2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Learning Heuristic A*: Efficient Graph Search using Neural Network

Soonkyum Kim[1] and Byungchul An[2]

Abstract— **In this paper, we consider the path planning problem on a graph. To reduce computation load by efficiently exploring the graph, we model the heuristic function as a neural network, which is trained by a training set derived from optimal paths to estimate the optimal cost between a pair of vertices on the graph. As such heuristic function cannot be proved to be an admissible heuristic to guarantee the global optimality of the path, we adapt an admissible heuristic function for the terminating criteria. Thus, proposed Learning Heuristic A* (LHA*) guarantees the bounded suboptimality of the path. The performance of LHA* was demonstrated by simulations in a maze-like map and compared with the performance of weighted A* with the same suboptimality bound.**

I. Introduction

Finding the optimal, usually the shortest, path from the current pose to the goal pose is a well-known and actively studied problem in computer science, artificial intelligence, and robotics. The global optimal path can be generated by Dijkstra's algorithm [1], which explores vertices in the order of travel costs from the initial vertex until reaching the goal vertex. This algorithm guarantees not only the completeness but global optimality also. The graph search algorithm can reach the goal vertex more efficiently in terms of exploration by adapting a heuristic function to estimate the cost to the goal vertex [2]. The heuristic function is admissible if it does not overestimate the cost to the goal vertex. The admissible heuristic function guides the exploration by pulling to the goal vertex with proper strength to guarantee the global optimality. An arbitrarily heuristic function can pull the exploration with stronger force to find a path faster but eventually fails to guarantee the global optimality.

The level of deterioration, i.e. suboptimality, by adapting an arbitrary heuristic function can be bounded if one designs a heuristic function based on an admissible heuristic function. The search can be guided by a heuristic function inflating an admissible heuristic function by $\epsilon(> 1)$ times. Such heuristic function guarantees the bounded suboptimality that the cost of the generated path is not greater than ϵ times of the cost of the global optimal path [3]. It is also possible that the admissible heuristic function works as a pivot function to guarantee the bounded suboptimality, while the exploration is guided by arbitrarily multiple heuristic functions which help each other to escape local minima efficiently [4]–[6].

While the admissible heuristic function contributes to find the global optimal path efficiently, the heuristic function is usually designed conservatively not to overestimate by ignoring obstacles or infeasible regions in the map. As the heuristic function estimates the cost closer to the optimal cost, the graph search algorithm can find the path quickly by reducing the number of expansions [6], [7].

Machine learning algorithms have been adapted to simplify the motion planning problem by reducing the dimension of the search space [8], [9]. Also, by adding pre-known paths or set of feasible edges to calculate heuristic costs, the search is guided or intrigued to follow pre-known paths [10]. As the preprocessing of computing heuristic function considering given path segments is computationally expensive, this algorithm was modified to adapt multiple heuristic functions while simplifying the costly computation of each heuristic function [11]. Still, the graph search based method suffers from generating paths in real-time or close to real-time with increasing size of the graph or the map. To overcome this limitation, there have been various efforts and algorithms including training heuristic function. For example, the search is guided by the heuristic function trained to minimize the expansions of the vertices while not focusing on the cost or the optimality of the generated path [12].

In this aspect, recent achievements and developments in the neural network are noteworthy. It allows more layers by backpropagation algorithm and adaption of various activation functions. These academic accomplishments influenced various fields in artificial intelligence and robotics. There has been huge innovation in the field of vision and recognitions [13]. Moreover, adapting deep neural network in reinforcement learning allows solving various problems in navigation and planning. The autonomous mobile robot can navigate the hallway with a mono-camera while learning how to avoid collision by increasing traveling distance with reinforcement learning [14]. Also, a mobile robot can navigate on a square grid with limited field of view or partial information by limitation of sensor range [15].

In this work, we model the learning heuristic function as a neural network [16]. If the search is guided by the exact optimal cost to the goal vertex, we can find the global optimal path while avoiding unnecessary expansions of vertices. However, in the presence of obstacles and constraints, it is practically impossible to calculate the exact distance or cost between arbitrary vertices without solving another path planning problem. Thus, we propose a method of estimating the distance or cost using a neural network. To train this neural network, the training set of the optimal paths is generated by solving path planning problems with random boundary conditions. Then, the trained neural network will serve as the heuristic function to guide search finding paths

*This work was supported by KIST Institutional Program (2E30260).
[1]Soonkyum Kim is Senior Research Scientist of Robot Media Institute at Korea Institute of Science and Technology, Republic of Korea kim.soonkyum@kist.re.kr. [2]Byungchul An is Post Doctoral Researcher of Robot Media Institute at Korea Institute of Science and Technology, Republic of Korea byungchul.an@kist.re.kr

between arbitrary pairs of vertices. The proposed Learning Heuristic A* guarantees the bounded suboptimality of the path by adapting an admissible heuristic function for the terminating criteria. The performance of the proposed algorithm is demonstrated in a maze-like environment, where the difference between the admissible heuristic cost and the optimal cost is larger than an open or empty environment.

II. Problem Definition

In this paper, we consider the path planning problem on a graph, $G = (V, E)$, where V is the set of feasible vertices and E is the set of edges connecting pairs of vertices in V, whose costs are $c(\cdot, \cdot)$. Usually, each feasible vertex corresponds to an obstacle-free configuration in a map or an environment. A feasible path from $v_i \in V$ to $v_g \in V$ can be represented as a sequence of vertices in V, $\pi = (v_0, v_1, v_2, ..., v_{N-1}, v_N)$, where $v_0 = v_i$ and $v_N = v_g$, while there exist all the edges connecting adjacent vertices, $\{v_{j-1}, v_j\} \in E$ for $\forall j \in [1, 2, ..., N]$. The cost of the path is defined as the sum of the cost of all the edges in the path, $\mathrm{g}(\pi) = \sum_{j=1}^{N} c(\{v_{j-1}, v_j\})$. Moreover, the path is the optimal path, π^*, connecting v_i and v_g if it is feasible and the cost of the path is not greater than the cost of any other feasible paths, $\mathrm{g}(\pi^*) \leq \mathrm{g}(\pi)$ for all feasible paths, π, with the same boundary condition. Also, let $\mathrm{g}^*(v_i, v_g) = \mathrm{g}(\pi^*)$. To find the optimal path, all the candidate vertices should be evaluated or explored to guarantee the global optimality. To reduce such evaluations, some variational graph search algorithms of A* have been developed to guarantee the bounded suboptimal path, of which the cost is not greater than ϵ times of the cost of the optimal path [3], [4]. In this work, the proposed Learning Heuristic A* (LHA*) algorithm is designed to reduce the computation load by adapting a trained neural network as the learning heuristic function while guaranteeing the bounded suboptimality.

III. Algorithm

In general the heuristic function estimates the cost from a given vertex, v, to the goal vertex, v_g, in the graph, which should be designed not to estimate the cost greater than the optimal cost connecting the same pair of vertices to guarantee the global optimality of the generated path, $h(v, v_g) \leq g^*(v, v_g)$. To satisfy this constraint, the most common heuristic functions in the path planning problem are Euclidean distance and Manhattan distance which are the shortest distance between two configurations in the Cartesian coordinate system. As these trivial heuristic functions do not consider the obstacles or topology of the map, there could be a difference between the estimated cost and the optimal cost. As illustrated in the previous literature, we can decrease unnecessary expansions of vertices by reducing the difference between the estimated cost and the optimal cost [6], [7]. To guide the search to the goal vertex more efficiently, a heuristic function is required that can estimate the cost between the pair of vertices closer to the optimal cost than trivial heuristic functions. Moreover, by adapting an admissible heuristic function to determine the termination condition of the search, the proposed algorithm guarantees the bounded suboptimality of the generated path.

A. Learning Heuristic Function

The heuristic function can be considered as a mapping from a pair of vertices to a cost variable,

$$(v_j, v_k) \mapsto h(v_j, v_k, \omega) \tag{1}$$

where $\omega \in \mathbb{R}^m$ is design parameter, $v_j, v_k \in \mathbb{R}^n$ and $h \in \mathbb{R}$, which imitates the optimal cost between the pair of vertices. Instead of satisfying constraints to be admissible heuristic, the proposed algorithm focuses on reducing the error between the estimated cost and the optimal cost.

The obstacles or infeasible regions of the map cause the optimal path to take a detour or be bent so that the optimal cost, g^*, between a pair of vertices to be a nonlinear and non-convex function. Unless an algorithm searches the optimal path of every vertex on the map to an arbitrary goal vertex, a discrepancy between heuristic h and optimal cost g^* is inevitable. Instead of exploring the whole vertices of the map, we will estimate the optimal cost $\tilde{g}$ from proper samples. We generate a set of optimal paths to collect a training set with pairs of vertices and the optimal cost between each pair, then $\tilde{g}$ is trained using a neural network. We adapt the estimation of the optimal cost $\tilde{g}$ as a learning heuristic function $h_l = \tilde{g}$ to guide exploration efficiently toward the goal vertex.

A pair of vertices, (v_j, v_k), is the input of the neural network. If the given map or the environment is represented as a directed graph, the order of vertices should be considered. Otherwise, the order of the vertices could be ignored. The cost of the optimal path connecting the pair of vertices is the output of the neural network. Thus, the input layer has input nodes twice of the dimension of workspace or vertex and the output layer has a single output, usually. The leaky rectified linear unit is used as the activation function except the output layer to get a non-zero gradient when the neuron is not activated. For the output layer, the normal rectified linear unit is adapted as the estimated cost should be non-negative.

To train the neural network, we need to build a training set which covers the most part of the map or the graph to avoid extrapolation which leads to poor estimation. If a path $\pi = (v_0, v_1, ..., v_N)$ is an optimal path connecting v_0 and v_N, any segment of π, $\pi_{jk} = (v_j, ..., v_k)$, where $0 \leq j < k \leq N$, is also an optimal path connecting v_j and v_k. Therefore, with an optimal path with $N+1$ vertices like the above example π, we can extract $\binom{N+1}{2} = \frac{(N+1)N}{2}$ pairs of vertices and corresponding optimal costs. Optimal paths to build training set will be generated by graph search algorithms like A*.

Error function to train the neural network is defined by

$$\min_{\omega, b} \sum_{k} \left(1 - \frac{\tilde{g}_k}{g_k}\right)^2 \tag{2}$$

where (ω, b) are parameters of the neural network, g_k is the optimal cost of the k_{th} data in the training set and $\tilde{g}_k$ is the corresponding estimated cost. Usually, the object function is defined as $\frac{1}{2} \left|\left|g - \tilde{g}\right|\right|^2$ to achieve the estimation $\tilde{g}$ closer to g.

However, since the cost of the path increases as the length of the path gets longer in general, the cost of a shorter path will not be taken into account fairly if we use the error function as usual. Especially, as the optimal cost becomes smaller as getting closer to the goal vertex, the usual error function can cause the estimated cost, which will serve as the heuristic, to lead path wandering around the goal. Hence the proposed error function (2) is used for training the neural network.

B. Learning Heuristic A*

The proposed LHA* algorithm is a variation of A* while guaranteeing the bounded suboptimality, which means the cost of the generated path is not greater than ϵ times of the cost of the global optimal path connecting the same vertices.

The following is the pseudo code of the proposed LHA*. From given initial vertex, v_i, and the goal vertex, v_g, we will utilize the learning heuristic function, $h_l(\cdot,\cdot)$, to guide the search efficiently and the admissible heuristic function, $h_0(\cdot,\cdot)$, to generate the path with the bounded suboptimality. Function $g(v)$ denotes the current best cost from v_i to v during searching. In the above Algorithm 1, $OPEN.pop()$

Algorithm 1 Learning Heuristic A*(v_i, v_g)

```
1:  $g_g = \infty,\ g(v) = \infty\ \forall v,\ bp(v_i) = bp(v_g) =$ null;
2:  $g(v_i) = 0$;
3:  $f(v_i) = g(v_i) + h_l(v_i, v_g)$;
4:  $OPEN = \emptyset$;
5:  $CLOSED = \emptyset$;
6:  insert $v_i$ into $OPEN$
7:  while $OPEN$ not empty do
8:      $v = OPEN.pop()$;
9:      insert $v$ into $CLOSED$
10:     if $v = v_g$ and $g(v) < g_g$ then
11:         $g_g = g(v)$;
12:         $bp(v_g) = bp(v)$;
13:     if $g_g \leq \epsilon\ minCost(OPEN, v_g)$ then
14:         terminate and return path pointed by $bp(v_g)$;
15:     for all $v' \in succ(v)$ do
16:         if $v' \notin CLOSED$ or $g(v') > g(v) + c(v, v')$ then
17:             $g(v') = g(v) + c(v, v')$;
18:             $f(v') = g(v') + h_l(v', v_g)$;
19:             $bp(v') = v$;
20:             insert $v'$ into $OPEN$
21: return empty path;
22: procedure minCost($S$, $v_g$)
23: $f_{min} = \infty$;
24: for all $v \in S$ do
25:     $f_{min} = min(f_{min}, g(v) + h_0(v, v_g))$;
26: return $f_{min}$
```

returns the element, v, in the set $OPEN$ having the smallest value of $f(v)$ and removes it from the set $OPEN$, the procedure **minCost(S, v_g)** finds the vertex in S which has the smallest sum of the current best cost from the initial vertex and the estimated cost to the goal vertex by the admissible heuristic, $g(v) + h_0(v, v_g)$, to return this value. This algorithm works similar to A* while expansions of vertices are guided by a learned heuristic, $h_l(\cdot,\cdot)$, which does not need to be admissible. However, the proposed LHA* algorithm guarantees the bounded suboptimality by adapting an admissible heuristic function for termination criteria at line 13.

As LHA* is based on Shared Multi-Heuristic A* [4], while adapting the resultant of the trained neural network as the unique additional heuristic function, it inherits the propitious properties. Thus, the proposed LHA* guarantees the bounded suboptimality. The efficiency and performance of the proposed LHA* will be demonstrated by numerical simulations in the following Section.

IV. Simulation Result

To demonstrate the performance of the proposed LHA*, numerous simulations are performed to find the shortest path between a pair of cells in a maze-like map, where the robot or the agent should detour obstacles or walls to cause large difference between the cost of the optimal path and common admissible heuristic costs like Euclidean distance or Manhattan distance. A maze-like map with the size of 201×201 shown in Fig. 2 is built where the white cells are feasible and the black cells are infeasible or obstacles. Then a graph is built to have set of vertices of feasible cells and a set of edges in the manner of 4-way connected graph, the robot or the agent can make four kinds of movements at one step, i.e. moving up, down, left or right to the neighbor vertices or cells. The cost of the edge is defined as unit distance and Manhattan distance of

$$h_0(v_j, v_k) = |v_{j,x} - v_{k,x}| + |v_{j,y} - v_{k,y}| \tag{3}$$

is used as the admissible heuristic function for all the graph search algorithms throughout simulations.

As LHA* guarantees the bounded suboptimality, performances are compared with weighted A* which also guarantees the bounded suboptimality. The parameters of weighted A* and LHA* are designed to have the same bounded suboptimality of $\epsilon = 10$ in this simulation. As the reference costs and performances are necessary, the global optimal paths were generated by the original A* algorithm.

To consider the path planning problem on a plain, the neural network of the learning heuristic has four inputs of x and y coordinates of two vertices, $v_j, v_k \in \mathbb{R}^2$, and one output of the estimated cost. The hidden layers are designed to be fully connected with 16 hidden layers with 200 neurons for each layer.

To train the neural network of the learning heuristic function, by solving $100,000$ path planning problems with random boundary conditions with A*, we accumulate a large set of pairs of vertices and corresponding optimal costs. We extracted a small random subset as a test set to compare the performances of trained neural networks while the complement subset is used as the training set. Fig. 1(a) shows the training result of the neural network, where red dots are the minimum or maximum estimated costs among the cases with the same optimal cost, blue dots are the mean value of estimated costs with the same optimal cost. The green dashed line represents the case that the estimated cost is the same with the optimal cost, $\tilde{g}_k = g_k$. The blue plot, the mean value, in Fig. 1(a) approaches closer to the green dashed line as the optimal cost getting smaller, which is consistent with the intention to design the weighting parameter of the error function (2). Therefore, good estimated costs around the goal vertex are expected when exploring the graph. The average error, $\left|1 - \frac{\tilde{g}_k}{g_k}\right|$, for the test set is 0.062, which means the trained neural network estimates the optimal cost precisely to guide the graph search toward the goal vertex efficiently and successfully.

The performance of the proposed LHA* algorithm was demonstrated by two evaluation parameters, the ratio of the number of expansions and the ratio of the cost of the path with respect to those values of A*.

$$r_e^l = \frac{\text{\# of expansions by LHA*}}{\text{\# of expansions by A*}}$$
$$r_c^l = \frac{\text{cost of path by LHA*}}{\text{cost of path by A*}}$$

while r_e^w and r_c^w are the same measures of the weighted A*. The ratio of the number of expansions measures the efficiency and the ratio of the cost of the path measures the quality of the generated path.

Also, the ratio of the computation time to solve the path planning problem can be considered. However, computation time depends on various factors like code implementation, compile options, power management of CPU and existence/performance of GPU in case of LHA*, while having a strong correlation with the number of expansions. Therefore, the number of expansions can be more adequate object measure than computation time, so the above two measures are considered to demonstrate the performance.

To compare the performance of the LHA* statistically, $10,000$ shortest path planning problems are solved with random boundary conditions. Each problem is solved by A*, weighted A* and LHA*. The plots in Fig. 1(b)-(e) illustrate the simulation result for each graph search algorithm. In Fig. 1(b)-(e), x-axis and y-axis are the number of expansions and the cost of the path, and the green, red and blue dots represent the performance of each path planning problem solved by A*, weighted A* and LHA*, respectively.

Comparing to the green dots in Fig. 1(b), the performance of reference algorithm A*, the red dots, weighted A*, in Fig. 1(c) tend to shift to the left side, which means less number of expansions, and upward, which means higher costs of paths. This result coincides with the characteristic of weighted A*, which guides expansions of vertices strongly toward the goal vertex to reduce the number of expansions while sacrificing the optimality of the path up to a certain ratio, ϵ. However, the blue dots, LHA*, in Fig. 1(d) are squeezed to the left while not moving upward. In other words, LHA* expands less vertices than A* while the cost of the path is similar to that of the global optimal path.

The plots in Fig. 3 show details of the simulation result. The blue dots in Fig. 3(a) are the number of expansions by LHA*. The green dashed line represents the cases which have the same number of expansions with reference A*. Most of the blue dots lie under the green dashed line, i.e. in most cases, LHA* expands less vertices than A*. Based on our simulation, the mean of r_e^l is 0.497, so on average LHA* expands only half of the vertices compared with A*. The worst case value of r_e^l is 1.152 and LHA* expands only 15% more vertices than A*. Among $10,000$ simulations, LHA* expanded more vertices than A* by only 32 cases, less than 1%. As expected, LHA* expands less vertices than A* very consistently and steadily. The red dots in Fig. 3(b) show the number of expansions by weighted A*. The mean and worst value of r_e^w are 0.400 and 2.161, respectively. Comparing plots in Fig. 3(a) and Fig. 3(b), more red dots lie above the green dashed line than the blue dots, i.e. weighted A* expands more vertices than A* by 197 cases.

TABLE I

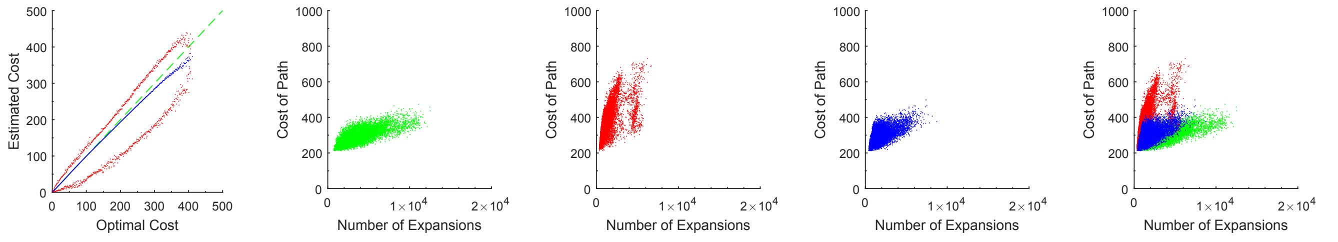
STATISTICAL ANALYSIS OF EVALUATION PARAMETERS OF LHA* AND WEIGHTED A* FOR $10,000$ EVALUATIONS.

measure		Learning Heuristic A*	weighted A*
r_c	mean	1.004 ± 0.009	1.266
	max	1.100	2.460
r_e	min	0.154	0.092
	mean	0.497 ± 0.139	0.400
	max	1.152	2.161

The blue dots in Fig. 3(c) represent the cost of the path generated by LHA*. As the reference algorithm A* always finds the global optimal solution, the blues dots cannot lie under the green dashed line. However, the blues dots lie very close to or right above the green dashed line, i.e. the cost of the path by LHA* is the same as or quite close to the global optimal cost. The simulation data shows that the mean and worst case value of r_c^l are 1.004 and 1.100, respectively. Also, the cost of the path by LHA* is the same with the global optimal cost for $7,027$ cases among $10,000$ simulations, 70%. The simulation result shows that LHA* finds the global optimal path or at least very close path to the global optimal path consistently and steadily. However, the red dots in Fig. 3(d) represent the cost of the path by weighted A*. The red dots are way above the green dashed line, and the weighted A* finds the path with global optimal cost 117 cases among $10,000$ simulations, about 1.2%.

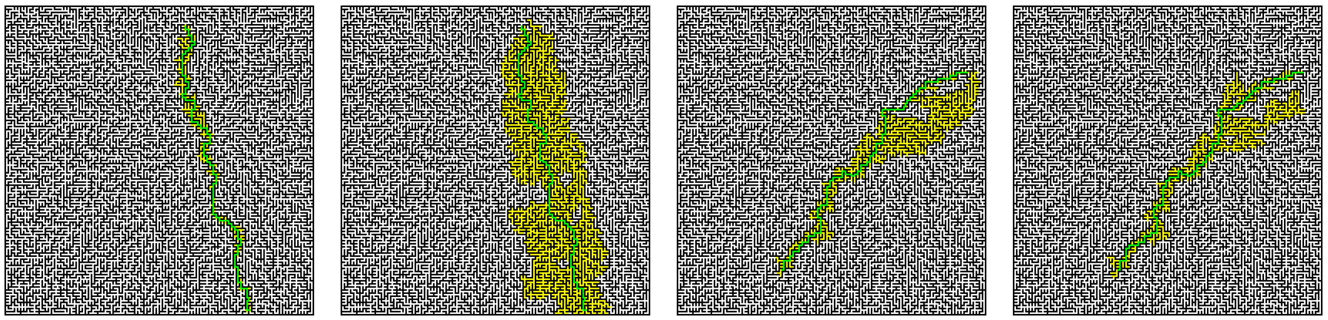
Selected examples of simulations are shown in Fig. 2 which compare the best and the worst case of LHA* and A* according to the measure r_e^l. For the best cases, costs of resultant paths are exactly the same, while the set of expanded vertices of LHA* is much smaller than that of A*. For the worst cases, however, as LHA* takes detours in some path segments, more expansions are attempted. Nevertheless, the trained neural network guides the path not to stray far from the optimal path, avoiding unnecessary expansions.

Table I summarizes the statistical analysis of the simulation by two graph search algorithms. Considering the statistical analysis in Table I and the plots in Fig. 3, weighted A* algorithm expands less vertices than LHA* algorithm on



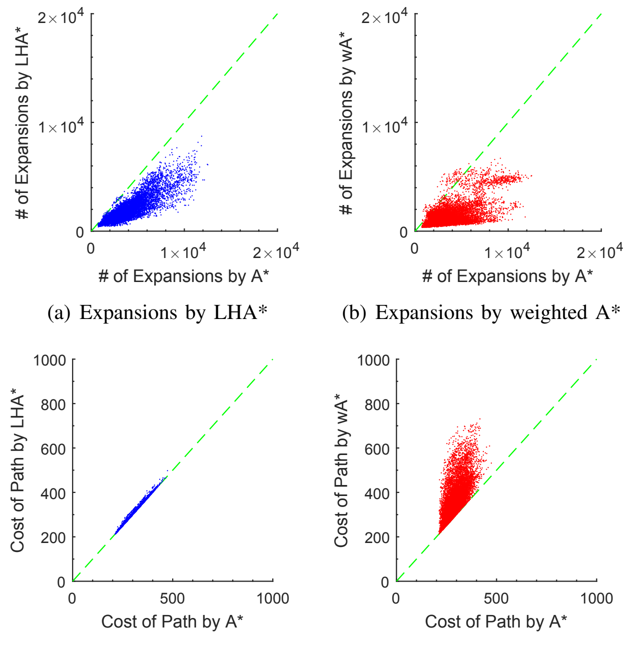
(a) The estimated cost by trained neural network. (b) Performance of A* (c) Performance of weighted A* (d) Performance of LHA* (e) Performance of all the algorithms

Fig. 1. The plots in (a) are comparison of the estimated cost by trained neural network with the optimal cost. The x-axis and y-axis represent the optimal and the estimated cost, respectively. The red dots are the minimum or maximum estimated costs and the blue dots are the mean values among the same optimal cost. The green dashed line represents the case that the estimated cost is the same with the optimal cost, $\tilde{g}_k = g_k$. The plots in (b)-(e) are comparisons of the performances of graph search algorithms. The x-axis and y axis are the number of expansions and the cost of the path, respectively. The green, red and blue dots represent the performance of A*, weighted A* and LHA*, respectively.



(a) Best r_e^l case by LHA* (b) Best r_e^l case by A* (c) Worst r_e^l by LHA* (d) Worst r_e^l by A*

Fig. 2. Paths and expanded vertices of the best and worst cases, where the green and yellow cells are the generated paths and the expanded vertices, respectively. The best case means r_e^l is the smallest among the cases of $r_c^l = 1$. The worst case means r_e^l is the largest.

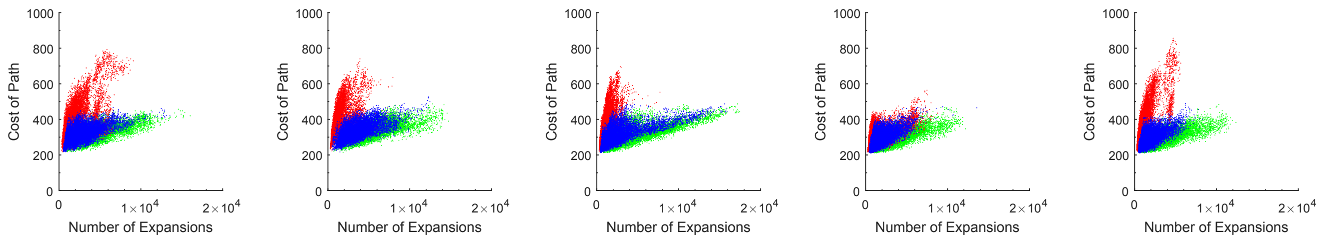


(a) Expansions by LHA* (b) Expansions by weighted A* (c) Cost of path by LHA* (d) Cost of path by weighted A*

Fig. 3. Performance comparison between LHA* (blue) and weighted A* (red). (a) and (b) are the number of expansions. (c) and (d) are costs of paths. The green dashed lines represent the cases that have the same performance with the reference algorithm, A*.

average, $0.4 < 0.497$, and for the best case, $0.092 < 0.154$, while sacrificing the optimality of the path up to more than twice. However, LHA* generates a path with cost equal or close to the global optimal cost while expanding the only half number of vertices than A* on average. Also, considering the worst case performance, LHA* outperformed weighted A*. So we conclude that proposed LHA* is a reliable algorithm to find paths of good quality (equal to or close to the optimal cost) consistently and steadily.

To demonstrate the robustness of the LHA*, we performed additional simulations on modified maps. We switched N feasible cells in the map to infeasible, also N infeasible cells or obstacles to feasible. These cells are chosen randomly among the cells which can change the topology of the map. The same simulations were performed for each $N = \{30, 50, 100\}$ with $10,000$ random boundary conditions while removing trivial case that the optimal path is not affected by the change of the map. The plots in Fig. 4 shows the performance result. As the map changes more, larger N, the performance of LHA* decreases slightly. However, still LHA* shows robust performance of mean and maximum of r_e^l for $N = \{30, 50, 100\}$ are $(0.618, 0.657, 0.561)$ and $(1.231, 1.200, 1.161)$, respectively. Also, the worst value of r_c^l is 1.222 for all the cases. Therefore, we can conclude that the proposed LHA* performs robustly with a change of the map or environment so that can be used for the path planning in dynamic environments.



(a) $N = 30$ (b) $N = 50$ (c) $N = 100$ (d) $\epsilon = 2$ (e) $\epsilon = 100$

Fig. 4. Performance comparisons for additional simulations. x-axis and y-axis are the number of expansions and the cost of the path, and the green, red and blue dots represent the performance of A*, weighted A* and LHA*, respectively. (a)-(c) are performances in modified maps with different Ns. (d) and (e) are performances with different suboptimality bounds, ϵ.

Another propitious feature of LHA* is that the optimality of the generated path is insensitive to the change of suboptimality bound ϵ. The plots in Fig. 4(d) and Fig. 4(e) shows the simulation result of $10,000$ path planning problems for $\epsilon = 2$ and $\epsilon = 100$, respectively. Comparing to the result of $\epsilon = 10$ in Fig. 1(e), distributions of blue plots are consistent, which means the performance of LHA* is robust with respect to the change of ϵ, while distributions of red plots are sensitive to the change of parameter ϵ. Thus, the suboptimality bound should be sagaciously chosen to achieve desired performance when using weighted A*. On the other hand, LHA* will perform robustly and consistently, users are not required to spend time on tuning parameters.

In this Section, the performance of LHA* was demonstrated with numerous simulations to show the efficiency, consistency and robustness.

V. Conclusion

In this work, we proposed the algorithm of Learning Heuristic A* to solve the path planning problem on a graph, which trains the heuristic function with the training set of optimal paths. Instead of adapting designed admissible heuristic function, the expansions of vertices are guided by the trained neural network to reduce unnecessary explorations. By adapting an admissible heuristic function for the terminating condition, LHA* guarantees the bounded suboptimality like other variations of A*, for example, weighted A*. The performance of LHA* was demonstrated by simulations and compared with weighted A* to find paths of good quality consistently and steadily while efficiently exploring the graph. The algorithm also works robustly with the partial change of the map and the change of the suboptimality bound. To apply LHA* to practical applications more efficiently, we need to find the optimal design of a neural network of the learning heuristic function and the proper size of the training set corresponding to the map and the neural network, which are remained for future work.

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec. 1959.

[2] P. E. H. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, pp. 100–107, July 1968.

[3] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artif. Intell.*, vol. 1, no. 3–4, pp. 193–204, 1970.

[4] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a*," in *Proc. Roboti.: Sci. Syst.*, 2014.

[5] F. Islam, V. Narayanan, and M. Likhachev, "Dynamic multi-heuristic a*," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 2376–2382, 2015.

[6] S. Kim and M. Likhachev, "Path planning for a tethered robot using multi-heuristic a* with topology-based heuristics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 4656–4663, 2015.

[7] S. Kim, S. Bhattacharya, H. Heidarsson, G. Sukhatme, and V. Kumar, "A topological approach to using cables to separate and manipulate sets of objects," in *Proc. Robot.: Sci. Syst.*, June 24-28 2013.

[8] S. Kim and F. C. Park, "Fast robot motion generation using principal components: Framework and algorithms," *IEEE Trans. Ind. Electron.*, vol. 55, no. 6, pp. 2506–2516, 2008.

[9] B. An, H. Kang, and F. C. Park, "Grasp motion learning with gaussian process dynamic models," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, pp. 1114–1119, 2012.

[10] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs," in *Proc. Roboti.: Sci. Syst.*, 2012.

[11] S. Aine, C. Sharma, and M. Likhachev, "Learning to search more efficiently from experience: A multi-heuristic approach," in *Proc. Int. Symp. Comb. Search*, 2015.

[12] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," in *Proc. Conf. Robot. Learn.*, 2017.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, 2012.

[14] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 1–8, 2018.

[15] A. I. Panov, K. S. Yakovlev, and R. Suvorov, "Grid path planning with deep reinforcement learning: Preliminary results," *Procedia Computer Science*, vol. 123, pp. 347–353, 2018.

[16] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.