Graph Learning for Numeric Planning

Dillon Z. Chen[1,2] **Sylvie Thiébaux**[1,2]
[1]LAAS-CNRS, University of Toulouse [2]The Australian National University
{dillon.chen,sylvie.thiebaux}@laas.fr

Abstract

Graph learning is naturally well suited for use in symbolic, object-centric planning due to its ability to exploit relational structures exhibited in planning domains and to take as input planning instances with arbitrary numbers of objects. Numeric planning is an extension of symbolic planning in which states may now also exhibit numeric variables. In this work, we propose data-efficient and interpretable machine learning models for learning to solve numeric planning tasks. This involves constructing a new graph kernel for graphs with both continuous and categorical attributes, as well as new optimisation methods for learning heuristic functions for numeric planning. Experiments show that our graph kernels are vastly more efficient and generalise better than graph neural networks for numeric planning, and also yield competitive coverage performance compared to domain-independent numeric planners. Code is available at `https://github.com/DillonZChen/goose`

1 Introduction

Planning requires long range reasoning over combinatorially large state spaces. Numeric planning is an extension of classical planning in which states have numeric variables and the underlying transition system is built from inequality conditions and assignments over arithmetic expressions of such variables. It was formalised in PDDL 2.1 [FL03] and is undecidable in the general case [Hel02] which makes it more difficult than classical planning which is PSPACE-complete [Byl94]. Numeric planning is a well-established problem in the symbolic AI community and exhibits significant research effort [CCFL13, IM17, SHTR20, KSP+22, KSB23, SKB23], but this expressivity result implies that building a general, scalable numeric planner is a challenging problem.

Learning for Planning (L4P) is a research direction which focuses on learning to solve problems from a specified domain in an automated supervised manner [TTTX20, STT20, FGT+22, KS21, SBG22, SBG23, MLTK23, CTT24a, SDS+24, RTG+24, APK24]. Planning tasks in L4P are assumed to exhibit a factored, symbolic representation, which allow us to generate training data in a matter of seconds from easy to solve tasks with a domain-independent planner. We can then learn domain knowledge in a supervised manner that scales planners to significantly larger planning tasks.

This is in contrast to Reinforcement Learning where agents do not require access to well-defined models but spend significant amounts of time exploring and learning from rewards [SB98]. Regardless, several works have showed that encoding or learning symbolic models for sequential decision making reasoning and embodied AI tasks [LCZ+21, ZYP+22, LSS+22, SCK+23, KVS+23, LPM23] provided better performance and transparency over end-to-end reinforcement learning methods. Furthermore, it was shown recently that classical ML methods are much better suited for L4P than deep learning methods for symbolic planning [CTT24b] as they (1) can generalise well from small training data, (2) are orders of magnitude more efficient to train and evaluate than deep learning methods, which is important in time sensitive tasks such as planning, and (3) have interpretable features to understand what is being learned.

In this paper we study whether this fact carries over to Learning for Numeric Planning (L4NP) [WT24] which now requires reasoning over logic and arithmetic. It is reasonable to think that because neural networks are function approximators, they may offer better reasoning capabilities over numbers than

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

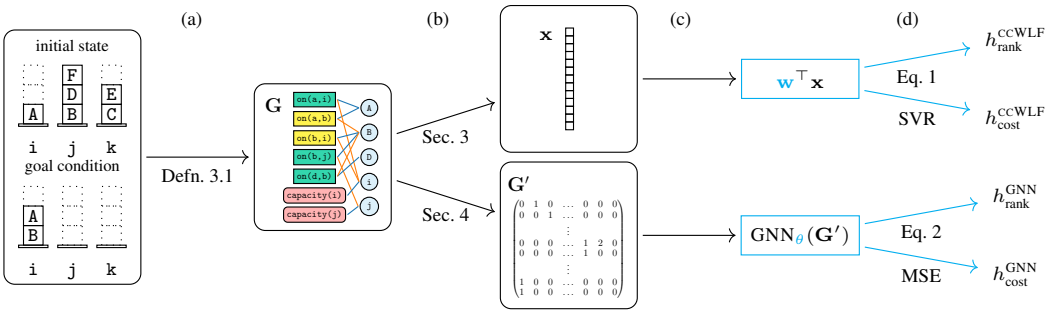


Figure 1: The GOOSE framework for learning heuristic functions for numeric planning. Cyan colours indicate components that are influenced by the training phase. (a) A numeric planning state and goal condition is encoded into a graph $\mathbf{G}$ via the νILG representation defined in Defn. 3.1. (b) Graphs are either embedded into vectors $\mathbf{x}$ in Euclidean space with the CCWL kernel from Sec. 3 or transformed into a graph $\mathbf{G}'$ with a real-valued matrix representing node features as inputs into GNNs described in Sec. 4. (c) Features $\mathbf{x}$ are fed into a linear model, whereas transformed graphs $\mathbf{G}'$ are fed into GNNs. (d) Linear models are either trained by the ranking formulation in Eq. 1 or by Support Vector Regression (SVR) with a linear kernel. GNN models are either trained by the ranking formulation in Eq. 2 or by backpropagation minimising the mean squared error (MSE) loss.

just symbols alone. In this paper, we describe the GOOSE[1] framework with classical ML and deep learning configurations for learning heuristic or value functions for use with search in L4NP. Fig. 1 illustrates the GOOSE framework and we outline our contributions as follows.

- We introduce a new graph representation of numeric planning tasks for use with classical and deep graph learning methods, namely graph kernels and graph neural networks, respectively.
- We extend the WL kernel [SSVL+11] to handle graphs with both continuous and categorical attributes in a meaningful way which we call the CCWL kernel.
- We introduce new ranking formulations [GKL16, CEKP23, HTT+24] for learning heuristic or value functions with linear programs.

The structure of the remainder of the paper is as follows. In Sec. 2, we provide the necessary formalism and background for numeric planning, as well as relevant notation. In Sec. 3, we introduce a new graph encoding νILG and CCWL kernel for generating features for numeric planning tasks. In Sec. 4, we introduce a deep learning architecture for L4NP using graph neural networks. In Sec. 5, we describe optimisation methods for L4NP, involving a new ranking formulation for learning heuristic functions. In Sec. 6, we describe our experimental setup and results. Related work is discussed in Sec. B in the appendix. We conclude the paper with final comments in Sec. 8.

2 Background

Numeric Planning Task. A numeric planning task can be viewed as a compact representation of a deterministic, goal-conditioned Markov Decision Process with the use of predicate logic and relational numeric variables. A majority of the remainder of this section formalises the necessary components of a numeric planning task we use in the paper.

A numeric planning task [FL03] is given by a tuple $\Pi = \langle X_p, X_n, A, s_0, G \rangle$ where X_p is a finite set of propositional variables with domain $\{\top, \bot\}$ and X_n is a finite set of numeric variables with domain $\mathbb{R}$. Let $X = X_p \cup X_n$ denote the set of state variables, where a state is a total assignment of values the propositional and numeric variables. The variables implicitly induce a possibly infinite set of states S, where s_0 is the initial state.

A propositional condition is a positive (resp. negative) literal $x = \top$ (resp. $\bot$) for some propositional variable $x \in X_p$, and a numeric condition has the form $\xi \trianglerighteq 0$ where ξ is an arithmetic expression over numeric variables and $\trianglerighteq \in \{\geq, >, =\}$. We write $[x]^s$ (resp. $[\xi]^s$) for the value of a state variable x (resp. expression ξ) in state s, and $V(\xi)$ for the set of numeric state variables in ξ. A state s satisfies a set of conditions (i.e. a set of propositional and numeric conditions) if each condition in the set evaluates to true given the values of the state variables in s. The goal G is a set of conditions and we write G_p (resp. G_n) for the subset of propositional (resp. numeric) goal conditions.

[1] **G**raphs **O**ptimised f**O**r **S**earch **E**valuation

The set A contains a finite number of actions, each consisting of preconditions and effects. Action preconditions $\mathrm{pre}(a)$ are sets of conditions, and action effects assign Boolean values to propositional variables and assign the value of an arithmetic expression to numeric variables. An action a is applicable in a state s if s satisfies $\mathrm{pre}(a)$, in which case its successor $a(s)$ is the state where the effects $\mathrm{eff}(a)$ are applied to the state variables in s. If a is not applicable in s, we set $a(s) = s_\perp \notin S$. Each action a has a cost $c(a)$ given by an arithmetic expression.

A plan for a numeric planning task is a sequence of actions $\pi = a_1, \ldots, a_n$ such that $s_i = a_i(s_{i-1}) \neq s_\perp$ for all $1 \leq i \leq n$ and s_n satisfies G; we call $s_0, s_1, \ldots, s_n$ the plan trace of the plan. The plan length $|\pi|$ and the plan cost are the number of actions in the plan, and the sum of their cost, respectively. A plan is optimal if it has the minimum cost among all plans. A numeric planning task is solvable if there exists a plan for it, and is unsolvable otherwise. A state s is a deadend if the task with the initial state replaced with s is unsolvable. Satisficing planning refers to the problem of finding a plan if it exists, or proving that the problem is unsolvable. Optimal planning refers to the problem of finding an optimal plan if it exists, or proving that the problem is unsolvable.

Lifted representation. Numeric planning tasks can be compactly encoded in a lifted representation $\langle \mathcal{O}, \Sigma_p, \Sigma_f, \Sigma_a, \mathcal{A}, s_0, G \rangle$ whereby state variables are derived from a set of predicates, functions, and objects. Formally Σ_p and Σ_f are sets of predicate and function symbols, respectively. Each symbol $\sigma \in \Sigma_p \cup \Sigma_f$, has an arity $n_\sigma \in \mathbb{N} \cup \{0\}$ which depends on σ. Predicates and functions take the form $p(x_1, \ldots, x_{n_p})$ and $f(x_1, \ldots, x_{n_f})$, respectively, where the x_is are their arguments. Given the set $\mathcal{O}$ of objects, the propositional and numeric variables are obtained by substituting objects for the arguments of the predicates and functions, resulting in the grounded form $p(o_1, \ldots, o_{n_p})$ and $f(o_1, \ldots, o_{n_f})$, respectively, where the o_is are objects. Similarly, actions can be represented in a lifted form via a set Σ_a of action symbols and a set $\mathcal{A}$ of action schemata mapping action symbols to their lifted precondition and effect definitions in terms of predicates and functions. Grounding the set of action schemata results in the set of actions A of the planning task. Details are not needed to understand this paper. A domain is a set of numeric planning tasks sharing the same set of Σ_p, Σ_f, Σ_a, and $\mathcal{A}$, and may have constant objects, objects which are shared across all tasks in the domain.

Example: Capacity Constrained Blocksworld. To help digest some of the definitions of numeric planning, we provide an example with a planning domain we call *Capacity Constrained Blocksworld* (ccBlocksworld). It is an extension of the original Blocksworld domain in which state consists of towers of blocks and the objective is to stack and unstack blocks to achieve a goal configuration. It is also a special case of the Hydraulic Blocksworld domain for planning with state constraints, in which blocks are placed on top of pistons which rise or fall depending on the configurations of other pistons [HIR$^+$18].

In ccBlocksworld, we have a maximum number of tower locations, and each tower has a base limited by the number of blocks it can hold. To model this domain in the lifted representation, we retain the predicate $\texttt{on}(x, y)$ from the original Blocksworld, which indicates block x is on another block or base y. Next, we also introduce the function $\texttt{capacity}(z)$ which denotes the remaining number of blocks that are allowed to be placed on base z. The numeric variables instantiated from `capacity` may increase or decrease depending on whether blocks are unstacked from the tower or stacked on top of it. Action schemata preconditions constrain whether a block can be placed on a tower with base that has reached its capacity limit or not. The leftmost figure in Fig. 2 illustrates an example of a ccBlocksworld problem with an initial state and goal condition. We refer to the Sec. A of the appendix for the complete state representation of the problem as well as its PDDL encoding.

Heuristics and Greedy Best First Search. State-of-the-art methods for both satisficing and optimal numeric planning [SHTR20, KSP$^+$22, CT24] employ some variant of heuristic search. A heuristic function maps a state s to $\mathbb{R} \cup \{\infty\}$ representing an estimate of the cost to reach the goal from the current state, where a value of ∞ estimates that s is a deadend. The optimal heuristic h^* maps a state to the cost of an optimal plan if it exists, and ∞ otherwise. The Greedy Best First Search (GBFS) algorithm consists of a priority queue initialised with the initial state as the only element, and a main loop which performs the following steps while the queue is non-empty: (1) pop a state s with the lowest heuristic value and some tie-breaking criterion from the queue, (2) generate the successors of s via all applicable actions, and (3) check if a successor s' is a goal, in which case terminate with the plan traced back from s', and otherwise add s' to the queue if it has not been seen before. The algorithm determines a problem is unsolvable if the main loop completes, in which case the problem has finitely many states of which all have been seen.

Graph and other notations. Let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}_{\text{cat}}, \mathbf{F}_{\text{con}}, \mathbf{L} \rangle$ denote a graph with nodes $\mathbf{V}$, undirected edges $\mathbf{E} \subseteq \binom{\mathbf{V}}{2}$, categorical node features $\mathbf{F}_{\text{cat}} : \mathbf{V} \to \Sigma_{\text{V}}$ where Σ_{V} is a finite set, continuous node features $\mathbf{F}_{\text{con}} : \mathbf{V} \to \mathbb{R}^d$ with $d \in \mathbb{N}$, and edge labels (categorical edge features) $\mathbf{L} : \mathbf{E} \to \Sigma_{\text{E}}$ where Σ_{E} is a finite set. The neighbourhood of a node $u \in \mathbf{V}$ in a graph with respect to an edge label ι is defined by $\mathbf{N}_\iota(u) = \{v \mid \exists e \in \mathbf{E}, \text{ s.t. } e = \langle u, v \rangle = \langle v, u \rangle \wedge \mathbf{L}(e) = \iota\}$. We use $\|$ to denote the concatenation operator for vectors, and $[\![N]\!]$ to denote $\{1, \ldots, N\}$.

3 Relational features for numeric planning

In this section, we describe an automatic method for generating embeddings for numeric planning tasks that may be used for any downstream inference model. The method is an extension of the feature generation method for classical planning [CTT24b] and consists of two main steps: (1) generating a graph representation of a planning task, and (2) running a variant of the WL-algorithm for generating features for the graph [SSVL$^+$11]. Extending the first step of the method is simple as it is easy to extend the graph encoding to capture numeric information of the task. This is done in Sec. 3.1 where we introduce the Numeric Instance Learning Graph (νILG) representation for numeric planning tasks. The second step is more difficult as we require constructing a WL-algorithm variant that can handle both categorical and continuous nodes features in a meaningful way for numeric planning. This is where we introduce the CCWL algorithm in Sec. 3.2 that handle such node features. Thus, we can generate features for numeric planning tasks by first converting them into the νILG representation, and then running the CCWL algorithm on them.

3.1 Graph encoding of numeric planning tasks

We begin by describing our graph encoding of a planning task, namely the Numeric Instance Learning Graph (νILG). Similarly to the classical case, the graph representation does not encode the transition model of the planning task nor requires grounding all possible variables in the planning task. Thus, our encoding only requires a first-order representation of states, and therefore applies to problems whose transition model is unknown such as in model-free reinforcement learning.

We begin with a descriptive definition of the graph with an example Fig. 2 illustrating a subgraph of the νILG representation of the example ccBlocksworld problem. In the figure, the nodes in the graph represent the objects (light blue), propositional variables true in the state (green), numeric variables (red), propositional goals (yellow) and numeric goals (not present in the example) of the problem. Blue (resp. orange) edges connect object nodes to goal and variable nodes where the object is instantiated in the first (resp. second) argument of the corresponding node variable or condition.

We provide the formal definition below. Let $X_p(s)$ denote the set of propositional variables that are true in s, $X_n(s)$ the set of numeric variables, and $X(s) = X_p(s) \cup X_n(s)$.

Definition 3.1 (Numeric Instance Learning Graph)**.** The *Numeric Instance Learning Graph (νILG)* of a numeric planning task in the lifted representation $\Pi = \langle \mathcal{O}, \Sigma_p, \Sigma_f, \Sigma_a, \mathcal{A}, s_0, G \rangle$ is a graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}_{\text{cat}}, \mathbf{F}_{\text{con}}, \mathbf{L} \rangle$ with

- nodes $\mathbf{V} = \mathcal{O} \cup X(s_0) \cup G$, where we assume w.l.o.g. that $V(g) \subseteq X(s_0)$ for all $g \in G_n$,
- edges $\mathbf{E} = \bigcup_{p=\sigma(o_1,\ldots,o_{n_\sigma}) \in X(s_0) \cup G_p} \{\langle p, o_i \rangle \mid i \in [\![n_\sigma]\!]\} \cup \bigcup_{\xi \succeq 0 \in G_n} \{\langle \xi, v \rangle \mid v \in V(g)\}$,
- categorical node features $\mathbf{F}_{\text{cat}} : \mathbf{V} \to \Sigma_{\text{V}}$ with $\mathbf{F}_{\text{cat}}(u) =$

$$\begin{cases} \text{OBJ}(u) & \text{if } u \in \mathcal{O} \\ \text{FUNC}(u) & \text{if } u \in X_n(s_0) \\ (\text{COMP}(u), \text{ACH}(u)) & \text{if } u \in G_n \end{cases} \begin{cases} (\text{PRED}(u), \texttt{achieved_propositional_goal}) & \text{if } u \in X_p(s_0) \cap G_p \\ (\text{PRED}(u), \texttt{unachieved_propositional_goal}) & \text{if } u \in G_p \setminus X_p(s_0) \\ (\text{PRED}(u), \texttt{achieved_propositional_nongoal}) & \text{if } u \in X_p(s_0) \setminus G_p \end{cases}$$

  where $\text{OBJ}(u) = u$ if u is a constant object and `object` otherwise, $\text{PRED}(u)$/$\text{FUNC}(u)$ returns the predicate/function symbol from which a proposition/fluent was instantiated from, $\text{COMP}(u) \in \{\geq, >, =\}$ encodes the comparator type of the numeric goal condition u, and $\text{ACH}(u) \in \{\texttt{unachieved_numeric_goal}, \texttt{achieved_numeric_goal}\}$ encodes whether s_0 satisfies u,
- continuous node features $\mathbf{F}_{\text{con}} : \mathbf{V} \to \mathbb{R}$ where $\mathbf{F}_{\text{con}}(u) = [u]^{s_0}$ for nodes $u \in X_n(s_0)$, $\mathbf{F}_{\text{con}}(u) = [\xi]^{s_0}$ for nodes $u = \xi \succeq 0 \in G_n$ with $[\xi]^{s_0} \not\succeq 0$, and $\mathbf{F}_{\text{con}}(u) = 0$ otherwise, and
- edge labels $\mathbf{L} : \mathbf{E} \to \Sigma_{\text{E}}$ where for edges of the form $e = \langle p, o_i \rangle$, we have $\mathbf{L}(e) = i$, and otherwise for edges $e = \langle \xi, v \rangle$, we have $\mathbf{L}(e) = 0$.

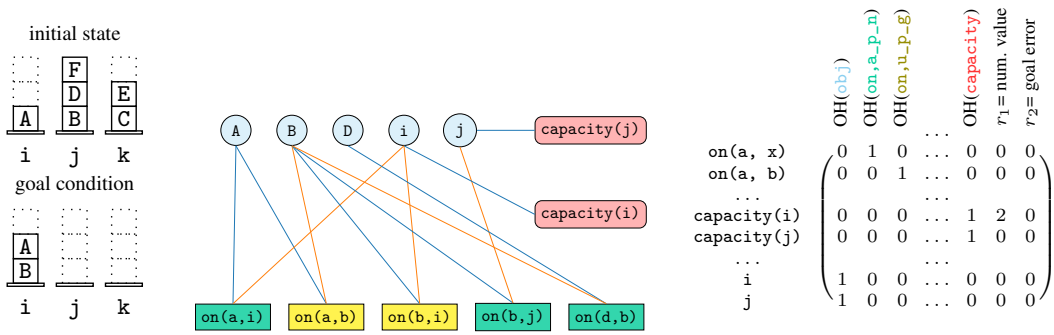


Figure 2: An example ccBlocksworld task where each base has capacity 3 (left), a subgraph of its ILG representation (middle), and the matrix representation of the node features of the ILG (right).

In general, given a domain with predicate and function symbols Σ_p and Σ_n, we have that there are $|\Sigma_{\mathrm{V}}| = 5 + 3\,|\Sigma_p| + |\Sigma_n| + |\text{constant_objects}|$ categorical node features representing the semantics of a node. Continuous node features indicate the value of numeric variables and the error of the expression in s_0 of unachieved numeric goals, and are set to zero for any other node.

3.2 The CCWL algorithm for numeric planning

The WL algorithm [WL68] has been adapted to computing features for graphs with categorical node attributes by [SSVL+11]. A variant of the WL algorithm for graphs with continuous node attributes has been proposed by [TGL+19] for the purpose of computing kernels with the Wasserstein distance between graph embeddings. However, the graph embeddings themselves are not invariant to the order of graphs in the nodes. Furthermore, from [CTT24b], non-linear kernels result in poorer generalisation compared to linear models in the context of L4P due to overfitting to the range of training targets. Morris et al. [MKKM16] constructed kernels for continuous node attributes by hashing Euclidean embeddings into categorical features but such a method loses the semantic meaning of numbers. Thus, we propose a new variant of the WL algorithm for graphs with both categorical and continuous node attributes for generating graph embeddings (CCWL algorithm). This algorithm is summarised in Alg. 1 and also depicted in Fig. 3.

Algorithm 1: CCWL algorithm

Data: A graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}_{\text{cat}}, \mathbf{F}_{\text{con}}, \mathbf{L} \rangle$ with continuous and categorical attributes, a deterministic and injective HASH function, allowed colours $\mathbf{C} = [\![|\mathbf{C}|]\!]$, a pooling function POOL, and number of CCWL iterations L.

Result: Feature vector of size $\mathbb{R}^{(1+d)|\mathbf{C}|}$.

1 $\kappa^0(v) \leftarrow \mathbf{F}_{\text{cat}}(v), \forall v \in \mathbf{V}$

2 **for** $j \in [\![L]\!]$ **do for** $v \in \mathbf{V}$ **do**

3 $\quad \kappa^j(v) \leftarrow \text{HASH}\left(\kappa^{j-1}(v), \bigcup_{\iota \in \Sigma_{\mathrm{E}}} \{(\kappa^{j-1}(u), \iota) \mid u \in \mathbf{N}_\iota(v)\}\right)$

4 $\mathbf{M} \leftarrow \bigcup_{j \in \{0\} \cup [\![L]\!]} \{\!\{\kappa^j(v) \mid v \in \mathbf{V}\}\!\}$

5 $\vec{v}_{\text{cat}} \leftarrow \left[\text{COUNT}(\mathbf{M}, c_1), \ldots, \text{COUNT}(\mathbf{M}, c_{|\mathbf{C}|})\right]$

6 $\mathbf{S}_i = \left\{v \mid v \in \mathbf{V} \text{ s.t. } \exists j \in \{0\} \cup [\![L]\!], \kappa^j(v) = c_i\right\}, \forall i \in \mathbf{C}$

7 $\vec{v}_{\text{con}} \leftarrow [\text{con}(1) \,\|\, \ldots \,\|\, \text{con}(|\mathbf{C}|)],\ \text{con}(i) = \text{POOL}_{v \in \mathbf{S}_i}(\mathbf{F}_{\text{con}}(v))$

8 **return** $\vec{v}_{\text{cat}} \,\|\, \vec{v}_{\text{con}}$

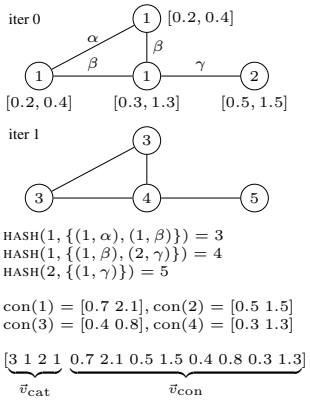


Figure 3: CCWL with one iteration, POOL $= \sum$, and $\mathbf{C} = [4]$.

Lines 1–3 of Alg. 1 are the original steps of the WL algorithm for generating graph embeddings by iteratively refining categorical node features, which we call colours, with two differences. Firstly, we replaced the multi-set with a set in the input of the hashing function. This is because in planning, unseen colours arise from graphs with increasing degrees which occur for out-of-distribution testing problems of increasing size. This problem is limited by relaxing the hash input with a set, which trades expressivity for generalisation. Secondly, we make use of edge labels in the hashing function.

Lines 4–5 collect the counts of allowed colours $\mathbf{C}$ seen during the main loop of the algorithm to generate the categorical feature vector in the form of a histogram. We assume by relabelling colours that $\mathbf{C} = [\![|\mathbf{C}|]\!]$. Lines 6–7 generate features from pooling the continuous attributes from different

groups of nodes. More specifically, for each colour $c \in \mathbf{C}$, we find the set of nodes which have been assigned the colour c some time during the refinement process and pool the continuous attributes of these nodes. Thus, we have $|\mathbf{C}|$ pooled continuous feature vectors which we concatenate together. We note that this pooling and concatenation process is invariant to the order of nodes in a graph in contrast to the intermediate graph embeddings generated for Wasserstein WL graph kernels by Togninalli et al. [TGL+19]. The algorithm returns the concatenation of the categorical and continuous feature vectors as the final feature vector output for the graph in Line 8. We note that $d = 1$ when running CCWL on the νILG representation of a numeric planning task.

We note that a drawback of the algorithm is that continuous attributes are not refined directly. This could be done by introducing one or more aggregation functions as parameters to the algorithm and refining continuous attributes by concatenating the aggregations of their neighbouring attributes. However, this method introduces an increase in the size of the continuous feature vector exponential in the number of layers, with base equal to the number of aggregation functions chosen. Moreover, we noted from informal experiments that this method led to overfitting of models to a large number of blended continuous features that do not have an obvious relation to the learning target.

Assuming a constant time hashing function, the complexity of the CCWL algorithm is $O(nL(\delta + d))$ where $n = |V|$ of the input graph, $\delta = \max_{u \in \mathbf{V}} \sum_{\iota \in \mathbf{L}} |\mathbf{N}_\iota(u)|$ is the degree of the graph, d is the dimension of the continuous node attributes, and L is the number of layers. The main computation comes from Line 3 which is performed nL times and the hashing function takes an input of size δ. Collecting the categorical feature vector takes the same time, while collecting the continuous feature vector takes $O(nLd)$ time. For reasonably sized $d \lesssim \delta$, as in the case of νILG where $d = 1$, this is the same complexity as the original WL algorithm for generating graph features, which is $O(nL\delta)$.

4 Relational neural networks for numeric planning

Deep learning architectures such as graph neural networks (GNNs) [SGT+09, GSR+17] benefit in generating latent representations automatically with backpropagation when trained end-to-end [LBH15]. GNNs also benefit from being able to train and evaluate on arbitrary sized graphs. However, it is generally understood that the expressive power of GNNs is limited by the WL-algorithm and counting logics with two variables [XHLJ19, BKM+20]. This result translates to the impossibility result of GNNs not being able to learn features that can work well for arbitrary planning domains [SBG22, CTT24a]. Nevertheless, their application to numeric planning tasks, in which both logical and numeric reasoning is required, is less well understood. Thus, we still propose GNNs as an additional baseline for L4NP and empirically evaluate their performance for numeric planning in Sec. 6.

For our GNN architecture, we perform a transformation on the node features of the νILG from 3.1 as input for GNNs that can handle edge labels. More specifically, given a νILG $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}_{\text{cat}} : \mathbf{V} \to \Sigma_{\text{V}}, \mathbf{F}_{\text{con}} \to \mathbb{R}, \mathbf{L} \rangle$, we construct a new graph $\mathbf{G}'$ with continuous node attributes $\mathbf{X} : \mathbf{V} \to \mathbb{R}^{|\Sigma_{\text{V}}|+2}$ defined by $\mathbf{X}(u) = \text{OH}(\mathbf{F}_{\text{cat}}(u)) \,\|\, [r_1, r_2]$, where $\text{OH}(\mathbf{F}_{\text{cat}}(u)) \in \{0,1\}^{|\Sigma_{\text{V}}|} \subseteq \mathbb{R}^{|\Sigma_{\text{V}}|}$ denotes a one-hot encoding of the categorical node feature of u, and r_1 denotes the numerical value of numeric variable nodes defined by $r_1 = [u]^{s_0}$ if $u \in X_n(s_0)$ and $r_1 = 0$ otherwise, and r_2 denotes the goal error for numeric goal nodes defined by $r_2 = [u]^{s_0}$ if $u \in G_n$ and $r_2 = 0$ otherwise. We denote the νILG for GNNs by $\langle \mathbf{V}, \mathbf{E}, \mathbf{X}, \mathbf{L} \rangle$ with notation for categorical features removed. Thus, we can use this graph encoding of numeric planning tasks as input into any downstream GNN that can handle edge labels or features.

Fig. 2 illustrates the node feature matrix representation of the νILG encoding of a ccBlocksworld task for input to a GNN. Each row represents a node in the graph, with columns representing the semantics of the node as well as the value of the numeric variables in the state and error of numeric goal nodes. We note however, that the ccBlocksworld example does not have any numeric goals and thus the last column is zero for all entries.

5 Optimisation formulations for learning heuristic functions

In this section, we describe two optimisation methods used for learning heuristic functions from training data, namely by minimising cost-to-go estimate error and ranking estimate error. Fig. 4 illustrates examples of learned heuristic functions on states of a planning task when trained to zero loss with both the cost-to-go and ranking formulations. We assume that training data for our models consist of a set of numeric planning tasks $\Pi_1, \ldots, \Pi_n$ with corresponding optimal plans $\pi_1, \ldots, \pi_n$.

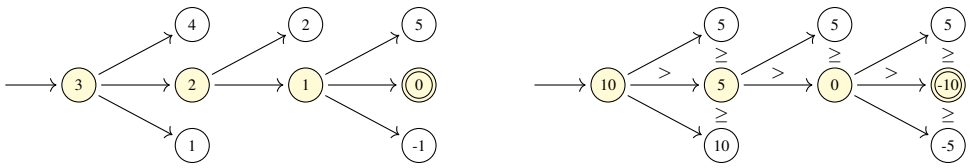


Figure 4: Examples of heuristic functions that achieve 0 loss when optimising cost-to-go (left) and ranking (right) on an optimal plan. Coloured nodes indicate states on the optimal plan with the goal state indicated by a double circle. Edges indicate successors of a node. A cost-to-go heuristic can achieve 0 loss on the plan trace but may not generalise correctly to state successors. A ranking heuristic does not need represent correct cost-to-go values and only need to satisfy ranking constraints. GBFS will return a plan in linear time for the ranking heuristic here but not for the cost-to-go heuristic.

We note that a numeric planning task can offer more training data by generating additional tasks and plans from different states in the state space of the task. Each plan is denoted $\pi_i = a_1^{(i)}, \ldots, a_{|\pi_i|}^{(i)}$ with plan trace $s_0^{(i)}, s_1^{(i)}, \ldots, s_{|\pi_i|}^{(i)}$. Each state s in a plan trace induces a new planning task by replacing s_0 with s of the original task, with which we can construct graph or vector representations from our aforementioned models.

Heuristic functions from cost-to-go estimates. We can use planning tasks and corresponding optimal plans as training data for learning a heuristic function representing the estimated cost-to-go to the plan. Each task and corresponding plan π_i contributes training data $s_j^{(i)}$ with targets $h^*(s_j^{(i)})$ for each state $s_j^{(i)}$ in the plan trace of π_i. Then given an estimator $\mathcal{H}$, we may try to find weights that minimise the mean squared error (MSE) $L(\theta) = \frac{1}{N}\sum_{i=1}^{n}\sum_{j=0}^{|\pi_i|}\big(h^*(s_j^{(i)}) - \mathcal{H}_\theta(s_j^{(i)})\big)^2$ where N is the normalisation constant and $\mathcal{H}_\theta$ denotes the estimator with weights θ.

Heuristic functions from ranking estimates. The MSE loss is a simple but naive method for training a heuristic function. Various researchers have instead proposed to use the concept of ranking to learn heuristic functions [GKL16, CEKP23, HTT+24]. However, a drawback of the formulation of the ranking optimisation of previous works is that a state in a plan trace is marked as strictly better as its siblings when it could be the case that the siblings may have the same h^* value. Furthermore, the formulation in [CEKP23] scales quadratically in the plan trace. We offer a novel ranking optimisation criterion that (1) fixes the problem of siblings being misclassified and (2) also results in a sparse model. We also offer a corresponding differentiable loss function for use with any end-to-end model.

Our first ranking formulation requires solving an LP as the optimisation problem, similarly to [FCGP19] but only using states from the plan trace, whereas the latter work uses states from the entire state space of the problem. It can also be viewed as an LP encoding of the formulation by Garrett et al. [GKL16] but fixing the problem of misrepresented siblings and learning sparse weights. Let $\textsc{succs}(s)$ denote the set of successors of the state s in a planning task by applying all applicable actions at s. Hence the set of siblings of state $s_j^{(i)}$ in Π_i's state space is $\textsc{siblings}(s_j^{(i)}) = \textsc{succs}(s_{j-1}^{(i)}) \setminus \{s_j^{(i)}\}$. Let φ denote our feature generation function with $\varphi(s) \in \mathbb{R}^d$ for any state s. Then we can define our optimisation problem as a linear program defined by

$$\min_{\mathbf{w},\mathbf{z}} \sum_{i,j,k} \mathbf{z}_{i,j,k} + \|\mathbf{w}\|_1 \quad \text{s.t.} \quad \mathbf{z}_{i,j,k} \geq 0, \forall i,j,k \tag{1}$$

$$\mathbf{w}^\top(\varphi(s_{j-1}^{(i)}) - \varphi(s_j^{(i)})) \geq \text{cost}(a_j^{(i)}) - \mathbf{z}_{i,j,0} \qquad \forall i \in [\![n]\!], j \in [\![|\pi_i|]\!]$$

$$\mathbf{w}^\top(\varphi(s_\alpha) - \varphi(s_j^{(i)})) \geq -\mathbf{z}_{i,j,\alpha} \qquad \forall i \in [\![n]\!], j \in [\![|\pi_i|]\!], s_\alpha \in \textsc{siblings}(s_j^{(i)}).$$

The vector $\mathbf{w}$ represents the weights our linear model aims to learn, and the nonnegative slack variables $\mathbf{z}$ model the soft inequality constraints representing the ranking of states. The optimisation problem is to minimise the the slack variables corresponding to the error of the constraints, and the ℓ_1 norm of the weights to encourage sparsity.

We next offer a differentiable loss function version of the previous model which we can use as a fair comparison when combining it with our GNN architecture in Sec. 4 compared to combining (1) with features generated in Sec. 3. The idea is to replace the slack variables with the $\max$ function:

$$L(\theta) = \sum_{i,j}\Big(\max\big(0, \mathcal{H}_\theta(s_j^{(i)}) - \mathcal{H}_\theta(s_{j-1}^{(i)}) + c(a_j^{(i)})\big) + \sum_{s_\alpha \in \textsc{siblings}(s_j^{(i)})} \max\Big(0, \mathcal{H}_\theta(s_j^{(i)}) - \mathcal{H}_\theta(s_\alpha)\Big)\Big). \tag{2}$$

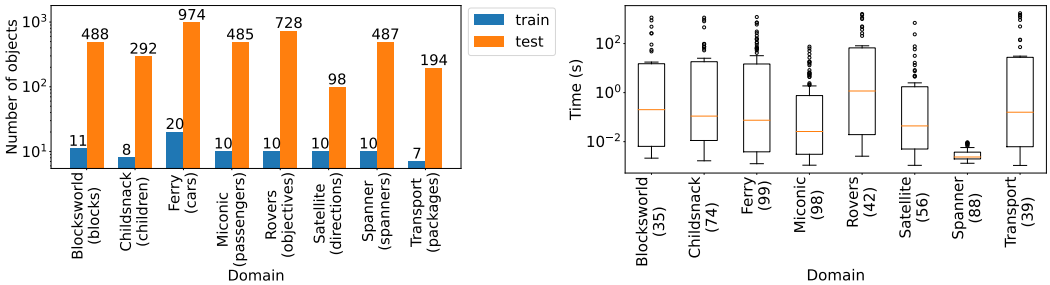


Figure 5: Number of objects in training and testing problems (left) and distributions of training data generation time with number of training problems (right) per domain. Note the log scales.

6 Experiments

6.1 Numeric planning benchmarks

We take 8 domains out of 10 domains from the International Planning Competition 2023 Learning Track (IPC-LT) [SSA23] and either convert them to equivalent numeric formulations, or introduce numeric variables to model extra features such as capacity constraints. The two domains from the IPC-LT that we do not convert into numeric domains are Floortile and Sokoban which do not have any benefit from compilation to a numeric representation nor exhibit any interesting features that can be modelled with numeric variables. The domains we considered from the IPC-LT are summarised in Fig. 5 alongside the sizes of training and testing tasks, and time to generate training data. Each domain consists of 90 testing problems and at most 99 small training problems for which the median time for generating an optimal training plan is less than a second and a few outliers taking more than a minute. We refer to the appendix for further details on the domains.

6.2 Experimental setup

Training. As discussed in Sec. 5, we only consider optimal plans from small problems as training data. We compute them with the Numeric Fast Downward planner [AN17] using A* search and the admissible h^{LMCUT} heuristic [KSP+22], with a 30 minute timeout and 8GB main memory.

We consider 4 model configurations. Firstly, we use CCWL features from Sec. 3 with Support Vector Regression and the linear dot product kernel to learn a linear model for cost-to-go estimation ($h^{\text{CCWLF}}_{\text{cost}}$). Next, we use CCWL features in optimisation problem in (1) with CPLEX version 22.11 and a timeout of 600 seconds for ranking estimation ($h^{\text{CCWLF}}_{\text{rank}}$). Both $h^{\text{CCWLF}}_{\text{cost}}$ and $h^{\text{CCWLF}}_{\text{rank}}$ models have allowed colours $\mathbf{C}$ in Alg. 1 given by all the refined colours seen during training. We also have cost-to-go ($h^{\text{GNN}}_{\text{cost}}$) and ranking ($h^{\text{GNN}}_{\text{rank}}$) estimation models using GNNs operating on νILG representations of planning tasks and optimised with the MSE loss function and (2), respectively. For the backbone GNN, we use a Relational Graph Convolution Network [SKB+18] but replacing the mean aggregation function with the element-wise max operator in the message-passing update step: $\mathbf{h}_u^{(l+1)} = \sigma(\mathbf{W}_0\mathbf{h}_u^{(l)} + \sum_{\iota\in\Sigma_{\text{E}}} \max_{v\in\mathbf{N}_\iota(u)} \mathbf{W}_\iota^{(l)}\mathbf{h}_v^{(l)})$, where l denotes the GNN layer, σ is implemented with the leaky ReLU function, and $\mathbf{W}_0$ and $\mathbf{W}_\iota^{(l)}$ are learnable weight matrices. Each GNN has a hidden dimension of 64, and is trained with the Adam optimiser [KB15] with an initial learning rate of 10^{-3} and batch size of 16. A scheduler reduces the training loss by a factor of 10 if loss does not improve after 10 epochs. Training then terminates if the learning rate falls below 10^{-5}. Let L denote the iterations hyperparameter for CCWL models and number of layers for GNN models.

Evaluation. We consider several numeric planners as baselines for benchmarking the effectiveness of learning. We first include h^{LMCUT} as the only optimal planner baseline as it is also the training data generator but solves a more difficult problem of optimal planning compared to satisficing planning. We consider the Metric-FF planner (M-FF) [Hof03], and the h^{ADD}, h^{MRP}, h^{MRP}+hj and M$(3h\|3n)$ configurations in the ENHSP planner [SHTR20, SSSG20, CT24]. We have that h^{ADD} and h^{MRP} are planners that perform GBFS with a single heuristic only, while h^{MRP}+hj and M$(3h\|3n)$ use additional techniques (macro actions, multiple queues, and novelty heuristics) to boost planning performance. Our CCWL and GNN models are all used in single-queue GBFS with the learned heuristic function, with Numeric Fast Downward as the backend search implementation. All baselines and models are run on a single Intel Xeon Platinum 8268 (2.90 GHz) core with a 5 minute timeout for search and

Table 1: Coverage of numeric domain-independent, the new learning planners ($h^{\text{GNN}}_{\text{cost}}$, $h^{\text{GNN}}_{\text{rank}}$, $h^{\text{CCWLF}}_{\text{cost}}$, $h^{\text{CCWLF}}_{\text{rank}}$) with $L = 1$, and the best learner configuration score on each domain (Best Learner). Higher values are better (↑), with the top three scores in each row except the rightmost entry indicated by the cell colour intensity. All planner configurations except $h^{\text{LMCUT}}_{\text{opt}}$ are satisficing planners.

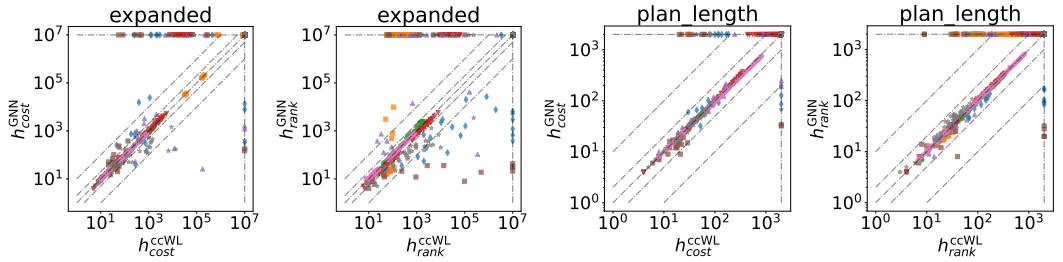
	Planner Baselines						Learners (**new**)				
					GBFS + heuristic						
Domain	$h^{\text{LMCUT}}_{\text{opt}}$	h^{MRP}+hj	M-FF	M($3h\|3n$)	h^{MRP}	h^{ADD}	$h^{\text{GNN}}_{\text{cost}}$	$h^{\text{GNN}}_{\text{rank}}$	$h^{\text{CCWLF}}_{\text{cost}}$	$h^{\text{CCWLF}}_{\text{rank}}$	Best Learner
Blocksworld	6	18	9	23	19	16	18	**24**	22	19	**29**
Childsnack	20	49	14	53	25	20	17	22	22	**90**	**90**
Ferry	33	60	60	57	60	57	60	60	70	**71**	**73**
Miconic	30	68	65	61	64	51	63	64	**90**	**90**	**90**
Rovers	10	**34**	17	30	18	15	18	14	22	23	30
Satellite	18	**38**	24	29	21	23	19	14	23	16	26
Spanner	30	6	35	76	42	42	**90**	**90**	**90**	**90**	**90**
Transport	12	**55**	49	40	32	40	34	38	40	46	48
Σ	159	328	273	369	281	264	319	326	379	**445**	**476**

8GB of main memory. Tab. 1 summarises the coverage results of all considered planners on the benchmarks, with more details provided in the appendix.

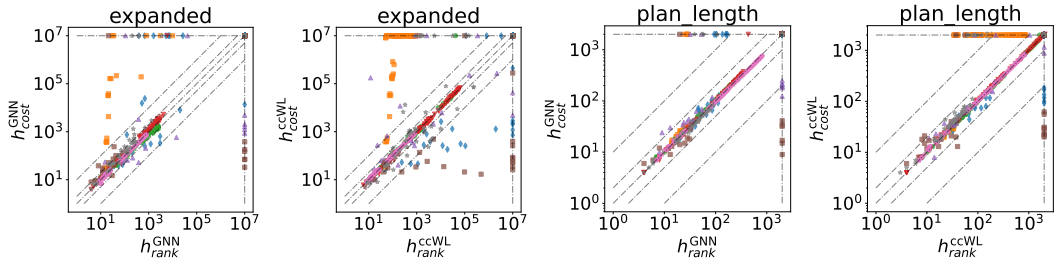
How do learning approaches compare to domain-independent numeric planners? From Tab. 1, we note that our best performing model with $L = 1$ is $h^{\text{CCWLF}}_{\text{rank}}$ and outperforms all domain-independent planners for satisficing planning on 4 out of 8 domains. Increasing L to 2 brings $h^{\text{CCWLF}}_{\text{rank}}$ to achieve the best coverage on Blocksworld. The domains which learners fall behind on are Rovers, Satellite and Transport, even when taking the best hyperparameter configuration. The former two are difficult as they require features more expressive than those generated by graph learning approaches to capture the semantics of reasoning required to solve the problems [SBG22], while the latter requires path finding which is not possible for learners with finite receptive fields [TTTX20]. These results hold for classical planning and thus also for our extension to numeric planning. Generally the best performing planner on a domain expands fewer nodes than the other planners. With regards to plan length, $h^{\text{CCWLF}}_{\text{rank}}$ performs best for Blocksworld but is marginally worse than the best of the domain-independent numeric planners for Rovers, Satellite and Spanner.

How do CCWL models compare to GNN models? From Tab. 1, we see that the CCWL models always have similar or better performance than the corresponding GNN models, when comparing cost-to-go and ranking estimates. The performance of a planners which use GBFS and a heuristic depend on the heuristic evaluation speed, in which more search can be done in the time limit, or the quality of the heuristic, in which search can be more informed. Fig. 8 in the appendix shows that GNN are generally at least an order of magnitude slower than CCWL models for heuristic evaluation due to performing intensive matrix operations. We note that GNN models are evaluated on CPUs and could be sped up with access GPUs. Fig. 6a illustrates the number of node expansions of GNN and CCWL models and we note that there is no clear winner between the two approaches across all domains, with the exception of $h^{\text{CCWLF}}_{\text{rank}}$ generalising perfectly on Childsnack where other models could not. Thus, we can conclude with respect to planning efficiency that CCWL models generally outperform their GNN counterparts due to faster heuristic evaluation speeds, while generally both models have similar generalisation performance.

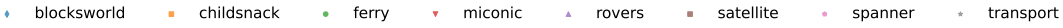
How do ranking models compare to cost-to-go models? From Tab. 1, ranking models outperform cost-to-go models in total coverage. However, their performance is incomparable across domains even when looking at Fig. 6b with the exception of CCWL being able to achieve perfect performance on Childsnack. Nevertheless, on 8 domain-model pairs for $L = 1$, ranking models achieve strictly better coverage, while the converse is only true for 4 domain-model pairs. This suggests a bias favouring ranking models which can be explained by their advantages covered in Sec. 5, namely that



(a) GNN ($h^{\text{GNN}}_{\text{cost}}$, $h^{\text{GNN}}_{\text{rank}}$) vs. CCWL models ($h^{\text{CCWLF}}_{\text{cost}}$, $h^{\text{CCWLF}}_{\text{rank}}$).



(b) Cost-to-go ($h^{\text{GNN}}_{\text{cost}}$, $h^{\text{CCWLF}}_{\text{cost}}$) vs. ranking heuristic models ($h^{\text{GNN}}_{\text{rank}}$, $h^{\text{CCWLF}}_{\text{rank}}$).



(c) Plot legend.

Figure 6: Plot comparisons of expanded nodes and plan length of selected pairs of models with $L = 1$. A point (x, y) represents the metric of the models indicated on the x and y axis on the domain. Points on the top left (resp. bottom right) triangle favour the model on the x-axis (resp. y-axis).

they implicitly use more training data by considering successor states of plan trace states, and have a larger solution space as they are not restricted to predicting an exact value.

What is the effect of number of iterations for CCWL models and layers for GNNs? The hyperparameter L, which denotes the number of iterations (resp. layers) for CCWL (resp. GNN) models, generally plays an important role in planning performance. This is because increasing L improves model expressivity and reasoning capabilities, but comes at the cost of heuristic evaluation time and increased possibility of overfitting to the training data. From Tab. 2 in the appendix, we note that surprisingly for most domains and models $L = 0$ or $L = 1$ provides the best coverage, while increasing L rarely improves coverage. This suggests that heuristic evaluation time plays an important role in planning performance for domains that cannot be solved with the learner's expressivity.

7 Limitations

The setup of our work is limited to the assumption that the problems being solved can be explicitly represented in a symbolic language such as PDDL. The assumption of the existence of PDDL encodings of planning problems allows us to generate training data quickly with domain-independent numeric planners for supervised training. Furthermore, experiments and theoretical insights also show that our proposed techniques have room for improvement as there are still classes of numeric planning tasks with which our models cannot learn and generalise well in.

8 Conclusion

We have proposed a new graph embedding algorithm, the CCWL algorithm, and optimisation criterion for learning heuristic functions for numeric planning. Planning tasks are encoded as Numeric Instance Learning Graphs (νILG) on which we run our CCWL algorithm for generating features. Our numeric planning features are interpretable and efficient to generate. Experimental results show the effectiveness of our approach by achieving competitive performance over both deep learning architectures and domain-independent numeric planners. Furthermore, we have identified future work by improving the expressivity of our algorithms for capturing more complex numeric domains. Lastly, one can learn forms of domain knowledge different from heuristic functions with our new numeric planning features and graph representations such as policies [WT24], portfolios [MFH+20] and detecting relevant objects [SCC+21].

Acknowledgements

The authors thank the reviewers and Giacomo Rosa for the helpful comments and suggestions. The computing resources for the project was partially supported by the Australian Government through the National Computational Infrastructure (NCI) under the ANU Startup Scheme. ST was supported by the Australian Research Council grant DP220103815, by the Artificial and Natural Intelligence Toulouse Institute (ANITI) under the grant agreement ANR-19-PI3A-0004, and by the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No. 101070149.

References

[ACSJ22] Javier Segovia Aguas, Sergio Jiménez Celorrio, Laura Sebastiá, and Anders Jonsson. Scaling-up generalized planning as heuristic search with landmarks. In *SOCS*, 2022.

[AJJ18] Javier Segovia Aguas, Sergio Jiménez, and Anders Jonsson. Computing hierarchical finite state controllers with classical planning. *J. Artif. Intell. Res.*, 62, 2018.

[AJJ21] Javier Segovia Aguas, Sergio Jiménez, and Anders Jonsson. Generalized planning as heuristic search. In *ICAPS*, 2021.

[AN17] Johannes Aldinger and Bernhard Nebel. Interval based relaxation heuristics for numeric planning with action costs. In *KI*, 2017.

[APK24] Forest Agostinelli, Rojina Panta, and Vedant Khandelwal. Specifying goals to deep neural networks with answer set programming. In *ICAPS*, 2024.

[BG18] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In *IJCAI*, 2018.

[BG20] Blai Bonet and Hector Geffner. Qualitative numeric planning: Reductions and complexity. *J. Artif. Intell. Res.*, 69, 2020.

[BKM+20] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

[BPG09] Blai Bonet, Héctor Palacios, and Hector Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *ICAPS*, 2009.

[BPG10] Blai Bonet, Héctor Palacios, and Hector Geffner. Automatic derivation of finite-state machines for behavior control. In *AAAI*, 2010.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69, 1994.

[CAJ19] Sergio Jiménez Celorrio, Javier Segovia Aguas, and Anders Jonsson. A review of generalized planning. *Knowl. Eng. Rev.*, 34, 2019.

[CCFL13] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *J. Artif. Intell. Res.*, 46, 2013.

[CEKP23] Leah Chrestien, Stefan Edelkamp, Antonín Komenda, and Tomás Pevný. Optimize planning heuristics to rank, not to estimate cost-to-goal. In *NeurIPS*, 2023.

[CHŠ24] Dillon Z. Chen, Rostislav Horčík, and Gustav Šír. Deep learning for generalised planning with background knowledge. *CoRR*, abs/2410.07923, 2024.

[CT24] Dillon Z. Chen and Sylvie Thiébaux. Novelty heuristics, multi-queue search, and portfolios for numeric planning. In *SOCS*, 2024.

[CTT24a] Dillon Z. Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *AAAI*, 2024.

[CTT24b] Dillon Z. Chen, Felipe Trevizan, and Sylvie Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *ICAPS*, 2024.

[FCGP19] Guillem Francès, Augusto B. Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *IJCAI*, 2019.

[FGT+22] Patrick Ferber, Florian Geißer, Felipe Trevizan, Malte Helmert, and Jörg Hoffmann. Neural network heuristic functions for classical planning: Bootstrapping and comparison to other methods. In *ICAPS*, 2022.

[FL03] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20, 2003.

[GAC+22] Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *ICAPS*, 2022.

[Gef18] Hector Geffner. Model-free, model-based, and general intelligence. In *IJCAI*, 2018.

[GKL16] Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to rank for synthesizing planning heuristics. In *IJCAI*, 2016.

[GRH24] Claudia Grundke, Gabriele Röger, and Malte Helmert. Formal representations of classical planning domains. In *ICAPS*. AAAI Press, 2024.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[GT04] Charles Gretton and Sylvie Thiébaux. Exploiting first-order regression in inductive policy selection. In *UAI*. AUAI Press, 2004.

[Hel02] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. In *AIPS*, 2002.

[HG11] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, 2011.

[HG13] Yuxiao Hu and Giuseppe De Giacomo. A generic technique for synthesizing bounded finite-state controllers. In *ICAPS*, 2013.

[HIR+18] Patrik Haslum, Franc Ivankovic, Miquel Ramírez, Dan Gordon, Sylvie Thiébaux, Vikas Shivashankar, and Dana S. Nau. Extending classical planning with state constraints: Heuristics and search for optimal planning. *J. Artif. Intell. Res.*, 62, 2018.

[Hof03] Jörg Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res.*, 20, 2003.

[HTT+24] Mingyu Hao, Felipe Trevizan, Sylvie Thiébaux, Patrick Ferber, and Jörg Hoffmann. Guiding GBFS through learned pairwise rankings. In *IJCAI*, 2024.

[IKVM22] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Intell. Res.*, 73, 2022.

[IM17] Leon Illanes and Sheila A. McIlraith. Numeric planning via abstraction and policy guided search. In *IJCAI*, 2017.

[IM19] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *AAAI*, 2019.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Kha99] Roni Khardon. Learning action strategies for planning domains. *Artif. Intell.*, 113(1-2), 1999.

[KS21] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*, 2021.

[KSB23] Ryo Kuroiwa, Alexander Shleyfman, and J. Christopher Beck. Extracting and exploiting bounds of numeric variables for optimal linear numeric planning. In *ECAI*, 2023.

[KSP+22] Ryo Kuroiwa, Alexander Shleyfman, Chiara Piacentini, Margarita P. Castro, and J. Christopher Beck. The lm-cut heuristic family for optimal numeric planning with simple conditions. *J. Artif. Intell. Res.*, 75, 2022.

[KVS+23] Ken Kansky, Skanda Vaidyanath, Scott Swingle, Xinghua Lou, Miguel Lázaro-Gredilla, and Dileep George. Pushworld: A benchmark for manipulation planning with tools and movable obstacles. *CoRR*, abs/2301.10289, 2023.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nat.*, 521(7553), 2015.

[LCF+22] Xiaoyou Lin, Qingliang Chen, Liangda Fang, Quanlong Guan, Weiqi Luo, and Kaile Su. Generalized linear integer numeric planning. In *ICAPS*, 2022.

[LCZ+21] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Scalable rail planning and replanning: Winning the 2020 flatland challenge. In *ICAPS*, 2021.

[LPM23] Xiaotian Liu, Héctor Palacios, and Christian Muise. Egocentric planning for scalable embodied task achievement. In *NeurIPS*, 2023.

[LSS+22] Leonardo Lamanna, Luciano Serafini, Alessandro Saetti, Alfonso Gerevini, and Paolo Traverso. Online grounding of symbolic planning domains in unknown environments. In *KR*, 2022.

[MFH+20] Tengfei Ma, Patrick Ferber, Siyu Huo, Jie Chen, and Michael Katz. Online planner selection with graph neural networks and adaptive scheduling. In *AAAI*, 2020.

[MKKM16] Christopher Morris, Nils M. Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *ICDM*, 2016.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540), 2015.

[MLTK23] Jiayuan Mao, Tomás Lozano-Pérez, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. What planning problems can A relational neural network solve? In *NeurIPS*, 2023.

[MV21] Andrea Micheli and Alessandro Valentini. Synthesis of search heuristics for temporal planning via reinforcement learning. In *AAAI*. AAAI Press, 2021.

[RTG+24] Nicholas Rossetti, Massimiliano Tummolo, Alfonso Emilio Gerevini, Luca Putelli, Ivan Serina, Mattia Chiari, and Matteo Olivato. Learning general policies for planning through GPT models. In *ICAPS*, 2024.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[SBG22] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS*, 2022.

[SBG23] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *KR*, 2023.

[SCC+21] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *AAAI*, 2021.

[SCK+23] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. In *AAAI*, 2023.

[SDS+24] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. In *AAAI*, 2024.

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20, 2009.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587), 2016.

[SHTR20] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Subgoaling techniques for satisficing and optimal numeric planning. *J. Artif. Intell. Res.*, 68, 2020.

[SIZ08] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *AAAI*, 2008.

[SKB+18] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, volume 10843, 2018.

[SKB23] Alexander Shleyfman, Ryo Kuroiwa, and J. Christopher Beck. Symmetry detection and breaking in linear cost-optimal numeric planning. In *ICAPS*, 2023.

[Sri10] Siddharth Srivastava. *Foundations and Applications of Generalized Planning*. PhD thesis, University of Massachusetts Amherst, 2010.

[Sri23] Siddharth Srivastava. Hierarchical decompositions and termination analysis for generalized planning. *J. Artif. Intell. Res.*, 77, 2023.

[SSA23] Jendrik Seipp and Javier Segovia-Aguas. International planning competition 2023 - learning track. `https://ipc2023-learning.github.io/`, 2023.

[SSSG20] Enrico Scala, Alessandro Saetti, Ivan Serina, and Alfonso Emilio Gerevini. Search-guidance mechanisms for numeric planning through subgoaling relaxation. In *ICAPS*, 2020.

[SSVL+11] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12, 2011.

[STT20] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *ICAPS*, 2020.

[SZIG11] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *AAAI*, 2011.

[TGL+19] Matteo Togninalli, M. Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten M. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *NeurIPS*, 2019.

[TTTX20] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *JAIR*, 68, 2020.

[WL68] Boris Weisfeiler and A. A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2, 1968.

[WT24] Ryan X. Wang and Sylvie Thiébaux. Learning generalised policies for numeric planning. In *ICAPS*, 2024.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[ZYP+22] Yichi Zhang, Jianing Yang, Jiayi Pan, Shane Storks, Nikhil Devraj, Ziqiao Ma, Keunwoo Peter Yu, Yuwei Bao, and Joyce Chai. DANLI: deliberative agent for following natural language instructions. In *EMNLP*, 2022.

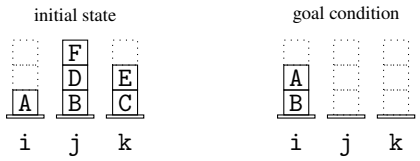


Figure 7: Left: initial state of a ccBlocksworld problem, where base `i`, `j`, and `k` each have a load limit of 3 blocks. Right: the goal condition where `A` is on top of `B` which is on top of `i`.

A More Details for the ccBlocksworld Example

We repeat the running ccBlocksworld example in Fig. 7. Listings 2 and 1 provide the explicit PDDL domain and problem encodings for the running ccBlocksworld example. An optimal plan for the problem is given as follows on the left, and an optimal plan without capacity constraints on the right.

```
 1.  (unstack f d j)
 2.  (stack f a i)
 3.  (unstack d b j)
 4.  (stack d f i)
 5.  (pickup b j)
 6.  (stack b e k)
 7.  (unstack d f i)
 8.  (putdown d j)
 9.  (unstack f a i)
10.  (stack f d j)
11.  (pickup a i)
12.  (stack a f j)
13.  (unstack b e k)
14.  (putdown b i)
15.  (unstack a f j)
16.  (stack a b i)
```

```
 1.  (unstack f d j)
 2.  (stack f e k)
 3.  (unstack d b j)
 4.  (stack d f k)
 5.  (pickup a i)
 6.  (stack a d k)
 7.  (pickup b j)
 8.  (putdown b i)
 9.  (unstack a d k)
10.  (stack a b i)
```

Listing 1: PDDL encoding for the ccBlocksworld problem in Fig. 7.

```
(define (problem running-example)
    (:domain ccblocksworld)
    (:objects
      a b c d e f - block
      i j k - base
    )
    (:init
        (arm_empty)
        (= (capacity i) 2)
        (= (capacity j) 0)
        (= (capacity k) 1)
        (clear a)
        (on a i)
        (above a i)
        (clear f)
        (on f d)
        (on d b)
        (on b j)
        (above f j)
        (above d j)
        (above b j)
        (clear e)
        (on e c)
        (on c k)
        (above e k)
        (above c k)
    )
    (:goal (and
        (clear a)
        (on a b)
        (on b i)
    ))
)
```

Listing 2: PDDL encoding for the ccBlocksworld domain.

```
(define (domain ccblocksworld)
    (:requirements :strips :typing :numeric-fluents)
    (:types
        block base - object
    )
    (:predicates
        (on         ?x - block      ?y - object)
        (above      ?x - block      ?y - base  )
        (clear      ?x - object                )
        (holding    ?x - block                 )
        (arm_empty                             )
    )
    (:functions
        (capacity   ?x - base                  )
    )
    (:action pickup
        :parameters (?block - block ?base - base)
        :precondition (and
            (on ?block ?base)
            (above ?block ?base)
            (clear ?block)
            (arm_empty))
        :effect (and
            (not (on ?block ?base))
            (not (above ?block ?base))
            (not (clear ?block))
            (clear ?base)
            (holding ?block)
            (not (arm_empty))
            (increase (capacity ?base) 1))
    )
    (:action putdown
        :parameters (?block - block ?base - base)
        :precondition (and
            (holding ?block)
            (clear ?base)
            (<= 1 (capacity ?base)))
        :effect (and
            (not (holding ?block))
            (not (clear ?base))
            (on ?block ?base)
            (above ?block ?base)
            (clear ?block)
            (arm_empty)
            (decrease (capacity ?base) 1))
    )
    (:action unstack
        :parameters (?block_a - block ?block_b - block ?base - base)
        :precondition (and
            (on ?block_a ?block_b)
            (above ?block_a ?base)
            (clear ?block_a)
            (arm_empty))
        :effect (and
            (not (on ?block_a ?block_b))
            (not (above ?block_a ?base))
            (not (clear ?block_a))
            (clear ?block_b)
            (holding ?block_a)
            (not (arm_empty))
            (increase (capacity ?base) 1))
    )
    (:action stack
        :parameters (?block_a - block ?block_b - block ?base - base)
        :precondition (and
            (holding ?block_a)
            (clear ?block_b)
            (above ?block_b ?base)
            (<= 1 (capacity ?base)))
        :effect (and
            (not (holding ?block_a))
            (not (clear ?block_b))
            (on ?block_a ?block_b)
            (above ?block_a ?base)
            (clear ?block_a)
            (arm_empty)
            (decrease (capacity ?base) 1))
    )
)
```

B Related Work

Two related fields to Learning For Planning (L4P) and Learning For Numeric Planning (L4NP) are Generalised Planning (GP) and Reinforcement Learning (RL). In the following subsections, we outline the main difference between L4P with the respective related fields as well as corresponding related work.

B.1 Generalised planning

GP consists of automatically characterising the solution of a (possibly infinite) set of planning tasks [Sri10, SIZ08]. The most common characterisations are action policies, but other characterisations also include finite state controllers [BPG09, BPG10, HG11, HG13, AJJ18], and programs with branching and looping [AJJ21, ACSJ22]. Logic programming approaches involving decision lists [Kha99, GT04] and Datalog programs [GRH24, CHŠ24] have also been used to characterise solutions for planning domains. We refer to articles [CAJ19] and [Sri23] for more detailed surveys of GP. The difference L4P and GP can be subtle given that there is a non-empty intersection between the two fields, and works in both fields generally aim to compute structures that solve problems from a given domain. The way we differentiate the two fields is that L4P follows generally follows traditional supervised learning approaches, whereas GP can be likened to performing program synthesis.

With regards to numeric planning, Srivastava et al. [SZIG11] introduced Qualitative Numeric Planning (QNP) which is a subset of numeric planning where numeric variables exhibit non-negative domains, and actions increase or decrease the value of numeric variables by indeterminate amounts. A solution for a QNP is a policy which can be used to represent solutions for sets of planning tasks. QNP has been shown to be equivalent to fully observable non-deterministic (FOND) planning [BG20] arising from the non-determinism of action effects, and the connection between FOND and GP has often shown itself when used to synthesise generalised policies [BG18, IM19]. Lin et al. [LCF$^+$22] studies GP for a more expressive class of numeric planning, by allowing for integer numeric variables and employing linear expressions in conditions and action effects. Their approach involves synthesising programs that allow for branching and looping. Lastly, νASNets [WT24] extends ASNets [TTTX20] in order to learn policies with a neural network architecture for planning.

B.2 Reinforcement Learning

RL is a learning paradigm for decision making that does not have access to a model and instead learns from rewards [SB98]. RL has achieved promising results in games when combined with deep learning [MKS$^+$15, SHM$^+$16]. A major difference between RL and L4P is that the former requires reasoning over dense reward functions, whereas the latter requires reasoning over logic [Gef18]. Nevertheless, there has been some preliminary work looking at the intersection of RL and planning. Reward machines [IKVM22] are a logical language used for specifying reward functions for RL problems, inspired by the declarative nature of the planning as modelling paradigm. RL has also been applied directly into planning tasks, as done by [MV21] for temporal planning. Rewards are mostly sparse, with 1 being reward for achieved goals, minor 10^{-5} rewards for achieved goal propositions, and no reward otherwise. Gehring et al. [GAC$^+$22] explored introducing denser reward functions to planning through domain-independent heuristics to allow for RL approaches. Supervised RL has also been used for learning planning policies [SBG23]. Nevertheless, the use cases for RL and planning are generally different, with RL being more suited for control tasks in continuous or dynamic environments such as in robotics, and planning being more suited for combinatorial tasks in discrete or abstract environments such as in logistics.

C Description of Benchmark Domains

C.1 Numeric (Capcity Constrained) Blocksworld

This domain was described in Sec. 2. A task from the domain consists of n blocks stacked on top of one another to form towers on top of b bases. Each base has a capacity of how many blocks it can support. The goal is to stack and unstack blocks to achieve a target tower configuration. The numeric component of this domain arises from modelling the capacity of bases. Training problems have $n \in [2, 11]$ blocks while testing problems have $n \in [5, 488]$ blocks.

C.2 Numeric Childsnack

A task from the domain consists of feeding c children with sandwiches in l locations, of which some are allergic to gluten. There are a finite amount of gluten-free (GF) and non-GF ingredients.

GF sandwiches can only be made from GF ingredients, whereas non-GF sandwiches can be made with any ingredients. Children allergic to gluten are only allowed to eat GF sandwiches while the remaining children can eat any type of sandwich. Thus, the problem has deadends because resources are finite and can be wasted. The goal is to make sandwiches and feed all the children satisfying the aforementioned rules. The numeric component of the domain arises from modelling the ingredient and sandwich resources. Training problems have $c \in [1, 8]$ children while testing problems have $c \in [4, 292]$ children.

C.3 Numeric Ferry

A task from the domain consists of c cars spread across l locations. A ferry is able to transport up to a fixed amount of cars around to different locations. The goal of the domain is to transport the cars with the ferry to various target locations. The numeric component of the domain arises from modelling the capacity of the ferry. Training problems have $c \in [1, 20]$ cars while testing problems have $c \in [4, 974]$ cars.

C.4 Numeric Miconic

A task from the domain consists of p passengers with different weights spread across f floors. There is a single elevator with a fixed load capacity that can transport passengers between floors. Furthermore, if the load of the elevator exceeds a secondary threshold, it takes twice as long to move between floors. The goal of the domain is to move the passengers to their target floors. The numeric component of the domain arises from modelling the weight of the passengers and load capacity of the elevator. Training problems have $p \in [1, 10]$ passengers while testing problems have $p \in [1, 485]$ passengers.

C.5 Numeric Rovers

A task from the domain consists of r rovers some of which can sample rock and soil data, while others have cameras that can take images of objectives. The goal of each problem is to sample rock and soil data as well as take images of objectives and communicate all g data to the lander. The rovers can move around a map with w waypoints and the rover is only able to communicate data to the lander from a subset of waypoints. Furthermore, rovers have a limited energy supply that is consumed with any action, but they can recharge with solar panels at certain waypoints. Thus, the problem has deadends because rovers have limited energy and could exhaust them in waypoints where they cannot recharge. The numeric component of the domain arises from modelling the energy supply of the rovers. Training problems have $g \in [1, 10]$ goals while testing problems have $[2, 728]$ goals problems

C.6 Numeric Satellite

A task from the domain consists of s satellites, each carrying a subset of i instruments that can take pictures of space using a subset of m modes. Satellites can rotate to take pictures of d locations in space. Each satellite has a fixed amount of fuel that is consumed when rotating, and a fixed amount of data capacity that is consumed when taking pictures. Thus, the problem has deadends because resources are finite and can be wasted. The goal of a Satellite problem is to take pictures of a set of locations in space with specified modes while adhering to the fuel and data capacity constraints. The numeric component of the domain arises from modelling the fuel and data capacity features. Training problems have $s \in 2, 10$ satellites and testing problems have $s \in [4, 98]$ satellites.

C.7 Numeric Spanner

A task from the domain consists of s spanners scattered along a one-way hallway with l locations, and n nuts at the end of the hallway that have to be fixed. Each spanner can only be used to fix a single nut before it breaks. The goal of the domain is to fix all the nuts. The problem has deadends if not enough spanners are picked up before reaching the end of the hallway. The numeric component of the domain arises from modelling the number of spanners and nuts. Training problems have $s \in [1, 10]$ spanners while testing problems have $s \in [1487]$ spanners.

C.8 Numeric Transport

A task from the domain consists of p packages spread across l locations, with t number of trucks that can transport pick up and transport packages on a map. Each truck has a limited capacity of packages it can carry. The goal of the problem is to transport all the packages to their target locations. The numeric component of the domain arises from modelling the capacity of the trucks. Training problems have $p \in [1, 7]$ packages while testing problems have $p \in [1, 194]$ packages.

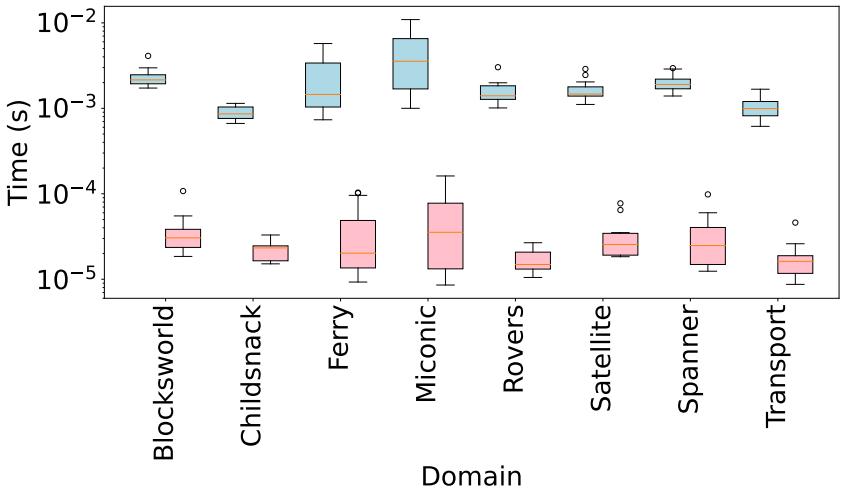


Figure 8: Distributions of heuristic evaluation time for GNN and CCWL models with $L = 1$ on problems where both were able to solve in the given timeout. Blue box plots correspond to GNN models and red box plots correspond to CCWL models.

D GNN and CCWL heuristic evaluation times

We refer to Fig. 8 for distributions of heuristic evaluation time for GNN and CCWL models with $L = 1$. Times are computed by taking the total search time for each problem and dividing by the number of heuristic evaluations made by the planner. We assume that the heuristic evaluation is the bottleneck of the search which is confirmed with informal profiling experiments.

E Effect of number of iterations and layers

We refer to Tab. 2 for the coverage of models with different L values.

Table 2: Coverage and median of expansions of solved problems for each CCWL and GNN model with varying number of iterations and layers. Higher values are better for coverage, and lower values are better for expansions. The best value per domain and metric coloured. OOM denotes that the training process exceeded the memory limit.

(a) $h^{\text{GNN}}_{\text{cost}}$

Domain	Coverage				
	0	1	2	3	4
Blocksworld	22	18	**24**	19	21
Childsnack	**18**	17	15	14	13
Ferry	**60**	**60**	**60**	**60**	**60**
Miconic	**67**	63	62	60	58
Rovers	**21**	18	11	13	13
Satellite	**22**	19	11	13	14
Spanner	**90**	**90**	**90**	**90**	77
Transport	**36**	34	31	31	30
Σ	**336**	319	304	300	286

(b) $h^{\text{GNN}}_{\text{rank}}$

Domain	Coverage				
	0	1	2	3	4
Blocksworld	20	**24**	22	**24**	22
Childsnack	18	22	26	29	**37**
Ferry	**60**	**60**	**60**	**60**	**60**
Miconic	**70**	64	62	62	59
Rovers	**22**	14	13	12	11
Satellite	**22**	14	14	17	14
Spanner	**90**	**90**	**90**	**90**	**90**
Transport	36	**38**	33	27	28
Σ	**338**	326	320	321	321

(c) $h^{\text{CCWLF}}_{\text{cost}}$

Domain	Coverage				
	0	1	2	3	4
Blocksworld	**28**	22	19	21	22
Childsnack	**22**	**22**	20	20	20
Ferry	**73**	70	65	62	58
Miconic	**90**	**90**	88	87	85
Rovers	**30**	22	19	15	15
Satellite	**26**	23	5	7	5
Spanner	**90**	**90**	89	88	86
Transport	**48**	40	37	30	27
Σ	**407**	379	342	330	318

(d) $h^{\text{CCWLF}}_{\text{rank}}$

Domain	Coverage				
	0	1	2	3	4
Blocksworld	25	19	**29**	23	22
Childsnack	22	**90**	20	23	23
Ferry	**73**	71	70	68	68
Miconic	**90**	**90**	89	87	85
Rovers	21	**23**	15	22	21
Satellite	**17**	16	7	OOM	OOM
Spanner	**90**	**90**	89	89	89
Transport	**48**	46	46	29	35
Σ	386	**445**	365	341	343

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: the abstract and introduction provide a summary of the problem we are solving and also the main content of the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: we discuss limitations in the corresponding Limitations section.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: we do not formalise any theoretical results in this paper, but refer to previous work for theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: we have done our best in the given space limits to describe in detail the experimental setup and algorithms used. We have also provided descriptions of the new datasets used.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: we aim to provide open access to our code and new datasets.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: details of training and test details are given in the corresponding experimental section of the paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: evaluation of planning tasks is also expensive to run and thus are only ran once. Informal experiments also show that training with different seeds offer minor variance in performance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: we provide hardware information as well as time and memory limits

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: the authors have read the NeurIPS Code of Ethics and believe that the research conforms with it.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: our research is mainly foundational and not tied to particular applications. It is mainly focused on improving the efficiency and effectiveness of underlying algorithms.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: there is no obvious method for our data (PDDL files) or models (planners) that may have a high risk of misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: original owners of datasets from which we create new benchmarks are properly cited and their licenses respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: we will provide detailed documentation of our new datasets and planners when we release them.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: we do not do crowdsourcing experiments nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: same justification as previous question.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.