# Learning Heuristic Functions with Graph Neural Networks for Numeric Planning

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

[1] University of Brescia
[2] La Sapienza University of Rome
valerio.borelli@unibs.it, valerio.borelli@uniroma1.it, alfonso.gerevini@unibs.it, enrico.scala@unibs.it, ivan.serina@unibs.it

## Abstract

In this paper, we investigate the application of heuristics based on Graph Neural Networks (GNNs) to lifted numeric planning problems, an area that has been relatively unexplored. Building upon the GNN approach for learning general policies proposed by Ståhlberg, Bonet, and Geffner (2022b), we extend the architecture to make it sensitive to the numeric components inherent in the planning problems we address. We achieve this by observing that, although the state space of a numeric planning problem is infinite, the finite subgoal structure of the problem can be incorporated into the architecture, enabling the construction of a finite structure. Instead of learning general policies, we train our models to serve as heuristics within a best-first search algorithm. We explore various configurations of this architecture and demonstrate that the resulting heuristics are highly informative and, in certain domains, offer a better trade-off between guidance and computational cost compared to state-of-the-art heuristics.

**Code** — https://github.com/AfkValerik/LeapNP

## Introduction

Several recent works have shown how deep learning can be used to solve planning problems, either as heuristic functions for classical planning, or as value functions for general policies. For classical planning, interesting results have been achieved by different approaches: PlanGPT (Rossetti et al. 2024), which uses a GPT-based model to find a plan given a domain and a problem; general policies approaches (Ståhlberg, Bonet, and Geffner 2022a,b; Toyer et al. 2018), which use different types of graph neural network architectures to learn a generalized policy for a classical planning domain; and by heuristic approaches such as hypergraph networks (Shen, Trevizan, and Thiébaux 2020) and relational heuristic networks (Karia and Srivastava 2021).

Graph Neural Networks (GNNs) demonstrated significant promise in learning heuristic functions and general policies for classical planning problems. The work of Ståhlberg, Bonet, and Geffner (2022a) extends GNNs for learning value functions in classical planning scenarios, showcasing their capability to exploit relational structures inherent in planning states. In this context, the work of Horčík and Šír (2024); Horčik et al. (2025) provides an analysis of GNN-based heuristics, investigating how different state encodings affect both the expressiveness of the learned models and their effectiveness during search, thus highlighting the trade-offs between representational power and computational efficiency. For numeric planning, the ASNets architecture (Wang and Thiébaux 2024) is a recent approach which builds on the work by Toyer et al. (2018); this work enables reasoning about numeric fluents and their interactions. The introduction of specialized network modules and a dynamic exploration algorithm represents a significant advancement in training efficiency and policy effectiveness for numeric domains. It provides a neural architecture capable of reasoning about numeric variables and their interactions, bridging a critical gap in numeric planning tasks. ASNets provides a neural architecture trained as a general policy rather than a heuristic function, aiming to solve planning problems directly through action selection rather than by guiding search. A second recent approach presented by Chen and Thiébaux (2024) builds on the architecture introduced by Chen, Trevizan, and Thiébaux (2024), proposing a framework that uses novel graph kernels and optimization techniques tailored for numeric planning, addressing the challenges of numeric fluents and mixed attributes. Their method demonstrates superior efficiency and generalizability compared to traditional GNN approaches, and shows competitive performance against domain-independent numeric planners. This method highlights the potential of using graph learning for numeric planning, while ensuring data efficiency and interoperability.

In this work, starting from the work of Ståhlberg, Bonet, and Geffner (2022b) on GNNs for classical planning, we investigate the main issues that need to be addressed to exploit GNN models for solving numeric planning problems. As observed by Ståhlberg, Bonet, and Geffner (2023), GNN models can be used for learning value functions or heuristic values; our focus is on the latter. We design an architecture that encodes numeric planning states into relational graphs, capturing both boolean and numeric structure. We identify the representational and architectural challenges introduced by numeric conditions, and propose techniques to integrate numeric information into the message-passing framework of GNNs. We evaluate the resulting heuristics across several numeric planning domains, comparing them to both non-

numeric GNN baselines and state-of-the-art learning-based heuristics for numeric planning. Our experiments show that the proposed approach provides more informative guidance in many domains, leading to improved coverage and reduced search effort.

## Background

**Numeric Planning** We first informally introduce the lifted numeric planning problem, and then detail its semantics. This corresponds to PDDL2.1 level 2; for further details see (Fox and Long 2003; Haslum et al. 2019).

A lifted numeric planning problem is a pair $\Pi = \langle D, I \rangle$. The domain $D$ is a pair $\langle \mathcal{V}, \mathcal{A} \rangle$ where $\mathcal{V} = \langle \mathcal{P}, \mathcal{X} \rangle$ consists of a set $\mathcal{P}$ of boolean fluents (predicates), and a set $\mathcal{X}$ of numeric fluents, and $\mathcal{A}$ is the set of action schemas. Boolean and numeric fluents are defined in terms of their name and a list of variables. An action schema $a \in \mathcal{A}$ is a tuple $\langle Par(a), Pre(a), Eff(a) \rangle$, with $Par(a)$ being a list of parameters of the action schema (lifted variables); $Pre(a)$ is a conjunction over $\mathcal{V}$ that contains both propositional literals, such as $p(?x)$, or numeric conditions, such as $\xi \geq 0$, where $\xi$ is a numeric expression over $\mathcal{X}$. $Eff(a)$ are assignments over $\mathcal{P}$, such as $(p(?x) := \top)$, or over $\mathcal{X}$, such as $(f(?x) := \xi)$ with $\xi$ being a numeric expression over $\mathcal{X}$.

Actions, boolean fluents, numeric fluents, and numeric conditions can be grounded against a set of objects $O$. Intuitively, the grounding consists of generating as many actions as there are valid substitutions for the parameters and propagating the mapping over preconditions and effects, similarly, for boolean fluents, numeric fluents, and numeric conditions with their variables. A grounded numeric planning problem is the resulting formulation. A state for a grounded numeric planning problem is a full assignment to all grounded boolean and numeric fluents. Full semantics and grounding process can be found at Fox and Long (2003). A planning instance $I$ is a tuple $\langle O, s_0, G \rangle$ where $O$ is a set of objects; $s_0$ is a state, called the initial state. $G$ is the goal of the problem, it is a set of grounded boolean and numeric conditions over $O$. The semantics of a lifted numeric planning problem can be defined in terms of its grounded representation $\langle s_0, A, G, V \rangle$, where all variables have been replaced by objects in $O$. We say that a grounded action $a \in A$ is applicable in a state $s$ iff its preconditions $Pre(a)$ are true in $s$. Its execution changes $s$ according to its assignments $Eff(a)$. A plan $\pi = \langle a_1, ..., a_n \rangle$ is a finite sequence of grounded actions, and it is considered a solution for the planning problem iff all grounded actions it consists of are iteratively applicable in the state just before their execution, and $G$ is true in the last state. Throughout the paper, we will use the term *atom* to uniformly refer to both grounded boolean conditions (e.g., $p(o_1)$) and grounded numeric conditions (e.g., $x(o_1) + y(o_2) \leq 5$). By definition, every boolean fluent can be seen as a boolean condition, i.e., a propositional statement that may hold or not in a given state. This unified terminology allows us to treat boolean and numeric components consistently within the graph-based architecture used by the GNN.

**Graph Neural Networks (GNNs)** A GNN is a type of neural network designed to work with graph-structured data, producing vector embeddings for nodes or entire graphs. Following the notation of Hamilton (2020), a GNN updates each node representation through message passing. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node features $X \in \mathbb{R}^{d \times |\mathcal{V}|}$, where $d$ represents the number of features for a node, the embedding of node $u \in \mathcal{V}$ at layer $l$ is updated by aggregating information from its neighbors $\mathcal{N}(u)$:

$$m_{\mathcal{N}(u)}^{(l)} = \text{AGGREGATE}^{(l)} \left( \left\{ h_v^{(l)} \mid v \in \mathcal{N}(u) \right\} \right) \quad (1)$$

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

After L layers, the final node embeddings $z_u = h_u^{(L)}, \forall u \in \mathcal{V}$ can be used for node, edge or graph level prediction, typically via a READOUT function in the case of graph-level tasks.

In our setting, the input graph encodes a planning state, and the GNN performs graph-level regression: a READOUT over the final node embeddings yields a heuristic estimate of the state's distance to the goal. The specific aggregation and prediction mechanisms are detailed in the architecture section.

**GNNs for Classical Planning** GNNs have been used for learning heuristic values and general policies for classical planning. In the work by Ståhlberg, Bonet, and Geffner (2022a,b), GNNs are used to construct general policies, with a network that represents, instead of a graph, a relational structure defined by a graph. This is done to reflect the fact that a state $s$ in our planning problems represents a relational structure that occurs among the objects. Such relational structure for $s$ is defined by the set of objects $o \in O$, the set of domain predicates $p \in \mathcal{P}$, where each predicate $p$ is a relational symbol $r$ of the relational structure, and the atoms $q = r(o_1, ..., o_m)$ that are true in $s$. 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 $s$. These considerations lead the authors to define a graph structure where objects represent the nodes, atoms represent labeled edges, connecting objects to atoms if the object appears in the atom, and the types of predicates define the labels (or types) of these edges. For example, if $r(o_1, o_2)$ is true in a state $s$, the input graph for $s$ contains a labeled edge for $r(o_1, o_2)$ connected to both $o_1$ and $o_2$, labeled with $r$. Each predicate is associated with a doppelganger predicate, called "goal predicate", that is used whenever the predicate is actually used as a goal in an instance. For this reason, together with the types of arcs defined by the predicates, the relational structure also presents an arc for each such goal predicate.

For each relation $r$, there will be a different Multi-Layer Perceptron ($MLP_r$). The AGGREGATE and UPDATE functions are applied to all the nodes in the GNN (i.e. to all the objects in the problem) in every layer. The READOUT function is then used to compute a value, which is used to map a given planning state *s* into a real value *V(s)*. The architecture uses one $MLP_r$ for each relational symbol $r$, one $MLP_U$ for the UPDATE function, and two nets

$MLP_1$ and $MLP_2$ for the READOUT function (this will be better explained later in Algorithm 1). The key idea of this approach is to train a model on some instances, and use the trained parameters for other instances, too. The transfer of knowledge happens through the MLPs associated to each relational symbol. That is, when we approach a new instance, the architecture that we build will use the learned parameters off-the-shelf. In what follows, we explain how to adapt this mechanism to numeric planning.

## Architecture for Numeric Planning

Our architecture is inspired by the work of Ståhlberg, Bonet, and Geffner (2022b) on general policies for classical planning. We formally extend it to handle numeric planning problems, addressing two crucial challenges: (i) representing numeric fluents and conditions in a graph-based structure; (ii) integrating numeric values into node embeddings.

**Graph Construction** For a given numeric planning problem, firstly we define the set $R$ of relational symbols corresponding to the types of edges. $R$ is obtained from the domain $D$ of the problem, using the set of domain predicates, the set of numeric preconditions that is extracted from the action schemas in the domain, and the set of possible numeric conditions in the goals. Specifically, $R$ is defined as:

$$R = R_B \cup R_{BG} \cup R_C \cup R_{CF} \cup R_G \cup R_{GT} \tag{3}$$

where:

- **Boolean predicates** $R_B$ and $R_{BG}$ represent boolean (domain/goal) conditions.
- **Numeric preconditions** $R_C$ and $R_{CF}$ represent satisfied/unsatisfied numeric preconditions.
- **Numeric goals** $R_G$ and $R_{GT}$ represent possible/satisfied numeric goal conditions.

Then, from a specific instance $I$ of the planning problem, we extract the set of objects $O$, that will be the nodes of our graph. Finally, from a specific state $s$ of the planning problem, we extract the set of grounded atoms, i.e., the boolean and numeric conditions that are either satisfied or unsatisfied in the current state. These atoms define the edges of the graph, labeled by the corresponding relational symbols in $R$. Each edge connects the objects involved in the condition. Formally, the input graph for a state $s$ is defined as $G_s = (V_o, E_s, X)$, where:

- $V_o$ are the nodes, corresponding to objects $o \in O$.
- $E_s \subseteq V_o \times V_o \times R$ are labeled edges, with each label $r \in R$ representing a relational symbol (boolean or numeric condition).
- $X \in \mathbb{R}^{|V_o| \times d}$ is the node feature matrix, with $d \in \mathbb{N}$ being the embedding dimension.

**Numeric Goals and Preconditions** Boolean atoms can be directly used as graph edges since they have binary truth values. For numeric conditions, we define a relational symbol $r_c$ for each lifted numeric condition $c$ as follows:

$$r_c : (o_1, \ldots, o_m, x_1, \ldots, x_n) \mapsto \{0, 1\} \tag{4}$$

where $o_1, \ldots, o_m$ and $x_1, \ldots, x_n$ are the objects and numeric fluents of the condition $c$, respectively. Numeric *atoms* are obtained by *grounding the lifted numeric conditions* that occur in action preconditions and goal definitions. These atoms represent concrete numeric relations among objects in a specific state, and thus they can be evaluated as true or false. Numeric preconditions are specified in the domain of the problem, and therefore, they do not change among different instances of the same domain. On the other hand, numeric goals can be very different among instances, and handling general numeric goals is a complex task. Therefore, we lift numeric goals to a normalized form, and foresee the possible structure of lifted numeric goals that we can encounter for a given domain. In general, we support numeric goals that can be expressed as linear inequalities over numeric fluents, specifically those of form

$$\sum_{i=1}^{n} \alpha_i X_i \odot k \tag{5}$$

where:

- $\alpha_i \in \mathbb{R} \setminus \{0\}$ are constant coefficients,
- $X_i$ are numeric fluents,
- $n \in \mathbb{N}$ is the number of numeric fluents in the condition,
- $k \in \mathbb{R}$ is a constant,
- $\odot \in \{=, \leq, \geq, <, >\}$ is a comparison operator.

For example, the numeric atoms $3x(o_1) + 4y(o_2) \leq 0$ and $2x(o_3) - y(o_4) \geq 5$ are expressed by the same relation $r_{c1}$, which represents the lifted numeric condition $\alpha x(?o_1) + \beta y(?o_2) \leq k$, where $x, y$ are numeric fluents of the problem, $o_{1\ldots4}$ are objects of the problem, $?o_1$ and $?o_2$ are types of objects, $\alpha, \beta \in \mathbb{R} \setminus \{0\}$ are constant coefficients, and $k \in \mathbb{R}$ is a constant. Instead, another numeric atom of the form $y(o_1) + z(o_2) \leq 0$, where $y, z$ are numeric fluents, must be expressed with another relation $r_{c2}$. In general, two numeric atoms can be represented by the same relation as long as they involve the same set of lifted numeric fluents applied to the same number of object arguments. From Equation 5, we cannot have $\alpha_i = 0$ because it would mean having a different relation. Therefore, this formulation cannot support goals with a general structure that involves an increasing number of numeric fluents among different instances of the same domain.

**Handling Numeric Fluents** To handle numeric fluents, we associate their numeric values with the edges in the graph, which represent grounded numeric conditions. Each edge receives as input the embeddings of the objects involved, as well as the corresponding numeric values. In this way (i) object embeddings remain compact and have a fixed size, and (ii) each numeric condition (relation) has a fixed number of inputs (object embeddings plus numeric values), regardless of the instance. This strategy ensures a flexible and correct representation, supporting robust handling of numeric information within the graph. For a grounded numeric condition $q = r_c(o_1, \ldots, o_m, x_1, \ldots, x_n)$, the architecture computes a message $\mathbf{m}_q$ as follows:

$$\mathbf{m}_q = \text{MLP}_{rc}\left([\mathbf{h}_l(o_1), \ldots, \mathbf{h}_l(o_m), v_1, \ldots, v_n]\right) \tag{6}$$

where each $v_i$ is the value of the grounded numeric fluent $x_i$ associated with the condition, $\mathbf{h}_l(o_1), \ldots, \mathbf{h}_l(o_m)$ are the embeddings of the objects $o_1, \ldots, o_m$ at the iteration $l$, and $\text{MLP}_{rc}$ is the Multi-Layer Perceptron responsible for the relation $r_c$. This design keeps object embeddings compact and of fixed size, independently of the number of numeric fluents.

**Embedding Propagation and Update** The embedding propagation and update mechanism plays a crucial role in ensuring that the GNN can effectively integrate relational and numeric information while maintaining the structural invariances of the planning state representation. Specifically, each layer $l$ of the GNN enables the exchange of information among objects through the relational *atoms* (boolean or numeric) that connect them, capturing the dependencies crucial for evaluating heuristic values.

At each layer $l$, we perform two main steps for each object node $o$: *aggregation* and *update*. For each object $o$, we collect messages from all *atoms* $q$ in which $o$ appears. Each message $\mathbf{m}_q$ is computed as described in Equation 6, using the embeddings of the involved objects and the corresponding numeric values. We then aggregate these messages into a single vector:

$$\mathbf{a}_o^{(l)} = \text{AGGREGATE}\left(\{\mathbf{m}_q[o] \mid o \in q\}\right) \tag{7}$$

The choice of aggregation function (e.g., add, mean, or smooth-max) is a hyperparameter of the architecture determining how information from different relations is combined. After aggregation, we update the embedding of each object node $o$ using the aggregated vector:

$$\mathbf{h}_{l+1}(o) = \text{UPDATE}\left(\mathbf{h}_l(o), \mathbf{a}_o^{(l)}\right) \tag{8}$$

This function integrates the current embedding with the aggregated relational information, progressively refining the object representation across layers. After $L$ layers of propagation, the final object embeddings $\mathbf{h}_L(o)$ capture both the relational structure and the numeric context of each object. Finally, we compute the heuristic value for the state by passing the final object embeddings through a readout function:

$$h(s) = \text{MLP}_2\left(\sum_{o \in O} \text{MLP}_1(\mathbf{h}_L(o))\right) \tag{9}$$

This mechanism ensures that information flows efficiently through the graph, enabling the GNN to produce an informative and compact heuristic estimate for the current state.

**State Encoding and Heuristic Computation** Algorithm 1 summarizes the message-passing process and heuristic value computation. The input for a state $s$ is the set of objects $O$ and the set of *atoms* $Q$. $L$ and $d$ are hyperparameters describing the number of layers and the embedding dimension, respectively. Lines 1–5 handle the initialization of the nodes. The first part of the embedding is initialized with zeros, while the second part is randomized. The reason behind this partial randomization derives from (Abboud et al. 2021), where it is shown that: (i) a randomization in the embedding vector can greatly enhance the expressiveness of the GNN; (ii) for datasets with varying expressiveness requirements (as in our case), the best performances are achieved with a partial randomization. Lines 7–8 concern the message-passing operation, in which the messages flow from each object $o_i$ to each *atom* $q$ that includes $o_i$, $q = r(o_1, ..., o_m, x_1, ..., x_n), 1 \leq i \leq m$. The MLP associated with relation $r$ takes as input the embeddings of the objects involved in $q$, plus the numeric values of the involved numeric fluents (if $r$ represents a numeric condition). It then outputs a vector of size $d$ for each object involved in $q$, which is passed back to the corresponding object. Lines 9–10 correspond to the aggregation function, in which the messages received by each object are aggregated into a single vector of size $d$. The aggregation function is a hyperparameter. Line 11 regards the UPDATE function, where each object is updated with the aggregated message. In line 12, the final embeddings of each object are passed to the first MLP, summed together, and then passed to a second MLP to extract the heuristic value. All MLPs consist of an input layer with linear activation function, a dense layer with ReLU activation function, and a dense layer with linear activation function.

Algorithm 1: GNN mapping a state $s$ to heuristic value $h(s)$

**Input**: $\langle O, Q \rangle$. **Output**: $h(s)$.

1: **for** each $o \in O$ **do** ▷ Initialize encoding values
2: **for** $i \in (0, d/2)$ **do**
3: $h_0(o)[i] = 0$
4: **for** $i \in (d/2, d)$ **do**
5: $h_0(o)[i] = random(0, 1)$
6: **for** $l = 0$ to $L - 1$ **do**
7: **for** each $q = r(o_1, \ldots, o_m, x_1, \ldots, x_n) \in Q$ **do**
8: $\mathbf{m}_q = MLP_r([h_l(o_1), \ldots, h_l(o_m), v_1, \ldots, v_n])$
9: **for** each $o \in O$ **do**
10: $a_o^{(l)} = AGG(\{\mathbf{m}_q[o] \mid o \in q\})$
11: $h_{l+1}(o) = MLP_U(h_l(o), a_o^{(l)})$
12: $h(s) = MLP_2(\sum_o MLP_1(h_L(o)))$

**Extending to More General Numeric Goals** The number of objects and numeric fluents involved in a specific relation $r$ is fixed for the domain; therefore, goals in which the number of involved objects changes across different instances cannot be directly represented. For example, in the `Farmland` domain, one of the numeric goals is a function of *all* the objects of type farm that occur in the instance. E.g., with two objects, it would be $\alpha_1 X_1 + \alpha_2 X_2 \geq k$, and with four objects $\alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4 \geq k$, where each $\alpha_i \in \mathbb{R}$, $X_i$ is the numeric fluent associated with the $i^{\text{th}}$ farm object, and $k \in \mathbb{R}$ is a constant. In order to treat relations with arbitrary arity $n$, we can decompose each goal with $n$ objects into a set of "binary" goals, and represent it using only one relation, so that $\alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4 \geq k$ is handled by three binary goals: $\alpha_1 X_1 + \alpha_2 X_2 \geq k$, $\alpha_2 X_2 + \alpha_3 X_3 \geq k$ and $\alpha_3 X_3 + \alpha_4 X_4 \geq k$. Of course, mathematically this transformation is not equivalent to the original constraint, and it does not preserve its semantics.

However, notice that our architecture does not operate by interpreting the numeric expression as a global mathematical function. Instead, it reasons over two key components: (i) the edges that connect the involved objects in the graph, and (ii) the numeric values associated with those edges.

This goal decomposition can be carried out in multiple ways, as long as the two key components are preserved. E.g., if $\alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4 \geq k$ is decomposed into the two binary goals $\alpha_1 X_1 + \alpha_2 X_2 \geq k$ and $\alpha_3 X_3 + \alpha_4 X_4 \geq k$, we lose the relational connection between the objects of $X_2$ and $X_3$, splitting the goal into two *dis*connected subgoals. This would hinder the ability of the model to reason over interactions between the full set of objects involved in the original constraint. Our decomposition strategy avoids this issue by using a chain of overlapping binary conditions, which ensures all objects remain indirectly connected, while minimizing the number of edges required to represent the original constraint. Specifically, from a goal condition of arity $n$, we generate $n - 1$ binary goal conditions. This results in a chain-like structure that captures the overall dependency with fewer connections, reducing representational complexity while maintaining the relational and numeric information needed by our architecture. This decomposition is applied exclusively within the heuristic function and does not alter the original structure of the planning problem.

## Experimental Evaluation

Our experimental analysis aims at evaluating the performance of the proposed approach to learning heuristics for numeric planning problems. A question that the analysis tries to answer is whether our formulation improves on the state of the art in learning-based heuristics. We compare the heuristic learned by our architecture with one in which the numeric structure of the problem is completely ignored, namely $h^0$, which corresponds to Ståhlberg, Bonet, and Geffner (2023) formulation, and with $h^{ccWLF}_{rank}$, the Graph-based heuristic proposed by Chen and Thiébaux (2024), which is the state of the art in learning-based heuristics for numeric planning problems. To assess the performance gains from incorporating numeric information, we experimented two variants of our learned heuristic: $h^c$, which corresponds to our architecture without the encoding of numeric values on the edges, i.e, where numeric conditions are abstracted as boolean conditions, and $h^n$, which is our more advanced configuration where numeric values are provided to the MLP associated with each numeric condition. To ensure a fair comparison and assess which heuristic offers a better balance between informativeness and computational cost, all heuristics run using greedy best-first-search. That is, the search is guided by evaluation function $f(n) = h(n)$ with $h(n)$ be either $h^0$, $h^c$, $h^n$, or $h^{ccWLF}_{rank}$. This way, the planner becomes extremely sensitive to the quality of the heuristic. This choice aligns with the search algorithm used in the other systems (Chen and Thiébaux 2024; Scala et al. 2020), ensuring a consistent basis for comparison. Heuristics $h^0$, $h^c$, $h^n$ are run on-board of a Python planner that uses parsing from the Unified Planning Library (Micheli et al. 2025), and grounding from the ENHSP planning system.

**Benchmarks and Training Setup** As numeric planning problems, we selected nine domains from the International Planning Competition (IPC) 2023 Numeric Track (Taitler et al. 2024): `Counters`, `Fo-Counters`, `Sailing`, `Fo-Sailing`, `Mprime`, `Expedition`, `Hydropower`, `Farmland`, and `Fo-Farmland`. This set includes simple numeric problems (`Sailing`, `Counters`, `Farmland`) in which traditional heuristics excel, their linear numeric variants that are hard for traditional heuristics (`Fo-Counters`, `Fo-Sailing`, `Fo-Farmland`), and simple numeric problems that are hard for traditional heuristics due to the presence of dead-ends (`Expedition`, `Hydropower`). We also add `Mprime` to see what happens in a domain with a pronounced classical planning structure. Each domain consists of 20 instances, with increasing difficulty scaling from the first instance to the last. The datasets to train our networks is generated using ENHSP run as an optimal numeric planner (Scala et al. 2020).

For each domain, training is carried on small instances, obtained by taking the smallest instances in the benchmark suite and creating new ones by changing the initial values. More precisely, we perform a random walk starting from the initial state, and use the reached state as the new initial state. We then solve the problem and store every traversed state of the solution in our dataset, together with the actual distance to the goal. Details regarding the number of objects used in training, validation, and testing can be found in Table 1. The networks are therefore trained over tuples $(s, v)$ where $s$ is a state for a planning problem, and $v$ is the optimal distance to the goal. We collected N tuples $(s, v)$ for each domain, up to 40000 tuples for each instance. The best (tuned) number of layers $L$ is 30 for every domain, and we found the best results not using regularization, with the exception of `Fo-Sailing`, which is trained with L1 regularization set to 0.005. The best embedding size $d$ found is 60. The loss function used during training is MSE (minimum square error) with the Adam optimizer, and the learning rate is set to 0.0002. The training procedure has the maximum number of epochs set to 1000, with an early stopping that terminates the training early if the validation error does not improve after 30 epochs. In almost all cases, the training procedure stops before epoch 100, with the sole exception of `Hydropower`, which stops around epoch 150. Finally, while for the classical planning counterpart the smooth-max aggregation function showed the best results, in our architecture every domain performed better with the add-aggregation function. What we found is that, in particular with our final architecture that handles numeric values, the smooth-max aggregation function has the tendency to cancel the numeric part of our representation. The add and smooth-max aggregation functions are formalized in Hamilton (2020) and Ståhlberg, Bonet, and Geffner (2022a). We also tried two other aggregation functions, namely mean and max aggregation (explained in Hamilton (2020)), but they did not yield any positive results. For deriving $h^{ccWLF}_{rank}$, we used the training procedure following the authors' instructions in Chen and Thiébaux (2024).

The training process yields a model (heuristic function) that can be applied to any instance of the domain used for

| Domain | # Samples | | # Objects | | |
|---|---|---|---|---|---|
| | Train | Validation | Train | Validation | Test |
| Counters | 200000 | 41000 | [4–7] counters | 8 counters | [4–40] counters |
| Fo-Counters | 89000 | 30000 | [2–4] counters | [5–6] counters | [2–20] counters |
| Sailing | 75000 | 10000 | [1–2] boats, [1–5] people | 4 boats, 2 people \|1 boat, 6 people | [1–4] boats, [1–10] people |
| Fo-Sailing | 50000 | 12000 | 1 boat, [1–2] people | 1 boat, 3 people | [1–5] boats, [1–4] people |
| Mprime | 100000 | 20000 | [4–7] foods, [2–7] pains [1–3] pleasures | 10 foods, 1 pleasure, 7 pains 12 foods, 5 pleasures, 4 pains | [4–22] foods, [2–44] pains [1–16] pleasures |
| Expedition | 45000 | 12000 | [6–8] waypoints | 9 waypoints | [6–15] waypoints |
| Hydropower | 15000 | 2400 | [1010–1150] power | [1160–1190] power | [1010–2050] power |
| Farmland | 150000 | 50000 | [2–4] farms | 4 farms | [2–10] farms |
| Fo-Farmland | 80000 | 20000 | 2 farms | 2 farms | [2–10] farms |

Table 1: Number of samples for training and validation, and objects per split (train, validation, test) across domains.

| Domain | $h^{ccWLF}_{rank}$ | $h^0$ | $h^c$ | $h^n$ |
|---|---|---|---|---|
| Counters | 3 | 0 | 2 | **10** |
| Fo-Counters | 4 | 0 | 2 | **8** |
| Sailing | **5** | 0 | 0 | 2 |
| Fo-Sailing | **8** | 1 | 2 | **8** |
| Mprime | **13** | 7 | 9 | 8 |
| Expedition | 3 | 1 | 1 | **6** |
| Hydropower | – | 0 | 1 | **9** |
| Farmland | – | 0 | 0 | **10** |
| Fo-Farmland | – | 2 | 4 | **14** |
| **Total** | 36 | 11 | 21 | **75** |

Table 2: Coverage analysis among three variants of our architecture ($h^0$,$h^c$,$h^n$) and $h^{ccWLF}_{rank}$.

training. Experiments are run on a single Intel Xeon Gold 6140M (2.30GHz) core with a 5-minutes timeout for search and 8GB of memory. For our GNN heuristics, we use an NVIDIA v100 32GB GPU; for $h^{ccWLF}_{rank}$ we follow the authors' suggestion and we do not use it. We kept an 8GB memory limit for our models, including both GPU and CPU memory.

**Results** Table 2 reports results on the problem coverage obtained by all the considered heuristics. From these results it emerges quite clearly that our architecture benefits a lot from making it sensitive to the numeric information in the problem. Indeed, $h^n$ consistently outperforms the other variants in most domains, except for Mprime. This is a domain with few numeric conditions, and the $h^n$ infrastructure turns out to be overhead. Interestingly, Mprime is also the domain where $h^{ccWLF}_{rank}$ shines. $h^{ccWLF}_{rank}$ seems to be much more efficient when the domains have a small numeric component. Indeed, with the exception of the two sailing domains (Sailing, Fo-Sailing) and the aforementioned Mprime, $h^{ccWLF}_{rank}$ is outperformed by $h^n$. In Mprime, the numeric values are used only to check if there is at least one item of that type (i.e. if we want to take the action "drink", there is a precondition that checks if the number of drinks is at least 1). To reach the goal, knowing the numeric values is not important; knowing when a numeric condition is true or false is enough. The additional knowledge of $h^n$, in this case, just slows down the heuristic without adding useful information for reaching the goal.

To analyze the tradeoff between guidance and computational cost, in Table 3 we show the coverage, the average time and the average number of expanded nodes for each domain, comparing our final architecture with $h^{ccWLF}_{rank}$ and the state-of-the-art traditional heuristic $h^{mrp}$ (Scala et al. 2020), implemented in the ENHSP planning system. To ensure a fair comparison, the averages for time and expanded nodes were computed only on the instances successfully solved by both our heuristic and the one being compared. The results show that the decomposition introduced to handle goals involving an increasing number of numeric fluents is effective. This is particularly evident in the Fo-Farmland domain, where our heuristic significantly outperforms traditional ones, both in terms of guidance (fewer nodes expanded) and planning efficiency (lower computation time). Overall, the tested approaches seem very complementary.

**Comparison with $h^{ccWLF}_{rank}$** There are two key differences between Chen and Thiébaux (2024) approach and ours. The first is that in their work, instead of GNNs, they use graph kernels to capture the relational structure of planning domains, leading to heuristics that are faster compared to GNN-based heuristics. In our work, we focus on the informativeness of our heuristics, leading to heuristics that are slower, but usually more informative. The second difference is that their graph structure considers objects, grounded numeric or boolean fluents, propositional goals, and numeric fluents in the goals as nodes, and there are edges between an object and a fluent, or a goal, if the object is instantiated in the fluent or in the goal. Numeric values are incorporated directly in the node features of numeric fluents. However, numeric conditions. i.e., the logical expressions that must be satisfied for an action to be applicable or for a goal to be achieved (such as $x_1 + x_2 \leq 10$) are not explicitly repre-

| Domain | $h^{mrp}$ | | | $h^n$ | | | $h^{ccWLF}_{rank}$ | | | $h^n$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | C | T | N | C | T | N | C | T | N | C | T | N |
| Counters | **12** | 3.06 | 6272.90 | 10 | 13.60 | **215.90** | 3 | 10.13 | 35.66 | **10** | **0.40** | **8.30** |
| Fo-Counters | 5 | 1.47 | 8214 | **8** | **1.23** | **9.40** | 4 | 19.57 | 103199 | **8** | **0.98** | **7.25** |
| Sailing | **20** | **0.99** | **175.50** | 2 | 12.94 | 182.50 | **5** | * | * | 2 | * | * |
| Fo-Sailing | 1 | 17.72 | 611480 | **8** | **6.45** | **69** | **8** | **9.73** | **176.50** | **8** | 27.92 | 244.50 |
| Mprime | **12** | **1.03** | **25.62** | 8 | 19.77 | 375.75 | **13** | 28.91 | 137.85 | 8 | **0.98** | **11.57** |
| Expedition | 3 | 12.71 | 213120 | **6** | **4.8** | **61** | 3 | 15.68 | 198475 | **6** | **4.8** | **61** |
| Hydropower | 1 | **0.77** | 3818 | **9** | 2.26 | **16** | 0 | – | – | **9** | **18.89** | **117** |
| Farmland | **20** | **1.21** | **234.60** | 10 | 22.31 | 853.40 | 0 | – | – | **10** | **22.31** | **853.40** |
| Fo-Farmland | 5 | 52.14 | 531201 | **14** | **2.130** | **56.30** | 0 | – | – | **14** | **25.65** | **634.85** |

Table 3: Performance comparison for coverage (C), average time in seconds (T), and average expanded nodes (N). The LHS of the table shows the comparison of $h^{mrp}$ and $h^n$ for the instances solved by both; the RHS of the table shows the comparison of $h^{ccWLF}_{rank}$ and $h^n$ for the instances solved by both. "–" denotes that the heuristic has not been considered due to lack of coverage. * For Sailing, no task was solved by both $h^{ccWLF}_{rank}$ and $h^n$, hence no values are reported for T and N.

sented in their graph. Instead, the graph only includes the numeric fluents themselves, and in the case of numeric goals, a scalar value called goal error is used to indicate how far the current state is from satisfying that numeric goal condition. This means that the structure of numeric conditions is flattened into a numeric distance, and the logical dependencies between objects involved in a condition are not captured structurally in the graph. In our work, instead, the nodes in the graph represent only the objects, and each boolean fluent, boolean goal, and grounded numeric condition (either a numeric precondition or a numeric goal) is represented as an edge connecting the objects involved. This let us reason directly over conditions, rather than fluents, capturing more faithfully the relational and logical structure of the numeric planning problem. This is relevant in domains where the numeric relations themselves (rather than just the values of individual fluents) are essential to understanding progress toward the goal. A similar distinction arises also comparing symbolic heuristics (Scala and Bonassi 2025).

This difference can be seen in Table 2: in Sailing and Fo-Sailing, where goals are boolean conditions, and the numeric structure of the problem (the coordinates of boats and people to be saved) is already encapsuleted in the values of the grounded numeric fluents, $h^{ccWLF}_{rank}$ works better. On the other hand, in the domains where goals are numeric conditions (Counters and Fo-Counters) and in those where numeric preconditions are important to capture the structure of the problem (Expedition), our architecture works significantly better. Unfortunately, we do not have a comparison for Hydropower, Farmland, and Fo-Farmland since $h^{ccWLF}_{rank}$ does not support them. In particular, Hydropower is unsupported by Numeric Fast Downward (Aldinger and Nebel 2017), the planner underlying $h^{ccWLF}_{rank}$. As for Farmland and Fo-Farmland, their goals could not be expressed in a form compatible with $h^{ccWLF}_{rank}$, which only supports numeric goals in the form $X(\leq, <, =)Y$.

## Discussion and Conclusion

In this work, we have proposed a new approach for tackling numeric planning problems using GNNs as heuristic functions. Our method extends the architecture of Ståhlberg, Bonet, and Geffner (2022a) to handle numeric conditions and numeric values, integrating the learned heuristics into a numeric planner implemented in Python. Compared to classical planning, applying learning-based techniques to numeric planning presents several additional challenges. First, the lack of a dedicated learning track in past planning competitions for numeric domains means that publicly available training data is limited. Many numeric planning domains also lack instance generators, making it difficult to produce large and diverse datasets. Second, scaling remains a more significant obstacle in numeric planning. Finding valid plans often becomes computationally difficult even with a relatively small number of objects. As a result, the diversity in training instances, especially in terms of object counts, is typically low. Additionally, unlike in classical planning where suboptimal plans are often used to enrich the training data, generating useful suboptimal plans in numeric domains has proven problematic. Our experiments using suboptimal plans consistently led to large validation errors, and other recent work such as Chen and Thiébaux (2024) also relies exclusively on optimal plans.

Finally, a distinctive difficulty in numeric planning is the high sensitivity of plan length to small changes in problem instances. For example, adding a single object can drastically increase the number of actions required to solve the problem. This poses a generalization challenge: a model trained only on plans with up to 100 actions may struggle to handle similar instances requiring 400 actions. This effect is evident in our experimental results, particularly in the Sailing domain, where even a small change in the problem leads to a significant increase in plan length and a corresponding drop in performance. Despite these challenges, our experimental analysis demonstrates that the proposed approach yields promising results compared to the state of the art. We believe these results highlight the potential of learning-based methods for numeric planning, and we plan to further refine our architecture, expand training datasets, and explore new strategies for improving generalization in future work.

## Acknowledgments

This work has been supported by: MUR PRIN-2020 project RIPER (n. 20203FFYLK); PNRR MUR project PE0000013-FAIR, cascade funding call, ResilientPlans; PNRR MUR project SERICS (PE00000014), cascade funding call, SOS-AI.

## References

Abboud, R.; Ceylan, İ. İ.; Grohe, M.; and Lukasiewicz, T. 2021. The Surprising Power of Graph Neural Networks with Random Node Initialization. In Zhou, Z., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 2112–2118. ijcai.org.

Aldinger, J.; and Nebel, B. 2017. Interval based relaxation heuristics for numeric planning with action costs. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, 15–28. Springer.

Chen, D.; and Thiébaux, S. 2024. Graph learning for numeric planning. *Advances in Neural Information Processing Systems*, 37: 91156–91183.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 68–76.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Hamilton, W. L. 2020. *Graph representation learning*. Morgan & Claypool Publishers.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. ISBN 978-3-031-00456-8.

Horčík, R.; and Šír, G. 2024. Expressiveness of graph neural networks in planning domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 281–289.

Horčik, R.; Šír, G.; Šimek, V.; and Pevnỳ, T. 2025. State Encodings for GNN-Based Lifted Planners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26525–26533.

Karia, R.; and Srivastava, S. 2021. Learning generalized relational heuristic networks for model-agnostic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 8064–8073.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Rossetti, N.; Tummolo, M.; Gerevini, A. E.; Putelli, L.; Serina, I.; Chiari, M.; and Olivato, M. 2024. Learning General Policies for Planning through GPT Models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 500–508.

Scala, E.; and Bonassi, L. 2025. On Using Lazy Greedy Best-First Search with Subgoaling Relaxation in Numeric Planning Problems. volume 35, 245 – 249. Cited by: 0; All Open Access, Gold Open Access.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.*, 68: 691–752.

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, 574–584.

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 International Conference on Automated Planning and Scheduling*, volume 32, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 474–483.

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

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fiser, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Mag.*, 45(2): 280–296.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Wang, R. X.; and Thiébaux, S. 2024. Learning Generalised Policies for Numeric Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 633–642.