HUZAR preprocessor

Piotr Rafał Gzubicki[1], **Bartosz Piotr Lachowicz**[1], **Alvaro Torralba**[1]
1. Aalborg university
Department of Computer Science
Denamrk
piotr.gzubicki97@gmail.com, jendrekpb@gmail.com, alto@cs.aau.dk

1 Introduction

This document describes the implementation of the HUZAR planner and the techniques used to create it. The presented work is an outcome of the project undertaken by our study group with the guidance of the supervisor during the 10th semester of the Computer Science program at Aalborg University. The primary objective of this project was to develop an advanced tool capable of optimizing automated planning processes. To achieve this, we leveraged the cutting-edge technology of Graph Neural Networks.

Building upon a prototype that was developed in the previous semester, our development process aimed to create a comprehensive tool encompassing hyperparameter tuning and feature engineering. The ultimate goal was to produce a fully functional tool that could significantly reduce the search state space for automated planners while retaining the essential information required for efficient plan computation.

2 Background

The developed solution utilizes the graph nature of the planning problems. Furthermore, by encapsulating the instance into the appropriate structure it is fed into the Graph Neural Network which performs the classification of operators with the attempt to correctly remove sparse actions.

2.1 Graph Neural Networks

Graph Neural Networks are a class of neural networks designed for data that is represented in graph form. These networks leverage the connections between nodes, utilizing the edges to propagate information between connected nodes. Graph Neural Networks have found diverse applications, including the modelling of social networks (Wu et al., 2020), physical systems in natural science (Sanchez-Gonzalez et al., 2018), and protein-protein interface networks (Fout et al., 2017).

Bronstein, Bruna, Cohen, and Veličković provide a comprehensive analysis of Graph Neural Networks (GNNs) (Bronstein et al., 2021) in which they identify that all modern GNN architectures stem from one of three types: Convolutional, Attentional, and Message Passing. The relationship between these types is such that Convolutional GNNs are a subset of Attentional GNNs, which in turn are a subset of Message Passing GNNs. Since the Message Passing GNN is the most general, we will describe its logic further.

The Message Passing GNN consists of two functions: an aggregate function and an update function. To compute a hidden state h_u for a node u, the GNN first aggregates all feature vectors x_v for all neighbours $v \in N_u$ where N_u is the set of all nodes that have a direct edge to node u. Then, the GNN combines the node features x_u with the aggregated information from the neighbours and updates the hidden representation h_u as follows:

$$h_u^{t+1} = \phi\left(h_u^t, \bigoplus_{v \in N_u} \psi(h_v^t)\right) \tag{1}$$

The permutation invariant $\bigoplus$ controls the aggregation and accepts an arbitrary number of inputs (one per neighbour $v \in N_u$). The permutation invariant could be a sum or a pooling mechanism such as max, mean, or min similar to conventional convolutional networks. The trainable function $\psi(x_u, x_v)$ transforms the features of the node u and the feature vectors coming from each neighbouring node that are passed to the permutation invariant. Finally, to update the hidden representation h_u, the feature vector of node u is combined with the aggregated information z through a trainable non-linear function ϕ (e.g., ReLU).

In the context of Graph Neural Networks, the number of message-passing functions defined in the network architecture determines the depth of the network. Suppose we aim to construct a computational graph for a Graph Convolution Network (GCN), which is a specialized variant of a Message Passing GNN. In GCN, the function ψ is replaced with trainable weight matrix W, which is utilized to perform a matrix multiplication operation on the feature vectors of all neighbouring nodes for a given node, resulting in the final following formula.

$$\psi(h_v^t) = W * h_v^t \tag{2}$$

$$h_u^{t+1} = \phi\left(h_u^t, \bigoplus_{v \in N_u} W * h_v^t\right) \tag{3}$$

3 Mapping problems onto garphs

In order to fully comprehend the solution first the basic concepts of graphs and problem mappings must be explored. A *Graph* G can be defined as a collection of nodes V and edges E. Each node $v \in V$ is characterized by its specific features, denoted as x_v. In a graph with n nodes, the relationships between nodes are represented by a binary adjacency matrix $A = V \times V$, where a value of 1 in the $A(i, j)$ entry indicates that nodes i and j are connected by an edge. Additionally, two nodes v_1 and v_2 are considered neighbors if $A(v_1, v_2) = 1$.

Various kinds of graphs are available to represent planning tasks. However, through experimentation, the Problem Description Graph (PDG) (Pochter et al., 2011) was found to be the most promising option. Because of certain practical nuances, the PDG was slightly modified and does not correspond exactly to what was proposed in (Pochter et al., 2011) however the main structure of the graph is the same.

A Problem Description Graph is a graph representation of a planning problem exploiting the relationship between actions and predicates. To create the *Problem Description Graph*, the relationships between operators and predicates are leveraged. Furthermore, the PDG is considered a heterogeneous graph due to its inclusion of multiple types of nodes and links. Specifically, four distinct node types are identified in the graph:

- operator nodes
- value nodes
- variable nodes

Four distinct types of edges can be observed in the graph, each of which reflects the relationships between objects in their grounded representation. These edges are mapped to the PDG in the following manner:

- **Value-Variable edge**: A Value and Variable have an edge if the Value belongs to the Variable, the edge is not directed. The variable can have multiple values but a value node can only be connected to one variable node.
- **Value-Operator (Precondition)** A precondition Value and an Operator are related if the Value is in the preconditions of the action from which the operator was derived. This edge is pointing from value towards the operator
- **Operator-Value (Effect)** An effect Value and Operator are related if the Value is in the effects of the action from which the operator was derived. For this type of connection the operator node point towards the value node.

Value nodes are connected to one or more operator nodes (by either precondition or effect edge), if said values appear in either the precondition or effect of the action definition associated with the operator. This relationship can be visualized in the figure below.

4 Node Classification

It is a classification problem of nodes inside the graph. In the case of this project, the PDG is a heterogeneous type of graph. Consequently, only a subset of nodes is targeted with classification tasks, namely the operators. The others: *Variables* and *Values* will not take part in the classification task. The PDG will be fed to the GNN to build a hidden representation of the nodes. Operator nodes afterwards are going to go through a Sigmoid function in order to get the probability of an operator (action) being good or bad.

5 Hyperparameters optimization

We employed an automated SMAC tool in our preprocessor to achieve hyperparameter optimisation.

SMAC, known as sequential model-based algorithm configuration (Lindauer et al., 2022), is a versatile tool that optimizes algorithm parameters. It can be applied to various scenarios, including automating processes, evaluating functions such as simulations, and adjusting parameters accordingly. The metrics used for optimization are not the model performance indicators such as test set loss but the performance of the planner which uses the trained preprocessor to compute plans. Therefore, the configurations which perform the best in terms of coverage are selected. Furthermore, they are analyzed based on the number of operators passed to the planner - the actual performance of the GNN classifier.

Optimizing hyperparameters, and selecting suitable configurations of the model and preprocessor are essential for achieving accurate and effective performance in planning domains. By considering these factors and leveraging planning metrics, the GNN model can be tuned to enhance the planner's performance and deliver improved results in real-world planning tasks. Consequently, all the parameters will be incorporated into the SMAC optimization tool for monitoring and analysis within the model's specific planning domain. The insights gained from these reflections may contribute to further optimization improvements and a better understanding of best practices.

6 Conclusion

Throughout the project, we conducted extensive research, implemented innovative methodologies, and performed rigorous testing to ensure the effectiveness and reliability of our solution. The final outcome is a complete tool that successfully optimizes automated planning processes, contributing to the advancement of this field.

We believe that our work provides valuable insights and contributes to the growing body of knowledge in the area of automated planning optimization. By developing this tool, we aim to facilitate more efficient planning processes, opening up possibilities for improved problem-solving in various domains.

References

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein interface prediction using graph convolutional networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9.

Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. 2011. Exploiting problem symmetries in state-based planners. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. 2018. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4470–4479.

Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. 2020. Graph convolutional networks with markov random field reasoning for social spammer detection. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1054–1061. AAAI Press.