Model-free Automated Planning Using Neural Networks

Michaela Urbanovská and **Jan Bím** and **Leah Chrestien** and **Antonín Komenda** and **Tomáš Pevný**
Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague
{michaela.urbanovska,chreslea,antonin.komenda,pevnytom}@fel.cvut.cz,
jan.bim@datamole.cz

Abstract

Automated planning for problems without an explicit model is an elusive research challenge, which, however, if tackled, could provide general approach to problems in real-world unstructured environments. There are currently two strong research directions in the area of Artificial Intelligence (AI), namely, machine learning and symbolic AI. The former provides techniques to learn models of unstructured data but does not provide further problem solving capabilities on such models. The latter provides efficient algorithms for general problem solving, but requires a model to work with. In this paper, we propose a combination of these two areas, namely deep learning and classical planning, to form a planning system that works without a human-encoded model. The deep learning part extracts the model in a form of a transition system and a goal-distance heuristic estimator; the classical planning part uses such a model to efficiently solve the planning problem. Besides the design of such planning systems, we provide experimental evaluations comparing the implemented technique to classical model-based methods.

Introduction

The main focus of this work is to analyze the possibilities and limitations of using deep learning in combination with classical planning. Instead of replacing the planning process as a whole and trying to make the network learn a search algorithm, we decided to focus on partial replacement of two components involved in many standard planning algorithms, namely, the transition system and heuristic functions.

Classical planning provides great methods for general problem solving. Unfortunately, these methods can struggle in large unstructured domains. On the other hand, deep learning methods have been demonstrated to work well on many domains without a clear structure. Therefore, combining both of these methods may remove the need for an explicit planning model.

Asai and Fukunaga in (Asai and Fukunaga 2017) and (Asai and Fukunaga 2018) connected deep learning and classical planning by creating LatPlan, which is a system that takes in an initial and a goal state of a problem instance and returns a visualised plan execution. The image on the input is transformed and processed in order to generate a standardized problem representation, which can then be solved by classical planning methods. In (Garrett, Kaelbling, and Lozano-Pérez 2016), Garret et al. uses machine learning techniques to create heuristic functions that improve a search algorithm. Finally, in (Gomoluch et al. 2019), learning policies for search algorithms is provided.

This work can be understand as a follow-up work of Asai and Fukunaga's architecture. In contrast to their work, we use maze-like problems and images of these mazes are the input to our algorithm. The solution is then produced as a sequential plan navigating through our designed transition system, which in turn is generated by our learned model. To increase efficiency of the search, we use heuristic principle proposed by Garret et al.

Background

Classical Planning

Let's first focus on STRIPS, which provides a symbolic representation to a model-based planning problem instance.

Definition 1 (STRIPS Planning Task) *A STRIPS planning task* Π *is a tuple*

$$\Pi = \langle F, O, s_i, s_g, c\rangle$$

where $F = \{f_1, f_2, ..., f_n\}$ *is a set of facts, which can hold in the world. A state of the world is defined by facts which hold in the state* $s \subseteq F$. $O = \{o_1, o_2, ..., o_m\}$ *is a set of operators transforming the world,* s_i *is the initial state, which consists of facts that hold in the initial state,* s_g *is a goal state condition, which contains facts that hold in every goal state and* c *is a cost function* $c(o) : o \rightarrow R^+$ *which gives each operator a positive cost.*

Every operator $o \in O$ *is a tuple where* $o = \langle pre(o), add(o), del(o)\rangle$, $pre(o) \subseteq F$ *is a set of preconditions, which are facts, that have to hold in a state for the operator* o *to be applicable in that state,* $add(o) \subseteq F$ *is a set of facts, which are added to the state after applying the operator* o *in* s *and* $del(o) \subseteq F$ *are delete effects, which are facts that are no longer true after using the operator* o.

To create a STRIPS representation of a problem, we have to be able to construct the facts and the operators from the problem definition. To find a solution of such a problem, we construct a state-transition system, in which we look for a path to the goal state from the initial state.

Definition 2 (Transition System) *A transition system is a tuple* $\Sigma = \langle S, A, \gamma, c\rangle$, *where*

- S *is a finite set of states*
- A *is a finite set of actions*

- *$\gamma : S \times A \rightarrow S$ is a state-transition function. $\gamma(s, a)$ is defined iff a is applicable in s, with $\gamma(s, a)$ being the predicted outcome.*
- *$cost : A \rightarrow [0, \infty)$ is a cost function assigning a value to each action. The cost value can have various meanings, for example time, price or anything we want to optimize.*

Any problem Π defined as STRIPS by Definition 1 can be translated to a transition system Σ and solved by the means of path-searching algorithms.

In order to find a solution to a planning problem, we need to find a path through the induced transition system, which is typically done by one of the heuristic state-space search algorithms.

Definition 3 (State-Space Search) *State-space search algorithm performs search over a graph $G = (N, E)$, where N is a set of nodes and E is a set of edges. Having a planning problem $\Pi = \langle F, O, s_i, s_g, c\rangle$ and its induced transition system $\Sigma = \langle S, A, \gamma, c\rangle$, N corresponds to S and E corresponds to A. The search starts in s_i, expanding each found state with γ, until a goal state is reached. In that case, plan π can be returned as a sequence of actions applied at each expansion of the search in order to reach the goal state.*

State space of many problems can be very large and exhaustive search may not be the most efficient way to look for the solution. Generally speaking, state-space search can be done blindly, however, additional information can greatly improve its performance. This additional information is added in the form of heuristic functions.

Heuristic function $h(s) : s \rightarrow R^+$ maps any state $s \in S$ to a positive value. Heuristic function gives us an estimate of path length from the current state s to a goal state. A function, which always maps $h(s)$ to the length of shortest possible path is called perfect or optimal heuristic and is denoted as h^*.

Neural Networks

Neural networks have proved to be a very powerful tool in many different domains. Here, we use the current state of the art approach which uses feed-forward neural networks that learn through back-propagation using Stochastic Gradient Descent (Ruder 2016).

Our primary aim lies in creating two networks, each one to substitute a different part of the state-space search algorithm. First network is used to replace the state-transition function γ in order to generate possible successor states in the state-transition system as defined in Definition 2. The second network is used to replace a heuristic function $h(s)$, which returns a number representing an estimate of the distance from state s to a goal state.

These two networks are implemented using convolutional neural networks (CNNs) as described in (LeCun, Bengio, and Hinton 2015).

The problem domains, in our case, have image-like grid structures which makes CNNs a viable choice in trying to extract information from the representations.

By replacing these parts of the state-space search, we avoid the need of creating a symbolic representation of the states which would have been necessary if we were to adhere to classical architectures. Thus, in this manner, we make our problem domains model free.

Attention for Neural Networks

In the recent past, by introducing the Transformer architecture, attention networks have proven to be a great success as shown in (Vaswani et al. 2017). In general, attention allows the network to focus only on subsets of inputs and requires the creation of attention masks.

In this work, we use soft attention in which the network focuses on input values that are between 0 and 1 as opposed to hard attention where the network focuses on either zeroes or ones.

Masks are generated using convolutional layer and a softmax layer of the same width and heigth as the input, with all values summing up to one. On having such a mask, we can then multiply the input features, resulting in a modified input with some of the features "emphasized" by the attention mask. The layers, which generate the attention masks are also trained with the whole architecture.

Data Domains

One of the key challenges of implementing the proposed approach is obtaining a good quality data set in order to train the networks. As CNNs are well suited to process data sets organized in grids, we consider problem domains which can be represented by a grid.

We use four problem domains; for each one of them, we create generators in order to obtain enough data. Examples of all four domains are shown in Figure 1.

First domain is a maze with one agent and one goal. The agent can move only in its 4-neighborhood and every free cell in the maze is accessible by the agent.

The second domain is the same, except we have multiple goals in the map. Therefore, we call this one the multi-goal maze. The goal in this domain is reached when the agent arrives in one of the goals.

The third domain is a multi-agent maze where the same rules apply, but we have the number of agents greater than one and the same number of goals in the maze. All the agents have to move at the same time and the goals are not assigned to a specific agent. The goal is reached when every agent in the map stands on a goal.

The last domain we use is the Sokoban puzzle which is similar to the maze domains in terms of movement, but it is more complicated because of the added box entity. Agent can push a box if there's a free space behind it and it's not possible to push multiple boxes at the same time. The main goal is to move every box to an arbitrary goal position in the map. Once every box is on a goal position, we reach the goal state.

Expansion Network

In the state-space search (Definition 3), transition function takes in a current state of the search and returns all its suc-

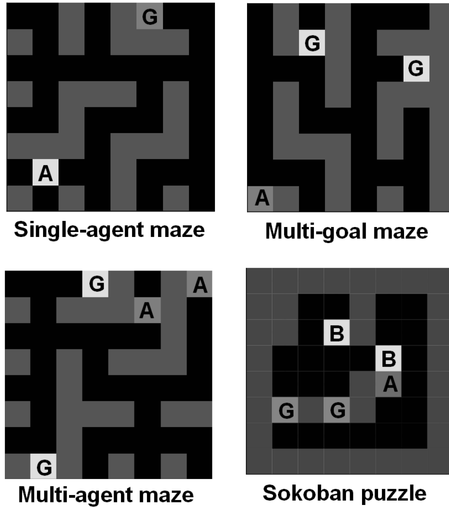

Figure 1: Examples of all four problem domains - **A** denotes agent, **G** denotes goal, **B** denotes box (in Sokoban domain)

cessors. The expansion network is used to generate these successors.

By observing pairs of states (in form of images of the mazes with the agent) without knowledge about actions that connect them, we want to learn possible actions for the problem. Even for the most simple maze domain, the size of the data set is important in order to train the network. The task does not only lay in the locating the free spaces around the agent. We need to make sure that the maze structure remains the same and no rules are broken on performing the learned actions.

We work with mazes represented as images, therefore, as mentioned earlier, it is convenient to use a CNN for this task. As we want to focus on the 4-neighborhood of the agent, we choose the kernel as a 3x3 window. Since we want to preserve the size of the input through the whole network, the padding for our CNN is equal to one.

To additionally improve the network, we use residual connections. A residual connection is an architecture modification, which is often used in deep learning and has achieved great results in learning an identity function, for example, in ResNet image classification network (He et al. 2016). Furthermore, residual connections resulted in a reduction in the complexity of the network and an improvement in the results.

Our expansion network has one residual connection which connects the input data with the output of the three chained convolutional layers. After concatenating these two parts of data, we process them through a 1x1 convolution to adjust the number of channels to match the input. In order to obtain even better results, we added normalization in the form of dropout between the first three convolutional layers. We can see the whole architecture in Figure 2.

Input and Output

The input of this network is the visual representation of the problem encoded in one-hot representation. This gives us a one-hot encoded vector for each cell in the problem's image, which tells us, which entity is in the cell based on the placement of the ones and zeros in the vector.

The output of this network is exactly the same in size, and it is similar to the one-hot encoding, however, the values on the output give us a distribution of the reachable next states. We can then use a threshold in order to extract the possible future states from this output.

Loss Function

In this network, we want to learn probabilistic distributions of the possible successor states of a current state. Our data is one-hot encoded, which means that there is a vector for each cell in the map or maze. According to the entity type placed on that cell, we put 1 to the corresponding index in the vector.

To train the network properly, we have to use a loss function suited for measuring the accuracy between true distribution and learned distribution of the possible successor states. Therefore, we use logistic cross entropy as our loss function for this network.

Heuristic Network

Another function in the state-space search (Definition 3) is the heuristic function which aids the search process. Computation of a heuristic is a non-trivial problem, especially in the case of non-simplified visual representation.

Since the input data is still based on the visual representation of the problem, convolutional networks are a good direction to explore. Inspired by the classical planning approaches for computing heuristics, we use attention to simplify the problem. Such simplification is usually used in classical planning heuristics in the form of relaxations or abstractions (Ghallab, Nau, and Traverso 2016).

Input and Output

The heuristic network receives a one-hot encoded visual representation of an input state. The label is the value of the h^* heuristic which is the length of the shortest path from the input state to the closest goal state. Therefore, we want the network to produce a single value for each input as well. This generated value is the heuristic in the planning algorithm.

Although learning a heuristic estimator from optimal plans sounds rather nonsensical (why to learn something we know the ground truth for), it acts in this work more as a proof of concept. The overall idea is to learn the heuristic from the best information about the distances to the goal. Since the learning process generalizes, it should be able to provide heuristic estimates even for cases in which it did not learn.

Loss Function and Data Set Structure

Training a network to return heuristic values requires optimal plan lengths as labels for all the data. For each maze

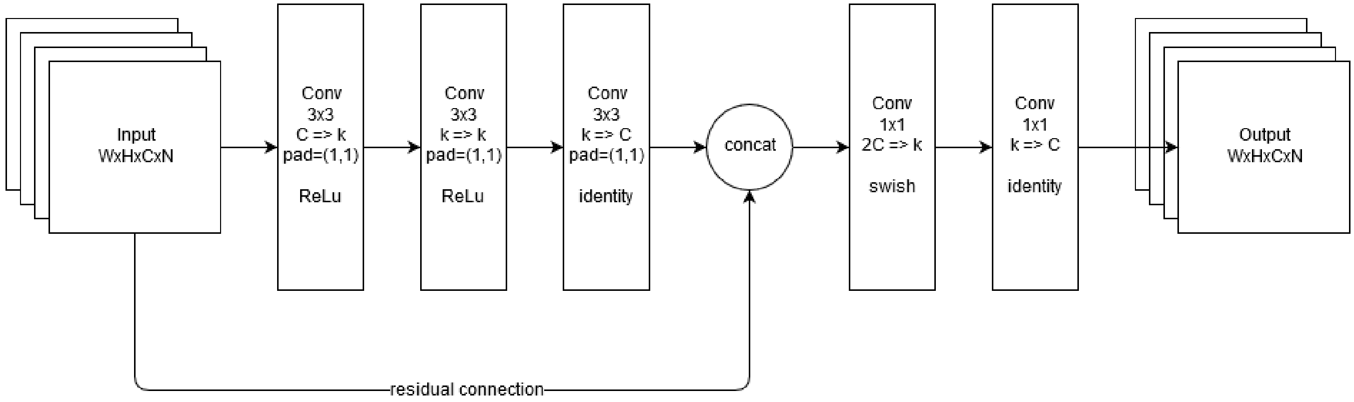

Figure 2: Expansion network architecture. Size of the input is width (W), heigth (H), number of channels (C) and number of samples in a batch. Each convolutional layer **Conv** has size of its kernel, number of input channels and output channels of the layer (C ⇒ k means **C** input channels and **k** output channels), padding and activation function. In this network, we use rectified linear unit (**ReLu**), **swish** (modified ReLu) or no activation (**identity**).

instance in the original data set, we randomly picked multiple positions from all possible agent placements and added those randomly picked samples with their computed labels (h^* values) into the data set. To train the network, we created the batches by taking a selected number of positions from each maze instance so that there were always multiple different agent placements in the maze at one batch. This is important, because loss function in this case has to be focused on the relation between the values in the same problem instance and not just the pair (state, value).

Since neural networks represent a black-box approximation scheme, we cannot expect to ensure any properties of the generated values. Therefore, we used satisficing planning for our experiments. In our case, the additional heuristic information was provided by the trained heuristic network.

Another property which can influence the performance, also held by the labels is monotonicity. Having monotonic heuristic values for all the states provides us with a possibility of selecting the best states on just following the descent of the state heuristic values. To get as close as possible to this property, we implemented a custom loss function, which measures how far off is the monotonicity of the learned values when compared to the h^* values.

```
function loss(mx,y)
partial_losses = []
for every maze instance in batch
    ixs = all indexes of the instance
    data_diffs = mx[ixs] − transpose(mx[ixs])
    labels_diffs = y[ixs] − transpose(y[ixs])
    tmp = −data_diffs ∗ sign(labels_diffs)+1
    l = sum(max(0, tmp)
    partial_losses.add(l)
end
return sum(partial_losses)
end
```

Listing 1: Pseudocode of loss for the heuristic network

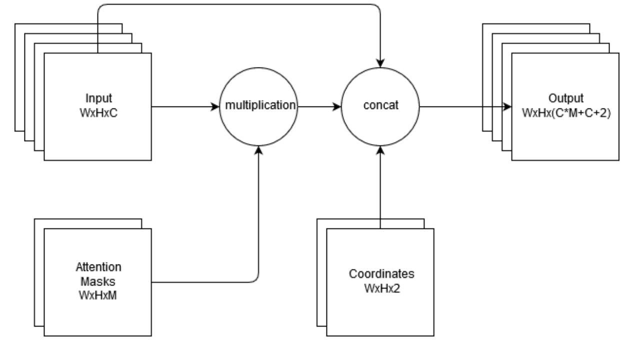

Figure 3: Attention block used in the heuristic network architecture - **W** is width of the input, **H** is height of the input, **C** is number of channels of the input structure

Architecture

One of the most notable features in this architecture is the usage of attention as described in section *Attention for Neural Networks*. The is analogous to the relaxation technique (Ghallab, Nau, and Traverso 2016) used in planning. If we imagine looking at a maze and identifying interesting parts of it, such as crossroads or long straight paths, we might simplify the problem enough to obtain a distance estimate from the agent to the goal.

Implementation of attention was done by using convolutional layers and using softmax over the first two dimensions of the input. Meaning, at the end, we received the attention mask, which has the same width and height as the input and all its values sum up to one (soft attention). One problem which comes up on using attention is deciding on the number of attention masks required to find enough attention-worthy places in the data. We experimented with different numbers of attentions in the architecture while selecting the networks that are further used in the planning experiments.

The input of this network is of the size $W \times H \times C \times N$; first two dimensions are width and height of the data; the third is the number of channels and the last one is number of samples in the processed mini-batch. Size of the mini-batch has been deliberately omitted in the architecture diagrams for simplification.

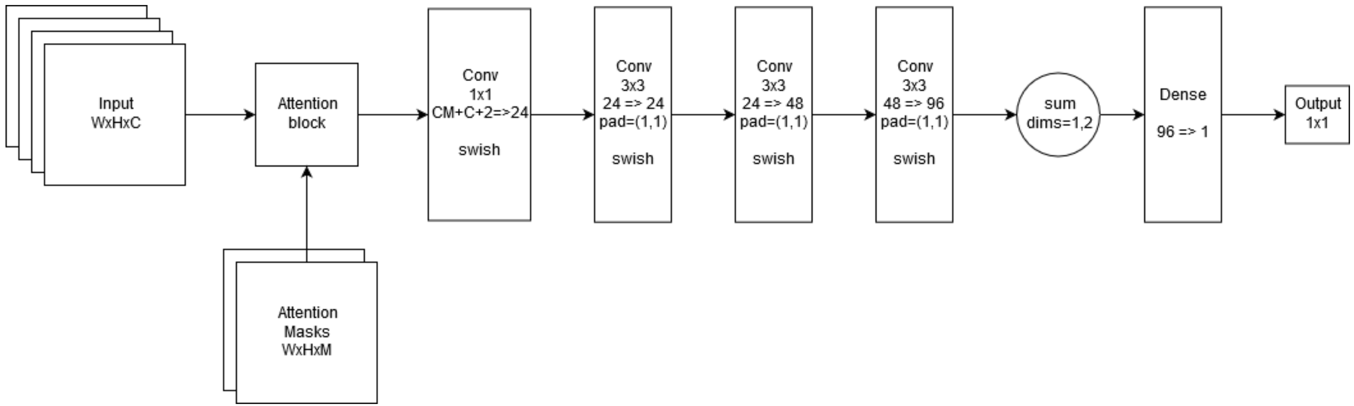

Figure 4: Heuristic network architecture

After creating M attention masks, we multiply the input data by each of the masks, thereby concatenating the results. This results in a data tensor of size $M \times C + C + 2$. We multiply each channel of the data by each attention and after concatenating the results with the original data, this multiplication is denoted in Figure 3 as "multiplication". Then, we add two last channels which are x and y coordinates for the mazes. It was shown in the (Wei et al. 2019) that for learning spacial information, it can be highly beneficial to provide coordinates for the data. Therefore, we added coordinates at the end of our data tensor. This whole computation happens in the "attention block" displayed in the Figure 3.

This created data tensor is then processed by multiple convolutional layers with same padding that keeps the width and height of the data the same throughout the whole network. After processing through the convolutional layers, aggregation in the form of a sum is performed over the first two dimensions, creating a vector of the same size as the number of channels in the last convolutional layer. Then, it is processed through a dense layer, returning one value, which is our final heuristic value for the input.

That is the top level description of the architecture. We experimented it with multiple small modifications to see if they influence the results. One modification is the number of attention layers which we mentioned earlier. The second modification which we denoted as an "attention block" states, how many times we repeat creating the attention masks and the large multiplication of these with the input data. One case is using is only once, as described above, at the beginning of the network. The other case is using five of them, one between each of the two convolutional layers. This case is displayed in Figure 5.

Experiments

Experiments in this work were conducted by training all described networks and then comparing their performance with techniques used in classical planning. To obtain each of the networks used in the planning experiments, we had to train dozens of its versions with different hyper-parameters to obtain the best possible one. Comparison of these trained networks was performed by evaluation functions.

In case of the expansion network, the adjusted hyper-parameters were number of channels in the convolutional layers, size of the convolutional kernel (in case of Sokoban), padding and number of epochs.

In case of the heuristic network, adjusted hyper-parameters were padding, number of channels in the convolutional layers, number of attention masks, number of used attention blocks and number of epochs.

Expansion Network Evaluation

Evaluation function for the expansion network is mostly used to check how accurately it can generate the possible successor states while also checking whether the network structure stays the same during the process. All the successor states in the output distribution also have to be valid and actually reachable. Same goes the other way - it is not desired to obtain any unreachable states in the output distribution. Based on these factors, we tested the trained expansion networks.

Since we use three domains which are very similar, we trained one expansion network for maze, multi-goal maze and multi-agent maze domains. Since the input dimension is different for Sokoban, we trained a separate expansion network for the Sokoban domain.

In Table 1, we can see results of the evaluation. **Wall difference** denotes the largest value assigned to a cell which is not supposed to be a wall. This means that we aim for the lowest possible values. In the evaluation table, it is obvious that wall placement is not a big problem for any of the networks.

Minimal correct step denotes the smallest value assigned to a cell which should contain an agent. The smaller the values, the less probable it is that agent can be located on the cell. In case of maze-exp-net, we can see that the value is 0.23101, which means that the least probable correct agent placement has this value. In the case of sokoban-exp-net, this value is very small, which means, that some of the possible successor states will not be discovered at all.

Maximal wrong step is analogical to the previous case. Here, we look for very small values because it denotes the highest possible wrong agent placement. In case of maze-exp-net, the value is very small, so there are no invalid successor states generated. In case of sokoban-exp-net, the value is very high, which means that invalid successor states can be generated during the search.

Therefore, the expansion network for Sokoban has not been successfully implemented and the complexity of the problem is probably higher then we first anticipated. This results in bad results for the planning experiments that use this expansion network as a state-transition function.

Heuristic Network Evaluation

Evaluation of the heuristic network is a more complicated problem since we're trying to train a monotonic function. Therefore, we took a set of optimal plans and each step of the plan was assigned a value by the heuristic network. Then we ordered the steps

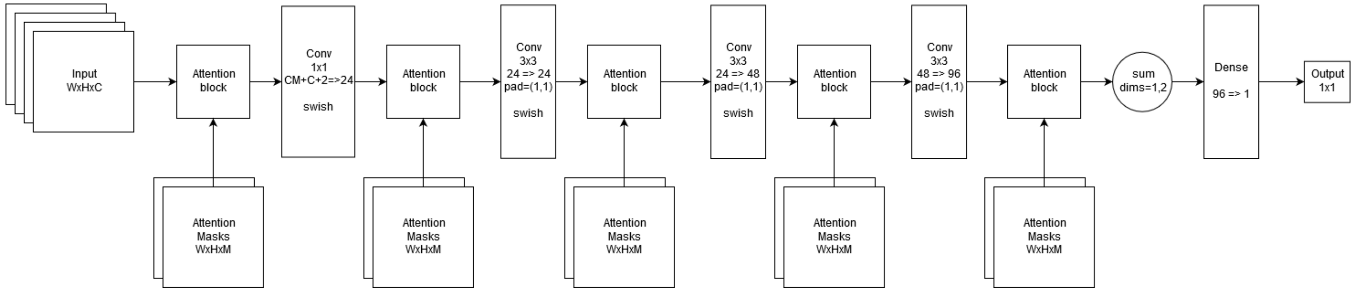

Figure 5: Heuristic network architecture using 5 attention blocks

Model	Wall difference	Minimal correct step	Maximal wrong step
maze-exp-net	1.85346e-5	**0.23101**	1.5056f-7
sokoban-exp-net	0.00241	1.33777e-11	0.96589

Table 1: Evaluation of expansion networks

by the heuristic network generated values and measured how far is the ordering from the ordering in the original optimal plan.

Planning Experiments

Based on the evaluation functions, we chose both expansion and heuristic networks which are used in the planning experiments. There are two expansion networks, one for Sokoban and one for all the three maze-related domains. There are also four heuristic networks, one for each problem domain.

For every domain, we implemented a planner with the same algorithms and the same heuristics. There are three state-space search algorithms. The first one is **greedy best-first search** (GBFS) which is guided by the computed heuristic values. State with the lowest value is always expanded first because the heuristic suggests it is closest to the goal.

Next one is the **best-first search algorithm** (BFS) with the same h value computation as in A*. This means, we don't expand the states based only on the heuristic value but we use sum of the heuristic with length of the path from the initial state to the current state.

The last one is **multi-heuristic search** (MH-GBFS) using tie-breaking (Röger and Helmert 2010). It uses a main heuristic and a second heuristic in case there's a tie between the main heuristic values. Since the order of heuristics is important in this case, we computed the experiments with all possible combinations of pairs of heuristics.

For heuristic, we can select a blind heuristic, Euclidean distance, H^{FF} heuristic (Hoffmann 2001), LM-cut (Pommerening and Helmert 2013) and the heuristic neural network. For use in multi-heuristic search, they can be arbitrarily combined.

Since we're in the field of satisficing planning, we want to compare our proposed methods against the state of the art in the field. One of the most successful planners in the community is LAMA (Richter and Westphal 2010), which uses H^{FF} heuristic together with landmarks. Therefore, we decided to implement both H^{FF} and LM-cut (which uses landmarks) and compare the approach with our heuristic network.

For each problem domain, we created 50 new problem instances which are not in any of the training data sets we used earlier. To compare the performance of all the planner configurations, we measured length of the output plans and number of expanded states during the search. There is also coverage denoted as **cvg**, which states the percentage of solved problems from the set of 50. All the experiments were executed with time limit set to 10 minutes per one problem instance on the same hardware.

Coverage Comparison We decided to compare the regular state-transition function and the expansion network by measuring coverage of the planners. Coverage tells us how many of the given problems were solved by the planner in a given time limit per problem.

In Tables 2, 3 and 4, we can see coverage for each of the state-space search algorithms used in the planning experiments. One big difference between the regular state-transition function and the expansion network is the zero percent coverage on Sokoban. The expansion network for Sokoban was not a success and we did not manage to train a well functioning expansion network for this domain as we described earlier. On the other hand, for the rest of the domains, the coverage is the same as in the case of the regular state-transition function.

Heuristic Performance Comparison In order to compare performances of all heuristics provided in the planner, including the heuristic network, we measured length of the resulting plan and number of expanded states during the search.

In the maze domain, both GBFS and BFS results show that the performance of the heuristic network is comparable to the other heuristics. Length of the resulting plans is the same in most cases and the only heuristic outperforming the heuristic network in terms of expanded states is the LM-cut heuristic. In MH-GBFS, using the heuristic network as a primary heuristic function provides us with results comparable to other heuristics. LM-cut seems to be the only heuristic which is giving a lot better results in terms of the expanded states. Results of all three search algorithms are in Figure 6.

] In the multi-goal maze domain, the results are very similar. GBFS performance is the best possible, same as H^{FF} and LM-cut. In BFS, heuristic network even outperforms both of these in the average number of expanded states. The MH-GBFS results show that heuristic network as a primary heuristic gives great results. All results are in Figure 7.

In the multi-agent domain, the number of expanded states are a lot higher for the heuristic network, as we can see in Figure 8 for all three search algorithms. This can be caused by the complexity of this problem. Simultaneous movement of multiple agents in the maze is a lot more complicated than movement of just one. Also,

	State-transition function				Expansion Network			
	maze	mg-maze	ma-maze	sokoban	maze	mg-maze	ma-maze	sokoban
blind	1	1	1	1	1	1	1	0
euclid	1	1	1	1	1	1	1	0
hff	1	1	1	0	1	1	1	0
lmcut	1	1	0.94	0	1	1	0.94	0
nn	1	1	1	1	1	1	1	0

Table 2: Coverage comparison for GBFS

	State-transition function				Expansion Network			
	maze	mg-maze	ma-maze	sokoban	maze	mg-maze	ma-maze	sokoban
blind	1	1	1	1	1	1	1	0
euclid	1	1	1	1	1	1	1	0
hff	1	1	1	0	1	1	1	0
lmcut	1	1	0.94	0	1	1	0.94	0
nn	1	1	1	1	1	1	1	0

Table 3: Coverage comparison for BFS

	State-transition function				Expansion Network			
	maze	mg-maze	ma-maze	sokoban	maze	mg-maze	ma-maze	sokoban
euclid, hff	1	1	1	0	1	1	1	0
euclid, lmcut	1	1	0.74	0	1	1	0.78	0
euclid, nn	1	1	1	1	1	1	1	0
hff, eucl	1	1	1	0	1	1	1	0
hff, lmcut	1	1	0.98	0	1	1	0.96	0
hff, nn	1	1	1	0	1	1	1	0
lmcut, eucl	1	1	0.98	0	1	1	0.96	0
lmcut, hff	1	1	0.98	0	1	1	0.96	0
lmcut, nn	1	1	0.98	0	1	1	0.96	0
nn, eucl	1	1	1	1	1	1	1	0
nn, hff	1	1	1	0	1	1	1	0
nn, lmcut	1	1	0.82	0	1	1	0.82	0

Table 4: Coverage comparison for MH-GBFS

the goals are not assigned, therefore it becomes even more complicated in terms of computing the distance estimate to a goal state.

In Sokoban domain, H^{FF} and LM-cut were not capable of solving the problems in the given time limit. The coverage for both of these heuristics is equal to zero. In the GBFS, heuristic network found the shortest plans on average. Also, in BFS, the average number of expanded states was also the lowest when using the heuristic network. We can see the results in Figure 9.

This shows us that the heuristic network is performing better in domains with one agent. Another advantage of the heuristic network is its computation time. Compared to H^{FF} and LM-cut, the computation is much faster, especially on a complicated domain like Sokoban, as reflected in the coverage comparison.

Conclusions

In this work, we have proposed replacement of two key parts of search-based automated planning algorithm by deep neural networks. One network learns the planning model from image representation of state transitions. The other network learns heuristic function from image representations of states and their distances to a goal. Such architecture allows for use of automated planning for model-free problems. Experimentally, we have shown the efficiency of such search is on par with the classical planning heuristics and therefore a viable direction for future research.

Although the work provides promising results, it is preliminary in several aspects. First of all, the heuristic function is learned from optimal plans, which makes sense only as an optimistic placeholder for cleverly generated (sub-)sequences of actions towards goals (e.g. by action backward chaining). Other future work is to design a neural network usable for variable sizes of the inputs, i.e. one neural network for different maze/Sokoban puzzle sizes.

Acknowledgements The work was supported by the Czech Science Foundation (grant no. 18-24965Y).

References

Asai, M., and Fukunaga, A. 2017. Classical planning in deep latent space: From unlabeled images to pddl (and back). In *NeSy*.

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2016. Learning to rank for synthesizing planning heuristics. *arXiv preprint arXiv:1608.01302*.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

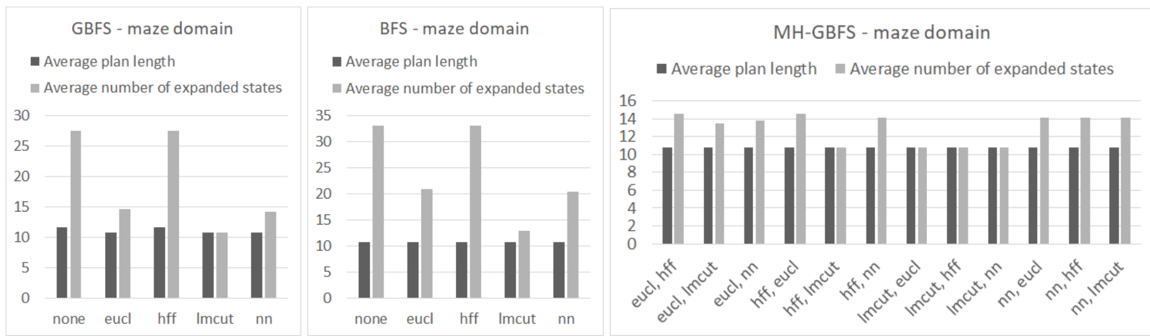

Figure 6: Performance of heuristics for maze domain in all search algorithms

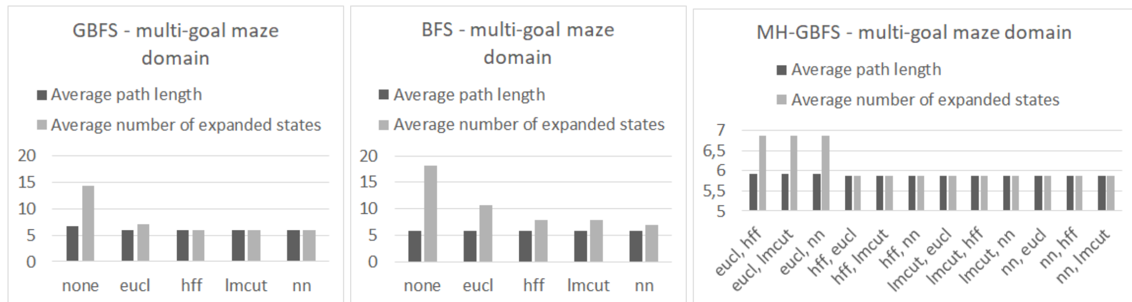

Figure 7: Performance of heuristics for multi-goal maze domain in all search algorithms

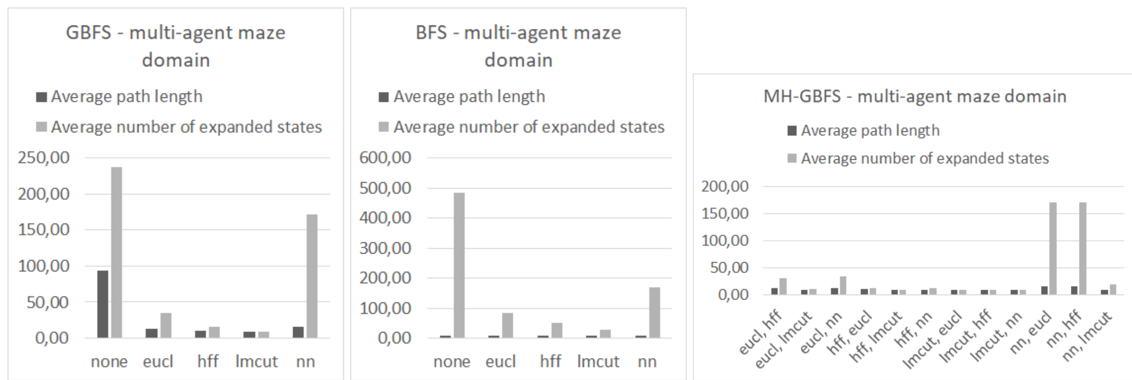

Figure 8: Performance of heuristics for multi-agent maze domain in all search algorithms

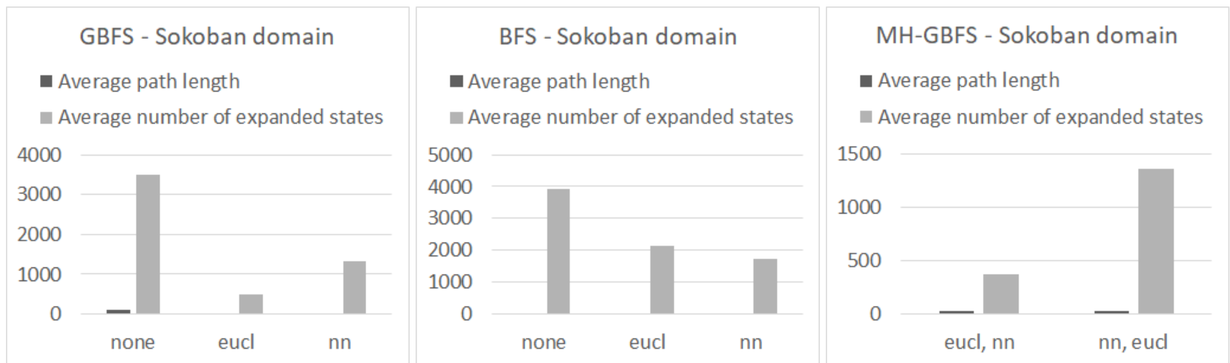

Figure 9: Performance of heuristics for Sokoban domain in all search algorithms

Gomoluch, P.; Alrajeh, D.; Russo, A.; and Bucchiarone, A. 2019. Learning neural search policies for classical planning. *arXiv preprint arXiv:1911.12200*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hoffmann, J. 2001. Ff: The fast-forward planning system. *AI magazine* 22(3):57–57.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553):436–444.

Pommerening, F., and Helmert, M. 2013. Incremental lm-cut. In *Twenty-Third International Conference on Automated Planning and Scheduling*.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Twentieth International Conference on Automated Planning and Scheduling*.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wei, X.; Bârsan, I. A.; Wang, S.; Martinez, J.; and Urtasun, R. 2019. Learning to localize through compressed binary maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 10316–10324.