Learning Heuristics with Graph Neural Networks for Numeric Planning: A Preliminary Study

Valerio Borelli[1,2,*], Alfonso Emilio Gerevini[1], Enrico Scala[1] and Ivan Serina[1]

[1]*University of Brescia, Italy*
[2]*Sapienza University of Rome, Italy*

Abstract
We consider the problem of learning heuristics for numeric planning domains, using Graph Neural Networks. The problem has been approached multiple times, from different perspectives and with varying results for classical planning, but is relatively new for numeric planning. The goal is to extend the work proposed by Ståberg, Bonet, and Geffner [1] to handle numeric planning problems.

Keywords
Numeric planning, Graph Neural Networks, Heuristic search

1. Introduction

In recent years different works have shown how deep learning approaches can be used to solve planning problems, either as heuristic functions for classical planning, as value functions for general policies, or even as a separate system that can find plans with little to no external help from a planner or an agent. Interesting results for classical planning problems have been achieved with standalone systems, such as PlanGPT [2], which uses a GPT-based model to find a plan given a domain and a problem; with general policies approaches [1, 3, 4] that uses different types of graph neural networks architecture to learn a generalized policy for a classical planning domain; with heuristic approaches, such as hypergraph networks [5], or relational heuristic networks [6].
The first extension that tries to tackle numeric planning problems has been numeric ASNets [7], it builds on the architecture described in [4], adding the possibility to reason about numeric fluents.
In this work, we describe the main issues that need to be addressed, in order to handle a numeric planning problem, we propose to build upon the architecture described by tålberg, Bonet and Geffner [1]. Even though the scope of the architecture is to learn value functions for general policies, it can be easily used to learn heuristic values, as has already been done in [8].
The paper is organized as follows: first, we cover the background (numeric planning and GNNs). Then we describe the architecture that we want to extend. Finally, we explain the main issues that must be addressed to handle numeric planning problems.

2. Numeric Planning

A numeric planning problem is a tuple $\Pi = \langle s_0, A, G, V \rangle$ where V is the set of variables, which can be either propositional or numeric, $s0$ is the initial state of the problem, A is a set of actions, either numeric or propositional, and G is the goal of the problem, which consists in a set of propositional and numeric goal conditions.
An action $a \in A$ is a pair $\langle Pre(a), Eff(a) \rangle$, in which $Pre(a)$ represents the set of preconditions that must

Doctoral Consortium at the 23rd International Conference of the Italian Association for Artificial Intelligence Bolzano, Italy, November 25-28, 2024.
[*]Corresponding author.
valerio.borelli@unibs.it (V. Borelli); alfonso.gerevini@unibs.it (A. E. Gerevini); enrico.scala@unibs.it (E. Scala); ivan.serina@unibs.it (I. Serina)
0009-0007-3833-4654 (V. Borelli); 0000-0001-9008-6386 (A. E. Gerevini); 0000-0003-2274-875X (E. Scala); 0000-0002-7785-9492 (I. Serina)

be verified in a state s before executing the action, and $Eff(a)$ represents the set of effects of the action, that will be applied to the actual state to reach the next state.
A plan $\pi = \langle a_1, ..., a_n \rangle$ is a sequence of actions, and is considered a solution for the planning problem if applying the sequence of actions, starting from the initial state of the problem, leads to a final state in which all the goals of the problem are satisfied.

3. Graph Neural Networks

A Graph Neural Network (GNN) is a type of neural network specifically designed to work with graph-structured data. Traditional neural networks are designed to represent data structured in grids or sequences, but they struggle in real-world datasets structured as graphs, like social networks or molecular structures.
A Graph Neural Network takes as input a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of node features $X \in \mathbb{R}^{d \times |V|}$, where d represents the number of features for a node, and generates node embeddings $z_u, \forall u \in \mathcal{V}$. These node embeddings can then be used to predict nodes, edges or even the entire graph.
The nodes in the GNN exchange information during a message-passing iteration, with an iteration consisting of three operations:

- Message, in which messages are sent from each node towards its neighbors,
- Aggregation, in which all messages sent to a node are aggregated together in a unique message,
- Update, in which the node representation is updated using the aggregated message.

Each node u in the network has a hidden embedding $h_u^{(k)}$, that is updated according to information aggregated from the neighborhood of u after each iteration.
The update of the embedding of a node u can be written as:

$$m_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}^{(k)} \left(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\} \right) \tag{1}$$

$$h_u^{(k+1)} = \text{UPDATE}^{(k)} \left((h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)} \right), \tag{2}$$

where UPDATE and AGGREGATE are arbitrary differentiable functions, and $m_{\mathcal{N}(u)}$ is the aggregation of the messages that flows from u's neighborhood towards u [9].
At each iteration k of the GNN the AGGREGATE function takes as input the set of the nodes embedding in the neighborhood for each node and computes an aggregated message $m_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}^{(k)}(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\}, \forall u \in \mathcal{V}$. The UPDATE function combines the previous node embedding with the message generated by the AGGREGATE function for each node, to generate a new embedding $h_u^{(k+1)} = \text{UPDATE}^{(k)} \left((h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)} \right), \forall u \in \mathcal{V}$.
After K iterations of the GNN, the output of the final layer defines the embeddings for each node:

$$z_u = h_u^{(K)}, \forall u \in \mathcal{V}. \tag{3}$$

3.1. Node and Graph classification

GNNs are used for three main tasks: node classification, graph classification, and relation (or link) prediction.
Node classification is the most common, in which the network is trained to classify new nodes between a one-hot vector indicating the possible class of nodes.
Similar to node classification, the main difference in graph classification is that instead of classifying nodes, the GNN is trained to classify the whole graph or part of it. To do that the node embeddings of the final layer (eq. 3) are used as input in a READOUT function, that summarizes the node embeddings and classifies the graph. Lately, this has also been extended with success to graph regression, in these instances instead of a softmax classification loss function, a squared-error loss function is usually used. In order to learn either a value function for general policies or a heuristic for planning, the task of the GNN is graph regression.

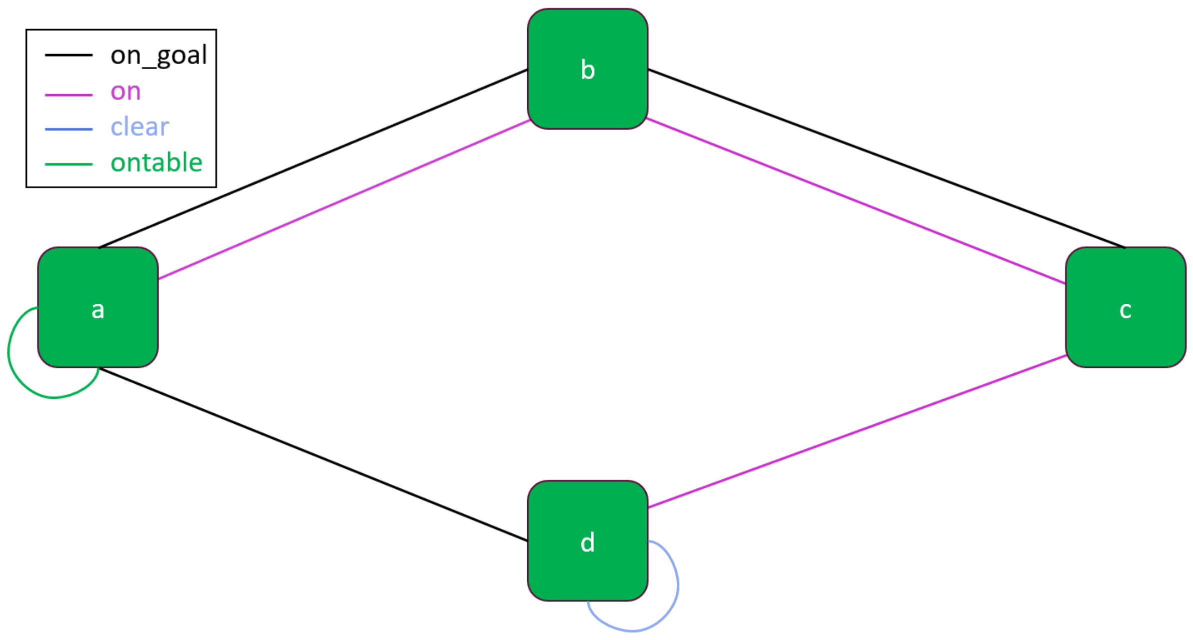

Figure 1: Blocksworld GNN architecture example, for representation purposes we show the example without the q atoms

4. Graph Neural Networks for Classical Planning

The GNN architecture for learning heuristic values follows the one used by Stålberg, Bonet and Geffner [1, 3] for learning value functions, in which, instead of a graph, our networks represent relational structures.
This is because a state *s* in planning doesn't represent a graph, but instead a relational structure that occurs between the object of the problem. The relational structure for a state *s* is defined by the set of objects, the set of domain predicates and the atoms $p(o_1, ..., o_m)$ that are true in *s*. In particular, the set of domain predicates is fixed for a specific domain, the set of objects may change from one instance to another, and the value of the atoms is specific for the state *s*. Those considerations lead to a structure in which the objects represent the node of our network, the predicates are possible types of arches, and the atoms are the adjacency matrix: if $p(o_1, o_2)$ is true in a state *s*, the relational structure for *s* will have an arch of type *p* between o_1 and o_2. Technically, instead of messages flowing directly from objects to objects, they flow to the objects towards the atoms q in which they are involved, and from q to the objects involved in q.

The types of arches in the relational structure for a particular planning domain are given by the lifted predicates of the domain and the goal predicates. The GNNs map planning states *s* into real values *V(s)*.
Practical example with blocksworld:

- Predicates: on(2), handempty(0), ontable(1),clear(1),holding(1)
- Goal predicates: on_goal(2), handempty_goal(0), ontable_goal(1),clear_goal(1),holding_goal(1)
- Objects: block a, b, c, d
- Actions: pickup(?b), putdown(?b), stack(?b1,?b2), unstack(?b1,?b2)
- Current state: (on-table a) (on b a) (on c b) (on d c) (clear d) (handempty)
- Goal: (on b a) (on c b) (on a d)

As we can see in the example in Fig.1, the nodes of the GNN are the object of the planning problem, but instead of having a graph in which messages flow from an object towards its neighbors, we have a relational structure in which messages flows from objects o_i to the true atoms q in s that include o_i, $q = p(o_1, ..., o_m)$, $1 \leq$ i $\leq$m, and from such atoms q to all the objects o_j involved in q (Fig.2).

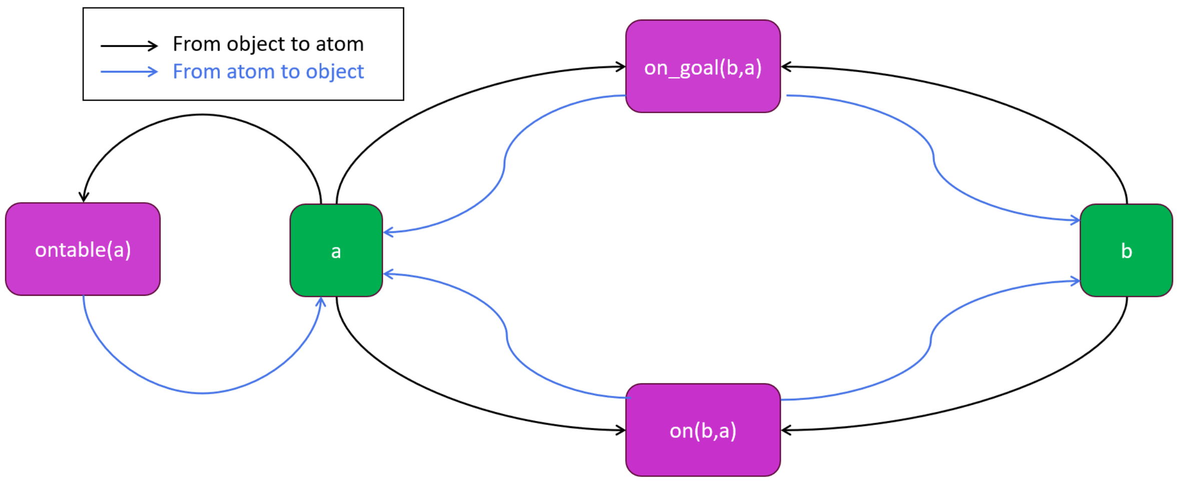

Figure 2: Blocksworld GNN architecture example, the actual representation of the arches between block a and block b, with the atoms involved

The aggregation functions used mimic h^{max} and h^{add}, and the same aggregation function is applied to all the nodes in the GNN (i.e. to all the objects in the problem) in every layer. The same update function is then applied for every node and in every layer, combining the previous embedding of the node with the array given as output by the aggregation function for that node.

5. Extension to Numeric Planning

In classical planning in order to univocally describe a state for a particular problem, knowing the true values of the predicates in the current state and the predicates that must be true in the goal is enough. While in classical planning action preconditions, action effects and goals can be described only with the truth values of predicates, in numeric planning instead action preconditions and goals might be described as numeric conditions, and action effects as numeric assignments. For example, a goal could be $x(object1) + 3 < x(object2)$.
Therefore, to extend the previous architecture to numeric planning, different issues must be addressed:

- Handling numeric fluents is necessary to achieve an input state for the GNN that can represent a state in our numeric planning problem univocally,
- Numeric goals must have a different representation since they can't be directly represented by the numeric fluents of the problem,
- Action preconditions might also be taken into account, which wasn't useful before since they were truth values of the boolean predicates, but now they represent different numeric conditions and henceforth they might be useful information to exploit in our architecture.

Our goal is to extend the existing architecture to solve these issues, and then train our models with a dataset generated using ENHSP [10] as a teacher search, with optimal plans when possible given the domain, and with the best combinations of search algorithm and heuristic when finding the optimal solution is not feasible.

6. Preliminary results

We are working on different approaches to handle numeric fluents and distances. While we still have to test the new architectures that we created with a planner, we already have some interesting results regarding the models obtained; in table.1 we show the minimum squared error obtained after training

Table 1
Minimum squared error obtained after the training of the model

Domain	Aggregate function	MSE
fo_counters	add	0.16
block_grouping	max	0.2

the models, using respectively the aggregation function based on h^{add} for first-order counters, and h^{max} for block grouping.

If we consider that suboptimal plans can have hundreds of actions in those domains, especially in first-order counters, an MSE of 0.16 is a good result.
From here, the next step will be testing the models inside a planner, to compare them to traditional numeric heuristics.

References

[1] S. Ståhlberg, B. Bonet, H. Geffner, Learning generalized policies without supervision using gnns, arXiv preprint arXiv:2205.06002 (2022).

[2] N. Rossetti, M. Tummolo, A. E. Gerevini, L. Putelli, I. Serina, M. Chiari, M. Olivato, Learning general policies for planning through gpt models, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 34, 2024, pp. 500–508.

[3] S. Ståhlberg, B. Bonet, H. Geffner, Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 32, 2022, pp. 629–637.

[4] S. Toyer, F. Trevizan, S. Thiébaux, L. Xie, Action schema networks: Generalised policies with deep learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[5] W. Shen, F. Trevizan, S. Thiébaux, Learning domain-independent planning heuristics with hypergraph networks, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 30, 2020, pp. 574–584.

[6] R. Karia, S. Srivastava, Learning generalized relational heuristic networks for model-agnostic planning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021, pp. 8064–8073.

[7] R. X. Wang, S. Thiébaux, Learning generalised policies for numeric planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 34, 2024, pp. 633–642.

[8] S. Ståhlberg, B. Bonet, H. Geffner, Muninn, Tenth International Planning Competition (IPC-10) Learning Track: Planner Abstracts (2023).

[9] W. L. Hamilton, Graph representation learning, Morgan & Claypool Publishers, 2020.

[10] E. Scala, P. Haslum, S. Thiébaux, M. Ramirez, Interval-based relaxation for general numeric planning, in: ECAI 2016, IOS Press, 2016, pp. 655–663.