Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Speeding Up Search-Based Motion Planning using Expansion Delay Heuristics

Jasmeet Kaur, Ishani Chatterjee, Maxim Likhachev
The Robotics Institute, Carnegie Mellon University

Abstract

Suboptimal search algorithms are a popular way to find solutions to planning problems faster by trading off solution optimality for search time. This is often achieved with the help of inadmissible heuristics. Prior work has explored ways to learn such inadmissible heuristics. However, it has focused on learning the heuristic value as an estimate of the cost to reach a goal. In this paper, we present a different approach that computes inadmissible heuristics by learning Expansion Delay for transitions in the state space. Expansion Delay is defined as the number of states expanded during the search between expansions of two successive states. It can be used as a measure of the depth of local minima regions i.e., regions where the heuristic(s) are weakly correlated with the true cost-to-goal (Vats, Narayanan, and Likhachev 2017). Our key idea is to learn this measure in order to guide the search such that it reduces the total Expansion Delay for reaching the goal and hence, avoid local minima regions in the state space. We analyze our method on 3D (x, y, theta) planning and humanoid footstep planning. We find that the heuristics computed using our technique result in finding feasible plans faster.

Introduction

A* is a widely used search algorithm (Hart, Nilsson, and Raphael 1968) which starts with a node, generating a graph until the goal node is reached. To find an optimal solution faster, the search is guided by a heuristic function. The heuristic function for a search, $h(n)$ estimates the cost from node n to a goal node. An admissible heuristic never overestimates the path cost of any node to the goal. In other words, $h(n) \leq h^*(n)$ for any node n, where $h^*(n)$ is the cost of a least-cost path from node n to a goal node (Hart, Nilsson, and Raphael 1968). For many cases, the condition of admissibility can be relaxed and inadmissible heuristics are used to find a feasible path faster (Karpas and Domshlak 2012), (Scala, Haslum, and Thiébaux 2016), (Aine et al. 2016). In this paper, we present a novel learning-based method to compute inadmissible heuristics. Prior work ((Thayer, Dionne, and Ruml 2011), (Samadi, Felner, and Schaeffer 2008)) that explores learning-based approaches, learns a heuristic as an estimate of the cost to reach a goal. Differently, we learn an estimate of search effort spent in reaching a goal and guide the search in a way that reduces this effort. We achieve this by learning Expansion Delay for transitions in the state space (Valenzano and Xie 2016). Expansion Delay is defined as the number of states expanded during the search between expansions of two successive states. (Vats, Narayanan, and Likhachev 2017) shows that it can be used as a measure for depth of local minima regions i.e., regions where the heuristic(s) are weakly correlated with the true cost-to-goal. The aim of this paper is to identify these regions in the graph where the search ends up doing unnecessary exploration and use this information to avoid them.

Our contribution is a two-phase algorithm that computes heuristics by using Expansion Delay. In the first phase, i.e., the learning phase, we train a Feed-Forward Network to predict Expansion Delay with data obtained from prior planning experience. In the second phase, i.e., the planning phase, for a given planning query, we compute the heuristic with the help of Expansion Delay values predicted by the learned model and use it for planning. Experimentally, we show that this method of heuristic computation significantly improves planning time by avoiding regions of local minima for 3D (x, y, theta) planning and humanoid footstep planning.

Related Work

Prior work has studied ways to learn heuristics for achieving faster planning times. (Thayer, Dionne, and Ruml 2011) proposes a method of transforming admissible heuristics to inadmissible heuristics using learning. (Samadi, Felner, and Schaeffer 2008) presents a technique for combining an arbitrary number of features into a single cost-to-go estimate. They use an artificial neural network (ANN) to map these values to an estimate of the cost-to-go using $h^*(n)$ as the target value. (Bhardwaj, Choudhury, and Scherer 2017) presents an imitation learning-based approach to come up with heuristics that explicitly reduces search efforts. Our work is significantly different as we do not learn the heuristic as an estimate of the cost-to-go but as an estimate of search-effort-to-go.

We aim to use this estimate to avoid unnecessary exploration of local minima regions. Previous work has explored the idea of devising methods to avoid local minima regions. (Vats, Narayanan, and Likhachev 2017) uses knowledge from previously known planning episodes to come up with new adaptive motion primitives to circumvent local

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

minima regions for similar planning queries. (Phillips et al. 2012) and (Xu, Fern, and Yoon 2007) present techniques to compute heuristics to do so. Additionally, (Valenzano and Xie 2016) detects local minima or plateau regions and use random exploration. However, there is no previous work that learns Expansion Delay and uses it to compute heuristics. (Chatterjee et al. 2019) uses the property of conservativeness for transitions to minimize search efforts. The work presented in this paper utilizes a different property altogether. To the best of our knowledge, none of the prior work learns Expansion Delay to compute heuristics for reducing search efforts.

Definitions, Notations and Problem Description

Consider a graph $G = (S, E, c)$, where S is the set of states, $E \subseteq S \times S$ denotes the set of feasible transitions or edges in the graph and c is a cost-function such that $c(s_i, s_j)$ is the cost of the edge (s_i, s_j). A planning problem consists of finding a path $\pi(s_{start}, s_{goal})$ in G from s_{start} to s_{goal}. $\pi^*(s, s_{goal})$ denotes the least-cost path between s to s_{goal}. The cost of a path is sum of cost of all the edges in a path $\pi(s, s_{goal})$, denoted by $c(\pi(s, s_{goal}))$. Let $h_1 : S \longrightarrow N$ be a consistent heuristic function estimating cost-to-goal. A heuristic function is said to be consistent if $h_1(s_{goal}) = 0$ and for each $s_i \neq s_{goal}$ and each successor s_{succ} of s_i, $h_1(s_i) \leq h_1(s_{succ}) + c(s_i, s_{succ})$, where $c(s_i, s_{succ})$ is the cost of the edge between s_i and s_{succ}.

Expansion Delay Heuristics

Many planning problems use heuristics that are computed by solving simpler planning problems in an abstract space which is obtained by relaxing some constraints in the original space (Bulitko et al. 2007), (Holte et al. 1996). We take a similar approach and define an abstract space in the following manner.

Abstract Space

Let $\lambda : S \longrightarrow \tilde{S}$ be many-to-one mapping representing the projection of each state in S to an abstract space $\tilde{S}$, such that $|\tilde{S}| < |S|$. Moreover, $\lambda^{-1}(\tilde{s}) = s \in S | \lambda(s) = \tilde{s}$. For heuristic computation, a state s in S is projected onto $\tilde{S}$ using $\lambda(s)$ and the heuristic is computed by finding a path in this space.

The abstract space has its own set of transitions $\tilde{E} \subseteq \tilde{S} \times \tilde{S}$. Let the graph $\tilde{G}$ be defined by $\tilde{S}$ and $\tilde{E}$. $\pi(\tilde{s_i}, \tilde{s_j})$ denotes a path in $\tilde{G}$ from $\tilde{s_i}$ to $\tilde{s_j}$ and $c(\pi(\tilde{s_i}, \tilde{s_j}))$ denotes its cost in $\tilde{G}$. We assume states in the goal-set S_g map to one goal-state $\tilde{s_g}$ in the heuristic space, i.e., $\tilde{s_g} = \lambda(s_g) \forall s_g \in S_g$.

Expansion Delay

We also define a property called Expansion Delay (Vats, Narayanan, and Likhachev 2017) for the edges in graph G in the following manner:

- Let $e(s)$ be the total number of nodes that have been expanded before s is expanded during a search in graph G. Expansion delay Δe for (s, s') where s is the parent node and s' is its successor, is:

$$\Delta e(s, s') = e(s') - e(s) \tag{1}$$

- Let A be the set of state-pairs (s, s') in G that project to $(\tilde{s}, \tilde{s}')$ in $\tilde{G}$. For the edge $(\tilde{s}, \tilde{s}')$, $\Delta\tilde{e}(\tilde{s}, \tilde{s}')$ is the expected value of $\Delta e(s, s')$ over the set A:

$$\Delta\tilde{e}(\tilde{s}, \tilde{s}') = \mathbf{E}(\Delta e(s, s')) \tag{2}$$

A high expansion delay means that the search has spent a significant amount of time expanding states in a local minimum. Hence, Expansion Delay can be used as a measure of depth of local minima for a region in the state space (Vats, Narayanan, and Likhachev 2017).

Learning Expansion Delay Offline

The focus of this paper is to learn the expected Expansion Delay values for edges in graph $\tilde{G}$ for a given environment. For this, we train a Feed-Forward Network which takes these edges in $\tilde{G}$ as input features and Expansion Delay as target values. Since we are trying to learn the expected value of Δe, we can use *mean square error* (MSE) as the loss function (Goodfellow, Bengio, and Courville 2016). To obtain Expansion Delay values, we run a search (A* or Weighted A*) using a consistent heuristic in G and projecting these values from G to $\tilde{G}$.

Algorithm 1 shows how we obtain data for training. Let D be the set of input features and target value pairs for training. The training data is obtained by following steps :

- Randomly sample start and goal from $G : (S_{st}, S_g)$
- Let S_{sol} = $(s_1, s_2, ...s_N)$ be the ordered set of states expanded by the search for the given start-goal pair, on the solution path. For each pair of successive states, store the values of Expansion Delay $\Delta\tilde{e}$.
- For every abstract edge obtained by projecting states in S_{sol} to $\tilde{G}$, obtain features related to this edge, $(\tilde{s}, \tilde{s}')$ and chosen start and goal, and use Expansion Delay $\Delta\tilde{e}$ as the label.
- Add the information obtained for each abstract edge to the training dataset, D.

We now train a model using D. For a given start and goal, we can use Expansion Delay values predicted by the model to compute heuristic in the following manner.

Heuristic Computation

We want to compute a heuristic function such that it guides the search in graph G along paths that reduce the total Expansion Delay before reaching a goal state. This in turn means that we are less likely to encounter local minima regions and will therefore reduce our search effort spent in reaching the goal state.

Consider the graph $\tilde{G}$ = ($\tilde{S}$, $\tilde{E}$, $\tilde{c}$), defined for the abstract space. For any s$\in$ S and $\tilde{s}$ = λ(s), we define h_{e-d}(s) = c($\pi^*(\tilde{s}$, $\tilde{s_g}$)), where π^* is the optimal path in $\tilde{G}$ from $\tilde{s}$ to $\tilde{s_g}$.
Let $N(\tilde{s_i}, \tilde{s_j})$ be the value of expansion delay predicted by

Algorithm 1: Get Training Data (G)

1 Let n be the number of randomly selected (Start:S_{st} and Goal:S_g) pairs
  $i \leftarrow 1$
  while $i \leq n$ **do**
2 Search(S_{st}, S_g)
    for *each pair of successive states (s,s') on the found path* **do**
3 $\Delta\tilde{e} \leftarrow ExpansionDelay(\tilde{s}, \tilde{s}')$
      $D \leftarrow D \cup [(\tilde{S_{st}}, \tilde{S_g}, \tilde{s}, \tilde{s}'), \Delta\tilde{e}]$
4 **end**
5 increment i
6 **end**
7 **return** D

Algorithm 2: Heuristic Computation

1 OPEN = $\{\tilde{s}_g\}$ CLOSED = 0;
2 **while** $OPEN \neq \phi$ **do**
3 remove $\tilde{s}$ from OPEN with minimum $g(\tilde{s})$
    insert $\tilde{s}$ into CLOSED
    for *every predecessor* $\tilde{s}'$ *of* $\tilde{s}$ *s.t.* $\tilde{s}'$ *not in CLOSED* **do**
4 $N(\tilde{s}, \tilde{s}') \leftarrow prediction(\tilde{S}, \tilde{G}, \tilde{s}, \tilde{s}')$
      $\tilde{c}(\tilde{s}, \tilde{s}') \leftarrow N(\tilde{s}, \tilde{s}')$
      if $g(\tilde{s}') > g(\tilde{s}) + \tilde{c}(\tilde{s}, \tilde{s}')$ **then**
5 $g(\tilde{s}') = g(\tilde{s}) + \tilde{c}(\tilde{s}, \tilde{s}')$
        Insert $\tilde{s}'$ into OPEN with $g(\tilde{s}')$ as key
6 **end**
7 **end**
8 **end**

the model for the abstract edge $(\tilde{s_i}, \tilde{s_j})$. If $\tilde{c}(\tilde{s_i},\tilde{s_j})$ is the cost of an edge between $\tilde{s_i}$ and $\tilde{s_j}$,

$$\tilde{c}(\tilde{s_i}, \tilde{s_j}) = N(\tilde{s_i}, \tilde{s_j}) \quad (3)$$

Algorithm 2 shows the offline heuristic computation in detail. We compute the shortest path in $\tilde{G}$ from $\tilde{s_g}$ to every state in $\tilde{G}$. This is done by running a backward Dijkstra's search on $\tilde{G}$ from $\tilde{s_g}$ to every state in $\tilde{G}$. $\tilde{G}$ is implicitly constructed: for each expanded $\tilde{s}$ and predecessor $\tilde{s}'$, we assign costs according to Eq.(3).

For each state s generated by search in the original graph G, we first find its projection $\tilde{s} = \lambda$(s). $g(\tilde{s})$ which was updated when $\tilde{s}'$ was expanded in the heuristic computation search, is the cost of the shortest path in $\tilde{G}$. Therefore, $h_{ed}(s) = g(\tilde{s})$ where h_{ed} denotes Expansion Delay Heuristic.

Experimental Analysis

We now evaluate the effectiveness of Expansion Delay heuristic h_{ed}.

The Feed-Forward network used has three hidden layers and uses an adam optimizer with a learning rate of 0.001. We trained feature-target pairs taken from at least 25 random planning instances for each of different enviroments, the exact number varying by domain. We train the model untill it converged.

Figure 1: Results with h_b, h_c and h_{ed}.

Heuristic	No. of Expansions	Planning Time(s)	Solution Cost
h_b	1659±806	1.69±0.842	111±10
h_c	145±13	0.23±0.008	136±4
h_{ed}	113±20	0.19±0.04	105±16

Table 1: Comparison of h_{ed} with baselines h_b and h_c for planning in (x, y, theta)

Planning in 3D (x, y, theta) : **Abstract Space** : $\tilde{G}$ is defined by dropping the orientation, or in other words, $\lambda([x, y, theta]) = [x, y]$. To be exact, $\tilde{G}$ is a 2D 8-connected grid.

Results : We compare h_{ed} with two baseline heuristics : h_b is the regular euclidean heuristic and h_c is the conservative heuristic computed as in (Chatterjee et al. 2019). We use Weighted A* with $w = 100$ and h_b as the heuristic function to obtain training data D (Algorithm 1, Line 2).

Results (Table 1) show that h_{ed} has lesser number of state expansions, as compared to both h_b and h_c. It performs better than both in terms of planning times as well. Figure 1(left) shows a path found using euclidean heuristic, h_b. Figure 1(right) shows how h_{ed} finds a different path by avoiding the region explored by h_b. Additionally, it finds a path through the narrow passage using non-conservative edges as compared to h_c which uses only conservative edges (Figure 1(center)).

Humanoid Footstep Planning : For this domain, states = [x, y, θ, id] $\in$ S consists of position and orientation of the active foot (foot that is supposed to move next) and id of the active foot. The environment is divided into 400x400 cells and θ has a resolution of $45°$. The active foot moves relative to the pivot foot. For each foot as pivot, there are 12 feasible motions of the active foot in the form of $a = [\delta x, \delta y, \delta\theta]$ relative to pivot foot. The cost of transitions are proportional to euclidean distance between the centers of the feet.

Abstract Space : $\tilde{G}$ is a 2D 8-connected grid that corresponds to the position of the center of the robot in the map. λ projects the 2 feet-positions into the 2D grid by computing their mean.

Results : Experiments for this domain compare h_{ed} with a baseline heuristic h_b which is the cost of the optimal path in (x,y) space but with regular euclidean 2D edge-costs. We use A* with h_b as the heuristic function to obtain training data,

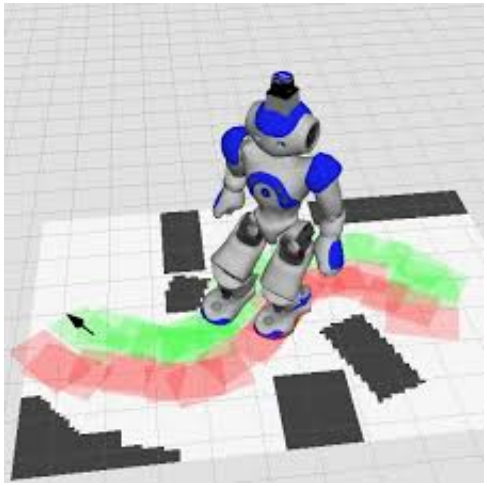

Figure 2: A humanoid footstep path in an enviroment (Garimort, Hornung, and Bennewitz 2011).

Heuristic	No. of Expansions	Planning Time(s)	Solution Cost
h_b	477777±75685	17.74±4.11	9226±754
h_{ed}	183174±120187	5.09±3.47	18951±4393

Table 2: Comparison of h_{ed} with baseline h_b for humanoid footstep planning using A*

D (Algorithm 1, Line 2). Table 2 compares performance of h_{ed} with h_b. h_{ed} reduces planning time by a factor of ≈ 3 and number of expansions by a factor of ≈ 2.5. However, to ensure that this behaviour does not result from using just a greedy heuristic and that h_{ed} is actually informative, we ran another set of experiments. In this case, we used Weighted A* with a weight of 100 and h_b as the heuristic function to obtain the training data, D. We not only compare h_{ed} with h_b but also compare its performance with Weighted A* with a really high weight (specifically, 10000). Table 3 shows these results. h_{ed} reduces planning time by a factor of ≈ 2 and number of expansions by a factor of ≈ 2. Figure 3(left) and 3(center) show a planning instance with Weighted A* using w = 100 and w = 10000 respectively. Figure 3(right) shows path found using h_{ed}. It can be seen that h_{ed} actually provides useful information to cirvumvent the local minima region and hence results in performing the best. It's important to note that the performance of h_{ed} would depend on the search being used for obtaining the training data.

Conclusions

We presented a novel approach for computing inadmissible heuristics based on Expansion Delay learned offline. The method shows significant improvements in planning time and number of state expansions for two different planning domains. Currently, we learn from planning instances over fixed environments. Future work includes finding a generalization that transfers learning across different enviroments effectively. It would also be interesting to see how simpler

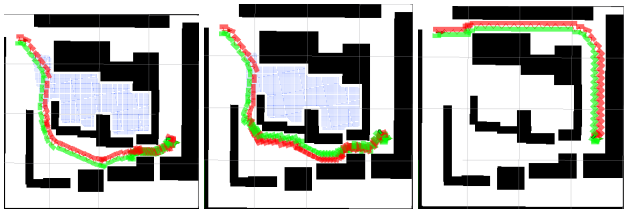

Figure 3: Results with h_b using w=100 (left), h_b using w=10000 (center) and h_{ed} (right). Left footsteps are marked as red, right footsteps are marked as green and states expanded during search are marked as blue.

Heuristic	No. of Expansions	Planning Time(s)	Solution Cost
$h_b(w = 100)$	183373±42156	5.08±1.439	13503±2641
$h_b(w = 10000)$	170072±43352	4.768±1.24	16362±3546
h_{ed}	89985±89400	2.28±2.28	14129±3937

Table 3: Comparison of h_{ed} with baseline h_b for humanoid footstep planning using Weighted A*

learning models perform in comparison to the neural net. Changing the weights in wA* could change the performance of h_{ed} and an analysis of the perfomance using different weights would be interesting future work as well. Additionally, we would like to use online learning for learning Expansion Delay.

Acknowledgements

This work was in part supported by ONR grant N00014-18-1-2775.

References

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2016. Multi-heuristic a. *The International Journal of Robotics Research* 35(1-3): 224–243.

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning heuristic search via imitation. *arXiv preprint arXiv:1707.03034* .

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research* 30: 51–100.

Chatterjee, I.; Likhachev, M.; Khadke, A.; and Veloso, M. 2019. Speeding up search-based motion planning via conservative heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 674–679.

Garimort, J.; Hornung, A.; and Bennewitz, M. 2011. Humanoid navigation with dynamic footstep plans. In *2011 IEEE International Conference on Robotics and Automation*, 3982–3987. IEEE.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

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.

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1-2): 321–361.

Karpas, E.; and Domshlak, C. 2012. Optimal Search with Inadmissible Heuristics. In *ICAPS*. Citeseer.

Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-Graphs: Bootstrapping Planning with Experience Graphs. In *Robotics: Science and Systems*, volume 5, 110.

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

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, 32283234. AAAI Press. ISBN 9781577357704.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Twenty-First International Conference on Automated Planning and Scheduling*. Citeseer.

Valenzano, R. A.; and Xie, F. 2016. On the completeness of best-first search variants that use random exploration. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Vats, S.; Narayanan, V.; and Likhachev, M. 2017. Learning to Avoid Local Minima in Planning for Static Environments. In *ICAPS*, 572–576.

Xu, Y.; Fern, A.; and Yoon, S. W. 2007. Discriminative Learning of Beam-Search Heuristics for Planning. In *IJCAI*, 2041–2046.