Learning Graph Search Heuristics

Michal Pándy *
University of Cambridge

Rex Ying
Stanford University

Gabriele Corso
MIT

Petar Veličković
DeepMind

Jure Leskovec
Stanford University

Pietro Liò
University of Cambridge

Abstract

Searching for a path between two nodes in a graph is one of the most well-studied and fundamental problems in computer science. In numerous domains such as robotics, AI, or biology, practitioners develop search heuristics to accelerate their pathfinding algorithms. However, it is a laborious and complex process to hand-design heuristics based on the problem and the structure of a given use case. Here we present PHIL (Path Heuristic with Imitation Learning), a novel neural architecture and a training algorithm for discovering graph search and navigation heuristics from data by leveraging recent advances in imitation learning and graph representation learning. At training time, we aggregate datasets of search trajectories and ground-truth shortest path distances, which we use to train a specialized graph neural network-based heuristic function using backpropagation through steps of the pathfinding process. Our heuristic function learns graph embeddings useful for inferring node distances, runs in constant time independent of graph sizes, and can be easily incorporated in an algorithm such as A* at test time. Experiments show that PHIL reduces the number of explored nodes compared to state-of-the-art methods on benchmark datasets by 40.8% on average and allows for fast planning in time-critical robotics domains.

1 Introduction

Search heuristics are essential in several domains, including robotics, AI, biology, and chemistry [1, 2, 3, 4, 5, 6]. For example, in robotics, complex robot geometries often yield slow collision checks, and search algorithms are constrained by the robot's onboard computation resources, requiring well-performing search heuristics that visit as few nodes as possible [1, 4]. In AI, domain-specific search heuristics are useful for improving the performance of inference engines operating on knowledge bases [3, 5]. Search heuristics have been previously also developed to reduce search efforts in protein-protein interaction networks [6] and in planning chemical reactions that can synthesize target chemical products [2]. This broad set of applications underlines the importance of good search heuristics that are applicable to a wide range of problems.

While there has been significant progress in designing search heuristics, it remains a challenging problem. Classical approaches [7, 8] tend to hand-design search heuristics, which require domain knowledge and a lot of trial and error. Domain-independent classical approaches [9, 10] develop useful meta-heuristics; however, learning-based methods demonstrate that this process can be learned from data. Learning-based methods face a different set of challenges. Firstly, the data distribution is not i.i.d., as newly encountered graph nodes depend on past heuristic values, which means that supervised learning-based methods [11, 12, 13, 14, 15] under-perform methods that take into account the sequential decision making aspect of the problem [1]. Secondly, heuristics should run fast, with ideally constant time complexity. Otherwise, the overall asymptotic time complexity of the search

*Correspondence to mpmisko@gmail.com.

Physical Reasoning and Inductive Biases for the Real World at NeurIPS 2021.

procedure could be increased. Finally, as the environment (search graph) sizes increase, reinforcement learning-based heuristic learning approaches tend to perform poorly [16]. State-of-the-art imitation learning-based methods can learn useful search heuristics [1]; however, these methods still rely on feature-engineering for a specific domain and do not generally guarantee a constant time complexity with respect to graph sizes.

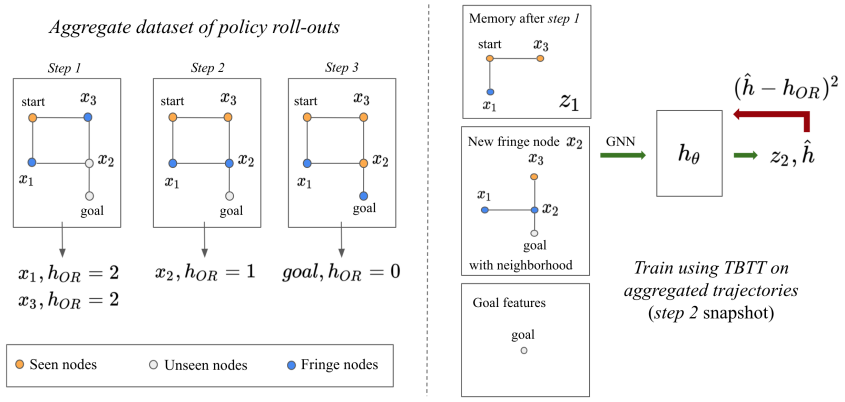


Figure 1: Main components of PHIL: On the left, we roll-out and aggregate search trajectories from the start node to the goal node. Each trajectory step contains a set of newly added fringe nodes with bounded random subsets of their 1-hop neighborhoods and their oracle (h^*) distances to the goal node. On the right, we use truncated backpropagation through time on each collected trajectory to train h_θ, where $\hat{h}$ is the predicted distance between x_2 and x_g, and z_2 is the updated state of the memory.

In this paper, we propose *Path Heuristic with Imitation Learning* (PHIL, Figure 1), a framework that extends the recent imitation learning-based heuristic search paradigm with a learnable *explored graph memory*. This means that PHIL learns a representation that allows it to capture the structure of the so far explored graph, so that it can then better select what node to explore next. We train our approach to predict the oracle node-to-goal distances of graph nodes during search. Key to our approach is a *specialized graph neural network architecture*, which allows us to apply PHIL to diverse graphs from different domains and encodes search-specific inductive biases in a constant time complexity.

2 Preliminaries

Graph search. Suppose that we are given an unweighted connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes, and $\mathcal{E}$ a corresponding set of edges. Further suppose that each node $i \in \mathcal{V}$ has corresponding features $x_i \in \mathbb{R}^{D_v}$, and each edge $(i, j) \in \mathcal{E}$ has features $e_{ij} \in \mathbb{R}^{D_e}$. Assume that we are also given a start node $v_s \in \mathcal{V}$ and a goal node $v_g \in \mathcal{V}$. At any stage of our search algorithm, we can partition the nodes of our graph into three sets as $\mathcal{V} = \mathcal{V}_{seen} \cup \mathcal{V}_{fringe} \cup \mathcal{V}_{unseen}$, where $\mathcal{V}_{seen}$ are the nodes already explored, $\mathcal{V}_{fringe}$ are candidate nodes for exploration (i.e., all nodes connected to any node in $\mathcal{V}_{seen}$, but not yet in $\mathcal{V}_{seen}$), and $\mathcal{V}_{unseen}$ is the rest of the graph. Each *expansion* moves a node from $\mathcal{V}_{fringe}$ to $\mathcal{V}_{seen}$, and adds the neighbors of the given node from $\mathcal{V}_{unseen}$ to $\mathcal{V}_{fringe}$. We call the set of newly added fringe nodes $\mathcal{V}_{new}$ at each search step. At the start of the search procedure, $\mathcal{V}_{seen} = \{v_s\}$ and we expand the nodes until v_g is encountered (i.e., until $v_g \in \mathcal{V}_{seen}$).

Greedy best-first search. We can perform *greedy best-first search* using a greedy fringe expansion policy, such that we always expand the node $v \in \mathcal{V}_{fringe}$ that minimizes $h(v, v_g)$. Here, $h : \mathcal{V} \times \mathcal{V} \longrightarrow \mathbb{R}$ is a tailored heuristic function for a given use case. In our work, we are interested in learning a function h that predicts shortest path lengths, this way minimizing $|\mathcal{V}_{seen}|$ in a *greedy best-first search* regime.

Imitation of perfect heuristics. Partially observable Markov decision processes (POMDPs) are a suitable framework to describe the problem of learning search heuristics [1]. We can have $s = (\mathcal{V}_{seen}, \mathcal{V}_{fringe}, \mathcal{V}_{unseen})$ as our state, an action $a \in \mathcal{A}$ corresponds to moving a node from $\mathcal{V}_{fringe}$ to $\mathcal{V}_{seen}$, and the observations $o \in \mathcal{O}$ are the features of newly included nodes in $\mathcal{V}_{fringe}$. We also

define a history $\psi \in \Psi$ as a sequence of observations $\psi = o_1, o_2, o_3, ...$ Our work leverages the observation that using a heuristic function during greedy best-first search that correctly determines the length of the shortest path between fringe nodes and the goal node will also yield minimal $|\mathcal{V}_{seen}|$. For training, we adopt a perfect heuristic h^*, similar to [1], which has full information about s during search. Such oracle can provide ground-truth distances $h^*(s, v, v_g)$, where $v \in \mathcal{V}_{fringe}$. To conclude, we define a *greedy best-first search policy* π_θ that uses a parameterized heuristic h_θ to expand nodes from $\mathcal{V}_{fringe}$ with minimal heuristic values.

3 Approach

Training objective. With the aim of minimizing $|\mathcal{V}_{seen}|$ after search, our goal is to train a parameterized heuristic function $h_\theta : \Psi \times \mathcal{V} \times \mathcal{V} \longrightarrow \mathbb{R}$ to predict ground-truth node distances h^* and use h_θ within a greedy best-first policy π_θ at test time. More specifically, we assume access to a distribution over graphs $P_{\mathcal{G}}$, a start-goal node distribution $P_{v_{sg}}(\cdot \,|\, \mathcal{G})$, and a time horizon T. Moreover, we assume a joint state-history distribution $s, \psi \sim P_s(\cdot \mid \mathcal{G}, t, \pi_\theta, v_s, v_g)$, where P_s represents the probability our search being in state s, at time $0 \leq t \leq T$ on graph $\mathcal{G}$ with pathfinding problem (v_s, v_g), with a greedy best-first search policy π_θ using heuristic h_θ. Hence, our goal is minimizing:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\substack{\mathcal{G}\sim P_{\mathcal{G}},\\ (v_s,v_g)\sim P_{v_{sg}}\\ t\sim\mathcal{U}(0,\ldots,T),\\ s,\psi\sim P_s}} \Big[\frac{1}{|\mathcal{V}_{new}|} \sum_{v\in\mathcal{V}_{new}} (h^*(s, v, v_g) - h_\theta(\psi, v, v_g))^2\Big] \tag{1}$$

Imitation learning algorithm. The high-level idea of our algorithm (Appendix C) is that we aggregate trajectories of search traces (i.e., sequences of new fringe nodes) and use truncated backpropagation through time to optimize h_θ after each data-collection step. In particular, after sampling a graph $\mathcal{G}$ and a search problem v_s, v_g, we execute our greedy learned policy π_θ induced by h_θ for $t \sim \mathcal{U}(0, \ldots, T - t_\tau)$ expansions, where T is the episode time horizon, and t_τ is the roll-out length. We obtain a new state $s = (\mathcal{V}^0_{seen}, \mathcal{V}^0_{fringe}, \mathcal{V}^0_{unseen})$, and an initial memory state z_t. Afterward, we execute/roll-out for t_τ steps our mixture policy π_{mix}, which is obtained by probabilistically blending π_θ and the greedy best-first policy induced by the oracle heuristic π^*. In a roll-out, we collect sequences of new fringe nodes, together with their ground-truth distances to the goal v_g, given by h^*. Once the roll-out is complete, we aggregate the obtained trajectory and the initial state for the following optimization using backpropagation through time.

Recurrent GNN architecture. In each forward pass, h_θ obtains a set of new fringe nodes $\mathcal{V}_{new}$, the goal node v_g, and the memory z_t at time step t. We represent each node in $\mathcal{V}_{new}$ using its features $x_i \in \mathbb{R}^{D_v}$, and likewise the goal node v_g using its features $x_g \in \mathbb{R}^{D_v}$. Further, for each $i \in \mathcal{V}_{new}$, we uniformly sample an $n \in \mathbb{N}_{\geq 0}$ bounded set of nodes present in the 1-hop neighborhood of i, calling this set $\mathcal{N}_i$, with $|\mathcal{N}_i| \leq n$. This sampling step produces a set of neighboring node features, where each $j \in \mathcal{N}_i$ has features $x_j \in \mathbb{R}^{D_v}$, and corresponding edge features $e_{ij} \in \mathbb{R}^{D_e}$.

Algorithm 1: Heuristic func. (h_θ) forward pass

Obtain $x_i, x_j, e_{ij}, x_g\ z_t$;
$x_i \leftarrow f(x_i, x_g, D_{EUC}(x_i, x_g), D_{COS}(x_i, x_g))$;
$x_j \leftarrow f(x_j, x_g, D_{EUC}(x_j, x_g), D_{COS}(x_j, x_g))$;
$g_i \leftarrow \phi(x_i, \oplus_{j\in\mathcal{N}_i} \gamma(x_i, x_j, e_{ij}))$;
$g'_i, z_{i,t+1} \leftarrow \text{GRU}(g_i, z_t)$;
$z_{t+1} \leftarrow \overline{z_{i,t+1}}$;
$\hat{h}_i \leftarrow \text{MLP}(g'_i, x_g)$;
return $\hat{h}_i, z_{t+1}$;

h_θ **forward pass.** In Algorithm 1, $f, \phi, \gamma, \text{GRU}[17], \text{MLP}$ are each parameterised differentiable functions, with ϕ, γ representing the *update* and *message* functions [18] of a graph neural network, respectively. In our forward pass, using the function f, we first project x_i, x_j into a node embedding space, together with the goal features x_g, and their Euclidean (D_{EUC}) and cosine distances (D_{COS}). After that, using a 1-layer GNN, we perform a single convolution over each x_i and the corresponding neighborhood $\mathcal{N}_i$, to obtain g_i. Our graph convolution processing step allows us to easily incorporate edge features and work with variable sizes of $\mathcal{N}_i$. After the graph convolution, we apply the GRU module over each embedding g_i to obtain hidden states $z_{i,t+1}$, and new embeddings g'_i. We compute the sample mean of $z_{i,t+1}$ for each node $i \in \mathcal{V}_{new}$ to obtain a new hidden state z_{t+1}, and process g'_i with x_g using an MLP to compute the distances between the graph nodes.

Permutation invariant $\mathcal{V}_{new}$ embedding. There is a trade-off between processing new fringe nodes in batch, as in Algorithm 1, and processing them sequentially. Namely, when we process the nodes in

batch, we do not use the in-batch observations to predict batch node values, which means that z_t is slightly outdated. On the other hand, in PHIL, batch processing allows us to compute the heuristic values of all $v \in \mathcal{V}_{new}$ in parallel on a GPU and preserves the memory's permutation invariance with respect to nodes in $\mathcal{V}_{new}$. This approach provides additional scalability as we can process values in parallel and PHIL does not have to infer permutation invariance in $\mathcal{V}_{new}$ from data.

4 Experiments

Heuristic search in grids. In this section, we evaluate PHIL on 8, 200×200 8-connected grid graph-based benchmark datasets by Bhardwaj *et al.* [1]. Each dataset contains 200 training graphs, 70 validation graphs, and 100 test graphs. Example graphs from each dataset can be found in Table 1. For a detailed description of datasets and baselines, please refer to Appendix F.

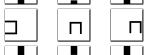
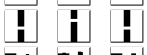
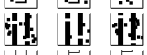
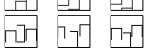
Dataset	Graph Examples	SaIL	SL	CEM	QL	h_{euc}	h_{man}	A*	MHA*	PHIL
Alternating gaps		1.000	11.077	1.077	25.641	25.641	25.641	25.641	25.641	**0.615**
Single Bugtrap		1.000	1.354	**0.361**	6.329	1.165	1.215	6.329	1.772	0.544
Shifting gaps		1.000	4.462	9.615	9.615	4.865	5.663	9.615	7.731	**0.260**
Forest		1.000	1.194	1.333	3.361	1.139	1.194	27.778	2.083	**0.778**
Bugtrap+Forest		1.000	2.612	1.238	6.803	2.789	2.293	6.803	3.177	**0.810**
Gaps+Forest		1.000	4.525	4.525	4.525	4.525	4.525	4.525	4.525	**0.213**
Mazes		1.000	2.311	4.650	3.874	1.796	1.660	9.709	2.709	**0.495**
Multiple Bugtraps		1.000	1.002	2.088	1.743	1.353	1.288	2.088	1.829	**0.382**

Table 1: The number of expanded graph nodes of PHIL with respect to SaIL. We can observe that out of all baselines, SaIL performs best. PHIL outperforms SaIL by 48.8% on average over all datasets, with a maximal search effort reduction of 78.7% in the *Gaps+Forest* dataset.

As we can see in Table 1, PHIL outperforms the best baseline (*SaIL*) on all datasets, with an average reduction of explored nodes before v_g is found of 48.8%. Even with *CEM* performing better than PHIL on *Single Bugtrap*, PHIL reduces the necessary search effort compared with the best baseline on each dataset by 40.8% on average. Qualitatively, observing Figure 2, we can attribute these results to PHIL's ability to **reduce the redundancy in explored nodes** during a search, as can be seen in Appendix A. Further, PHIL converges in up to $N = 36$ iterations, with $t_\tau = 32$ (i.e., after observing less than $N * t_\tau * max(|\mathcal{V}_{new}|) \approx 9,216$ shortest path distances, where we take $max(|\mathcal{V}_{new}|) = 8$ as the maximal size of $\mathcal{V}_{new}$). According to figures reported in [1], this is approximately $5\times$ less data than it takes for SaIL to converge. Although neither the SaIL or PHIL code-bases were optimized for runtime speed, we found that our implementation of PHIL runs about $7\times$ faster than the publicly available implementation of SaIL on the *Gaps+Forest* dataset.

Dataset	SL	A*	h_{euc}	BFS	PHIL	Shortest path
Room simple	1.124	76.052	1.000	291.888	**0.978**	0.938
Room adversarial	2.022	67.215	1.000	238.768	**0.895**	0.853

Table 2: Results of PHIL in the context of planning for indoor UAV flight. PHIL outperforms all baselines in both the *room simple* and *room adversarial* environments while remaining close performance-wise to the optimal number of expansions.

Planning for drone flight. In our final experiment, we use PHIL to plan collision-free paths in a practical drone flight use case within an indoor environment. For more detail about each environment, please refer to the supplementary material. We discretize the environments into 3D grid graphs of size $50 \times 50 \times 25$, and randomly remove 5 sub-graphs of size $5 \times 5 \times 5$ both during training and testing, this way simulating real-life planning scenarios with random obstacles. In Table 2 we report the ratio of expanded nodes with respect to h_{euc}. As we can observe in Table 2, PHIL outperforms all baselines in both environments. Interestingly, PHIL expands only approximately 4.2% more nodes in the simple room than least possible and 4.9% more in the adversarial room case. The same figures for the greedy method (h_{euc}) are 6.6% and 17.2%, respectively. These results indicate that PHIL is capable of learning planning strategies that are close to optimal in both *simple* and *adversarial* graphs[2], while the performance of naive heuristics degrades.

[2]We provide a video demonstration of PHIL running in room adversarial: `https://cutt.ly/eniu5ax`.

References

[1] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. "Learning heuristic search via imitation". In: *Conference on Robot Learning*. 2017.

[2] Binghong Chen et al. "Retro*: Learning retrosynthetic planning with neural guided A* search". In: *ICML*. 2020.

[3] Martin Gebser et al. "Domain-specific heuristics in answer set programming". In: *AAAI*. 2013.

[4] Thi Thoa Mac et al. "Heuristic approaches in robot path planning: A survey". In: *Robotics and Autonomous Systems*. 2016.

[5] Abhishek Sharma and Keith M. Goolsbey. "Identifying useful inference paths in large commonsense knowledge bases by retrograde analysis". In: *AAAI*. 2017.

[6] Cheng-Yu Yeh et al. "Pathway detection from protein interaction networks and gene expression data using color-coding methods and A* search algorithms". In: *The Scientific World booktitle*. 2012.

[7] Danish Khalidi, Dhaval Gujarathi, and Indranil Saha. "T*: A heuristic search based path planning algorithm for temporal logic specifications". In: *ICRA*. 2020.

[8] Bhargav Adabala and Zlatan Ajanovic. "A multi-heuristic search-based motion planning for autonomous parking". In: *30th International Conference on Automated Planning and Scheduling: Planning and Robotics Workshop*. 2020.

[9] Nir Lipovetzky and Hector Geffner. "Best-first width search: Exploration and exploitation in classical planning". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[10] Florent Teichteil-Königsbuch, Miquel Ramırez, and Nir Lipovetzky. "Boundary Extension Features for Width-Based Planning with Simulators on Continuous-State Domains." In: *IJCAI*. 2020, pp. 4183–4189.

[11] Jes ús Virseda, Daniel Borrajo, and Vidal Alcázar. "Learning heuristic functions for cost-based planning". In: *Planning and Learning*. 2013.

[12] Christopher Makoto Wilt and Wheeler Ruml. "Building a heuristic for greedy search." In: *SOCS*. 2015.

[13] Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Learning to rank for synthesizing planning heuristics". In: *IJCAI*. 2016.

[14] Jordan Thayer, Austin Dionne, and Wheeler Ruml. "Learning inadmissible heuristics during search". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. 2011.

[15] Soonkyum Kim and Byungchul An. "Learning heuristic A*: efficient graph search using neural network". In: *ICRA*. 2020.

[16] Sanjiban Choudhury et al. "Data-driven planning via imitation learning". In: *The International Journal of Robotics Research* 37.13-14 (2018), pp. 1632–1672.

[17] Kyunghyun Cho et al. "Learning phrase representations using rnn encoder–decoder for statistical machine translation". In: *EMNLP*. 2014.

[18] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *ICML*. 2017.

[19] Weihua Hu et al. "Open graph benchmark: Datasets for machine learning on graphs". In: *NeurIPS*. 2020.

[20] Prithviraj Sen et al. "Collective classification in network data". In: *AI magazine*. 2008.

[21] Oleksandr Shchur et al. "Pitfalls of graph neural network evaluation". In: *arXiv preprint arXiv:1811.05868*. 2018.

[22] Marinka Zitnik and Jure Leskovec. "Predicting multicellular function through multi-layer tissue networks". In: *Bioinformatics*. 2017.

[23] Christopher Morris et al. "Tudataset: A collection of benchmark datasets for learning with graphs". In: *arXiv preprint arXiv:2007.08663*. 2020.

[24] Geoff Boeing. "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks". In: *Computers, Environment and Urban Systems*. 2017.

[25] Miltiadis Allamanis. "The adverse effects of code duplication in machine learning models of code". In: *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 2019.

[26] Zhenqin Wu et al. "Moleculenet: a benchmark for molecular machine learning". In: *Chemical science*. 2018.

[27] Jiaxuan You, Rex Ying, and Jure Leskovec. "Position-aware graph neural networks". In: *ICML*. 2019.

[28] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *ACM SIGKDD*. 2016.

[29] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 2002.

[30] Sandip Aine et al. "Multi-heuristic A*". In: *The International booktitle of Robotics Research*. 2016.

[31] Edo Cohen-Karlik, Avichai Ben David, and Amir Globerson. "Regularizing towards permutation invariance in recurrent models". In: *NeurIPS*. 2020.

[32] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *NeurIPS Deep Learning Workshop*. 2013.

[33] Pieter-Tjerk De Boer et al. "A tutorial on the cross-entropy method". In: *Annals of operations research*. 2005.

[34] E. Rohmer, S. P. N. Singh, and M. Freese. "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework". In: *IROS*. 2013.

[35] Daniel Lenton et al. "Ivy: Templated deep learning for inter-framework portability". In: *arXiv preprint arXiv:2102.02886*. 2021.

[36] Petar Velickovic et al. "Graph attention networks". In: *ICLR*. 2018.

[37] Guohao Li et al. "Deepergcn: All you need to train deeper gcns". In: *arXiv preprint arXiv:2006.07739*. 2020.

[38] Petar Velickovic et al. "Neural execution of graph algorithms". In: *ICLR*. 2020.

[39] Petar Velickovic. *TikZ*. `https://github.com/PetarV-/TikZ`, last accessed on 01/6/21.