Heuristic Search Optimisation Using Planning and Curriculum Learning Techniques

Leah Chrestien(✉), Tomáš Pevný, Stefan Edelkamp, and Antonín Komenda

Czech Technical University in Prague, Jugoslávských partyzánů 3, 160 00 Praha, Czech Republic
leah.chrestien@aic.fel.cvut.cz, {pevnytom,stefan.edelkamp}@fel.cvut.cz, antonin.komenda@agents.fel.cvut.cz

Abstract. Learning a well-informed heuristic function for hard planning domains is an elusive problem. Although there are known neural network architectures to represent such heuristic knowledge, it is not obvious what concrete information is learned and whether techniques aimed at understanding the structure help in improving the quality of the heuristics. This paper presents a network model that learns a heuristic function capable of relating distant parts of the state space via optimal plan imitation using the attention mechanism which drastically improves the learning of a good heuristic function. The learning of this heuristic function is further improved by the use of curriculum learning, where newly solved problem instances are added to the training set, which, in turn, helps to solve problems of higher complexities and train from harder problem instances. The methodologies used in this paper far exceed the performances of all existing baselines including known deep learning approaches and classical planning heuristics. We demonstrate its effectiveness and success on grid-type PDDL domains, namely Sokoban, maze-with-teleports and sliding tile puzzles.

Keywords: Planning · Optimizing heuristic functions · Deep learning

1 Introduction

Classical Planning has always relied on strong heuristic functions to approximate distances to the nearest goal [3]. Its quality is measured by how well it performs when used inside a planner, i.e., it depends on the quality of the solution and the time taken to generate it. A major drawback of classical planning is the need to formulate problems by extensively capturing information from the environment. Recent years observe a progress in using visual representations to capture the specifics of a problem [2]. Yet, there is still a big gap between the length of optimal plans and the plans found by planners using learnt heuristic functions.

A significant amount of importance is given to developing deep networks that are able to learn strong heuristics [5] and policies [22]. In learning for planning,

N. Moniz et al. (Eds.): EPIA 2023, LNAI 14115, pp. 495–507, 2023.
https://doi.org/10.1007/978-3-031-49008-8_39

the methods rely heavily on either hand-coded logical problem representations [26] or deep convolution neural networks [8] that learns to imitate an expert. While there exists successful approaches in training neural networks (NNs) to learn heuristic estimates of various problem domains [8,25], designing a meaningful NN architecture to extract the relevant information from the data set is still an open-ended problem.

This work extends the work by [8,16] by addressing limitations of convolutional neural network, which capture only local dependencies. We propose to use self-attention and position encoding [24], as we believe a strong heuristic function needs to relate "distant" parts of the state space.

In our default experimental settings, NNs realizing heuristic functions are trained on plans of small problem instances created by classical planners. While this allows us to generalize across more difficult instances such that we can measure distances to optimal plan lengths, they do not achieve the best results for two reasons. First, even though the generalization of A*-NN is surprisingly good as will be seen below, there is still a scope of large-scale improvement on previously unseen, larger and more complex environments. Second, classical domain independent planners can solve only small problem instances anyway, which means that obtaining plans from large ones is difficult. We demonstrate that this problem can be partially mitigated by curriculum learning [4], where the NN is retrained/fine-tuned using plans from problems it has previously solved.

The proposed approach is compared to state-of-the-art domain-independent planners, namely SymBA*[21], Lama [15], and Mercury [10] and to currently best combination of A* and CNNs [8] on three grid domains: (1) **Sokoban** where each maze consists of walls, empty spaces, boxes, targets and an agent; the goal is to push the boxes to target locations; the boxes can only be pushed and not pulled in the game; (2) **Maze-with-Teleports** where the goal for an agent is to reach the goal position via interconnected teleports; (3) **Sliding-Tile** where blocks are moved to achieve an end configuration.

The paper is organised as follows. We first discuss the prior art in deep learning for planning, especially on problems where state can be represented as a tensor. Next, we review formal basics of classical planning. Then, we highlight the shortcomings of a prior state of the art and propose a solution that addresses some of these shortcomings. Here, we introduce the basics of the attention mechanism from NLP and explain the role of positional encoding in learning distances. Finally, the proposed networks are compared to other state of the art methods. In the last section, we discuss a few possible extensions of our work.

2 Related Work

The application of learning algorithms to improve planning dates back to the original STRIPS planner [6], which learned triangle tables or macros that could later be exploited by the planner. This approach attracted more interest as machine learning algorithms and has gained a steady popularity since. Earlier uses of NN to learn policies and heuristics for deducing strategic positions and

moves known to us considered chess boards [20] and backgammon [19]. Their use in Go [18] raised a considerable interest in public once it beat the top players in the game. Heuristic functions were also learnt for single agent games such as Sokoban [8,14] and Rubik's cube [1]. In 2011, a special learning track was introduced in the international planning competition (IPC), which concluded that the performance of NNs is promising in learning heuristic functions. A perpendicular approach to the above is to learn functions combining a portfolio of existing heuristic function [25] or to select a heuristic function from a portfolio [11]. A very interesting problem is to learn a transition operator as in [2] together with a visual execution of the plan, but this is outside the scope of our work.

Our work differs from the above as it focuses on (i) identifying *good building blocks* of the neural networks for grid domains and (ii) discusses the difficult and importance of a training set and shows that curriculum learning can be of a great help. The resulting networks are general and their performance exceeds that of prior art including SOTA classical planners.

3 Classical Planning

We construct our problem domains in a classical setting, i.e. fully observable and deterministic.

In classical planning, a STRIPS [6] planning task is defined by a tuple $\Pi = \langle F, A, I, G \rangle$. F denotes a set of facts which can hold in the environment (for instance, in Sokoban, a particular box at a particular position is a fact). A state s of the environment is defined as a set of facts holding in that particular s, i.e. $s \subseteq F$. The set of all states is, therefore, defined as all possible subsets of F as $S = 2^F$. $I \in S$ is the initial state of the problem and $G \subseteq F$ is a goal condition comprising facts which has to hold in a goal state. An action a, if applicable and applied, transforms a state s into a successor state s' denoted as $a(s) = s'$ (if the action is not applicable, we assume it returns a distinct failure value $a(s) = \perp$). All actions of the problem are contained in the action set A, i.e. $a \in A$. The sets S and A define the state-action transition system.

Let $\pi = (a_1, a_2, \ldots, a_l)$, we call π a plan of length l solving a planning task Π iff $a_l(\ldots a_2(a_1(I))\ldots) \supseteq G$. We assume a unit cost for all actions, therefore the plan length and plan cost are equal. Moreover, let π_s denote a plan from a state s, not I. An optimal solution (plan) is defined as a minimal length solution of a problem Π and is denoted as π^* together with its length $l^* = |\pi^*|$.

A heuristic function h is defined as $h : S \to R^{\geq 0}$ and provides an approximation of the optimal plan length from a state s to a goal state $s_g \supseteq G$, formally $h(s) \approx l^*$, where $l^* = |\pi_s^*|$.

In our experiments, we choose domains encoded in PDDL [7], where a planning problem is compactly represented in a lifted form based on predicates and operators. This representation is grounded into a STRIPS planning task Π, which is subsequently solved by the planner using a heuristic search navigating in the state-action transition system graph and resulting in a solution plan π.

4 Planner's Architecture

Given the initial state I, each partial plan $\pi = (a_1, a_2, \ldots, a_k)$, $k < l$ induces a sequence of states $(s_0 = I, s_1, s_2, \ldots, s_k)$ with $s_k = a_k(\ldots a_2(a_1(I))$.

States are for the purpose of the the neural network encoded as a binary values of all prepositions, which are in case of grid domains arranged in the same grid. The input to a neural network encoding a state is therefore a binary tensor, which is very well suited for contemporary deep learning libraries and execution on GPU. The output of the network is the heuristic value provided by the *value head*, which is in some cases supplemented by the distribution on all possible next actions provided by the *policy head* (the rationale for policy head is that according to [8], it improves the quality of the learnt heuristic). The training sets for neural networks consists of optimal plans for selected problem instances, which were generated by the optimal planner SymBA* [21]. More precisely, given the plans in the training set, we generate pairs $(s_i, \delta(s_i))$, where δ is cost of an optimal plan from s_i to the goal. For the sake of simplicity, $\delta(s_i)$ is the distance $l - i$ of the state s_i to the goal s_l in the optimal plan $(s_0 = I, s_1, s_2, \ldots, s_i, \ldots, s_l)$. Evaluating the network for a given state, directly serves as an estimator in our heuristic search planner. In curriculum learning, the training set with the optimal plans is augmented by newly found plans on more difficult problem instances. These newly found plans do not have to be optimal but are typically very close to being so.

For some domains (e.g. Sokoban), where the policy head is useful, the plans also contain the action input (s_{i-1}, a_i) needed to train the policy head. Since our aim is finding (close-to-)optimal plans, we used A* [9] as the search algorithm for exploring the planning state space. The training used ADAM [12] variant of stochastic gradient descent with default settings and a batch-size of 500.

5 The Proposed Neural Network

The best architecture of NN implementing a heuristic function for Sokoban known to us was proposed by Groshev [8]. We believe its biggest drawback is that it relies solely on convolution (which is strictly a local operator) thus limiting the neural network in synthesizing information from two distant parts of the maze. To understand the aforementioned statement, let us introduce some formal notations.

Let the input to the neural network be denoted by $\boldsymbol{x} \in R^{h,w,d_0}$, where h and w is the height and width of the maze respectively, and d_0 varies with the number of channels as explained above. Intermediate outputs are denoted by $\boldsymbol{z}^i = L_i(\boldsymbol{z}^{i-1})$, where L is some neural network layer (consisting of convolution C etc.) and for the sake of convenience, we set $\boldsymbol{z}^0 = \boldsymbol{x}$. All $\boldsymbol{z}^{\cdot}$ are three dimensional tensors, i.e. $\boldsymbol{z}^i \in R^{h,w,d_i}$. Notice that all intermediate outputs $\boldsymbol{z}^i$ have the same width and height as the maze (ensured by padding), while the third dimension which is the number of output filter(s) differs. Value $\boldsymbol{z}^i_{u,v}$ denotes a vector created from $\boldsymbol{z}^i$ as $(\boldsymbol{z}^i_{u,v,1}, \boldsymbol{z}^i_{u,v,2}, \ldots, \boldsymbol{z}^i_{u,v,d_i})$. Below, this vector will be called a *hidden vector* at position (u, v) and can be seen as a description of the properties of this position.

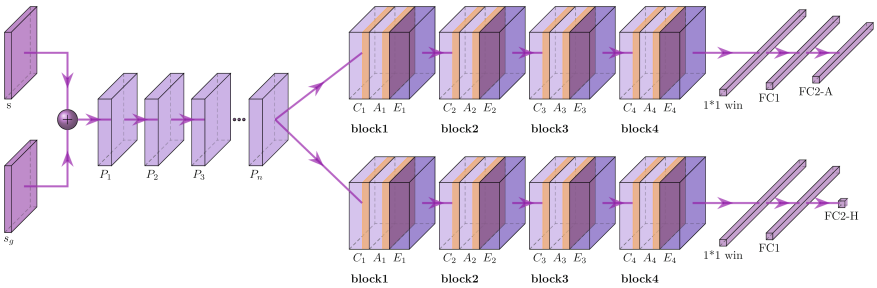

Fig. 1. The structure of our neural network. A current state s and a goal state s_g are fed into a variable number of pre-processing convolution (pre-conv) layers, $P_1..P_n$. In our case, we use 7 pre-conv layers. All convolution filters in the pre-conv layers are of the same shape 3×3 with 64 filters. Then the network splits into two branches and each branch has four blocks, each block containing 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 through padding. The output from block 4 is flattened by applying a 1×1 window around the agent's location before being passed onto the fully connected layers (FC1) and the action prediction output (FC2-A) and a single output for heuristic prediction (FC2-H). For the sake of picture clarity, skip connections are not shown in the neural network.

In Groshev's architecture [8] consisting of only convolution layers, the hidden vector $z^{i+1}_{u,v,\cdot}$ is calculated from hidden vectors $\{z^{i}_{u',v',\cdot} | u' \in \{u-1, u, u+1\}, v' \in \{v-1, v, v+1\}\}$, where the convolution has dimensions 3×3 and therefore uses information from a close neighborhood. Yet, we believe that any good heuristic requires features that relay information from different parts of the maze since Sokoban, Sliding-Tile and Maze-with-Teleports are all non-local problems. To address this issue, our network (see Fig. 1) features two additional types of layers, namely, the attention and the positional encoding layer, described below.

Convolution, Attention, and Position Encoding: The self-attention mechanism, [24] first introduced in NLP, allows to relate distant parts of input together. The output of self-attention from z^i is calculated in the following manner. At first, the output from previous layer z^i is divided into three tensors of the same height, width, and depth, i.e.

$$\boldsymbol{k} = z^{i}_{\cdot,\cdot,j} \quad j \in \left\{1, \ldots, \frac{d_i}{3}\right\}$$

$$\boldsymbol{q} = z^{i}_{\cdot,\cdot,j} \quad j \in \left\{\frac{d_i}{3} + 1, \ldots, \frac{2d_i}{3}\right\}$$

$$\boldsymbol{v} = z^{i}_{\cdot,\cdot,j} \quad j \in \left\{\frac{2d_i}{3} + 1, \ldots, d_i\right\}$$

then, the output z^{i+1} at position (u, v) is calculated as

$$\boldsymbol{z}_{u,v}^{i+1} = \sum_{r=1,s=1}^{h,w} \frac{\exp(\boldsymbol{q}_{u,v} \cdot \boldsymbol{k}_{r,s})}{\sum_{r'=1,s'=1}^{h,w} \exp(\boldsymbol{q}_{u,v} \cdot \boldsymbol{k}_{r',s'})} \cdot v_{r,s} \tag{1}$$

Self attention, therefore, makes a hidden vector $z_{u,v}^{j+1}$ dependent on all hidden vectors $\{z_{r,s}^{j} | r \in \{1, \ldots, h\}, s \in \{1, \ldots, w\}\}$, which is aligned with our intention. The self-attention also preserves the size of the maze. A multi-head variant of self-attention means that z^i is split along the third dimension in multiple $\boldsymbol{k}$s, $\boldsymbol{q}$s, and $\boldsymbol{v}$s. The weighted sum is performed independent of each triple (k, q, z) and the resulting tensors are concatenated along the third dimension. We refer the reader for further details to [24].

While self-attention captures information from different parts of the maze, it does not have a sense of a distance. This implies that it cannot distinguish close and far neighborhoods. To address this issue, we add positional encoding, which augments the tensor $\mathbf{z}^i \in R^{h,w,d_i}$ with another tensor $\mathbf{e} \in R^{h,w,d_e}$ containing outputs of harmonic functions along the third dimension. Harmonic functions were chosen, because of their linear composability properties [24].[1] Because our mazes are two dimensional, the distances are split up into row and column distances where $p, q \in [0, d_i/4)$ assigns positions with sine values at even indexes and cosine values at odd indexes. The positional encoding tensor $\mathbf{e} \in R^{h,w,d_e}$ has elements equal to

$$\begin{aligned} \boldsymbol{e}_{u,v,2p} &= \sin\left(\theta(p)u\right) & \boldsymbol{e}_{u,v,2p+1} &= \cos\left(\theta(p)u\right) \\ \boldsymbol{e}_{u,v,2q+\frac{d_e}{2}} &= \sin\left(\theta(q)v\right) & \boldsymbol{e}_{u,v,2q+1+\frac{d_e}{2}} &= \cos\left(\theta(q)v\right), \end{aligned}$$

where $\theta(p) = \frac{1}{10000^{\frac{4p}{d_e}}}$. On appending this tensor to the input z^i along the third dimension, we get

$$z_{u,v,\cdot}^{i+1} = [z_{u,v,\cdot}^{i}, e_{u,v,\cdot}].$$

With respect to the above, we propose using blocks combining Convolution, Attention, and Position encoding, in this order (we call them CoAt blocks), as a part of our NN architecture. The CoAt blocks can therefore relate hidden vectors from a local neighborhood through convolution, from a distant part of the maze through attention, and calculate distances between them through position encoding, as has been explained in [23]. Since CoAt blocks preserve the size of the maze,[2] they are "scale-free" in the sense that they can be used on a maze of any size.

The input to the network is the current state of the game and a goal state, s and s_g, respectively. Each state is represented by a tensor of dimensions equal to

[1] The composability of harmonic functions is based on the following property $\cos(\theta_1 + \theta_2) = \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) = (\cos(\theta_1), \sin(\theta_1)) \cdot (\sin(\theta_1), \sin(\theta_2))$, where $\cdot$ denotes the inner product of two vectors, which appears in Eq. (1) in inner product of $\mathbf{q}_{u,v}$ and $\mathbf{k}_{r,s}$.

[2] Convolution layers are appropriately padded to preserve sizes.

width and height (fixed to 10×10 for Sokoban, 15×15 for Maze-with-Teleports, and 5×5 for Sliding-Tile) of the maze $\times$ objects. The objects stand for one-hot encoding of the object states on a grid position (e.g., for Sokoban, we have wall, empty, box, agent and box target, for Maze-with-Teleport agent, wall, floor, goal and teleports 1–4, for sliding tile, we have a channel for each number, all of which can be derived automatically from the grounded representation.

6 Curriculum Learning

The second contribution of our work is to promote curriculum learning [4]. One of the drawbacks in learning the heuristic functions for planning is that when generating training set with existing planners, we quickly hit the limit of their capability in solving more complex problems. This can limit the learnt heuristic function in solving more complex problems. To further improve our heuristic function to scale to bigger problems, we re-train our network on an extended training set, which includes harder problem instances.

The protocol used in the experimental section of this paper is as follows. We first train the heuristic network on a training set containing problem instances that are quickly solvable by an optimal planner. Then we use this NN as a heuristic function inside A* search to solve more difficult problem instances. Their solutions are used to extend the training set on which the neural network is re-trained. By doing so, the NN is gradually trained on more difficult problem instances which improves its quality. As this procedure is fairly intuitive and yet computationally expensive, we demonstrate its effects on the Sokoban domain.

7 Experimental Results

This section first briefly describes the details of training the NN and then presents the experimental results on the selected benchmark domains: Sokoban, Sliding Tile and Maze-with-Teleports. For all the three domains, we use the output from the heuristic network inside A* to generate solutions. A* algorithms with learnt heuristic functions realized by the proposed convolution-attention-position networks (further denoted as A*-CoAt) are compared to A* with learned heuristic function realized by convolutional networks as proposed in [8] (denoted as A*-CNN), and to the state of the art planners LAMA [15], SymBA* [21], and Mercury [10]. We emphasize that A*-CNN and A*-CoAt uses vanilla A* search algorithm [9] without any additional tweaks. In case of Sokoban, we also compare our planner to a solution based on Reinforcement Learning [14].

On all the compared domains, we analyse the strength of our learnt heuristic and generalization property by solving grid mazes of increasing complexities, approximated by the number of boxes in Sokoban, grids of higher dimensions in Sliding-Tile and Maze-with-Teleports, and rotated mazes in Maze-with-Teleports.

7.1 Training

Sokoban: The training set for Sokoban was created by randomly generating 40000 Sokoban instances using gym-sokoban [17]. Each instance has dimension 10×10 and it always contains only 3 boxes (and an agent). In each plan trajectory, the distance from a current state to the goal state is learned as the heuristic value, $h(s_i)$. In line with [8], the neural network also uses the policy head during training.

Maze-with-Teleports: The training set contained 10000 maze problems of dimension 15×15, generated by using a maze creator.[3] Random walls were broken to create teleports. We added a total of 4 pairs of teleports that connect different parts of the maze inside each training sample. The mazes for training were generated such that the initial position of the agent was in the upper-left corner and the goal was in the lower-right corner. Later, in our evaluations, we rotate each maze to investigate whether the heuristic function is rotation independent.

Sliding puzzle: The training set contained 10000 puzzles of size 5×5. These puzzles were generated using.[4] During evaluation, we test our approach on puzzles of higher dimensions such as 6×6 and 7×7, all of which were generated with.[5] We ensured that all the puzzles in the test and train set are solvable.

7.2 Comparison to Prior State-of-the-Art

Sokoban: The evaluation set consists of 2000 mazes of dimensions 10×10 with 3, 4, 5, 6 or 7 boxes (recall that the training set contains mazes with only 3 boxes). Unless said otherwise, the quality of heuristics is measured by the relative number of solved mazes, which is also known as *coverage*. Table 1 shows the coverage of compared planners, where *all* planners were given 10 minutes to solve each Sokoban instance. We see that the classical planners solved all test mazes with 3 and 4 boxes but as the number of boxes increase, the A*-NN starts to have an edge. On problem instances with 6 and 7 boxes, A*-CoAt achieved the best performance, even though it was trained only on mazes with 3 boxes. Thus, the NNs have successfully managed to *extrapolate* to environments with more complex problems. The same table shows, that A*-CoAt offers better coverage than A*-CNN, and we can also observe that curriculum learning (see column captioned curr.) significantly improves the coverage.

We attribute SymBA*'s poor performance to its feature of always returning optimal plans while we are content with sub-optimal plans. LAMA had even lower success in solving more complicated mazes than SymBA*, despite having the option to output sub-optimal plans. To conclude, with an increase in the complexity of the mazes, the neural networks outshine the classical planners which makes them a useful alternative in the Sokoban domain.

[3] https://github.com/ravenkls/Maze-Generator-and-Solver.
[4] https://github.com/levilelis/h-levin/.
[5] https://github.com/YahyaAlaaMassoud/Sliding-Puzzle-A-Star-Solver.

Table 1. Fraction of solved Sokoban mazes (coverage, higher is better) of SymBA* (SBA*), Mercury (Mrcy), LAMA, A*-CNN (caption CNN) and the proposed A*-CoAt (caption CoAt). A*-CNN and A*-CoAt (with caption normal) use networks trained on mazes with three bozes; A*-CoAt (with caption curr.) used curriculum learning. The quality of plans (not shown here) generated by CoAt are very close to the optimal while in the case of CNN, it is not always so.

				Normal		curr.
#b	SBA*	Mrcy	LAMA	CNN	CoAt	CoAt
3	**1**	**1**	**1**	0.92	0.94	0.95
4	**1**	**1**	**1**	0.87	0.91	0.93
5	**0.95**	0.75	0.89	0.83	0.89	0.91
6	0.69	0.60	0.65	0.69	**0.76**	**0.85**
7	0.45	0.24	0.32	0.58	**0.63**	**0.80**

CoAt network without curriculum learning is also on par with Deep Mind's implementation of Reinforcement Learning (DM-RL) in solving Sokoban [14]. Instead of re-implementing DM-RL by ourselves, we report the results on their test set[6] containing 10×10 Sokoban mazes with 4 boxes. While DM-RL had a coverage of 90%, our A*-CoAt (trained on mazes with three boxes) has a coverage 87%. A*-CoAt with curriculum learning has a coverage of 98.29%,[7] which greatly improves over the DM-RL. Taking into account that DM-RL's training set contained 10^{10} state-action pairs from mazes **with 4 boxes**, A*-CoAt achieves higher coverage using a training set which is several orders of magnitude smaller.

Maze-with-Teleports: The evaluation set contains a total of 2100 training samples of dimensions 15×15, 20×20, 30×30, 40×40, 50×50, 55×55 and 60×60. Each maze in the evaluation set contains 4 pairs of teleports that connect different parts of the maze. From Table 2, we see that the performance of A*-CNN and A*-CoAt (initially trained on 15×15 mazes) is the same as SymBA*[8] for dimensions up to 40×40 and is consistently better for problem instances of size 50×50, 55×55 and 60×60.

All "No Rotation" mazes were created such that the agents start in the top left corner and the goal is in the bottom right corner. This allows us to study to which extent the learnt heuristic is rotation-independent (domain independent planners are rotation invariant by default). The same Table therefore reports fraction of solved mazes that have been rotated by 90°, 180° and 270°. The results clearly show that the proposed heuristic function featuring CoAt blocks generalizes better than the one utilizing only convolutions, as the solved rotated

[6] Available at https://github.com/deepmind/boxoban-levels.

[7] https://github.com/deepmind/boxoban-levels/blob/master/unfiltered/test/000.txt.

[8] The planners and NNs were given 10 minutes to solve each maze instance.

Table 2. Fraction of solved mazes with teleports (coverage) of SymBA*, A* algorithm with convolution network [8] (denoted as CNN) and that with the proposed Convolution-Position-Attention (CoAt) network. 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.

		No Rotation		90° rotation		180° rotation		270° rotation	
Size	SBA*	CNN	CoAt	CNN	CoAt	CNN	CoAt	CNN	CoAt
15×15	1	1	1	1	1	1	1	1	1
20×20	1	1	1	1	1	1	1	1	1
30×30	1	1	1	1	1	1	1	1	1
40×40	1	1	1	1	1	1	1	1	1
50×50	0.92	0.94	**1**	0.91	1	0.92	1	0.91	1
55×55	0.55	0.78	**0.89**	0.71	0.85	0.70	0.87	0.69	0.87
60×60	–	0.73	**0.76**	0.68	0.75	0.66	0.74	0.68	0.75

Table 3. Fraction of solved Sliding-tile mazes (coverage, higher is better) of SymBA* (SBA*), Mercury (Mrcy), LAMA, A*-CNN (caption CNN) and the proposed A*-CoAt (caption CoAt). A*-CNN and A*-CoAt (with caption normal) use networks trained on mazes with three different dimensions; A*-CoAt (with caption curr.) used curriculum learning.

				Normal		Curr.
Size	SBA*	Mrcy	LAMA	CNN	CoAt	CoAt
5×5	0.54	0.32	0.89	0.72	0.83	**0.92**
6×6	0.21	–	0.25	0.56	0.72	**0.81**
7×7	–	–	–	0.35	0.41	**0.62**

instances of A*-CoAt network are comparable to the non-rotated case. Rotating mazes have no effect on SymBA* (the complexity is solely dependent on the grid size) and the coverage rate stays unaffected.

From the results in Table 2, it can be concluded that the CoAt blocks (1) improve detection of non-local actions (teleports) compared to state-of-the-art planners such as SymBA*; (2) learn 'useful' information from the mazes which makes the network robust to rotations; (3) learn to approximate distances inside the mazes which results in a scale-free heuristic function.

Sliding-Tile: The evaluation set contains a total of 200 test samples of dimensions 5×5, 6×6 and 7×7. From Table 3, we see that the performance of A*-CNN and A*-CoAt (initially trained on 5×5 mazes) consistently outperforms the performance of planners.[9] SymBA* is the most reliable of all the planners but performs poorly when compared to the NNs. Of the NNs, the CoAt network

[9] The planners and NNs were given 10 minutes to solve each maze instance.

solves a larger number of instances as compared to the CNN network and records an even higher improvement in coverage after curriculum learning.

8 Conclusion and Future Work

We have proposed a building blocks of neural network able to learn strong heuristic function for PDDL domains with an underlying grid structure without the need for any specific domain knowledge. It is to be noted that even though we have generated training data from a classical planner on small problem sizes, our proposed architecture is able to generalize and successfully solve more difficult problem instances, where it surpasses classical domain-independent planners, while improving on previously known state-of-the-art.

Our experiments further suggest that the learnt heuristic can further improve, if it is retrained/fine-tuned on problem instances it has previously solved. This form of curriculum learning aids the heuristic function in solving mainly large and more complex problem instances that are otherwise not solvable by domain independent planners within 10 minutes.

As future work, our next goal would be to better understand if the learnt heuristic function is similar to something that is already known, or something so novel that it can further enrich the field; i.e., what kind of underlying problem structure we can learn by which network type, possibly in the form of studying generic types [13].

We believe that an improvement in the heuristic function is tied to the generation of problem instances that inherently possess the right level of difficulty, by which we mean that they have to be just on the edge of solvability, such that the plan can be created and added to the training set. We are fully aware that the problem instance generation itself is a hard problem, but we cannot imagine the above solution to be better than specialized domain-dependent Sokoban solvers without such a generator (unless the collection of all Sokoban mazes posses this property).

We also question the average estimation errors minimized during learning of the heuristic function. It might put too much emphasis on simple problem instances that are already abundant in the training set while neglecting the difficult ones. We wish to answer some of the above question in the future in an endeavour to generate strong, scale-free heuristics.

Acknowledgments. This work has been supported by project numbers 22-32620S and 22-30043S from Czech Science Foundation and OP VVV project CZ.02.1.01/0.0/0.0/16_019/0000765 “Research Center for Informatics”.

References

1. Agostinelli, F., McAleer, S., Shmakov, A., Baldi, P.: Solving the rubik's cube with deep reinforcement learning and search. Nature Mach. Intell. **1**(8), 356–363 (2019)
2. Asai, M., Fukunaga, A.: Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. arXiv preprint arXiv:1705.00154 (2017)
3. Bonet, B., Geffner, H.: Planning as heuristic search. Artif. Intell. **129**(1–2), 5–33 (2001)
4. Elman, J.L.: Learning and development in neural networks: the importance of starting small. Cognition **48**(1), 71–99 (1993)
5. Ernandes, M., Gori, M.: Likely-admissible and sub-symbolic heuristics. In: Proceedings of the 16th European Conference on Artificial Intelligence, pp. 613–617 (2004)
6. Fikes, R.E., Nilsson, N.J.: Strips: a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3–4), 189–208 (1971)
7. Fox, M., Long, D.: Pddl2. 1: An extension to pddl for expressing temporal planning domains. J. Artif. Intell. Res. **20**, 61–124 (2003)
8. Groshev, E., Goldstein, M., Tamar, A., Srivastava, S., Abbeel, P.: Learning generalized reactive policies using deep neural networks. arXiv:1708.07280 (2017)
9. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
10. Katz, M., Hoffmann, J.: Mercury planner: Pushing the limits of partial delete relaxation. In: IPC 2014 Planner Abstracts, pp. 43–47 (2014)
11. Katz, M., Sohrabi, S., Samulowitz, H., Sievers, S.: Delfi: Online planner selection for cost-optimal planning. In: IPC-9 Planner Abstracts, pp. 57–64 (2018)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
13. Long, D., Fox, M.: Automatic synthesis and use of generic types in planning. In: AAAI, pp. 196–205. AAAI Press (2000)
14. Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., et al.: Imagination-augmented agents for deep reinforcement learning. Adv. Neural. Inf. Process. Syst. **30**, 5690–5701 (2017)
15. Richter, S., Westphal, M.: The lama planner: Guiding cost-based anytime planning with landmarks. J. Artif. Intell. Res. **39**, 127–177 (2010)
16. Schaal, S.: Is imitation learning the route to humanoid robots? Trends Cogn. Sci. **3**(6), 233–242 (1999)
17. Schrader, M.P.B.: gym-sokoban. github.com/mpSchrader/gym-sokoban (2018)
18. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
19. Tesauro, G.: Programming backgammon using self-teaching neural nets. Artif. Intell. **134**(1–2), 181–199 (2002)
20. Thrun, S.: Learning to play the game of chess. Adv. Neural. Inf. Process. Syst. **7**, 1069–1076 (1994)
21. Torralba, A., Alcázar, V., Borrajo, D., Kissmann, P., Edelkamp, S.: Symba*: A symbolic bidirectional a* planner. In: International Planning Competition, pp. 105–108 (2014)
22. Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill acquisition via transfer learning and advice taking. In: European Conference on Machine Learning, pp. 425–436. Springer (2006)

23. Tsai, Y.H.H., Bai, S., Yamada, M., Morency, L.P., Salakhutdinov, R.: Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. arXiv:1908.11775 (2019)
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł, Polosukhin, I.: Attention is all you need. Adv. Neural. Inf. Process. Syst. **30**, 5998–6008 (2017)
25. Virseda, J., Borrajo, D., Alcázar, V.: Learning heuristic functions for cost-based planning. Plan. Learn. 6 (2013)
26. Yoon, S.W., Fern, A., Givan, R.: Inductive policy selection for first-order mdps. arXiv preprint arXiv:1301.0614 (2012)