Muninn

Simon Ståhlberg[1], **Blai Bonet**[2], **Hector Geffner**[3,1]

[1]Linköping University, Linköping, Sweden
[2]Universitat Pompeu Fabra, Spain
[3]RWTH Aachen University, Germany
simon.stahlberg@liu.se, bonetblai@gmail.com, hector.geffner@ml.rwth-aachen.de

Abstract

Our planner, Muninn, uses a message-passing neural network model to learn and optimize value functions for a given domain. Muninn employs a two-step learning process to first obtain an optimal value function and then a suboptimal value function. Initially, the model learns an optimal value function that represents the fewest actions needed to reach a goal state. Subsequently, the value function is fine-tuned to generate suboptimal solutions when used with a greedy policy. The purpose of the second step is to tackle domains that are intractable in the optimal case but not in the suboptimal case. The final learned value function is used in two ways. First, it is employed in a "hill-climbing"-like search algorithm that can exploit the learned model if it encodes a weak (or strong) policy. Second, it serves as a heuristic function within an A* search algorithm, used as a fallback when the learned model does not encode any policy.

Overview

Muninn is a planner that uses a message-passing neural network architecture to learn a value function for a given planning domain. In our previous work (Ståhlberg, Bonet, and Geffner 2022a,b), we learned general policies by training value functions and employing a greedy policy based on them. However, although not formally proven, it is likely that our model's expressive power is limited by C_2 (two-variable first-order logic with counting quantifiers). If the planning domain necessitates features that cannot be captured by C_2, the greedy policy based on the learned value function will fail to generalize to larger instances. Moreover, there is a possibility that the learned value function cannot even solve the training instances. Consequently, the inclusion of search algorithms becomes crucial to compensate for this limitation in expressive power.

To learn a value function, Muninn follows a two-step learning process, dedicating an equal amount of time to each step. Initially, the model learns an optimal value function that represents the minimum number of actions required to reach a goal state. We learn this value function in a supervised fashion by expanding the entire reachable state space and computing the value for each state. However, the lack of expressive power is not the only scenario where policies based on learned value functions fail. In cases where finding optimal solutions is NP-hard but finding suboptimal solutions is tractable, it is preferable to learn a value function that yields suboptimal plans when used as a greedy policy. Thus, in the second step, Muninn fine-tunes the optimal value function to generate suboptimal solutions when used as a greedy policy. While it is possible to directly learn suboptimal policies, it generally takes longer to converge. Hence, the first step aims to speed up learning process.

The learned value function in Muninn is used in two ways, dedicating an equal amount of time to each way. Firstly, it is used in a "hill-climbing"-like search algorithm, where A* is used from the current state to identify another state where the value is significantly better than the current value. At this point, the search is reset to avoid maintaining a large open list. This approach tends to find solutions quickly but sometimes yields plans of lower quality. Secondly, the learned value function acts as a heuristic function within an A* search algorithm, which discovers solutions of high quality but at the cost of increased time and memory consumption. We note that we use our value function as a heuristic, but it is not admissible. In other words, even when used with A* algorithm, it does not guarantee optimal solutions.

For detailed information on learning an optimal value function through supervised learning, we kindly direct the reader to our ICAPS paper (Ståhlberg, Bonet, and Geffner 2022a), where our architecture was originally introduced. Additionally, we invite the reader to explore our KR paper (Ståhlberg, Bonet, and Geffner 2022b) for insights into how we learn suboptimal policies using value functions. This paper also delves into the trade-off between optimality and generality for NP-hard domains. In these papers, we refrain from incorporating the learned value functions into search algorithms. Our aim was to obtain crisp results, avoiding any potential masking of flaws in the learned models by the search algorithm. However, in a competitive environment, the goal is to maximize coverage; thus, using search algorithms is very useful. Furthermore, since we only need the state and the goal to evaluate the heuristic value for each state, we use a *lifted planner* (Ståhlberg 2023) to avoid grounding the action schemas as a preprocessing step.

Analysis

We take a closer look at the final results from the competition, and outline the configuration that we ended up using.

Environment

The learner and planner was run on a single core from an Intel Xeon Gold 6130 (no GPU) with 32 GB of memory for 24 hours and 30 minutes, respectively. We want to note that the amount of computation available is unlikely to be sufficient for learning a good value function using our method. This is because we cannot exploit parallelism with a GPU or multiple cores, which much of deep learning relies on.

Hyperparameters

In this competition, we used the following hyperparameters:

- **Embedding Size**: The chosen embedding size for each object was set at $k = 48$. The selection of this embedding size ensures a balanced trade-off between computational complexity and the richness of representation.
- **Optimal**: The amount of time allocated to learn an optimal value function was 11 hours.
- **Suboptimal**: The amount of time allocated to learn an suboptimal value function was 11.5 hours.
- **Dataset**: We expanded instances with up to 10 million states and up to 16 GB of memory for training and validation data. In other words, we reserved 16 GB of memory exclusively for the training process, to ensure it is not terminated due to memory constraints.

We want to note that some of these hyperparameters (and planner implementation) have been chosen with this particular environment in mind.

Learning

The logs made available are trimmed to save space, which means we cannot track the training and validation loss over time. This only lets us see if the final loss was near 0.0 when learning a suboptimal value function. But we cannot tell if: (1) the training settled at some loss above zero; or (2) we could not learn an optimal value function but managed to learn an suboptimal one. Also, we can only see the last model's validation loss, not the best one's. Keep in mind that both training and validation losses fluctate during training, so the end results might not show the whole picture. We interpret a big difference between validation and training loss that the learned model does not generalize well.

Table 1 shows our understanding of the logs. First of all, the organizers did a great job creating small instances that can be fully expanded and stored in memory. The domain with the least expanded instances in the training and validation set is Rovers with 16 instances, this should provide ample data to learn from. In some cases, the logs were cut short, so we could not see the exact number of expanded instances. The (in)equalities in the table show whether this information was trimmed. Based on our experience, the validation loss should be below 0.0001 for a model to function correctly as a policy.[1] However, it seems no model hit that mark. But since we use these models as part of heuristic search, we are a bit more flexible. We consider a model to have converged if the *training loss* is about 0.0005 and it is generalizing well if the *validation loss* is 0.005 or lower. However, this could be a very lenient view, and with more computing power, we should get even lower losses.

Planning

We used our learned models in two modes: as a *weak policy*, where we use it to greedily find partial plans to states that have a lower value than the starting state; and as a heuristic for A*.

Weak Policy. If the learned model encodes a strong policy for the domain, we can greedily follow its suggestions to solve the problem. In this scenario, each partial plan consists of a single action. However, to account for inadequate training or a lack of expressive power, we perform an A* search until we find a state with a lower value. This is repeated until a goal state is found.

A* Search. If the learned model cannot be used even as a weak policy, we can use it as a heuristic function for A* search by simply interpreting the learned value function as a heuristic function.

Coverage. Table 1 shows that the learned value functions can serve as both a weak policy (and possibly a strong policy). This is because using A* search did not make a big difference in results for any domain. Moreover, this approach was needed to solve certain instances within the resource limits. Specifically, the weak policy managed to solve 225 instances, while A* only solved 185. Out of all the instances, there were just 3 that A* solved which the weak policy could not. The downside to the weak policy is that it might give solutions of lower quality. Even though A* could potentially offer higher quality solutions, in the experiments, this did not happen very often.

Discussion

Drawing conclusions from the experimental results is difficult because the models were trained on just one CPU core. If more computing power had been available, the models might have been significantly more informative. This could have led to a more focused search with A*. Moreover, testing the model on larger instances without a GPU is often slow. For instance, for the Sokoban instance "easy-p13," we found a plan 25 steps long (with a quality of 0.92) by expanding just 117 states. However, it took 8.3 seconds to evaluate the 393 generated states. So, for this instance, the learned model was informative, but it was time-consuming to evaluate because it did not run on a GPU. That said, even if the learned models were perfect, they might also simply be too expensive to use for larger instances.

[1] This low loss is due to the way the loss function is set up: it uses an inequality.

Domain	# Exp.	Converged	Generalized	Weak Policy Coverage	A* Coverage	Total Coverage
Blocks	$= 30$	✓	✓	39	24	39
Childsnack	$= 27$	×	×	11	9	11
Ferry	≥ 32	×	×	42	32	42
Floortile	$= 17$	×	×	0	0	0
Miconic	≥ 24	×	×	30	30	30
Rover	$= 16$	✓	×	15	9	15
Satellite	$= 21$	✓	×	15	16	16
Sokoban	≥ 32	×	×	24	26	26
Spanner	≥ 25	✓	✓	32	30	32
Transport	$= 20$	×	×	17	13	17
Total	≥ 244	-	-	225	189	228

Table 1: The column "# Exp." indicates the number of instances expanded during learning. The "Converged" and "Generalized" columns show if the training appeared to have settled and if the model seemed likely to generalize well, based on validation loss. The "Weak Policy Coverage" and "A* Coverage" columns show how many instances were solved using the learned model in each method, while "Total Coverage" indicates the overall number of instances solved.

Acknowledgements

Big thanks to Jendrik Seipp and Javier Segovia-Aguas for organizing the IPC learning track. Your hard work made the event a success, and we truly appreciate your dedication to the planning community. Great job, and thank you!

References

Ståhlberg, S. 2023. Lifted Successor Generation by Maximum Clique Enumeration. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022*.