

Learning Heuristic Estimates for Planning in Grid Domains by Cellular Simultaneous Recurrent Networks

Michaela Urbanovská and Antonín Komenda

*Department of Computer Science (DCS), Faculty of Electrical Engineering (FEE),
Czech Technical University in Prague (CTU), Karlovo namesti 293/13, Prague, 120 00, Czech Republic*

Keywords: Classical Planning, Simultaneous Recurrent Neural Networks, Heuristic Learning, Deep Learning.

Abstract: Automated planning provides a powerful general problem solving tool, however, its need for a model creates a bottleneck that can be an obstacle for using it in real-world settings. In this work we propose to use neural networks, namely Cellular Simultaneous Recurrent Networks (CSRN), to process a planning problem and provide a heuristic value estimate that can more efficiently steer the automated planning algorithms to a solution. Using this particular architecture provides us with a scale-free solution that can be used on any problem domain represented by a planar grid. We train the CSRN architecture on two benchmark domains, provide analysis of its generalizing and scaling abilities. We also integrate the trained network into a planner and compare its performance against commonly used heuristic functions.

1 INTRODUCTION

Classical planning is a powerful tool in terms of general problem solving. Its performance relies heavily on heuristic functions which provide further information about the problem and suggest which states to expand in search of the solution. There are many widely used heuristic functions that provide informed state-goal distance estimate, however, to compute such estimate we need to model the problem we try to solve.

Problem representation for classical planning often has to be done by hand and some problems can be too hard to describe and model in standardized languages such as PDDL (Aeronautiques et al., 1998). That creates a bottleneck which complicates usage of classical planning techniques on complex or very large domains that are too hard to be modeled by a human. That holds for many real-world problems such as logistic problems, distribution warehouses or fork-lift fleets.

One way to avoid creating such representation is using a graphical representation of the problem. In conjunction with neural networks we take the modelling part out of the equation and base the heuristic computation on graphical features and not a hand-coded model.

Combining classical planning with neural networks is currently widely discussed problem which is being tackled from many different perspectives.

In (Shen et al., 2020), authors use hypergraph neural networks together with the standardized modelling language to compute heuristic value for a given state. In (Groshev et al., 2018) the authors take similar path to ours where they use graphic representation of the problem and compute a policy which should be followed in the search algorithm.

Another approach which uses image representation is (Asai and Fukunaga, 2017) and (Asai and Fukunaga, 2018) where authors convert the image into latent space and further evaluate it using neural networks to obtain a heuristic.

One type of neural networks which have been successful with problems represented by an image-like grid are the Cellular Simultaneous Recurrent Networks (CSRN). In particular, they were used to solve the maze traversal problem. As mentioned in (Ilin et al., 2008), output of the network can be used to greedily guide an algorithm from a start state to a goal using values generated by the network as a heuristic function which means that it can be used in classical planning algorithms. There are works regarding the CSNR architecture such as (Ilin et al., 2006) and (White et al., 2010) which mostly focused on improving its training which is a great bottleneck of this approach.

In this work, we implemented CSRN for both maze and Sokoban domains based on implementation in (Ilin et al., 2008). We analyze training of the

CSRN architecture, discuss the parameters, training and evaluation. At the same time, we focus on its ability to generalize over different problem instances and discuss how to use it as a domain-independent heuristic function. At last, we integrate the CSRN in a classical planner and compare its performance against other heuristic functions.

2 BACKGROUND

Classical planning is a form of general problem solving that starts in a fully defined initial state and looks for a goal state by applying deterministic actions.

For each problem we need a set of facts which create the states and a set of actions which can be applied to said states. The two sets form a graph which can be referred to as a state-space of the problem. In this state-space we then look for a solution using state-space search algorithms.

2.1 State-space Search

Any search algorithm starts at the initial state and in each step applies all applicable actions to create more states which are further expanded until a goal state is reached. We can then reconstruct the path leading to solution by taking every expansion that led to the goal state from the beginning of the algorithm.

Such graph traversal algorithm can be computed blindly and states can be expanded in first-in first-out manner as a breadth-first search (Bundy and Wallen, 1984). However, the state-spaces tend to be large and exhaustive search may not be the most efficient way to find the solution. The performance of a search algorithm can be improved by using a heuristic function to guide it. There are many state-space search algorithms which use a heuristic function. In this work, we use Greedy Best-first Search (GBFS) which relies solely on the heuristic function when making a decision. Because we use a neural network to estimate the heuristic value and we have no guarantees on the generated heuristic values, we use GBFS in our experiments.

2.2 Heuristic Functions

Heuristic function is a function $h(s) \rightarrow \mathbb{R}^{0+}$ that takes a state s on the input and returns a single value that estimates how far is the given state from a goal. Based on the values provided by the heuristic function, the search can make much more informed decisions in which state to expand next.

In this work, we use multiple different heuristic functions in the experiments to compare their performance. The most simple heuristic is a blind heuristic, which gives each estimate equal to zero. Next used heuristic function is Euclidean Distance (ED) which computes Euclidean distance from current state to goal state on a grid. These two heuristics are very fast and simple to compute, but the information they add to the search may not be very valuable.

Another heuristics, we use in the experiments, are h^{FF} (Hoffmann, 2001) and LM-cut (Pommerening and Helmert, 2013) which are both widely used domain-independent heuristics that are used in planners such as LAMA (Richter and Westphal, 2010) or SymBA* (Torralba et al., 2014).

The h^{FF} heuristic uses a principle called relaxation of a problem. Relaxed problem is a modified version of a problem where we lift certain constraints (in this particular case, we enforce monotonicity of the state transitions) to make it easier to solve. h^{FF} then solves the relaxed version of the problem and reports the solution cost as a heuristic estimate for the original problem.

The LM-cut heuristic uses the principle of action landmarks. For every problem, there is a set of actions which have to be present in every existing solution. For example, to get to the goal in a maze, we always have to step to the tile next to the goal. Therefore, action that moves an agent to goal from a neighboring tile has to be present in the action landmark. The heuristic value can be then computed as a sum of the landmark action costs.

3 CELLULAR SIMULTANEOUS RECURRENT NETWORKS

Neural networks have proved to be a very powerful and versatile tool. Their possible connection with planning is still being explored (in works as stated in Section 1). In contrast to the existing work, here, we use a minimalistic recurrent neural network which is used as a component of the whole CSRN architecture.

CSRN can be used on domains which are represented as a grid. In (Ilin et al., 2008), it was used on mazes. The CSRN architecture technically copies the structure of the input and each of the grid tiles has its own small neural network that processes it. We will refer to this kind of network as a cell network. Every tile has its own cell network, but all the cell networks share weights, such that no matter how large the grid input is, the number of trained weights stays the same. That is one of the great advantages of this architecture. It also allows CSRN to scale over arbi-

trarily large inputs, i.e. being scale-free, which is not case in any of the previous works, to our best knowledge.

3.1 Cell Network

The cell network which operates over one grid tile can have arbitrary structure. We implemented the CSRN according to the (Ilin et al., 2008), therefore we decided to use the same cell network architecture as well.

The cell network is a small recurrent network that uses two fully connected layers as displayed in Figure 1. It takes a vector on the input which contains information about the tile (red arrows), its neighbors (green arrows) and values of the hidden states from the last iteration (blue arrows). The first fully connected layer (FC-1) takes the input vector of size v and computes h hidden states. The second fully connected layer (FC-2) takes the h hidden states and creates one value which represents heuristic value for the given grid tile. In case of the maze problem, the number estimates how many steps the agent has to take in order to find a goal in the maze if he starts at the given tile.

As we mentioned, the cell network is recurrent so the hidden states are a part of the input vector every following iteration. In every iteration, each cell network generates an output value using FC-2 (yellow arrow) which is then being fed into the neighboring cell networks in the next iteration (green arrows). All the cell networks run for the same number of iterations.

At the end of the last iteration, all cell networks return the final heuristic value for their cell. That leaves us with a matrix that contains a value for each tile of the input grid. These values represent the heuristic estimates for all the initial positions in the grid.

3.2 Training CSRN

In (Werbos and Pang, 1996), the authors showed that training such structure using Stochastic Gradient Descent (SGD) methods does not converge efficiently. We propose to use Bayesian Optimization (BO), as it is a universal approach, which does not require any additional assumptions on the objective function. E.g., for mentioned SGD, this function has to be differentiable, which heuristic functions are not in general. To train the CSRN, we use BO with Gaussian processes to optimize the objective function.

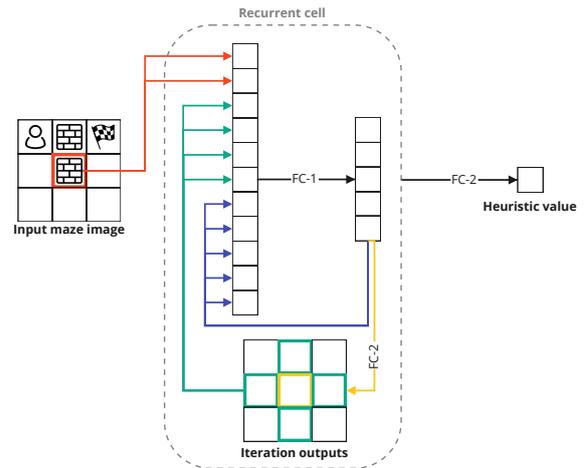


Figure 1: Cell network architecture. **FC-1** represents a fully connected layer which takes v inputs and returns h outputs; **FC-2** represents a fully connected layer which takes h inputs and returns one value; **red arrows** - fixed grid tile encoding; **green arrows** - neighbor values from previous iteration; **blue arrows** - feeding hidden states to next iteration; **yellow arrow** - computation of intermediate result for one iteration using FC-2.

3.3 Architecture Parameters

There are parameters of the cell network, but also for the CSRN structure as a whole.

The cell network, as described in (Ilin et al., 2008), requires **number of recurrent iterations** and **number of hidden states** as parameters.

The number of iterations influences how much of the actual search the cell network simulates during its run but it does not influence the number of trainable weights. It does, however, influence the run time of the network.

On the other hand, the number of the hidden states directly influences the number of trainable weights in the whole architecture.

Let us say that the input vector v consists of n values which describe the grid tile, m values which contain values of all tile's neighbors and h values which are the hidden state values from the previous iteration. The total number of trainable weights w in such network is then computed as

$$w = (n + m + h) * h + h \quad (1)$$

The CSRN can be used for any size of the input grid, therefore it is a scale-free architecture. For all our domains in this work, we parametrize the locality of the cells as a 4-neighborhood, as we only assume that the state can change by moving one cell up, down, left and right (intuitively representing a robot or agent on the grid). However, it is possible to define the neighborhood in a different manner depending on

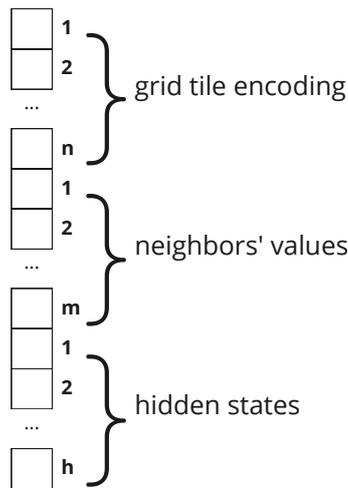


Figure 2: General input vector encoding. First n values represent the grid tile encoding. Next m values are iteration outputs from neighboring tiles. The last h values are hidden states computed in the last iteration.

the actions available in the domain we are solving. Deeper analysis of the action structure and its manifestation in the locality parametrization of the CSRNs is left for future work.

4 PROBLEM DOMAINS

As we already mentioned, (Ilin et al., 2008) uses maze traversal problem as the only domain which is also used in (Urbanovská et al., 2020). One property of the maze domain is that the 2D grid representation contains the whole state space of the problem. We can create every state in the state space by simply moving the agent in the maze onto every free tile on the grid. That led us to believe that the CSRNs' success may be caused simply by this property of the maze domain, which is in core a simple problem from the planning perspective. Intuitively, the only process the CSRN has to learn is iterative gradient propagation through the maze as shown in Figure 3. That creates an optimal heuristic for such problem, provided that the already filled cells are not overwritten.

Therefore, we decided to test out the CSRN architecture on Sokoban domain, which is one of the hardest classical planning problems (it falls into the PSPACE-complete class). It can be easily represented by a grid which makes it a perfect candidate for the CSRN architecture, but its whole state space certainly cannot be represented by one-dimensional 2D grid.

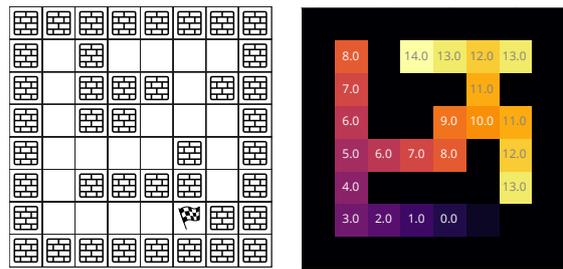


Figure 3: Gradient propagated through an example maze instance. The goal tile should be equal to 0, each tile next to it 1 and so on.

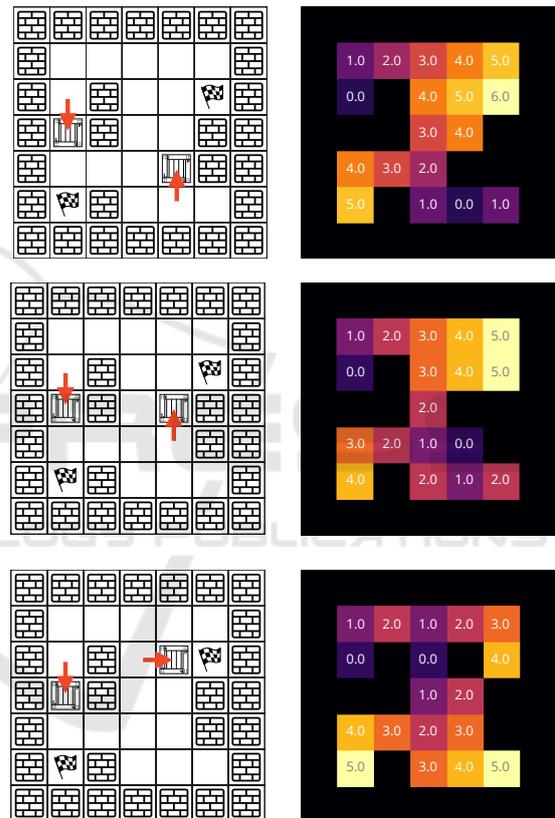


Figure 4: Gradient propagated through an example Sokoban instance in 3 steps. The desired actions are for each map configuration are displayed as red arrows. The gradient navigates the agent to a tile where it can make a decision that creates a new state that is closed to a goal state.

4.1 Input Encoding

The cell network takes a vector on the input, so its corresponding grid tile has to be processed accordingly. In Figure 2, we can see the general input vector structure for the cell network which is very similar for both maze and Sokoban domain.

First n values of the vector describe the objects present at the given grid tile. In case of maze it is rep-

resented by 2 binary values, one for a goal and one for a wall. In case of Sokoban there is one more binary value that represents the presence of a box. These values stay the same in every recurrent iteration of the cell network.

Next m values are output values from m neighboring grid tiles. In both domains we have only 4 possible actions which represent 4 possible movements on the grid, so 4 values in total. Each problem instance is surrounded by walls so the agent cannot step out of the world, therefore we can wrap the neighborhood around and not break any rules of the domain. That is crucial because we need every cell network to have input of the same size. The neighbor values change in every iteration based on the previous results of the neighboring cells.

Last h values are values of the hidden states. Their number is set by a selected parameter. These values change every iteration and at the start they are initiated to -1 .

4.2 CSRN Output

As described in Section 1, each cell network returns one value at the very end of the CSRN computation. In this section, we focus on the output as a whole where we work with all the values returned by all the cell networks.

Each grid tile is evaluated by a cell network, so we can create an output grid with all the generated values that has the same size as the input grid which represents the problem as shown in Figure 3.

As we already stated, complete state-space of a maze can be represented by the grid per se and we can clearly see how to interpret the output of the CSRN in this domain. Each cell in the output grid corresponds to a heuristic value for a state where agent stands on the given tile.

That is a great advantage for such simple domain as the maze domain because we can run the CSRN once and get heuristic values for the whole state-space at the same time. That means that the heuristic computation does not have to be repeated through the rest of the search algorithm which can greatly contribute to the algorithm's performance. Sadly this only holds for domains, where we can represent the whole state-space by the output grid.

Sokoban is a more complex domain and its state-space is much bigger and depends on different elements than just the position of the agent. Because of the boxes that are present, the problem becomes exponential and to create one output with all heuristic values we would have to create exponentially many grids with all box combinations to be able to evaluate

the CSRN only once.

Creating a multi-dimensional output of the CSRN is certainly an interesting direction for future research, but in this work we decided to use 2D projection of the problem to be able to interpret the output grid. The downside of this approach is that it is necessary to evaluate the CSRN every time a box moves. That is still less heuristic function calls than many heuristics use because it is very common to evaluate the heuristic function for every encountered state.

Thanks to this approach, we can then create a 2D output grid which contains heuristic values for all the agent positions in the world with particular box configuration. Therefore it is basically a projection over two variables (x, y) that describe agent's position. We could do the same for any two variables and create a 2D representation which can be used as a source of heuristic values for a subset of states. However, using agent's position seemed the most intuitive in this case.

Selection of the variable which are suitable for such projection is a direction of research of itself and it is not a problem, we will further discuss in this work. However, with good selection strategies, it would be possible to create such projection for arbitrary problem and therefore create a domain-independent framework that uses graphical representation on a grid.

5 CSRN TRAINING

Training of the CSRN architecture is a well-known bottleneck of the approach as stated in (Werbos and Pang, 1996). There have been many approaches to training the architecture, but overall the training can be very slow especially on a large number of samples. We use a very small number of training samples of small sizes because in this work we mainly focus on generalizing and scaling abilities of the CSRN architecture.

Due to these reasons, we decided to use BO with 5 steps and 3 restarts which runs for 5000 iterations. The objective function we want to optimize counts number of incorrect decisions the network would make in search using the learned heuristic. It is a function analogous to function used in (Ilin et al., 2008), thus the optimization algorithm aims to decrease the number of incorrect decisions made based on the learned heuristic values. Incorrect decision is such decision that is not present in any solution for the problem.

5.1 Objective Function

Before training the network, we have to create a structure that contains all "correct decisions". To do so, we generate the whole state-space for the given instance and find all solutions for every existing state. Such exhaustive search is costly, therefore, we only train the network on smaller number of problem instances that can be solved exhaustively in a reasonable time.

Once we have all plans for all possible states in the state-space, we go through all the plans and mark every decision that occurs in the plans. We know for sure that every one of these decisions leads to a goal state, because it is a part of a solution to the problem. That way, we are able to train the network not only on a subset of solutions but on all the existing solutions which hopefully provides us with more general knowledge about the problem domain.

In case of the maze domain, the exhaustive evaluation does not create a problem, as we can run a basic search algorithms from the goal and go through every tile in the maze which guarantees that we find all the possible plans.

Exhaustive evaluation for one Sokoban map is a very costly procedure. Therefore, we had to train the network only on very small samples we were able to exhaustively evaluate in a reasonable amount of time. One evaluation requires creating states for all possible box and agent position combinations and looking for all solutions in every created state.

It might seem that the evaluation function and creating all possible plans is rather a tedious approach. There are examples in the literature which use only optimal plans for training (Shen et al., 2020). Our reasoning behind training on all existing plans is that the network might be able to make more informed decisions in problematic parts of the state-space where other heuristics struggle or where the optimal solution may never lead to. We also hope it is going to prevent reaching dead-ends¹ which is a great problem specifically in the Sokoban domain. This approach prevents expanding the dead-end states, however, its behaviour in the dead-end is not being learned.

5.2 Parameter Selection

As stated in Section 3.3, there are several parameters in the CSRN architecture that have to be set. For the **number of recurrent iterations** and **number of hidden states** we trained architectures with different pa-

¹A dead-end state in planning is a state from which no goal state is reachable. An example of a dead-end in Sokoban is a state, where a box is in a corner, which is not a goal.

parameter combinations. In the (Ilin et al., 2008), the authors used 20 recurrent iterations and 15 hidden states for the network. However, that was for the maze domain. That is unfortunately not much information on how to modify the values in order to process the Sokoban domain. Therefore, we trained multiple differently parametrized architectures and based on their performance we used the most successful one in the planning experiments.

Another mentioned parametrization of the architecture is based on the neighborhood definition. In this case, in both the maze and Sokoban domains we have one agent which is allowed to move by one tile in its 4-neighborhood. Therefore the neighborhood function can be reused for both domains.

5.3 CSRN Convergence

Since we are using BO to train the network we wanted to demonstrate how the network converges on one maze sample. We are testing convergence of the objective function, thus we want the number of incorrect decisions made by the trained network equal to zero. The training was deployed 400 times with the limit of 1000 iterations and the results can be seen in Figure 5. We have observed that the success of the optimization heavily relies on the initial weights which are sampled randomly from given intervals.

In Figure 5 we see how the objective function converges to zero during the training. The lowest number of iterations is 0 which means that the initial iteration was good enough to find weights that yielded no wrong decisions. On the other hand, we see that the number of iterations can lead to hundreds. Even though the correct gradient propagation in only one maze instance is a very simple task, the initialization plays a great part in the speed of training. Lucky initialization can lead to almost immediate success while a bad initialization can lead to great prolonging. Some of the runs did not even finish with no mistakes in the given 1000 iterations.

6 EXPERIMENTS

We want to look at multiple properties of the trained CSRN and decide whether it is a suitable approach for neural heuristic learning for graphically represented problems.

First, we want to take the trained architectures and see their generalizing capabilities on different unseen problem instances. That will show us if training on a small number of samples provides the network with

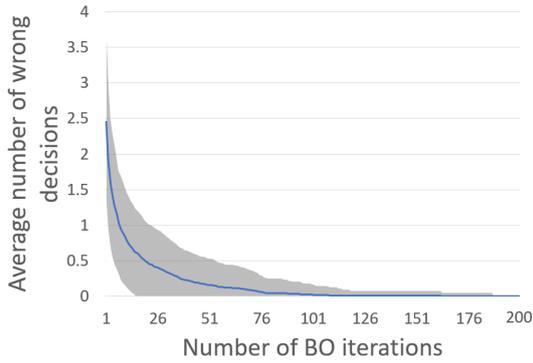


Figure 5: Convergence graph for training CSRN on one maze instance 400 times for the first 200 iterations. Blue line represents average number of wrong decisions per training iteration. The grey band represents standard deviation.

a knowledge base that can be reused on different but similar problems.

Next, we want to see how the CSRN scales-up with the learned knowledge. In case of the maze domain, we use it on larger instances and in case of the Sokoban, we create larger instances and instances with more boxes which can both increase the complexity of the problem.

The last experiments regard the planning performance which is also an important aspect of our work. We integrated the CSRN as a heuristic function into our classical planning solver and we want to compare the performance of CSRN in comparison with other heuristic functions on the same data sets.

All codes used for the experiments were implemented in Julia, including the domain-specific planners which use naive implementation of heuristics² described in Section 2.2. That way, we have a fair comparison in terms of the evaluation.

Due to the nature of our data we have implemented domain-specific planners in Julia together with all the mentioned heuristic functions. All experiments were performed on the same machine with 64 CPU cores, 346 GB of memory and a GPU for computation of the neural networks required in the planning process.

To evaluate the results we use the same metrics for all the experiments. First we look at **average path length** denoted as "avg_pl" which shows average solution length over the given data set. Next is the **average number of expanded states** denoted as "avg_ex" which shows average number of expanded states during the search. And last is the **coverage** denoted as "cvg" which shows solving success rate.

²Experimental codes were based on <https://github.com/urbanm30/nm-planning>.

6.1 Generalization and Scaling Experiments

In Section 5.2 we mentioned that we had to train multiple networks with different parameter configurations. We then integrated trained network in a planner and used them to solve different unseen data sets. Each data set in this branch of experiments contains 5 unseen problem instances. For each data set we also provided reference solution that has been achieved by using GBFS and the blind heuristic which is denoted as "ref" in Table 1 and Table 2. In both tables, each CSRN configuration is encoded as "number of recurrent iterations"—"number of hidden states".

CSRNs for the maze domain were trained on 5 problem instances of size 5x5. The parameters were selected from following

- number of recurrent iterations = [10, 20]
- number of hidden states = [5, 10, 15].

We then evaluated the trained networks on data set 5x5 which contains 5 unseen samples of size 5x5 and on data set 10x10 which contains 5 unseen samples of size 10x10.

CSRNs for the Sokoban domain were trained on 1 problem instance of size 3x3 with one box. The parameters were selected from following

- number of recurrent iterations = [10, 20, 30]
- number of hidden states = [15, 30].

Sokoban is a more complicated domain in terms of difficulty so we decided to scale-up both the size and the number of boxes. Each mentioned data set contains 5 unseen problem instances. First we have 2 data sets of size 3x3 with one and two boxes. Then we have 3 data sets with samples of size 6x6 with one, two and three boxes.

The search for the generalizing and scaling experiments was limited by number of expanded states for each sample which was set to 2x the reference solution.

6.1.1 Maze Domain

In Table 1 we can see results for all mentioned data sets and CSRN configurations. All configurations solved all problems in both data sets with the best possible solution lengths. The only difference is in the number of expanded states. All CSRN configurations managed to expand less states than the reference solution in both data sets. The configurations with best results are the **20-15** which achieved lowest number of expanded states on the 5x5 data set and the **10-10** configurations which achieved the lowest number of

expanded states on the 10x10 data set. These two configurations are further used in the planning experiments.

6.1.2 Sokoban Domain

In Table 2 we see results for all the CSRN configurations trained for Sokoban. The first two data sets which both have samples of size 3x3 were not a problem in terms of coverage because all CSRNs were able to solve all the problems. The first unsolved instances appeared in the 6x6 data set with one box.

Configuration with the highest coverage (most solved problems) is clearly **10-30** which did not solve only one problem and also has lower number of expanded states on 4/5 data sets.

From the solution length perspective, no CSRN was able to produce shorter solutions than the reference solver. Four configurations were able to find the same average solution length for 2 data sets. From these four configurations we selected the **30-30** because it has the highest coverage and also expanded less states than the reference point in 3/5 data sets compared to the other 3 configurations.

Based on the results, configurations **10-30** and **30-30** are further used in the planning experiments.

6.2 Planning Experiments

Each data set for the planning experiments contains 50 samples which are all unseen for the trained CSRN architectures. Each problem instance in the planning experiments is limited by a time limit of 10 minutes. For each domain, we used two trained networks that were evaluated as the best ones. In case of maze, we use CSRN configurations **10-10** and **20-15** which are denoted as CSRN-1 and CSRN-2. In case of Sokoban, we use CSRN configurations **10-30** and **30-30** which are placed in the same row in Table 3 as CSRN-1 and CSRN-2.

We can see all the results in Table 3. First four columns contain planning experiments where we used four data sets of different sized mazes - 8x8, 16x16, 32x32, 64x64. Last three columns in Table 3 contains three data sets of Sokoban data - 8x8, 10x10 - Boxoban, 16x16. The data used in 10x10 data set were sampled from the original Boxoban (Guez et al., 2018) data set.

The CSRNs were successful in the maze domain in terms of coverage because they were able to solve every problem instance in all four data sets. In terms of solution length, they also found optimal solutions for 3/4 data sets. A drawback we can see in the CSRN performance is the number of expanded states which is greater than even the simplest heuristic. That means

that the heuristic generated by CSRN was often misleading for the search and caused selection of wrong states in the algorithm. However, since the heuristic is precomputed once for the whole maze (see Section 4.2), the final efficiency of the search does not lag behind the classical methods.

The Sokoban domain is a lot more complex and size of the instances in the planning experiments is higher than in the generalizing and scaling experiments. In the 8x8 data set, we can see that both CSRN configurations were able to achieve 0.98 coverage and expand nearly half the states compared to the blind heuristic. The performance gets worse in the 16x16 data set where CSRNs cannot solve any of the given instances. In the 10x10 - Boxoban data set we can see that the CSRNs solved at least some of the instances but the coverage is very low.

We also want to address the low coverage of the LM-cut and h^{FF} heuristics which is caused by the complexity of the domain as well as the computational time they require. We can see that CSRN has an advantage over these two heuristics especially in the 8x8 data set. As we stated in Section 4.2, CSRN has to be evaluated only once for a new box configuration so the evaluation is not running for every new state and its outputs can be saved and reused. That seems to be a great advantage in the Sokoban domain. While using the CSRN we can expand more states quicker because when we compute heuristic (run the CSRN) it provides heuristic values for a whole subset of states instead of just one state.

6.3 Discussion

Results presented in 6.1 and 6.2 show us that the CSRN architecture is able to scale-up for both given domains. In case of the maze domain, we see impressive performance for all presented data sets and the planning experiments. This result is to be expected due to the domain's complexity.

Much more interesting are the Sokoban results where we see CSRN's ability to generalize over incomparably more difficult problem instances. Considering the fact that the CSRN was trained on empty 3x3 maze with one box its performance on 6x6 and 8x8 instances is very impressive. More importantly, it was fully able to adapt to problems with higher number of boxes. Adding boxes to the Sokoban problem presents a large number of complications for the heuristic. There are overlapping trajectories, extended movement rules as only one box can be pushed at a time and other aspects which make the problem more complex.

Table 1: CSRN evaluation performed on maze domain. Reference solver is GBFS with blind heuristic. Each CCSRN configuration is named by its number of recurrent iterations and number of hidden states connected with a dash. Best results are written in bold lettering.

	ref		10-5		10-10		10-15		20-5		20-10		20-15		
	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	
5x5	5.6	19.2	1	12.6	5.6	12.8	1	5.6	11.4	1	5.6	10.8	1	5.6	10.0
10x10	11.2	80.8	1	48.8	11.2	33.6	1	11.2	46.0	1	11.2	51.0	1	11.2	40.6

Table 2: CCSRN evaluation performed on Sokoban domain. Reference solver is GBFS with blind heuristic. Each CCSRN configuration is named by its number of recurrent iterations and number of hidden states connected with a dash. Best results are written in bold lettering.

size	boxes	ref		10-15		10-30		20-15		20-30		30-15		30-30		
		avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	
3x3	1	6.0	23.6	1	22.4	6.4	21.6	1	6.0	28.4	1	6.0	22.8	1	6.0	29.6
3x3	2	5.2	10.8	1	18.4	5.4	18.4	1	5.2	20.0	1	5.2	20.0	1	5.4	7.6
6x6	1	21.0	340.8	1	469.8	29.0	261.8	1	26.2	454.2	1	26.25	430.75	0.8	27.0	465.2
6x6	2	21.8	1210.6	1	950.5	0.4	49.25	997.5	0.8	39.5	951.5	0.4	40.75	1496.75	0.8	29.5
6x6	3	28.6	12421.0	1	18217.5	0.4	72.8	10731.0	1	57.8	11195.8	1	79.0	13134.25	0.8	51.5

Table 3: Planning experiments for both maze and Sokoban domains.

	Maze												Sokoban								
	8x8		16x16		32x32		64x64		8x8		10x10 - Boxoban		16x16		8x8		10x10 - Boxoban		16x16		
	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	avg pl	avg ex	
blind	11.64	27.42	23.36	96.82	46.2	267.28	104.72	1085.66	1	111.24	3.5k	1	1.2k	66k	1	30k	52k	0.54	0.54		
EID	10.76	14.58	23.36	48.14	46.2	129.7	104.72	561.44	1	31.10	0.5k	1	45.64	8.2k	1	115.33	12.6k	0.54	0.54		
h^*F	10.76	10.76	23.36	23.36	46.2	46.2	104.72	104.72	1	-	-	0.04	-	-	0	-	-	0	0		
LM-cut	10.76	10.76	23.36	23.36	-	-	-	-	0	-	-	0	-	-	0	-	-	0	0		
CSR-N-1	10.88	28.98	23.36	105.6	46.2	357.5	104.72	1572.68	1	41.65	1.7k	0.98	37.67	2.1k	0.06	-	-	0	0		
CSR-N-2	10.88	26.9	23.36	105.02	46.2	356.44	104.72	1607.86	1	47.76	2.1k	0.98	36.0	968.0	0.04	-	-	0	0		

Based on the results we can report that the CSRN is capable of generalizing and scaling not only in the maze domain which we consider a simple baseline but also in the Sokoban domain.

7 CONCLUSION

We successfully trained CSNR architecture on both maze and Sokoban domains and evaluated its scaling and generalizing ability on unseen problem instances. We also integrated trained CSRNs into a planner and compared their performance with other commonly used heuristic functions. As we already stated, we are using the image-like grid representation of the problems. Thanks to that, we can say that we work with a model-free planning framework because our planner does not require a domain / problem model for the computation.

The generalizing and scaling experiments showed that the CSRN architecture is able to generalize very well on maze domain where it outperformed the reference solution in all data sets. In case of the Sokoban domain, we were able to achieve 96% coverage and even though the CSRN was finding longer solution in general, it was also able to decrease number of expanded states compared to the reference solution.

The planning experiments showed that the CSRN for maze domain provided comparable results to the classical heuristics, however, it expanded a larger amount of states in the process. In the Sokoban domain, we saw a great coverage on the 8x8 data set which contained larger instances than the data sets in generalizing and scaling experiments. However, we also saw limitations of the trained configurations as the results on the other two data sets showed next to no coverage. That is caused by the size of the problem instances which influence the complexity as well. Still, training the network on one 3x3 sample provided us with great results on the 8x8 data set and a promising direction for a follow-up research of the CSRN ability to generalize.

These results show us that the CSRN architecture might be the right tool for grid-based domains in terms of heuristic computation. Other further research direction would be to explore achieving domain-independence of this approach. So far, we have two ways of achieving that. One is creating an algorithm that would select appropriate variables in the problem domain which would allow us to create a 2D projection of the problem. The other way is creating an alternative representation of the problem which would be still processed by the CSRN architecture but without the condition of the grid structure, similarly as

in Natural Language Processing using a linear vector representation, for instance.

Addressing these ideas could lead to a model-free, scale-free and domain-independent heuristic function learned by a neural network on small tractable problem samples. In the future, we would like to focus on these mentioned challenges and provide such framework that could process any given problem.

ACKNOWLEDGEMENTS

The work of Michaela Urbanovská was supported by the OP VVV funded project CZ.02.1.01/0.0/0.0/16019/0000765 “Research Center for Informatics” and the work of Antonín Komenda was supported by the Czech Science Foundation (grant no. 21-33041J).

REFERENCES

- Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., Weld, D., SRI, D. W., Barrett, A., Christianson, D., et al. (1998). Pddl— the planning domain definition language. Technical report, Technical Report.
- Asai, M. and Fukunaga, A. (2017). Classical planning in deep latent space: From unlabeled images to PDDL (and back). In Besold, T. R., d’Avila Garcez, A. S., and Noble, I., editors, *Proceedings of the Twelfth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy 2017, London, UK, July 17-18, 2017*, volume 2003 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- 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*.
- Bundy, A. and Wallen, L. (1984). Breadth-first search. In *Catalogue of artificial intelligence tools*, pages 13–13. Springer.
- Groshev, E., Tamar, A., Goldstein, M., Srivastava, S., and Abbeel, P. (2018). Learning generalized reactive policies using deep neural networks. In *2018 AAAI Spring Symposium Series*.
- Guez, A., Mirza, M., Gregor, K., Kabra, R., Racaniere, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., Wayne, G., Silver, D., Lillcrap, T., and Valdes, V. (2018). An investigation of model-free planning: boxoban levels. <https://github.com/deepmind/boxoban-levels/>.
- Hoffmann, J. (2001). Ff: The fast-forward planning system. *AI magazine*, 22(3):57–57.
- Ilin, R., Kozma, R., and Werbos, P. J. (2006). Cellular sn trained by extended kalman filter shows promise for

- adp. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 506–510. IEEE.
- Ilin, R., Kozma, R., and Werbos, P. J. (2008). Beyond feed-forward models trained by backpropagation: A practical training tool for a more efficient universal approximator. *IEEE Transactions on Neural Networks*, 19(6):929–937.
- Pommerening, F. and Helmert, M. (2013). Incremental lmcut. 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.
- Shen, W., Trevizan, F., and Thiébaux, S. (2020). Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 574–584.
- Torralba, A., Alcazar, V., Borrajo, D., Kissmann, P., and Edelkamp, S. (2014). Symba: A symbolic bidirectional a planner. *International Planning Competition*, pages 105–109. cited By 27.
- Urbanovská, M., Bím, J., Chrestien, L., Komenda, A., and Pevný, T. (2020). Model-free automated planning using neural networks. In *Proceedings of the 1st Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL) of the International Conference on Automated Planning and Scheduling*, pages 7–15.
- Werbos, P. J. and Pang, X. (1996). Generalized maze navigation: Srn critics solve what feedforward or hebbian nets cannot. In *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, volume 3, pages 1764–1769. IEEE.
- White, W. E., Iftekharuddin, K. M., and Bouzerdoum, A. (2010). Improved learning in grid-to-grid neural network via clustering. In *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*, pages 1–7. IEEE.