MOOSE: Satisficing and Optimal Generalised Planning via Goal Regression

Dillon Z. Chen[1,2,3], **Till Hofmann**[4], **Toryn Q. Klassen**[1,3], **Sheila A. McIlraith**[1,3]
[1]Vector Institute, Toronto, Canada [2]LAAS-CNRS, University of Toulouse, France
[3]University of Toronto, Canada [4]RWTH Aachen University, Germany

Abstract

Generalised Planning refers to the task of synthesising programs that can solve families of related planning problems. We propose a novel generalised planner MOOSE which employs powerful, decades-old ideas from the knowledge representation and planning communities: regression rewriting, relaxation, and database algorithms. We formalise and describe a new class of generalised problems for which our approach can learn sound and complete generalised plans. MOOSE is also the first learning approach for provably optimal planning across problems with arbitrarily many objects. Experimental results show that MOOSE solves more problems than state-of-the-art planners over classical, numeric, and optimal planning settings on the tested benchmarks. Notably, MOOSE uses less than 1GB of memory to synthesise and instantiate generalised plans across all experiments.

1 Introduction

Generalised Planning (GP) aims to compute generalised plans: programmatic plans that can solve families of related planning problems. A grand goal of GP is to overcome the scalability issues of general-purpose planners which are sound and complete for decidable planning problems, but in turn are not able to leverage patterns in and solve similarly structured problems more efficiently. Indeed, several planning domains exist that are computationally easy to solve and exhibit satisficing policies, such as variants of the package delivery domain [Helmert, 2003]. UPS™ delivered over 20 million packages every day across over 200 countries and territories in 2024 [UPS, 2025]. However, state-of-the-art planners struggle to scale up to a simplified version of the delivery problem with 100 packages [Taitler *et al.*, 2024].

A GP problem typically comprises a planning domain $\mathbb{D}$ representing an action theory, and a set of planning problems $\mathbb{P}$. Figure 1 illustrates the two main modules characteristic of many approaches for GP. The first module (green) takes as input an action theory $\mathbb{D}$ and a set of example problems $\mathbb{P}_{\text{train}}$ drawn from $\mathbb{P}$ in order to synthesise a generalised plan for solving problems in $\mathbb{P}$. The second module (blue) instantiates the generalised plan on a given input problem sharing the same action theory in order to provide a goal-achieving plan for that problem. Problems that are input into the second module may exhibit any number of objects.

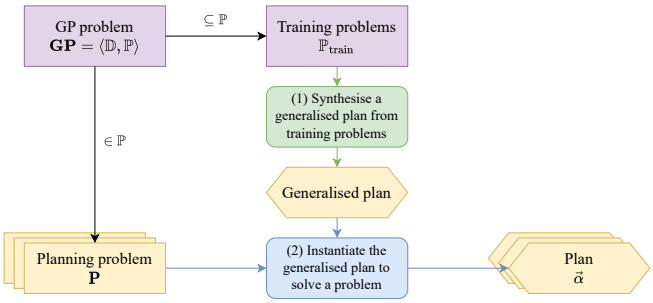

Figure 1: A common GP setup consisting of an action theory $\mathbb{D}$ and set of planning problems $\mathbb{P}$ satisfying the action theory. A generalised planner consists of two modules: (1) synthesis and (2) instantiation. See text for more details.

Srivastava *et al.* [2011a] introduced several metrics for measuring the effectiveness of a generalised planner, including the size of the set of planning domains an approach can solve (*expressiveness*), the time and data it takes to synthesise a generalised plan (*synthesis cost*) for a given domain, and the cost it takes to instantiate and execute the generalised plan for new planning problems (*execution cost*) of a given domain. Generalised planners aim to amortise the cost of synthesising a generalised plan against the cost of solving each planning problem individually. However, recent planning competition results also show that current generalised planners are still outperformed by traditional planners on standard planning domains [Taitler *et al.*, 2024], suggesting much room for improvement across all three GP metrics.

In this paper, we introduce MOOSE, a new generalised planner which draws upon insights and long-standing ideas from the knowledge representation and reasoning (KRR) and planning communities to advance the state of the art in GP. To capture expressive policies we employ *regression* rewriting, a preimage computation of a state under a sequence of actions [Fikes *et al.*, 1972; Waldinger, 1977; Reiter, 1991]. We use regression to identify sufficient and minimal state conditions required for generating lifted, goal-achieving rules [Muise *et al.*, 2012; Illanes and McIlraith, 2019] from our training plans. To lower synthesis costs, we *relax* the

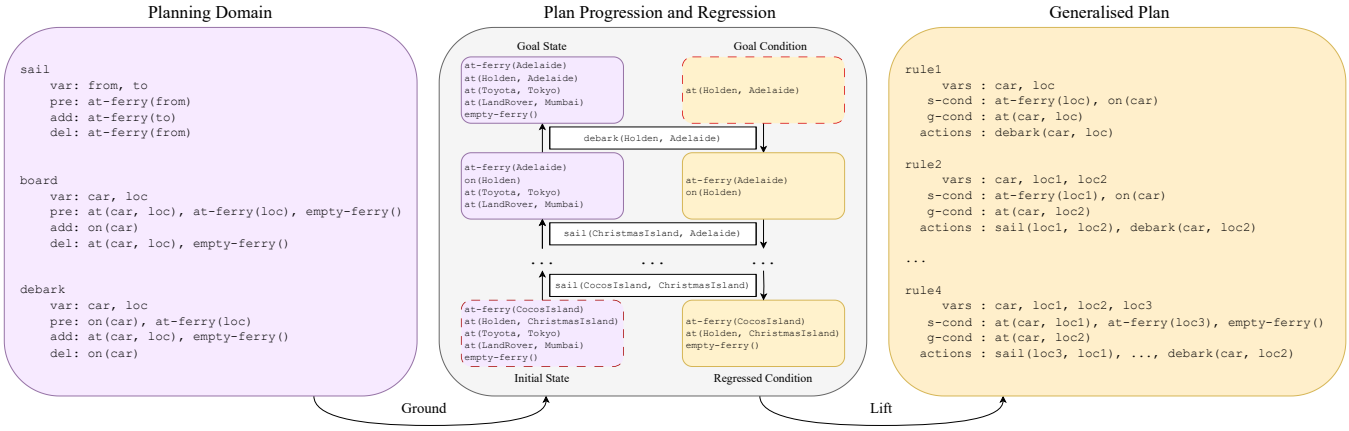

Figure 2: Left: the Ferry planning domain consisting of three PDDL action schemata for a ferry to sail, board a car, and debark a car. Middle: progression of the initial state of a planning problem (purple) via the plan in the center (white), and the regression of the goal condition (yellow) via the same plan. Right: the generalised plan created by lifting the plan actions, regressed states, and the goal condition.

goal by splitting conjunctive goals into singleton goals, under the observation that lifted policies are agnostic to the counting and unique identification of objects. Relaxing the policy synthesis problem parallels the planning technique of relaxing the problem to derive tractable and powerful heuristics for solving planning problems [Hoffmann and Nebel, 2001; Edelkamp, 2002; Helmert and Domshlak, 2009]. Lastly, to optimise execution costs, we leverage *database* algorithms in order to query and execute lifted policy rules efficiently.

We also theoretically formalise the conditions under which MOOSE learns sound and complete generalised plans as well as pruning rules which preserve optimal solutions when used in search. This is done by providing a refined classification of planning domains via goal decoupling, and an equivalence of planning problems via object isomorphism. We then conduct experiments on Easy-to-Solve, Hard-to-Optimise (ESHO) planning domains, P-time solvable and NP-hard to solve optimally domains. Experimental results show that MOOSE solves 33/61/23% more planning problems than state-of-the-art planners for classical/numeric/optimal planning, respectively. We summarily list our contributions as follows.

- We introduce MOOSE, a generalised planner for synthesising and executing generalised plans, by making use of regression rewriting, relaxation, and database algorithms.
- We formalise the conditions under which MOOSE is sound and complete for satisficing and optimal planning.
- We conduct experiments and demonstrate the effectiveness of MOOSE in satisficing, optimal and numeric planning settings relative to state-of-the-art planners.

2 Planning Background and Notation

We adopt standard notation and terminology for representing planning problems via the Planning Domain Definition Language (PDDL) [McDermott *et al.*, 1998; Haslum *et al.*, 2019]. A planning problem is represented by two components: a domain, consisting of lifted predicates and action schemata describing the action theory, and a problem specification, consisting of a finite set of objects, an initial state, and a goal condition. We begin with some general notation.

Mathematical Notation Let $\mathbb{N}/\mathbb{N}_0$ denote the natural numbers excluding/including 0, $\vec{\alpha}$ denote an ordered sequence of items, $\vec{\alpha}_i$ denote the ith element of $\vec{\alpha}$, $\vec{\alpha}_{[i:]}$ all elements from the ith element onwards in $\vec{\alpha}$ inclusive, and $|s|$ the size of a set or length of a sequence s.

Definition 1 (Planning Domain)**.** A planning domain is a tuple $\mathbb{D} = \langle \mathcal{P}, \mathcal{C}, \mathcal{A} \rangle$, where $\mathcal{P}$ is a set of predicates, $\mathcal{C}$ a set of constant objects, and $\mathcal{A}$ a set of action schemata. A predicate $p \in \mathcal{P}$ has a set of argument terms $x_1, \ldots, x_{n_p}$ where $n_p \in \mathbb{N}_0$ depends on p. An action schema $a \in \mathcal{A}$ is a tuple

$$a = \langle \mathit{var}(a), \mathit{pre}(a), \mathit{add}(a), \mathit{del}(a) \rangle$$

where $\mathit{var}(a)$ is a set of parameter variables, and preconditions $\mathit{pre}(a)$, add $\mathit{add}(a)$ and delete $\mathit{del}(a)$ effects are finite sets of predicates from $\mathcal{P}$ with arguments instantiated with variables or objects from $\mathit{var}(a) \cup \mathcal{C}$.

Definition 2 (Planning Problem)**.** A planning problem is a tuple $\mathbf{P} = \langle \mathbb{D}, s_0, g, \mathcal{O} \rangle$ where $\mathbb{D} = \langle \mathcal{P}, \mathcal{C}, \mathcal{A} \rangle$ is a planning domain, s_0 the initial state, g the goal condition, and $\mathcal{O} \supseteq \mathcal{C}$ a finite set of objects. A (ground) atom is a predicate whose argument terms are all instantiated with objects. A state in a planning problem is a set of atoms and operate under the closed world assumption: any atom not in a state is presumed false. The initial state s_0 and goal condition g are both sets of atoms. A state s is a goal state if $g \subseteq s$. We say that the problem $\mathbf{P}$ belongs to the domain $\mathbb{D}$.

A (ground) action is an action schema a where each parameter term is instantiated with an object, denoted $a(o_1, \ldots, o_n)$ with $o_1, \ldots, o_n \in \mathcal{O}$. An action a is applicable in a state s if $\mathit{pre}(a) \subseteq s$, in which case we define the successor

$$\mathrm{succ}(s, a) = (s \setminus \mathit{del}(a)) \cup \mathit{add}(a). \tag{1}$$

Otherwise, a is not applicable in s and $\mathrm{succ}(s, a) = \bot$. A plan for a planning problem is a finite sequence of actions $\vec{\alpha} = a_1, \ldots, a_n$ where $s_i = \mathrm{succ}(s_{i-1}, a_i) \neq \bot$ for $i = 1, \ldots, n$ and s_n is a goal state. We overload the notation of successor for sequences of actions with $\mathrm{succ}(s, \vec{\alpha}) = s_n$ as if $s = s_0$. The length of a plan $\vec{\alpha}$ is the number of actions it

contains. A problem $\mathbf{P}$ is solvable if a plan exists for $\mathbf{P}$. Satisficing (resp. optimal) planning refers to the task of finding any plan (resp. any plan with the lowest length) for $\mathbf{P}$.

Notational Shorthands We now introduce a few additional notational shorthands which will be useful for later. Let $\mathbf{P}[\omega]$ denote the ω component of a problem $\mathbf{P}$; e.g., $\mathbf{P}[s_0]$ is the initial state of $\mathbf{P}$. Furthermore, given a state s and set of atoms g' for a problem $\mathbf{P}$, let $\mathbf{P}_s = \langle \mathbb{D}, s, \mathbf{P}[g], \mathbf{P}[\mathcal{O}] \rangle$ denote the same problem with the initial state replaced with s, and $\mathbf{P}_{s,g'} = \langle \mathbb{D}, s, g', \mathbf{P}[\mathcal{O}] \rangle$ denote the same problem with the initial state and goal replaced with s and g'.

Goal Regression Goal regression refers to the act of computing the preimage of a goal under a sequence of actions via regression rewriting [Fikes *et al.*, 1972; Waldinger, 1977; Reiter, 1991; Reiter, 2001]. It is a powerful technique for computing the minimal set of goal relevant atoms. It has been used for heuristic synthesis [Bonet and Geffner, 2001; Scala *et al.*, 2016], plan monitoring [Fritz and McIlraith, 2007], policy synthesis for lifted Markov decision processes [Gretton and Thiébaux, 2004; Sanner and Boutilier, 2006; Sanner and Boutilier, 2009], nondeterministic planning [Muise *et al.*, 2012; Muise *et al.*, 2024], symbolic search [Pang and Holte, 2011; Alcázar *et al.*, 2013; Torralba, 2015; Speck *et al.*, 2025], numeric planning [Illanes and McIlraith, 2017], generating macro-actions [Hofmann *et al.*, 2020], and GP [Illanes and McIlraith, 2019; Yang *et al.*, 2022].

Definition 3 (STRIPS Regression [Rintanen, 2008])**.** A set of atoms g is regressable over an action a if $\mathit{add}(a) \cap g \neq \emptyset$ and $\mathit{del}(a) \cap g = \emptyset$, in which case we define the regression

$$\mathrm{regr}(g, a) = (g \setminus \mathit{add}(a)) \cup \mathit{pre}(a). \tag{2}$$

Otherwise, g is not regressable over a and $\mathrm{regr}(g, a) = \bot$.

3 Generalised Planning via Goal Regression

We first introduce the generalised planning (GP) problem as a set of planning problems sharing the same domain following from [Levesque, 2005; Srivastava *et al.*, 2008; Bonet *et al.*, 2009; Srivastava *et al.*, 2011a; Hu and De Giacomo, 2011; Celorrio *et al.*, 2019; Segovia-Aguas *et al.*, 2024].

Definition 4 (Generalised Planning Problem)**.** A generalised planning (GP) problem is a tuple of a domain and a possibly infinite set of solvable planning problems $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ where all planning problems $\mathbf{P} \in \mathbb{P}$ belong to $\mathbb{D}$.

A generalised plan π for a GP problem $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ is a programmatic plan that can be instantiated on any planning problem $\mathbf{P} \in \mathbb{P}$ to return a plan $\pi(\mathbf{P}) = \vec{\alpha}$ for $\mathbf{P}$. We call a generalised plan π a **solution** if for every $\mathbf{P} \in \mathbb{P}$, the plan $\pi(\mathbf{P})$ is a valid solution for $\mathbf{P}$. Examples of programmatic plans may include (memoryless) finite state controllers [Bonet *et al.*, 2009; Bonet *et al.*, 2010; Hu and De Giacomo, 2011; Aguas *et al.*, 2018], policies derived from lifted rules [Srivastava *et al.*, 2011b; Illanes and McIlraith, 2019; Francès *et al.*, 2021] and general-purpose programs [Levesque, 2005; Srivastava *et al.*, 2008; Segovia-Aguas *et al.*, 2024; Silver *et al.*, 2024]. Figure 1 illustrates the pipeline with which one synthesises generalised plans. In practice, the generalised plan is synthesised using a representative subset of the problems in $\mathbb{P}$, i.e., $\mathbb{P}_{\text{train}} = \{\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(n)}\} \subseteq \mathbb{P}$. We will later discuss the selection of this subset and the implications it has for the completeness of the resultant generalised plan of our approach.

MOOSE Program We name our generalised plans as MOOSE programs. MOOSE programs are found in a two-step process as described in Section 3.1: (a) decompose the set of training problems $\mathbb{P}_{\text{train}}$ into smaller problems constituting singleton goal conditions and generate optimal plans for each in order, and (b) apply goal regression from the singleton goals using the order of the optimal plans found in (a) to generate a set of lifted rules. MOOSE programs can be instantiated into a plan for a problem by deriving an action from the rule set at every state until the goal is reached (Section 3.2), or used to guide search for optimal planning (Section 3.3).

MOOSE programs are sets of lifted rules which indicate an action or macro action to execute, conditioned on a partial state and a goal that has not yet been achieved. A distinct feature of such rules is that the antecedent of a single rule compactly captures a set of states. Lifted rules can then be grounded on states if their antecedent condition is satisfied. Our rules are similar to existing lifted rules [Khardon, 1999; Illanes and McIlraith, 2019; Yang *et al.*, 2022] with the extension that we may now have macro actions in rule heads. Furthermore, each rule has an associated precedence value which determines its execution priority as is common in logic programming. Figure 2 illustrates the synthesis procedure and structure of MOOSE programs.

Definition 5 (MOOSE Rule)**.** Let $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ be a GP problem. A MOOSE rule r is a tuple

$$r = \langle \mathit{var}(r), \mathit{stateCond}(r), \mathit{goalCond}(r), \mathit{actions}(r) \rangle$$

where $\mathit{var}(r)$ is a finite set of free variables, $\mathit{stateCond}(r)$ and $\mathit{goalCond}(r)$ are finite sets of predicates instantiated with terms in $\mathit{var}(r)$, and $\mathit{actions}(r)$ is a finite sequence of action schemata instantiated with terms in $\mathit{var}(r)$.

Definition 6 (Grounding)**.** Let r be a MOOSE rule, $\mathbf{P}$ be a problem and s a state in the state space of $\mathbf{P}$. A grounding of r in s is an assignment of objects to variables $f : \mathit{var}(r) \to \mathbf{P}[\mathcal{O}]$ such that $\mathit{stateCond}(r)|_f \subseteq s$ and $\mathit{goalCond}(r)|_f \subseteq (\mathbf{P}[g] \setminus s)$, the set of goal atoms not yet achieved. The $|_f$ notation denotes replacing every occurrence of a free variable term with the corresponding object in f. In the case that a grounding f exists, we define the nondeterministic function

$$\mathrm{grounding}(r, s, \mathbf{P}[g]) = \mathit{actions}(r)|_f, \tag{3}$$

where f is some grounding. Otherwise, if no grounding exists then $\mathrm{grounding}(r, s, \mathbf{P}[g]) = \bot$.

Definition 7 (MOOSE Program)**.** A MOOSE program π is a set of MOOSE rules $\mathcal{R}$ and a function $\mathcal{R} \to \mathbb{N}$ representing a precedence ranking on the rules for execution.

As to be described later, if several rules are applicable in a given state, the rule with the lowest precedence ranking with ties broken arbitrarily is chosen for execution. Relatedly, Yang *et al.* [2022] specify a total order on policy rules, whereas MOOSE specifies more relaxed partial order. Next, we define lifting of a ground plan and set of atoms to a set of

Algorithm 1: MOOSE Program Synthesis

Input: Training problems $\mathbb{P}_{\text{train}} = \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(n_t)}$, and number of goal permutations $n_p \in \mathbb{N}$ (default: 3).
Output: MOOSE program π.

1 $\pi \leftarrow \emptyset$
2 **for** $i = 1, \ldots, n_t$ **do**
3 $\quad n_g \leftarrow |\mathbf{P}^{(i)}[g]|$
4 $\quad$ **for** $j = 1, \ldots, \min(n_p, n_g!)$ **do**
5 $\quad\quad s \leftarrow \mathbf{P}^{(i)}[s_0]$
6 $\quad\quad \vec{g} \leftarrow \text{newPermutation}(\mathbf{P}^{(i)}[g])$
7 $\quad\quad$ **for** $k = 1, \ldots, n_g$ **do**
8 $\quad\quad\quad g' \leftarrow \{\vec{g}_k\}$
9 $\quad\quad\quad \vec{\alpha} \leftarrow \text{optimalPlan}(\mathbf{P}^{(i)}_{s,g'})$
10 $\quad\quad\quad$ **if** $\vec{\alpha} = \bot$ **then continue**
11 $\quad\quad\quad \pi \leftarrow \pi \cup \text{extractRules}(\vec{\alpha}, g')$ `// Alg. 2`
12 $\quad\quad\quad s \leftarrow \text{succ}(s, \vec{\alpha})$ `// Eqn. (1) for plans`
13 **return** π

Algorithm 2: Rule Extraction Routine

Input: Sequence of actions $\vec{\alpha}$ and set of atoms g.
Output: MOOSE rules with precedence values π.

1 $\pi \leftarrow \emptyset; s \leftarrow g$
2 **for** $i = |\vec{\alpha}|, \ldots, 1$ **do**
3 $\quad s \leftarrow \text{regr}(s, \vec{\alpha}_i)$ `// Eqn. (2)`
4 $\quad r \leftarrow \text{lift}(s, g, \vec{\alpha}_{[i:]})$ `// Eqn. (4)`
5 $\quad \pi \leftarrow \pi \cup \{(r, |\vec{\alpha}| - i + 1)\}$
6 **return** π

quantified actions and predicates. Lifting will be used in the synthesis module to generate reusable rules.

Definition 8 (Lifting)**.** Let s and g be finite sets of ground atoms and $\vec{\alpha} = a_1, \ldots, a_m$ a sequence of ground actions. Then let $o_1, \ldots, o_q$ be the union of all objects from the atoms and actions that are not in $\mathcal{C}$. Next we define the set of free variable terms $\mathit{var} = \{v_1, \ldots, v_q\}$ and lift each action and atom by replacing each constant o_i with its corresponding free variable v_i in *var*. We denote

$$\text{lift}(s, g, \vec{\alpha}) = \langle \mathit{var}, s', g', \vec{\alpha}' \rangle \tag{4}$$

where $\vec{\alpha}'$ is the sequence of ground actions lifted by variables in *var*, and similarly for s' and g' the sets of lifted atoms.

3.1 Synthesising MOOSE Programs

Algorithm 1 summaries the main MOOSE program synthesis procedure. The input is a set of unlabelled training problems and a number n_p representing the effort spent on extracting information from a single problem. The main idea is that the problem is relaxed by decoupling the goals and greedily solving them optimally and in order. Each resulting plan is used to regress the corresponding singleton goal, and the regressed goal and plan is then lifted into a lifted macro action rule.

The main algorithm gradually builds an empty rule set (Line 1) by iterating over all training problems (Line 2) and the specified number n_p of goal orderings (Line 4). For each goal ordering and training problem, MOOSE finds plans via an optimal planner (Line 9) with the singleton goals (Line 8)

Algorithm 3: MOOSE Program Instantiation

Input: A planning problem $\mathbf{P}$ and MOOSE program π.
Output: A plan $\vec{\alpha}$ and success or failure status.

1 $s \leftarrow \mathbf{P}[s_0]$
2 $\vec{\alpha} \leftarrow []$ `// empty sequence`
3 **while** $\mathbf{P}[g] \not\subseteq s$ **do**
4 $\quad \vec{\beta} \leftarrow \bot$
5 $\quad$ **for** $r \in \pi$ *in ascending precedence values* **do**
6 $\quad\quad \vec{\beta} \leftarrow \text{grounding}(r, s, \mathbf{P}[g])$ `// Eqn. (3)`
7 $\quad\quad$ **if** $\vec{\beta} \neq \bot$ **then break**
8 $\quad$ **if** $\vec{\beta} = \bot$ *or detected cycle* **then return** $\vec{\alpha}$, failure
9 $\quad \vec{\alpha} \leftarrow \vec{\alpha}; \vec{\beta}$ `// sequence concatenation`
10 $\quad s \leftarrow \text{succ}(s, \vec{\beta})$
11 **return** $\vec{\alpha}$, success

in order (Line 7), while progressing the current state along the way (Line 12). If a plan exists, then we extract rules from the plan and add them to the incumbent plan (Line 11) as we will describe in more detail in the next paragraph. Otherwise if no plan exists, i.e. if the problem is unsolvable with the current state and singleton goal pair, no rules are extracted and the state is not progressed (Line 10).

Algorithm 2 describes how to extract lifted rules from a plan trace and set of goal atoms. It begins by initialising the to-be-regressed state s by the goal (Line 1). Next, it regresses s in reverse order of the plan $\vec{\alpha}$ (Lines 2 to 3) and then lifts (Line 4) the corresponding regressed state s, goal g, and suffix of the plan into a rule r. Then we compute the precedence value of the rule to be the cost-to-go from the partial state s to the goal g with respect to the input plan suffix and append it to the incumbent plan (Line 5).

3.2 Satisficing Planning via Policy Execution

A learned MOOSE program can be used for satisficing planning by repeatedly choosing and executing a rule to progress the initial state to a goal state. Algorithm 3 summaries the execution procedure for an input planning problem and MOOSE program. Summarily, each iteration of the algorithm's main loop queries the set of rules in order of ascending precedence values until a rule associated with goals not yet achieved can be grounded (Lines 5 to 7), from which the corresponding macro action is added to the incumbent plan and applied to the current state (Lines 9 to 10). The loop breaks once the goal is reached (Line 3), no actions can be queried from the set of rules, or a cycle is encountered (Line 8).

3.3 Optimal Planning via Action Pruning

Optimal planning can be performed via a synthesised MOOSE program by extending the corresponding planning problem with MOOSE rules. The rules, ignoring precedence values, are encoded into PDDL axioms [Thiébaux *et al.*, 2005] representing search control for optimal planners that support axioms. Theorem 22 later formalises conditions under which encodings of MOOSE programs preserve optimal solutions.

We now extend a given GP domain $\mathbb{D} = \langle \mathcal{P}, \mathcal{C}, \mathcal{A} \rangle$ with a MOOSE program π. Firstly, we add predicates p_g and p_{goalCond} for each $p \in \mathcal{P}$, representing goals in a planning

problem and goals that not been achieved in the state, respectively. Then each state in a problem $\mathbf{P}$ is extended with atoms $p_g(\vec{o})$ for each goal atom $p(\vec{o}) \in \mathbf{P}[\mathcal{G}]$ following Martín and Geffner [2004]. For each predicate p we introduce the axiom

$$p_{goalCond}(\vec{x}) \leftarrow p_g(\vec{x}) \wedge \neg p(\vec{x})$$

for computing unachieved goals. Secondly, for each action schema $a \in \mathcal{A}$ we add a new predicate a_π to $\mathcal{P}$ and $pre(a)$. Then for each MOOSE rule $\langle \vec{x}, stateCond, goalCond, \vec{\alpha} \rangle$, we introduce an axiom

$$(\vec{\alpha}_1)_\pi(\vec{x}) \leftarrow \bigwedge_{p(\vec{y}) \in stateCond} p(\vec{y}) \wedge \bigwedge_{p(\vec{y}) \in goalCond} p_{goalCond}(\vec{y})$$

for restricting the application of an action with the MOOSE rule condition. In this way, the axioms restrict the set of applicable actions at any ground state to the first action of each macro action that the MOOSE rules would generate, and hence prune the entire search space. Differently to previous works that prune the search space with lifted rules [Bacchus and Kabanza, 2000; Yoon *et al.*, 2008; Krajnanský *et al.*, 2014], our approach does not require writing new solvers but making use of existing planners that support PDDL axioms.

4 Soundness and Completeness Conditions

In this section, we provide theoretical results concerning the soundness and completeness of MOOSE programs for both satisficing and optimal planning (Theorems 19 and 22). Proofs of all statements are provided in Appendix B. The idea is that under assumptions on the complexity of a GP problem $\langle \mathbb{D}, \mathbb{P} \rangle$ and given sufficient training problems, Algorithm 1 synthesises generalised plans that are sound and complete for solving problems in $\mathbb{P}$, and furthermore finds optimal plans when MOOSE rules are used for search as described in Section 3.3. We begin by classifying planning domains based on the separability of goals, refining the class of Easy-to-Solve, Hard-to-Optimise (ESHO) planning domains, the class of P-time solvable and NP-hard to solve optimally domains.

Goal Independence Early works in planning worked under the assumption that conjunctive goals can be split apart into their individual components and achieved independently. The Sussman [1973] anomaly illustrates a simple Blocksworld example for how this was not true in general, giving rise to algorithms which aim to achieve goals simultaneously [Waldinger, 1977] and to provably complete algorithms in the current planning age. Regardless, most planning domains are P-time solvable and furthermore exhibit goals that can be achieved independently from one another. In this section, we formalise two variants of goal independence (TGI and SGI) and provide their computational complexity.

Definition 9 (True Goal Independence)**.** A planning problem $\mathbf{P}$ exhibits true goal independence (TGI) if for all orderings $\vec{g}$ of goal atoms in $\mathbf{P}[g]$, the following greedy algorithm is sound and complete: (1) set $s = \mathbf{P}[s_0]$ and then (2) iterate over goal atoms $\vec{g}_i$ in $\vec{g}$ in order by (2a) finding any optimal plan $\vec{\alpha}^{(i)}$ from s to a goal state containing $\vec{g}_i$ and (2b) progressing s via $\vec{\alpha}^{(i)}$. We say that $\mathbf{P}$ exhibits polynomial TGI (pTGI) if step (2a) can run in polynomial time. Lastly, we say that $\mathbf{P}$ exhibits TGI with respect to $C \in \mathbb{N}$, denoted TGI$_C$, if all optimal plans in step (2a) have plan length bounded by C.

Definition 10 (Sequential Goal Independence)**.** A planning problem $\mathbf{P}$ exhibits sequential goal independence (SGI) if there exists an ordering $\vec{g}$ of goals $\mathbf{P}[g]$ such that the greedy algorithm operating on $\vec{g}$ is sound and complete. Similarly, we say that $\mathbf{P}$ exhibits polynomial SGI (pSGI) if step (2a) in the greedy algorithm runs in polynomial time.

Next, we say that a GP problem $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ exhibits TGI if every problem in $\mathbb{P}$ exhibits TGI, and analogously for SGI. We then let PLANSAT($\mathbf{GP}$) denote the computational problem of deciding if a plan exists for a problem in $\mathbb{P}$. The following proposition shows that without the polynomial time constraint of step (2a) of the aforementioned greedy algorithm, TGI and SGI do not make planning any easier.

Proposition 11. PLANSAT*(*$\mathbf{GP}$*) of a GP problem* $\mathbf{GP}$ *exhibiting TGI is* PSPACE*-complete.*

Corollary 12. PLANSAT*(*$\mathbf{GP}$*) of a GP problem* $\mathbf{GP}$ *exhibiting SGI is* PSPACE*-complete.*

Once we add the polynomial time constraint of step (2a), both pTGI and pSGI become easier. However, only pTGI becomes tractable while pSGI becomes NP-complete.

Proposition 13. PLANSAT*(*$\mathbf{GP}$*) of a GP problem* $\mathbf{GP}$ *exhibiting pTGI is in* PTIME*.*

Corollary 14. *Let* $C \in \mathbb{N}$*.* PLANSAT*(*$\mathbf{GP}$*) of a GP problem* $\mathbf{GP}$ *exhibiting TGI*$_C$ *is in* PTIME*.*

Proposition 15. PLANSAT*(*$\mathbf{GP}$*) of a GP problem* $\mathbf{GP}$ *exhibiting pSGI is* NP*-complete.*

Planning Problem Equivalence Before we state the assumptions required for MOOSE to synthesise sound and complete generalised plans, we define the notion of equivalence for (lifted) problems. We define equivalence via bijection between objects, in contrast to work by Sievers *et al.* [2019] which reduce problems to graph automorphisms.

Definition 16 (Equivalence Relation)**.** Given a GP problem $\langle \mathbb{D}, \mathbb{P} \rangle$, we define a relation $\sim_U$ on planning problems in $\mathbb{P}$ by $\mathbf{P}_1 \sim_U \mathbf{P}_2$ if there exists a bijective mapping $f : \mathbf{P}_1[\mathcal{O}] \to \mathbf{P}_2[\mathcal{O}]$ such that $f(c) = c$ for $c \in \mathcal{C}$, $F(\mathbf{P}_1[s_0]) = \mathbf{P}_2[s_0]$ and $F(\mathbf{P}_1[g]) = \mathbf{P}_2[g]$ where $F(s) := \{p(f(o_1), \ldots, f(o_n)) \mid p(o_1, \ldots, o_n) \in s\}$.

Indeed the defined relation is an equivalence relation and furthermore defines a natural notion of equivalence for planning problems, where reflexivity, symmetry and transitivity follows from bijective functions in the definition of $\sim_U$.

Proposition 17. *The relation* $\sim_U$ *on planning problems of any given GP problem is an equivalence relation.*

Proposition 18. *Suppose* $\mathbf{P}_1 \sim_U \mathbf{P}_2$ *and let* $f : \mathbf{P}_1[\mathcal{O}] \to \mathbf{P}_2[\mathcal{O}]$ *be the bijective mapping satisfying the definition of* $\sim_U$*. Then a sequence of actions* $a_1, \ldots, a_n$ *is a plan for* $\mathbf{P}_1$ *if and only if* $a'_1, \ldots, a'_n$ *is a plan for* $\mathbf{P}_2$*, where* a'_i *is defined by* $a'_i = a(f(o_1), \ldots, f(o_n))$ *if* $a_i = a(o_1, \ldots, o_n)$ *for some* $a \in \mathcal{A}$ *and* $o_1, \ldots, o_n \in \mathbf{P}_1[\mathcal{O}]$*.*

Soundness and Completeness of MOOSE Now we state and prove the main theorems of the section. The main idea of the statement is that given enough training data MOOSE can construct a database of rules for TGI$_C$ problems which can

solve all possible problems with singleton goals. By assuming a bound in the definition of TGI_C of plan lengths, this database has a finite size.

Theorem 19 (MOOSE is sound and complete for PLANSAT)**.** *Let $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ be a GP problem exhibiting TGI_C. There exists a set of training problems $\mathbb{P}_{train}$ where Algorithm 3 using the plan Π synthesised from Algorithm 1 with $\mathbb{P}_{train}$ is sound and complete for $\mathbb{P}$ for satisficing planning.*

The bound on the size of training problems in the proof of Theorem 19 is exponential in the domain size. This bound may be tight and unavoidable given that it has been shown that GP under the QNP [Srivastava *et al.*, 2011b] framework is provably equivalent to FOND planning [Bonet and Geffner, 2020] which is known to be EXPTIME-complete. A fruitful next step is to develop a sound and complete learning algorithm to learns to generate and select what training data is required, possibly given implicitly in the input GP problem [Srivastava *et al.*, 2011a; Grundke *et al.*, 2024].

To prove optimality we require an additional assumption which we see is no harder than SGI.

Definition 20 (Optimal Goal Independence)**.** A planning problem exhibits Optimal Goal Independence (OGI) if there *exists* an ordering $\vec{g}$ of goals $\mathbf{P}[g]$ such that the algorithm described in Definition 9 is optimally sound and complete with the change that it is now nondeterministic and step (2a') is changed to "*guess* an optimal plan $\vec{\alpha}^{(i)}$ such that the concatenation of plans is optimal".

Proposition 21. *Let $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ be a GP problem exhibiting OGI. The computational problem* PLANOPT*$(\mathbf{GP})$ of deciding if there exists a plan for any problem in $\mathbb{P}$ with length less than an input k is* NP*-complete.*

Now we state the main theorem for optimal planning. We also provide a counterexample to the theorem for when the OGI assumption is dropped in Example 1 in the Appendix.

Theorem 22 (MOOSE is sound and complete for PLANOPT)**.** *Let $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ be a GP problem exhibiting TGI_C and OGI. There exists a set of training problems $\mathbb{P}_{train}$ where an optimal planner run on the transformation in Section 3.3 via the generalised plan Π learned from Algorithm 1 with $\mathbb{P}_{train}$ is sound and complete for $\mathbb{P}$ for optimal planning.*

5 Experiments

In this section we aim to understand how MOOSE places in performance in comparison to state-of-the-art planners. We first describe the experimental setup as follows.

Implementation We implement MOOSE from scratch in Python, but make use of the following tools: the `pddl` [Favorito *et al.*, 2025] parser for parsing PDDL problems; the `SQLite` [Hipp, 2020] database system for grounding in Algorithm 3 as planning states can be viewed as databases and rules as queries [Corrêa *et al.*, 2020; Corrêa and De Giacomo, 2024]; (NUMERIC) FAST DOWNWARD's implementation of A* with the (Numeric) LM-cut heuristic [Helmert and Domshlak, 2009; Kuroiwa *et al.*, 2022] for generating (numeric) optimal plans in Line 9 of Algorithm 1; and SYMK [Speck *et al.*, 2019; Speck *et al.*, 2025] for optimal planning with MOOSE as described in Section 3.3.

Additional Extensions We introduce an optimisation for generalised plan instantiation which tries to fire the previous successfully fired rule first during Line 9 of Algorithm 3. Furthermore, we introduce a validation and refinement procedure for handling domains which do not exhibit the TGI_C assumption in Appendix C. We also extend our approach to handle numeric planning problems formalised in Appendix A.

Benchmarks For classical planning, we make use of the training and testing problems of the 2023 International Planning Competition (IPC) [Taitler *et al.*, 2024] Learning Track for the Childsnack, Ferry, Miconic, Rover, Satellite and Transport domains, and also introduce new training and testing problems for the Barman and Logistics domains. For numeric planning, we make use of the training and testing problems from benchmarks by [Chen and Thiébaux, 2024a] for Childsnack, Ferry, Miconic, and Transport, and introduce new problems for the PDDL Minecraft domain [Benyamin *et al.*, 2024], as well as a new planning domain 'Adultsnack' which extends Childsnack to an arbitrary number of dietary restrictions. We modified Rover and Transport to remove their path-finding component. All domains are ESHO, P-time to solve but NP-hard to solve optimally, with the exception of Adultsnack and Childsnack which are also P-time to solve optimally. Detailed distributions of training and testing problem object sizes are provided in Appendix D.

Baselines For classical planning, we compare against LAMA [Richter and Westphal, 2010], the state-of-the-art standalone satisficing planner in the 2023 IPC Learning Track, and the sketch learner (SLEARN) [Drexler *et al.*, 2022] with width in $\{0, 1, 2\}$. In our results, we take the best SLEARN configuration for every problem. For numeric planning, we compare against the multi-queue (M$(3h\|3n)$) [Chen and Thiébaux, 2024b] and the multi-repetition relaxed plan heuristic with jumping actions (MRP+HJ) [Scala *et al.*, 2020] configurations of ENHSP. For optimal planning, we compare against blind A* search, A* search with the LM-cut heuristic, SCORPION [Seipp *et al.*, 2020], the state-of-the-art standalone optimal planner according to the 2023 IPC Optimal Track, and SYMK [Speck *et al.*, 2025].

Resources For synthesis, satisficing, and optimal planning, we gave MOOSE and all baselines an 8GB memory and 1800s runtime limit, and ran them on a cluster with Intel Xeon Platinum 8274 CPUs (3.20GHz). An exception is that SLEARN was given *more* compute for synthesis (32GB memory and 12 hours for each width configuration). For all experiments, we set the only hyperparameter in the MOOSE learning procedure in Algorithm 1 to be $n_p = 3$.

Results We first note that MOOSE completed synthesis in the given synthesis budgets while SLEARN failed to learn domain knowledge for 3 domains despite having access to substantially more compute and multiple configurations. Figure 3 summarises synthesis metrics and planner performance and we refer to Appendix E for more detailed results.

Satisficing Numeric Planning MOOSE solves all 540 numeric testing problems. Figure 3 shows that each problem is solved in under 100 seconds. In comparison, state-of-the-art, numeric planners solve fewer than two thirds of the bench-

	Time (s)		Memory (GB)	
	MOOSE	SLEARN	MOOSE	SLEARN
Ba	380	–	< 1	–
Ch	113	5593	< 1	2
Fe	17	15	< 1	< 1
Lo	92	–	< 1	–
Mi	18	16	< 1	< 1
Ro	667	–	< 1	–
Sa	530	5964	< 1	6
Tr	23	30	< 1	< 1

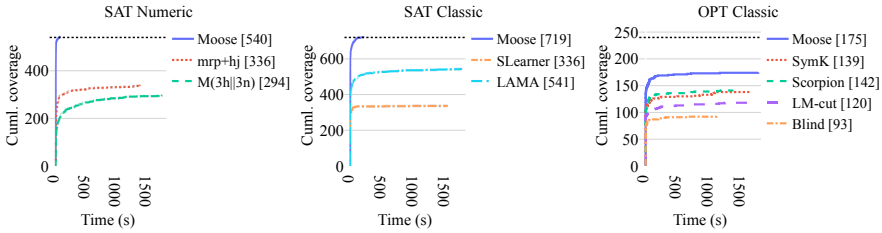

Figure 3: Left: synthesis time and memory usage (↓) of MOOSE and SLEARN. We present the lowest synthesis metrics over multiple SLEARN configurations, while MOOSE has only a single configuration. Cells marked – indicate that synthesis failed with the given budgets. Right: cumulative coverage (↑) of planners over time for various planning settings. The dotted black line indicates the total number of problems.

mark problems with the 1800s limit. MOOSE achieves the highest quality plans for Adultsnack and Childsnack, while plan quality is similar to the baselines for all other domains.

Satisficing Classical Planning MOOSE solves 719 out of 720 classical testing problems, outperforming all tested baselines. The generalised planner SLEARN manages to learn domain knowledge for 5 out of 8 domains but only solves all problems from one domain (Miconic). From Figure 27 in the Appendix, MOOSE has better/worse plans than LAMA on 5/3 domains (Barman, Childsnack, Ferry, Rover and Transport)/ (Logistics, Miconic and Satellite). The single failure was due to the rules not returning an action for a (rare) Logistics state not captured in the training or elsewhere in the testing set. However, when rules are queried in descending instead of ascending order of precedence values in Line 5 of Algorithm 3, the state is avoided and this problem is solved.

Optimal Classical Planning MOOSE solves 175 out of 240 classical testing problems optimally. MOOSE achieves the best (tied) performance for 5 domains out of 8, while the next best optimal planner, Scorpion, achieves the best (tied) performance for 4 domains. We also note that MOOSE matches or outperforms the base planner SYMK on all domains. This fact suggests that the reduction in search space from encoding learned policies via axioms outweighs the cost of evaluating such axioms. Although it is not guaranteed that the transformation from policies learned from finite training data preserves optimality, the plans output by MOOSE are empirically optimal in comparison to plans output by other optimal planners. We also note that MOOSE learns provably optimal *policies* for Num-Adultsnack and (Num-)Childsnack.

6 Related Work

Gretton and Thiébaux [2004] employed regression rewriting for GP in the context of lifted MDPs. Lifted regression was used to generate relevant features for building decision-tree policies via inductive logic programming. Their approach handles optimal, probabilistic planning although the horizons of testing problems were bounded by those seen in the training set. LOOM [Illanes and McIlraith, 2019] is the work that is most closely related to MOOSE theory-wise. LOOM automatically synthesises an abstraction from a single planning problem of a GP problem via bagging equivalent objects [Fuentetaja and de la Rosa, 2016; Riddle *et al.*, 2016; Dong *et al.*, 2025] into a nondeterministic, quantified problem. The quantified problem is then solved with an extension of the PRP planner [Muise *et al.*, 2012] to synthesise generalisable policies via regression which satisfy a policy termination test [Srivastava *et al.*, 2011b]. MOOSE takes inspiration from the powerful regression rewriting technique employed in these works, but differs in the methodology. MOOSE takes a *bottom-up* approach of performing ground regression from example plans to generate ground condition-action pairs that are then lifted into rules. Gretton and Thiébaux takes a *top-down* approach of performing lifted regression to generate relevant lifted features, and LOOM uses ground regression on a top-down abstraction of the GP problem for synthesising generalised policies. Furthermore, MOOSE implicitly generalises over goals expressed by Skolem functions, whereas LOOM is restricted to Skolem constants (nullary Skolem functions) due to the dependency on bagging.

More generally, MOOSE lies in the class of generalised planners that synthesise generalised plans by sampling from training problems. Examples of recent work includes PG3 [Yang *et al.*, 2022] which performs heuristic search over a space of generalised policies [Segovia-Aguas *et al.*, 2021; Segovia-Aguas *et al.*, 2024] and also uses goal regression for handling 'missed' states. Toyer *et al.* [2018; 2020] introduced ASNets, the first deep learning approach for GP, which learns generalised policies with graph neural networks and training data. Chen *et al.* [2024] introduced WL-GOOSE which synthesised generalised plans as heuristic functions via statistical graph learning. Silver *et al.* [2024] employed language models for synthesising Python code as generalised plans. Celorrio *et al.* [2019] surveys more prior GP work which synthesise generalised plans from training problems.

7 Conclusion

We introduce a new generalised planner, MOOSE, for both satisficing and optimal planning based on the KRR concept of goal regression. We formally classify and define the classes of planning domains and problems for which MOOSE is sound and complete. Experimental results show that our approach significantly advances the state of the art for classical, numeric, and optimal (generalised) planning for a substantial class of planning domains. MOOSE is also memory efficient for both generalised plan synthesis and instantiation, taking less than 1GB of memory to perform synthesis and instantiation across all experiments. Future work involves handling domains requiring transitive closure computations and assumptions weaker than those introduced in the our theory.

Acknowledgements

The first two authors carried out this work while at the Vector Institute, where the first author was working as a Research Intern. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada CIFAR AI Chairs Program. The second author was supported by an RWTH IRS Theodore von Kármán Fellowship under the grant ID G:(DE-82)ZUK2-TvK-GSO101 and the Excellence Strategy of the Federal Government and the Länder, Germany. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

References

[Aguas *et al.*, 2018] Javier Segovia Aguas, Sergio Jiménez, and Anders Jonsson. Computing hierarchical finite state controllers with classical planning. *J. Artif. Intell. Res.*, 62:755–797, 2018.

[Alcázar *et al.*, 2013] Vidal Alcázar, Daniel Borrajo, Susana Fernández, and Raquel Fuentetaja. Revisiting regression in planning. In *IJCAI*, 2013.

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116:123–191, 2000.

[Benyamin *et al.*, 2024] Yarin Benyamin, Argaman Mordoch, Shahaf Shperberg, and Roni Stern. Solving minecraft tasks via model learning. In *PRL Workshop Series - Bridging the Gap Between AI Planning and Reinforcement Learning*, 2024.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129:5–33, 2001.

[Bonet and Geffner, 2020] Blai Bonet and Hector Geffner. Qualitative numeric planning: Reductions and complexity. *J. Artif. Intell. Res.*, 69:923–961, 2020.

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

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

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[Celorrio *et al.*, 2019] Sergio Jiménez Celorrio, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *Knowl. Eng. Rev.*, 34:e5, 2019.

[Chen and Thiébaux, 2024a] Dillon Z. Chen and Sylvie Thiébaux. Graph learning for numeric planning. In *NeurIPS*, 2024.

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

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

[Corrêa and De Giacomo, 2024] Augusto B. Corrêa and Giuseppe De Giacomo. Lifted planning: Recent advances in planning using first-order representations. In *IJCAI*, 2024.

[Corrêa *et al.*, 2020] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès. Lifted successor generation using query optimization techniques. In *ICAPS*, 2020.

[Dong *et al.*, 2025] Hao Dong, Zheyuan Shi, Hemeng Zeng, and Yongmei Liu. An automatic sound and complete abstraction method for generalized planning with baggable types. *AAAI*, 39(14):14875–14884, 2025.

[Drexler *et al.*, 2022] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *ICAPS*, 2022. Code accessed from commit `7a7ea6` in https://github.com/drexlerd/sketch-learner.

[Edelkamp, 2002] Stefan Edelkamp. Symbolic pattern databases in heuristic search planning. In *AIPS*, 2002.

[Erol *et al.*, 1995] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76:75–88, 1995.

[Favorito *et al.*, 2025] Marco Favorito, Francesco Fuggitti, and Christian Muise. pddl, 2025. Accessed from https://github.com/ai-Planning/pddl.

[Fikes *et al.*, 1972] Richard Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artif. Intell.*, 3(1-3):251–288, 1972.

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

[Francès *et al.*, 2021] Guillem Francès, Blai Bonet, and Hector Geffner. Learning general planning policies from small examples without supervision. In *AAAI*, 2021.

[Fritz and McIlraith, 2007] Christian Fritz and Sheila A. McIlraith. Monitoring plan optimality during execution. In *ICAPS*, 2007.

[Fuentetaja and de la Rosa, 2016] Raquel Fuentetaja and Tomás de la Rosa. Compiling irrelevant objects to counters. special case of creation planning. *AI Commun.*, 29(3):435–467, 2016.

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

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

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

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers, 2019.

[Hearn and Demaine, 2005] Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, 2009.

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

[Helmert, 2003] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artif. Intell.*, 143(2):219–262, 2003.

[Hipp, 2020] Dwayne Richard Hipp. Sqlite, 2020. Accessed from https://www.sqlite.org/index.html.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[Hofmann *et al.*, 2020] Till Hofmann, Tim Niemueller, and Gerhard Lakemeyer. Macro operator synthesis for ADL domains. In *ECAI*, 2020.

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

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

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

[Khardon, 1999] Roni Khardon. Learning action strategies for planning domains. *Artif. Intell.*, 113:125–148, 1999.

[Krajnanský *et al.*, 2014] Michal Krajnanský, Jörg Hoffmann, Olivier Buffet, and Alan Fern. Learning pruning rules for heuristic search planning. In *ECAI*, 2014.

[Kuroiwa *et al.*, 2022] 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.*, 75:1477–1548, 2022.

[Levesque, 2005] H. J. Levesque. Planning with loops. In *IJCAI*, 2005.

[Martín and Geffner, 2004] Mario Martín and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Appl. Intell.*, 20:9–19, 2004.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. PDDL-the planning domain definition language. Technical report, 1998.

[Muise *et al.*, 2012] Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*, 2012.

[Muise *et al.*, 2024] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. PRP rebooted: Advancing the state of the art in FOND planning. In *AAAI*, 2024.

[Pang and Holte, 2011] Bo Pang and Robert C. Holte. State-set search. In *SOCS*, 2011.

[Reiter, 1991] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial and Mathematical Theory of Computation*. Academic Press, 1991.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.

[Riddle *et al.*, 2016] Pat Riddle, Jordan Douglas, Mike Barley, and Santiago Franco. Improving performance by reformulating PDDL into a bagged representation. In *Proceedings of the 8th Workshop on Heuristic Search for Domain-independent Planning*, 2016.

[Rintanen, 2008] Jussi Rintanen. Regression for classical and nondeterministic planning. In *ECAI*, 2008.

[Sanner and Boutilier, 2006] Scott Sanner and Craig Boutilier. Practical linear value-approximation techniques for first-order MDPs. In *UAI*, 2006.

[Sanner and Boutilier, 2009] Scott Sanner and Craig Boutilier. Practical solution techniques for first-order MDPs. *Artif. Intell.*, 173(5-6):748–788, 2009.

[Scala *et al.*, 2016] Enrico Scala, Patrik Haslum, and Sylvie Thiébaux. Heuristics for numeric planning via subgoaling. In *IJCAI*, 2016.

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

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

[Segovia-Aguas *et al.*, 2024] Javier Segovia-Aguas, Sergio Jiménez Celorrio, and Anders Jonsson. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artif. Intell.*, 330:104097, 2024.

[Seipp *et al.*, 2020] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *J. Artif. Intell. Res.*, 67:129–167, 2020.

[Sievers *et al.*, 2019] Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Theoretical foundations for structural symmetries of lifted PDDL tasks. In *ICAPS*, 2019.

[Silver *et al.*, 2024] 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.

[Speck *et al.*, 2019] David Speck, Florian Geißer, Robert Mattmüller, and Álvaro Torralba. Symbolic planning with axioms. In *ICAPS*, 2019.

[Speck *et al.*, 2025] David Speck, Jendrik Seipp, and Álvaro Torralba. Symbolic search for cost-optimal planning with expressive model extensions. *J. Artif. Intell. Res.*, 82:1349–1405, 2025.

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

[Srivastava *et al.*, 2011a] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2):615–647, 2011.

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

[Sussman, 1973] Gerald Jay Sussman. *A Computational Model of Skill Acquisition*. PhD thesis, MIT, 1973.

[Taitler *et al.*, 2024] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fiser, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 international planning competition. *AI Mag.*, 45:280–296, 2024.

[Thiébaux *et al.*, 2005] Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artif. Intell.*, 168:38–69, 2005.

[Torralba, 2015] Álvaro Torralba. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. PhD thesis, Universidad Carlos III de Madrid, 2015.

[Toyer *et al.*, 2018] Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, 2018.

[Toyer *et al.*, 2020] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68, 2020.

[UPS, 2025] UPS. Global reporting initiative. Accessed from https://about.ups.com/content/dam/upsstories/images/our-impact/reporting/2024-UPS-GRI-Report.pdf, 2025.

[Waldinger, 1977] Richard J. Waldinger. Achieving several goals simultaneously. *Machine Intelligence*, 8:94–136, 1977.

[Yang *et al.*, 2022] Ryan Yang, Tom Silver, Aidan Curtis, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PG3: policy-guided planning for generalized policy generation. In *IJCAI*, 2022.

[Yoon *et al.*, 2008] Sung Wook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *J. Mach. Learn. Res.*, 9:683–718, 2008.

Appendix

Table of Contents

A Numeric Planning Definitions

We can extend the definition of a planning problem in Definition 2 to handle numeric variables, numeric conditions and numeric effects by making use of the fragment of PDDL 2.1 [Fox and Long, 2003] excluding durative actions. In this paper, we consider the fragment of numeric planning where numeric conditions are restricted to involve one numeric variable per formula with comparisons in $\{\geq, >, =\}$, and numeric action effects to additions by a constant value. This fragment is very expressive as it is undecidable by reduction from an abacus program [Helmert, 2002, Theorem 12].

Definition 23 (Abacus Program Numeric Planning Problem)**.** A (lifted) numeric planning problem is a tuple $\Pi = \langle \mathcal{P}, \mathcal{F}, \mathcal{O}, \mathcal{A}, s_0, g \rangle$ where $\mathcal{P}$ is a set of lifted predicates, $\mathcal{F}$ is a set of lifted (numeric) functions, $\mathcal{O}$ is a set of objects, $\mathcal{A}$ is a set of action schemata, s_0 is the initial state, and g the goal condition. Predicates are defined as in Definition 2 and functions are similar where each function $f \in \mathcal{F}$ has a set of argument terms $x_1, \ldots, x_{n_f}$ where $n_f \in \mathbb{N}_0$ depends on f. A propositional variable is a predicate whose argument terms are all instantiated with objects, and has a range in $\{\top, \bot\}$. A numeric variable is a function whose argument terms are all instantiated with objects, and has a range in $\mathbb{R}$. We let N_p/N_n denote the set of propositional/numeric variables for Π given by all possible instantiations of predicates/functions. A state is a value assignment to all N_p and N_n.

A literal is a predicate p or its negation $\neg p$. A propositional condition is a positive (resp. negative) literal $x = \top$ (resp. $\bot$), and a numeric condition has the form $f \trianglerighteq c$ where f is a function, $c \in \mathbb{R}$ and $\trianglerighteq \in \{\geq, >, =\}$. 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.

An action schema $a \in \mathcal{A}$ is a tuple $\langle \mathit{var}(a), \mathit{pre}(a), \mathit{add}(a), \mathit{del}(a), \mathit{num\_eff}(a) \rangle$ where $\mathit{var}(a)$ is a set of parameter terms, the preconditions $\mathit{pre}(a)$ is a set of conditions, $\mathit{add}(a)$ and $\mathit{del}(a)$ are add and delete lists of predicates as in the classical case, $\mathit{num\_eff}(a)$ is a set of numeric conditions of the form $f(x_1, \ldots, x_{n_f}) = f(x_1, \ldots, x_{n_f}) + c$ for $c \in \mathbb{R}$ with argument terms instantiated with terms or objects from $\mathit{var}(a) \cup \mathcal{O}$. An action a is applicable in a state s if s satisfies $\mathit{pre}(a)$. In this case, its successor $\mathrm{succ}(s, a)$ is the state where the effects $\mathit{num\_eff}(a)$ are applied to the numeric variables in s, and propositional variables are modified in the same way as in the classical case. If a is not applicable in s, we have $\mathrm{succ}(s, a) = \bot$. The definition of plan is the same as in the classical case.

Next, we extend the definition of logical regression for classical planning in Definition 3 to handle numeric conditions and effects for the class of abacus program numeric problems.

Definition 24 (Abacus Program Numeric Planning Regression)**.** A set of conditions g is regressable over an action a if the classical conditions are regressable over a as in Definition 3, with no restriction for numeric conditions. In this case, we define the regression of g under a by transforming numeric conditions $f \trianglerighteq c$ to $f \trianglerighteq c - v$ if there is an effect of a of the form $f = f + v$, and if there exists any precondition $\xi = (f \trianglerighteq c)$ in a, the associated numeric condition corresponding to f in g is transformed to ξ.

B Proofs for Section 4

Proof Sketch for Proposition 11. Membership follows from PSPACE-completeness of planning for fixed domains [Bylander, 1994; Erol *et al.*, 1995], and hardness by reduction from the Rush Hour problem which has singleton goals and has been shown to be PSPACE-hard [Hearn and Demaine, 2005]. □

Proof Sketch for Corollary 12. Membership again follows from PSPACE-completeness of planning for fixed domains. Hardness follows from Proposition 11 as TGI is a special case of SGI. □

Proof Sketch for Proposition 13. This follows by noting that the greedy algorithm described in the definition of TGI runs in polynomial time under the assumption that step (2a) can run in polynomial time. □

Proof Sketch for Corollary 14. TGI_C of a GP problem implies pTGI as optimal search in step (2a) of the greedy TGI algorithm runs in polynomial time with exponent C but C is a constant. □

Proof Sketch for Proposition 15. For NP membership, one guesses the correct ordering of goals after which running the greedy algorithm is in polynomial time. For NP-hardness, we reduce from the Hamiltonian path problem which is NP-complete [Garey and Johnson, 1979, p. 60]. The Hamiltonian path problem asks to find a path on a graph that visits every vertex exactly once. To encode this as a planning domain, one would require goals of the form *visited*(x) and actions which traverse a graph but make each vertex untraversable once it has been visited, such as by deleting initially true *clear*(x) atoms. Then the Hamiltonian path problem is equivalent to finding an correct ordering of goals representing (adjacent) vertices to visit. □

Proof Sketch for Proposition 17. Reflexivity, symmetry and transitivity follows from the usage of bijective functions in the definition of $\sim_U$. □

Proof Sketch for Proposition 18. The statement follows by definitions and bijectiveness of f. □

Proof Sketch for Theorem 19. Let $\mathbf{GP} = \langle \mathbb{D}, \mathbb{P} \rangle$ be a GP problem exhibiting TGI_C as in the statement assumption, with bound $C \in \mathbb{N}$ from Definition 9. To show completeness, it suffices to prove that there exists a finite number of MOOSE rules π, representing all possible optimal plans for singleton goals in step (2a) of the definition of TGI in Definition 9, which when executed with Algorithm 3 can solve any problem in $\mathbf{GP}$ with singleton goals. Then by the TGI assumption, any arbitrary problem $\mathbf{P}$ in $\mathbb{P}$ can be solved because execution of π would achieve the goals $\mathbf{P}[g]$ in a monotonic fashion. Soundness follows by the fact that application of rules lifted from regression are sound and thus, the execution of Algorithm 3 is sound.

Now, note that the set of possible singleton goals in $\mathbf{GP}$ is finite modulo the equivalence relation $\sim_U$ restricted to sets of atoms, as there are a finite number of predicates and instantiations of nonequivalent objects. Furthermore by the TGI_C assumption any optimal plan for any possible singleton goal g has plan length less than C. Thus the set of all possibles sequences of actions that can be regressed from g is bounded by $\sum_{k=0}^{C}(|\mathcal{A}| \cdot (kN' + M')^N)^k$ where

- $N = \max_{a \in \mathcal{A}}(|var(a)|)$ is the maximum arity of schemata,
- M is the maximum arity of predicates,
- $N' = N + |\mathcal{C}|$, and
- $M' = M + |\mathcal{C}|$.

Note that by Proposition 18, it suffices to count the equivalence class of plans under $\sim_U$. This is because modulo $\sim_U$ there are at most $(kN' + M')^N$ possible instantiations of an action where there are at most $kN' + M'$ possible objects across all actions in a length-k plan and the singleton goal under $\sim_U$. By a similar argument, there are $|\mathcal{P}| \cdot M'^M$ possible singleton goal instantiations, and hence it takes a finite number of up to $n = \sum_{k=0}^{C}(|\mathcal{A}| \cdot (kN' + M')^N)^k \cdot (|\mathcal{P}| \cdot M'^M)$ different problems to synthesise a general policy for $\mathbf{GP}$. □

Proof Sketch for Proposition 21. NP membership follows by definition of OGI, more specifically the nondeterministic algorithm defined within. NP-hardness follows by the NP-hardness of SGI which can be viewed as a general case of OGI. □

Proof Sketch for Theorem 22. Completeness follows from the completeness of rules as discussed in Theorem 19. To show soundness for optimal planning, we note that given enough training problems, bounded by $n = \sum_{k=0}^{C}(|\mathcal{A}| \cdot (kN' + M')^N)^k \cdot (|\mathcal{P}| \cdot M'^M)$ from the previous theorem, learned rules do not throw away any optimal actions at every state. Then the proof follows from the definition of OGI. □

Example 1 (Theorem 22 counterexample without the OGI assumption)**.** *A counterexample to the previous theorem for when the OGI assumption is dropped occurs if we can find a case where achieving a singleton goal for a TGI problem suboptimally is necessary to achieve optimality for the whole problem. In the planning problem illustrated by the state space in Figure 4 with goal $\{g_1, g_2\}$, an optimal plan to either g_1 or g_2 has plan length 2 and the greedy algorithm in Definition 9 of TGI returns a plan of length 4. However, the optimal plan has length 3.*

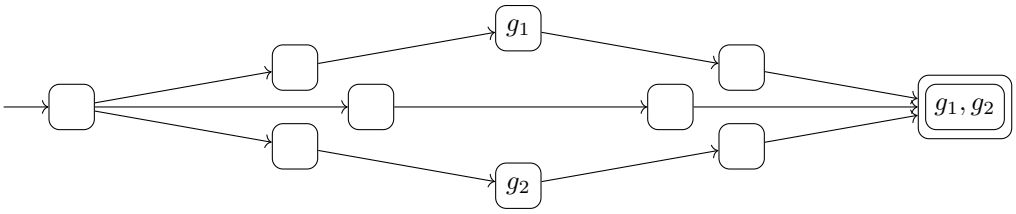

Figure 4: A planning problem illustrating the necessity of the OGI assumption for learning provably optimal policies in Theorem 22.

Algorithm 4: Validation and Refinement Routine

Input: MOOSE program π, and training problems $\mathbb{P}_{\text{train}} = \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(n_t)}$.
Output: MOOSE program π and *execution* mode for π.

1 $execution \leftarrow$ greedy
2 **do**
3 **for** $i = 1, \ldots, n_t$ **do**
4 $\vec{\alpha}, status \leftarrow \text{exec}(\mathbf{P}^{(i)}, \pi)$ `// Alg. 3`
5 **if** $status =$ success **then continue**
6 **if** *reason for failure* = π *caused a deadend* **then**
7 $r \leftarrow$ rule that caused the deadend
8 $\pi.r.precedence \leftarrow \pi.r.precedence + (0, 1)$
9 $execution \leftarrow$ conservative
10 **else if** *reason for failure* = π *ran into a cycle* **then**
11 $r \leftarrow$ last executed rule
12 $\pi.r.precedence \leftarrow \pi.r.precedence + (-1, 1)$
13 $execution \leftarrow$ conservative
14 **else if** *reason for failure* = π *derived no rule* **then**
15 $\pi \leftarrow \pi \cup \text{learn}(\{\mathbf{P}^{(i)}_{\text{succ}(\mathbf{P}^{(i)}[s_0], \vec{\alpha})}\})$
16 **while** π *has changed*
17 **return** π, *execution*

C Validation and Refinement of MOOSE Rules

It may be the case that MOOSE generates rules successfully in Algorithm 1 but does not solve problems from a domain if it does not exhibit the TGI assumption, i.e. if there exists an ordering of goals that must not be ignored. In order to handle such cases, we introduce a validation and refinement procedure for the synthesis of MOOSE programs, as well as different execution modes for MOOSE programs. First, we modify the type of the precedence ranking function from $\mathcal{R} \to \mathbb{N}$ to $\mathcal{R} \to \mathbb{N} \times \mathbb{N}$. Line 5 in Algorithm 2 is then changed to $\pi \leftarrow \pi \cup \{(r, (|\vec{\alpha}| - i + 1, 1))\}$.

Now, the execution of a MOOSE program π may fail due to a number of reasons. The validation and refinement procedure displayed in Algorithm 4 aims to resolve some failures by iterating over all training problems and trying to execute the generalised plan (Line 4) and checking for success (Line 5). If the generalised plan fails, it is caused by one of the following reasons: the generalised plan caused a deadend, in which case the precedence of the rule which caused the deadend is lowered (Line 8); the generalised plan ran into a cycle, in which case the rule which caused the cycle is strictly lowered (Line 12); or the generalised plan has not encountered a solvable state before, in which case learning is done on the new state (Line 15). In the former two cases, the generalised plan is switched to a 'conservative mode' as it has learned that some actions are dangerous if executed greedily. Otherwise, the generalised plan is kept in a 'greedy mode' which means that it uses Algorithm 3 for instantiation. Although the generalised plan is not always guaranteed to stabilise, it does so for all domains used in the experiments.

Algorithm 5 shows how one instantiates a MOOSE program when the synthesis procedure has determined that the plan be executed in a conservative mode. The algorithm is equivalent to Algorithm 3 with two additions highlighted in blue. Rules are now queried in order of descending precedence values (Line 5) and furthermore, only the first action of each rule head is executed (Line 9).

Algorithm 5: MOOSE Conservative Plan Instantiation

Input: A planning problem $\mathbf{P}$ and MOOSE program π.
Output: A plan $\vec{\alpha}$ and success or failure status.

$s \leftarrow \mathbf{P}[s_0]$
$\vec{\alpha} \leftarrow []$ `// empty sequence`
while $\mathbf{P}[g] \not\subseteq s$ **do**
  $\vec{\beta} \leftarrow \perp$
  for $r \in \pi$ *in descending precedence values* **do**
    $\vec{\beta} \leftarrow \text{grounding}(r, s, \mathbf{P}[g])$ `// Eqn.` (3)
    if $\vec{\beta} \neq \perp$ **then break**
  if $\vec{\beta} = \perp$ *or detected cycle* **then return** $\vec{\alpha}$, failure
  $\vec{\beta} \leftarrow [\vec{\beta}_0]$
  $\vec{\alpha} \leftarrow \vec{\alpha}; \vec{\beta}$ `// sequence concatenation`
  $s \leftarrow \text{succ}(s, \vec{\beta})$
return $\vec{\alpha}$, success

D Training and Testing Task Size Distributions

D.1 Classical Planning

Domain	Object Types	Train Min	Train Max	Test Min	Test Max
Barman	Σ	16	27	21	853
	cocktail	3	7	4	393
	dispenser	3	3	3	30
	hand	2	2	2	2
	ingredient	3	3	3	30
	level	3	3	3	3
	shaker	1	1	1	1
	shot	1	8	5	394
Childsnack	Σ	6	32	20	1326
	bread-portion	1	6	4	292
	child	1	6	4	292
	content-portion	1	6	4	292
	place	1	3	3	3
	sandwich	1	9	4	437
	tray	1	2	1	10
Ferry	Σ	3	8	7	1461
	car	1	2	2	974
	location	2	6	5	487
Logistics	Σ	29	29	10	1260
	airplane	3	3	1	64
	city	3	3	1	64
	location	15	15	2	960
	package	5	5	5	108
	truck	3	3	1	64
Miconic	Σ	3	11	5	681
	floor	2	7	4	196
	passenger	1	4	1	485
Rover	Σ	10	36	12	596
	camera	1	4	1	99
	lander	1	1	1	1
	mode	3	3	3	3
	objective	1	10	1	236
	rover	1	4	1	30
	store	1	4	1	30
	waypoint	2	10	4	197
Satellite	Σ	5	43	11	402
	direction	2	10	4	98
	instrument	1	20	3	195
	mode	1	3	1	10
	satellite	1	10	3	99
Transport	Σ	6	17	12	354
	location	2	7	5	99
	package	1	4	1	194
	size	2	3	3	11
	vehicle	1	3	3	50

Table 1: Object distribution of training and testing splits for *classical* planning.

D.2 Numeric Planning

Domain	Objects	Train Min	Train Max	Test Min	Test Max
Num-adultsnack	Σ	8	39	43	2644
	at_kitchen_bread	1	10	12	876
	at_kitchen_content	1	10	12	876
	food	3	3	3	3
	hungry	1	10	12	876
	place	1	3	3	3
	tray	1	3	1	10
Num-childsnack	Σ	5	10	16	889
	at_kitchen_bread	1	2	4	292
	at_kitchen_content	1	2	4	292
	hungry	1	2	4	292
	place	1	2	3	3
	tray	1	2	1	10
Num-ferry	Σ	7	9	11	1465
	car	1	2	2	974
	ferry-capacity	4	4	4	4
	location	2	3	5	487
Num-miconic	Σ	7	15	9	685
	floor	2	7	4	196
	lift-capacity	4	4	4	4
	passenger	1	4	1	485
Num-minecraft	Σ	5	5	15	2100
	cell	4	4	11	1800
	pogo_sticks_to_make	1	1	4	300
Num-transport	Σ	5	14	13	605
	capacity	1	4	4	262
	location	2	4	5	99
	package	1	4	1	194
	vehicle	1	2	3	50

Table 2: Object distribution of training and testing splits for *numeric* planning.

E Additional Experimental Results

E.1 Coverage Tables

Domain	M(3h‖3n)	MRP+HJ	MOOSE
Num-Adultsnack	32	23	**90**
Num-Childsnack	64	49	**90**
Num-Ferry	60	61	**90**
Num-Miconic	63	71	**90**
Num-Minecraft	30	68	**90**
Num-Transport	45	64	**90**
∑ (540)	294	336	**540**

Domain	LAMA	SLEARN-0	SLEARN-1	SLEARN-2	MOOSE
Barman	49	0	0	0	**90**
Childsnack	35	0	66	0	**90**
Ferry	69	67	67	60	**90**
Logistics	77	0	0	0	**89**
Miconic	**90**	**90**	**90**	**90**	**90**
Rovers	66	0	0	0	**90**
Satellite	89	0	47	48	**90**
Transport	66	0	63	46	**90**
∑ (720)	541	157	333	244	**719**

Domain	Blind	LMCUT	SCORPION	SYMK	MOOSE
Barman	0	0	0	12	**26**
Childsnack	9	9	9	15	**17**
Ferry	10	18	17	18	**30**
Logistics	8	15	**22**	10	16
Miconic	**30**	**30**	**30**	**30**	**30**
Rovers	15	17	18	**20**	**20**
Satellite	12	22	**26**	21	21
Transport	9	9	**20**	13	15
∑ (240)	93	120	142	139	**175**

Table 3: Planning coverage for satisficing numeric planning (left), satisficing classical planning (middle), and optimal classical planning (right). Domains have 90/30 problems each for SAT/OPT planning.

E.2 Satisficing Numeric Planning

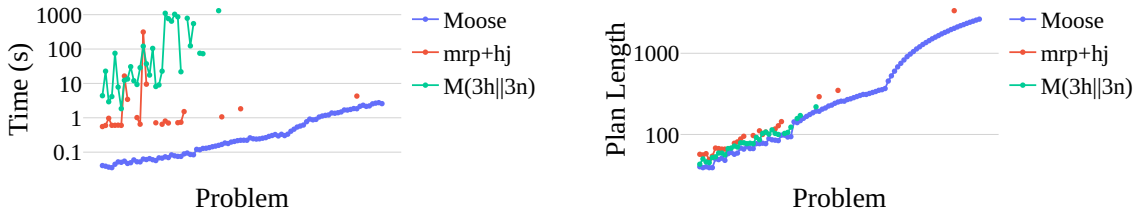

Figure 5: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Numeric Adultsnack**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

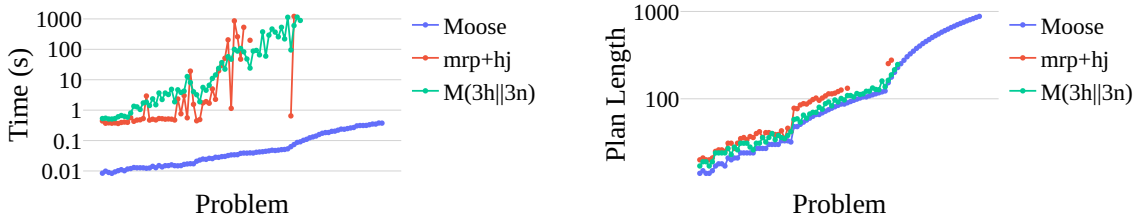

Figure 6: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Numeric Childsnack**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

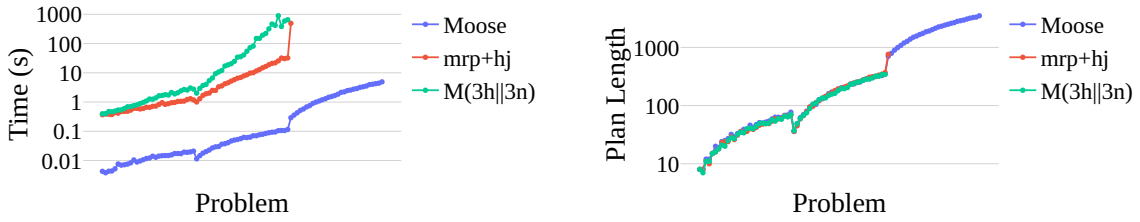

Figure 7: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Numeric Ferry**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

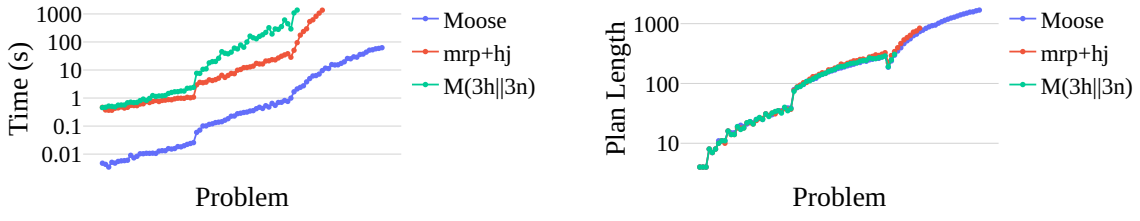

Figure 8: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Numeric Miconic**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

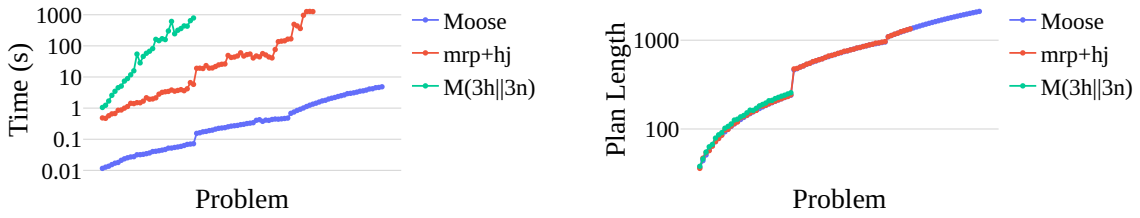

Figure 9: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Numeric Minecraft**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

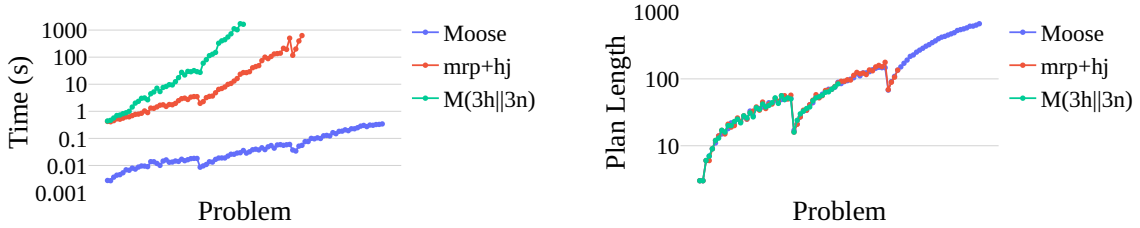

Figure 10: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Numeric Transport**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

E.3 Satisficing Classical Planning

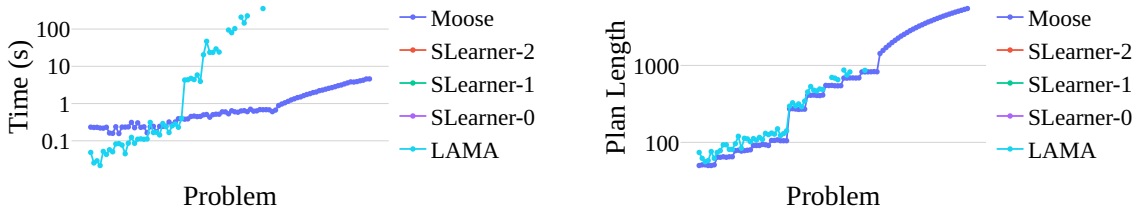

Figure 11: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Barman**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

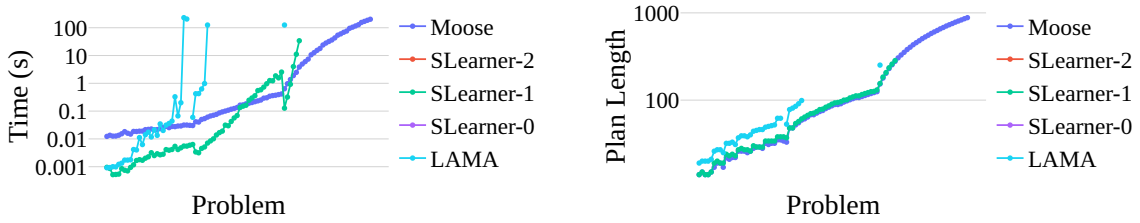

Figure 12: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Childsnack**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

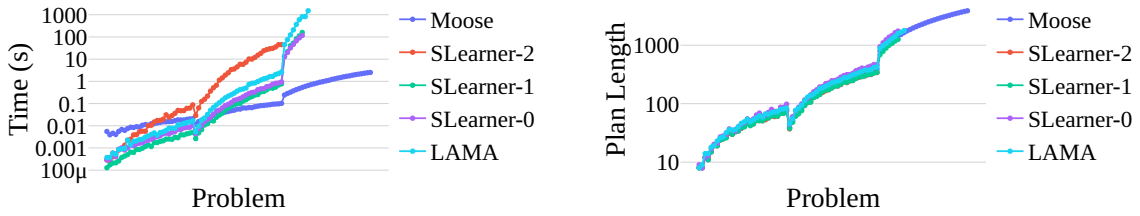

Figure 13: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Ferry**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

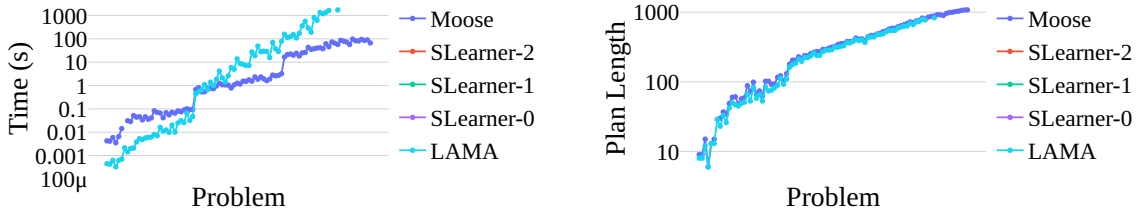

Figure 14: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Logistics**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

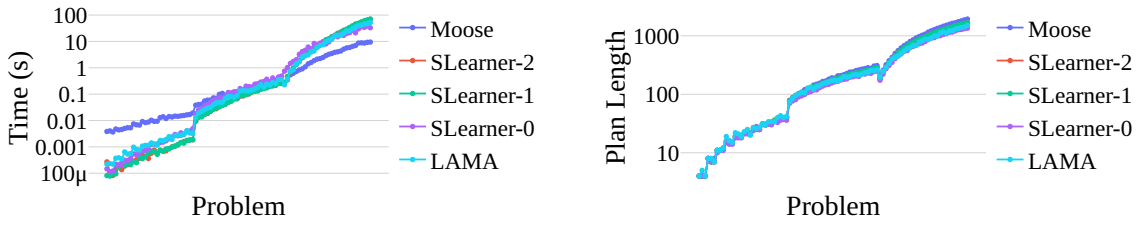

Figure 15: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Miconic**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

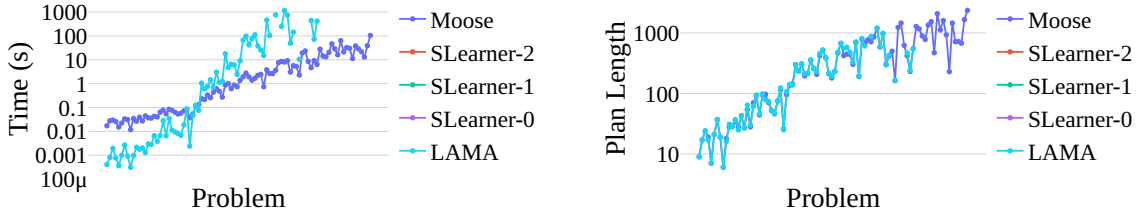

Figure 16: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Rover**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

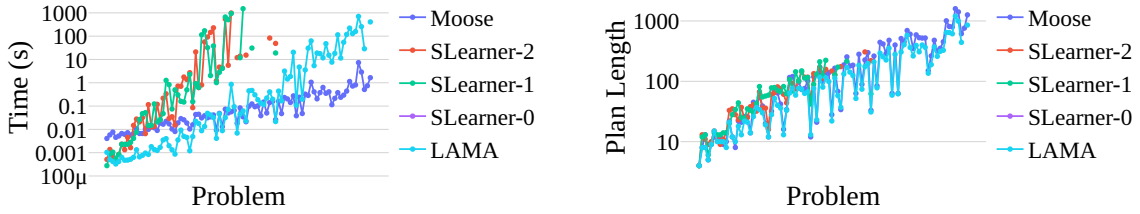

Figure 17: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Satellite**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

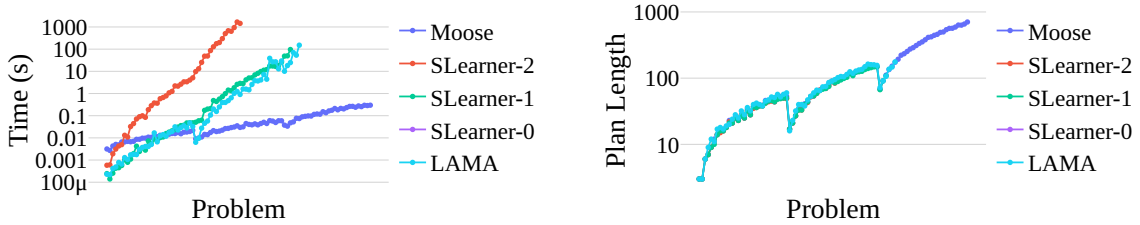

Figure 18: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Transport**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime and plan length.

E.4 Optimal Classical Planning

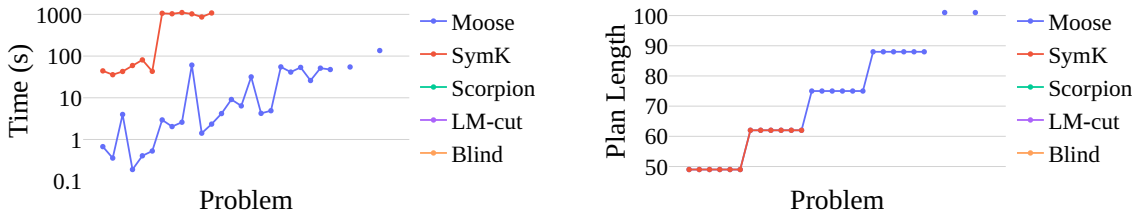

Figure 19: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Barman**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

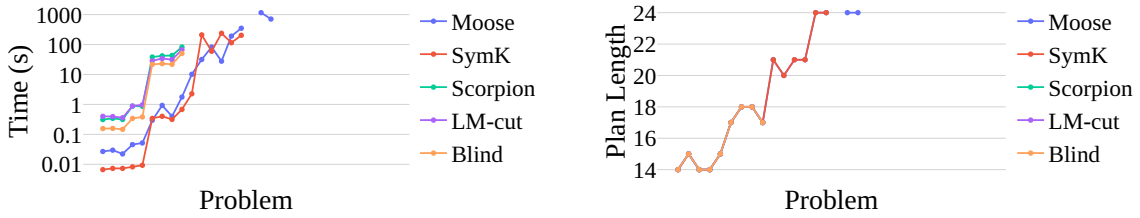

Figure 20: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Childsnack**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

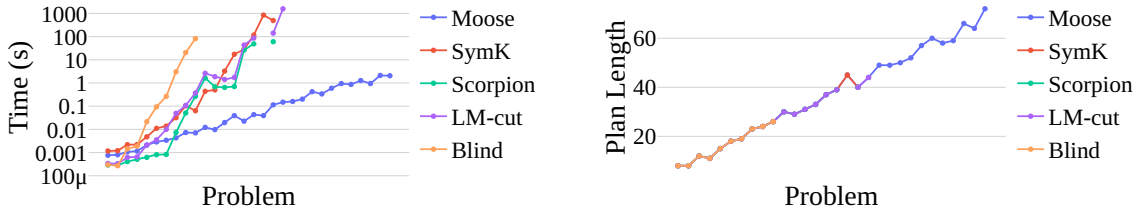

Figure 21: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Ferry**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

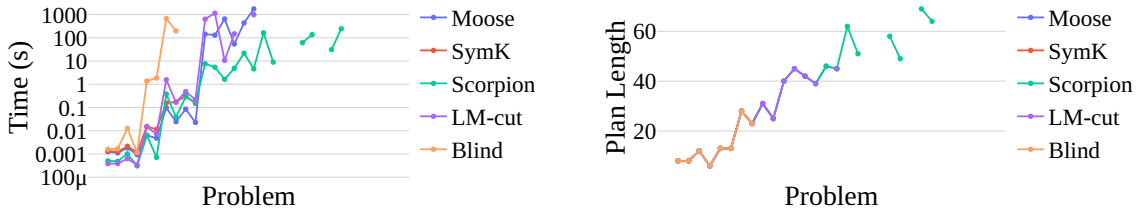

Figure 22: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Logistics**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

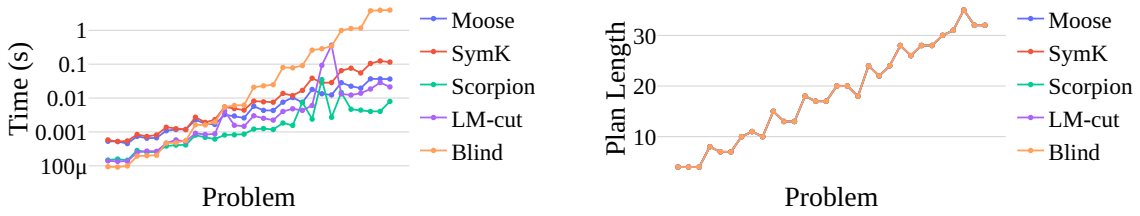

Figure 23: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Miconic**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

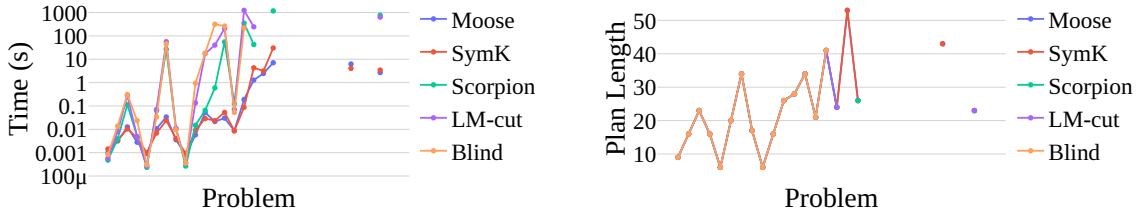

Figure 24: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Rover**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

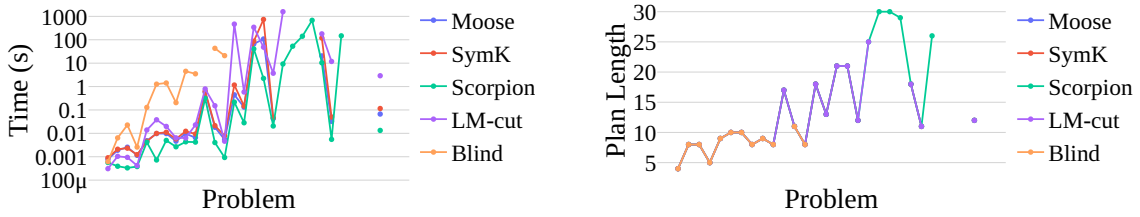

Figure 25: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Satellite**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

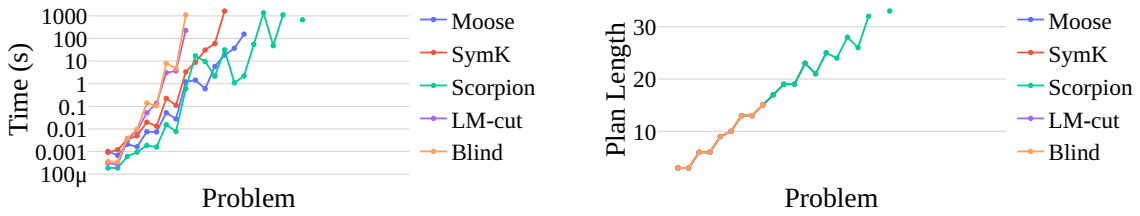

Figure 26: Line plots for time in seconds (left) and plan length (right) of planners across solved problems for **Transport**. Planning problem difficulty increases across the x-axis, and lower y-axis values are better (↓). Note the log scale of the y-axis for runtime.

E.5 MOOSE vs. LAMA on Plan Quality

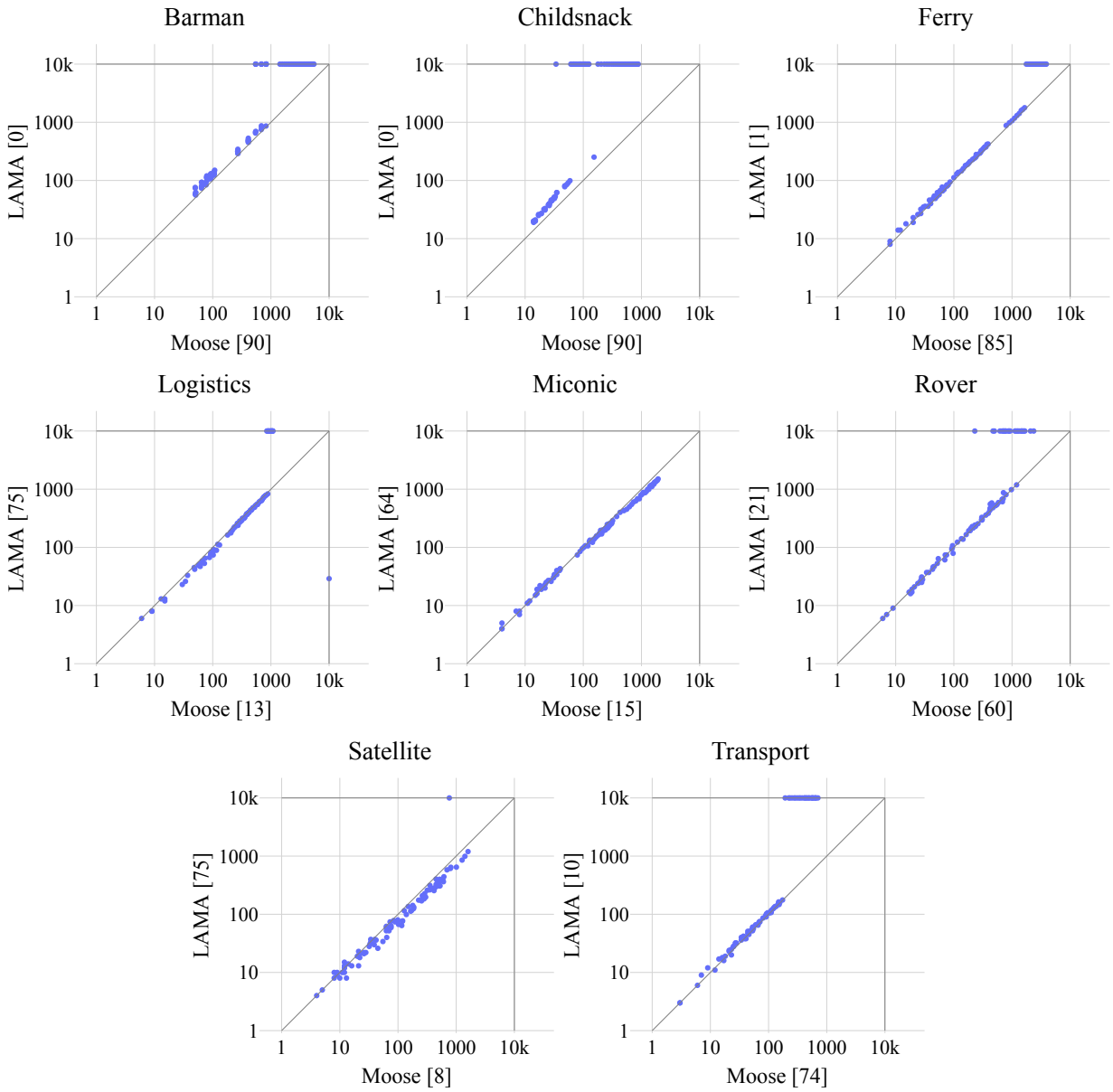

Figure 27: Plot comparisons of expanded nodes of LAMA (y-axis) and MOOSE (x-axis) for different classical planning domains. A point (x, y) represents the metric of the models indicated on the x and y axis on the domain. The number in the brackets next to each model indicates how many planning problems the model returned a higher quality plan than the model on the other axis. Points on the top left (resp. bottom right) triangle favour MOOSE (resp. LAMA).