Explainability Insights to Cellular Simultaneous Recurrent Neural Networks for Classical Planning

Michaela Urbanovská and Antonín Komenda
Department of Computer Science (DCS), Faculty of Electrical Engineering (FEE), Czech Technical University in Prague (CTU), Karlovo náměstí 293/13, 120 00 Prague, Czech Republic

Keywords: Classical Planning, Cellular Simultaneous Recurrent Neural Networks, Semantically Layered Representation, Learning Heuristic Functions.

Abstract: The connection between symbolic artificial intelligence and statistical machine learning has been explored in many ways. That includes using machine learning to learn new heuristic functions for navigating classical planning algorithms. Many approaches which target this task use different problem representations and different machine learning techniques to train estimators for navigating search algorithms to find sequential solutions to deterministic problems. In this work, we focus on one of these approaches which is the semantically layered Cellular Simultaneous Neural Network architecture (slCSRN) (Urbanovská and Komenda, 2023) used to learn heuristic for grid-based planning problems represented by the semantically layered representation. We create new problem domains for this architecture - the Tetris and Rush-Hour domains. Both do not have an explicit agent that only modifies its surroundings unlike already explored problem domains. We compare the performance of the trained slCSRN to the existing classical planning heuristics and we also provide insights into the slCSRN computation as we provide explainability analysis of the learned heuristic functions.

1 INTRODUCTION

Classical planning is a field of symbolic Artificial intelligence that focuses on general problem-solving. Its connection to statistical machine learning has been explored in many different directions. Planning problems are often represented by a standardized language PDDL (Ghallab et al., 1998) that is difficult to formulate as a neural network input due to its logic-like structure.

Therefore, there are many different representations used across the existing works. Image representation has been used at (Asai and Fukunaga, 2017), (Asai and Fukunaga, 2018) and (Asai and Muise, 2020) to explore planning in the latent space and generating PDDL from problem images.

Learning heuristic functions and search policies has also been greatly explored. In (Toyer et al., 2020) authors use neural network architecture tailored according to the problem structure to avoid using the PDDL and then learn action policies for problem domains. Authors in (Shen et al., 2020) use hypergraph neural networks to learn a heuristic function from state-value pairs, using the PDDL language for building the neural network input. Another widely used approach is Graph Neural Networks (GNN) that were used to compute action policies by authors in (Ståhlberg et al., 2021), (Ståhlberg et al., 2022) and (Ståhlberg et al., 2023) exploring different learning methods in each of the works. The GNN-driven approach also works more with the logical representation of the problem.

Part of the existing approaches uses planning problems that are implicitly defined on a grid and can be easily transformed into an intuitive tensor representation that can act as a neural network input. One of the first works has been shown in (Groshev et al., 2018) where authors used Convolutional Neural Networks (CNN) to learn policies for search algorithms. Another approach was shown in (Urbanovská and Komenda, 2021) where authors used CNN and Recurrent Neural Networks (RNN) architectures to compute heuristic values for problems on grids. Another architecture used for this purpose is the Cellular Simultaneous Recurrent Networks (CSRN) that was used initially in (Ilin et al., 2008) to solve the maze-traversal problem and its usage was extended to different problem domains in (Urbanovska and Komenda, 2023) together with analysis of training this architecture. The last extension of this architec-

Urbanovská, M. and Komenda, A.
Explainability Insights to Cellular Simultaneous Recurrent Neural Networks for Classical Planning.
DOI: 10.5220/0012375800003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 592-599
ISBN: 978-989-758-680-4; ISSN: 2184-433X

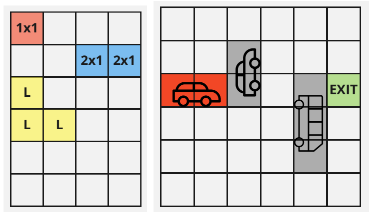

Figure 1: Example of a Tetris problem instance with 1 block of each type (left) and a Rush Hour problem instance with one small car and one large car (right).

ture has been proposed in (Urbanovská and Komenda, 2023) where the authors use a tensor representation motivated by the problems' semantics to train the semantically layered CSRN (slCSRN). The architectures trained in this work are all based on the architecture described in (Urbanovská and Komenda, 2023).

However, the number of problem domains available for the semantically layered representation proposed in (Urbanovská and Komenda, 022a) is still limited. Therefore, in this work, we extend the number of available domains by the Tetris and Rush Hour planning domains and propose their one-layer and multi-layer semantically layered representations. These domains differ from the existing (maze, Sokoban) domains since they do not have one explicitly defined agent who performs all the actions, but they contain independent movement of different objects in the problem. We train the slCSRN architecture variants for both of these domains and compare their performance with existing classical planning heuristics. We also analyze the computation of the architectures and provided several insights into what are the slCSRNs actually learning. That way we can make a step towards AI explainability, which we consider an important feature of machine learning in the modern days.

2 PROBLEM DOMAINS

We use two problem domains in this work. The first one is Tetris domain, inspired by the famous puzzle game. This problem domain has been used in the International Planning Competition. In the planning version of Tetris, there are three types of blocks (1×1, 2×1, and an L block) that are randomly placed on a grid as shown in Figure 1. The goal is to rotate and move the blocks, so they all fit into the bottom half of the grid, leaving the top half of the grid empty.

The second problem domain is the Rush Hour puzzle, inspired by the already existing board game. In Rush Hour, there is a 6×6 grid map that contains randomly placed small (1×2) and large (1×3) cars that are blocking the route for the red car that you

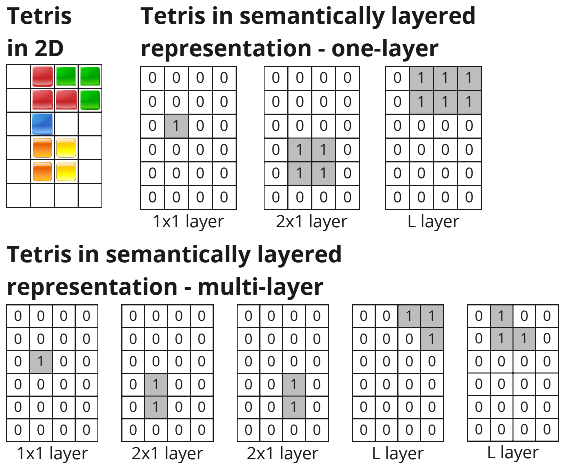

Figure 2: Semantically layered representation for a Tetris problem instance - both one-layer and multi-layer representation.

have to slide across the whole grid to reach the exit. The complexity of this puzzle is P-SPACE complete (Flake and Baum, 2002).

Both of the used problems do not contain a single agent that performs the actions. We can also think of these problems as of multi-agent ones.

Next, we introduce the one-layer and multi-layer semantical representations for both of these problem domains. All described representations also follow the rules stated in (Urbanovská and Komenda, 2023) and before the semantically layered representation acts as an input to the slCSRN, we pad it with "walls" and add one layer for this padding.

2.1 Tetris Representation

The Tetris domain has an explicitly defined grid in its PDDL representation by stating a number of grid cells and their connections. The one-layer representation depends on the number of object types in the problem. In this case, we have three types of blocks. Therefore, the one-layer representation has one layer for every block type. That creates a quite compact input, but it can also cause problems when distinguishing the individual blocks of one type, as shown in (Urbanovská and Komenda, 022a).

The multi-layer representation has a number of layers equal to the number of individual blocks. The representation can get a lot larger compared to the one-layer one, but each block has its layer and can be identified without problems, which gives us more information about the individual blocks. Both representations are shown in Figure 2.

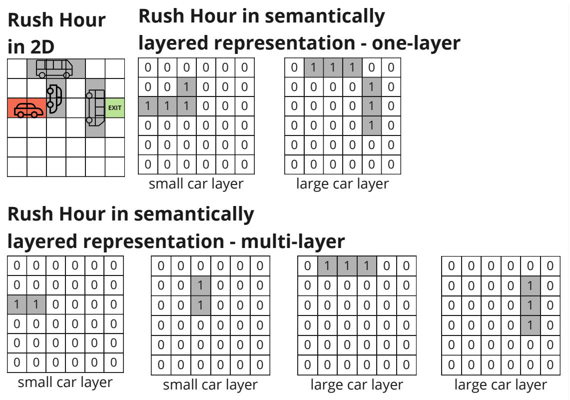

Figure 3: Semantically layered representation for a Rush Hour problem instance - both one-layer and multi-layer representation.

2.2 Rush Hour Representation

The Rush Hour puzzle is played on a 6×6 grid, however, the problems can be created on an arbitrarily sized grid with an arbitrary number of cars. The size of the grid is defined in the PDDL similarly to the Tetris grid. The number of object types in this case is only two. Therefore, in the one-layer representation, we have two layers, each for one type of car. In the layer with small cars (1×2), we also place the "red car" which is the most important one in the game since it is the car that has to exit the map. The next layer is for the large cars (1×3).

The multi-layer representation has one layer for every car. That has a clear advantage for the "red car" that is easier to identify in this representation. Both representations are shown in Figure 3.

3 TRAINING

We used the exact architecture as proposed in (Urbanovská and Komenda, 2023). That allowed us to train 4 versions of the slCSRN architecture

- slCSRN with one-layer representation
- slCSRN with multi-layer representation
- unfolded slCSRN with one-layer representation
- unfolded slCSRN with multi-layer representation

Differences between the representations are described in Section 2. The slCSRN uses weight sharing among its recurrent networks, the unfolded slCSRN uses one set of weights for one recurrent iteration.

Another parametrization of the training comes from the architecture itself, as we can set the number of recurrent iterations and hidden states of the architecture. We stuck to the previous works that used the combinations of parameters

- number of recurrent iterations = $[10, 20, 30]$
- number of hidden states = $[5, 15, 30]$

The loss function used for training is the monotonicity inducing loss as proposed in (Urbanovská and Komenda, 2021) that focuses rather on the monotonicity of the learned heuristic values rather than on the absolute values themselves. All the configurations were trained for 2000 epochs.

The training data for both domains were generated randomly. Tetris training data was of size 2×2 and 4×2 with $1-4$ blocks. The Rush-Hour training data was of size 3×3, 3×4, 4×3, and 4×4 with up $2-5$ small cars and $0-2$ large cars placed on the map.

As labels, we used the optimal solution lengths that we acquired using the Breadth-first Search algorithm.

For every slCSRN version, we selected the best configuration for the planning experiments. The selected configurations for both problem domains are shown in Table 1.

4 PLANNING EXPERIMENTS

The planning experiments are performed by using the selected trained slCSRN architectures as heuristic functions in a search algorithm. Similarly to the previous works, we are using the Greedy Best-first Search that uses solely the heuristic value to guide the search.

It is not our main goal to outperform any existing heuristics, we simply want to see the performance of the machine learning-based approaches to see the comparison and get more insight into the learned heuristic functions. We run the experiments of the blind heuristic, h^{max} (Bonet and Geffner, 2001), h^{add} (Bonet and Geffner, 2001) and h^{FF} (Hoffmann, 2001).

The blind heuristic serves as a baseline that gives us the equivalent of a random search, as it always returns 0 for any state. The other three heuristic functions are widely used in the field of classical planning. slCSRN and unfolded slCSRN configuration used in these experiments are shown in Table 1.

The planning experiments were performed on one dataset for each domain with 50 problem instances. Each problem instance had a time limit of 10 minutes. We used 3 metrics to compare the results. The most important one is coverage (cvg) which shows the percentage of solved problems from the given set. It is commonly used in classical planning. Other than that,

Table 1: Selected slCSRN and unfolded slCSRN architectures for each semantically layered representation of Tetris and Rush hour. These will be further used in the planning experiments.

		Tetris		Rush hour	
Architecture	Representation	Recurrent iterations	Hidden states	Recurrent iterations	Hidden states
slCSRN	one-layer	10	5	20	30
slCSRN	multi-layer	10	30	10	30
unfolded slCSRN	one-layer	20	30	10	5
unfolded slCSRN	multi-layer	10	5	10	30

Table 2: Planning experiments for the Tetris domain. All the best results are in bold lettering. The slCSRN architecture input representation is denoted ol for one-layer and ml for multi-layer. This representation is followed by architectures' parameters.

	Tetris		
	avg pl	avg ex	cvg
blind	4.55	4045.26	0.84
h^{max}	5.16	471.63	**0.98**
h^{add}	5.14	18.08	**0.98**
h^{FF}	5.73	36.57	**0.98**
slCSRN-ol-10-5	4.21	4.38	0.48
slCSRN-ml-10-30	8.10	19.29	0.62
unfolded slCSRN-ol-20-30	4.35	4.74	0.46
unfolded slCSRN-ml-10-5	8.30	11.5	0.6

Table 3: Planning experiments for the Rush Hour domain. All the best results are in bold lettering. The slCSRN architecture input representation is denoted ol for one-layer and ml for multi-layer. This representation is followed by architectures' parameters.

	Rush Hour		
	avg pl	avg ex	cvg
blind	73.81	5210.19	0.96
h^{max}	74.62	2765.02	0.96
h^{add}	77.21	3343.23	0.94
h^{FF}	76.61	2765.53	**0.98**
slCSRN-ol-20-30	84.3	3580.9	0.8
slCSRN-ml-10-30	81.49	4952.08	0.78
unfolded slCSRN-ol-10-5	86.68	5035.64	0.94
unfolded slCSRN-ml-10-30	83.95	4104.16	0.76

we also use the average plan length (*avg pl*) and the average number of expanded states (*avg ex*) to gain more information.

The dataset used in these experiments contains problem instances that were not present in the training data, and the grids are larger than the training samples for both the Tetris and the Rush Hour domain. The Tetris grids in this set are of sizes 4×4 and 6×4 with $3-6$ blocks. The Rush Hours problem instances are all on 6×6 grid and were taken from the database available at (Fogleman, 2023). We took the 50 puzzles with the longest solutions that contained no walls for the planning dataset.

The complete results can be seen in Table 2 for the Tetris domain and Table 3 for the Rush Hour domain.

4.1 Tetris Result Discussion

The results of the planning experiment in Table 2 show that the coverage metric is a lot lower than in the case of classical planning heuristics. This can be caused by multiple things, including computational time, therefore we decided to provide a further analysis of the planning results for the Tetris domain by comparing each learned heuristic with the classical planning heuristics separately only using the solved problem instances. That way, we can compare each trained slCSRN version fairly and see if there is any advantage in its performance in terms of the other metrics.

Table 4: More detailed analysis of the Tetris planning experiments results for each configuration's solved instances. Each section of the table relates to the corresponding slCSRN configuration and the best results are in bold lettering.

	Tetris	
	avg pl	avg ex
h^{max}	3.71	25.92
h^{add}	**3.58**	8.21
h^{FF}	4.08	13.67
slCSRN-ol-10-5	4.21	**4.38**
h^{max}	4.03	113.45
h^{add}	**4.0**	**12.68**
h^{FF}	4.52	13.58
slCSRN-ml-10-30	8.1	19.29
h^{max}	3.61	15.74
h^{add}	**3.52**	6.48
h^{FF}	4.0	9.96
unfolded slCSRN-ol-20-30	4.35	**4.74**
h^{max}	4.1	240.47
h^{add}	**4.03**	15.4
h^{FF}	4.63	25.0
unfolded slCSRN-ml-10-5	8.3	**11.5**

Even though the overall coverage does not seem very impressive for the learned heuristics for the Tetris domain, we can see that the individual learned heuristics are performing greatly in the instances they solved. In Table 4, we can see that three out of the four trained slCSRN versions dominate in the average number of expanded states.

That means that the learned heuristics were able to navigate the search algorithm more reliably than the classical planning heuristics. Since we are using a greedy search algorithm, the path length comparison does not carry a lot of importance since the search does not have a guarantee on the solutions' length.

4.2 Rush-Hour Result Discussion

Results for the Rush-Hour domain can be seen in Table 3. In this case, the coverage is much closer to the classical planning heuristics. We can see that the unfolded slCSRN that uses one-layer representation was able to achieve 94% coverage, which is on par with the classical planning heuristics. Even the architecture with the least coverage was able to solve 76% of the problems. Both average path length and average number of expanded states are between the blind search and the other classical planning heuristics. That implies that the learned heuristics are working better than no heuristic at all, but their informativeness does not seem to be up to the classical heuristic standards.

5 EXPLAINABILITY INSIGHTS

The explainability of Artificial Intelligence is an important and well-discussed topic. Many methods have been developed to support explainability on images (Haar et al., 2023) or for large language models (Zhao et al., 2023). However, there are not many approaches that would be usable on a recurrent message passing architecture such as slCSRN. Therefore, we wanted to approach the explanation a little more intuitively and provide some insights into the slCSRN computations using classical planning. We would also like to mention that this analysis does not fully explain the performance of the individual trained models in the planning experiments, but rather provides a deeper understanding of the heuristic computation by the slCSRN.

To keep the results compact, we decided to perform the explainability analysis with the trained networks chosen for the planning experiments for both problem domains.

We used the same strategy for both problem domains. We created a very simple toy problem instance with only one movable object that was solvable simply by moving it in one direction for a number of steps.

The toy problem for the Tetris domain is shown in Figure 4. The problem consisted of only one square block and a long, narrow path. The smallest instance's path length was one, the longest was 30. The final size 30 was taken from the maximum number of recurrent iterations that we used in the training of the models. For each of these created instances, we generated all possible states starting from the given instance and also saved the number of steps needed to solve each state.

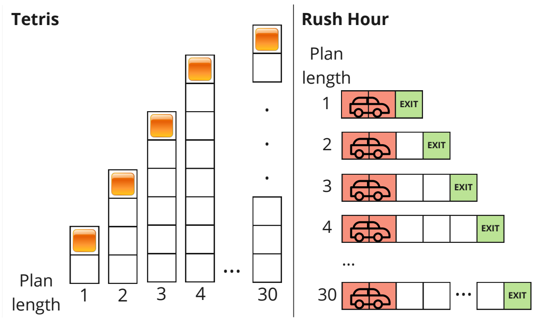

Figure 4: Toy data for Tetris (left side) and Rush Hour (right side) used for the explainability analysis. We can see the plan length near each problem instance.

For the Rush Hour domains, the dataset was created accordingly as shown in Figure 4. The smallest instance only requires one step to reach the goal, the largest instance requires 30 steps.

The next step is letting the slCSRN architecture compute a heuristic value for every single generated state in the dataset. This data can be shown in a plot for each sample to see the slCSRN behavior for every problem instance over all of its states.

We can see an example of this plot in Figure 5 where each line consists of heuristic values computed by the corresponding slCSRN for the Tetris domain. We can see that the architecture converged to a certain pattern in that has been satisfactory for the loss function during the training. This behaviour has been the same for all the architectures.

The same plots for the Rush Hour domain can be seen in Figure 7. Each line in the graph represents one problem instance, and each value along the way shows the heuristic estimate at the given number of steps away from the goal. In this case, we do not see the patterns that were obvious for the Tetris domain, but we can see that the variance of the generated values greatly differs across the models.

To learn more about the actual quality of the heuristic and not just the shape of the learned heuristic function, we can take the heuristic values for each problem instance and for every state on the way from the initial state to the goal and compare it to its neighboring values. As we already mentioned, the loss function we used is focused on monotonicity. Therefore, we can count how many heuristic values for each number of steps are correctly placed within the array of values. In Figure 6, we can see that the monotonicity over the different number of steps to the goal is not very high overall. We see that the majority of correctly generated values in terms of monotonicity appear very close to the goal. For the rest of the path, the architecture is rather uncertain about the goal distance estimate and does not place the heuristic values in the correct order very often.

In the case of the Rush Hour domain, we can see in Figure 8 that the number of correctly placed heuristic values slightly differs for each displayed architecture version. We can assume that high values mark particular distances, where the network makes a decision that is statistically important for the search.

The slCSRN-ml-10-30 was the weakest of these four architectures in terms of performance in the planning experiments. We can see the highest number of correctly generated values around 6–7 steps from the goal, but before and after the values are very low, which could cause the performance issues.

The unfolded slCSRN-ol-10-5 has quite a high number of correctly generated heuristic values up to 11 steps away from the goal. That implies that the navigation is a lot more reliable around the goal and once the number of steps needed to solve the problem goes higher, the network does not provide a very reliable estimate.

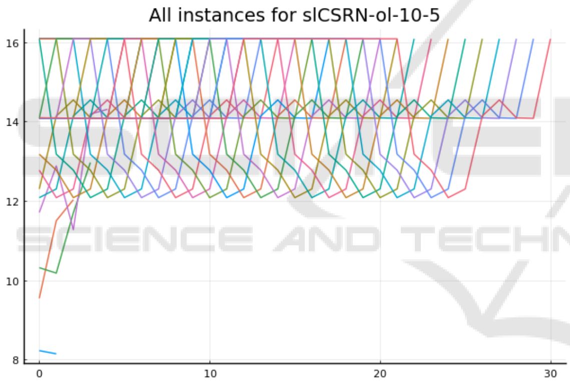

Figure 5: Heuristic values for the Tetris explainability dataset.

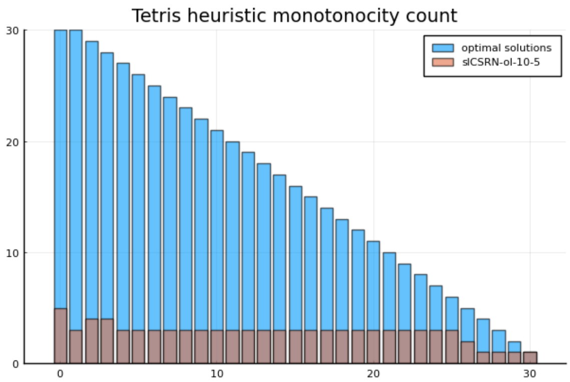

Figure 6: Monotonicity counts for the Tetris heuristic values. The blue values show numbers for the perfect monotonicity, the orange values are achieved by the corresponding slCSRN.

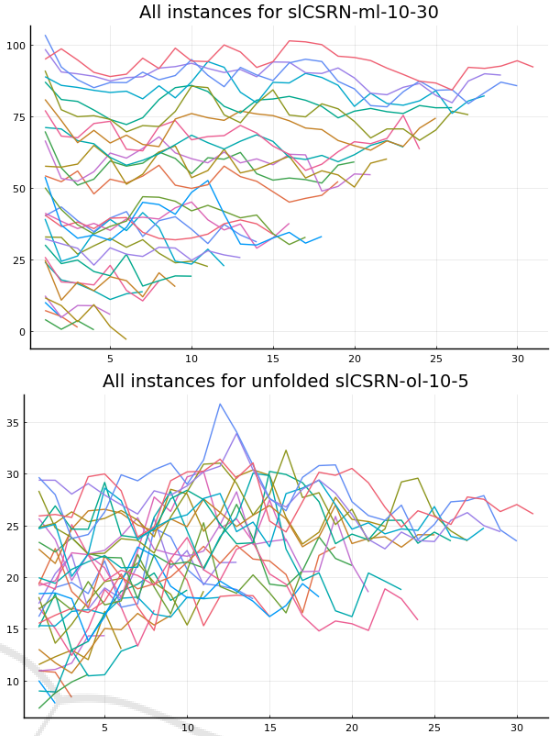

Figure 7: Heuristic values for the Rush Hour explainability dataset.

5.1 Explainability Discussion

Overall, we can see that the training had a different effect on each of the two domains. Let us discuss the possible outcomes of the proposed explainability analysis.

The architectures trained on the Tetris domain learned a repetitive pattern that occurs for instances of various plan lengths shown in Figure 5. This might imply that there really is a simple algorithm-like pattern generating heuristic values for this problem domain. On the other hand, this pattern can also be caused by any problem in the training dataset, just as an insufficient amount of training samples that cause overfitting.

The Rush Hour sample analysis shown in Figure 7 seems to have a better representation of the networks' computations, and the monotonicity counting analysis indicated that the number of steps estimable by the models is certainly limited. However, even the less informed estimates far away from the goal can be informative for the search algorithms. We can see that the estimates further from the goal are forming smoother curves, and therefore the values are less accurate, which is caused by the limited number of iterations of the slCSRN architecture.

Another insight based on the explainability analysis can be read as the ability of the learning process to

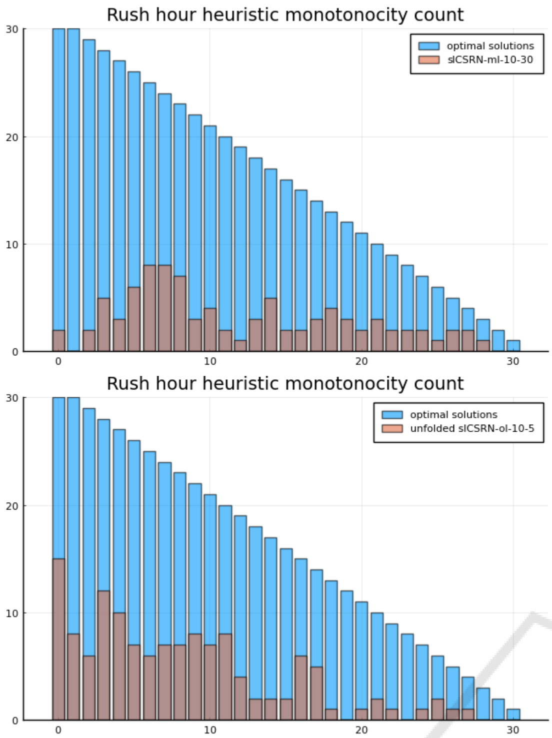

Figure 8: Monotonicity counts for the Rush Hour heuristic values. The blue values show numbers for the perfect monotonicity, the orange values are achieved by the corresponding slCSRN.

reflect the need for a more systematic or randomized approach to solve different planning problems. On the one hand, the Tetris problems need a more systematic approach, as the problem is not overly constrained. The Tetris pieces are in most cases loose and movable in all directions, only with the exception, of when they are getting stacked to the goal part of the grid. On the other hand, the cars in the Rush Hour are always highly constrained in movement, (a) only in two directions, (b) by other cars around. The more constrained problems are more easily solvable by more randomized strategies. Intuitively, if we ”randomly shuffle” the Rush Hour puzzle with a bias toward the goal, we will have a higher chance of solving the problem than if we randomly shuffle the Tetris domain, where we will only get most of the pieces into the empty part of the grid and jiggle them around their positions (that will happen even with a bias towards the goal area because some pieces have to wait for correct positioning of others).

The Table 2 exhibits low coverage results, as for the machine-learning-based heuristics it is more complicated to learn systematic heuristic evaluation, than randomization. However, as Table 3 shows, in the case of Rush Hour the randomization gets on par in the coverage of the solved problems with the classical systematic heuristics as h^{max}, h^{add}, h^{FF}. A similar pattern can be seen in the explainability analysis of the heuristic values in Figure 7, which exhibits clearly noisy behavior in Rush Hour, especially for the best performing network unfolded slCSRN-ol-10-5. In the case of Tetris, the neural network, however, behaves systematically, providing a repetitive pattern of the heuristic values estimation based on the distance from the goal.

6 CONCLUSION

We have successfully tested the set of available domains for heuristic learning by slCSRN architecture on two domains — Tetris and Rush Hour. Both of these domains show properties different from the domains previously used with slCSRN as they contain multiple movable objects that influence the available actions of many different objects in the problem.

We have trained both slCSRN and unfolded slCSRN architectures with both one-layer and multi-layer semantically layered representations for both of the domains. We also compared the performance of these trained models in a heuristic search to classical planning heuristic functions.

On top of that, we analyzed the behavior of the trained models and created an explainability technique that allows us to describe the behavior. We created an explainability dataset consisting of differently sized levels of a simple toy problem that allowed us to analyze how the networks behave regarding the plan length. With this dataset, we analyzed the shape of the learned heuristic functions and also the monotonicity of produced values. These insights gave us a better idea of what the models learn and how the learning differs for domains requiring more systematic or randomized heuristic functions.

In the future,

In the future, a potential direction could involve focusing on domains that are not explicitly defined on a grid and creating a universal translating algorithm for any PDDL problem into a semantically layered representation. Moreover, there is an interest in ongoing exploration of the explainability field, examining diverse methods that might contribute to a deeper understanding of these approaches.

ACKNOWLEDGEMENTS

The work of Michaela Urbanovská was supported by the European Union’s Horizon Europe Research and Innovation program under the grant agreement

TUPLES No 101070149 and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS22/168/OHK3/3T/13. The work of Antonín Komenda was supported by the Czech Science Foundation (grant no. 22-30043S).

REFERENCES

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*.

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*.

Asai, M. and Muise, C. (2020). Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2676–2682. ijcai.org.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33.

Flake, G. and Baum, E. (2002). Rush hour is pspace-complete, or "why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270:895–911.

Fogleman, M. (2023). Rush hour instance database.

Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., and Weld, D. (1998). Pddl - the planning domain definition language.

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*.

Haar, L. V., Elvira, T., and Ochoa, O. (2023). An analysis of explainability methods for convolutional neural networks. *Engineering Applications of Artificial Intelligence*, 117:105606.

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

Ilin, R., Kozma, R., and Werbos, P. J. (2008). Beyond feedforward models trained by backpropagation: A practical training tool for a more efficient universal approximator. *IEEE Transactions on Neural Networks*, 19(6):929–937.

Shen, W., Trevizan, F. W., and Thiébaux, S. (2020). Learning domain-independent planning heuristics with hypergraph networks. In Beck, J. C., Buffet, O., Hoffmann, J., Karpas, E., and Sohrabi, S., editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 574–584. AAAI Press.

Ståhlberg, S., Bonet, B., and Geffner, H. (2021). Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. *CoRR*, abs/2109.10129.

Ståhlberg, S., Bonet, B., and Geffner, H. (2022). Learning generalized policies without supervision using gnns. In Kern-Isberner, G., Lakemeyer, G., and Meyer, T., editors, *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022*.

Ståhlberg, S., Bonet, B., and Geffner, H. (2023). Learning general policies with policy gradient methods. In Marquis, P., Son, T. C., and Kern-Isberner, G., editors, *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, pages 647–657.

Toyer, S., Thiébaux, S., Trevizan, F. W., and Xie, L. (2020). Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68.

Urbanovská, M. and Komenda, A. (2021). Neural networks for model-free and scale-free automated planning. *Knowledge and Information Systems*, pages 1–36.

Urbanovská, M. and Komenda, A. (2022a). Grid representation in neural networks for automated planning. In Rocha, A. P., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 14th International Conference on Agents and Artificial Intelligence, ICAART 2022, Volume 3, Online Streaming, February 3-5, 2022*, pages 871–880. SCITEPRESS.

Urbanovska, M. and Komenda, A. (2023). Analysis of learning heuristic estimates for grid planning with cellular simultaneous recurrent networks. *SN Computer Science*, 4.

Urbanovská, M. and Komenda, A. (2023). Semantically layered representation for planning problems and its usage for heuristic computation using cellular simultaneous recurrent neural networks. In Rocha, A. P., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 15th International Conference on Agents and Artificial Intelligence, ICAART 2023, Volume 3, Lisbon, Portugal, February 22-24, 2023*, pages 493–500. SCITEPRESS.

Zhao, H., Chen, H., Yang, F., Liu, N., Deng, H., Cai, H., Wang, S., Yin, D., and Du, M. (2023). Explainability for large language models: A survey. *CoRR*, abs/2309.01029.