Efficient Black-Box Planning Using Macro-Actions with Focused Effects

Abstract
The difficulty of deterministic planning increases exponentially with search-tree depth. Black-box planning presents an even greater challenge, since planners must operate without an explicit model of the domain. Heuristics can make search more efficient, but goal-aware heuristics for black-box planning usually rely on goal counting, which is often quite uninformative. In this work, we show how to overcome this limitation by discovering macro-actions that make the goal-count heuristic more accurate. Our approach searches for macro-actions with focused effects (i.e. macros that modify only a small number of state variables), which align well with the assumptions made by the goal-count heuristic. Focused macros dramatically improve black-box planning efficiency across a wide range of planning domains, sometimes beating even state-of-the-art planners with access to a full domain model.

1. Introduction
In classical planning, an agent must select a sequence of deterministic, durationless actions to transition from a known initial state to a desired goal state. Planning assumes the agent has access to a model of the effects of its actions, which it uses to reason about potential plans. Usually this model takes the form of a PDDL description or finite-domain representation (Fox and Long 2003; Helmert 2009), which specifies the preconditions and effects of each action. However, in black-box planning (Lipovetzky, Ramirez, and Geffner 2015; Jinnai and Fukunaga 2017), the model is instead defined implicitly by a simulator that the agent can query to generate state transitions.

In general, planning is hard: determining whether a plan exists to reach the goal is \textit{PSPACE}-complete (Bylander 1994). Heuristic search eases this computational burden by guiding the search towards promising solutions. Of course, heuristic search is only useful with a good heuristic. In classical planning, much work has gone into the development of domain-independent methods that automatically construct heuristics to exploit as much problem structure as possible from the formal PDDL problem description (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert 2006; Helmert and Domshlak 2009; Helmert et al. 2014; Pomerening et al. 2015; Keyder, Hoffmann, and Haslum 2014; Domshlak, Hoffmann, and Katz 2015). However, black-box planners have no formal domain description to exploit, and are therefore limited to less-informed heuristics.

One simple, domain-independent heuristic that is compatible with simulator-based planners is the goal-count heuristic (Fikes and Nilsson 1971), which counts the number of state variables that differ between a given state and the goal. The two basic assumptions of the goal-count heuristic are: a factored state space (i.e. there are state variables to count), and a known goal state (i.e. there is a reason to fix variables). A third, more subtle assumption is that the problem can be decomposed into subproblems, where each state variable can be treated as an approximately independent subgoal. Unfortunately, this subgoal independence assumption is invalid for most planning problems of practical interest, and thus the goal-count heuristic is often misleading.

A second domain-independent strategy for improving planning efficiency is to use abstraction in the form of high-level macro-actions. When macro-actions are added to the set of low-level actions, they can reduce search tree depth at the expense of increasing the branching factor. In some cases, this has been shown to improve planning efficiency, particularly when the macro-actions cause the problem’s subgoals to become independent (Korf 1985). We further explore this idea in the context of black-box planning by constructing macros that are well aligned with the goal-count heuristic.

We begin by examining why goal-counting becomes uninformative for certain sets of actions. We show that both goal-count accuracy and planning efficiency are linked to how many state variables actions can modify at once. Our investigation suggests a compelling strategy for improving the usefulness of the goal-count heuristic: learning \textit{focused} macro-actions that modify as few variables as possible, so as to align with the assumptions made by the goal-count heuristic. This approach also seems well-aligned with human problem solving, for example, among expert Rubik’s cube solvers, where focused macros are essential for the most efficient planning strategies.

We describe a method for discovering focused macro-actions and test it on several classical planning benchmarks, restricting our attention to quickly finding feasible plans, rather than optimal ones, with the goal of minimizing the number of simulator queries. Our learned macro-actions en-
able reliable and efficient planning, making dramatically fewer calls to the simulator and improving solve rate on most domains. Our approach is designed to improve the goal-count heuristic, but it is compatible with more sophisticated black-box planning techniques as well—with similar improvement. In some cases, black-box planning with focused macros is even competitive with approaches that have access to much more detailed problem information.

2. Background

We consider the problem of black-box planning (Jinnai and Fukunaga 2017), where the planning agent does not have access to a declarative action description. Formally, we define a black-box planning domain using the following quantities:

- A set of states $S$, represented as vectors of state variables (or literals), with each variable $v_i$ taking on a finite number of values in some domain $D(v_i)$;
- An action function $A(s)$ that outputs the set of valid grounded actions for the given state $s \in S$;
- A deterministic simulator function $Sim(s, a)$, which the agent can query to determine the next state $s'$ after executing the action $a \in A(s)$ from state $s \in S$.

Each planning problem instance additionally contains:

- A start state, $s_0$, represented as a vector of state variables;
- A goal $G$, represented as a conjunction of literals.

The planner’s objective is to find a plan that connects state $s_0$ to a state $s_G$ that satisfies $G$ via a sequence of actions. In general, actions can have associated costs, and an optimal plan is one that minimizes the sum of its action costs. Here we are concerned with planning efficiency, so we focus on satisficing solutions—that is, finding a plan as quickly as possible, regardless of cost. We measure planning efficiency in terms of the number of simulator queries (equivalently, the number of generated states) before finding a plan.

The goal-count heuristic, $\#g$, is defined in terms of the problem-specific goal, $G$. For any goal conjunction $G$, $\#g(s)$ counts the number of literals in $G$ that are not true in state $s$, with $\#g(s) = 0$ if and only if $s$ satisfies $G$. Note that if $G$ is a single state, $\#g(s)$ simply counts the number of state variables in $s$ whose values differ from those in $G$.

A well-known downside of the goal-count heuristic is its dependence on the size of $G$. In the extreme case where $|G| = 1$, the goal-count heuristic only separates goal states from non-goal ones. Nevertheless, due to the relative lack of information in black-box planning, the goal-count is often the only goal-aware heuristic available. Other black-box planners may add additional components, such as state novelty (Francès et al. 2017), but these still rely on goal counting at their core.

Macro-Actions

A macro-action (or macro), is a deterministic sequence of actions, typically for the purpose of accomplishing some useful subgoal. To avoid confusion, we often refer to the original non-macro actions as “primitive” actions. Macros have parameters, preconditions, and effects, just like primitive actions, but in black-box planning, we again assume that the planner does not have access to such a declarative description. Instead, when planning with macro-actions, there are two alternatives. Either the action function $A(s)$ and the simulator function $Sim(s, m)$ are updated to compute macro-action validity and effects in a single step, or each primitive action in the macro must be simulated sequentially, with longer macros requiring more simulator queries.

3. Effect Size and Goal-Count Accuracy

The goal-count heuristic implicitly treats each state variable as an independent subgoal. There are two ways to satisfy this assumption exactly. The first is if each subgoal can be achieved in one step without modifying any other state variable. The second, more general way, explored by Korf (1985), is if each subgoal can be achieved in one step (possibly modifying other state variables) and the subgoals are serializable—i.e. there is an ordering of the state variables that retains previously-solved subgoals when solving new ones.

In general, an action can of course change many state variables, and the problem representation may not allow the subgoals to be serialized—both of which can cause the goal-count heuristic to be uninformative. However, for the goal-count heuristic to be useful, it does not need to be perfect; it simply needs to be rank correlated with the distance to the goal: higher true distances should correspond to higher heuristic values. When the heuristic is perfectly rank correlated, there is a monotonic relationship between heuristic and true cost, and best-first search will always expand nodes in order of their true distance from the goal.

We hypothesize that if each action modifies only a small number of state variables, the problem will better match the assumptions of the goal-count heuristic, and thus the heuristic and true goal distance will be more positively rank correlated. We informally say such actions have “focused” effects, and we formalize this idea with the following definitions:

Definition 1. The effect size of an action is the maximum number of state variables whose values change by executing the action, over all states where the action is applicable.

Definition 2. The effect size of a macro-action is the maximum number of state variables, measured at the end of macro-action execution, that are different from their starting values, over all states where the macro-action is applicable, even if additional variables were modified during execution.

If our hypothesis above is correct, we expect the goal-count heuristic to be more accurate for domains where actions have smaller effect size, and we further expect this to lead to an improvement in planning efficiency. In the following experiment, we see better rank correlation between heuristic and true distance for domains whose actions have low average effect size, and we see that this leads to an approximately exponential improvement in planning efficiency.

incorporate state information; however such skills are beyond the scope of this work.
We run two experiments using the goal-count heuristic and examine action effect size for Suitcase Lock. To study the relationship between effect size and planning efficiency, we measure the number of generated states needed to solve each instance of the planning problem. Focused actions improve planning efficiency, we introduce the Suitcase Lock domain. The Suitcase Lock is a planning problem whose solution requires entering a combination on a lock with \( N \) dials, each with \( M \) digits, and \( 2N \) actions, half which increment a deterministic subset of the dials (modulo \( M \)), and half which decrement the same dials (see Figure 1). For each problem instance, the start state, goal state, and action set are generated randomly, and a parameter \( k \) controls the mean effect size across all actions. This allows us to examine action effect size while holding other problem variables fixed. For implementation details, see Appendix A.

Focused Actions Improve the Goal-Count Heuristic

We first investigate the accuracy of the goal-count heuristic for two small Suitcase Lock problems. For each possible effect size, we compute the true distance between all pairs of states, and compare the results with the goal-count, treating the second state of each pair as the goal. We compute the average heuristic value for each true distance, and then compute the Pearson correlation and Spearman rank correlation coefficients between heuristic and distance. The results are shown in Table 1, where we see that actions with more focused effects (i.e. lower \( k \)) lead to significantly higher correlation.

<table>
<thead>
<tr>
<th>Effect Size</th>
<th>( N=10, M=2 )</th>
<th>( N=5, M=4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>( \rho_P )</td>
<td>( \rho_S )</td>
</tr>
<tr>
<td>1</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.200</td>
<td>0.179</td>
</tr>
<tr>
<td>3</td>
<td>0.110</td>
<td>0.092</td>
</tr>
<tr>
<td>4</td>
<td>0.060</td>
<td>0.041</td>
</tr>
<tr>
<td>5</td>
<td>0.020</td>
<td>0.013</td>
</tr>
<tr>
<td>6</td>
<td>0.000</td>
<td>-0.007</td>
</tr>
<tr>
<td>7</td>
<td>0.000</td>
<td>0.001</td>
</tr>
<tr>
<td>8</td>
<td>0.000</td>
<td>-0.001</td>
</tr>
<tr>
<td>9</td>
<td>0.000</td>
<td>0.005</td>
</tr>
</tbody>
</table>

These results on the Suitcase Lock domain suggest that reducing effect size is a viable strategy for improving planning efficiency. To further investigate this idea, we propose a method for learning macro-actions with low effect size.

4. Learning Macros with Focused Effects

We search for macro-actions using best-first search with a simulation budget of \( B_M \) state transitions. We start the search at a randomly generated state, and explore the macro-action effect size—or infinity if the macro-action modifies zero variables—plus the number of primitive actions in the macro. We save the \( N_M \) macro-actions with the lowest effect size, and ignore duplicate macro-actions that have the same net effect. To encourage diversity in the macros, we can optionally repeat the search \( R_M \) times, each time generating a new random starting state in which none of the existing saved macro-actions are valid, or until we fail to find such a starting state. This ensures that even if macros have constraining preconditions, we will still find macros that apply in most situations. (See Algorithm 1 in Appendix B.)

5. Experiments

We evaluate our method by learning macro-actions in a variety of black-box planning domains and subsequently using them for planning. Code is included in supplementary materials. We use PDDLGym (Silver and Chitnis 2020) to automatically construct black-box simulators from classical PDDL planning problems. Additionally, we use two domain-specific simulators (for 15-puzzle and Rubik’s cube) that have a different state representation to show the generality of our approach. See Appendix C for implementation details.

We select the domains to give a representative picture of how the method performs on various types of planning problems. For PDDLGym compatibility reasons, we restrict the domains to those requiring only strips and typing. For the domain-specific simulators, we select 15-puzzle and Rubik’s cube in particular, because they present opposing challenges for our macro-learning approach. In 15-puzzle,
primitive actions have very focused effects (each modifies only the blank space and one numbered tile), but naively chosen macro-actions tend to have much larger effect sizes, and both primitive actions and macros have state-dependent preconditions. In Rubik’s cube, actions and macros have no preconditions, but primitive actions are highly non-focused (each modifies 20 of the simulator’s 48 state variables) and the state space is so large (∼4.3×10^19 unique states (Rokicki 2014)) that black-box planning is unable to solve the problem efficiently.

**Methodology**

For each planning domain, we generate 100 problem instances with unique random starting states and a fixed goal condition. The planner has access to the simulator function, the action availability function, a vector of state information, and the goal condition. We emphasize that although the PDDLGym domains are specified using PDDL, the planner never sees the PDDL during either macro search or planning.

We learn focused macro-actions as described in Sec. 4 and add them to the set of primitive actions, which ensures that the same set of states can still be reached. These macros are then used to update the simulator and action availability functions, allowing the learned macros to execute in a single step for improved computational efficiency. Note that updating the simulator in this way does not reduce search effort, only time. Even if the primitive actions in a macro were simulated one-by-one, the intermediate states are neither stored nor explored, and hence do not count towards the number of generated states.

The macros are learned once, for the first problem instance, and then reused on all remaining problem instances for that domain. In general, it can be challenging to incorporate macros into any planning algorithm, since one must weigh their search benefits against the increased branching factor. For simplicity, our experiments fixed the number of macros \( N_M \) (see Table 2), but in principle \( N_M \) could be chosen automatically based on which macros reduce the problem’s average effect size.

To solve each planning problem, we use greedy best-first search (GBFS) with the goal-count heuristic and compare performance with the additional learned macro-actions versus with primitive actions alone. We measure planning efficiency as the number of simulator queries that the planner makes before finding a plan. This choice of performance metric is the most natural fit for black-box planning, and it allows for fair comparisons of algorithms across different implementation languages and hardware configurations.

In Table 2, we show the average solve rate and number of generated states (i.e., simulator queries) for each domain. Except in the case of Depot, we see that planning with focused macros increases solve rate and improves planning efficiency up to an order of magnitude versus planning with primitive actions alone. While we only pay the macro-learning cost for the first problem instance, in Rubik’s cube, focused macros still perform better, even when we account for the additional simulation budget required to learn them.

**Comparisons with Other Planners**

In addition to greedy best-first search (GBFS) with the goal-count heuristic, we also evaluate our method in conjunction with Best-First Width Search, or BFWS (Lipovetzky and Geffner 2017), a family of search algorithms that augment their search heuristic with a novelty metric computed using Iterated Width (IW) search (Lipovetzky and Geffner 2012).

We specifically use the best-performing black-box planning version of BFWS: BFWS(\( R_G^* \)) (Francès et al. 2017). This version starts by running IW up to two times, with increasing precision, to generate a set \( R_G^* \) of goal-relevant atoms. During search, each state \( s \) is evaluated based on how many relevant atoms were satisfied at some point along the path to \( s \). This forms a relevance count \( #r(s) \), which is combined with the goal-count \( #g(s) \) to compute the nov-

<table>
<thead>
<tr>
<th>Domain</th>
<th>( N_M )</th>
<th>( B_M )</th>
<th>GBFS(A)</th>
<th>GBFS(A+M)</th>
<th>BFWS(A)</th>
<th>BFWS(A+M)</th>
<th>LAMA(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen</td>
<td>Sol</td>
<td>Gen</td>
<td>Sol</td>
<td>Gen</td>
<td>Sol</td>
<td>Gen</td>
</tr>
<tr>
<td>Depot</td>
<td>8</td>
<td>50K</td>
<td>58275.9</td>
<td>0.74</td>
<td>55132.4</td>
<td>0.60</td>
<td>75966.9</td>
</tr>
<tr>
<td>Doors</td>
<td>8</td>
<td>5K</td>
<td>3050.7</td>
<td>1.00</td>
<td>512.6</td>
<td>1.00</td>
<td>4660.9</td>
</tr>
<tr>
<td>Ferry</td>
<td>8</td>
<td>5K</td>
<td>1875.8</td>
<td>1.00</td>
<td>1151.4</td>
<td>1.00</td>
<td>1209.9</td>
</tr>
<tr>
<td>Gripper</td>
<td>8</td>
<td>5K</td>
<td>7314.8</td>
<td>1.00</td>
<td>6277.0</td>
<td>1.00</td>
<td>44945.9</td>
</tr>
<tr>
<td>Hanoi</td>
<td>8</td>
<td>100K</td>
<td>78433.6</td>
<td>0.78</td>
<td>63588.1</td>
<td>1.00</td>
<td>63455.2</td>
</tr>
<tr>
<td>Miconic</td>
<td>8</td>
<td>5K</td>
<td>7559.4</td>
<td>1.00</td>
<td>1907.1</td>
<td>1.00</td>
<td>10269.2</td>
</tr>
</tbody>
</table>

*Table 2: Black-box planning results for PDDLGym-based simulators (top), and domain-specific simulators (bottom). (A) - primitive actions only; (A+M) - primitive actions + focused macros; \( N_M \) - number of macros; \( B_M \) - macro-learning budget; Gen - generated states; Sol - solve rate; (bold) - best performance of each planner. The efficiency of both GBFS and BFWS(\( R_G^* \)) are improved by adding focused macros. Note that LAMA is an informed planner with access to much more information than black-box planners, and is only included for reference.*
Table 3: Planning results for 15-puzzle and Rubik’s cube comparing different action spaces. Random macros perform significantly worse than both primitive actions and focused macros. Trials with macros also include the primitive actions.

<table>
<thead>
<tr>
<th></th>
<th>Generated States</th>
<th>Remaining Errors (#g)</th>
<th>Solve Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>15-Puzzle</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitives only</td>
<td>30840.5</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Random macros</td>
<td>120764.8</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Focused macros</td>
<td><strong>3980.4</strong></td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Rubik’s Cube</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitives only</td>
<td>&gt;2M</td>
<td>11.8</td>
<td>0.0</td>
</tr>
<tr>
<td>Random macros</td>
<td>&gt;2M</td>
<td>16.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Focused macros</td>
<td><strong>171331.4</strong></td>
<td><strong>0.0</strong></td>
<td><strong>1.0</strong></td>
</tr>
<tr>
<td>Expert macros</td>
<td>30229.1</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

We ran BFWS($R^*_G$) on each domain and measured its planning efficiency (see Table 2). We followed Lipovetzky and Geffner (2017) and limited the width precision to $w \in \{1, >1\}$ on Depot and Rubik’s cube to save computational resources.

Again we find that focused macros substantially improve planning efficiency, likely because the heuristic still uses goal counting at its core. Surprisingly, we found that BFWS did not perform significantly better than the primitive-action GBFS baseline. In fact, comparing against GBFS, we observe that focused macros alone are more beneficial for planning than switching to a more sophisticated novelty-based heuristic.

As a point of reference, we also compared against LAMA (Richter and Westphal 2010) which has full access to a declarative representation of the problem—information far beyond what is available to black-box planners. We ran the first iteration of LAMA on the same problems we used with PDDLGym, as well as a SAS$^+$ representation of the Rubik’s cube, adapted from Büchner (2018). On a different PDDL version of Rubik’s cube, LAMA failed to complete the translation step before running out of memory (16GB). We find our method is competitive with LAMA, across the majority of domains, despite the fact that LAMA has access to more information. On the 100 hardest Rubik’s cube problems from Büchner (2018), which neither primitive-action baseline can solve, we find that LAMA generates 9.1 million states on average, whereas our approach generates only 171 thousand.

**Comparison with Random Macros**

One might wonder whether the improvements in planning efficiency are due to the macros’ focused effects, or simply the fact that we are using macros at all. To isolate the source of the improvement, we conducted a second experiment using 15-puzzle and Rubik’s cube. Here we compared the focused macro-actions against an equal number of “random” macro-actions of the same length, which were generated (for each random seed) by selecting actions uniformly at random from the valid actions at each state.

We present the results in Table 3, where we observe that random actions perform significantly worse than both the primitive actions and the focused macros. We inspected the random macros for both domains, and found that they consistently had larger effect sizes than the learned, focused macros. Figure 2 shows a visualization of Rubik’s cube macro effect size versus macro length. We suspect the higher planning cost with random macros is partly due to their increased effect size.

**Comparison with Expert Macros in Rubik’s Cube**

Expert human “speedcubers” use macro-actions to help them manage the Rubik’s cube’s highly non-focused actions. In speedcubing, the goal is to solve the cube as quickly as possible, without necessarily finding an optimal plan. Most speedcubers learn a collection of macro-actions (called “algorithms” in Rubik’s cube parlance) and then employ a strategy for sequencing those macro-actions to solve the cube. Expert macro-actions tend to affect only a small number of state variables, and proper sequencing enables speedcubers to preserve previously-solved parts of the cube while solving the remainder. Common solution methods typically involve multiple levels of hierarchical subgoals and produce plans approximately twice as long as optimal.

As a benchmark, we consider a simplification of the most common expert strategy, where macros are composed of just primitive actions. We select a set of six expert macro-actions to perform various complementary types of permutations. We visualize one of these macro-actions, which swaps three corner pieces, in Figure 4a. Since our simulator uses a fixed cube orientation, we consider all 96 possible variations of each macro (to account for orientation, mirror-flips, and inversions), resulting in 576 total macros—the same number used in 5

---

5Macro-action sequences included in Appendix D.
for the random macro and focused macro trials.

In Figure 2, we plot the effect size and length of each macro, labeled by macro type. We can see that the focused macros have significantly smaller effect size than primitive actions or random macros, and begin to approach the effect size of the expert macro-actions. We note that our learned macros are somewhat shorter on average than the expert macros, and we suspect that increasing the search budget would result in learning macros with even smaller effects.

In Table 3 and Figure 3, we compare planning with the expert macros against the other macro types and see that while planning with the expert macros is the most efficient, the learned, focused macros are not far behind. By contrast, the random macros and primitive actions never solved the problem within the simulation budget. We also found that the average solution length for focused and expert macro-actions was about an order of magnitude longer than typical human speedsolve solutions (378 and 319 primitive actions, respectively, vs. ~55 (Speedsolving Wiki 2019)), which suggests that there are additional insights to be mined from human strategy beyond just learning focused macro-actions.

Interpretability of Focused Macros

We examined the learned focused macros for several domains and found that in addition to having low effect size, they were also frequently easy to interpret. In 15-Puzzle, one type of macro swapped the blank space with a central tile; another type exchanged three tiles without moving the blank space. In Rubik’s cube, one macro (Figure 4b) swapped three edge-corner pairs while keeping them connected. In Tower of Hanoi, macros moved stacks of disks at a time from one peg to another. We remark that this is quite similar to the interpretability of the human expert macros in Rubik’s cube.

Generalizing to Novel Goal States

Since our macro-generation step is goal-independent, we can reuse previously learned macros to solve problems with novel goal states. To demonstrate this, we generate 100 random goal states for 15-puzzle and Rubik’s cube and then solve the puzzles again. In both domains, we find that planning time and solve rate remain effectively unchanged for novel goal states (see Table 4 in Appendix F).

6. Related Work

The concept of building macro-actions to improve planning efficiency is not new. Dawson and Siklossy (1977) considered two-action macros and analyzed domain structure to remove macros that were invalid or that had no effect. Korf’s [1985] Macro Problem Solver investigated how to learn macros in problems with decomposable operators and serializable subgoals. Macro-FF (Botea et al. 2005) and MUM (Chrpa, Vallati, and McCluskey 2014) learned macros from training problem instances and later used them to improve the planning efficiency of testing instances. The macros we discover in Sections 4 and 5 can similarly be reused across problem instances; however, our macro discovery procedure requires neither goal information nor training instances. Our method is perhaps most similar to MARVIN (Coles and Smith 2007), which used macros to escape plateaus during heuristic search, and CAP (Asai and Fukunaga 2015), which decomposed planning problems into subgoals and then found macros to achieve those subgoals. In all of the prior approaches, the learned macros were found to be beneficial for planning, but they also required an explicit model of the domain. Our method is more general than these methods, as it is designed to handle the unique challenges of black-box planning without an explicit model.

Lipovetzky and Geffner (2012) introduced Iterated Width (IW) search, a “blind” planner compatible with black-box simulators, and Lipovetzky, Ramirez, and Geffner (2015) subsequently applied it to planning in Atari video game simulators without known goal states. This work led to the goal-informed Best-First Width Search (BFWS) (Lipovetzky and Geffner 2017; Francès et al. 2017), which we include in our experimental evaluation. Jinnai and Fukunaga (2017) formalized black-box planning and described a method for pruning primitive actions and short macros to avoid generating duplicate states; however their approach did not incorporate goal information.

Recent work by Agostinelli et al. (2019) investigated how to train black-box planning heuristics with neural networks and dynamic programming by strategically resetting the simulator to states near the goal state. Their approach learned

**6. Related Work**

The concept of building macro-actions to improve planning efficiency is not new. Dawson and Siklossy (1977) considered two-action macros and analyzed domain structure to remove macros that were invalid or that had no effect. Korf’s [1985] Macro Problem Solver investigated how to learn macros in problems with decomposable operators and serializable subgoals. Macro-FF (Botea et al. 2005) and MUM (Chrpa, Vallati, and McCluskey 2014) learned macros from training problem instances and later used them to improve the planning efficiency of testing instances. The macros we discover in Sections 4 and 5 can similarly be reused across problem instances; however, our macro discovery procedure requires neither goal information nor training instances. Our method is perhaps most similar to MARVIN (Coles and Smith 2007), which used macros to escape plateaus during heuristic search, and CAP (Asai and Fukunaga 2015), which decomposed planning problems into subgoals and then found macros to achieve those subgoals. In all of the prior approaches, the learned macros were found to be beneficial for planning, but they also required an explicit model of the domain. Our method is more general than these methods, as it is designed to handle the unique challenges of black-box planning without an explicit model.

Lipovetzky and Geffner (2012) introduced Iterated Width (IW) search, a “blind” planner compatible with black-box simulators, and Lipovetzky, Ramirez, and Geffner (2015) subsequently applied it to planning in Atari video game simulators without known goal states. This work led to the goal-informed Best-First Width Search (BFWS) (Lipovetzky and Geffner 2017; Francès et al. 2017), which we include in our experimental evaluation. Jinnai and Fukunaga (2017) formalized black-box planning and described a method for pruning primitive actions and short macros to avoid generating duplicate states; however their approach did not incorporate goal information.

Recent work by Agostinelli et al. (2019) investigated how to train black-box planning heuristics with neural networks and dynamic programming by strategically resetting the simulator to states near the goal state. Their approach learned
heuristics for several domains, including 15-puzzle and Rubik’s cube, that supported fast, near-optimal planning. However, training their neural network requires more than 1000 times the simulation budget of our approach, and results in a heuristic that is only informative for a single goal state, whereas ours works for arbitrary goal states.

7. Discussion and Conclusion

We have described a method of learning focused macro-actions that enables reliable and efficient black-box planning across a variety of classical planning domains. While our approach is designed to match the assumptions of the goal-count heuristic, we find that it also improves the performance of more sophisticated black-box planners. Moreover, our method is even competitive with a state-of-the-art LAMA planner, despite the latter having access to a declarative description of the problem.

We are encouraged to see that many of the learned macro-actions had intuitive, interpretable meaning in the task domain. This suggests that our method may be useful for improving explainability in addition to planning efficiency.

This work employed a two-level hierarchy where macro-actions are composed of primitive actions. One extension to bring this method more in line with human-expert techniques would be incorporating several levels of action hierarchy (i.e., macros composed of other macros), or macros that permit side-effects to certain unsolved variables, combined with macros to subsequently solve those remaining variables. We leave an exploration of these ideas for future work.

References


