Satisficing and Optimal Generalised Planning via Goal Regression (Extended Version)

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 (GP) refers to the task of synthesising programs that solve families of related planning problems. We introduce a novel, yet simple method for GP: given a set of training problems, for each problem, compute an optimal plan for each goal atom in some order, perform goal regression on the resulting plans, and lift the corresponding outputs to obtain a set of first-order *Condition* $\rightarrow$ *Actions* rules. The rules collectively constitute a generalised plan that can be executed as is or alternatively be used to prune the planning search space. We formalise and prove the conditions under which our method is guaranteed to learn valid generalised plans and state space pruning axioms for search. Experiments demonstrate significant improvements over state-of-the-art (generalised) planners with respect to the 3 metrics of synthesis cost, planning coverage, and solution quality on various classical and numeric planning domains.

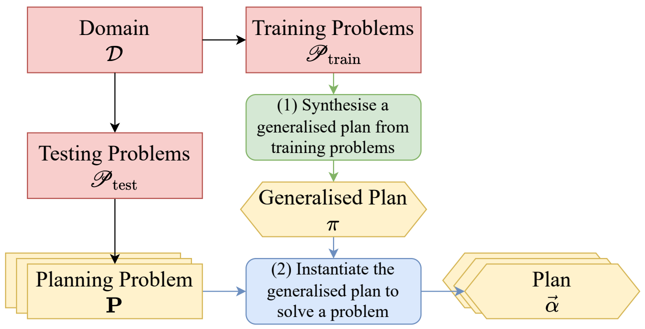

Domain
$\mathcal{D}$
Training Problems
$\mathscr{P}_{\text{train}}$
(1) Synthesise a generalised plan from training problems
Testing Problems
$\mathscr{P}_{\text{test}}$
Generalised Plan
π
Planning Problem
P
(2) Instantiate the generalised plan to solve a problem
Plan
$\vec{\alpha}$

Figure 1: A common GP setup consisting of a planning domain $\mathcal{D}$, and a set of training problems $\mathscr{P}_{\text{train}}$ and testing problems $\mathscr{P}_{\text{test}}$. A generalised planner consists of two modules: (1) synthesis and (2) instantiation. See text for details.

1 Introduction

Generalised planning (GP) aims to compute generalised plans: programs that solve families of related planning problems. A grand goal of GP is to amortise synthesis costs by solving families of planning problems faster than general-purpose planners that solve each problem individually. Indeed, there exists several planning domains that are computationally easy to solve and exhibit satisficing policies, such as variants of the package delivery domain (Helmert 2003). In the real world, UPS$^{\text{TM}}$ delivered over 20 million packages daily across over 200 countries and territories in 2024 (UPS 2025). However, state-of-the-art, general-purpose planners struggle to scale up to a simplified version of the delivery problem with 100 packages (Taitler et al. 2024).

We consider the GP problem that consists of a planning domain $\mathcal{D}$, and a set of training problems $\mathscr{P}_{\text{train}}$ and testing problems $\mathscr{P}_{\text{test}}$ drawn from $\mathcal{D}$, as depicted in Figure 1. A generalised planner synthesises a generalised plan from $\mathcal{D}$ and $\mathscr{P}_{\text{train}}$ that solves problems in $\mathscr{P}_{\text{test}}$. Metrics for evaluating the effectiveness of a generalised planner (Srivastava, Immerman, and Zilberstein 2011) include the resources it takes to synthesise a generalised plan (*synthesis cost*), the time it takes to instantiate generalised plans on unseen problems (*instantiation cost*), and the quality of instantiated plans (*solution quality*).

In this paper, we introduce a new generalised planner that draws upon insights and long-standing ideas of *goal regression* from the knowledge representation community and *problem relaxation* from the planning community to advance the state of the art in GP over the three aforementioned metrics. Our approach consists of an efficient three step process of (1) solving for each goal of each problem in $\mathscr{P}_{\text{train}}$ optimally in an arbitrary order, (2) performing goal regression (Fikes, Hart, and Nilsson 1972; Waldinger 1977; Lozano-Perez, Mason, and Taylor 1984; Reiter 1991, 2001) over the resulting plans, and (3) lifting the corresponding partial-state, macro-action pairs into sets of first-order rules constituting a generalised plan. We treat planning states as databases (Corrêa et al. 2020) and lifted rules as queries, and correspondingly employ database algorithms for instantiating generalised plans quickly on problems in $\mathscr{P}_{\text{test}}$. We also leverage derived predicates and axioms to encode learned rules for state space pruning in search.

We formalise the conditions under which our approach learns sound and complete generalised plans, and under which the encoding of learned rules as axioms gives rise to provably optimal plans when combined with optimal search planners. We manifest our contributions in the MOOSE planner[1] and conduct experiments with MOOSE on classical and numeric planning domains and satisficing and optimal planning settings. We observe that MOOSE outperforms state-of-the-art baseline planners by large margins on Easy-to-

[1]Available at https://github.com/dillonzchen/moose

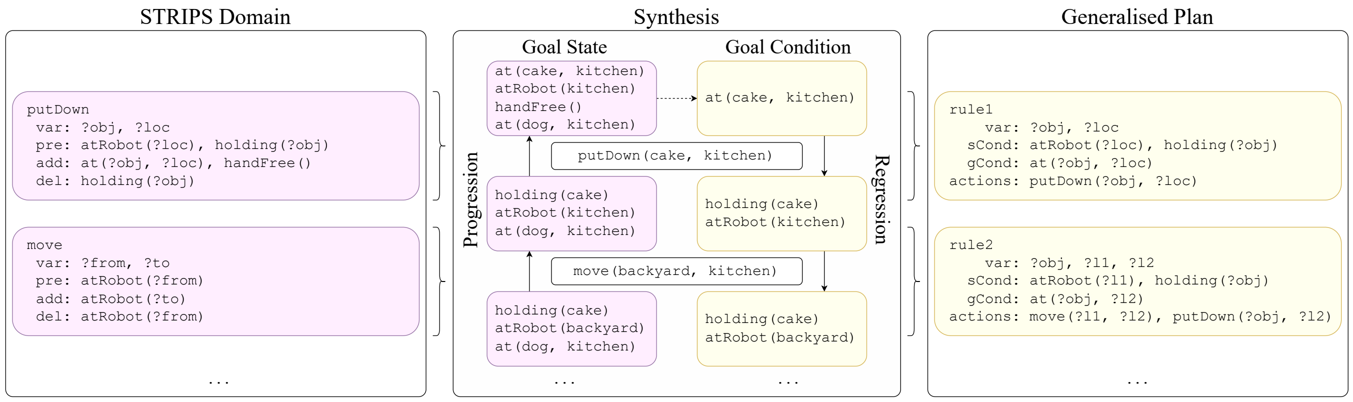


Figure 2: **Left:** a simplified STRIPS transportation domain. **Middle:** state progression (purple) and goal regression (yellow) via a `putDown` action. **Right:** the generalised plan created by lifting the regressed states, goal condition, and plan actions.

Solve, Hard-to-Optimise (ESHO) planning domains — P-time solvable and NP-hard to solve optimally domains.

We summarily list our contributions as follows.

- We introduce algorithms for synthesising and instantiating generalised plans that employ goal regression, problem relaxation, and first-order query techniques.
- We formalise the conditions under which our approach is sound and complete for satisficing and optimal planning.
- We conduct experiments and demonstrate the effectiveness of our algorithms in satisficing and optimal planning over various classical and numeric planning domains.

2 Planning Background and Notation

We adopt standard notation for representing planning problems via the STRIPS fragment of the Planning Domain Definition Language (PDDL) (McDermott et al. 1998; Haslum et al. 2019). Our approach also handles a fragment of numeric planning but we defer such definitions to Appendix A.

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.

STRIPS Planning 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.

A *planning domain* is a tuple $\mathcal{D} = \langle \mathcal{P}, \mathcal{C}, \mathcal{A} \rangle$, 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 $\mathit{var}(a) \cup \mathcal{C}$.

Example 1 (Transportation Domain)**.** *We can model a simplified transportation domain where an agent can transport items between locations in STRIPS, partially depicted in Figure 2, as follows. We define a domain* $\mathcal{D} = \langle \mathcal{P}, \mathcal{C}, \mathcal{A} \rangle$ *with* $\mathcal{P} = \{\mathit{at}(?x, ?y), \mathit{atRobot}(?x), \mathit{handFree}(), \mathit{holding}(?x)\}$, $\mathcal{C} = \emptyset$, *and* $\mathcal{A}$ *containing the three action schemata:*

$\mathit{putDown}$

$$\begin{aligned} \mathit{var} &= \{?\mathit{obj}, ?\mathit{loc}\} \\ \mathit{pre} &= \{\mathit{atRobot}(?\mathit{loc}), \mathit{holding}(?\mathit{obj})\} \\ \mathit{add} &= \{\mathit{at}(?\mathit{obj}, ?\mathit{loc}), \mathit{handFree}()\} \\ \mathit{del} &= \{\mathit{holding}(?\mathit{obj})\} \end{aligned}$$

move

$$\begin{aligned} \mathit{var} &= \{?\mathit{from}, ?\mathit{to}\} \\ \mathit{pre} &= \{\mathit{atRobot}(?\mathit{from})\} \\ \mathit{add} &= \{\mathit{atRobot}(?\mathit{to})\} \\ \mathit{del} &= \{\mathit{atRobot}(?\mathit{from})\} \end{aligned}$$

pickUp

$$\begin{aligned} \mathit{var} &= \{?\mathit{obj}, ?\mathit{loc}\} \\ \mathit{pre} &= \{\mathit{atRobot}(?\mathit{loc}), \mathit{at}(?\mathit{obj}, ?\mathit{loc}), \mathit{handFree}()\} \\ \mathit{add} &= \{\mathit{holding}(?\mathit{obj})\} \\ \mathit{del} &= \{\mathit{at}(?\mathit{obj}, ?\mathit{loc}), \mathit{handFree}()\} \end{aligned}$$

A *planning problem* is a tuple $\mathbf{P} = \langle \mathcal{D}, s_0, g, O \rangle$ with $\mathcal{D}$ a planning domain, s_0 the initial state, g the goal condition, and $O \supseteq \mathcal{C}$ 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 $\mathcal{D}$. 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 $o_1, \ldots, o_n \in O$. An action a is applicable in a state s if $\mathit{pre}(a) \subseteq s$, in which case we define the *successor*

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

Otherwise, a is not applicable in s and $\text{succ}(s, a) = \bot$.

A *plan* for a planning problem is a finite sequence of actions $\vec{\alpha} = a_1, \ldots, a_n$ where $s_i = \text{succ}(s_{i-1}, a_i) \neq \bot$ for $i = 1, \ldots, n$ and s_n is a goal state. We overload the notation of successor for sequences of actions with $\text{succ}(s, \vec{\alpha}) = s_n$ as if $s = s_0$. The length of a plan $\vec{\alpha}$ is its number of actions. 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 $\mathbf{P}$.

Example 2 (Transportation Problem)**.** *We model a transportation problem where a robot and a dog are in the kitchen, and a cake that is in the backyard has to be brought into the kitchen as a STRIPS problem* $\mathbf{P} = \langle \mathcal{D}, s_0, g, O \rangle$ *where* $\mathcal{D}$ *is the transportation domain in Example 1,* $s_0 = \{\mathit{at}(\mathit{cake}, \mathit{backyard}), \mathit{at}(\mathit{dog}, \mathit{kitchen}), \mathit{atRobot}(\mathit{kitchen}), \mathit{handFree}()\}$, $g = \{\mathit{at}(\mathit{cake}, \mathit{kitchen})\}$, *and* $O = \{\mathit{backyard}, \mathit{cake}, \mathit{dog}, \mathit{kitchen}\}$. *A plan is given by* $\vec{\alpha} = \mathit{move}(\mathit{kitchen}, \mathit{backyard}), \mathit{pickUp}(\mathit{cake}, \mathit{backyard}), \mathit{move}(\mathit{backyard}, \mathit{kitchen}), \mathit{putDown}(\mathit{cake}, \mathit{kitchen})$. *The problem and plan are partially depicted in Figure 2.*

We introduce some additional notational shorthands to be used 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 $\mathbf{P}$. Next, given a state s and set of atoms g' for a problem $\mathbf{P}$, let $\mathbf{P}_s = \langle \mathcal{D}, s, \mathbf{P}[g], \mathbf{P}[O] \rangle$ denote the same problem with the initial state replaced with s, and $\mathbf{P}_{s,g'} = \langle \mathcal{D}, s, g', \mathbf{P}[O] \rangle$ denote the same problem with the initial state and goal replaced with s and g'.

Goal Regression Goal regression refers to the computation of the preimage of a goal under a sequence of actions via regression rewriting. Goal regression computes the minimal set of goal relevant atoms, and has been used for heuristic synthesis (Bonet and Geffner 2001; Scala, Haslum, and Thiébaux 2016), plan monitoring (Fritz and McIlraith 2007), policy synthesis for lifted Markov decision processes (Gretton and Thiébaux 2004; Sanner and Boutilier 2009), nondeterministic planning (Muise, McIlraith, and Beck 2012, 2024), symbolic search (Pang and Holte 2011; Alcázar et al. 2013; Speck, Seipp, and Torralba 2025), numeric planning (Illanes and McIlraith 2017), generating macro-actions (Hofmann, Niemueller, and Lakemeyer 2020), GP (Illanes and McIlraith 2019; Yang et al. 2022), and embodied AI (Kaelbling and Lozano-Pérez 2011; Xu et al. 2019; Liu et al. 2025).

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*

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

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

Problem Statement: Generalised Planning

We introduce the generalised planning (GP) problem as a set of planning problems sharing the same domain. We describe the variant involving a set of training problems as seen commonly in recent GP works (e.g. (Francès, Bonet, and Geffner 2021; Drexler, Seipp, and Geffner 2022; Yang et al. 2022)).

A *generalised planning problem* is a tuple $\mathbf{GP} = \langle \mathscr{P}_{\text{train}}, \mathscr{P}_{\text{test}} \rangle$ where $\mathscr{P}_{\text{train}}$ (resp. $\mathscr{P}_{\text{test}}$) is a finite (resp. possibly infinite) set of problems belonging to the same domain $\mathcal{D}$. A *generalised plan* π for a GP problem is a programmatic plan that is synthesised from the domain $\mathcal{D}$ and $\mathscr{P}_{\text{train}}$, and can be instantiated on any planning problem $\mathbf{P} \in \mathscr{P}_{\text{test}}$ to return a valid plan $\pi(\mathbf{P}) = \vec{\alpha}$ if a plan exists for $\mathbf{P}$, or otherwise determine that no plan exists for $\mathbf{P}$. Examples of programmatic plans include finite state controllers (Bonet, Palacios, and Geffner 2009, 2010; Hu and De Giacomo 2011; Aguas, Jiménez, and Jonsson 2018), policies derived from lifted rules (Srivastava et al. 2011; Illanes and McIlraith 2019; Francès, Bonet, and Geffner 2021) and general-purpose programs (Levesque 2005; Srivastava, Immerman, and Zilberstein 2008; Segovia-Aguas, Celorrio, and Jonsson 2024; Silver et al. 2024).

3 Generalised Planning via Goal Regression

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 $\mathscr{P}_{\text{train}}$ 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 that determines its execution priority as is common in logic programming. Figure 2 illustrates the synthesis procedure and structure of MOOSE programs.

Definition 1 (MOOSE Rule)**.** Let $\mathbf{GP} = \langle \mathscr{P}_{\text{train}}, \mathscr{P}_{\text{test}} \rangle$ 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 2 (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 $f : \mathit{var}(r) \to \mathbf{P}[O]$ 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

Algorithm 1: MOOSE Program Synthesis

```
Input: Training problems $\mathscr{P}_{\text{train}} = \{\mathbf{P}^{(1)}, \dots, \mathbf{P}^{(n_t)}\}$,
       and number of goal permutations $n_p \in \mathbb{N}$
       (default: 3).
Output: MOOSE program $\pi$.
1  $\pi \leftarrow \emptyset$
2  for $i = 1, \dots, n_t$ do
3      $n_g \leftarrow |\mathbf{P}^{(i)}[g]|$
4      for $j = 1, \dots, \min(n_p, n_g!)$ do
5          $s \leftarrow \mathbf{P}^{(i)}[s_0]$ ; $\vec{g} \leftarrow \text{newPermutation}(\mathbf{P}^{(i)}[g])$
6          for $k = 1, \dots, n_g$ do
7              $g' \leftarrow \{\vec{g}_k\}$
8              $\vec{\alpha} \leftarrow \text{optimalPlan}(\mathbf{P}^{(i)}_{s,g'})$
9              if $\vec{\alpha} = \perp$ then continue
10             $\pi \leftarrow \pi \cup \text{extractRules}(\vec{\alpha}, g')$ // Alg. 2
11             $s \leftarrow \text{succ}(s, \vec{\alpha})$
12 return $\pi$
```

function $\text{grounding}(r, s, \mathbf{P}[g]) = \mathit{actions}(r)|_f$, where f is some grounding. Otherwise, $\text{grounding}(r, s, \mathbf{P}[g]) = \perp$.

Definition 3 (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 quantified actions and predicates. Lifting will be used in the synthesis module to generate reusable rules.

Definition 4 (Lifting)**.** Let s and g be finite sets of ground atoms and $\vec{\alpha} = a_1, \dots, a_m$ a sequence of ground actions. Let $o_1, \dots, o_q$ 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 $\mathit{var} = \{v_1, \dots, v_q\}$ and lift each action and atom by replacing each constant o_i with its corresponding free variable v_i in var. We denote

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

with $\vec{\alpha}'$ 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 regresses the corresponding singleton goal, and the regressed goal and plan is then lifted into a lifted macro action rule.

The main algorithm gradually builds from 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

Algorithm 2: Rule Extraction Routine

```
Input: Sequence of actions $\vec{\alpha}$ and set of atoms $g$.
Output: MOOSE rules with precedence values $\pi$.
1  $\pi \leftarrow \emptyset$; $s \leftarrow g$
2  for $i = |\vec{\alpha}|, \dots, 1$ do
3      $s \leftarrow \text{regr}(s, \vec{\alpha}_i)$ ; $r \leftarrow \text{lift}(s, g, \vec{\alpha}_{[i:]})$
4      $\pi \leftarrow \pi \cup \{(r, |\vec{\alpha}| - i + 1)\}$
5  return $\pi$
```

Algorithm 3: MOOSE Program Instantiation

```
Input: A planning problem $\mathbf{P}$ and MOOSE program $\pi$.
Output: A plan $\vec{\alpha}$ and success or failure status.
1  $s \leftarrow \mathbf{P}[s_0]$; $\vec{\alpha} \leftarrow []$ // empty sequence
2  while $\mathbf{P}[g] \not\subseteq s$ do
3      $\vec{\beta} \leftarrow \perp$
4      for $r \in \pi$ in ascending precedence values do
5          $\vec{\beta} \leftarrow \text{grounding}(r, s, \mathbf{P}[g])$
6          if $\vec{\beta} \neq \perp$ then break
7      if $\vec{\beta} = \perp$ or detected cycle then return $\vec{\alpha}$, failure
8      $\vec{\alpha} \leftarrow \vec{\alpha}; \vec{\beta}$ // sequence concatenation
9      $s \leftarrow \text{succ}(s, \vec{\beta})$
10 return $\vec{\alpha}$, success
```

each goal ordering and training problem, MOOSE finds plans via an optimal planner (Line 8) with the singleton goals (Line 7) in order (Line 6), while progressing the current state along the way (Line 11). 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 9). Otherwise, if a plan exists, then we extract rules from the plan and add them to the incumbent plan (Line 10) as described in Algorithm 2. 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 $\vec{\alpha}$ and then lifts the corresponding regressed state s, goal g, and suffix of the plan into a rule r (Lines 2 to 3). Then we append the rule alongside its cost-to-go from the partial state s to goal g under the plan suffix as its precedence value (Line 4).

Example 3 (Transportation Program Synthesis)**.** *We illustrate Lines 8 to 10 of Algorithm 1 with our running transportation example from Example 2 as a training problem. Note that the problem already has a singleton goal $g = \{at(cake, kitchen)\}$. The plan $\vec{\alpha}$ from the example is the only optimal plan and thus is the output of Line 8. Since a plan exists for the problem, Line 9 does nothing. Line 10 triggers Algorithm 2 which begins by setting s to g. The first regressed state under the final action in the plan $a = putDown(cake, kitchen)$ is $\text{regr}(s, a) = \{atRobot(kitchen), holding(cake)\}$. Note that the fact $at(dog, kitchen)$ is ignored during regression, indicating that it is irrelevant towards the goal.*

The regressed state and the singleton goal is lifted to con-

struct the following rule

$$\begin{aligned} var &= \{?obj, ?loc\} \\ stateCond &= \{atRobot(?loc), holding(?obj)\} \\ goalCond &= \{at(?obj, ?loc)\} \\ actions &= putDown(?obj, ?loc) \end{aligned}$$

Repeating the procedure again under the penultimate action in the plan gives us the next regressed state $\{atRobot(backyard), holding(cake)\}$ *which is lifted to construct the rule*

$$\begin{aligned} var &= \{?obj, ?l1, ?l2\} \\ stateCond &= \{atRobot(?l1), holding(?obj)\} \\ goalCond &= \{at(?obj, ?l2)\} \\ actions &= move(?l1, ?l2), putDown(?obj, ?l2) \end{aligned}$$

The procedure is repeated two more times as $\vec{\alpha}$ *has four actions, resulting in four rules added to* π. *The first two steps of regression and lifting are further illustrated in Figure 2.*

3.2 Satisficing Planning with MOOSE

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 summarises the execution procedure for an input planning problem and MOOSE program. 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 4 to 6), from which the corresponding macro action is added to the incumbent plan and applied to the current state (Lines 8 to 9). The loop breaks once the goal is reached (Line 2), no actions can be queried from the set of rules, or a cycle is encountered (Line 7).

3.3 Optimal Planning with MOOSE

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, Hoffmann, and Nebel 2005) representing search control for optimal planners that support axioms. Theorem 18 later formalises conditions under which encodings of MOOSE programs preserve optimal solutions.

We now extend a given GP domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{C}, \mathcal{A} \rangle$ with a MOOSE program π. We add predicates p_g and p_{ug} for each $p \in \mathcal{P}$, representing goals in a planning problem and unachieved goals in the current state, respectively. Then each state in a problem $\mathbf{P}$ is extended with atoms $p_g(\vec{o})$ for each goal atom $p(\vec{o}) \in \mathbf{P}[g]$ following Martín and Geffner (2004). For each predicate p we introduce the axiom

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

for computing unachieved goals. Then for each action schema $a \in \mathcal{A}$ we add a new derived 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

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

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. We note that the axioms do not exhibit recursion and thus can be encoded via disjunctive preconditions (Davidson and Garagnani 2002). Differently to previous works that prune the search space with lifted rules (Bacchus and Kabanza 2000; Yoon, Fern, and Givan 2008; Krajnanský et al. 2014), our approach does not require writing new solvers but instead makes use of existing planners that support more expressive PDDL features.

	General Case		PTIME Subplans	
TGI (5)	PSPACE-cmpl.	(8)	in P	(10)
SGI (6)	PSPACE-cmpl.	(9)	NP-cmpl.	(11)
OGI (7)	PSPACE-cmpl.	(12)	NP-cmpl.	(13)

Table 1: Computational complexity of planning problems exhibiting notions of goal independence. Definitions and theorem references are indicated in brackets.

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 17 and 18). Proofs of all statements are provided in Appendix B. The idea is that under assumptions on the complexity of a GP problem $\langle \mathscr{P}_{\text{train}}, \mathscr{P}_{\text{test}} \rangle$ and given sufficient training problems, Algorithm 1 synthesises generalised plans from $\mathscr{P}_{\text{train}}$ that are sound and complete for solving problems in $\mathscr{P}_{\text{test}}$, and furthermore finds optimal plans when MOOSE rules are used for search as described in Section 3.3. We begin classifying planning domains based on the separability of goals.

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, as we see later in our experiments, a non-trivial portion of benchmark planning domains are P-time solvable and furthermore exhibit goals that can be achieved independently from one another. In this section, we formalise three variants of goal independence (TGI, SGI, OGI) depending on whether serialisation is required and the effects of goal independence on plan optimality. We further prove their computational complexities summarised in Table 1.

Our first notion of goal independence, TGI, describes planning problems that can be solved by solving for each goal conjunct optimally in *any order*.

Definition 5 (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.

Our second notion of goal independence, SGI, generalises TGI by relaxing the requirement that the aforementioned greedy algorithm is valid for any order of goal conjuncts. SGI only requires that the greedy algorithm is valid for *at least one order* of goal conjuncts.

Definition 6 (Serialisable Goal Independence)**.** A planning problem $\mathbf{P}$ exhibits serialisable 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.

Our final notion of goal independence, OGI, strengthens the previous independence notion by ensuring that there is at least one order of goal conjuncts for which the greedy algorithm solves the problem *optimally*.

Definition 7 (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 5 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". pOGI is defined similarly where (2a) guesses an optimal plan in polynomial time.

Korf (1987, Sections 4.3 and 4.4) introduce similar concepts of goal independence corresponding to our TGI and SGI definitions that form the foundation of heuristics for search. We next theoretically analyse the computational complexity of problems exhibiting the GI definitions with proofs in Appendix B. We say that a GP problem $\mathbf{GP} = \langle \mathscr{P}_{\text{train}}, \mathscr{P}_{\text{test}} \rangle$ exhibits TGI if every problem in $\mathscr{P}_{\text{train}} \cup \mathscr{P}_{\text{test}}$ exhibits TGI, and analogously for SGI and OGI. We then let PLANSAT($\mathbf{GP}$) denote the computational problem of deciding if a plan exists for a problem in $\mathscr{P}_{\text{test}}$. The following statements show that without the polynomial time constraint of step (2a) of the aforementioned greedy algorithm, TGI and SGI do not make planning easier.

Proposition 8. *PLANSAT($\mathbf{GP}$) of a GP problem $\mathbf{GP}$ exhibiting TGI is* PSPACE*-complete.*

Corollary 9. *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 10. *PLANSAT($\mathbf{GP}$) of a GP problem $\mathbf{GP}$ exhibiting pTGI is in* P*.*

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

Next, given that SGI is a special case of OGI, we also have the following statements.

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

Corollary 13. *PLANSAT($\mathbf{GP}$) of a GP problem $\mathbf{GP}$ exhibiting pOGI 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 similarly to (Drexler et al. 2024), in contrast to work by Sievers et al. (2019) which reduces problems to graph automorphisms.

Definition 14 (Equivalence Relation)**.** Given a GP problem $\langle \mathscr{P}_{\text{train}}, \mathscr{P}_{\text{test}} \rangle$, we define a relation $\sim_U$ on planning problems in $\mathscr{P}_{\text{train}} \cup \mathscr{P}_{\text{test}}$ by $\mathbf{P}_1 \sim_U \mathbf{P}_2$ if there exists a bijective mapping $f : \mathbf{P}_1[O] \to \mathbf{P}_2[O]$ 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 15. *The relation $\sim_U$ on planning problems of any given GP problem is an equivalence relation.*

Proposition 16. *Suppose $\mathbf{P}_1 \sim_U \mathbf{P}_2$ and let $f : \mathbf{P}_1[O] \to \mathbf{P}_2[O]$ 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 $o_1, \ldots, o_n \in \mathbf{P}_1[O]$.*

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 sufficiently many training data, MOOSE can construct a database of rules for TGI$_C$ problems that 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 which is exponential in the input in the worst case.

Theorem 17 (PLANSAT soundness and completeness conditions)**.** *There exists a set $\mathscr{P}_{train}$ such that for all GP problems $\mathbf{GP} = \langle \mathscr{P}_{train}, \mathscr{P}_{test} \rangle$ exhibiting TGI$_C$, Algorithm 3 using the plan π synthesised from Algorithm 1 with $\mathscr{P}_{train}$ is sound and complete for $\mathscr{P}_{test}$ for satisficing planning.*

The bound on the size of training problems in the proof of Theorem 17 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. 2011) framework is provably equivalent to fully observable non-deterministic (FOND) planning (Bonet and Geffner 2020) which is known to be EXPTIME-complete. A fruitful next step is to develop a learning algorithm that learns to generate and select what training data is required, possibly given implicitly in the input GP problem (Srivastava, Immerman, and Zilberstein 2011; Grundke, Röger, and Helmert 2024). Now we state the main theorem for optimal planning and provide a counterexample to the theorem for when the OGI assumption is dropped in example 4 in the Appendix.

Theorem 18 (PLANOPT soundness and completeness conditions)**.** *There exists a set $\mathscr{P}_{train}$ such that for all GP problems $\mathbf{GP} = \langle \mathscr{P}_{train}, \mathscr{P}_{test} \rangle$ exhibiting TGI$_C$ and OGI, an optimal planner run on the transformation in Section 3.3 via the generalised plan π learned from Algorithm 1 with $\mathscr{P}_{train}$ is sound and complete for $\mathscr{P}_{test}$ for optimal planning.*

5 Experiments

We conduct experiments to answer the following questions. (**Q1**) How often can planning benchmark problems be solved by the TGI algorithm (Definition 5)? (**Q2**) How does MOOSE compare to existing generalised planners in *synthesis costs*? (**Q3**) How does MOOSE compare to existing (generalised) planners in *instantiation costs*? (**Q4**) How does MOOSE compare to existing (generalised) planners in *solution quality*? (**Q5**) Can learned MOOSE rules encoded as axioms help speed up existing *optimal planners*?

MOOSE Implementation MOOSE is implemented primarily in Python, but make use of the following tools and planners: the `pddl` parser (Favorito, Fuggitti, and Muise 2025) 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 8 of Algorithm 1; and SYMK (Speck et al. 2019; Speck, Seipp, and Torralba 2025) for optimal planning with MOOSE rules encoded as axioms described in Section 3.3. We further implement an optimisation for generalised plan instantiation that tries to fire the previous successfully fired rule first during Lines 4 and 6 of Algorithm 3 to reduce grounding effort.

(Q1) How often can planning benchmark problems be solved by the TGI algorithm (Definition 5)? To answer this question, we use the STRIPS 1998-2023 International Planning Competition (IPC) benchmark suite which consists of 38 domains. Inspired by the experimental setup by Lipovetzky and Geffner (2012) for testing effective width of planning problems, we test the number of planning problems that can be solved by the TGI algorithm. For each problem, we randomise a goal ordering and run FAST DOWNWARD's implementation of A* search with the LM-cut heuristic (Helmert and Domshlak 2009) on each goal atom in order as described in Definition 5 with an 8GB memory and 7200s runtime limit. The output is then validated (Howey, Long, and Fox 2004).

In 13 (34.2%) domains, all returned outputs under the resource constraints are valid, which suggests the possibility that these domains exhibit TGI. In 19 (50.0%) domains, at least half of the returned outputs are valid and in 27 (38.0%) domains, at least one returned output is valid. In total, of 1184 problems for which an output was returned in the resource constraints, 590 (49.8%) were valid. These results suggest that a non-trivial portion of planning domains can be solved by the greedy TGI algorithm.

(Q2) How does MOOSE compare to existing generalised planners in *synthesis costs*? To answer this question, we make use of the classical ESHO domains Barman, Ferry, Gripper, Logistics, Miconic, Rovers, Satellite, and Transport, where path-finding components are removed from Rovers and Transport. Some but not all domains exhibit the TGI assumption. Appendix D provides further benchmark details, distributions of problem object sizes, and number of training and testing problems. We compare against the sketch learner (SLEARN) (Drexler, Seipp, and Geffner 2022) with width hyperparameters in $\{0, 1, 2\}$. All approaches are given a 32GB memory and 12 hour runtime limit for generalised plan synthesis. Each experiment is repeated 5 times.

Figure 3 summarises synthesis metrics. MOOSE completed synthesis in the given synthesis budgets while SLEARN did not learn domain knowledge for 3 domains with more compute across all hyperparameters. The best SLEARN configuration is faster than MOOSE at synthesising generalised plans for simpler domains (Ferry, Miconic and Transport) while MOOSE is faster on 5 out of 8 domains. However, MOOSE consumes less than 1GB of memory and uses less memory than SLEARN across all 8 domains.

(Q3) How does MOOSE compare to existing (generalised) planners in *instantiation costs*? We use the domains and problems from the previous question and the additional numeric planning domains NFerry, NMiconic, NMinecraft, and NTransport. Further details are again provided in Appendix D. As baselines, we compare against LAMA (Richter and Westphal 2010), the state-of-the-art standalone satisficing planner in the 2023 IPC Learning Track and the best SLEARN configuration for every problem for classical planning. 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. MOOSE and all baselines are given an 8GB memory and 1800s runtime limit for planning. SLEARN and MOOSE experiments are repeated at most 5 times corresponding to the repeated 5 trained models from the previous question.

Figure 3 displays the cumulative coverage of all planners, where a point (x, y) denotes the number y of problems that can be individually solved in x seconds by a planner. The dotted black line denotes the number of problems in each benchmark suite. All 5 seeds of MOOSE solve all numeric planning and classical testing problems except 2 Logistics seeds that fail on a single problem. In comparison from looking at Figure 3, other baseline planners solve $> 20\%$ fewer problems. From Table 4 in Appendix E.1, only MOOSE seeds can solve every problem for each domain.

(Q4) How does MOOSE compare to existing (generalised) planners in *solution quality*? We employ the experiments from the previous question and compare MOOSE to the best performing classical and numeric planners LAMA and MRP+HJ, respectively. Comparisons are illustrated in Figures 25 and 26 in Appendices E.5 and E.6. MOOSE achieves higher quality plans on 3 domains than MRP+HJ (NMiconic, NMinecraft, and NTransport), and worse plan quality on 1 domain (NFerry) for numeric planning. MOOSE achieves

	Time (s)				Memory (MB)			
	SLEARN-0	SLEARN-1	SLEARN-2	MOOSE	SLEARN-0	SLEARN-1	SLEARN-2	MOOSE
Barman	–	–	–	**202**	–	–	–	**184**
Ferry	21	12	**2**	9	184	134	76	**52**
Gripper	**3**	9	45	10	66	142	391	**64**
Logistics	–	–	–	**71**	–	–	–	**73**
Miconic	57	**1**	3	12	381	56	125	**52**
Rovers	–	–	–	**534**	–	–	–	**187**
Satellite	–	–	1559	**514**	–	–	7598	**82**
Transport	–	**12**	**12**	21	–	114	129	**80**

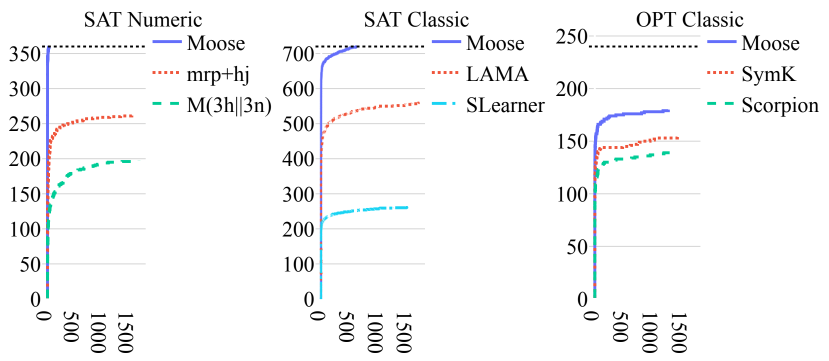


Figure 3: **Left:** average time and memory usage for synthesis (↓). Lowest values are indicated in colour and bold font. **Right:** cumulative coverage (y-axis) of planners over time (x-axis) for different planning settings. Higher values are better (↑).

higher quality plans than LAMA on 5 domains (Barman, Ferry, Miconic, Rovers, Transport), and worse plans on 3 domains (Gripper, Logistics, Satellite) for classical planning.

We further conducted experiments via regressing over goal conjuncts instead of singleton goals in Algorithm 1. More rules are synthesised this way leading to higher instantiation costs, but in turn plan quality improved across 7 out of 8 classical domains. The effects of regressing over goal conjuncts are presented in greater detail in Appendix F.

(Q5) Can learned MOOSE rules encoded as axioms help speed up existing *optimal planners*? For optimal planning, we compare against SCORPION (Seipp, Keller, and Helmert 2020), the best standalone optimal planner in the 2023 IPC Optimal Track, and SYMK (Speck, Seipp, and Torralba 2025). As before, MOOSE and the baselines are given an 8GB memory and 1800s runtime limit. We note that SCORPION and SYMK are theoretically optimal planners while MOOSE is not guaranteed optimal in practice.

MOOSE solves on average 182.8 out of 240 classical testing problems optimally. Although the transformation from policies learned from finite training data does not guarantee the preservation of optimal plans, the plans output by MOOSE are empirically optimal. Both MOOSE and Scorpion achieve the best (tied) performance on 4 domains out of 8. We also note that MOOSE matches or outperforms the base planner SYMK on all domains except Gripper. This observation suggests that the reduction in search space from encoding learned policies via axioms usually outweigh the cost of evaluating such axioms.

6 Related Work, Discussion and Conclusion

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 which in turn means that the horizons of testing problems are often bounded by those seen in the training set. LOOM (Illanes and McIlraith 2019) 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 FOND planner PRP (Muise, McIlraith, and Beck 2012) to synthesise generalisable policies via regression which satisfy a policy termination test (Srivastava et al. 2011). 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 take a *top-down* approach of performing lifted regression to generate relevant lifted features, and LOOM employs ground regression on a top-down abstraction for synthesising generalised policies.

More generally, MOOSE lies in the class of generalised planners that synthesise generalised plans by sampling from training problems (cf. the survey by Celorrio, Segovia-Aguas, and Jonsson (2019) for more examples). PG3 (Yang et al. 2022), which performs heuristic search over a space of generalised policies (Segovia-Aguas, Jiménez, and Jonsson 2021; Segovia-Aguas, Celorrio, and Jonsson 2024), also uses goal regression for handling 'missed' states. Generalised plans synthesised from sampling have been represented as (deep) policies (Toyer et al. 2018, 2020; Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021; Ståhlberg, Bonet, and Geffner 2022; Chen et al. 2025), heuristic functions (Shen, Trevizan, and Thiébaux 2020; Karia and Srivastava 2021; Chen, Thiébaux, and Trevizan 2024; Corrêa, Pereira, and Seipp 2025) and Python code (Silver et al. 2024).

In conclusion, we have introduced a new generalised planner, MOOSE, for both satisficing and optimal planning by leveraging goal regression and goal independence. 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. Future work involves extending MOOSE to handle domains requiring transitive closure computations and weaker planning domain assumptions.

Acknowledgements

DC and TH carried out this work at the Vector Institute, where DC was 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. TH was funded by the Federal Ministry of Education and Research (BMBF) and the Ministry of Culture and Science of the German State of North Rhine-Westphalia (MKW) under the Excellence Strategy of the Federal Government and the Länder. 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, J. S.; Jiménez, S.; and Jonsson, A. 2018. Computing Hierarchical Finite State Controllers With Classical Planning. *J. Artif. Intell. Res.*, 62: 755–797.

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting Regression in Planning. In *IJCAI*.

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

Benyamin, Y.; Mordoch, A.; Shperberg, S.; and Stern, R. 2024. Solving Minecraft Tasks via Model Learning. In *Proceedings of the PRL Workshop Series: Bridging the Gap Between AI Planning and Reinforcement Learning*.

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *AAAI*.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artif. Intell.*, 129: 5–33.

Bonet, B.; and Geffner, H. 2020. Qualitative Numeric Planning: Reductions and Complexity. *J. Artif. Intell. Res.*, 69: 923–961.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners. In *ICAPS*.

Bonet, B.; Palacios, H.; and Geffner, H. 2010. Automatic Derivation of Finite-State Machines for Behavior Control. In *AAAI*.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.*, 69(1-2): 165–204.

Celorrio, S. J.; Segovia-Aguas, J.; and Jonsson, A. 2019. A review of generalized planning. *Knowl. Eng. Rev.*, 34: e5.

Chen, D. Z.; and Thiébaux, S. 2024a. Graph Learning for Numeric Planning. In *NeurIPS*.

Chen, D. Z.; and Thiébaux, S. 2024b. Novelty Heuristics, Multi-Queue Search, and Portfolios for Numeric Planning. In *SOCS*.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *AAAI*.

Chen, D. Z.; Zenn, J.; Cinquin, T.; and McIlraith, S. A. 2025. Language Models For Generalised PDDL Planning: Synthesising Sound and Programmatic Policies. In *Proceedings of the 18th European Workshop on Reinforcement Learning (EWRL)*.

Corrêa, A. B.; and De Giacomo, G. 2024. Lifted Planning: Recent Advances in Planning Using First-Order Representations. In *IJCAI*.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *ICAPS*.

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code. In *NeurIPS*.

Davidson, M.; and Garagnani, M. 2002. Pre-processing planning domains containing Language Axioms. In *Proc. of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2002)*.

Dong, H.; Shi, Z.; Zeng, H.; and Liu, Y. 2025. An Automatic Sound and Complete Abstraction Method for Generalized Planning with Baggable Types. *AAAI*, 39(14): 14875–14884.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width. In *ICAPS*.

Drexler, D.; Ståhlberg, S.; Bonet, B.; and Geffner, H. 2024. Equivalence-Based Abstractions for Learning General Policies. In *KR*.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artif. Intell.*, 76: 75–88.

Favorito, M.; Fuggitti, F.; and Muise, C. 2025. pddl. Accessed from https://github.com/ai-Planning/pddl.

Fikes, R.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artif. Intell.*, 3(1-3): 251–288.

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

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In *AAAI*.

Fritz, C.; and McIlraith, S. A. 2007. Monitoring Plan Optimality During Execution. In *ICAPS*.

Fuentetaja, R.; and de la Rosa, T. 2016. Compiling irrelevant objects to counters. Special case of creation planning. *AI Commun.*, 29(3): 435–467.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. ISBN 0-7167-1044-7.

Gretton, C.; and Thiébaux, S. 2004. Exploiting First-Order Regression in Inductive Policy Selection. In *UAI*.

Grundke, C.; Röger, G.; and Helmert, M. 2024. Formal Representations of Classical Planning Domains. In *ICAPS*.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers.

Hearn, R. A.; and Demaine, E. D. 2005. 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.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *AIPS*.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *ICAPS*.

Hipp, D. R. 2020. SQLite. Accessed from https://www.sqlite.org/index.html.

Hofmann, T.; Niemueller, T.; and Lakemeyer, G. 2020. Macro Operator Synthesis for ADL Domains. In *ECAI*.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI*.

Hu, Y.; and De Giacomo, G. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. In *IJCAI*.

Illanes, L.; and McIlraith, S. A. 2017. Numeric Planning via Abstraction and Policy Guided Search. In *IJCAI*.

Illanes, L.; and McIlraith, S. A. 2019. Generalized Planning via Abstraction: Arbitrary Numbers of Objects. In *AAAI*.

Kaelbling, L. P.; and Lozano-Pérez, T. 2011. Pre-image Backchaining in Belief Space for Mobile Manipulation. In *Robotics Research*.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In *AAAI*.

Khardon, R. 1999. Learning Action Strategies for Planning Domains. *Artif. Intell.*, 113: 125–148.

Korf, R. E. 1987. Planning as Search: A Quantitative Approach. *Artif. Intell.*, 33(1): 65–88.

Krajnanský, M.; Hoffmann, J.; Buffet, O.; and Fern, A. 2014. Learning Pruning Rules for Heuristic Search Planning. In *ECAI*.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2022. The LM-Cut Heuristic Family for Optimal Numeric Planning with Simple Conditions. *J. Artif. Intell. Res.*, 75: 1477–1548.

Levesque, H. J. 2005. Planning with Loops. In *IJCAI*.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *ECAI*.

Liu, X.; Pesaranghader, A.; Li, H.; Sukcharoenchaikul, P.; Kim, J.; Sadhu, T.; Jeon, H.; and Sanner, S. 2025. Open-World Planning via Lifted Regression with LLM-Inferred Affordances for Embodied Agents. In *ACL*.

Lozano-Perez, T.; Mason, M. T.; and Taylor, R. H. 1984. Automatic synthesis of fine-motion strategies for robots. *Int. J. Robot. Res.*, 3(1): 3–24.

Martín, M.; and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Appl. Intell.*, 20: 9–19.

McDermott, D.; Ghallab, M.; Howe, A. E.; Knoblock, C. A.; Ram, A.; Veloso, M. M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL-the planning domain definition language. Technical report.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2024. PRP Rebooted: Advancing the State of the Art in FOND Planning. In *AAAI*.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *ICAPS*.

Pang, B.; and Holte, R. C. 2011. State-Set Search. In *SOCS*.

Reiter, R. 1991. 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.

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

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Riddle, P.; Douglas, J.; Barley, M.; and Franco, S. 2016. Improving performance by reformulating PDDL into a bagged representation. In *Proceedings of the 8th Workshop on Heuristic Search for Domain-independent Planning*.

Sanner, S.; and Boutilier, C. 2009. Practical solution techniques for first-order MDPs. *Artif. Intell.*, 173(5-6): 748–788.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *IJCAI*.

Scala, E.; Saetti, A.; Serina, I.; and Gerevini, A. E. 2020. Search-Guidance Mechanisms for Numeric Planning Through Subgoaling Relaxation. In *ICAPS*.

Segovia-Aguas, J.; Celorrio, S. J.; and Jonsson, A. 2024. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artif. Intell.*, 330: 104097.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2021. Generalized Planning as Heuristic Search. In *ICAPS*.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *J. Artif. Intell. Res.*, 67: 129–167.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *ICAPS*.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2019. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. In *ICAPS*.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. In *AAAI*.

Speck, D.; Geißer, F.; Mattmüller, R.; and Torralba, Álvaro. 2019. Symbolic Planning with Axioms. In *ICAPS*.

Speck, D.; Seipp, J.; and Torralba, Á. 2025. Symbolic Search for Cost-Optimal Planning with Expressive Model Extensions. *J. Artif. Intell. Res.*, 82: 1349–1405.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning Generalized Plans Using Abstract Counting. In *AAAI*.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2): 615–647.

Srivastava, S.; Zilberstein, S.; Immerman, N.; and Geffner, H. 2011. Qualitative Numeric Planning. In *AAAI*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *ICAPS*.

Sussman, G. J. 1973. *A Computational Model of Skill Acquisition*. Ph.D. thesis, MIT.

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

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.*, 168: 38–69.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *J. Artif. Intell. Res.*, 68: 1–68.

Toyer, S.; Trevizan, F. W.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies With Deep Learning. In *AAAI*.

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

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

Xu, D.; Martín-Martín, R.; Huang, D.; Zhu, Y.; Savarese, S.; and Fei-Fei, L. 2019. Regression Planning Networks. In *NeurIPS*.

Yang, R.; Silver, T.; Curtis, A.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PG3: Policy-Guided Planning for Generalized Policy Generation. In *IJCAI*.

Yoon, S. W.; Fern, A.; and Givan, R. 2008. Learning Control Knowledge for Forward Search Planning. *J. Mach. Learn. Res.*, 9: 683–718.

A Numeric Planning Definitions

We can extend the definition of a planning problem 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 19 (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 STRIPS planning 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 $\langle var(a), pre(a), add(a), del(a), num\_eff(a) \rangle$ where $var(a)$ is a set of parameter terms, the preconditions $pre(a)$ is a set of conditions, $add(a)$ and $del(a)$ are add and delete lists of predicates as in the classical case, $num\_eff(a)$ 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 $var(a) \cup \mathcal{O}$. An action a is applicable in a state s if s satisfies $pre(a)$. In this case, its successor $\mathrm{succ}(s, a)$ is the state where the effects $num\_eff(a)$ 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 STRIPS planning to handle numeric conditions and effects for the class of abacus program numeric problems.

Definition 20 (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 for STRIPS regression, 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 of Theorems

This section proves theorems stated in Section 4.

B.1 Complexity Proofs

Proof Sketch for Proposition 8. Membership follows from PSPACE-completeness of planning for fixed domains (Bylander 1994; Erol, Nau, and Subrahmanian 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 9. Membership again follows from PSPACE-completeness of planning for fixed domains. Hardness follows from Proposition 8 as TGI is a special case of SGI. □

Proof Sketch for Corollary 12. Membership again follows from PSPACE-completeness of planning for fixed domains. Hardness follows from Corollary 9 as SGI is a special case of OGI. □

Proof Sketch for Proposition 10. 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 Proposition 11. 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 Corollary 13. For NP membership, one guesses the correct ordering of goals after which running the greedy algorithm is in NP time. For NP-hardness, this follows from the NP-hardness of pSGI which is a special case of pOGI. □

B.2 Soundness and Completeness Proofs

Proof Sketch for Proposition 15. Reflexivity, symmetry and transitivity follows from the usage of bijective functions in the definition of $\sim_U$. ☐

Proof Sketch for Proposition 16. The statement follows by definitions and bijectiveness of f. ☐

Proof Sketch for Theorem 17. We need to show that there exists a set $\mathscr{P}_{\text{train}}$ such that a MOOSE plan synthesised via Algorithm 1 from $\mathscr{P}_{\text{train}}$ can solve any generalised planning problem $\mathbf{GP} = \langle \mathscr{P}_{\text{train}}, \mathscr{P}_{\text{test}} \rangle$ exhibiting TGI$_C$. 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 5, 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 $\mathscr{P}$ 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. Thus, the execution of Algorithm 3 is sound as if a plan exists and is returned, then the execution of actions in sequence is valid and achieves the goal, and otherwise if no plan exists, then the algorithm would terminate and not return a plan.

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 16, 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 Corollary 12. 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 18. Completeness follows from the completeness of rules as discussed in Theorem 17. 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 4 (Theorem 18 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 5 of TGI returns a plan of length 4. However, the optimal plan has length 3.*

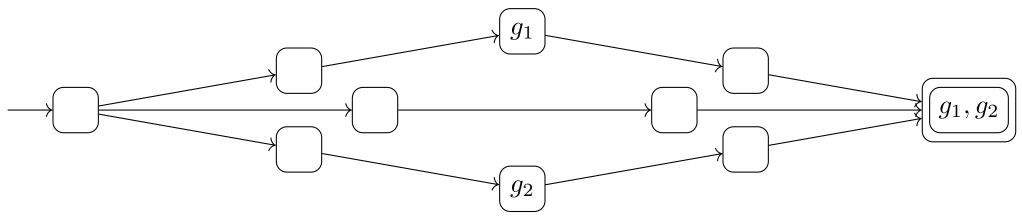


Figure 4: A planning problem illustrating the necessity of the OGI assumption for learning provably optimal policies in Theorem 18. Nodes represent states and arrows represent transitions between states.

C True Goal Independence Experiments

Results for the Agricola domain are omitted as no problem could be solved in the given resource limits.

Domain	OOR	Inval	Val	Val (%)
barman	1	23	16	41.0
blocks	0	35	0	0.0
childsnack	5	0	15	100.0
data	15	0	5	100.0
depot	1	14	7	33.3
driverlog	0	18	2	10.0
elevators	10	0	40	100.0
floortile	0	40	0	0.0
freecell	31	0	49	100.0
ged	1	19	0	0.0
grid	0	3	2	40.0
gripper	0	0	20	100.0
hiking	17	0	3	100.0
logistics	5	0	58	100.0
miconic	0	0	150	100.0
movie	0	30	0	0.0
mprime	11	0	24	100.0
mystery	8	4	18	81.8
nomystery	0	20	0	0.0
parking	0	40	0	0.0
pegsol	0	50	0	0.0
pipesworld	37	51	12	19.1
rovers	0	0	40	100.0
satellite	6	15	15	50.0
scanalyzer	0	46	4	8.0
snake	0	20	0	0.0
sokoban	0	48	2	4.0
spider	11	9	0	0.0
storage	0	27	3	10.0
termes	17	3	0	0.0
tetris	0	20	0	0.0
thoughtful	15	1	4	80.0
tidybot	1	2	17	89.5
tpp	8	0	22	100.0
transport	44	0	26	100.0
visitall	18	0	22	100.0
woodworking	0	48	2	4.0
zenotravel	0	8	12	60.0

Table 2: Results for the procedure described in (**Q1**) in Section 5. OOR denotes that planning did not complete in the given resources. Inval denotes that a returned plan is invalid or that the procedure encountered a deadend. Val denotes that a returned plan is valid. Val (%) is computed by $100\frac{\text{Val}}{\text{Val}+\text{Inval}}$.

D Additional Generalised Planning Benchmark Details

Table 3 displays the ranges of number of objects present in training and testing problems for each domain. The Ferry, Miconic, Rovers, Satellite, and Transport domains and problems are taken from the 2023 IPC Learning Track (Taitler et al. 2024) with modifications that remove the path-finding components of Rovers and Transport, and that increase the number of objects in Miconic problems. Barman, Gripper and Logistics have newly introduced training and testing problem splits for generalised planning. The NFerry, NMiconic, and NTransport domains are taken from (Chen and Thiébaux 2024a) with path-finding removed from NTransport. NMinecraft originates from the Minecraft Pogo Stick domain (Benyamin et al. 2024) while we introduced training and testing problem splits for generalised planning.

		Train		Test	
Domain	Obj. Types	Min	Max	Min	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
Ferry	Σ	3	8	7	1461
	car	1	2	2	974
	location	2	6	5	487
Gripper	Σ	3	5	11	48500
	balls	3	5	11	48500
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	1950
	floor	2	7	4	980
	passenger	1	4	1	970
Rovers	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

(a) Classical planning.

		Train		Test	
Domain	Objects	Min	Max	Min	Max
NFerry	Σ	7	9	11	1465
	car	1	2	2	974
	location	2	3	5	487
NMiconic	Σ	7	15	9	685
	floor	2	7	4	196
	passenger	1	4	1	485
NMinecraft	Σ	5	5	15	2100
	cell	4	4	11	1800
	pogo sticks	1	1	4	300
NTransport	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

(b) Numeric planning.

Table 3: Training and testing object distributions.

E Additional Generalised Planning Results

E.1 Coverage Tables

Domain	M(3h‖3n)	MRP+HJ	MOOSE
NFerry	60	61	**90.0**
NMiconic	63	71	**90.0**
NMinecraft	30	66	**90.0**
NTransport	44	64	**90.0**
∑ (360)	197	262	**360.0**

(a) Satisficing numeric planning.

Domain	SLEARN-0	SLEARN-1	SLEARN-2	LAMA	MOOSE
Barman	0.0	0.0	0.0	49	**90.0**
Ferry	15.0	67.0	60.0	69	**90.0**
Gripper	59.6	50.8	33.0	65	**90.0**
Logistics	0.0	0.0	0.0	77	**89.6**
Miconic	68.8	72.6	67.8	77	**90.0**
Rovers	0.0	0.0	0.0	66	**90.0**
Satellite	0.0	29.2	34.6	89	**90.0**
Transport	0.0	63.0	46.8	66	**90.0**
∑ (720)	143.4	282.6	242.2	558	**719.6**

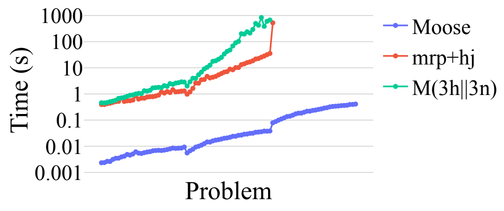
(b) Satisficing classical planning.

Domain	Blind	LMCUT	SCORPION	SYMK	MOOSE
Barman	0	0	0	12	**24.6**
Ferry	10	18	17	18	**30.0**
Gripper	9	8	7	**30**	27.0
Logistics	8	15	**22**	10	15.0
Miconic	**30**	**30**	**30**	**30**	**30.0**
Rovers	15	17	18	**20**	**20.0**
Satellite	12	22	**26**	21	21.4
Transport	9	9	**20**	13	15.0
∑ (240)	93	119	140	154	**183.0**

(c) Optimal classical planning.

Table 4: Planning coverage (↑). The best score for each domain is highlighted in colour and bold. Domains have 90 (resp. 30) problems each for satisficing (resp. optimal) planning.

E.2 Satisficing Numeric Planning



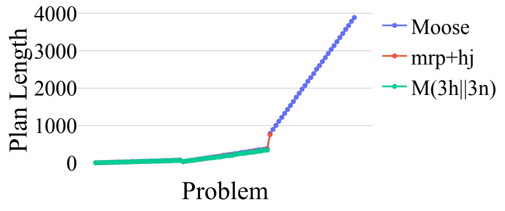
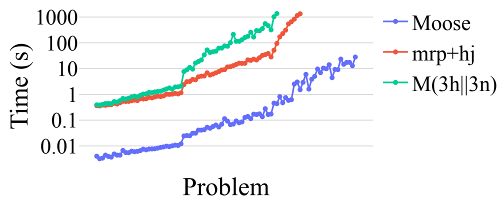


Figure 5: Average time (left) and plan length (right) of planners across solved problems for **Numeric Ferry**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.



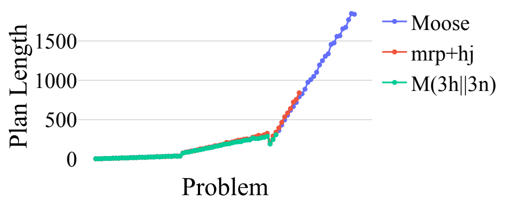


Figure 6: Average time (left) and plan length (right) of planners across solved problems for **Numeric Miconic**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

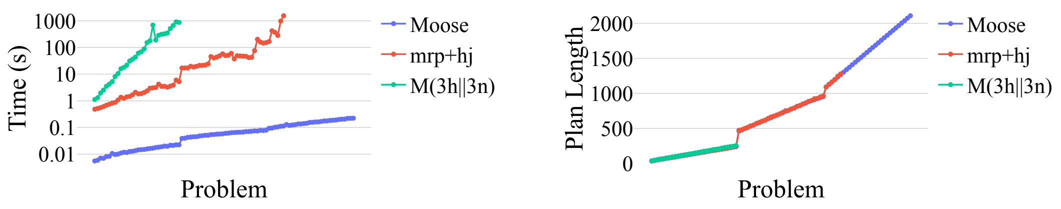


Figure 7: Average time (left) and plan length (right) of planners across solved problems for **Numeric Minecraft**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

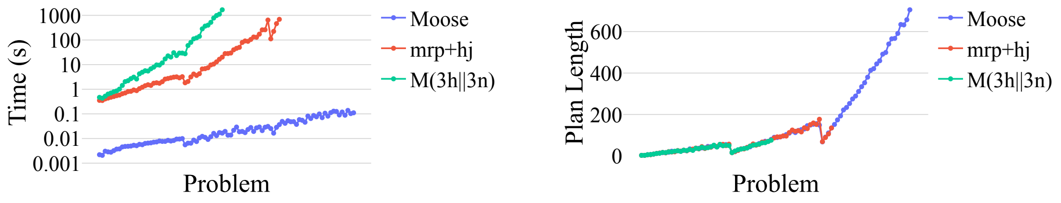


Figure 8: Average time (left) and plan length (right) of planners across solved problems for **Numeric Transport**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

E.3 Satisficing Classical Planning

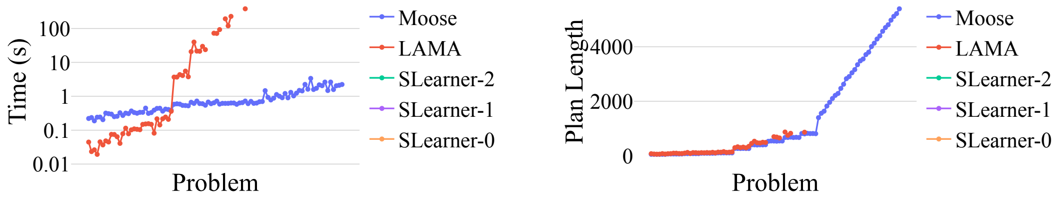


Figure 9: Average time (left) and plan length (right) of planners across solved problems for **Barman**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

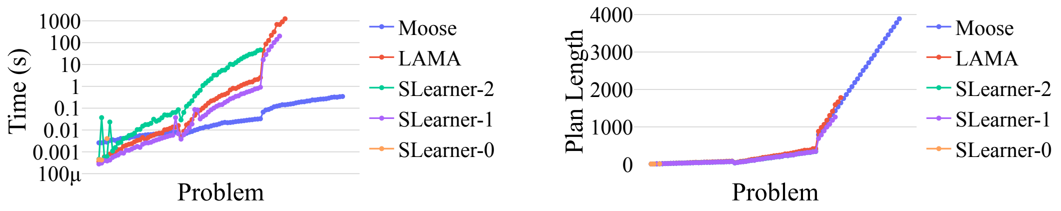


Figure 10: Average time (left) and plan length (right) of planners across solved problems for **Ferry**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

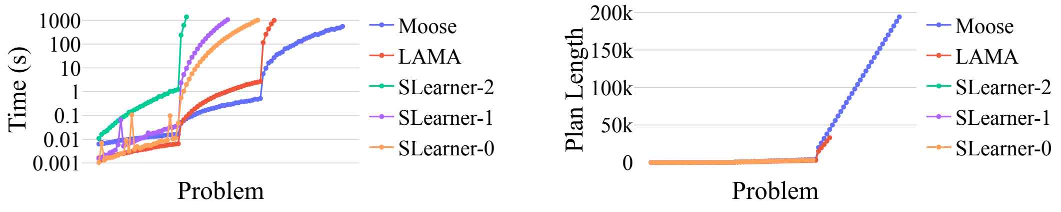


Figure 11: Average time (left) and plan length (right) of planners across solved problems for **Gripper**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

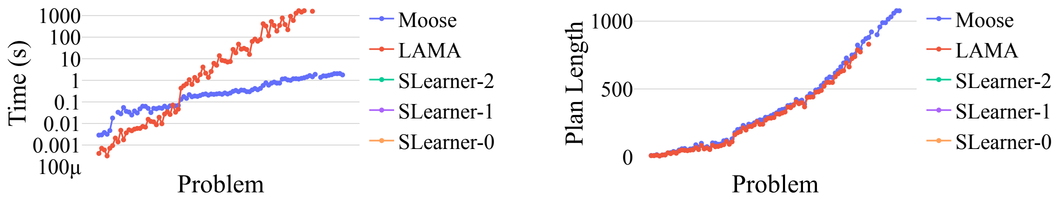


Figure 12: Average time (left) and plan length (right) of planners across solved problems for **Logistics**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

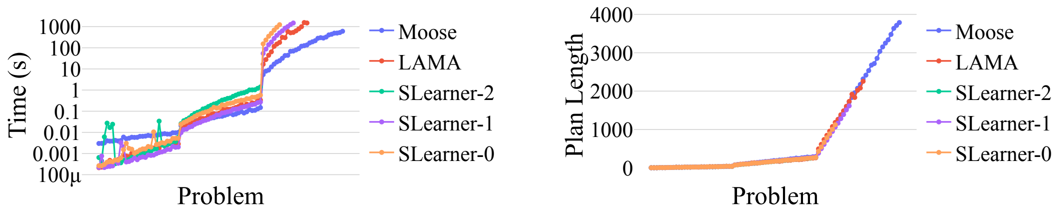


Figure 13: Average time (left) and plan length (right) of planners across solved problems for **Miconic**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

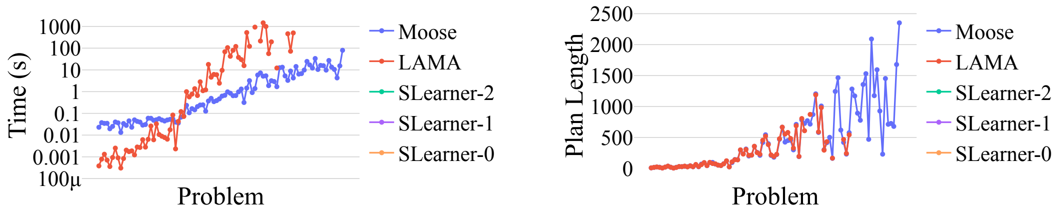


Figure 14: Average time (left) and plan length (right) of planners across solved problems for **Rovers**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

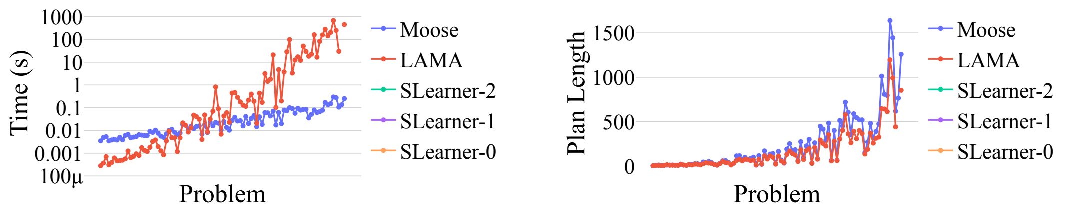


Figure 15: Average time (left) and plan length (right) of planners across solved problems for **Satellite**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

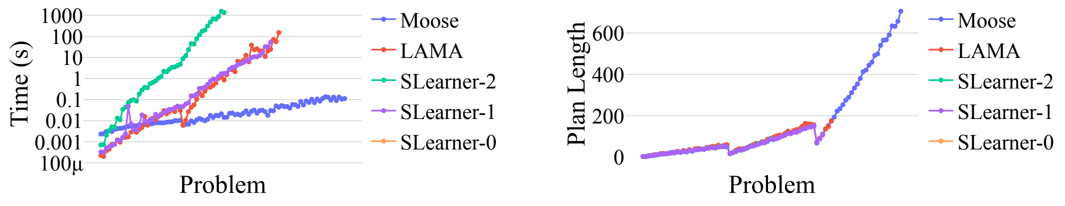


Figure 16: Average time (left) and plan length (right) of planners across solved problems for **Transport**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

E.4 Optimal Classical Planning

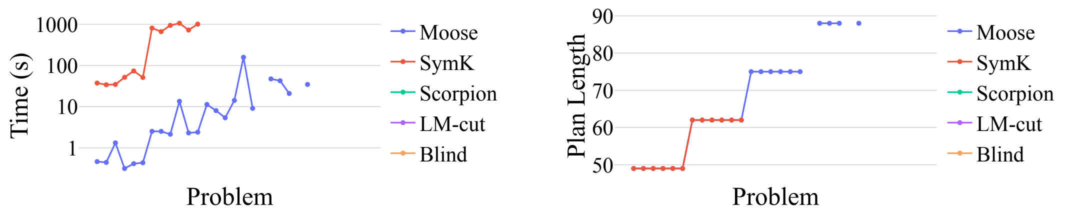


Figure 17: Average time (left) and plan length (right) of planners across solved problems for **Barman**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

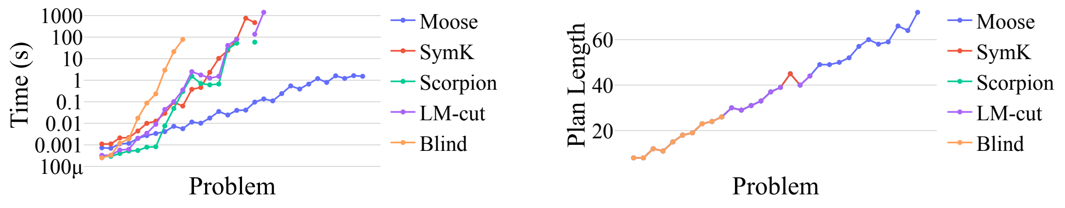
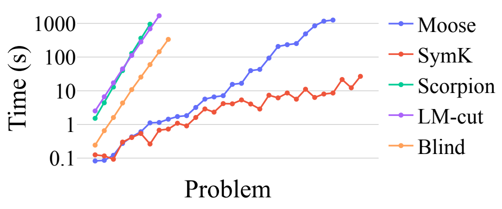


Figure 18: Average time (left) and plan length (right) of planners across solved problems for **Ferry**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.



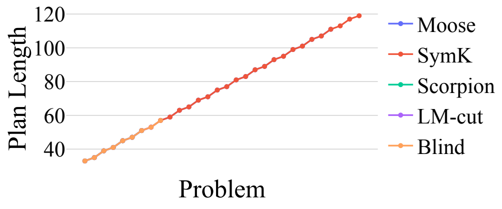
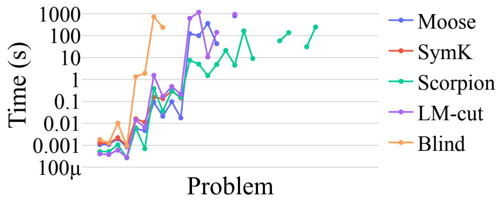


Figure 19: Average time (left) and plan length (right) of planners across solved problems for **Gripper**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.



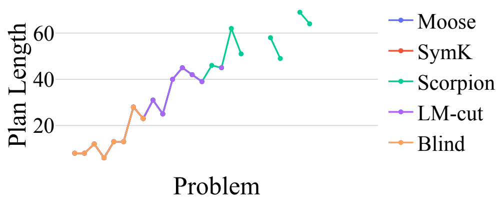
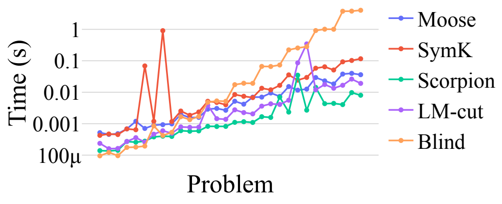


Figure 20: Average time (left) and plan length (right) of planners across solved problems for **Logistics**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.



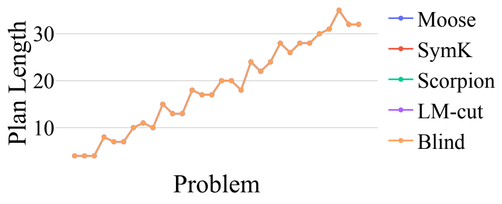
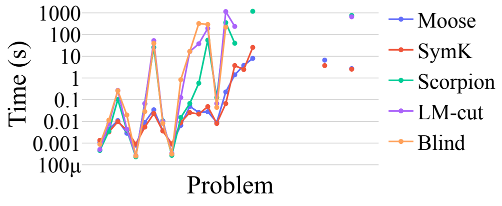


Figure 21: Average time (left) and plan length (right) of planners across solved problems for **Miconic**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.



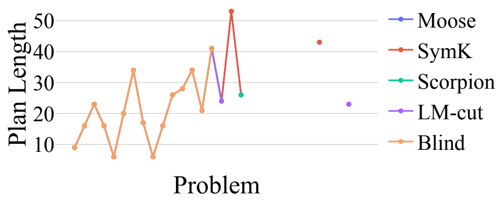


Figure 22: Average time (left) and plan length (right) of planners across solved problems for **Rovers**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

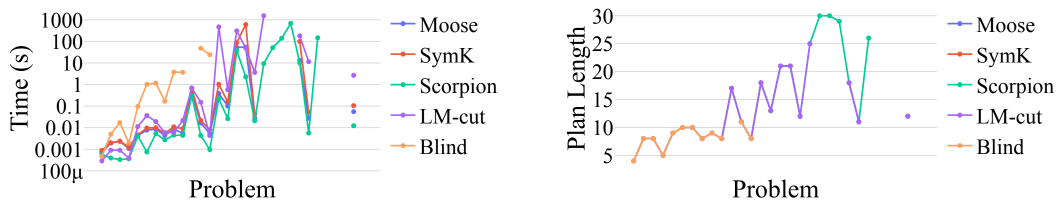


Figure 23: Average time (left) and plan length (right) of planners across solved problems for **Satellite**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

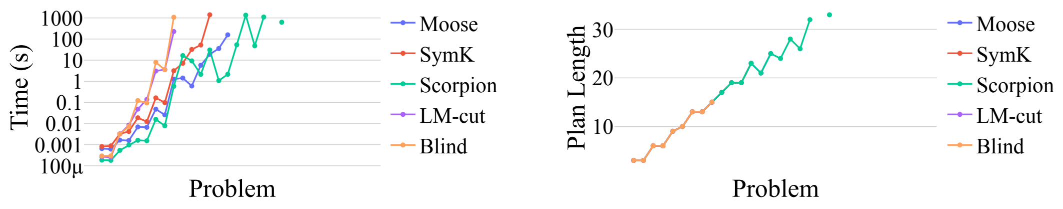


Figure 24: Average time (left) and plan length (right) of planners across solved problems for **Transport**. Planning problem difficulty increases across the x-axis. Lower y-axis values are better (↓). Note the y-axis log scale for runtime. Only points for which all seeds solve the problem are displayed.

E.5 Moose vs. Lama on Solution Quality

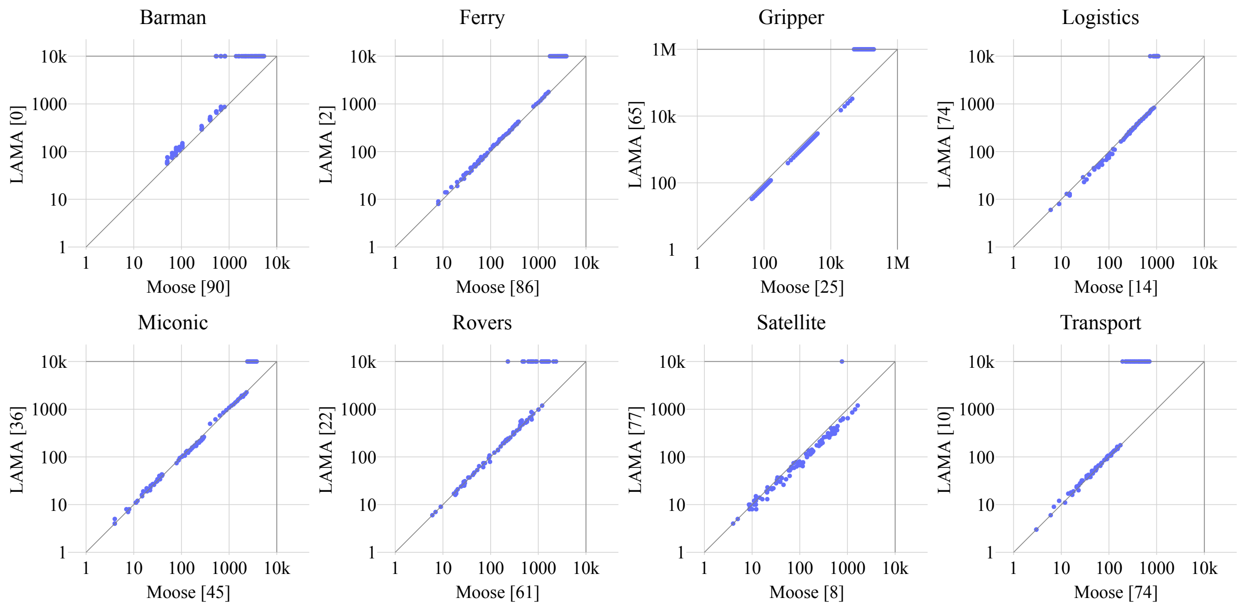


Figure 25: Plot comparisons of expanded nodes of Lama (y-axis) and Moose (x-axis) for different classical planning domains. A point (x, y) represents the metric of the models indicated on the x and y axis on the domain. The number in the brackets next to each model indicates how many planning problems 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).

E.6 Moose vs. Enhsp on Solution Quality

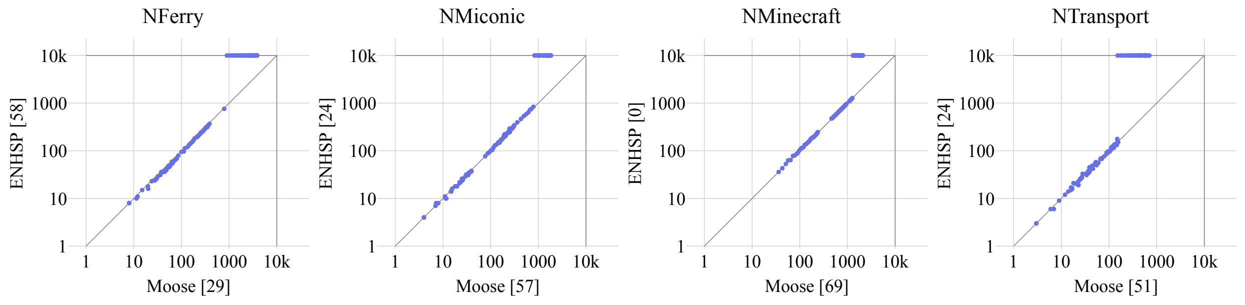


Figure 26: Plot comparisons of expanded nodes of the MRP+HJ configuration of Enhsp (y-axis) and Moose (x-axis) for different classical planning domains. A point (x, y) represents the metric of the models indicated on the x and y axis on the domain. The number in the brackets next to each model indicates 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. MRP+HJ).

F Conjunctive Goal Regression

In this section, we generalise the Moose algorithms by regressing over subsets of goals rather than singleton goals for synthesising more rules. We then conduct experiments and analyse the effect of regressing over subsets of goals on the 3 metrics of synthesis cost, instantiation cost, and plan quality.

F.1 Generalised Moose Algorithms

The updated synthesis algorithm is presented in Algorithm 4 which modifies Algorithm 1. The changes are highlighted in blue and lie in Lines 6 to 9 and Line 11. The subroutine $\text{extractRules}'$ is the same as extractRules in Algorithm 2 with the changes where

(1) we modify the type of the precedence ranking function from $\mathcal{R} \rightarrow \mathbb{N}$ to $\mathcal{R} \rightarrow \mathbb{N} \times \mathbb{N}$ and Line 4 is then changed to $\pi \leftarrow \pi \cup \{(r, (-|g|, |\vec{\alpha}| - i + 1))\}$, and

(2) rules where the state condition and goal condition have a non-empty intersection are pruned.

Furthermore, Line 4 in Algorithm 3 now iterates over $r \in \pi$ in ascending precedence values in lexicographical order. For example, we would have $(-2, 3)$ queried before $(-2, 4)$ which in turn is queried before $(-1, 2)$. The intuition here uses the triangle inequality where optimal plans for subgoals may be higher quality than achieving individual goals optimally in sequence. Note however that this is not always the case, as two goals may be conflicting when trying to achieve them both as opposed to achieving each goal individually. An example is trying to place 2 objects in a box but the box can only fit one object.

Algorithm 4: Moose Program Synthesis via k-Subset Goal Regression

```
Input: Training problems $\mathscr{P}_{\text{train}} = \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(n_t)}$, size of regressed goals $n_r$, and number of goal permutations $n_p \in \mathbb{N}$ (default: 3).
Output: Moose program $\pi$.
1  $\pi \leftarrow \emptyset$
2  for $i = 1, \ldots, n_t$ do
3      $n_g \leftarrow |\mathbf{P}^{(i)}[g]|$
4      for $j = 1, \ldots, \min(n_p, n_g!)$ do
5          $s \leftarrow \mathbf{P}^{(i)}[s_0]$ ; $\vec{g} \leftarrow \text{newPermutation}(\mathbf{P}^{(i)}[g])$
6          for $l = 1, \ldots, n_r$ do
7              $u \leftarrow$ largest integer of the form $1 + k \cdot l$ such that $u \leq n_g - k + 1$
8              for $k = 1, 1 + n_r, \ldots, u$ do
9                  $g' \leftarrow \{\vec{g}_k, \ldots, \vec{g}_{\min(k+n_r-1, n_g)}\}$
10                 $\vec{\alpha} \leftarrow \text{optimalPlan}(\mathbf{P}^{(i)}_{s,g'})$
11                 if $\vec{\alpha} = \bot$ then continue
12                 $\pi \leftarrow \pi \cup \text{extractRules}'(\vec{\alpha}, g')$
13                 $s \leftarrow \text{succ}(s, \vec{\alpha})$
14 return $\pi$
```

F.2 Experiments

We perform experiments on the classical planning benchmarks with $n_r \in \{1, 2, 3\}$, where $n_r = 1$ corresponds to the algorithms presented in the main paper, over 5 repeats. Ferry results for $n_r = 3$ are omitted as no training problem exhibited more than 2 goal atoms. We report results concerning the 3 metrics of synthesis cost, instantiation cost, and solution quality.

Synthesis Cost Synthesis costs are summarised in Table 5. We observe that increasing n_r results in higher synthesis costs and the size of the synthesised generalised plan (# learned rules).

	Time (s)			Memory (MB)			# Learned Rules		
Domain	1	2	3	1	2	3	1	2	3
Barman	202	349	468	184	160	164	167	224	222
Ferry	9	14	–	52	59	–	5	37	–
Gripper	10	10	12	64	65	52	4	10	19
Logistics	71	113	159	73	113	148	90	980	2187
Miconic	12	44	26	52	150	54	11	62	79
Rovers	534	884	1061	187	209	317	88	2450	8481
Satellite	514	740	951	82	142	194	13	252	1309
Transport	21	31	43	80	84	59	5	82	184

Table 5: Average runtime and memory usage (↓) for synthesis, and number of learned rules over different number of $n_r \in \{1, 2, 3\}$ in Algorithm 4. Ferry results for $n_r = 3$ are omitted as no training problem exhibited more than 2 goal atoms.

Instantiation Cost Instantiation costs are illustrated in Figure 27. We observe that increasing n_r results in higher instantiation costs, sometimes by more than an order of magnitude for each n_r increment. This is due to the increase in the number of synthesised rules and the associated costs of querying such rules.

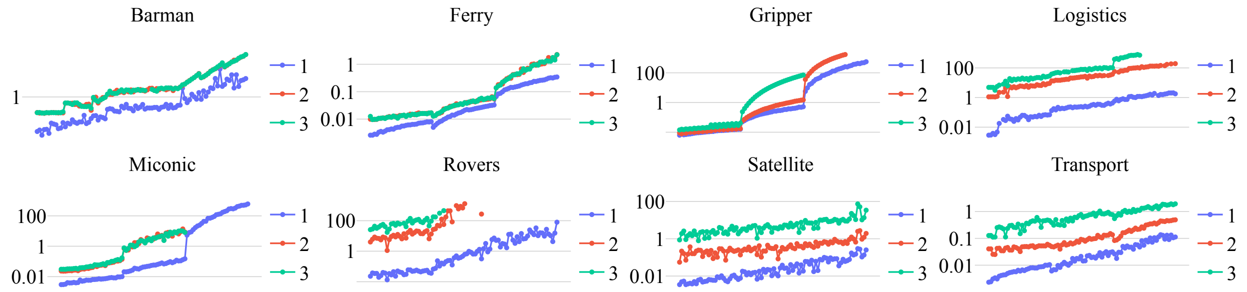


Figure 27: Average **time** in seconds in log scale (↓) of different n_r configurations across solved problems over different domains. Planning problem difficulty increases across the x-axis. Only points for which all seeds solve the problem are displayed.

Solution Quality Solution quality is aggregated in Table 6 by summing the plan length over problems that are solved by each n_r configuration. We observe that in all domains except Barman that increasing n_r above 1 improves solution quality. However, there is minimal change between $n_r = 2$ and $n_r = 3$.

Domain	1	2		3	
Barman	118778	121153	−2.0%	121167	−2.0%
Ferry	77760	70252	+9.7%	70247	+9.7%
Gripper	70800	53100	+25.0%	53190	+24.9%
Logistics	18224	18017	+1.1%	17967	+1.4%
Miconic	6293	6158	+2.1%	6172	+1.9%
Rovers	2027	1902	+6.2%	1888	+6.9%
Satellite	21923	18080	+17.5%	17862	+18.5%
Transport	14672	13586	+7.4%	13671	+6.8%

Table 6: Average **plan length** (↓) of different n_r configurations across solved problems over different domains. The percentages under the $n_r = 2$ and $n_r = 3$ columns indicate performance improvement relative to $n_r = 1$.