1 THEOREMS

Theorem 1 (Upper Bound)**.** *For a problem instance with states $\mathcal{S} = \mathcal{S}^n \cup \mathcal{S}^o$, denote*

$$\mathcal{R}^n = \left\{ s'' \in \mathcal{S}^n \mid \exists s' \in \mathcal{S}^0 \wedge (g(s') + h_\theta(s') \geq g(s'') + h_\theta(s'')) \right\}, \tag{1}$$

the quantity $|\mathcal{R}^n|$ is an upper bound on the number of non-optimal states A expands during its search.*

Proof. A* algorithm always expands states with the lowest cost $g(s') + h_\theta(s')$. If a state from s'' is inserted in the open list, then A* might expand it, since the state on the optimal path might have higher cost. But since every state is expanded exactly once, each state from $\mathcal{R}^n$ can increase the number of non-optimal expanded states just once.

The proposed loss function minimizes the upper bound on $|\mathcal{R}^n|$ as

$$\begin{aligned} |\mathcal{R}^n| &= \sum_{s'' \in \mathcal{S}^n} [\![\max_{s' \in \mathcal{S}^o} (g(s') + h_\theta(s')) \geq g(s'') + h_\theta(s'')]\!] \\ &\leq \sum_{s' \in \mathcal{S}^o} \sum_{s'' \in \mathcal{S}^n} [\![g(s') + h_\theta(s') \geq g(s'') + h_\theta(s'')]\!]. \end{aligned}$$

The rationale behind optimizing the upper bound on $|\mathcal{R}^n|$ are following: (i) we expect parameters of h to be optimized using iterative algorithm, hence those $s' \in \mathcal{S}^n$ violating constraint

$$(\forall s' \in \mathcal{S}^o)(\forall s'' \in \mathcal{S}^n)(g(s') + h_\theta(s') < g(s'') + h_\theta(s'')) \tag{2}$$

but not being the maximum might continue violating in upcoming iterations, (ii) decreasing the total number of $s' \in \mathcal{S}^n$ violating constraint (2) decreases the probability of A* expanding corresponding non-optimal state.

□

Theorem 2 (Optimality on Training Set)**.** *If a loss function is zero for a given problem instance $\mathcal{S}$ with an optimal solution $\mathcal{S}^o$, then for all states $s_i \in \mathcal{S}^o$, we have $f(s_i) = f^*$.*

Proof. Let's denote the (optimal) shortest path from s_i to s_j $g_{i \to j}$ for any $s_i, s_j \in \mathcal{S}$. We break the proof into two parts. then we assume both $s_i, s_j \in \mathcal{S}^o$.

1. First we show that $\forall s_i \in \mathcal{S}^o$ and $\forall s_j \in \mathcal{S}^n$, $h_\theta(s_i) \leq g_{i \to j} + h_\theta(s_j)$. Since the loss is zero, it holds that $g(s_i) + h_\theta(s_i) < g(s_j) + h_\theta(s_j)$ due to requirement in Equation(2). Subtracting $g(s_i)$ we arrive to $h_\theta(s_i) < (g(s_j) - g(s_i)) + h_\theta(s_j) \leq g_{i \to j} + h_\theta(s_j)$. The fact that $g(s_j) - g(s_i) \leq g_{i \to j}$ stems from the fact that either the optimal path from the initial state to s_j goes over s_i, in which the equality holds, otherwise $g_i + g_{i \to j} > g_j$, hence $g_j - g_i < g_{i \to j}$, which completes this part of the proof.

2. Second, we show monotonicity $\forall s_i, s_j \in \mathcal{S}^o$. Without loss of generality, we assume $i < j$. From Equation

   $$(\forall s_i, s_j \in \mathcal{S}^o)(i < j)(g(s_i) + h_\theta(s_i) \leq g(s_j) + h_\theta(s_j), \tag{3}$$

   it holds that $g(s_i) + h_\theta(s_i) < g(s_j) + h_\theta(s_j)$. We again subtract $g(s_i)$ and since s_i and s_j are on the optimal path, it holds that $g(s_j) - g(s_i) = g_{i \to j}$, and therefore $h_\theta(s_i) < g_{i \to j} + h_\theta(s_j)$.

□

From the above, it should be clear that if the loss is zero on a given problem instance for h_θ, such a heuristic is optimal, which further implies that A* will find an optimal solution while expanding the minimal number of states. But, the constraints on a single problem instance are very narrow, which means that the very same h_θ might be poor on other problem instances, or in the same if for some reason a non-optimal state is expanded. Therefore, the loss is minimized on a large number of problem instances (the training set $\mathcal{T}$) which should minimize these cases as dictated by statistical learning theory.

1.1 SMOOTH APPROXIMATION OPTIMIZATION

The L^* loss defined as

$$\frac{1}{|\mathcal{S}^o||\mathcal{S}^n|}\sum_{s'\in\mathcal{S}^o}\sum_{s''\in\mathcal{S}^n}[\![g(s')+h_\theta(s')\geq g(s'')+h_\theta(s'')]\!]+$$
$$\frac{1}{|\mathcal{S}^o|(|\mathcal{S}^o|-1)}\sum_{i=2}^{|\mathcal{S}^o|}\sum_{j=1}^{i}[\![g(s_i)+h_\theta(s_i)\leq g(s_j)+h_\theta(s_j)]\!], \tag{4}$$

does not have an informative gradient (the gradient is either zero or does not exist), hence the parameters θ of the heuristic function h_θ can be optimized by only using zero-order optimization functions. To make it differentiable, the Iverson bracket therefore replaced by the differentiable surrogate, which in case of this work is logistic loss function $\mathrm{L}_l(x)=\log(1+\exp(-x))$. Using L_l, which we have used in experiments below, the optimized loss functions become

$$\mathrm{L}^*(\bar{\Gamma},h_\theta)=\frac{1}{|\mathcal{S}^o||\mathcal{S}^n|}\sum_{s'\in\mathcal{S}^o}\sum_{s''\in\mathcal{S}^n}\mathrm{L}_l\left(g(s'')+h_\theta(s'')-g(s')-h_\theta(s')\right)+$$
$$+\frac{1}{|\mathcal{S}^o|(|\mathcal{S}^o|-1)}\sum_{i=2}^{|\mathcal{S}^o|}\sum_{j=1}^{i}\mathrm{L}_l\left(g(s_j)+h_\theta(s_j)-g(s_i)-h_\theta(s_i)\right),$$

While the implementation of L_2 distance is straightforward, the implementation of L^* loss might be puzzling and is explained below in detail. When training a NN with L^*, all states $(s_1,\ldots,s_b)$ in the minibatch are from one problem instance, such that the states can be comparable as in Equations (2) and (3). When the states in the minibatch are projected by the neural network, they are represented by a vector of dimension $\hat{h}\in\mathbb{R}^b$, where $\hat{h}_i=h_\theta(s_i)$. We now form a matrix $\mathbf{H}\in\{-1,0,+1\}^{b,c}$ encoding constraints, where c is their total count in the given minibatch of size b. l^{th} constraint is encoded in the matrix, as follows: if we want $h_\theta(s_i)\leq h_\theta(s_j)$, then the $\mathbf{H}_{i,l}=-1$, $\mathbf{H}_{j,l}=1$, remaining values in the column are zeros and $l\in\{1,\ldots,c\}$ is the index of the constraint. The L^* with logistic loss (used in experiments below), is computed as

$$\frac{1}{c}\sum_{l=1^c}\log(1+\exp(\hat{h}^{\mathrm{T}}\mathbf{H}_{\cdot,l}), \tag{5}$$

where $\mathbf{H}_{\cdot,l}$ denotes l^{th} column. A source of confusion might be that on a minibatch of size b, the output of $(h_\theta(s_1),\ldots,h_\theta(s_b))^{\mathrm{T}}\mathbf{H}$ has dimension $c\gg b$.

1.2 GOAL-AWARENESS

One question to address is, how important is goal-awareness, i.e., the condition $h_\theta(s_n)=0$ where s_n is the goal. While possible, we haven't encoded that the heuristic is goal-aware in the loss function. Goal-awareness is, of course, an implication of h_θ being a lower bound (i.e., admissible), which, in turn, is always true for consistent heuristics.

But also for inconsistent heuristics, it is not a strict necessity to encode this requirement. As said, the heuristic h_θ re-weights the problem graph by setting the new weights from state s to s' to the old weights plus the difference $h_\theta(s')-h_\theta(s)$ Edelkamp and Schrödl (2012). In this setting, the weights of the new edges are non-negative, if and only if the heuristic is consistent so that A* simulates Dijkstra's shortest-path algorithm on the re-weighted problem graph. Another consequence is that the graph weights, and, thus, the order of nodes explored in the A* algorithm

does not change; if we have $h_\theta(s_n) = c$ and adapt h_θ by subtracting c from all the nodes, we get $(h_\theta(s) - c) - (h_\theta(s') - c) = h_\theta(s) - h_\theta(s')$. In other words, to learn A* 's behavior is translation-independent in h_θ, we have skipped to hard-wire this information in the loss function.

2 The Structure of the Neural Network

The structure of the neural networks implemented in the experiments is shown in Figure 1

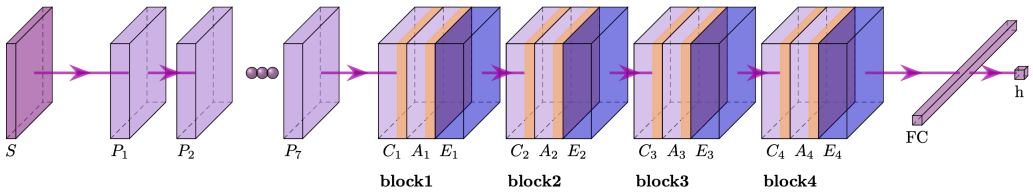

Figure 1: The structure of used neural network taken from Chrestien et al. (2021). A state S is processed through 7 convolution layers, $P_1..P_7$ of the same shape 3×3 with 64 filters. The output is passed to four blocks, where each block contains a convolution layer (C) followed by a multi-head attention operation with 2 heads (A) and a positional encoding layer (E). There are 180 filters in each of these convolution layers in the blocks. At all stages, the original dimension of the input is preserved. The output from block 4 is flattened by global average pooling along x and y coordinates before being passed onto a fully connected layer (FC) and a single heuristic prediction output h_θ. For the sake of picture clarity, skip connections are not shown in the neural network.

3 Average plan lengths and admissibility results on Sokoban

The results of the Sokoban experiments in section 5.2 state that the solutions found by CoAt optimizing L^*were close to optimum and longer by 1 step on an average . Table 1 shows the average plan length for the Sokoban mazes in the testing set. Here, we see that the average plan lengths generated by L^* ($\mathrm{CoAt} - \mathrm{L}^*$) closely follow the optimal plan lengths (see S(ym)BA* results in Table 1). Table 2 shows the number of times the heuristics generated by L* are admissible. Notice that admissibility is better for network optimizing L^*.

				CNN		CoAt	
n	SBA*	Merc	FD Stone Soup	L_2	L^*	L_2	L^*
3	21.40	29.32	27.89	30.56	28.67	22.90	22.02
4	34.00	41.00	37.00	43.42	41.33	35.11	35.03
5	38.82	45.76	42.37	45.34	44.83	40.12	40.12
6	41.11	-	-	49.82	46.32	42.11	41.65
7	-	-	-	58.23	56.33	53.33	53.19

Table 1: Average plan length of S(ym)BA* , Merc(ury14), Fast-Downward Stone Soupe, and all variants of A* with CoAt and CNN architectures optimizing L^* and L_2 loss functions. Column captioned #b indicates the number of boxes in different categories. The averages are over instances, where all planners solved the problem. The standard deviation of all repeated experiments was between 0.004 and 0.008 and it is not shown.

4 Rotation results on maze with teleports

The mazes in the testing set from Table 2 (see original paper) are rotated by 90°, 180° and 270° and the results are shown in Table 3.

#b	3	4	5	6	7	8	9
L^*-CoAt	97	96	93	94	93	92	95
L_2-CoAt	95	96	90	93	91	89	91

Table 2: Average number of admissible heuristic values of CoAt network optimizing $L^*$and L_2 loss functions. The number is estimated on mazes from the testing set solved in Table 1a (see original paper) from states on the founded path. The standard deviation of all repeated experiments was between 0.02 and 0.001 and it is not shown.

	90° rotation				180° rotation				270° rotation			
n	CNN		CoAt		CNN		CoAt		CNN		CoAt	
	L_2	L^*	L_2	L^*	L_2	L^*	L_2	L^*	L_2	L^*	L_2	L^*
50	90	92	97	98	91	92	97	98	91	92	96	97
55	77	78	84	87	75	77	83	88	75	76	85	86
60	67	69	74	76	66	69	72	74	68	69	71	74

Table 3: Average fraction of solved mazes with teleports (coverage) A* algorithm with convolution network (denoted as CNN), with the proposed Convolution-Position-Attention (CoAt) network and CoAt*. Only non-rotated mazes (No Rotation) of size 15×15 were used to train the heuristic function. On mazes rotated by $90°$, $180°$, $270°$, the heuristic function has to extrapolate outside its training set. All approaches have solved all mazes of size 15–40, therefore we have omitted them from the table.

5 5×5 SLIDING TILES PUZZLES

Table 4 shows the fraction of solved sliding tile puzzles (in percents) when the network is optimized only on puzzles it has solved previously. The protocol, neural networks, and all settings are exactly the same as in Section 5.3 of the manuscript. As in Table 3 in the original paper, the 0th row corresponds to A* with the heuristic function provided by an untrained network. The set of problem instances consists of 5000 puzzles downloaded from [1]. According to the results, A* with a heuristic function optimized with respect L^*consistently outperforms that optimized with respect to L_2 loss.

n	L^*	L_2
0	5	5
1	11	8
2	20	18
3	32	22
4	43	30
5	54	39
6	63	42
7	68	54

Table 4: Fraction of solved sliding Tiles puzzles (in percents), where the heuristic functions (implemented by CoAt networks) are optimized exclusively on previously solved puzzles.

6 PDDL DOMAINS

We have further compared $L^*$to L_2 on four problems encoded in PDDL taken from:[2] Blocksworld contains 732 problem instances of size 3–15, N-Puzzle contains 80, Gripper contains 50, and lastly Ferry contains 48. Since PDDL is a relational language, we adapted representation from Janisch et al. (2020), where each object in PDDL corresponds to one vertex in graph. Binary predicates are represented by edges between involved vertices. Binary vectors of vertices are used to encode unary and nullary predicates, where each predicate corresponds to a particular item of the vector. When unary predicate evaluates to true on an object, an item of the feature vector is set to one at the vertex

[1] https://github.com/levilelis/h-levin/

[2] https://github.com/williamshen-nz/STRIPS-HGN

	Blocks		Ferry		Gripper		N-Puzzle	
epoch	L^*	L_2	L^*	L_2	L^*	L_2	L^*	L_2
1	98	73	33	33	14	14	14	14
2	100	75	48	44	15	15	52	34
3	100	76	53	48	16	15	55	36
4	100	76	54	49	17	16	55	38
5	100	78	54	51	17	16	58	42
6	100	78	54	51	17	16	58	43
7	100	79	54	51	17	16	60	44
8	100	80	54	52	18	16	60	44
9	100	80	55	52	18	16	60	46
10	100	81	55	52	19	16	62	46

Table 5: Fraction of solved puzzles (in percents) encoded in PDDL. Heuristic functions implemented by a two-layer graph neural networks are optimized exclusively on previously solved puzzles.

corresponding to the object. When nullary predicate is true, a corresponding item of all vectors of all vertices is set to one. Since in a domain there can be multiple types of binary predicates, we represent them as multi-graph, i.e. there was a dedicated graph encoding exactly one type of binary predicates. The graph neural network projecting PDDL encoded in graph to a scalar value (heuristic) contained two graph attention layers Veličković et al. (2018) with output dimension 8, followed by reduction of vectors of vertices reduced to a single vector by mean and maximum, one Dense layer with relu non-linearity of dimension 32, and a linear dense layer with scalar output.

All neural networks were trained by a bootstrap protocol, where the neural network trains on problems that have been solved with A* that uses that particular network as heuristic functions. Inspired by Orseau and Lelis (2021), the protocol from Section 5.3 was adapted, such that the network was allowed to perform 1000 gradient steps of AdaBelief algorithm after every 32 new solved problems. In every epoch, A* algorithm tried to solve only unsolved problems for a total of 10 epochs. A* algorithm had a time limit 30s.

The fraction of solved problem instances is shown below. We again observe that A* with heuristic optimized with respect to L^*performs better than that with respect to L_2. In case of Blocks and n-puzzle problems, the improvement is large, in case of Ferry and Gripper, the improvement is small. We admit to not knowing the reason(s) behind this poor performance. One possible cause can be due to the small number of problems instances of Ferry and Gripper. The complete implementation of this experiment including problems is attached to clear any doubts. All experiments were repeated three times with a fixed random seed.

REFERENCES

Stefan Edelkamp and Stefan Schrödl. *Heuristic Search - Theory and Applications*. Academic Press, 2012.

Leah Chrestien, Tomáš Pevný, Antonín Komenda, and Stefan Edelkamp. Heuristic search planning with deep neural networks using imitation, attention and curriculum learning. *arXiv:2112.01918*, 2021.

Jaromír Janisch, Tomás Pevný, and Viliam Lisý. Symbolic relational deep reinforcement learning based on graph neural networks. *CoRR*, abs/2009.12462, 2020. URL `https://arxiv.org/abs/2009.12462`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Laurent Orseau and Levi HS Lelis. Policy-guided heuristic search with guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12382–12390, 2021.