *prolonged* with *prolongation factor* k_pr, until the number of nodes in the CLOSED list is k_{pr} times the number when the goal was reached. Such prolongation assures that more nodes in the neighborhood of the optimal path are explored. Datapoints are constructed such that, for each node n in the OPEN and CLOSED lists, the corresponding scenario structure (grid) and cost-to-go are stored in dataset $\mathcal{D}$. Cost-to-go from node n to goal node n_{G} is actually cost-to-come $g(n)$ in backward search. In this way, paths do not have to be reconstructed and all expanded nodes are used in the dataset.

Value Function Learning

Learning of the Value Function h_{ML}, in this approach, is a supervised learning problem (regression). The proposed h_{ML} takes as input features representing the current and the goal nodes (n_{I}, n_{G}) and a situation (obstacles $\mathcal{O}$), and returns a scalar value representing an estimated cost to reach the goal from that node.

As it is preferred that the heuristic function underestimates the exact cost (admissibility), a non-symmetric loss function can be used. Asymmetry can be introduced by augmenting the Mean Square Error Loss function as:

$$e_i = y_i - \hat{y}_i, \tag{1}$$

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} e_i^2 \cdot (\mathrm{sign}(e_i) + a)^2, \tag{2}$$

with parameter $a < 0$ to emphasize the penalty for positive errors e.

Using Learned Value Function as Heuristic function

The learned value function is subsequently used as a heuristic function h in the search, Algorithm 1. To assure search performance improvement, we bound the ML heuristic h_{ML} from the lower side by the admissible heuristic h_{adm}. By doing that, we always consider the more informative heuristic. Additionally, to provide guarantees on sub-optimality, we bound the ML heuristic from the upper side by the weighted-admissible heuristic. The weighted-admissible heuristic is computed by multiplying admissible heuristic h_{adm} with user-defined weight ε, as in and ARA* (Likhachev, Gordon, and Thrun 2004) (or Weighted A* (Pohl 1970)). In this way, the heuristic is always at most ε-admissible, so the solution is always at most ε times greater than the optimal solution (Pearl 1984). Values of ε closer to 1 guarantee smaller deviation from optimal solution but reduce computational performance. While the larger values of ε limit ML heuristics less, but provide a less meaningful guarantee.

$$h = \min(\max(h_{\mathrm{adm}}, h_{\mathrm{ML}}), \varepsilon \cdot h_{\mathrm{adm}}), \quad \varepsilon \geq 1. \tag{3}$$

An alternative approach would be to use Multi-Heuristic A* Search (MHA*) (Aine et al. 2016) that is limiting the effect of the h_{ML} directly in the search.

Experiments

For the experiments, we use a grid world domain with 4-connected neighbors and 33 % of cells are covered with obstacles in average. In total, 531 different random scenarios were used for dataset generation. From each scenario, multiple data points are generated. For comparison, two datasets were created. One dataset (representing the approach from (Groshev et al. 2018)) is using only nodes from optimal path ($\mathcal{D}_{\mathrm{VAN}}$, 12.007 datapoints) and the other (as proposed in this work) uses all nodes explored in Backward Prolonged Heuristic Search ($\mathcal{D}_{\mathrm{PHS}}$, 122.449 datapoints). The advantage of Backward Prolonged Heuristic Search is already visible, as from the same number of scenarios about 10 times more data points are generated, even in simple 2D problem. This is expected to be even larger in higher dimensional problems.

Value Function approximation using Deep Learning

In this experiment, a fully Convolutional Neural Network (CNN) was used for the value function approximation. The complete model architecture can be seen in Table 1. Each layer uses the *SELU* nonlinear activation (Klambauer et al. 2017). The networks were trained for 4096 steps using a batch size of 1024 images and a learning rate of 0.001. The networks were initialized using the variance-scaling initializer (He et al. 2015) and optimized with the *ADAM* (Kingma and Ba 2014) optimizer. In the asymmetric loss function, the parameter a is set to -2.5.

Using ML Value Function as Heuristic function

Three different heuristic functions are used in the experiment and compared based on solution quality (i.e. path length) and planning efficiency (i.e. number of explored nodes). The first heuristic function is an admissible heuristic function h_{adm} based on Manhattan distance. The second heuristic function is value function h_{VAN} trained on dataset $\mathcal{D}_{\mathrm{VAN}}$ generated from the solution path only. The third function h_{PHS} is trained on $\mathcal{D}_{\mathrm{PHS}}$ from the proposed Prolonged Heuristic Search.

Discussion

Experimental results support the initial premise that generating a dataset using Prolonged Heuristic Search improves the quality and learnability of the value function. Figure 2 shows the training and test loss for both the $\mathcal{D}_{\mathrm{PHS}}$ and $\mathcal{D}_{\mathrm{VAN}}$ datasets. It can be seen that training on $\mathcal{D}_{\mathrm{VAN}}$ performs poorer than training on $\mathcal{D}_{\mathrm{PHS}}$, with slower convergence and a bigger difference between the training and test set, indicating overfitting. One reason might be the fact that

Table 1: Model Architecture

layer	kernel	stride	dilation	avg pool	output size
conv1	3×3	1×1	1×1	none	$30 \times 30 \times 4$
conv2	3×3	1×1	2×2	none	$30 \times 30 \times 8$
conv3	3×3	1×1	4×4	none	$30 \times 30 \times 16$
conv4	3×3	1×1	8×8	4×4	$7 \times 7 \times 32$
conv5	3×3	1×1	1×1	2×2	$3 \times 3 \times 64$
conv6	3×3	1×1	1×1	2×2	$1 \times 1 \times 1$

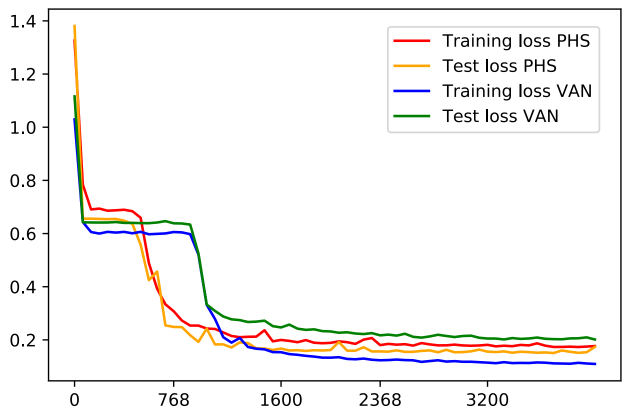

Figure 2: Training and test loss for both the $\mathcal{D}_{\text{PHS}}$ and $\mathcal{D}_{\text{VAN}}$.

$\mathcal{D}_{\text{VAN}}$ is smaller than $\mathcal{D}_{\text{PHS}}$, as the proposed approach manages to extract more data from the same scenarios. The other reason might be that the $\mathcal{D}_{\text{VAN}}$ was not diverse enough (i.e. many variations of the same scenario) and the network was not able to learn to generalize well. In contrast, the $\mathcal{D}_{\text{PHS}}$ offers more diverse training data (with many similar scenarios as all explored nodes are used), and the network is able to reduce the loss much further.

Additionally, the learned value functions were used as heuristic functions in the search. Totally 100 random scenarios were created and both h_{PHS} and h_{VAN} were used. Results were compared based on number of explored nodes (for search efficiency) and path length (for solution quality). Both were compared with the admissible heuristic h_{adm} while changing ε, which bounds the influence of ML heuristics h_{ML}. Figure 3 shows that the path length does not increase significantly even for $\varepsilon = 3.5$, which means that both ML heuristics still provide solutions close to optimal. While h_{VAN} has slightly worse performance than h_{adm}, h_{PHS} has equal performance to h_{adm} in this aspect. On the other hand, Figure 4 shows that both ML heuristics h_{PHS} expands fewer nodes than h_{adm}. In this example, the performance of h_{VAN} and h_{PHS} varies slightly on ε, and further study on different domains and scenarios is necessary. Figure 5 shows an example of h_{PHS} use. It is clear from the figure that h_{PHS} expands fewer nodes.

Conclusion

The presented approach offers the possibility to effectively include Machine Learning into a deterministic planning framework, promising significant performance improvements manifested in a reduced number of explored nodes compared to those obtained using an admissible heuristic (h_{adm}) while keeping guarantees on sub-optimality of the solution. The proposed approach uses the maximum of invested computational resources in planning as all expanded nodes in planning are used for learning. Experimental results showed significant improvement in search performance while keeping bounds on the sub-optimality of the solution.

Future steps would include studies of behavior in more scenarios and parameter variations, other domains (i.e. higher dimensional problems like robotic manipulation (Cohen, Chitta, and Likhachev 2010)), and kinodynamic motion (e.g., automated agile driving (Ajanović et al. 2023)). Additional venues for exploration are the extension of the approach to the exploration in end-to-end Model-Based Reinforcement Learning framework and adaptation to changing environment using Interactive Imitation Learning (Celemin et al. 2022).

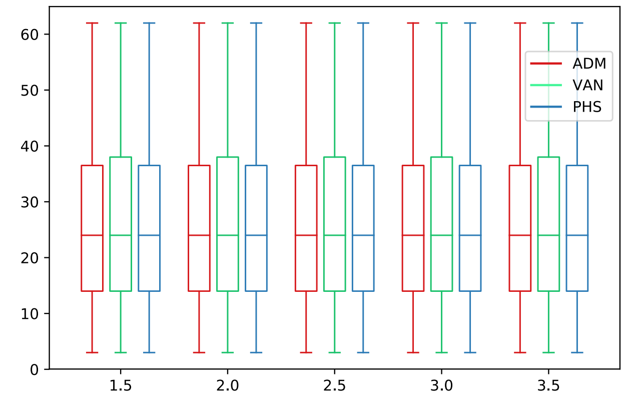

Figure 3: Path length for h_{adm}, h_{VAN} and h_{PHS} heuristic based on ε value.

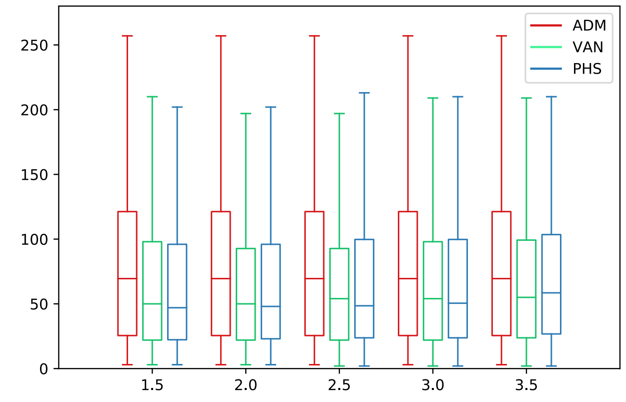

Figure 4: Number of explored nodes for h_{adm}, h_{VAN} and h_{PHS} heuristic based on ε value.

Acknowledgments

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's Cube with Deep Reinforcement Learning and Search. 1(8): 356–363.

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and

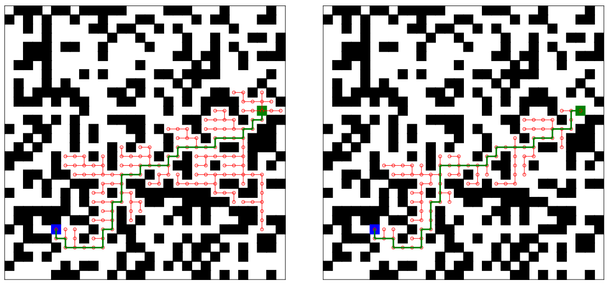
Figure 5: Nodes explored while planning using admissible heuristic h_{adm} (left) and trained ML heuristic h_{PHS} (right).

Likhachev, M. 2016. Multi-Heuristic A*. *The International Journal of Robotics Research*, 35(1-3): 224–243.

Ajanović, Z.; Regolin, E.; Shyrokau, B.; Ćatić, H.; Horn, M.; and Ferrara, A. 2023. Search-Based Task and Motion Planning for Hybrid Systems: Agile Autonomous Vehicles. 121: 105893.

Amos, B.; Jimenez, I.; Sacks, J.; Boots, B.; and Kolter, J. Z. 2018. Differentiable MPC for End-to-end Planning and Control. In *Advances in Neural Information Processing Systems*, 8289–8300.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2): 81–138.

Bellman, R. 1952. On the Theory of Dynamic Programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8): 716–719.

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning Heuristic Search via Imitation.

Celemin, C.; Pérez-Dattari, R.; Chisari, E.; Franzese, G.; family=Souza Rosa, p. u., given=Leandro; Prakash, R.; Ajanović, Z.; Ferraz, M.; Valada, A.; and Kober, J. 2022. Interactive Imitation Learning in Robotics: A Survey. 10(1-2): 1–197.

Choudhury, S.; Bhardwaj, M.; Arora, S.; Kapoor, A.; Ranade, G.; Scherer, S.; and Dey, D. 2018. Data-driven planning via imitation learning. *The International Journal of Robotics Research*, 37(13-14): 1632–1672.

Cohen, B. J.; Chitta, S.; and Likhachev, M. 2010. Search-based planning for manipulation with motion primitives. In *2010 IEEE international conference on robotics and automation*, 2902–2908. IEEE.

Ferber, P.; Helmert, M.; Hoffmann, J.; and rg. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. 2346–2353.

Groshev, E.; Tamar, A.; Goldstein, M.; Srivastava, S.; and Abbeel, P. 2018. Learning generalized reactive policies using deep neural networks. In *2018 AAAI Spring Symposium Series*.

Guez, A.; Weber, T.; Antonoglou, I.; Simonyan, K.; Vinyals, O.; Wierstra, D.; Munos, R.; and Silver, D. 2018. Learning to Search with MCTSnets.

Hart, P.; Nilsson, N.; 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.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

Helmert, M.; Röger, G.; et al. 2008. How Good is Almost Perfect? In *AAAI*, volume 8, 944–949.

Höfer, S.; Bekris, K.; Handa, A.; Gamboa, J. C.; Mozifian, M.; Golemo, F.; Atkeson, C.; Fox, D.; Goldberg, K.; Leonard, J.; et al. 2021. Sim2Real in robotics and automation: Applications and challenges. *IEEE transactions on automation science and engineering*, 18(2): 398–400.

Ichter, B.; Harrison, J.; and Pavone, M. 21.05.2018 - 25.05.2018. Learning Sampling Distributions for Robot Motion Planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7087–7094. IEEE. ISBN 978-1-5386-3081-5.

Kim, B.; Kaelbling, L. P.; and Lozano-Perez, T. 2017. Learning to guide task and motion planning using score-space representation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2810–2817. IEEE. ISBN 978-1-5090-4633-1.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization.

Klambauer, G.; Unterthiner, T.; Mayr, A.; and Hochreiter, S. 2017. Self-normalizing neural networks. In *Advances in neural information processing systems*, 971–980.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence*, 42(2-3): 189–211.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, 767–774.

Lowrey, K.; Rajeswaran, A.; Kakade, S.; Todorov, E.; and Mordatch, I. 2018. Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control.

Pándy, M.; Qiu, W.; Corso, G.; Veličković, P.; Ying, Z.; Leskovec, J.; and Lio, P. 2022. Learning Graph Search Heuristics.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Pohl, I. 1970. Heuristic Search Viewed as Path Finding in a Graph. 1(3): 193–204.

Ross, S.; and Bagnell, J. A. 2014. Reinforcement and Imitation Learning via Interactive No-Regret Learning.

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

Tamar, A.; WU, Y. I.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value Iteration Networks. In D. D. Lee; M. Sugiyama; U. V. Luxburg; I. Guyon; and R. Garnett, eds., *Advances* *in Neural Information Processing Systems 29*, 2154–2162. Curran Associates, Inc.

Vlastelica, M.; Rolinek, M.; and Martius, G. 2021. Neuro-Algorithmic Policies Enable Fast Combinatorial Generalization. In *Proceedings of the 38th International Conference on Machine Learning*, 10575–10585. PMLR.

Weber, T.; Racanière, S.; Reichert, D. P.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; Pascanu, R.; Battaglia, P.; Hassabis, D.; Silver, D.; and Wierstra, D. 2017. Imagination-Augmented Agents for Deep Reinforcement Learning.

Whitehead, S. D. 1991. A Complexity Analysis of Cooperative Mechanisms in Reinforcement Learning. In *AAAI*, 607–613.

Yonetani, R.; Taniai, T.; Barekatain, M.; Nishimura, M.; and Kanezaki, A. 2021. Path Planning Using Neural a* Search. In *International Conference on Machine Learning*, 12029–12039. PMLR.

Zhang, C.; Huh, J.; and Lee, D. D. 2018. Learning Implicit Sampling Distributions for Motion Planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3654–3661. IEEE. ISBN 978-1-5386-8094-0.