PG3: Policy-Guided Planning for Generalized Policy Generation

Ryan Yang[*] , **Tom Silver**[*] , **Aidan Curtis** , **Tomas Lozano-Perez** and **Leslie Kaelbling**
Massachusetts Institute of Technology
{ryanyang, tslvr, curtisa}@mit.edu, {tlp, lpk}@csail.mit.edu

Abstract

A longstanding objective in classical planning is to synthesize policies that generalize across multiple problems from the same domain. In this work, we study generalized policy search-based methods with a focus on the score function used to guide the search over policies. We demonstrate limitations of two score functions — policy evaluation and plan comparison — and propose a new approach that overcomes these limitations. The main idea behind our approach, Policy-Guided Planning for Generalized Policy Generalization (PG3), is that a candidate policy should be used to guide planning on training problems as a mechanism for evaluating that candidate. Theoretical results in a simplified setting give conditions under which PG3 is optimal or admissible. We then study a specific instantiation of policy search where planning problems are PDDL-based and policies are lifted decision lists. Empirical results in six domains confirm that PG3 learns generalized policies more efficiently and effectively than several baselines.

1 Introduction

How can we compile a transition model and a set of training tasks into a reactive policy? Can these policies generalize to large tasks that are intractable for modern planners? These questions are of fundamental interest in AI planning [Fikes *et al.*, 1972], with progress in *generalized planning* recently accelerating [Srivastava, 2011; Bonet and Geffner, 2015; Jiménez *et al.*, 2019; Rivlin *et al.*, 2020].

Generalized policy search (GPS) is a flexible paradigm for generalized planning [Levine and Humphreys, 2003; Segovia-Aguas *et al.*, 2021]. In this family of methods, a search is performed through a class of generalized (goal-conditioned) policies, with the search informed by a *score function* that maps candidate policies to scalar values. While much attention has been given to different policy representations, there has been relatively less work on the score function. The score function plays a critical role: if the scores are uninformative or misleading, the search will languish in less promising regions of policy space.

In this work, we propose Policy-Guided Planning for Generalized Policy Generation (PG3), which centers around a new score function for GPS. Given a candidate policy, PG3 solves the set of training tasks *using the candidate to guide planning*. When a plan is found, the agreement between the plan and the policy contributes to the score. Intuitively, if the policy is poor, the planner will ignore its guidance or fail to find a plan, and the score will be poor; if instead the policy is nearly able to solve problems, except for a few "gaps", the planner will rely heavily on its guidance, for a good score.

PG3 combines ideas from two score functions for GPS: *plan comparison*, which plans on the training tasks and records the agreement between the found plans and the candidate policy; and *policy evaluation*, which executes the policy on the training tasks and records the number of successes. While plan comparison provides a dense scoring over policy space, it works best when the solutions for a set of problems all conform to a simple policy. Absent this property, GPS can overfit to complicated and non-general solutions. Policy evaluation scores are extremely sparse within the policy space, effectively forcing an exhaustive search until reaching a region of the policy space with non-zero scores. However, the policies resulting from this approach are significantly more compact and general. PG3 combines the strengths of plan comparison and policy evaluation to overcome their individual limitations. See Figure 1 for an illustration.

In experiments, we study a simple but expressive class of *lifted decision list* generalized policies for PDDL domains [Mooney and Califf, 1995; Levine and Humphreys, 2003]. Since the policies are lifted, that is, invariant over object identities, they can generalize to problems involving new and more objects than seen during learning. We propose several *search operators* for lifted decision list policies, including a "bottom-up" operator that proposes policy changes based on prior planning experience.

This paper makes the following main contributions: (1) we propose PG3 as a new approach to GPS; (2) we provide conditions under which PG3 is optimal or admissible for policy search; (3) we propose a specific instantiation of GPS for PDDL domains; and (4) we report empirical results across six PDDL domains, demonstrating that PG3 efficiently learns policies that generalize to held-out test problems.

[*]Equal contribution.

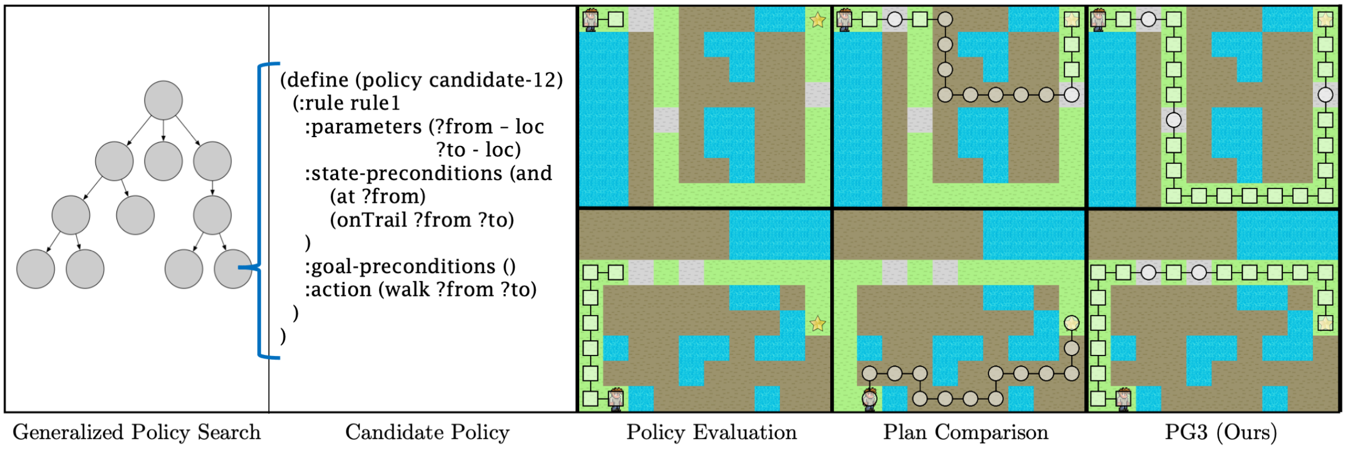

Figure 1: (Left) In this work, we study generalized policy search (GPS). Each node represents a candidate policy. (Left middle) A candidate policy for the Forest domain. In this domain, the agent *can* walk along the marked trail (green), but it can also walk in the dirt (brown) as long as it avoids water (blue). The candidate policy, which has the agent walk along the trail, would always succeed, except that the trails sometimes contain rocks (gray), which must be climbed; the policy thus fails when a rock is encountered. This policy is "close" to satisficing (see Definition 1 for a precise meaning), so we would like to give it a good score. Each of the next three columns illustrates a different score function evaluated using two training problems (top and bottom rows). Squares (circles) denote agreement (disagreement) between the candidate policy and plan. (Middle) Policy evaluation gives a trivial score to the candidate policy because it never reaches the goal. (Middle right) Plan comparison gives a poor score because the policy differs substantially from the paths found by a planner, which often leave the marked trail. (Right) We propose PG3, which evaluates the candidate by policy-guided planning, resulting in a good score.

2 Related Work

Policy search has been previously considered as an approach to generalized planning [Levine and Humphreys, 2003; Jiménez and Jonsson, 2015; Segovia-Aguas *et al.*, 2018]. Most prior work either uses *policy evaluation* as a score function, or relies on classical planning heuristics for the search through policy space. One exception is Segovia-Aguas et al. [2021], who extend policy evaluation with goal counting; we include this baseline in our experiments. Because of the limited search guidance and size of the policy space, all of these methods can typically learn only small policies, or require additional specifications of the problems that the policy is expected to solve [Illanes and McIlraith, 2019].

Another strategy for generalized planning is to construct a policy, often represented as a finite state machine, from example plans [Levesque, 2005; Srivastava *et al.*, 2011; Winner, 2008]. Such approaches are successful if the plans all conform to a single compact policy. This assumption is violated in many interesting problems including ones that we investigate here (c.f. *plan comparison*). Other approaches have been proposed to learn *abstractions* from example plans that lend themselves to compact policies [Bonet and Geffner, 2018]. While these approaches expand the space of problems that can be solved, they are still unable to learn policies that deviate dramatically from the example plans.

Recent work has proposed using deep learning to learn a generalized policy [Groshev *et al.*, 2018] or a heuristic function [Rivlin *et al.*, 2020; Karia and Srivastava, 2020]. Inspired by these efforts, we include a graph neural network baseline in experiments. Beyond classical planning in factored and relational domains, learning a goal-conditioned policy is of interest in reinforcement learning and continuous control [Schaul *et al.*, 2015; Pong *et al.*, 2018]. Unlike in generalized planning, these settings typically do not assume a known, structured domain model, and often make very different assumptions about state and action representations. The extent to which our insights in this work can be applied in these settings is an interesting area for future investigation.

We propose policy-guided planning as a mechanism for policy *learning*. Previous work has also considered the question of how to use a *fixed* (often previously learned) policy to aid planning [Yoon *et al.*, 2008].

3 Preliminaries

We begin with a brief review of classical planning and then define the generalized problem setting.

3.1 Classical Planning

In classical planning, we are given a *domain* and a *problem*, both often expressed in PDDL [McDermott *et al.*, 1998]. We use the STRIPS subset with types and negative preconditions.

A domain is a tuple $\langle P, A \rangle$ where P is a set of *predicates* and A is a set of *actions*. A predicate $p \in P$ consists of a name and an arity. An *atom* is a predicate and a tuple of *terms*, which are either *objects* or *variables*. For example, `on` is a predicate with arity 2; `on(X, Y)` and `on(b1, b2)` are atoms, where `X`, `Y` are variables and `b1`, `b2` are objects. Terms may be typed. A *literal* is an atom or its negation. An action $a \in A$ consists of a name and a tuple $\langle \textsc{Par}(a), \textsc{Pre}(a), \textsc{Add}(a), \textsc{Del}(a) \rangle$, which are the parameters, preconditions, add effects, and delete effects of the action respectively. The parameters are a tuple of variables. The preconditions are a set of literals, and the add and delete effects are sets of atoms over the parameters. A *ground action* is an action with objects substituted for parameters. For example, if `move` is an action with parameters `(?from, ?to)`, then `move(l5, l6)` is a ground action. In general, actions may be associated with variable costs; in this work, we focus on *satisficing* planning and assume all costs are unitary.

A domain is associated with a set of problems. A problem is a tuple $\langle O, I, G \rangle$ where O is a set of objects, I is an *initial state*, and G is a *goal*. States and goals are sets of atoms with predicates from P instantiated with objects from O. Given a state S, a ground action $\underline{a} = a(o_1, \ldots, o_k)$ with $a \in A$ and $o_i \in O$ is *applicable* if $\textsc{Pre}(\underline{a}) \subseteq S$.[1] Executing an applicable action $\underline{a}$ in a state S results in a *successor state*, denoted $\underline{a}(S) = (S \setminus \textsc{Del}(\underline{a})) \cup \textsc{Add}(\underline{a})$. Given a problem and domain, the objective is to find a *plan* $(\underline{a}_1, \ldots, \underline{a}_m)$ that *solves* the problem, that is, $\underline{a}_m \circ \cdots \circ \underline{a}_1(I) \subseteq G$ and each $\underline{a}_i$ is applicable when executed.

3.2 Problem Setting: Generalized Planning

Classical planning is traditionally concerned with solving individual planning problems. In *generalized planning*, the objective instead is to find a unified solution to multiple problems from the same domain. Here we are specifically interested in learning a *generalized policy* π, which maps a state S and a goal G to a ground action $\underline{a}$, denoted $\pi(S, G) = \underline{a}$. This form of π is very general, but our intention is to learn a *reactive* policy, which produces an action with minimal computation, and does not, for example, plan internally. Given a problem $\langle O, I, G \rangle$, the policy is said to *solve* the problem if there exists a plan $(\underline{a}_1, \ldots, \underline{a}_m)$ that solves the problem, and such that $\pi(S_i, G) = \underline{a}_i$ for each state S_i in the sequence of successors. In practice, we evaluate a policy for a maximum number of steps to determine if it solves a problem.

Our aim is to learn a policy that generalizes to many problems from the same domain, including problems that were not available during learning. We therefore consider a problem setting with a *training* phase and a *test* phase. During training, we have access to the domain $\langle P, A \rangle$ and a set of *training problems* $\Psi = \{\langle O_1, I_1, G_1 \rangle, \ldots, \langle O_n, I_n, G_n \rangle\}$. The output of training is a single policy π. During the test phase, π is evaluated on a set of held-out *test problems*, often containing many more objects than those seen during training. The objective of generalized policy learning is to produce a policy π that solves as many test problems as possible. A policy that solves all problems in the test set is referred to as *satisficing*.

4 Policy-Guided Planning for Policy Search

In this work, we build on *generalized policy search* (GPS), where a search through a set of generalized policies is guided by a *score function*. Specifically, we perform a greedy best-first search (GBFS), exploring policies in the order determined by $\textsc{Score}$, which takes in a candidate policy π, the domain $\langle P, A \rangle$, and the training problems Ψ, and returns a scalar value, where lower is better (Algorithm 1).

Example 1 (Forest). Forest is a PDDL domain illustrated in Figure 1; see the caption for a description. There is a simple satisficing policy for this domain: walk along the trail,

Algorithm 1 Generalized Policy Search via GBFS

Input: domain $\langle P, A \rangle$ and training problems Ψ
Input: search operators Ω
initialize: trivial generalized policy π_0
initialize: empty priority queue q
Push π_0 onto q with priority $\textsc{Score}(\pi_0, P, A, \Psi)$
for $i = 1, 2, 3, ...$ **do**
  Pop π from q
  for search operator $\omega \in \Omega$ **do**
    Push π' to q with $\textsc{Score}(\pi', P, A, \Psi)$
  end for
end for
return Best seen policy π^*

and climb over any rocks that are encountered. This policy is similar in spirit to the classic wall-following policy for mazes. The good-but-imperfect candidate policy illustrated in Figure 1 walks along the trail, but does not climb. When this candidate is considered during GPS, it will spawn many successor policies: one will add `(isRock ?to)` as a precondition to `rule1`; another will create a rule with the `climb` action; among others. Each successor will then be scored.

The score function has a profound impact on the efficacy of GPS. One possible score function is **policy evaluation**: the candidate policy is executed in each training problem, and the score is inversely proportional to the number of problems solved. A major limitation of this score function is that its outputs are trivial for all policies that do not completely solve any training problems, such as the policy described above.

Another possible score function is **plan comparison**: a planner is used to generate plans for the training problems, and the candidate policy is scored according to the agreement between the plans and the candidate policy (Algorithm 4). When there are multiple ways to solve a problem, this score function can sharply mislead GPS. For example, plan comparison gives a poor score to the follow-and-climb policy in Example 1, even though the policy is satisficing! This issue is not limited to the Forest domain; similar issues arise whenever goal atoms can be achieved in different orders, or when the same state can be reached through two different paths from the initial state. This phenomenon also arises in "bottom-up" generalized planning approaches (Section 2).

Our main contribution is **Policy-Guided Planning for Generalized Policy Generation (PG3)**. Given a candidate policy, PG3 runs *policy-guided planning* on each training problem (Algorithm 3). Our implementation of policy-guided planning is a small modification to A* search: for each search node that is popped from the queue, in addition to expanding the standard single-step successors, we roll out the policy for several time steps (maximum 50 in experiments[2], or until the policy is not applicable), creating search nodes for each state encountered in the process. The nodes created by policy execution are given cost 0, in contrast to the costs of the single-step successors, which come from the domain. These costs encourage the search to prefer paths generated by the policy.

[1] That is, if all positive atoms in $\textsc{Pre}(\underline{a})$ are in S, and no negated atoms in $\textsc{Pre}(\underline{a})$ are in S. We use this shorthand throughout.

[2] We did not exhaustively sweep this hyperparameter, but found in preliminary experiments that 50 was better than 10.

Algorithm 2 Score Function for Policy Search

input: candidate policy π
input: domain $\langle P, A\rangle$ and training problems Ψ
parameter: max plan horizon ℓ

initialize: $scores \leftarrow []$
for $\langle O, I, G\rangle \in \Psi$ **do**
  Run PolicyGuidedPlan(π, O, I, G, P, A)
  if a plan p is found **then**
    $score \leftarrow$ PlanComparison(π, p, I)
    $scores.append(score)$
  else
    $scores.append(\ell)$
  end if
end for
return $max(scores)$

For each training problem, if policy-guided planning returns a plan, we run *single plan comparison* (Algorithm 4) to get a score for the training problem. For each state and action in the plan, the policy is evaluated at that state and compared to the action; if they are not equal, the score is increased by 1. If a plan was not found, the maximum possible plan length ℓ is added to the overall score. Finally, the per-plan scores are accumulated to arrive at an overall policy score. To accumulate, we use *max* for its theoretical guarantees (Section 4.1), but found *mean* to achieve similar empirical performance.

Intuitively, if a candidate policy is able to solve a problem except for a few "gaps", policy-guided planning will rely heavily on the policy's suggestions, and the policy will obtain a high score from PG3. If the candidate policy is poor, policy-guided planning will ignore its suggestions, resulting in a low score. For example, consider again the good-but-imperfect policy for the Forest domain from Example 1. Policy-guided planning will take the suggestions of this policy to follow the marked trail until a rock is encountered, at which point the policy becomes stuck. The planner will then rely on the single-step successors to climb over the rock, at which point it will again follow the policy, continuing along the trail until another rock is encountered or the goal is reached.

Limitation. PG3 requires planning during scoring, which can be computationally expensive. Nonetheless, in experiments, we will see that PG3 (implemented in Python) can quickly learn policies that generalize to large test problems.

4.1 Theoretical Results

We now turn to a simplified setting for theoretical analysis.

Example 2 (Tabular). Suppose we were to represent policies as tables. Let $\{(S, G) \mapsto \underline{a}\}$ denote a policy that assigns action $\underline{a}$ for state S and goal G. Consider a single search operator for GPS, which changes one entry of a policy's table in every possible way. For example, in a domain with two states S_1, S_2, one goal G, and two actions $\underline{a}_1, \underline{a}_2$, the GPS successors of the policy $\{(S_1, G) \mapsto \underline{a_1}, (S_2, G) \mapsto \underline{a_1}\}$ would be $\{(S_1, G) \mapsto \underline{a_2}, (S_2, G) \mapsto \underline{a_1}\}$ and $\{(S_1, G) \mapsto \underline{a_1}, (S_2, G) \mapsto \underline{a_2}\}$. Here we are not concerned with generalization; training and test problems are the same, Ψ.

Our main theoretical results concern the influence of the PG3 score function on GPS. Proofs are included in Appendix A. We begin with a definition.

Algorithm 3 Policy-Guided Planning

input: policy π
input: problem $\langle O, I, G\rangle$ and domain $\langle P, A\rangle$
parameter: max plan horizon ℓ
return A* with succ:

for ground actions $\underline{a}$ applicable in S **do**
  yield $\underline{a}(S)$ with cost from domain
end for
for $i = 1, 2, \ldots, k$ **do**
  if π is not applicable in S, **stop**.
  $\underline{a}_i \leftarrow \pi(S, G)$
  $S \leftarrow \underline{a}_i(S)$
  yield S with cost 0
end for

Definition 1 (GPS cost-to-go)**.** *The* GPS cost-to-go *from a policy π_0 is the minimum k s.t. there exists sequences of policies $\pi_0, \ldots, \pi_k$ and search operators $\omega_0, \ldots, \omega_{k-1}$ s.t. π_k is satisficing, and $\forall j, \pi_{j+1} \in \omega_j(\pi_j)$.*

Note that in the tabular setting, the GPS cost-to-go is equal to the minimum number of entries in the policy table that need to be changed to create a policy that solves all problems.

Assumption 1. *The heuristic used for A* search in policy-guided planning (Algorithm 3) is admissible; planning is complete; and all costs in the original problem are unitary.*[3]

Theorem 1. *Under Assumption 1, in the tabular setting (Example 2), if a policy π solves all but one of the problems in Ψ, then the PG3 score for π is equal to the GPS cost-to-go.*

As a corollary, GPS with PG3 will perform optimally under the conditions of Theorem 1, in terms of the number of nodes expanded by GBFS before reaching a satisficing policy.

Theorem 2. *Under Assumption 1, in the tabular setting (Example 2), PG3 is a lower bound on the GPS cost-to-go.*

These results do not hold for other choices of score functions, including policy evaluation or plan comparison. However, the results do not immediately extend beyond the tabular setting; in general, a single application of a search operator could change a policy's output on every state, converting a "completely incorrect" policy into a satisficing one. In practice, we expect GPS to lie between these two extremes, with search operators often changing a policy on more than one, but far fewer than all, states in a domain. Toward further understanding the practical case, we next consider a specific instantiation of GPS that will allow us to study PG3 at scale.

4.2 Generalized Policies as Lifted Decision Lists

We now describe a hypothesis class of *lifted decision list* generalized policies that are well-suited for PDDL domains [Mooney and Califf, 1995; Levine and Humphreys, 2003].

Definition 2 (Rule)**.** *A rule ρ for a domain $\langle P, A\rangle$ is a tuple:*

[3]Unitary costs are assumed for simplicity; the proofs can be extended to handle any positive (nonzero) costs.

Algorithm 4 Single Plan Comparison

input: policy π, plan p
input: initial state I

initialize: $score$ to 0 and S to I
for $\underline{a}$ in p **do**
  if $\pi(S) \neq \underline{a}$ **then**
    Add 1 to $score$
  end if
  $S \leftarrow \underline{a}(S)$
end for
return $score$

- $\text{PAR}(\rho)$*:* parameters*, a tuple of variables;*
- $\text{PRE}(\rho)$*:* preconditions*, a set of literals;*
- $\text{GOAL}(\rho)$*:* goal preconditions*, a set of literals;*
- $\text{ACT}(\rho)$*: the rule's* action*, from A;*

with all literals and actions instantiated over the parameters, and with all predicates in P.

As with actions, rules can be *grounded* by substituting objects for parameters, denoted $\underline{\rho} = \rho(o_1, \dots, o_k)$. Intuitively, a rule represents an existentially quantified conditional statement: if there exists some substitution for which the rule's preconditions hold, then the action should be executed. We formalize these notions with the following two definitions.

Definition 3 (Rule applicability)**.** *Given a state S and goal G over objects $\mathcal{O}$, a rule ρ is* applicable *if $\exists(o_1, \dots, o_k)$ s.t. $\text{PRE}(\underline{\rho}) \subseteq S$ and $\text{GOAL}(\underline{\rho}) \subseteq G$, where $\underline{\rho} = \rho(o_1, \dots, o_k)$ and with each $o_i \in \mathcal{O}$.*

Definition 4 (Rule execution)**.** *Given a state S and goal G where rule ρ is applicable, let $(o_1, \dots, o_k)$ be the first[4] tuple of objects s.t. the conditions of Definition 3 hold with $\underline{\rho} = \rho(o_1, \dots, o_k)$. Then the* execution *of ρ in (S, G) is $\underline{a} = \text{ACT}(\underline{\rho})$, denoted $\rho(S, G) = \underline{a}$.*

Rules are the building blocks for our main generalized policy representation, the lifted decision list.

Definition 5 (Lifted decision list policy)**.** *A* lifted decision list policy *π is an (ordered) list of rules $[\rho_1, \rho_2, \dots, \rho_\ell]$. Given a state S and goal G, π is* applicable *if $\exists i$ s.t. ρ_i is applicable. If π is applicable, the* execution *of π is the execution of the first applicable rule, denoted $\pi(S, G) = \underline{a}$.*

See Appendix E.2 for examples of lifted decision list policies represented using PDDL-like syntax. Compared to PDDL operators, in addition to the lack of effects and the addition of goal preconditions, it is important to emphasize that the rules are *ordered*, with later rules only used when previous ones are not applicable. Also note these policies are *partial*: they are only defined when they are applicable.

Representational Capacity. We selected lifted decision lists because they are able to compactly express a rich set of policies across several domains of interest. For example, repeated application of a lifted decision list policy can lead to looping behavior. Nonetheless, note the absence of numeric values, transitive closures, universal quantifiers, and more sophisticated control flow and memory [Jiménez and Jonsson, 2015; Bonet *et al.*, 2019]. PG3 is not specific to lifted decision lists and could be used with richer policy classes.

4.3 Generalized Policy Search Operators

Here we present the search operators that we use for GPS with lifted decision list policies (Algorithm 1). Recall each operator $\omega \in \Omega$ is a function from a policy π, a domain $\langle P, A \rangle$, and training problems Ψ, to a set of *successor policies* Π'.

Add Rule. This operator adds a new rule to the given policy. One successor policy is proposed for each possible new rule and each possible position in the decision list. Each new rule ρ corresponds to an action with preconditions from the domain, and no goal conditions: for $a \in A$, $\rho = \langle \text{PAR}(a), \text{PRE}(a), \emptyset, a \rangle$. The branching factor for this operator is therefore $|A|(|\pi| + 1)$, where $|\pi|$ denotes the number of rules in the given decision list.

Delete Rule. This operator deletes a rule from the given policy. The branching factor for this operator is thus $|\pi|$.

Add Condition. For each rule ρ in the given policy, this operator adds a literal to $\text{PRE}(\rho)$ or $\text{GOAL}(\rho)$. The literal may be positive or negative and the predicate may be any in P. The terms can be any of the variables that are already in ρ. The branching factor for this operator is thus $O(4|\pi||P|k^m)$, where m is the maximum arity of predicates in P and k is the number of variables in ρ.

Delete Condition. For each rule ρ in the given policy, this operator deletes a literal from $\text{PRE}(\rho)$ or $\text{GOAL}(\rho)$. Literals in the action preconditions are never deleted. The branching factor is therefore at most $\sum_{\rho \in \pi} |\text{PRE}(\rho)| + |\text{GOAL}(\rho)|$.

Induce Rule from Plans. This final operator is the most involved; we describe it here at a high level and give details in Appendix B. Unlike the others, this operator uses plans generated on the training problems to propose policy modifications. In particular, the operator identifies a state-action pair that disagrees with the candidate policy, and amends the policy so that it agrees. The mechanism for amending the policy is based on an extension of the *triangle tables* approach of [Fikes *et al.*, 1972]. This operator proposes a single change to each candidate, so the branching factor is 1.

Given these operators, we perform GPS starting with an empty lifted decision list using the operator order: Induce Rule from Plans; Add Condition; Delete Condition; Delete Rule; Add Rule. In Appendix C, we describe two optimizations to improve GPS, irrespective of score function.

5 Experiments and Results

The following experiments evaluate the extent to which PG3 can learn policies that generalize to held-out test problems that feature many more objects than seen during training.

Experimental Setup. Our main experiments consider the fraction of test problems solved by learned policies. Note that here we are evaluating whether policies are capable of solving the problems on their own; we are not using the policies as planning guidance. Each policy is executed on each test problem until either the goal is reached (success); the policy is not applicable in the current state (failure); or a maximum

[4] First under a fixed, arbitrary ordering of object tuples, e.g., lexicographic, to avoid nondeterminism.

	PG3 (Ours)		Policy Eval		Plan Comp		Combo		Goal Count		GNN BC		Random	
Domains	Eval	Time	Eval	Time	Eval	Time	Eval	Time	Eval	Time	Eval	Time	Eval	Time
Delivery	**1.00**	1.5	0.00	–	0.10	5.8	0.20	2132.4	**1.00**	27.9	0.40	39.9	0.00	–
Forest	**1.00**	107.2	**1.00**	605.1	0.16	51.5	**1.00**	815.4	0.89	662.4	0.13	–	0.03	–
Gripper	**1.00**	79.5	0.00	–	0.20	0.8	0.00	–	0.00	–	0.06	–	0.00	–
Miconic	**1.00**	434.8	0.00	–	0.10	316.6	0.00	–	0.90	2415.3	0.12	–	0.13	–
Ferry	**1.00**	16.1	0.00	–	0.90	1.8	0.00	–	**1.00**	5705.2	0.11	11.8	0.00	–
Spanner	**1.00**	17.7	**1.00**	6.8	0.56	2.1	**1.00**	31.2	**1.00**	5.2	0.37	18.1	0.06	–

Table 1: **Policy learning results**. Eval columns report the fraction of test problems solved by the final learned policy. Time columns report the average wall-clock time (in seconds) required to learn a policy that solves $\geq 90\%$ of test problems, with a missing entry if such a policy was never found. All entries are means across 10 random seeds and 30 test problems per seed, with standard deviations shown in Table 2.

horizon is exceeded (failure) (Appendix D). All experimental results are over 10 random seeds, where training and test problem instances are randomly generated for each seed. All GPS methods are run for 2500 node expansions.

Domains. We use the following domains:

- **Forest**: See Figure 1 for description.
- **Delivery**: An agent must deliver packages to multiple locations while avoiding trap locations. The agent can pick up any number of packages from a home base, move between locations, and deliver a package to a location.
- **Miconic**: In this International Planning Competition (IPC) domain, an agent can move an elevator up or down and board or depart passengers. Each problem requires the agent to transport passengers to their desired floors.
- **Gripper**: In this IPC domain, an agent with grippers can pick, move, and drop balls. Each problem requires the agent to transport balls to their target locations.
- **Ferry**: In this IPC domain, an agent can sail, board, and debark cars. Each problem requires the agent to board cars, sail to their destination, and debark them.
- **Spanner**: In this IPC domain, an agent can move, pick up spanners, and tighten nuts using spanners. Each problem requires the agent to move along the corridor and pick up spanners needed to tighten nuts at the shed at the end of the corridor. The agent cannot move backwards.

See Appendix D for problem counts, sizes, and more details.

5.1 Approaches

We evaluate the following methods and baselines:

- **PG3 (Ours)**: Our main approach.
- **Policy evaluation**: GPS with policy evaluation.
- **Plan comparison**: GPS with plan comparison. Plans are collected on the training problems using A^* with h_{Add}.
- **Combo**: GPS with policy evaluation, but with ties broken using the plan comparison score function.
- **Goal count**: GPS with a score function that runs the candidate policy on each training problem and counts the number of goal atoms not yet achieved at the end of execution. Based on prior work [Segovia-Aguas *et al.*, 2021].
- **Graph neural network behavior cloning (GNN BC)**: Inspired by recent works that use graph neural networks (GNNs) in PDDL domains, this method learns a GNN policy via behavior cloning using plans from the training problems. The architecture and training method are taken from [Silver *et al.*, 2021]; see Appendix F.
- **Random**: a trivial baseline that selects among applicable actions uniformly at random on the test problems.

5.2 Results and Discussion

See Table 1 for our main empirical results. The results confirm that PG3 is able to efficiently guide policy search toward policies that generalize to larger held-out test problems. Qualitatively, the policies learned by PG3 are compact and intuitive. We highly encourage the reader to refer to Appendix E.2, which shows a learned policy for each domain.

Baseline performance is mixed, in most cases failing to match that of PG3. The strongest baseline is goal count, which confirms findings from previous work [Segovia-Aguas *et al.*, 2021]; however, even in domains with competitive final evaluation performance, the time required to learn a good policy can substantially exceed that for PG3. This is because goal count suffers from the same limitation as policy evaluation, but to a lesser degree: all policies will receive trivial scores until one is found that reaches at least one goal atom. Plan comparison has consistently good performance only in Ferry; in this domain alone, the plans generated by the planner are consistent with the compact policy that is ultimately learned. GNN BC is similarly constrained by the plans found by the planner. Combo improves only marginally on policy evaluation and plan comparison individually.

6 Conclusion

In this work, we proposed PG3 as a new approach for generalized planning. We demonstrated theoretically and empirically that PG3 outperforms alternative formulations of GPS, such as policy execution and plan comparison, and found that it is able to efficiently discover compact policies in PDDL domains. There are many interesting directions for future work, including applying PG3 in domains with stochasticity and partial observability, and integrating insights from other approaches to generalized planning.

Acknowledgements

We gratefully acknowledge support from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; and from MIT-IBM Watson Lab. Tom is supported by an NSF Graduate Research Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[Bonet and Geffner, 2015] Blai Bonet and Hector Geffner. Policies that generalize: Solving many planning problems with the same policy. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[Bonet and Geffner, 2018] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. *CoRR*, abs/1801.10055, 2018.

[Bonet *et al.*, 2019] Blai Bonet, Guillem Frances, and Hector Geffner. Learning features and abstract actions for computing generalized plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2703–2710, 2019.

[Fikes *et al.*, 1972] Richard E Fikes, Peter E Hart, and Nils J Nilsson. Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288, 1972.

[Groshev *et al.*, 2018] Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2018.

[Illanes and McIlraith, 2019] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7610–7618, Jul. 2019.

[Jiménez and Jonsson, 2015] Sergio Jiménez and Anders Jonsson. Computing plans with control flow and procedures using a classical planner. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS-15*, pages 62–69, 2015.

[Jiménez *et al.*, 2019] Sergio Jiménez, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *The Knowledge Engineering Review*, 34:e5, 2019.

[Karia and Srivastava, 2020] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. *arXiv preprint arXiv:2007.06702*, 2020.

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

[Levine and Humphreys, 2003] John Levine and David Humphreys. Learning action strategies for planning domains using genetic programming. In *Workshops on Applications of Evolutionary Computation*, pages 684–695. Springer, 2003.

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

[Mooney and Califf, 1995] Raymond J Mooney and Mary Elaine Califf. Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995.

[Pong *et al.*, 2018] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[Rivlin *et al.*, 2020] Or Rivlin, Tamir Hazan, and Erez Karpas. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, 2020.

[Schaul *et al.*, 2015] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[Segovia-Aguas *et al.*, 2018] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Computing hierarchical finite state controllers with classical planning. *Journal of Artificial Intelligence Research*, 62:755–797, 2018.

[Segovia-Aguas *et al.*, 2021] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Generalized planning as heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 569–577, 2021.

[Silver and Chitnis, 2020] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems. In *International Conference on Automated Planning and Scheduling (ICAPS) PRL Wor kshop*, 2020.

[Silver *et al.*, 2021] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *AAAI*, 2021.

[Srivastava *et al.*, 2011] Siddharth Srivastava, N. Immerman, S. Zilberstein, and Tianjiao Zhang. Directed search for generalized plans using classical planners. In *ICAPS*, 2011.

[Srivastava, 2011] Siddharth Srivastava. Foundations and applications of generalized planning. *AI Communications*, 24(4):349–351, 2011.

[Winner, 2008] Elly Zoe Winner. *Learning Domain-Specific Planners from Example Plans*. PhD thesis, Carnegie Mellon University, USA, 2008.

[Yoon *et al.*, 2008] Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9(Apr):683–718, 2008.