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

Leah Chrestien, Tomáš Pevný, Stefan Edelkamp, Antonín Komenda

Czech Technical University in Prague (CTU)

November 13, 2023

Background

The majority of research in heuristic search to estimate the cost-to-goal L_2 loss to generate optimal heuristics for many best-first heuristic search algorithms including A* or IDA^*.

$$L_2 = \frac{1}{N}\sum_{i}^{N}(h_i - h_i^*)^2$$

Issues with L_2

- A true cost-to-goal h^* does not guarantee that the search will find optimal solutions during minimum expansion of states.
- Optimizing cost-to-goal h^* does not utilize states off the solution path.
- Heuristic value for dead-end states are set to large values which can affect the stability of convergence of gradient-based optimization methods.

Our contribution

01 When optimizing a heuristic function, solve the ranking problem (see slide below) instead of training a cost-to-go heuristic NN.

02 State necessary and sufficient conditions for a strictly optimally efficient heuristic function.

03 Design two slightly different loss functions catered towards A* and GBFS search.

04 Experimentally demonstrate the advantages of ranking on eight problems (three grid and five PDDL).

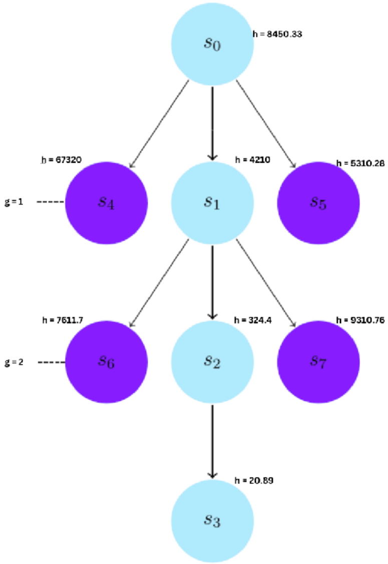
How do we rank states?

For a forward search merit function
$f(s) = \alpha g(s) + \beta h(s)$,
rank the states by setting these inequalities.

- $f(s_1) < f(s_4)$
- $f(s_2) < f(s_4)$
- $f(s_2) < f(s_6)$
- $f(s_2) < f(s_5)$
- $f(s_2) < f(s_7)$

The real heuristic values are not required!

Optimal states: s_0, s_1, s_2, s_3
Non optimal states: s_4, s_5, s_6, s_7



Perfect ranking heuristic

Definition 1 (Perfect ranking heuristic)

A heuristic function $h(s)$ is a perfect ranking in forward search with a merit function $f(s) = \alpha g(s) + \beta h(s)$ for a problem instance if and only if there exists an optimal plan $\pi = ((s_0, s_1), (s_1, s_2), \dots, (s_{l-1}, s_l))$ such that

- $g(s)$ is the cost from s_0 to s in a search-tree created by expanding only states on the optimal path π;
- $f(s_j) > f(s_i), \forall s_i \in \mathcal{S}^\pi \wedge s_j \notin \mathcal{S}^\pi$

where $\mathcal{S}$ denote all possible states $s \in \mathcal{S}$, $s_0 \in \mathcal{S}$ is the initial state.

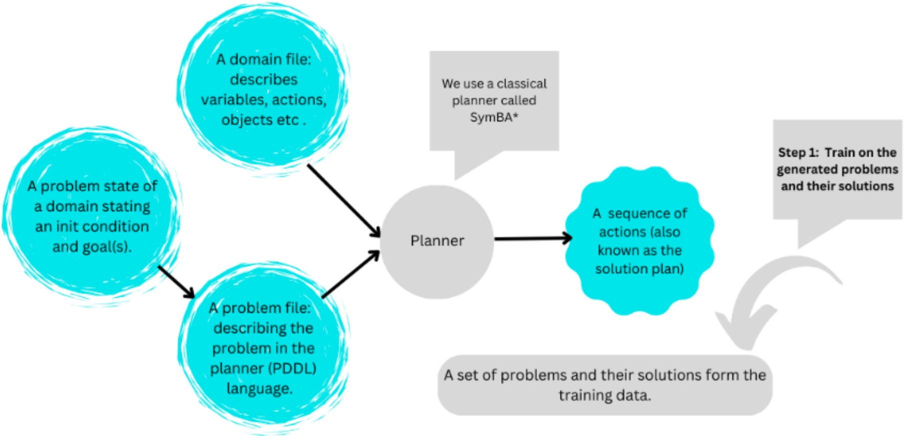
If the heuristic function satisfies these conditions, it is known as an optimally efficient heuristic function. This ensures that the search expands the minimum number of states on the optimal path.

Properties of an optimally efficient heuristic function

Pros and Cons

- 1. Rank optimization uses states off the solution path while cost-to-go doesn't.
- 2. Unlike cost-to-goal, a perfectly ranking heuristic does not provide a false sense of optimality.
- 3. Zero loss means optimal behavior.
- 4. Heuristic values for dead-end states are not needed.
- 5. The perfectly ranking heuristic is not goal-aware.

Experimental Method: 1. Generate initial training data



For details on SymBA*, see
https://homes.cs.aau.dk/ alto/papers/Planner-SymBA14.pdf

Experimental Method: 2. Instantiate the losses

Expand the A* and GBFS search trees on samples. Minimize the number of violated conditions/inequalities (see slide below) for a problem instance $(\Gamma, s_0, \mathcal{S}^*)$ and its optimal plan π.

For a heuristic function $h(s, \theta)$ with parameters $\theta \in \Theta$, the number of violated conditions can be counted as

$$\mathrm{L}_{01}(h, \Gamma, \pi) = \sum_{s_i \in \mathcal{S}^\pi} \sum_{s_j \in \mathcal{O}_i \setminus \mathcal{S}^{\pi:i}} [\![r(s_i, s_j, \theta) > 0]\!], \tag{1}$$

where

$$r(s_i, s_j, \theta) = \alpha(g(s_i) - g(s_j)) + \beta(h(s_i, \theta) - h(s_j, \theta)), \tag{2}$$

In practice, the Iverson bracket $[\![\cdot]\!]$ (also called 0-1 loss) is usually replaced by a convex surrogate such as the hinge-loss.

Experimental Method: 3. Train the NNs.

We instantiate the following loss functions that minimize the violated inequalities on states expanded during A* and GBFS search:

- L^*in A* search.
- $\mathrm{L}_{\mathrm{gbfs}}$ in GBFS search.
- L_2 or cost-to-go in regression.
- L_{rt} that compares states only on the solution trajectories.
- L_{be} known as Bellman loss. [1]
- L_{le}, a policy guided search with GBFS modified for efficiency. [2]

[1] Learning general optimal policies with GNNs: Expressive power, transparency, and limits.

[2] Policy-guided heuristic search with guarantees.

Coverage

problem	complx.	A* L^*	A* L_{gbfs}	A* L_{rt}	A* L_2	A* L_{be}	GBFS L^*	GBFS L_{gbfs}	GBFS L_{rt}	GBFS L_2	GBFS L_{be}	GBFS L_{le}
Blocks		**100**	**100**	**100**	99	**100**	**100**	**100**	**100**	**100**	**100**	99
Ferry		98	98	**100**	92	**100**	98	**100**	**100**	**100**	98	98
N-Puzzle		**89**	87	88	83	**89**	**92**	89	89	89	**92**	88
Spanner		**100**	89	**100**	84	92	**100**	**100**	**100**	**100**	**100**	**100**
Elevators		**91**	85	75	36	66	**92**	85	79	76	67	58
Sokoban	3 boxes	**99**	98	96	97	92	98	**100**	94	95	92	98
	4 boxes	**89**	89	85	81	82	87	**91**	84	83	84	84
	5 boxes	**80**	75	72	72	73	**78**	77	74	72	72	73
	6 boxes	**76**	69	59	51	53	**73**	71	56	51	54	64
	7 boxes	**55**	49	47	42	45	**51**	49	48	43	45	49
Maze w. t.	50×50	**92**	91	88	87	87	89	**90**	89	84	85	89
	55×55	**78**	75	73	72	74	74	**75**	74	72	**75**	74
	60×60	**49**	37	35	32	31	42	**48**	36	34	32	42
Sliding puzzle	5×5	**88**	83	84	80	82	86	**87**	84	84	84	85
	6×6	**51**	48	49	45	46	47	**49**	45	43	46	48
	7×7	**39**	35	36	32	34	35	**36**	35	32	34	35