# Effective Data Generation and Feature Selection in Learning for Planning

**Mingyu Hao[a], Dillon Z. Chen[b], Felipe Trevizan[a] and Sylvie Thiébaux[a,b]**

[a]Australian National University, Australia
[b]LAAS-CNRS, Université de Toulouse, France

**Abstract.** Previous studies have shown that leveraging data beyond optimal training plans improves the learning of search guidance for planning. Specifically, state ranking information can be extracted from states on optimal plan traces and their siblings. In this paper, we generalise this approach by extracting additional rankings from the A$^\star$ search tree for generating optimal training plans. As in the previous approach, we incur no additional search effort and negligible computational overhead for data extraction. However, extracting more data in this way may introduce many redundant features and states which slows down training. We formalise the problem of sound, redundant feature pruning and show that it is NP-complete to solve. Furthermore, we introduce several algorithms and approximations for redundant feature pruning. Experiments show that rankings learned by extracting more data from search trees for generating optimal training plans improve planner coverage. However, pairing with unsound pruning methods often results in diminishing performance, while our sound feature pruning methods provide consistent improvements across tested domains.

## 1 Introduction

In the age of deep learning and large models, it is still the case in the context of learning to plan that classical machine learning approaches reign supreme [24, 25, 29, 9, 7]. Under the hood, the currently most effective approaches have two key ingredients. The first is the use of automatically generated, symbolic features for planning, namely Description Logic Features [21] and Weisfeiler-Leman Features [27]. Such features are at least as expressive as deep learning models and have the additional benefits of being much faster to compute as well as being interpretable. The second ingredient is methods for learning search guidance in the form of rankings between pairs of states [12, 15, 5]. These methods learn to rank states on the optimal path as preferable to their siblings. The learning task is framed as a classification problem, from which a pointwise ranking function can be derived and integrated with off-the-shelf classical planners. This approach has shown improved performance over regression-based heuristics trained to learn the optimal cost-to-goal ($h^*$) of states on the optimal path [1, 26, 17, 10, 6], in part because sibling states are included in the learning process without needing to compute their $h^*$ value.

However, these approaches are not without their drawbacks. Existing ranking approaches restrict the exploration boundary only to the siblings of optimal-path states, leaving the rest of the state space unaccounted for [8, 15]. In this paper, we generalise pairwise ranking beyond this boundary by introducing new relationships that can be automatically extracted from the search space explored by A$^\star$ to compute an optimal plan. As before, these new relationships are obtained without extra search effort and with negligible computational overhead. As a result, our approach generates a substantial amount of new relationships and features not present in previous approaches.

Unfortunately, as the problem size grows, the number of features and relationships increases significantly, resulting in more time and memory being required to train a model. A cause of this blowup is that the generated features can be semantically equivalent and symmetric with one another, raising concerns about redundancy, which may slow down training and increase overfitting. A naive approach might be to compute all redundant sets of features and arbitrarily choose one feature from each set, discarding the rest. However, this naive method is unsound as it may prune features that are necessary for the computation of more complex features that are not redundant. We define such features as 'dependent' features. We show that sound pruning of redundant and dependent features is NP-complete, and provide a range of sound and unsound feature pruning methods.

Our experiments on the 2023 International Planning Competition Learning Track demonstrate that our new ranking method generates one order of magnitude more data on average and up to two orders of magnitude more. Moreover, the learnt ranking improves the planner coverage when the new data is combined with sound feature pruning.

The paper is organised as follows. Section 2 provides the necessary background. Section 3 presents our first main contribution, namely our new ranking data generation approach leveraging A$^\star$ search trees. Section 4 introduce our second main contribution about (sound) pruning of redundant and dependent features. Section 5 describes our experimental results, and Sections 6 and 7 presents related and future work, respectively.

## 2 Background

This section provides background on the planning task, optimal ranking functions, and Weisfeiler-Leman Features (WLFs) for planning. WLFs provide a concrete example of the redundant and dependent features we deal with in Section 4 and are used in the experiments in Section 5.

### *2.1 Planning*

Let $[\![n]\!]$ denote the set of integers $\{1, \ldots, n\}$. A planning task can be understood as a state transition model [13] given by a tuple $\mathbf{P} =$

$\langle \mathsf{S}, \mathsf{A}, \mathsf{C}, s_0, \mathsf{G} \rangle$ where $\mathsf{S}$ is a set of states, $\mathsf{A}$ a set of actions, $s_0 \in \mathsf{S}$ an initial state, $\mathsf{C}$ an action cost function, and $\mathsf{G} \subseteq \mathsf{S}$ a set of goal states. Each action $a \in \mathsf{A}$ is a function $a : \mathsf{S} \to \mathsf{S} \cup \{\bot\}$ where $a(s) = \bot$ if $a$ is not applicable in $s$, and otherwise $a(s) \in \mathsf{S}$ is the successor state when $a$ is applied to $s$. We denote the set of successor states of $s$ by $\mathsf{N}_s = \{a(s) \mid a \in \mathsf{A}, a(s) \neq \bot\}$. The cost function $\mathsf{C}(a) \in \mathbb{R}_{>0}$ returns the cost of applying action $a$. A unit-cost task is one for which all actions have cost 1. A solution for a planning task is a plan: a sequence of actions $\pi = a_1, \ldots, a_n$ where $s_i = a_i(s_{i-1}) \neq \bot$ for $i \in [\![n]\!]$ and $s_n \in \mathsf{G}$. A planning task is solvable if there exists at least one plan. The cost of a plan is given by the sum of its action costs: $\mathsf{C}(\pi) = \sum_{i \in [\![n]\!]} \mathsf{C}(a_i)$, and a plan is optimal if its cost is minimal among all plans for a task.

A$^\star$ search is the de facto search algorithm for optimal planning. It makes use of a heuristic function of the form $h : \mathsf{S} \to \mathbb{R}$ representing an estimated cost to reach the goal from a state if it is solvable, and $\infty$ otherwise. The optimal heuristic $h^*(s)$ of a state $s$ is the optimal cost to reach the goal from $s$ if a plan exists from $s$, otherwise $h^*(s) = \infty$. A$^\star$ search prioritises state expansion with the evaluation function $f(s) = g(s) + h(s)$ where $g(s)$ is the cost of reaching state $s$ from $s_0$ and $h(s)$ is the heuristic value of $s$. A heuristic $h$ is admissible if, $\forall s \in \mathsf{S}, h(s) \leq h^*(s)$. A$^\star$ uses two lists: (1) an open list ($\mathcal{O}$) containing states that have been generated but not yet expanded, sorted by increasing $f$ values, and (2) a closed list ($\mathcal{C}$) containing states that have already been expanded. At each iteration, the state with the lowest $f$ value in $\mathcal{O}$ is popped and moved to $\mathcal{C}$. If the state is not a goal, the successors of the state are generated and added to $\mathcal{O}$. The operations are repeated until a goal is found. For a solvable task, A$^\star$ Search with node reopening guarantees to find an optimal solution if the heuristic function $h(s)$ is admissible.

## 2.2 Optimal Ranking

The recent advancements in machine learning have motivated the development of new methods to learn search guidance. Most of these try to learn a heuristic by formulating the problem as a regression task with the optimal heuristic $h^*$ as a target, and train the model with state-value pairs $(s^*, h^*(s))$ extracted from optimal plans generated by off-the-shelf planners from small instances [17, 10, 26, 6]. Since computing $h^*$ is expensive, the training data is usually limited to one or a few plans. The model lacks knowledge of the state space outside these plans, and the learned heuristic may not generalise well to unseen states.

It has been shown that a better strategy is to learn to rank states instead [12, 8, 15]. These methods focus on learning the ranking between states on the optimal path and their siblings without incurring additional search costs. A ranking relation $\preceq$ is a total quasi-order defined over a state space $\mathsf{S}$ with the following properties.

- *Totality*: $\forall a, b \in \mathsf{S} \;\; a \preceq b \vee b \preceq a$;
- *Transitivity*: $\forall a, b, c \in \mathsf{S} \;\; a \preceq b \wedge b \preceq c \Rightarrow a \preceq c$.

Moreover, totality implies *reflexivity*, i.e., $\forall a \in \mathsf{S} \;\; a \preceq a$. Given a ranking relation $\preceq$ we define $a \prec b \iff (a \preceq b \wedge b \not\preceq a)$.

Hao et al. [15] introduced the optimal ranking relation for planning where ties between multiple optimal plans are arbitrarily broken by the data generation process. We relax the definition of an optimal ranking to encode all optimal plans by making the ranking between all siblings non-strict as follows:

**Definition 2.1** (Optimal Ranking $\preceq^{\mathrm{o}}$)**.** Given a planning task $\mathbf{P}$ and an associated optimal plan $\pi^*$ and its trace $\mathsf{S}_{\pi^*} = [s_0^*, s_1^*, ..., s_n^*]$, the optimal ranking $\preceq^{\mathrm{o}}$ is a ranking relation where, for all $s_i^* \in \mathsf{S}_{\pi^*}$ with $i \in [\![n]\!]$, we have $s_i^* \prec^{\mathrm{o}} s_{i-1}^*$ ($s_i^*$ is strictly better than its parent) and, $\forall s' \in \mathsf{N}_{s_{i-1}^*} \setminus \mathsf{S}_{\pi^*}, s_i^* \preceq^{\mathrm{o}} s'$ ($s_i^*$ is better than its siblings).

**Algorithm 1:** WL algorithm

**Data:** A graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L} \rangle$, injective HASH function, and number of iterations $L$.
**Result:** Multiset of colours.

1 $c^0(v) \leftarrow \mathbf{F}(v), \forall v \in \mathbf{V}$
2 **for** $l = 1, \ldots, L$ **do for** $v \in \mathbf{V}$ **do**
3 $\quad c^l(v) \leftarrow$ HASH $\left(c^{l-1}(v), \bigcup_{\iota \in \Sigma_{\mathrm{E}}} \{\!\{(c^{l-1}(u), \iota) \mid u \in \mathbf{N}_\iota(v)\}\!\}\right)$
4 **return** $\bigcup_{l=0,\ldots,L} \{\!\{c^l(v) \mid v \in \mathbf{V}\}\!\}$

Optimal ranking was used by Hao et al. [15] as the learning target for a deep neural network (DNN) specialised in learning ranks. In this paper, we adopt the current state-of-the-art and employ classical machine learning techniques using features derived from the graph representation of states [7] as opposed to DNNs. In the next section, we introduce *WL Features*, which is the feature space used in our experiments. It is important to note that our pruning methods are general and can be applied to any feature space such as Description Logic Features (DLFs) [21].

## 2.3 WL Features for Planning

Computing the WLF for a planning task has two components: encoding the planning task into a graph and extracting features of the obtained graph. To introduce the graph encoding, we denote a graph with categorical node features and edge labels by a tuple $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L} \rangle$. We have that $\mathbf{V}$ is a set of nodes, $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ a set of edges, $\mathbf{F} : \mathbf{V} \to \Sigma_{\mathrm{V}}$ the categorical node features, and $\mathbf{L} : \mathbf{E} \to \Sigma_{\mathrm{E}}$ the edge labels, where $\Sigma_{\mathrm{V}}$ and $\Sigma_{\mathrm{E}}$ are finite sets of symbols. The neighbourhood of a node $u \in \mathbf{V}$ in a graph is defined by $\mathbf{N}(u) = \{v \in \mathbf{V} \mid \langle u, v \rangle \in \mathbf{E}\}$. The neighbourhood of a node $u \in \mathbf{V}$ with respect to an edge label $\iota$ is defined by $\mathbf{N}_\iota(u) = \{v \in \mathbf{V} \mid e = \langle u, v \rangle \in \mathbf{E} \wedge \mathbf{L}(e) = \iota\}$. Graphs with edge features are viewed as 'relational structures' in other communities, from which we can derive relational features with various sorts of algorithms. It is not necessary to understand how to encode planning tasks into graphs for this paper, except that it is possible to do so.

The second component of the WLF pipeline involves running the 1-Weisfeiler-Leman (WL) algorithm on graphs to generate vector features. We begin by describing the underlying concept of the original WL algorithm, which involves iteratively updating node colours based on the colours of their neighbours, before describing how we can use it to generate features. The original WL algorithm was designed for graphs without edge labels, while we present an extension which supports edge labels [2] in Algorithm 1. The algorithm's input is a graph with node features and edge labels alongside a hyperparameter $L$ determining how many iterations to perform. The output of the algorithm is a canonical form for the graph that is invariant to node orderings. Line 1 initialises node graph colours as their categorical node features. Lines 2 and 3 iteratively update the colour of each node $v$ in the graph by collecting all its neighbours and the corresponding edge label $(u, \iota)$ into a multiset. This multiset is then hashed alongside $v$'s current colour with an injective function to produce a new refined colour. In practice, the injective hash function is built lazily, where every time a new multiset is encountered, it is mapped to a

new, unseen hash value. After $L$ iterations, the multiset of all node colours seen throughout the algorithm is returned.

The WL algorithm has been used to generate features for the WL graph kernel [27]. Each node colour constitutes a feature, and the value of a feature for a graph is the count of the number of nodes that exhibit the colour. Then, for a fixed number of WL iterations, given a set of colours $\mathbf{C}$ known a priori, the WL algorithm can return a fixed-sized feature vector of size $|\mathbf{C}|$ for every input graph. In a learning for planning pipeline, we collect the colours $\mathbf{C}$ from a set of training planning tasks, followed by using the colours to embed arbitrary graphs (i.e. converted from either training or testing tasks) into fixed-sized feature vectors from such colours. These steps are formalised as follows:

1. We construct $\mathbf{C}$ from a given set of graph representations of planning tasks $\mathbf{G}_1, \ldots, \mathbf{G}_m$ by running the WL algorithm with the same HASH function and number of iterations $L$ on all of them, and then taking the set union of all multiset outputs, i.e. $\mathbf{C} = \bigcup_{i \in [\![m]\!]} \text{WL}(\mathbf{G}_i)$.
2. Now suppose we have collected a set of colours and enumerated them by $\mathbf{C} = \{c_1, \ldots, c_{|\mathbf{C}|}\}$. Then given a graph $\mathbf{G}$ and its multi-set output $\mathbf{M}$ from the WL algorithm, we can define an embedding of the graph into Euclidean space by the feature vector

$$[\text{COUNT}(\mathbf{M}, c_1), \ldots, \text{COUNT}(\mathbf{M}, c_{|\mathbf{C}|})] \in \mathbb{R}^{|\mathbf{C}|}, \tag{1}$$

where $\text{COUNT}(\mathbf{M}, c_i)$ is an integer which counts the occurrence of the colour $c_i$ in $\mathbf{M}$. We note importantly that any colours returned in $\mathbf{M}$ that are not in $\mathbf{C}$ are defined as *unseen* colours and are entirely ignored in the output.

We can view colours as features, i.e. functions that map planning tasks to real values, with definition given by $c_i(\mathbf{P}) = \text{COUNT}(\mathbf{M}, c_i)$ where $\mathbf{M}$ is the multiset output of WL on the graph encoding of $\mathbf{P}$.

## 3 Ranking States from A$^\star$ Search Trees

Recall that optimal ranking only relates states in the optimal trace and their siblings; however, A$^\star$ often expands a large search tree to compute an optimal plan and all states not in the optimal plan nor among its siblings are discarded. This also holds for other approaches that learn rankings (e.g., [12]) and heuristics (e.g., [26, 17, 6, 7]). Before we show how to derive additional ranking relationships by leveraging the already generated A$^\star$ search tree, we first define a generalisation of optimal ranking called *search space ranking* and denoted by $\preceq^{\text{s}}$.

**Definition 3.1** (Search Space Ranking)**.** Given a planning task $\mathbf{P}$, the search space ranking $\preceq^{\text{s}}$ is the ranking relation over the state space $\mathsf{S}$ such that, for all states $s$ and $s'$ in $\mathsf{S}$: $s \preceq^{\text{s}} s'$ if $h^*(s) \leq h^*(s')$ and $s \prec^{\text{s}} s'$ if $h^*(s) < h^*(s')$.

Different from optimal ranking, search space ranking is defined over all states in a planning task and represents the ranking induced by the optimal heuristic $h^*$. Note that computing search space ranking explicitly can be costly since it requires solving multiple optimal planning problems, one for each required $h^*$ value. Before we show how to derive search space ranking relationships without explicitly computing $h^*$ for states outside the optimal plan, we show that search space ranking generalises optimal ranking in Prop. 3.2.

**Proposition 3.2.** *For all states $s$ and $s'$ in $\mathsf{S}$, if $s \preceq^{\text{o}} s'$ then $s \preceq^{\text{s}} s'$ and if $s \prec^{\text{o}} s'$ then $s \prec^{\text{s}} s'$.*

*Proof.* (i) If $s \preceq^{\text{o}} s'$, then there exists an optimal plan $\pi^*$ s.t. $s \in \mathsf{S}_{\pi^*}$ and $s'$ is a sibling of $s$. Therefore $h^*(s) \leq h^*(s')$ and $s \preceq^{\text{s}} s'$. (ii) If $s \prec^{\text{o}} s'$, then there exists an optimal plan $\pi^*$ s.t. either both $s$ and $s'$ are in its trace $\mathsf{S}_{\pi^*}$ and $s'$ is an ancestor of $s$, or only $s \in \mathsf{S}_{\pi^*}$, and there exists $s'' \in \mathsf{S}_{\pi^*}$ such that $s''$ is an ancestor of $s$ and a sibling of $s' \notin \mathsf{S}_{\pi^*}$. In the first case, $h^*(s) < h^*(s')$ and $s \prec^{\text{s}} s'$. In the second case, $h^*(s) < h^*(s'')$, and $s'' \preceq^{\text{o}} s'$ since $s'$ is a sibling of $s''$ which by (i) implies $h^*(s'') \leq h^*(s')$. Therefore, $h^*(s) < h^*(s')$ and $s \prec^{\text{s}} s'$. □

Now we show how to derive pairs of states satisfying $\preceq^{\text{s}}$ and $\prec^{\text{s}}$ without explicitly computing $h^*$ for both states. We achieve this by utilising admissible heuristics and the search tree computed by A$^\star$ to find an optimal plan resulting in more data at a negligible computational cost. Consider a task $\mathbf{P}$ with an optimal plan $\pi^*$ with trace $\mathsf{S}_{\pi^*} = [s_0^*, s_1^*, ..., s_n^*]$. Proposition 3.3 formalises a set of search space rankings obtained through an admissible heuristic.

**Proposition 3.3** ($\preceq^{\text{s}}$ rankings from admissible heuristics)**.** *Let $s^* \in \mathsf{S}_{\pi^*}$ and $h$ be an admissible heuristic. Then, for all $s \in \mathsf{S} \setminus \mathsf{S}_{\pi^*}$, we have that $s^* \preceq^{\text{s}} s$ if $h^*(s^*) \leq h(s)$ and $s^* \prec^{\text{s}} s$ if $h^*(s^*) < h(s)$.*

The proof is trivial since $h(s) \leq h^*(s)$ for all $s \in \mathsf{S}$ by the definition of admissible heuristic. Notice that, not all pairs derived by Prop. 3.3 need to be represented directly because of transitivity. Formally, let $s_i^* \in \mathsf{S}_{\pi^*}$ be the state with largest $h^*$ value s.t. $h^*(s_i^*) < h(s)$. By Prop. 3.3, we have that $s_j^* \preceq^{\text{s}} s$ for all $j \in \{i, \ldots, n\}$ since $h^*(s_n^*) < h^*(s_{n-1}^*) < \ldots h^*(s_i^*)$. Due to transitivity, it suffices to only explicitly represent $s_i^* \preceq^{\text{s}} s$ and the other relationships will follow from the rankings in the optimal plan.

In the remainder of this section, we show how to extract rankings from a search tree $T$ generated by A$^\star$ when an optimal plan is found. We denote by $h$ the heuristic used to generate $T$ and assume that $h$ is an admissible heuristic. The search tree $T$ is a directed acyclic graph where each node represents a state in the planning task, and edges represent the actions taken to reach those states. The root of the tree is the initial state $s_0$. Each node in the tree has an f-value $f(s) = g(s) + h(s)$ where $s$ is the corresponding state. Lastly, let $f^*(s) = g(s) + h^*(s)$ denote the optimal plan cost through $s$.

The ranking relationships defined in Prop. 3.3 relies only on the fact that $h$ is an admissible heuristic and does not leverage the knowledge of the A$^\star$ tree $T$. In the next proposition, we assume that $T$ also contains the ordering in which nodes were expanded. This can be trivially done by adding a counter to the node data structure with minimal impact on both computational time and memory usage. We only consider the ordering of nodes that ended up in the closed list and the last time they were expanded. We remove the node's ordering if it was reopened and ended up in the open list. These extra relationships satisfying $\preceq^{\text{s}}$ and $\prec^{\text{s}}$ are defined in Prop. 3.4.

**Proposition 3.4.** *Let $s \in T \setminus \mathsf{S}_{\pi^*}$ and $s^* \in \mathsf{S}_{\pi^*}$ s.t. $s$ was expanded before $s^*$ in $T$. If $h(s^*) \leq h(s)$ then $s^* \preceq^{\text{s}} s$ and, if $h(s^*) < h(s)$ then $s^* \prec^{\text{s}} s$.*

*Proof.* Since $s$ was expanded before $s^*$, we have that $f(s) \leq f(s^*)$. Moreover, $f^*(s) \geq f^*(s^*)$ since $s^* \in \mathsf{S}_{\pi^*}$ and $s \notin \mathsf{S}_{\pi^*}$. Applying the definition of the evaluation function $f$, we obtain $g(s) + h(s) \leq g(s^*) + h(s^*)$ and $g(s) + h^*(s) \geq g(s^*) + h^*(s^*)$, respectively. By subtracting the former from the latter, we have

$$\begin{aligned} g(s^*) + h^*(s^*) - (g(s^*) + h(s^*)) &\leq g(s) + h^*(s) - (g(s) + h(s)) \\ h^*(s^*) - h(s^*) &\leq h^*(s) - h(s) \\ h(s) + h^*(s^*) - h(s^*) &\leq h^*(s) \end{aligned} \tag{2}$$

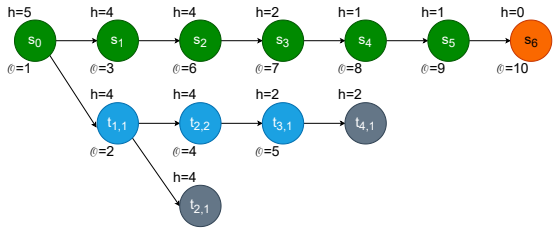


**Figure 1.** A* search tree with initial state $s_0$ and goal state $s_6$. The optimal trace is coloured in green. The heuristic is shown on the upper left corner of the corresponding node and the expansion order on the bottom left.

If $h(s^*) \leq h(s)$ (resp. $<$) then we can add (2) to it and obtain $h(s^*) + h(s) + h^*(s^*) - h(s^*) \leq h^*(s) + h(s)$ (resp. $<$) which simplifies to $h^*(s^*) \leq h^*(s)$ (resp. $<$). Therefore $s^* \preceq^{\mathrm{s}} s$ ($s^* \prec^{\mathrm{s}} s$). □

We conclude this section with an example of search space ranking relationships obtained from Props. 3.3 and 3.4 on the A* search tree.

**Example 3.5.** Consider the A* search tree shown in Figure 1 for a planning task with initial state $s_0$, goal set $\{s_6\}$ and a unit cost. The cost of the optimal plan is 6 and we assume an admissible heuristic $h$ was used to generate the search tree. The explicit optimal ranking relationships (Definition 2.1), which are also search space ranking relationships by Prop. 3.2, for our example are:

- On the optimal path: $s_6 \prec^{\mathrm{s}} s_5 \prec^{\mathrm{s}} s_4 \prec^{\mathrm{s}} s_3 \prec^{\mathrm{s}} s_2 \prec^{\mathrm{s}} s_1 \prec^{\mathrm{s}} s_0$.
- Siblings along the optimal path: $s_1 \preceq^{\mathrm{s}} t_{1,1}$.

All other optimal ranking relationships are implicitly derived by using transitivity.

Since $h^*(s_2)$=4, $h^*(s_3)$=3, $h^*(s_4)$=2, $h^*(s_5)$=1, by Prop. 3.3, we have: $s_2 \preceq^{\mathrm{s}} \{t_{1,1}, t_{2,1}, t_{2,2}\}$, $\{s_3, s_4\} \prec^{\mathrm{s}} \{t_{1,1}, t_{2,1}, t_{2,2}\}$, $s_4 \preceq^{\mathrm{s}} \{t_{3,1}, t_{4,1}\}$, and $s_5 \prec^{\mathrm{s}} \{t_{1,1}, t_{2,1}, t_{2,2}, t_{3,1}, t_{4,1}\}$ Notice that $\{s_3, s_4\} \prec^{\mathrm{s}} \{t_{1,1}, t_{2,1}, t_{2,2}\}$ are redundant because $s_4 \prec^{\mathrm{s}} s_3 \prec^{\mathrm{s}} s_2 \preceq^{\mathrm{s}} \{t_{1,1}, t_{2,1}, t_{2,2}\}$ by transitivity. Similarly, $s_2 \preceq^{\mathrm{s}} t_{1,1}$ is redundant because $s_2 \prec^{\mathrm{s}} s_1 \preceq^{\mathrm{s}} t_{1,1}$ as well as all relationships derived for $s_5$. Therefore, Prop. 3.3 adds 4 new relationships to be explicitly represented $s_2 \preceq^{\mathrm{s}} \{t_{2,1}, t_{2,2}\}$ and $s_4 \preceq^{\mathrm{s}} \{t_{3,1}, t_{4,1}\}$.

The expansion ordering of the nodes in our example is $[s_0, t_{1,1}, s_1, t_{2,2}, t_{3,1}, s_2, s_3, s_4, s_5, s_6]$ and we can use Prop. 3.4 to derive the following relationships: $\{s_1, s_2\} \preceq^{\mathrm{s}} t_{1,1}$, $\{s_3, \ldots, s_6\} \prec^{\mathrm{s}} \{t_{1,1}, t_{2,2}\}$, $s_2 \preceq^{\mathrm{s}} t_{2,2}$, $s_3 \preceq^{\mathrm{s}} t_{3,1}$, and $\{s_4, s_5, s_6\} \prec^{\mathrm{s}} t_{3,1}$. Removing pairs obtained by transitivity over the optimal ranking and Prop. 3.3 pairs, we have one new relationship to be explicitly represented: $s_3 \preceq^{\mathrm{s}} t_{3,1}$.

Thus, using space search ranking and utilising all data within the A* search tree, we extracted 5 new explicit ranking relationships more than optimal ranking. This nearly doubles the total explicit relationships for the same problem: 7 for optimal ranking and 12 for space search ranking. Considering explicit and implicit relationships due to transitivity, these numbers are 27 for optimal ranking and 47 for space search ranking.

Note that the branching factor in our example is very small and it equals 1 for all states except $s_0$ and $t_{1,1}$. The branching factor in most problems of interest largely exceeds 1, resulting in a larger A* search tree and substantially more ranking relationships. Indeed, the number of explicit relationships is linearly correlated with the size of the search tree. Therefore, although it is possible to obtain a significantly larger dataset, we need to select a subset of the data to train our models due to memory and time constraints, as well as to avoid overfitting. In the next section, we formalise this problem and introduce several methods to solve it.

## 4 Redundancy Pruning

To manage the substantial growth in both states and features, we present methods to prune redundant data from the training set. We formalise the concept of redundant features in Section 4.1 and present both sound and unsound methods for feature pruning in Sections 4.2 and 4.3, respectively.

### 4.1 Redundant Features

We define redundant features as features that have the same evaluations on a set of states. We then define dependent features as pairs of features $c_1, c_2$ where the computation of $c_1$ depends on the computation of $c_2$.

**Definition 4.1.** Let $\mathbf{C}$ be a set of features and $\mathsf{S}$ a set of states. We say that a feature $c \in \mathbf{C}$ is *redundant* with respect to $\mathsf{S}$ if there exists another feature $c' \in \mathbf{C} \setminus \{c\}$ such that $c(s) = c'(s)$ for all states $s \in \mathsf{S}$.[1] In this case we say that $c$ is redundant with $c'$. A *redundant set* of features is a set of features such that all features are redundant with one another. A feature is *unique* if it is not redundant.

**Definition 4.2.** A set of features $\mathbf{C}$ is *dependent* if there exists a pair of distinct features $c_1, c_2 \in \mathbf{C}$ where the computation of $c_1$ can provide us with the computation of $c_2$ with no additional cost. Here, we say that $c_1$ is dependent on $c_2$.

Semantically equivalent and hence redundant WLFs have been empirically observed previously. For example, the original WLF for planning paper [7, Fig. 7] presents a subset of WLFs collected for the Blocks World domain all of which semantically compute "the number of blocks correctly placed on the table and with the correct block above it". Although redundant and dependent features are not inherently problematic as they have no effect on expressiveness, these features introduce computational inefficiencies. For instance, a WLF computed at iteration $l$ depends on a set of WLFs from iteration $l-1$, thus deleting redundant features from iteration $l-1$ provides no runtime benefit as they remain necessary for computing features in iteration $l$. Similarly, DLFs for planning [21] are also features constructed iteratively based on a parameter $l$ denoting feature complexity. However, as we will discuss next, premature pruning of features at an iteration $l-1$ may lead to the loss of features necessary for the computation of more refined features at iteration $l$.

### 4.2 Unsound Feature Pruning

Issues arising from redundant and dependent features have been identified in earlier works concerning rule generation for generalised planning from DLF. An explicit feature pruning method for DLFs was introduced and implemented by Bonet et al. [4]. Let $\mathbf{C}^l$ denote the set of features collected at iteration $l$. Bonet et al. [4] prune features from $\mathbf{C}^l$ that are redundant in $\bigcup_{i=1}^{l} \mathbf{C}^i$ with respect to the set of training states in a learning pipeline. In the case that two features $c_a, c_b \in \mathbf{C}^l$, $c_a \neq c_b$ are redundant, we randomly discard one and keep the other. We denote this pruning method as `i-bfg19`, where the

[1] For all remaining definitions and mentions of 'redundant', we assume a fixed set of states for the definition to hold.

`i-` prefix emphasises that pruning is done iteratively during feature construction.

An issue with `i-bfg19` is its greedy nature which may preemptively prune features that could contribute to more complex features that are not redundant. For instance, it is possible that two features $c_a$ and $c_b$ generated at or before an iteration $l$ are redundant such that unique features $c'_a$ and $c'_b$ exist in the subsequent iteration $l+1$ which depend on $c_a$ and $c_b$, respectively. In preliminary experiments, we observed that `i-bfg19` pruning results in no WLFs to be generated after just $l = 2$ iterations despite the existence of more unique features for $l \geq 3$.

An orthogonal feature pruning approach involves the frequency with which a feature is observed with a non-zero value. Although there exists some body of work on both offline and online feature selection [28] from classical machine learning, such methods are limited to the training label context.[2] However, WLFs and DLFs are used in a variety of different pipelines which limits the applicability of such methods. A naive approach for pruning features is to count how often a colour/formula in WLF/DLF is encountered by summing the feature vectors across the training data, and then discarding features based on some threshold. We can perform this iteratively since any feature that depends on a feature pruned this way will have a strictly lower count. We denote this pruning method as `i-freq`. Note that there is no guarantee that the infrequent features pruned directly or indirectly are redundant.

### *4.3 Sound Feature Pruning*

To address the issue of premature pruning, we introduce the concept of sound feature pruning in the context of redundant and dependent features as follows.

**Definition 4.3.** Pruning a feature $c$ from a set of features $\mathbf{C}$ is *sound* if $c$ is redundant and no other feature in $\mathbf{C}$ is dependent on $c$.

We next introduce MaxFeaturePrune as the problem of maximising the number of features pruned in $\mathbf{C}$ while maintaining soundness and introduce a MaxSAT encoding for solving it. Indeed MaxSAT is a reasonable approach as we show that MaxFeaturePrune is NP-hard via reduction from the minimum cardinality hitting set problem [11, p. 222].

**Theorem 4.4.** *MaxFeaturePrune is NP-complete.*

*Proof.* Membership follows from the upcoming MaxSAT encoding. For hardness, we reduce from the minimum cardinality hitting set (MCHS) problem. Let $\{S_1, \ldots, S_n\}$ be a collection of sets. MCHS asks to find a subset $S \subseteq U := \bigcup_{i \in [\![n]\!]} S_i$ such that $|S \cap S_i| \geq 1$ for all $i \in [\![n]\!]$ and $|S|$ is minimal over all such subsets. To encode MCHS in MaxFeaturePrune, first let $u_1, \ldots, u_m$ be an enumeration of $U$. We then introduce a set of features $\mathbf{C} = C \cup \bigcup_{i \in [\![n]\!]} D_i$ where $C = \{c_1, \ldots, c_m\}$ and $D_i = \{d_{j,i} \mid u_j \in S_i\}$. We enforce that for $C$ and each of the $D_i$, are redundant sets, meaning that features within each set are redundant with one another. Furthermore, we have that the $d_{j,i}$ are dependent on the $c_j$. Figure 2 illustrates the reduction.

To see that this is a valid reduction, we note that we want to keep only one $d_{j,i}$ from each $D_i$ and minimise the number of kept features in $C$ to maximise the number of pruned features from the $D_i$ sets. Given that no features depend on the $d_{j,i}$ it is indeed optimal to only keep one $d_{j,i}$ from each $D_i$. Next, the dependency of the $d_{j,i}$ on the $c_j$ implies that we need to keep the $c_j$ that correspond to the kept $d_{j,i}$. Thus all that remains is to maximise the number of pruned $c_j$ in MaxFeaturePrune or equivalently minimise the number of kept $c_j$. This is equivalent to the MCHS problem as the kept $c_j$ correspond to the elements in a MCHS $S$, with the $d_{j,i}$ marking out each of the hit $S_i$ subsets. □

[2] Our iterative, dependent feature generation setup is a special case of online feature selection.

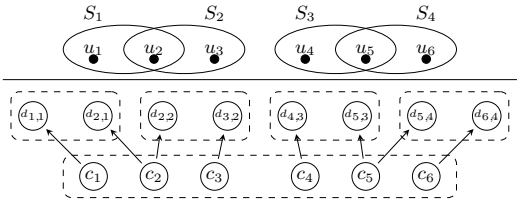


**Figure 2.** Reduction from a MCHS problem (top) to MaxFeaturePrune (bottom). Dashed rectangles indicate sets of redundant features and edges feature dependencies.

We now introduce the `msat` pruning approach, which solves MaxFeaturePrune by encoding it as a partially weighted MaxSAT encoding. A partially weighted MaxSAT problem consists of a set of Boolean variables, a set of positively weighted soft constraints and a set of hard constraints. The objective is to find a variable assignment satisfying all hard clauses and maximising the sum of the weights of the satisfied soft clauses.

To encode MaxFeaturePrune, we introduce a set of Boolean variables $x_{c_1}, \ldots, x_{c_{|\mathbf{C}|}}$ where $x_{c_i}$ is true if we decide to prune the feature $c_i$ and false otherwise. The soft clauses we introduce are simply the singleton clauses of the variables for all $c \in \mathbf{C}$, each with a weight of 1. This represents our goal of maximising the number of pruned features.

Next, we note that a feature set $\mathbf{C}$ can be partitioned into redundant sets $\mathbf{C}_1, \ldots, \mathbf{C}_p$ (some of which are singletons of a unique feature). The set of hard constraints we introduce involve (1) ensuring that at least one feature is kept from each redundant set with the clause $\bigvee_{c \in \mathbf{C}_i} x_c$ for all $i \in [\![p]\!]$, and (2) ensuring that if a feature is not pruned, then all the features it depends on are also not pruned, noted by the clause $\neg x_{c'} \vee x_c$ for all $c, c' \in \mathbf{C}$ where $c'$ depends on $c$.

We observed in our experiments that solving this MaxSAT problem takes a matter of seconds and it is not a bottleneck. Furthermore, compared to iterative, unsound pruning methods, `msat` has the disadvantage that it must generate all possible features a priori before any pruning is performed. However, this is an unavoidable tradeoff in order to ensure soundness which iterative methods cannot guarantee. In light of this tradeoff, we also introduce two unsound pruning methods based on the MaxSAT encoding: `i-msat` which solves the MaxSAT problem for each iteration; and `i-mf` which applies frequency pruning after solving the MaxSAT for the whole dataset.

## 5 Experiments

We use the domains and training and testing task splits from the IPC-23 Learning Track [23]. Each domain consists of 99 training tasks and 90 testing tasks divided into "easy", "medium" and "hard" difficulty. We ran A$^\star$ search with the LM-cut heuristic [16] to generate optimal plans and search trees for each training task with a 30 minutes timeout. States from the search tree are collected iteratively during training: we start with the states in the optimal trace and, at each iteration, add the siblings in the search tree of all states collected so far. We stop collecting states when the total data size (number of collected states × number of features) reaches one billion (this threshold was

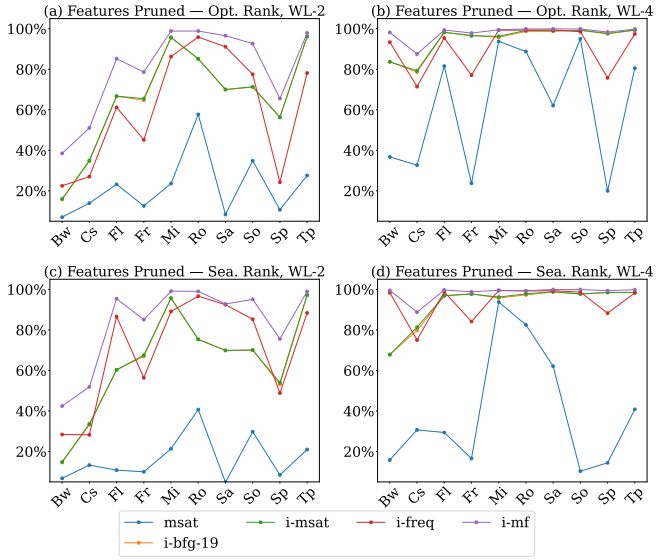


**Figure 3.** (a)–(d) Percentage of features pruned by different methods w.r.t no feature pruning, for optimal ranking (first row) and search space ranking (second row), for WL-2 (left column) and WL-4 (right column).

empirically determined). We apply feature pruning after collecting the data. Our experimental setup uses the Instance Learning Graph (ILG) representation of planning tasks [7] combined with the WLF described in Section 2.3 for feature computation, with 2 (WL-2) and 4 iterations (WL-4). For learning both optimal and search-space rankings, we experiment with two types of models: the Rank-based LP model (LP) [5] and the RankSVM model (SVM) [12]. We use a 35 GB memory limit for all training scripts. Similarly to Hao et al. [15], testing tasks are solved using GBFS and the learned ranking functions. We follow the IPC 2023 learning track testing configurations, which use a 30-minute timeout and 8GB cutoff for each task. For `i-freq` and `i-mf`, we prune the features that appear in 1% or less of the total number of collected states. The source code and dataset for all the experiments can be found at [14]. For all our results, we abbreviate the domains as follows: **BW** (blocksworld), **CS** (childsnack), **FL** (floortile), **FR** (ferry), **MI** (miconic), **RO** (rovers), **SA** (satellite), **SO** (sokoban), **SP** (spanner) and **TP** (transport). Table 1 summarises the total coverage of various configurations on the experimented benchmarks and next we address questions of interest from our empirical analysis.

**How effective is feature pruning?** Fig. 3 (a)–(d) show the percentage of features pruned relative to no feature pruning for optimal ranking and search space ranking and WLFs using 2 and 4 iterations. The number of features pruned by the sound feature pruning method, namely `msat`, ranges from 4.8% (search space ranking, WL-2, satellite) to 95.0% (optimal ranking, WL-4, sokoban). As this is a sound method, it reveals for the first time the extent of redundancy generated by WLFs for planning and provides insight into why WL-4 features may not improve planning performance. The unsound feature pruning methods are able to prune even more features, with the minimum being 14.58% (`i-bfg19`, search space ranking, WL-2, blocksworld) to 99.9% (`i-mf`, search space rankings, WL-4, satellite). Moreover, in several domains, the unsound approaches can prune more than 90% of the features. Comparing WL-2 and WL-4, we observe that all pruning methods are more effective in WL-4 due to the excess of redundant features generated by WL-4. Besides being effective, sound

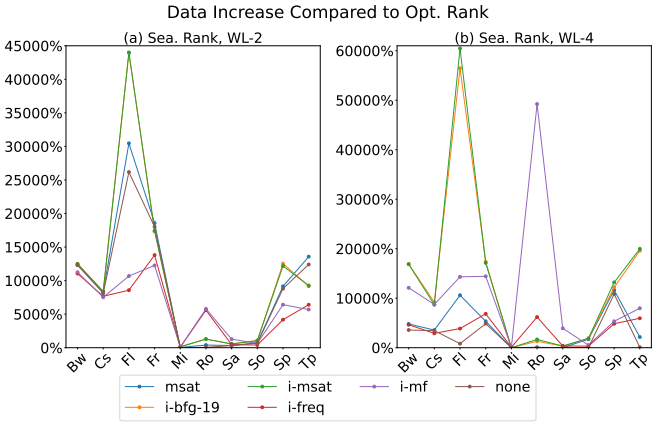


**Figure 4.** Increase in the number of non-zero entries in the training data matrix for different domains for Search Space ranking w.r.t. optimal ranking for different values of WL iterations.

feature pruning also helps learning better rankings as evidenced by the planning performance (Tab. 1). In all experiment settings, the `msat` feature pruning method has improved coverage compared to no feature pruning. However, for unsound pruning methods, the effect on coverage is domain-dependent. For example, unsound feature pruning methods tend to improve the coverage of floortile when search space ranking is used. However, the coverage of spanner is lower under the same setting. The performance decrease from unsound pruning is more evident on WL-4, suggesting that these methods are overly aggressive, resulting in the removal of too many features that may contain crucial information.

**How much more data can search space ranking extract?** Fig. 4 shows the increase in data provided by search space ranking relative to optimal ranking for the various feature pruning methods and numbers of WL iterations. Given the sparsity of our data, this increase is quantified as the increase in non-zero values within the matrix representing the training data. The most significant increase in data is seen in WL-4. The mean change across feature pruning methods ranges from 80% for miconic to 60461% for floortile. The overall average for all WL-2 approaches is a 8570% increase in data. Search space ranking can also extract more data than optimal ranking for WL-4 but in smaller average quantities of 8192% increase. The reason for this decrease in additional data is due to our fixed matrix size of one billion, which combined with the much larger feature space generated by WL-4 results in fewer states being selected. Regardless of the model (LP or SVM) or the number of WL iterations (WL-2 or WL-4), by leveraging the A$^\star$ search tree data, our search space ranking approach is able to surpass the current state-of-the-art in total coverage, namely optimal ranking without feature pruning. Moreover, search space ranking leads to a drop in the ratio of unseen colours during testing. Specifically, with optimal ranking, the average number of colors unseen during training that were observed during testing accounts for 45.66% (WL-2) and 42.75% (WL-4) of the latter, while these ratios drop to 31.30% and 38.93% when using state space ranking.

## 6 Related Work

With respect to feature pruning, Kuzelka and Železný [18] introduced "tree-like" relational features. These features can be viewed as more refined WLFs and are obtained by examining subsets of neighbours in Line 3 of the WL algorithm (Alg. 1). As a result, they observed

| | | | WL-2 | | | | | | | | | | | WL-4 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| model | tr. space | feature pr. | Bw | Cs | Fl | Fr | Mi | Ro | Sa | So | Sp | Tp | Σ | Bw | Cs | Fl | Fr | Mi | Ro | Sa | So | Sp | Tp | Σ |
| LP | Opt.rank | none | 73 | 13 | 2 | 76 | 79 | **42** | **51** | 32 | 71 | 40 | 479 | **83** | 13 | 2 | 76 | 80 | 39 | 52 | 28 | 70 | 39 | 482 |
| | | msat | 77 | 13 | 2 | 76 | 78 | **42** | **51** | 33 | **72** | 41 | 485 | **83** | 13 | 2 | 75 | 71 | 41 | **54** | 33 | 70 | 40 | 482 |
| | | i-bfg-19 | 75 | 14 | 2 | 75 | **80** | **42** | 50 | 31 | 70 | **43** | 482 | 71 | 19 | 2 | 74 | 76 | 42 | 48 | 33 | 71 | **44** | 480 |
| | | i-msat | 73 | 14 | 2 | 72 | **80** | **42** | 45 | 31 | 71 | **43** | 473 | 72 | 18 | 2 | 73 | **81** | 42 | 48 | 34 | 71 | 42 | 483 |
| | | i-freq | 72 | 13 | 2 | **79** | **80** | 40 | 37 | **34** | 70 | 39 | 466 | 78 | 13 | 2 | 77 | 76 | 41 | 42 | 30 | 64 | 39 | 462 |
| | | i-mf | 74 | 13 | 2 | 73 | 79 | 41 | 50 | **34** | 71 | 41 | 478 | 60 | 17 | 2 | 74 | 76 | 41 | **54** | 30 | **72** | 42 | 468 |
| | Sea. Rank | none | **80** | 30 | 4 | 75 | 79 | 41 | 47 | 31 | 71 | 38 | 496 | 72 | 58 | 10 | 75 | 74 | 42 | 52 | 29 | 70 | 41 | 523 |
| | | msat | **80** | 41 | 3 | 76 | **80** | **42** | 45 | 32 | 69 | 36 | **504** | 80 | **59** | 10 | 74 | 68 | 42 | **54** | 33 | 70 | 41 | **531** |
| | | i-bfg-19 | 76 | 24 | 4 | 75 | 79 | **42** | 45 | 32 | 67 | 40 | 484 | 55 | 53 | 16 | 76 | 76 | **43** | 46 | **35** | 33 | 39 | 472 |
| | | i-msat | 73 | 29 | 4 | 73 | **80** | **42** | 42 | 31 | 67 | **43** | 484 | 77 | 13 | 15 | **78** | **81** | 42 | 48 | 33 | 37 | 42 | 466 |
| | | i-freq | 75 | 22 | **6** | 74 | **80** | 40 | 41 | 33 | 68 | 35 | 474 | 48 | 56 | **22** | 49 | 75 | 41 | 45 | 31 | 56 | 37 | 460 |
| | | i-mf | 73 | 20 | 5 | 76 | **80** | 39 | 39 | 27 | 64 | 39 | 462 | 52 | 34 | 6 | **78** | 76 | 40 | 40 | 32 | 70 | 42 | 470 |
| SVM | Opt.rank | none | 73 | 31 | 2 | 74 | 74 | 39 | 39 | 29 | 67 | 37 | 465 | 75 | 28 | 2 | 71 | 74 | 37 | 46 | 23 | 66 | 38 | 460 |
| | | msat | 77 | 32 | 2 | 74 | 74 | 39 | 39 | 30 | 68 | 35 | 470 | 74 | 32 | 4 | 71 | 74 | 40 | 48 | 28 | 66 | 39 | 476 |
| | | i-bfg-19 | 74 | 32 | 2 | 65 | 74 | 39 | 36 | 30 | 67 | 40 | 459 | 62 | 31 | 2 | 70 | 74 | 41 | 43 | 33 | 67 | 39 | 462 |
| | | i-msat | 78 | 41 | 2 | 67 | 74 | 39 | 36 | 30 | 60 | 39 | 466 | 64 | 31 | 2 | 70 | 74 | 42 | 47 | 32 | 67 | 37 | 466 |
| | | i-freq | 79 | 30 | 2 | 74 | 74 | 39 | 37 | 31 | 69 | 35 | 470 | 67 | 27 | 4 | 71 | 74 | 40 | 35 | 29 | 48 | 36 | 431 |
| | | i-mf | 74 | 28 | 2 | 68 | 74 | 39 | 39 | 30 | 61 | 38 | 453 | 73 | 26 | 2 | 70 | 74 | 40 | 40 | 33 | 67 | 36 | 461 |
| | Sea. Rank | none | 72 | 50 | 5 | 70 | 74 | 39 | 40 | 31 | 57 | 34 | 472 | 78 | 31 | 2 | 69 | 73 | 37 | 41 | 26 | 66 | 38 | 461 |
| | | msat | 70 | 51 | 3 | 72 | 73 | 39 | 42 | 31 | 59 | 34 | 474 | 75 | 32 | 5 | 70 | 74 | 40 | 45 | 31 | 65 | 40 | 477 |
| | | i-bfg-19 | 73 | **54** | 4 | 68 | 74 | 39 | 36 | 31 | 46 | 41 | 466 | 64 | 30 | 2 | 71 | 74 | 42 | 44 | 32 | 55 | 35 | 449 |
| | | i-msat | 69 | 53 | 4 | 68 | 74 | 39 | 36 | 31 | 45 | 39 | 458 | 68 | 27 | 2 | 72 | 75 | 41 | 44 | 32 | 60 | 37 | 458 |
| | | i-freq | 64 | 53 | 4 | 72 | 74 | 39 | 33 | 31 | 30 | 34 | 434 | 41 | 54 | 3 | 71 | 74 | 41 | 33 | 27 | 63 | 33 | 440 |
| | | i-mf | 67 | 36 | 3 | 71 | 75 | 38 | 36 | 28 | 30 | 40 | 424 | 50 | 26 | 2 | 68 | 75 | 40 | 40 | 31 | 69 | 34 | 435 |

**Table 1.** Coverage of WL configurations using various feature pruning approaches. The best coverage in each column is indicated by the cell colouring intensity and the best one is in bold.

a greater increase in tree-like features and introduced pruning algorithms with a stronger notion of redundancy (Lavrac et al. 19; Kuzelka and Železný 18, Definition 11) but restricted to the context of supervised inductive logic programming. Our work differs by focusing on feature pruning methods that are agnostic to the downstream learning task while emphasising the soundness and efficiency for planning.

The concept of using an admissible heuristic to help learning $h^*$ is also presented in Núñez-Molina et al. [22]. They model heuristic learning as fitting a truncated Gaussian distribution, with the admissible heuristic as the lower bound and the cost of suboptimal plans as upper bounds. However, the fundamental issue of regression-based methods remains – the model imposes value bounds on states. In contrast, ranking-based approaches such as optimal ranking and search space ranking have shown that specifying a relative ordering between states is sufficient and results in better learning outcomes.

Regarding using states outside optimal plans and their siblings for learning, the closest work to ours is Drexler et al. [9] where Breadth-First Search is employed to expand the full state space. This data generation method is effective only for small problems as acknowledged by the authors while our approach scales to large training problems. Other approaches generate additional data by creating new problems based on the current ones through *backwards* random walks from the goal (e.g., [1, 20, 3]) differing in their stopping criteria and sampling strategies. These approaches are complementary to ours and are typically employed when no training set is available. Given a training set of problems that are solved optimally with A$^\star$, our approach provides additional data at minimal cost while sampling methods incur the cost of finding and solving interesting problems.

## 7 Conclusion and Future Work

In this paper, we proposed a novel pairwise ranking method to significantly expand the scope of training data. Our approach extracted an order of magnitude more training examples from an A$^\star$ search tree, which were then used to learn a state ranking with features generated by the WL algorithm. However, the number of features generally increases with the problem size and the number of WL-iterations. To address this challenge, we introduced and evaluated both sound and unsound feature pruning methods. Experimental results demonstrated that sound pruning methods consistently improved performance in learning-based search and planning tasks.

Our experiments used the LM-cut heuristic. In the future, it would be interesting to study the impact of different admissible heuristics on the generation of useful ranking relationships in the various domains. Moreover, consistent heuristics allow extracting further relationships, beyond those afforded by Prop. 3.4.

Another future work avenue is the exploration of state pruning. This paper focuses exclusively on pruning features. As we introduce an exponential increase in states within the training set, it remains uncertain whether all states contribute positively to performance. Investigating the impact of individual additional states and pruning redundant ones will free memory for adding more meaningful samples into the training set, hence further improving the learning outcomes.

## Acknowledgements

Mingyu Hao and Sylvie Thiébaux were supported by the Australian Research Council grant DP220103815. Sylvie was also supported by the Artificial and Natural Intelligence Toulouse Institute (ANITI) under the grant agreement ANR-23-IACL-0002, and by the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No. 101070149.

## References

[1] S. J. Arfaee, S. Zilles, and R. C. Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17), 2011.

[2] P. Barceló, M. Galkin, C. Morris, and M. A. R. Orth. Weisfeiler and leman go relational. In *Proceedings of the 1st Learning on Graphs Conference (LoG)*, 2022.

[3] R. V. Bettker, P. P. Minini, A. G. Pereira, and M. Ritt. Understanding sample generation strategies for learning heuristic functions in classical planning. *Journal of Artificial Intelligence Research*, 80:243–271, 2024.

[4] B. Bonet, G. Francès, and H. Geffner. Learning features and abstract actions for computing generalized plans. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[5] D. Z. Chen and S. Thiébaux. Graph learning for numeric planning. In *Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[6] D. Z. Chen, S. Thiébaux, and F. Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

[7] D. Z. Chen, F. Trevizan, and S. Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS)*, 2024.

[8] L. Chrestien, S. Edelkamp, A. Komenda, and T. Pevný. Optimize planning heuristics to rank, not to estimate cost-to-goal. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[9] D. Drexler, J. Seipp, and H. Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2022.

[10] P. Ferber, M. Helmert, and J. Hoffmann. Neural network heuristics for classical planning: A study of hyperparameter space. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*, 2020.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[12] C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning to rank for synthesizing planning heuristics. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

[13] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[14] M. Hao. Source code and data for ecai-25 proceeding "effective data generation and feature selection in learning for planning", Aug. 2025. URL https://doi.org/10.5281/zenodo.16832605.

[15] M. Hao, F. Trevizan, S. Thiébaux, P. Ferber, and J. Hoffmann. Guiding GBFS through learned pairwise rankings. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[16] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[17] R. Karia and S. Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[18] O. Kuzelka and F. Zelezný. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, 83(2):163–192, 2011.

[19] N. Lavrac, D. Gamberger, and V. Jovanoski. A study of relevance for learning in deductive databases. *The Journal of Logic Programming*, 40 (2-3):215–249, 1999.

[20] O. Marom and B. Rosman. Utilising uncertainty for efficient learning of likely-admissible heuristics. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[21] M. Martín and H. Geffner. Learning generalized policies in planning using concept languages. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2000.

[22] C. Núñez-Molina, M. Asai, P. Mesejo, and J. Fernández-Olivares. On using admissible bounds for learning forward search heuristics. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[23] J. Segovia and J. Seipp. Benchmarking Repository of IPC 2023 - Learning Track. https://github.com/ipc2023-learning/benchmarks, 2023.

[24] J. Segovia-Aguas, S. Jiménez, and A. Jonsson. Generalized planning as heuristic search. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS)*, 2021.

[25] J. Segovia-Aguas, S. J. Celorrio, and A. Jonsson. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artificial Intelligence*, 330, 2024.

[26] W. Shen, F. Trevizan, and S. Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[27] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12, 2011.

[28] J. Wang, P. Zhao, S. C. H. Hoi, and R. Jin. Online feature selection and its applications. *IEEE Trans. Knowl. Data Eng.*, 26(3):698–710, 2014.

[29] R. Yang, T. Silver, A. Curtis, T. Lozano-Pérez, and L. P. Kaelbling. PG3: policy-guided planning for generalized policy generation. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.