Learning to Search More Efficiently from Experience: A Multi-Heuristic Approach

Sandip Aine
Indraprastha Institute of Information Technology, New Delhi, India
sandip@iiitd.ac.in

Charupriya Sharma
Indraprastha Institute of Information Technology, New Delhi, India
charupriya11037@iiitd.ac.in

Maxim Likhachev
Carnegie Mellon University
Pittsburgh, PA, USA
maxim@cs.cmu.edu

Abstract

Learning from experience can significantly improve the performance of search based planners, especially for challenging problems like high-dimensional planning. Experience Graph (E-Graph) is a recently developed framework that encodes experiences, obtained from solving instances in the past, into a single bounded-admissible heuristic, and uses it to guide the search. While the E-Graph approach was shown to be very useful for repetitive problems, it suffers from two issues. First, computing the E-Graph heuristic is time consuming as it maintains the bounded admissibility constraints. Second, a single heuristic can get stuck in a local minimum, and thereby, degrade the performance. In this work, we present an alternative approach to improving the runtime of search from experience, based on a recently developed search algorithm Multi-heuristic A* (MHA*). This framework provides an improvement over the E-Graph planner for two reasons: a) MHA* uses multiple heuristics simultaneously to explore the search space, which reduces the probability of getting stuck in a local minimum, and b) the heuristics in MHA* can be arbitrarily inadmissible, which makes it very easy to compute them. The paper describes the framework, explains how to compute these (inadmissible) heuristics through offline and online processing and presents experimental analysis on two domains, motion planning for a 6D planar arm and large sliding tile puzzles.

Introduction

Planners are often asked to solve a set of problems that have some inherent similarities, for example, a robot may be required to perform several manipulation tasks in a given environment with some static and some dynamic obstacles, or we may design a planner to solve a fixed sized sliding tile puzzle problem. In such cases, the information acquired by solving a subset of instances may improve the planner. In other words, a planner can learn from experience, and thereby improve its performance.

For graph search based planning, one way to incorporate the learned information is to encode it as a heuristic function. One of the recent approaches that does so is called planning with Experience Graphs (E-Graphs) (Phillips et al. 2012). An Experience Graph G^E is essentially a collection of previously planned paths. The E-graph planner uses weighted A* (WA*) (Pohl 1970) to search the original search space G, but tries to maximally reuse paths in G^E. This is achieved by modifying the heuristic function, which now drives the search toward paths in G^E, i.e., the E-Graph heuristic (h^E) prioritizes the paths on G^E over G (by penalizing paths that are not part of G^E). This heuristic is computed in a manner that preserves a chosen bounded admissibility/consistency (i.e., $h^E < \epsilon_1 * h_0$, where h_0 is a consistent heuristic for the original problem and ϵ_1 is constant ≥ 1), which in turn ensures that the solutions obtained using the E-Graph planner are provably bounded sub-optimal. Planning with E-Graphs has been shown to be very successful for high dimensional problems, such as single arm planning or full body manipulation (Phillips et al. 2012). It has been also extended to run as an anytime and incremental planner (Phillips et al. 2013a), to learn from demonstrations (Phillips et al. 2013b).

While the E-Graph based planning can effectively use experience to enhance performance, it suffers from two problems. First, the E-Graph planner uses the planning experience (and a given consistent heuristic function) to compute a single heuristic function. Some recent approaches (Röger and Helmert 2010; Aine et al. 2014) have shown that performance of search algorithms can be improved considerably if we simultaneously search with multiple heuristics instead of relying on a single heuristic function. This is especially true for hard planning problems (such as high-dimensional planning) where computing a *single good* heuristic is often difficult. The same phenomenon can be observed in case of the E-Graph planner, as it can get stuck if the experience based heuristic leads the search to a local minimum. Second, the computation and update of the E-Graph heuristic is costly as it needs to ensure the bounded admissibility/consistency constraints. In particular, h^E for a given state s_0 is computed using the following equation,

$$h^E(s_0) = min_{\Pi} \Sigma_{i=0}^{i=N-1} min\{\epsilon_1 * h_0(s_i, s_{i+1}), c^E(s_i, s_{i+1})\} \quad (1)$$

where Π is a path $s_0 ... s_{N-1}$ and $s_N = s_{goal}$, c^E denotes the cost of an edge in the E-Graph (G^E), h_0 is a consistent heuristic function and ϵ_1 is a scalar ≥ 1, which is used to penalize the paths that are not in the E-Graph. Now, this computation does not impose much extra penalty when computed as a one shot algorithm on a pre-built graph (such as 2D and 3D Dijkstra based heuristics for motion planning

This work was supported by NSF Grant IIS-1409549 and by ARL, under the Robotics CTA program grant W911NF-10-2-0016.

problems), as we only need to alter the cost of the graph edges which do not belong to G^E and the rest of the process remains the same. However, for problems where the search graph is implicit and the heuristics for a state is computed on demand when the state is visited (for example, the Manhattan distance or the linear conflict heuristics for sliding tile puzzles), this computation becomes prohibitively expensive.

In this paper, we describe a learning based planner (MHA*-L) that attempts to overcome the above mentioned problems by using a recently developed search algorithm called Multi-heuristic A* (MHA*) (Aine et al. 2014) as the base level search algorithm (instead of WA*). MHA* offers two advantages over WA*, that are crucial for a learning based planner. First, it uses multiple heuristics simultaneously to explore the search space, which reduces the chance of getting stuck at local minima, as the search can try to reach the goal using diverse heuristics and if any of the heuristic functions (or a combination) can lead the search to the goal, it can succeed. More importantly, MHA* needs access to a single consistent heuristic to provide bounded sub-optimal solutions, all additional heuristics can be arbitrarily inadmissible, which offers a lot of flexibility to design these heuristics. If we do not need to maintain the bounded admissibility constraints, any experience (in form of a path $\Pi(s_A, s_B)$ between states s_A and s_B) can be converted to a heuristic using the following equation,

$$h^{A,B}(s) = \epsilon_1 * h_0(s, s_A) + C^E(\Pi(s_A, s_B)) + \epsilon_1 * h_0(s_B, s_{goal}) \quad (2)$$

where $C^E(\Pi(s_A, s_B))$ is the cost of the path from s_A to s_B obtained from experience and ϵ_1 (≥ 1) is a factor used to bias $h^{A,B}$ towards the already traversed path. As these heuristics need not be admissible, we do not need to use costly computation such as Eqn. 1 [1]

We present a learning framework that samples path segments from previously solved instances and partitions them offline using a clustering algorithm. When asked to solve a new instance, the planner picks up the best experience choices (path segments) from each cluster in terms of cost to compute the inadmissible heuristics (using Eqn. 2). These heuristics are then used by the MHA*-L framework to compute the plans. On one hand, the MHA*-L planner offers a flexible approach to incorporate learning in planning (compared to the E-Graph planner). On the other hand, it can also be viewed as a framework for generating high quality heuristics for a MHA* planner. In (Aine et al. 2014), different inadmissible heuristics were used for different problems, designed in a somewhat ad-hoc manner. In contrast, the MHA*-L framework provides a systematic approach for generating inadmissible heuristic using experience.

We investigate the performance of this framework (MHA*-L) on two problem domains, 6D arm planning (for the PR2 robot) and large sliding tile puzzles. Comparisons with WA*, MHA* without learning (MHA*-W) and the E-Graph planner (for the robot arm planning) show that the proposed approach is indeed very efficient in solving hard planning problems.

[1]Note that, for both E-graphs and MHA*-L, h_0 should be able to compute the cost-to-go heuristic between any two states, and not just between a state and the goal.

Algorithm

The MHA*-L planner works in the following manner. We start with a database of path samples ($\Pi(s_A, s_B)$) obtained from previous planning queries (or from demonstrations). Next, we partition this database (offline) according to the number of heuristics (n) we want to compute, using some similarity function. Now, given an instance we compute an inadmissible heuristic from each partition ($h_1, \ldots, h_n$) to ensure diversity (online). Finally, we run MHA* with h_0 as the anchor heuristic (as described in (Aine et al. 2014)) and $h_1, \ldots, h_n$ as the additional inadmissible heuristics. In Algorithm 1, we include the high level calls for the offline and online tasks for our planner.

Algorithm 1 MHA*-L: Overview

```
1: procedure OFFLINEPROCESSING
2:     PLDB ← BUILDDATABASE (S, k)        ▷ S: set of solved instances, k: samples per plan
3:     PARTITIONDATABASE (PLDB, n) ▷ n: number of additional heuristics
4: procedure ONLINEPROCESSING
5:     Input: Problem instance P_in, consistent heuristic h_0
6:     FINDHEURISTICCANDIDATES (P_in, n)
7:     PLANWITHMHA*
8:     if plan successful then
9:         UPDATEDATABASE
```

Plan Database Generation and Partitioning

The first component of this system is a database of complete/partial plans. The database creation part is very simple, we start with a set of previously computed plans. From each plan, we sample a few (say a pre-decided constant k) path segments and include them in the plan database (PLDB) where each entry is a feasible path segment $\Pi(s_A, s_B)$ and the cost of that $C(\Pi(s_A, s_B))$. Including partial paths in the plan database increases the coverage of our learning based heuristics, as now the number of entries in the database is not restricted to the number of previously solved instances. However, we may not want to include paths that are that are too small, for this we only sample path segments of length greater than half of the actual solution length (for a candidate plan of length x, we randomly choose $k-1$ partial segments of length l, where l is a random number between $x/2$ and x, and include the complete plan as the k^{th} segment).

Next, we partition this database into n-parts using the simple k-median clustering method, where n is the number of inadmissible heuristics used for MHA*-L. For clustering, we need to compute a similarity measure which compares to paths in the PLDB. For this, we use the dynamic time warping similarity metric (DTW) (Sakoe and Chiba 1990) with the distance between two intermediate points computed using the heuristic function (h_0). The idea behind clustering the database is to ensure maximal diversity among the heuristics which in turn reduces the probability of all of them getting trapped in similar local minima. We also note the centroid of each cluster formed so that we can quickly up-

Algorithm 2 MHA*-L: Offline tasks

```
1: procedure BUILDDATABASE(S, k)
2:     PLDB ← ∅
3:     for all plan P ∈ S do
4:         for i = 1 ... k − 1 do
5:             Randomly sample a plan segment p from P
6:             PLDB ← PLDB ∪ p
7:         PLDB ← PLDB ∪ P
8: procedure PARTITIONDATABASE(PLDB, n)
9:     Compute distance matrix for PLDB using DTW
10:    Partition PLDB in n clusters using k-median
11:    Store the centroid for each cluster PLDB_i
```

date the database when a new plan segment is computed. In Algorithm 2 we include the pseudocode for these steps.

Algorithm 3 MHA*-L: Online tasks

```
1: procedure FINDHEURISTICCANDIDATES(P_in, n)
2:     for i = 1, ..., n do
3:         hc_i ← arg min_{Π(s_A,s_B)} {h^{A,B}(s_start): ∀ Π ∈ PLDB_i }
4: procedure PLANWITHMHA*
5:     INITIALIZESEARCHES()
6:     while s_goal not expanded do
7:         for i = 1, 2, ..., n do
8:             if i^th search can satisfy the ε1*ε2 bound then
   ▷ See (Aine et al. 2014) for details
9:                 s ← OPEN_i.TOP()
10:                if s == StartState(hc_i) then
11:                    OPEN_i ← ∅
12:                    Update g(EndState(hc_i))
13:                    Replace h_i with h_0
14:                    Insert EndState(hc_i) in OPEN_i
15:                else
16:                    Expand state from the i^th search
17:            else
18:                Expand state from the anchor search
19: procedure UPDATEDATABASE(P_n)
20:    for i = 1 ... k − 1 do
21:        Randomly sample a plan segment p from P_n
22:        PLDB_p ← Cluster with centroid closest to p
23:        PLDB_p = PLDB_p ∪ p
24:    PLDB_n ← Cluster with centroid closest to P_n
25:    PLDB_n = PLDB_n ∪ P_n
26:    Re-cluster PLDB if necessary
```

Heuristic Generation, Search and Database Update

Pseudocodes for the online tasks of our planner is included in Algorithm 3. When the planner is asked to solve a problem instance P_{in} (with s_{start}, s_{goal} and a consistent heuristic h_0), we first decide a candidate plan segment from each cluster ($PLDB_i$) using which we will compute h_i. For this, we pick the path segment (hc_i) which has the minimum $h^{A,B}(s_{start})$, according to Eqn. 2 (s_A is the start state and s_B is the end state of the segment).

Once a candidate segment is selected, h_i for any state s is computed directly using Eqn. 2. We run MHA* using h_0 for the anchor search and n additional heuristics (one heuristic per cluster). In (Aine et al. 2014), two MHA* algorithms are described, IMHA* and SMHA*. While the MHA*-L framework can work with both the planners, in this work, we only consider the shared version of MHA* (as it was shown to be the better algorithm). Throughout the rest of the paper, we will use the term MHA* to refer to the SMHA* algorithm.

The parts where we deviate from the original MHA* are, i) we run the anchor search with an inflated heuristic ($\epsilon_1 * h_0$), but other searches are run without inflation as h_i computed using Eqn. 2 is already inflated, and ii) in lines 10-14, which says the if when we reach the start state (s_A) of hc_i we take a shortcut to the end state s_B, by adjusting its g value using $C(hc_i)$ and starting a new search from s_B with heuristic h_0 instead of h_i. This ensures that once we find a path to the candidate segment we do not try to traverse the segment again, instead the search now tries to compute a path from end of this segment to the goal.

Note that a plan computed by this method is bounded by $\epsilon_1 * \epsilon_2$ sub-optimality factor (same as MHA*). When we successfully compute a new plan, we again sample k random segments from the plan and add them to their closest cluster in $PLDB$. If the database has grown quite a bit from the last clustering, we re-cluster it.

Experimental Results

6D Robot Arm Planning

In this domain, we consider a 6-DOF robot arm with a fixed base situated at the center of a 2D environment with obstacles. The environment is the same as described in the publicly available SBPL library (http://www.sbpl.net). The planning objective here is to move the end-effector from its initial configuration to the goal configuration following a smooth path while avoiding obstacles. An action is defined as a change of a global angle of any particular joint. We compare the WA* (without state re-expansions (Likhachev, Gordon, and Thrun 2004)) planner, the MHA* planner without learning (MHA*-W), the E-Graph planner and the learning based MHA* planner (MHA*-L).

	WA*	E-Graphs	MHA*-W	MHA*-L
Instances Solved	41	49	70	84
Run time	1.44	0.97	1.29	1.00
Solution Costs	0.96	1.12	1.07	1.00

Table 1: Results for the arm planner, comparing WA*, E-Graphs, MHA*-W and MHA*-L. Run times and solution costs for each algorithm are reported as ratios over MHA*-L numbers for instances that are solved by both the planners.

For the first experiment, we generated a test environment discretized into a 200×200 2D grid with three randomly placed obstacles (one circular, one T-shaped and one rectangular) and created a test suite of 100 random start-goal pairs. We computed the consistent heuristic (h_0) by running a 2D Dijkstra search from goal to start. For MHA*-W, we generated 4 additional heuristics by choosing 4 random way-points on the grid ($s_1, \ldots, s_4$) and directing the

search through the chosen points ($h_i(s) = h_0(s, s_i) + h_0(s_i, s_{goal}),\ i = 1, \ldots, 4$). We used the plans computed by the WA* to form the initial databases for the MHA*-L' and the E-Graph planner. For MHA*-L database, we used 5 segments from each plan, we then clustered the database in 4 parts using the DTW metric. During the run, whenever a new instance was solved by the E-Graph or the MHA*-L planner, we added the new paths to their respective databases. We used a target bound of 10, for WA* we set $\epsilon = 10$, whereas for E-Graphs and MHA*(s) we set $\epsilon_1 = 5$ and $\epsilon_2 = 2$. We used a time limit of 60 seconds for solving each instance. The results of this experiment are shown in Table 1, which clearly highlight the efficacy of MHA*-L over the other algorithms. We observe that the E-Graphs perform better than WA* but not than MHA*-W, which is expected, as the random selection of start and goal states affected the E-Graph planner (the tasks were not necessarily repetitive), in contrast MHA* could solve more problem even without any learning. However, when we combine the learning with the MHA* approach, we get the best results (MHA*-L dominates MHA*-W in all three metrics), indicating that learning can indeed help in designing better heuristics for MHA* (when compared to the ad-hoc approaches).

Database % used	25	50	100
Instances solved	63	81	88
No. of heuristics	4	8	16
Instances solved	88	100	100

Table 2: Impact of amount of learning and number of heuristics on MHA*-L planner for robot arm planning.

In the next two experiments, we tested the impact of the amount of learning and the number of heuristics on the performance of MHA*-L. For this, we built a database by collecting sampled segments from all the instance solved in the first experiment. Next, in i) we compute 4 inadmissible heuristics using 25, 50 and 100% of the complete database, and in ii) we use the total database and compute 4, 8 and 16 heuristics. We stopped the online database updates for both i) and ii), and ran MHA*-L on a new test suite of 100 random instances (start-goal pair) with bound 10 and time limit 60 secs. The results of these experiments are shown in Table 2, which show that i) with fixed number of heuristics, learning from larger database increases the success rates, and ii) with the same amount of learning, using more heuristics improves the performance. In fact, with 8 and 16 heuristics, MHA*-L was able to solve all the instances within the time limit.

Sliding Tile Puzzles

In this section, we present the experimental results for large sliding tile puzzles (8×8, 9×9 and 10×10). For this domain, we only compare WA*, MHA*-W and MHA*-L, as the E-Graph planner was very inefficient (computing the ϵ-admissible E-Graph heuristic took too much time, and thus, none of the instances were solved within the time limit).

For the first experiment, we randomly generated 100 solvable instances (for each size). We used the Manhattan distance (MD) plus linear conflicts (LC) as the consistent heuristic ($h_0 = MD + LC$). For MHA* (both variants), we used a set of 5 heuristics (anchor + 4 inadmissible). For

	Instances Solved			Run Time		Solution Cost	
Size	WA*	MHA*-W	MHA*-L	WA*	MHA*-W	WA*	MHA*-W
8X8	65	77	91	0.93	1.04	1.46	1.21
9X9	32	59	88	0.98	0.95	1.17	1.06
10X10	11	19	35	0.90	1.04	0.93	1.06

Table 3: Results for the sliding tile puzzles comparing WA*, MHA*-W and MHA*-L. Run times and solution costs (for WA* and MHA*-W) are reported as ratios over MHA*-L numbers for instances that are solved by both the planners.

MHA*-W, the additional heuristics were obtained using the same scheme as described in (Aine et al. 2014), we computed the number of misplaced tiles (MT), and added MD, LC and MT with random weights between 1.0 and 5.0. For MHA*-L, similar to arm planning, we used the problem instances solved by WA* to create the initial database (for each problem size), and clustered them in 4 parts to compute the inadmissible heuristics. During the run, we also did the online updates for each successful plan (by MHA*-L). We use a bound of 10 ($\epsilon = 10$ for WA*, $\epsilon_1 = 5$ and $\epsilon_2 = 2$ for MHA*-W and MHA*-L), and a time limit of 180 seconds. We include the results of this experiment in Table 3. The results clearly show that MHA*-L dominates both WA* and MHA*-W, with the performance gap (in terms of instances solved) increasing with larger (and thus harder) problems.

Database % used	25	50	100
Instances solved	21	39	46
No. of heuristics	4	8	16
Instances solved	46	66	59

Table 4: Impact of amount of learning and number of heuristics on MHA*-L planner (10×10 puzzle).

Next, we tested the impact of learning and number of heuristics on MHA*-L (Similar to the arm planning). We collected all the instances solved in the first experiment and built an experience database, and performed two experiments, i) by computing 4 inadmissible heuristics learned by using 25, 50 and 100% of the database, and ii) by computing 4, 8 and 16 heuristics from the total database. For both i) and ii), we ran MHA*-L on a new test suite of 100 10×10 puzzles with bound 10 and time limit 180 secs and stopped the online update of the databases. The results of these tests are shown in Table 4, which shows the i) for the same number of heuristics, larger database helps solving more problems (same observation as arm planning), and ii) the success rate improves with increase in the number of heuristics from 4 to 8, but degrades thereafter (different from arm planning), mainly due to the fact that with 16 heuristics, the heuristic computation time starts to dominate the planning time.

Conclusions

We presented a learning based planning framework (MHA*-L) that uses previously generated plans to compute multiple heuristics and then uses those heuristics to run a multi-heuristic search. Experimental results obtained for two domains demonstrate that the proposed planner is more efficient when compared to the WA*, the MHA* planner without learning (MHA*-W) and the E-Graph planner.

References

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2014. Multi-Heuristic A*. In *Robotics: Science and Systems*.

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 16*. Cambridge, MA: MIT Press.

Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*.

Phillips, M.; Dornbush, A.; Chitta, S.; and Likhachev, M. 2013a. Anytime incremental planning with e-graphs. In *ICRA*, 2444–2451.

Phillips, M.; Hwang, V.; Chitta, S.; and Likhachev, M. 2013b. Learning to plan for constrained manipulation from demonstrations. In *Robotics: Science and Systems*.

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

Röger, G., and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *ICAPS*, 246–249.

Sakoe, H., and Chiba, S. 1990. Readings in speech recognition. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. chapter Dynamic Programming Algorithm Optimization for Spoken Word Recognition, 159–165.