Optimize Planning Heuristics to Rank, not to Estimate Cost-to-Goal

Leah Chrestien
Czech Technical University in Prague
leah.chrestien@aic.fel.cvut.cz

Tomáš Pevný
Czech Technical University in Prague
pevnytom@fel.cvut.cz

Stefan Edelkamp
Czech Technical University in Prague
edelkste@fel.cvut.cz

Antonín Komenda
Czech Technical University in Prague
antonin.komenda@fel.cvut.cz

1 Speed of convergence against the size of training set:

We use the classical results of statistical learning theory to show that estimating the cost-to-go converges more slowly with respect to the size of the training set than estimating the rank.

From Equation (1) in the main text it should be obvious that ranking is in its essence a classification problem if one considers a pair of states (s, s') as a single sample. For classification/ranking problems, a following bound on true error rate [3] holds with probability $1 - \eta$

$$R_c \leq \hat{R}_c + \sqrt{\frac{1}{n_p}\left[\left(1 + \kappa \ln \frac{2n_p}{\kappa}\right) - \ln \eta\right]},$$

where R_c denotes the true loss (Equation (1) in the main text) and $\hat{R}_c$, its estimate from n_p state pairs (s_i, s_j), and κ, the Vapnik-Chervonenkis dimension [3] of the hypothesis space.

Optimizing $h(s, \theta)$ with respect to cost-to-goal (Equation (5) in the main text) is a regression problem for which a different generalization bound on prediction error [1] holds with probability $1 - \eta$

$$R_r \leq \hat{R}_r \left[1 - \sqrt{\frac{1}{n_s}\left[\kappa\left(1 + \ln \frac{n_s}{\kappa}\right) - \ln \eta\right]}\right]_+^{-1}$$

where again R_r is the error of the estimator of cost-to-go and $\hat{R}_r$ is its estimate from n_s states (Equation (5) in the main text),[1] and $[x]_+$ is a shorthand for $\max\{0, x\}$.

From the above, we can see that excess error in the ranking case converges to zero at a rate $\frac{\sqrt{\ln n_p}}{\sqrt{n_p}}$, which is slightly faster than that of regression $\frac{\sqrt{\ln n_s}}{\sqrt{n_s} - \sqrt{\ln n_s}}$. But, the number of state pairs in the training set grows quadratically with the number of states; therefore, the convergence rate of the ranking problem for the number of states n_s can be expressed as $\frac{\sqrt{2 \ln n_s}}{n_s}$, which would be by at least $\frac{1}{\sqrt{n_s}}$ factor faster than that of regression. We note that bounds are illustrative, since the independency of samples is in practice violated, since samples from the same problem-instance are not independent.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

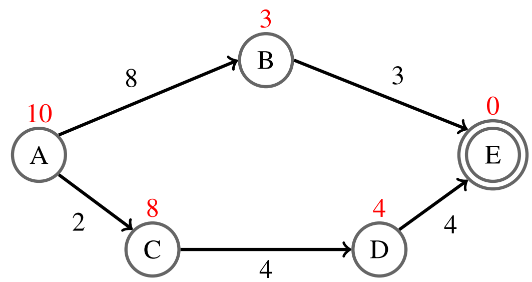

Figure 1: Problem instance where perfect heuristic is not strictly optimally efficient with GBFS. Numbers on the edges denote the cost of action and red numbers next to nodes denote the minimal cost-to-go.

2 Suboptimality of perfect heuristic in GBFS

Example 1. *While cost-to-goal h^* is the best possible heuristic for algorithms like A* (up to tie-breaking) in terms of nodes being expanded, for GBFS, h^* does not necessarily yield optimal solutions.*

Proof. A* explores all the nodes with $f(s) < f^*(s) = g(s) + h^*(s)$ and some with $f(s) = f^*(s) = g(s) + h^*(s)$, so nodes can only be saved with optimal $f^*(s) = g(s) + h^*(s)$. In fact, when given h^*as the heuristic, *only* nodes s with $f(s) = f^*(s)$ are expanded. Depending on the strategy of tie-breaking, the solution path can be found in the minimal number of node expansions or take significantly longer (e.g., in lower g-value first exploration of the search frontier). Any heuristic other than h^* is either overestimating, and, therefore, may lead to either non-optimal solutions in A*, or weaker than h^*, leading to more nodes being expanded.

Even if h^* is given, GBFS is not guaranteed to be optimal. Consider the following graph with five nodes A, B, C, D, E, and $w(A, B) = 8$, $w(B, E) = 3$, $w(A, C) = 2$, $w(C, D) = 4$, $w(D, E) = 4$, and $h^*(A) = 10$, $h^*(B) = 3$, $h^*(C) = 8$, $h^*(D) = 4$, $h^*(E) = 0$ (see Figure 1), initial node $s_0 = A$, goal node $E \in \mathcal{S}^*$. The numbers are the actual costs and the red numbers are the exact heuristic function. For finding a path from node A to node E, GBFS would return (A, B, E) following the heuristic function. However, the path (A, C, D, E) has cost 10 instead of 11. □

3 Theory

Below, we prove the claim in made in Experimental section stating that if a heuristic h is strictly optimally efficient for A* search, then it is also strictly optimally efficient for GBFS.

Theorem 1. *Let a heuristic h is a perfect ranking for A* search on a problem instance $\Gamma = (\langle \mathcal{S}, \mathcal{E}, w \rangle, s_0, \mathcal{S}^*)$ with a constant non-negative cost of actions $((\exists c \geq 0)\,(\forall e \in \mathcal{E})\,(w(e) = c))$. Then h is a perfect ranking for GBFS on Γ.*

Proof. Let $\pi = ((s_0, s_1), (s_1, s_2), \ldots, (s_{l-1}, s_l))$ be an optimal plan such that $\forall i \in \{1, \ldots, l\}$ and $\forall s_j \in \{s_j \mid \exists (s_k, s_j) \in \mathcal{E} \wedge s_k \in \mathcal{S}^{\pi_{:i-1}} \wedge s_j \notin \mathcal{S}^{\pi_{:i}}\}$ we have $g(s_j) + h(s_j) > g(s_i) + h(s_i)$, where $g(s)$ is the distance from s_0 to s in a search-tree created by expanding only states on the optimal path π. We want to proof that if all actions have the same positive costs, then $h(s_j) > h(s_i)$ as well.

We carry the proof by induction with respect to the number of expanded states.

At the initialization (step 0) the claim trivially holds as the Open list contains just a root node and the set of inequalities is empty.

Let's now make the induction step and assume the theorem holds for the first $i - 1$ step. We divide the proof to two parts. At first, we prove the claim for $(O)_i \setminus \mathcal{N}(s_{i-1})$ and then we proof the claim for $\mathcal{N}(s_{i-1})$, where $\mathcal{N}(s)$ denotes child states of the state s.

[1] The formulation in [1] contains constants c and a, but authors argue they can be set to $a = c = 1$ and hasubve been therefore omitted here.

1) Assume $s_j \in (O)_i \setminus \mathcal{N}(s_{i-1})$. Since h is strictly optimally efficient for A*, it holds that

$$
\begin{aligned}
g(s_j) + h(s_j) >& g(s_i) + h(s_i) \\
h(s_j) >& (g(s_i) - g(s_j)) + h(s_i) \\
h(s_j) >& h(s_i),
\end{aligned}
$$

where the last inequality is true because $g(s_i) - g(s_j) \geq 0$.

Assume $(s_j \in \mathcal{N}(s_{i-1}))(s_j \neq s_i)$. Since h is strictly optimall efficient for A*, it holds that

$$g(s_j) + h(s_j) > g(s_i) + h(s_i). \quad (1)$$

Since $g(s_j) = w((s_{i-1}, s_j)) = w((s_{i-1}, s_i)) = g(s_i)$, it holds

$$h(s_j) > h(s_i), \quad (2)$$

which finishes the proof of the theorem. □

4 Optimally efficient heuristic might not exists for GBFS

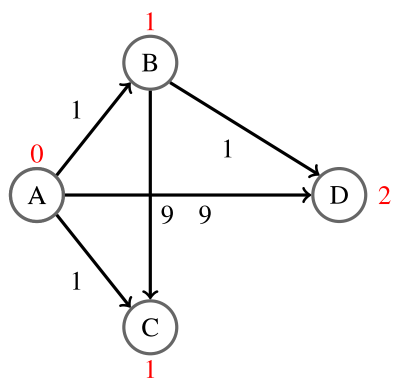

Figure 2: Problem instance where optimally efficient heuristic does not exists for GBFS.

Consider the following graph in Figure 2 with four nodes $A, B, C, D,$, and $w(A, B) = 1$, $w(B, D) = 1$, $w(A, C) = 1$, $w(A, D) = 9$, $w(B, C) = 9$, and $h^*(A) = 0$, $h^*(B) = 1$, $h^*(C) = 1$, $h^*(D) = 2$ where A is the goal state and D is the initial state.

We can see that for GBFS, the perfect heuristic does not exist for D. On expansion, the GBFS algorithm will put A and B to the open list with heuristic values h(A) = 0 and h(B) = 1. GBFS takes A from the open list and checks if it is a goal state. Since A is goal state, GBFS terminates returning path (D,A) as a solution. However, this is not the optimal path as a better (optimal) solution exists (D, B, A). Since the definition of optimal ranking requires the inequalities to hold for this optimal path, in GBFS, the perfect heuristic does not exist for all initial states.

The problem can be fixed if the definition of optimal ranking is changed to consider two cases in the merit function $f(s) = \alpha g(s) + \beta h(s)$: $\alpha > 0$ and $\beta > 0$ and $\alpha = 0$ and $\beta > 0$. In the first case, the optimal ranking should be defined with respect to the "optimal path" (this is the A*); in the latter case, it should be the path with minimal number of expansions. With GBFS, the user simply wants to find a solution but not care about its optimality. With this change, the heuristic function will exist for Figure 2.

5 Training set with multiple solution paths with the same length of the plan

The behavior of the learned heuristic function depends on the composition of the training set, which is illustrated below on a simple grid problem. The agent starts at position (4,4) and has to move to the goal position (0,0). There are no obstacles and the agent can only move one tile down or one tile left (the effects on his positions are either (-1,0) or (0,-1)), the cost of the action is one. The number of different solutions path grows exponentially with the size of the grid and all solution path has the same cost (for problem of size 5x5, there are 70 solutions). This problem is interesting, since merit function in A* with optimal heuristic function (cost to goal) is constant, equal to 8, for all states.

For the size of the grid (4,4), the heuristic is determined by a table with 25 values. Below, these values are determined by minimizing the proposed loss for A* algorithm with logistic loss surrogate by BFGS (with all zeros as initial solution). Since the loss function is convex, BFGS finds the solution quickly.

In the first case, the training set contains one solution path ([4, 4], [3, 4], [2, 4], [1, 4], [0, 4], [0, 3], [0, 2], [0, 1], [0, 0]), where the agent first goes left and then down. The learnt heuristic values h is shown in Table 1.

y/x	0	1	2	3	4
4	-191.53	-131.99	-76.94	-31.25	0.0
3	0.0	31.25	76.94	131.99	191.53
2	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
0	0.0	0.0	0.0	0.0	0.0

Table 1: Table showing one solution path.

An A* search using these heuristic values will first always take action moving the agent to left and then down. When moving down, the agent does not have another choice. Notice that the heuristic values of many states are not affected by the optimization, because they are not needed to effectively solve the problem.

In the second case, the training set contains two solution paths: the first is in Table 1 and in the second table 2, the agent goes first down and then left. The learnt heuristic values are shown in 2.

y/x	0	1	2	3	4
4	-95.76	-66.0	-38.47	80.14	0.0
3	0.0	15.62	38.47	131.99	80.14
2	0.0	0.0	0.0	38.47	-38.47
1	0.0	0.0	0.0	15.62	-66.0
0	0.0	0.0	0.0	0.0	-95.76

Table 2: Table showing two solution paths.

An A* search will now face tie at state (4,4), since states (3,4) and (4,3) have the same heuristic value. But the presence of the tie does not affect the optimal efficiency of the search, as A* will always expand states on one of the optimal path from the training set.

Finally let's consider a case, where all 70 possible solutions are in the training set. The learned heuristic values are shown in Table 3.

y/x	0	1	2	3	4
4	3.74	92.62	134.66	163.17	0.0
3	-46.61	2.35	91.93	134.38	163.17
2	-167.96	-47.7	1.94	91.93	134.66
1	-210.03	-168.66	-47.7	2.35	92.62
0	0.0	-210.03	-167.96	-46.61	3.74

Table 3: Table showing multiple solution paths.

Rank of heuristic values "roughly" corresponds to cost to goal. The reason, why some states are preferred over the others despite their true cost-to-goal being the same is that they appear in more solution paths. As shown in Table 3, the A* search with learnt heuristic values is strictly optimally efficient.

5.1 Baseline Comparison to breadth-first search

The fraction of solved problems for breadth-first search (5s time limit as used by solvers in the paper) is shown in Table 4.

domain	fraction
blocks	0.35
ferry	0.31
npuzzle	0.14
spanner	0.63
elevators	0.32

Table 4: Fraction of solved mazes by breadth-first search.

6 Training Details

For the grid domains, ADAM [2] training algorithm was run for 100×20000 steps for the grid domains.The experiments were conducted in the Keras-2.4.3 framework with Tensorflow-2.3.1 as the backend. While all solvers were always executed on the CPU, the training used an NVIDIA Tesla GPU model V100-SXM2-32GB. Forward search algorithms were given 10 mins to solve each problem instance.

For the PDDL domains, the training consumed approximately 100 GPU hours and evaluation consumed 1000 CPU hours.All training and evaluation were done on single-core Intel Xeon Silver 4110 CPU 2.10GHz with a memory limit of 128GB. The training algorithm AdaBelief [4] was allowed to do 10000 steps on the CPU. We emphasize though, that the training time does not include the cost of creating the training set.

		A*					GBFS					
problem	complx.	L^*	L_{gbfs}	L_{rt}	L_2	L_{be}	L^*	L_{gbfs}	L_{rt}	L_2	L_{be}	L_{le}
blocks		37	54	**27**	137	54	33	**28**	29	32	32	127
ferry		53	43	36	339	**20**	48	34	31	33	**20**	51
npuzzle		**294**	660	843	1936	641	297	311	333	591	**272**	418
spanner		55	546	**53**	807	416	61	56	**53**	117	65	148
elevators		**33**	35	52	657	310	**33**	**33**	43	115	198	73
Sokoban	3 boxes	**14**	**14**	**14**	17	**14**	**14**	**14**	**14**	**17**	**14**	**14**
	4 boxes	**32**	35	37	44	35	34	**32**	36	43	35	33
	5 boxes	**61**	67	68	72	63	65	62	66	77	64	**61**
	6 boxes	**171**	179	180	210	177	175	179	181	214	180	**174**
	7 boxes	**643**	651	653	755	654	645	641	**640**	754	655	643
Maze w. t.	50×50	**34**	37	**34**	41	40	34	**33**	35	43	40	35
	55×55	**51**	59	52	63	60	54	**52**	55	65	61	58
	60×60	**72**	78	75	83	78	73	**71**	77	89	79	84
Sliding puzzle	5×5	**1521**	1558	1534	1559	1545	**1524**	1539	1533	1556	1544	1531
	6×6	**2322**	2353	2334	2439	2388	2321	**2326**	2329	2334	2329	2329
	7×7	**3343**	3375	3347	3431	3411	3421	**3356**	3449	3512	3448	3379

Table 5: Average number of expanded states.

References

[1] Vladimir Cherkassky, Xuhui Shao, Filip M Mulier, and Vladimir N Vapnik. Model complexity control for regression using vc generalization bounds. *IEEE transactions on Neural Networks*, 10(5):1075–1089, 1999.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[3] Vladimir Vapnik. Principles of risk minimization for learning theory. *Advances in neural information processing systems*, 4, 1991.

[4] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806, 2020.

problem	complx.	SBA*	FDSS	A*					GBFS					
				L^*	L_{gbfs}	L_{rt}	L_2	L_{be}	L^*	L_{gbfs}	L_{rt}	L_2	L_{be}	L_{le}
blocks		100	100	100 ± 0	100 ± 1	100 ± 0	99 ± 1	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0	99 ± 1
ferry		100	95	98 ± 3	98 ± 3	100 ± 0	92 ± 8	100 ± 0	98 ± 3	100 ± 0	100 ± 0	100 ± 0	98 ± 4	98 ± 3
npuzzle		100	89	89 ± 1	87 ± 2	88 ± 0	83 ± 4	89 ± 1	92 ± 5	89 ± 1	89 ± 1	89 ± 1	92 ± 7	88 ± 3
spanner		100	90	100 ± 0	89 ± 2	100 ± 0	84 ± 6	92 ± 6	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0
elevators		100	37	91 ± 2	85 ± 10	75 ± 8	36 ± 6	66 ± 3	92 ± 3	85 ± 11	79 ± 8	76 ± 3	67 ± 4	58 ± 11
Sokoban	3 boxes	100	100	99 ± 0	98 ± 0	96 ± 0	97 ± 0	92 ± 0	98 ± 0	100 ± 0	94 ± 0	95 ± 0	92 ± 0	98 ± 0
	4 boxes	100	81	89 ± 2	89 ± 1	85 ± 2	81 ± 0	82 ± 3	87 ± 2	91 ± 1	84 ± 4	83 ± 3	84 ± 3	84 ± 3
	5 boxes	97	67	80 ± 1	75 ± 2	72 ± 2	72 ± 1	73 ± 0	78 ± 1	77 ± 1	74 ± 3	72 ± 3	72 ± 2	73 ± 1
	6 boxes	55	49	76 ± 2	69 ± 1	59 ± 3	51 ± 0	53 ± 3	73 ± 1	71 ± 1	56 ± 0	512	54 ± 0	64 ± 1
	7 boxes	46	31	55 ± 4	49 ± 3	47 ± 4	42 ± 3	45 ± 5	51 ± 3	49 ± 3	48 ± 3	43 ± 2	45 ± 2	49 ± 3
Maze w. t.	50×50	92	75	92 ± 0	91 ± 0	88 ± 1	87 ± 1	87 ± 0	89 ± 0	90 ± 1	89 ± 1	84 ± 0	85 ± 1	89 ± 0
	55×55	52	50	78 ± 1	75 ± 4	73 ± 1	72 ± 3	74 ± 3	74 ± 0	75 ± 3	74 ± 2	72 ± 3	75 ± 3	74 ± 2
	60×60	0	0	49 ± 1	37 ± 0	35 ± 3	32 ± 2	31 ± 1	42 ± 3	48 ± 1	36 ± 3	34 ± 3	32 ± 2	42 ± 0
Sliding puzzle	5×5	-	1	88 ± 1	83 ± 3	84 ± 3	80 ± 2	82 ± 0	86 ± 3	87 ± 1	84 ± 0	84 ± 1	84 ± 1	85 ± 1
	6×6	-	-	51 ± 2	48 ± 3	49 ± 2	45 ± 4	46 ± 3	47 ± 3	49 ± 3	45 ± 2	43 ± 3	66 ± 2	48 ± 3
	7×7	-	-	39 ± 3	35 ± 3	36 ± 3	32 ± 3	34 ± 3	35 ± 3	36 ± 3	35 ± 3	32 ± 3	34 ± 3	35 ± 3

Table 6: Average fraction of solved problem instances in percent with standard deviation. SBA* and FDSS denotes Fast Downward Stone Soup, They are domain independent planners.

problem	complx.	A*					GBFS					
		L^*	L_{gbfs}	L_{rt}	L_2	L_{be}	L^*	L_{gbfs}	L_{rt}	L_2	L_{be}	L_{le}
blocks		21.6	22.7	**21.4**	22.5	22.9	**21.8**	22.7	22.1	22.5	23.1	34.1
ferry		16.8	16.9	**16.7**	16.8	16.8	**16.8**	16.9	**16.8**	**16.8**	**16.8**	16.9
npuzzle		19.2	23.5	18.2	17.5	**17.3**	37.6	36.4	39.6	35.1	**34.1**	124.6
spanner		**49.1**	49.2	49.2	49.2	**49.1**	**49.0**	49.2	49.1	49.2	49.3	49.1
elevators		16.1	16.4	14.7	17.9	**15.8**	19.4	16.6	**15.9**	18.0	16.1	21.4
Sokoban	3 boxes	**13.1**	13.3	13.8	13.2	13.4	**13.1**	13.2	13.2	13.5	13.2	13.2
	4 boxes	15.4	**15.2**	16.1	15.7	16.8	16.7	**15.1**	16.1	15.6	16.3	15.8
	5 boxes	20.1	**19.2**	21.3	22.1	20.9	**19.8**	20.4	21.3	22.1	19.9	20.1
	6 boxes	29.6	29.3	27.7	28.2	**26.8**	28.3	**28.1**	29.6	29.4	29.9	30.1
	7 boxes	31.9	**31.4**	35.4	33.1	34.1	**30.1**	33.3	35.2	32.7	34.0	35.5
Maze w. t.	50×50	**24.1**	25.3	25.1	24.3	24.3	**24.3**	24.5	25.4	**24.3**	25.4	24.7
	55×55	34.1	**33.2**	35.0	34.2	33.9	**33.2**	33.1	34.6	34.6	36.5	36.3
	60×60	**41.2**	42.9	41.4	43.2	42.8	**42.1**	43.6	44.2	45.2	45.3	45.1
Sliding puzzle	5×5	**150.1**	153.7	155.2	154.6	154.5	154.5	153.5	153.9	155.0	**151.1**	152.1
	6×6	**252.3**	254.2	253.8	254.8	254.0	**255.9**	256.4	255.3	256.2	254.9	256.3
	7×7	**321.1**	324.1	322.2	324.3	320.4	**322.9**	324.1	323.4	327.1	324.6	323.7

Table 7: Average number of length of the solution. The average is computed only over problem instances solved by all 11 variants of forward search.