Optimizing the Optimization of Planning Domains by Automatic Action Schema Splitting

Mojtaba Elahi, Jussi Rintanen

Aalto University
mojtaba.elahi@aalto.fi, jussi.rintanen@aalto.fi

Abstract

Most planners are based on grounding, that is, generating all instances of a parameterized action during a preprocessing phase. For some problems the number of ground actions is too high, causing a performance bottleneck. Building upon an existing approach, we present an enhanced method to split action schemas automatically during the grounding phase, to reduce the number of ground actions. First, we propose to exploit the structural knowledge of the problems to have a more informative dependency graph. Then, we suggest a better objective function to define and choose the best split. Finally, we present a more effective search to find it. We experimentally measure the impact of each of these improvements, and show that our approach significantly outperforms the state of the art.

Introduction

Planning is a branch of Artificial Intelligence (AI) that studies the problem of choosing a sequence of actions to reach a given goal from a given initial state. Actions are usually specified in a schematic form (by parameterizing it) in languages such as PDDL (Ghallab et al. 1998), and most planners ground actions in their preprocessing step by instantiating the parameters with all possible value combinations. The resulting non-parametric *ground* actions are easier to handle computationally: all parameter-related decisions are made simultaneously by selecting a single ground action, without the need for explicit parameterization consideration during the search process. However, this simplification comes with a price: when the number of parameter combinations becomes impractically high, grounding-based planners stall in their preprocessing step.

There are multiple approaches to address this issue. First, planning problems can be solved without grounding (Penberthy and Weld 1992; Ernst, Millstein, and Weld 1997; Robinson et al. 2008; Ridder and Fox 2014; Wichlacz, Höller, and Hoffmann 2022). In this way, they are directly handled in their schematic representation. The state-of-the-art approach in this category, Powerlifted, efficiently finds the successors of a state by using relational algebra query evaluation techniques (Corrêa et al. 2020), and also extends the delete relaxation (Bonet and Geffner 2001) and FF (Hoffmann and Nebel 2001) heuristic functions to the schematic representation (Corrêa et al. 2021, 2022).

Another approach to tackle this issue is to manually reformulate the problems so that the number of ground actions stays feasible. Essentially, schematic actions corresponding to a large number of ground actions are broken down into sequences of smaller actions, thus reducing the number of ground actions substantially. This requires an expert familiar with domain knowledge and AI Planning. The formalizations of the Pipesworld domain (Milidiú, dos Santos Liporace, and de Lucena 2003) in the International Planning Competition (IPC) 2004 and Genome Edit Distance (Haslum et al. 2011) are examples of this approach.

Finally, Areces et al. (2014) have proposed to perform the aforementioned reformulation fully automatically. However, they split each action separately and only based on structural information about the action itself. This limits the effectiveness of the approach and the generalization of its hyperparameters. Thus, its hyperparameters need to be tuned for each problem, otherwise it might have an adverse effect on planner performance.

In this work, we take the ideas of Areces et al. further to make them better applicable to planners targeting a broad range of problems, not only hard-to-ground ones. Our goal is to exploit information explicitly expressed in the problem definition or derivable from it, such as the domain size of each type, the problem's invariants, and estimates of the numbers of applicable ground actions of action schemas. Using this information, not only can we provide a more effective splitting approach, but also our hyperparameters can be more generalized so that our approach can be applied to a wide range of problems without further tuning or a noticeable deterioration of performance.

To achieve our goals, we propose improvements on three components of the Areces et al. approach. Specifically, we consider improving the dependency graph, the objective function, and the search algorithm. We encode some invariant information about the problem into the dependency graph. We also take an additional criterion into account in converting the graph into a sequence of smaller actions. We also provide a more meaningful objective function as a priority list of features; the highest priority one is the most important but coarsest feature, and the lowest priority feature is the least important but the finest feature. Finally, we propose a new search algorithm, which potentially covers more of the search space, adjusting to the properties of the problem at hand.

The paper is structured as follows. We begin by providing background information on the planning problem. Next, we present an overview of the approach of Areces et al. (Areces et al. 2014). We then describe our contributions in three areas. In the following section, we report on and discuss our experiments. Finally, we conclude the paper.

Preliminaries

Following Areces et al. (2014), we focus on the STRIPS fragment of PDDL. However, our approach can easily be extended to support actions with conditional effects and other more advanced features, but for the presentational simplicity, we ignore these extensions.

We define a planning problem instance by a quintuple of $\Sigma = \langle \mathcal{O}, \mathcal{P}, \mathcal{A}, s_i, g \rangle$, where $\mathcal{O}$ is the set of objects, $\mathcal{P}$ is the set of predicates, $\mathcal{A}$ is the set of action schemas, s_i describes the initial state, and g denotes the goal condition. A predicate symbol $P^n \in \mathcal{P}$ has arity n and denotes a relation among n objects. For example, the literal $P^n(o_1, \ldots, o_n)$ states that the relation P^n holds among $o_1, \ldots, o_n$[1]. An action schema $a[X] \in \mathcal{A}$ is a pair of $a[X] = \langle p_X, e_X \rangle$, where X is the list of parameters of the action, p_X denotes its conjunctive precondition formula, and e_X specifies the effects. Each p_X and e_X is represented as a set of literals, with their arguments either from $\mathcal{O}$ or X.

We represent a state s by specifying all relations that hold in it, as the set of positive literals and assume the relations corresponding to those literals are holding in the state, and the rest of the other possible relations do not hold. Thus, we can denote the initial state s_i of the problem by a set of positive literals. We say s_g is a goal state if g is true in s_g ($s_g \models g$).

We can apply a ground action $a[O]$ to a state s to transit to another state s'. Here, $O = \langle o_1, \ldots, o_n \rangle, o_i \in \mathcal{O}$, and the ground action $a[O] = \langle p_O, e_O \rangle$ is an instance of $a[X] = \langle p_X, e_X \rangle$, which is constructed by an instantiation function $\sigma : X \to O$ that substitutes each parameter in X with their corresponding element in O, for all literals in p_X and e_X. In other words, p_O and e_O are the ground versions of p_X and e_X under σ. We can apply the action $a[O]$ on state s only if $s \models p_O$; in this case, the next state s' will be constructed by eliminating all negative literals of e_O from s and adding the positive literals of e_O to it.

Existing Approach to Splitting

Most planners ground action schemas in $\mathcal{A}$ at their preprocessing step. However, grounding will easily become intractable as the number of parameters and their possible values increase, potentially leading to an exponential increase in the number of ground instances. More precisely, the upper-bound number of ground instances of an action schema $a[X]$ is $|\mathcal{O}|^{|X|}$, where $|\mathcal{O}|$ and $|X|$ denote the size of $\mathcal{O}$ and X, respectively; this shows the high sensitivity of grounding on the domain size (number of objects) and the parameter size of actions.

To alleviate this issue, Areces et al. (2014) proposed an algorithm to split actions into chains of smaller actions, which we call *micro-actions*. For example, the action schema $a[X]$ can be split into a chain $\alpha = \langle a_1[X_1], \ldots, a_n[X_n] \rangle$ of micro-actions, which consequently decreases the parameter size $|X|$ (the exponent term). The number of ground instances is reduced to $\sum_i |\mathcal{O}|^{|X_i|}$. However, this reduction in the number of ground instances comes at the price of increasing plan length. By recognizing this trade-off, the proposed approach provides a mechanism to specify the optimal point for the ratio of these two quantities.

This approach starts by decomposing an action schema into its atomic parts; the atomic parts of an action schema are the literals in its precondition and effects because they cannot be broken into any smaller entities. Additionally, to distinguish the literals in precondition and effects, they are annotated with the role they have. In the next step, dependencies among those atomic parts are determined. Since an action's precondition needs to be checked before applying the effect, we should prevent modifying a relation until the precondition corresponding to that relation is checked. Moreover, in a ground action, there might be effects that specify a relation simultaneously exists and does not exist in the next state (effects may contain both a literal and its negation). To resolve this ambiguity, the positive literal is picked; or in the terms of splitting, we first apply the negative effect and then the positive one, by placing the positive literal in the same or later micro-actions in the chain. We can define a directed graph to show these dependencies.

Definition 1 (Dependency graph)**.** *The dependency graph $G = \langle N, E \rangle$ for an action schema $a[X]$ is a directed graph with a node $a_i[X_i] \in N$ for each atomic action part and an edge $\langle a_i[X_i], a_j[X_j] \rangle \in E$, describing the atomic part $a_i[X_i]$ should not be placed before $a_j[X_j]$. We have $\langle a_i[X_i], a_j[X_j] \rangle$ if and only if these two atomic action parts share the same predicate symbol P and*

- *$a_j[X_j]$ denotes a precondition and $a_i[X_i]$ is an effect, or*
- *$a_j[X_j]$ is a negative effect and $a_i[X_i]$ is a positive effect.*

Areces et al. (2014) showed that this dependency graph is acyclic and that any of its topological orderings correspond to a valid chain of micro-actions, satisfying the dependency requirements mentioned earlier. Furthermore, they demonstrated that any of its acyclic quotient graphs (constructed by aggregating some nodes into a new node) also correspond to a valid chain. Forming this search space of all possible quotient graphs, they defined an objective function to describe the optimal graph. The objective function employs a parameter γ to control the trade-off between reducing the parameter size of micro-actions and increasing the plan length.

An Enhanced Splitting Approach

We improve three aspects of the current approach: the dependency graph, the objective function, and the search algorithm. We will discuss each in turn.

[1]For brevity, we omit the superscript ^{n} if there is no ambiguity.

Dependency Graph

Instantiating the parameters of an action schema can be viewed as a search problem: some instantiations produce applicable ground actions, while others lead to instances that are inapplicable in all reachable states. Our goal is to efficiently find the applicable instances.

For example, consider the $load$ action schema in the well-known planning domain of Logistics. With the parameters of $\langle ?object, ?truck, ?location\rangle$, this action denotes loading a package into some truck at some location.

$$load\left[\left\langle \begin{matrix} ?object, \\ ?truck, \\ ?location \end{matrix}\right\rangle\right] = \left\langle \begin{matrix} \left\{\begin{matrix} package(?object), \\ truck(?truck), \\ location(?location), \\ at(?truck, ?location), \\ at(?object, ?location) \end{matrix}\right\}, \\ \left\{\begin{matrix} \neg at(?object, ?location), \\ in(?object, ?truck) \end{matrix}\right\} \end{matrix}\right\rangle$$

Now, let us instantiate the parameters one by one, and compare two different orders. In the first order, we begin with $?object$, and then $?truck$ (and leave the $?location$ as the last parameter to be instantiated). Since there is no explicit relation between these two variables in the precondition, we can consider any packages and any trucks; thus, it is possible that the package is not in the same place as the truck, and we will not realize this unless we try to instantiate $?location$. In another order, we can prevent this dead-end instantiation by choosing $?truck$ and then $?location$. Therefore, according to the precondition, we are restricted to choosing only those packages that are in the same place as the truck, to instantiate $?object$.

While this example illustrates the importance of ordering parameters in instantiations, we need to figure out the reason behind these parameter dependencies. Paying closer attention to the precondition, we realize that the relations $at(?truck, ?location)$ and $at(?object, ?location)$ are the causes of these dependencies. More precisely, since the relation $at(?object, ?location)$ can be viewed as an *array* that maps unloaded objects to their current locations, by specifying the index of the array its value can be uniquely determined. Thus, we can say the value ($?location$) depends on the index ($?object$).

Definition 2 (Parameter dependency)**.** *For an action schema $a[X]$, we say the parameter $u \in X$* depends *on the set of parameters $Y \subset X, u \notin Y$ if at any reachable state, for any two applicable ground actions $a[O_1]$ and $a[O_2]$ with the instantiation functions σ_1 and σ_2 such that $\forall y \in Y : \sigma_1(y) = \sigma_2(y)$, then we have $\sigma_1(u) = \sigma_2(u)$.*

In other words, we say u depends on the parameters Y if the values of Y uniquely determine the value of u.

Helmert (2009) has captured this information as *monotonicity invariants* with *weight* one. In his invariant synthesis approach, monotonicity invariants are pairs $\mathcal{I} = \langle V, \Phi\rangle$, in which V is a set of variables called *parameters*, and Φ is a set of first-order atoms. The free variables in Φ (which are not parameters) are called *counted variables*. By instantiating the parameters V in Φ, we can create an instance of $\mathcal{I}$. The weight of an instance of $\mathcal{I}$ in a state s is the number of ground atoms of Φ (which are obtained by further grounding the counted variables in the instance of $\mathcal{I}$) that are true in s. In other words, an instance of $\mathcal{I}$ can be described by a set of partially ground atoms of Φ having only counted variables uninstantiated. By instantiating the counted variables with all possible values, we get the set of *covered facts* of this instance of $\mathcal{I}$. We say $\mathcal{I} = \langle V, \Phi\rangle$ is a monotonicity invariant with weight one if at most one of the covered facts of each of its instances is true at any reachable state.

Theorem 1. *If a monotonicity invariant $\mathcal{I} = \langle V, \Phi\rangle$ and an action $a[X] = \langle p_X, e_X\rangle$ share a predicate symbol P, such that $P(r_1, \ldots, r_n) \in \Phi$ and $P(s_1, \ldots, s_n) \in p_X$, then any parameter $u \in \{s_i \mid r_i \notin V, s_i \in X\}$ depends on the set of parameters $Y = \{s_i \mid r_i \in V, s_i \in X\}$.*

Proof. Suppose, for the sake of contradiction, that in a reachable state, we have two applicable ground actions $a[O_1]$ and $a[O_2]$ with instantiation functions σ_1 and σ_2 such that $\forall y \in Y : \sigma_1(y) = \sigma_2(y)$, and we have $\sigma_1(u) \neq \sigma_2(u)$.

Thus, in this state, there are two different relations of P (ground atoms of P) among two sets of objects that share the objects corresponding to parameters Y with different objects for the parameter u; this means at least two covered facts of an instance of the invariant $\mathcal{I}$ is true at this state (the weight of $\mathcal{I}$ is at least two), which is a contradiction. □

Therefore, by utilizing monotonicity invariants, we can determine the dependencies among the parameters of an action schema. One way to utilize this information is to encode it in the dependency graph. However, the nodes in the dependency graph are atomic action parts, but we have a set of relations among parameters. Thus, we need to extend these relations to the dependency graph's nodes.

Definition 3 (Precondition dependency)**.** *For an action schema $a[X] = \langle p_X, e_X\rangle$, we say $P_1(r_1, \ldots, r_n) \in p_X$* depends *on $P_2(s_1, \ldots, s_m) \in p_X$, if there exists an invariant $\mathcal{I} = \langle V, \Phi\rangle$ such that $P_2(t_1, \ldots, t_m) \in \Phi$ and $\exists i, j.(r_j = s_i \wedge t_i \notin V)$.*

In other words, we say precondition $P_1(r_1, \ldots, r_n)$ depends on precondition $P_2(s_1, \ldots, s_n)$, if they both share a common parameter s_i, and s_i is uniquely determined by $P_2(s_1, \ldots, s_n)$.

By adding the edges corresponding to precondition dependencies to the dependency graph, we can prevent dead ends in the search, like our $load$ action example. However, adding new edges might form a cycle in the graph, which is not desirable. Thus, we omit the new edges that make a cycle in the dependency graph.

All edges in the dependency graph are not desirable. Our graph might have superfluous edges that only increase the length of the micro-actions chain. For example, if an effect has the same predicate symbol with a precondition, then we add an edge from the precondition to the effect because those two might instantiate with the same objects, and therefore, the effect might have some side effect on the precondition. Although we need to prevent such side effects, we are overestimating these dependencies by detecting them based on predicate symbols only. Depending on the rest of the preconditions, it might be concluded that there is no possible way to instantiate the precondition and the effect with the same objects. More precisely, there might be at least one argument that will be instantiated differently for the precondition and the effect, for all applicable ground actions. For example, this could be so when the precondition contains an inequality constraint for the variables corresponding to that argument. In the organic synthesis domains (Matloob and Soutchanski 2016), these inequality constraints can eliminate many superfluous edges in the dependency graph.

A more advanced approach is to exploit the invariant knowledge of the problem to infer such inequalities. In some domains, such as Pipesworld (Milidiú, dos Santos Liporace, and de Lucena 2003), these inequality constraints are not explicit; they can be deduced by the preconditions and the invariant knowledge of the problem. Although monotonicity invariants (Helmert 2009) are effective in discovering parameter dependencies, they only cover a subset of invariants. Therefore, we can utilize Rintanen's (2017) approach that extracts a wider class of invariants.

In the invariant synthesis approach proposed by Rintanen (2017), we can extract schematic invariants by analyzing a limited grounded instance of the problem. More precisely, instead of considering all possible objects, Rintanen showed that invariants can be correctly extracted from actions that have been instantiated with only a small subset of objects. This can significantly mitigate the grounding issue in hard-to-ground problems. Additionally, the analysis of the limited grounded instance can directly answer our question; for a precondition and an effect having the same predicate symbol, we only need to check if there is a ground action in the limited grounded instance of the problem that instantiates those two predicates with exactly the same objects. If there is no such applicable ground action, then we can remove the corresponding edge in the dependency graph.

Areces et al. (2014) showed that any topological ordering of the dependency graph or its acyclic quotient graph forms a valid chain of micro-actions. A directed acyclic graph might have several topological orderings. To choose among them, Areces et al. proposed to select the ordering with more preconditions in earlier micro-actions. This way, inapplicable actions might be detected sooner in a forward search. In addition to this, we also propose to prefer an ordering that introduces arguments with positive preconditions rather than negative preconditions or effects.

Definition 4 (Introducing argument)**.** *For an action split $\alpha = \langle a_1[X_1], \dots, a_n[X_n]\rangle$, we say the argument u is* introduced *in micro-action i, when $u \in X_i$ and $u \notin \bigcup_{j=1}^{i-1} X_j$. We also say the argument u is* introduced by a positive precondition*, when it is introduced in micro-action i and a_i contains a positive precondition with arguments including u.*

The same as preferring an ordering with a higher number of preconditions in earlier micro-actions, the reason behind this decision is that positive preconditions are more restrictive, thereby they can prune inapplicable actions sooner in a forward search.

Objective Function

Besides the dependency graph, we investigate the opportunities to improve the objective function. We pursued two goals; the first is to provide a more intuitive objective function with planner-dependent rather than problem-dependent hyperparameters. The second goal is to improve its accuracy by providing a more reasonable objective function.

To choose a quotient graph from the space of all possible acyclic quotient graphs of the dependency graph, Areces et al. (2014) proposed an objective function which can be simplified as $f(\alpha) = \gamma c_1 SplitSize_\alpha + (1-\gamma) c_2 IntSize_\alpha$, where c_1 and c_2 are constants for the action and the hyperparameter γ is basically a way to select the correct element of the Pareto front that the problem creates. Two factors affect the objective function: the maximum parameter (interface) size ($IntSize_\alpha$) and the length of the micro-actions chain ($SplitSize_\alpha$). By decreasing either the maximum parameter size or the chain length of micro-actions, the value of the objective function is improved. However, since decreasing one of them increases the other, this approach controls the optimal point through a hyperparameter, γ.

While decreasing the length of the micro-actions chain reduces the plan length, decreasing the maximum parameter size of micro-actions reduces the number of ground instances. Ideally, we want to have shorter plans, because the complexity of finding plans is worst-case exponential in terms of their length. This results in micro-actions with longer parameter sizes, which might lead to an unmanageable number of ground instances. Having the maximum number of ground actions can be managed by a planner as a threshold, we need to set the hyperparameter γ with a value that lets the optimization procedure choose a split with the least chain size, such that the maximum parameter size of micro-actions prevents crossing that threshold.

Not only is this not a well-defined task for tuning γ, but also it causes the hyperparameter to become dependent on each problem. Therefore, we need to readjust it whenever we encounter a new problem. To address this issue, we propose another simple yet more efficient approach that implements our main goal. Let us assume ω is a threshold that determines the manageable number of ground instances. In this case, we can increase the parameter size of micro-actions as long as the number of their grounded instances does not exceed ω. Then, the objective function will simply be the least possible length of the micro-actions chain.

Definition 5 (Number of ground instances)**.** *For an action split $\alpha = \langle a_1[X_1], \dots, a_n[X_n]\rangle$, we use $\#\alpha$ to indicate the total number of ground instances of all micro-actions in α.*

As mentioned earlier, the upper-bound number of ground instances of an action schema $a[X]$ is $|\mathcal{O}|^{|X|}$. Although we can use this estimate to calculate the number of ground instances of micro-actions, we take into account the static relations (that will not modify by any action) in the precondition of micro-actions to prune inapplicable instances, and thereby, will have a more accurate estimate.

By closely examining the chain of micro-actions, we realize that certain micro-actions have a branching factor of one. More precisely, micro-actions with zero new parameters have a branching factor of one because we do not need to choose a value for their arguments. In simpler terms, they do not contribute to the problem's complexity since they are not elements that require decision-making. Hence, we can ignore them in minimizing the length of the chain.

Definition 6 (Branching micro-action)**.** *For a split $\alpha = \langle a_1[X_1], \ldots, a_n[X_n] \rangle$, we say $a_i[X_i]$ is a* branching micro-action *if and only if we have:*

$$X_i - \left(\bigcup_{j=1}^{i-1} X_j \right) \neq \emptyset$$

We use $\|\alpha\|$ to denote the number of branching micro-actions *in α.*

Still, we need another criterion before formally defining our objective function. As discussed earlier, not all parameter instantiations lead to applicable actions. We leverage the invariant knowledge of the problem to prevent those instantiations to some extent, during a forward search. As another mechanism, the same as Areces et al. (2014), we prioritize a split with a higher number of preconditions in earlier micro-actions. In order to use this criterion, we define a function to denote the number of preconditions of each micro-action in a split.

Definition 7 (Precondition count)**.** *For an action split $\alpha = \langle a_1[X_1], \ldots, a_n[X_n] \rangle$, with micro-actions $a_i[X_i] = \langle p_{i_{X_i}}, e_{i_{X_i}} \rangle$, we define:*

$$c(\alpha) = \langle |p_{1_{X_1}}|, \ldots, |p_{n_{X_n}}| \rangle$$

In other words, $c(\alpha)$ maps a split α to a tuple of values with the same size as the split, where its i^{th} element denotes the number of preconditions in the i^{th} micro-actions. We use lexicographic order to compare two tuples. More precisely, we say $\langle v_1, \ldots, v_n \rangle$ is less than $\langle w_1, \ldots, w_m \rangle$ if and only if

- $\exists i. v_i < w_i$ and $v_j = w_j$ for $1 \leq j < i$, or
- $n < m$ and $v_j = w_j$ for $1 \leq j \leq n$.

Definition 8 (Objective function)**.** *For a threshold ω denoting the maximum number of ground actions and a split $\alpha = \langle a_1[X_1], \ldots, a_n[X_n] \rangle$, we define $f(\alpha)$ to specify the value of objective function for the split α as a tuple of values with the following elements:*

1. *max(0, $\#\alpha - w$)*
2. $\|\alpha\|$
3. $\lfloor \log_2(\#\alpha) \rfloor$
4. $-1 \times c(\alpha)$
5. $\#\alpha$

Our objective function maps a split to a tuple of feature values in order of their importance; the earlier elements of the tuple have higher priorities but are coarser features while the later elements have lower priorities but are finer features. The first element determines if the number of ground actions of the split crossed our threshold, and if so how much. Its value is zero for all splits with a lower number of ground actions than our threshold. Then, the second value specifies the number of branching micro-actions of the split; this value affects the number of branching points during the search. For two splits with the same values of the first two features, we prefer the one if its number of ground actions is significantly lower than the other one because this value affects the performance of different parts of the search, such as calculating the heuristic value of a state. If the numbers of their ground actions are close to each other, then we choose the one with more preconditions in its earlier micro-actions. Otherwise, if the values of their first four features are the same, we prefer the split with the least number of ground actions.

Monte Carlo

Areces et al. (2014) showed that *split optimization* is an NP-complete problem. Therefore, they proposed two greedy approximate methods to find the final split. However, since the complexity of this problem is proportional to the number of preconditions and effects, we expect the search space should not be too large in most cases. Thus, we adopted a variant of the *Monte Carlo* method to cover this space to the greatest extent feasible. More precisely, within a specified time limit, we conduct multiple random walks with two enhancing strategies: 1) we prevent a duplicate random walk (by tracking the fully explored nodes), and also 2) nodes with objective values worse than the best-founded objective value are pruned.

Here, a node is a partial chain of micro-actions in which some of the action's atomic parts are uniquely assigned to the chain's elements and the rest of them are left unassigned. Starting from an empty chain, each walk randomly follows a path to reach a final node, which is a complete chain. The successor of a node can be constructed by either assigning an unassigned atomic part to the last element of the chain or adding a new (empty) micro-action to the end of the chain. To keep the search space small, we prune some sub-optimal successors, such as those having two consecutive empty micro-actions or assigning an atomic part whose parameters are disjoint with the parameters of the last micro-action.

Evaluation

We conducted two experiments to evaluate the effectiveness of our approach [2]. In the first experiment, we sought to evaluate the contribution of each of our proposed components in solving hard-to-ground problems. In the second experiment, our goal was to understand how our approach influences the performance of planners when dealing with a broader range of problems.

We chose four planners with different technologies for our experiments: LAMA (Richter and Westphal 2010), a heuristic-based planner; Maidu (Corrêa et al. 2023b), a portfolio-based planner and the winner of IPC'23 in the classical-satisficing track; a variant of BFWS (Lipovetzky and Geffner 2017), featuring width-based search; and Powerlifted (Corrêa et al. 2023a), the state-of-the-art planner for hard-to-ground problems. The variant of BFWS

[2]The source code is available at: https://github.com/melahi/enhanced-action-splitter

that we used in our experiment is a combination of two modes of this planner. First, the planner is run in `goalcount-only` mode for at most 15 minutes, then it switches to `f5-initstate-relevant`. Additionally, we used a slightly modified version of this planner with more efficient memory usage. In our experiments, we ran the planners with a 30 minute time limit on 2.50 GHz Intel Processor with an 8 GB memory limit.

To determine the threshold ω in our approach, we roughly estimated that the upper bound for the number of ground actions that a planner can deal with is 10^6. Therefore, we set $\omega = \frac{10^6}{|\mathcal{A}|}$, where $\mathcal{A}$ is the set of action schemas. Also, we considered a time limit of $max(50, \frac{500}{|\mathcal{A}|})$ for the Monte Carlo search. In the report of our experimental results, we refer to our approach as EAS (Enhanced Action Splitter).

Components Evaluation

We proposed three improvements to the Areces et al. (2014) approach. In our first experiment, we sought to evaluate the effectiveness of each of them. We conducted our experiment on a set of hard-to-ground domains, including *Pipesworld-Tankage*, *Genome Edit Distance (GED)*, *Organic Synthesis-Alkene*, *Organic Synthesis-MIT*, and *Organic Synthesis-Original*. We started with the Areces et al. approach with the hyperparameter $\gamma = 0$ as our baseline. In the first step, we only improved its dependency graph (EAS_g). Then, we changed its objective function to ours (EAS_obj). Finally, in the last step, we switched to the Monte Carlo search to find the best chain of micro-actions (EAS_M). Table 1 reports the number of solved instances in each case.

According to the results, EAS_g considerably improved the performance of BFWS. Its influence is also noticeable for the Organic Synthesis-Original domain for LAMA and Maidu. Moreover, the results show our proposed objective function remarkably increases the coverage of planners for the Pipesworld domain. However, the hill-climbing search (Areces et al. 2014) in EAS_obj might not completely benefit from this advantage, due to finding local optima. By switching to the Monte Carlo search in EAS_M, we could benefit from the advantage of our proposed objective function to a greater extent. Overall, we can conclude all three proposed improvements significantly contribute to enhancing the performance of planners in this experiment.

General Evaluation

We have a minimalistic design philosophy behind our proposed approach: we tried to introduce as few hyperparameters as possible with intuitive meanings that depend on the planner rather than the problem. This way, we can easily tune our approach for a wide range of problems. Additionally, as another goal coming from our design philosophy, it tries not to intervene until there is a necessity. In other words, it does not split actions, unless it detects that the number of ground actions will be overwhelmingly high.

To evaluate our goals, we designed another experiment with a large set of domains, including hard-to-ground domains and domains from IPC'14, IPC'18, and IPC'23, except those having universally quantified effects, as our implementation does not support them yet. In this experiment, we ran LAMA, Maidu, and BFWS in four different cases. In the first case, we fed the planner with the original problems in the domains. In the second and third cases, we use Areces et al. approach (2014) with the hyperparameter $\gamma = 0$ and $\gamma = 0.8$, respectively, and in the last case, we use our approach. To complete our experiment, we also evaluated the performance of Powerlifted, as a state-of-the-art planner for solving hard-to-ground problems. Table 2 shows the number of solved instances of each case in this experiment.

According to the results, the Areces et al. approach (2014) with $\gamma = 0$ improves the performance of LAMA and Maidu for hard-to-ground problems. However, it has a considerable overhead on other domains, and also on the performance of the BFWS planner, even for hard-to-ground domains. On the other hand, with hyperparameter $\gamma = 0.8$ this approach has little overhead on the IPC benchmarks, but it is not as effective as $\gamma = 0$ for the hard-to-ground problems. Analyzing the results, we realize that not only does our approach effectively improve the performance of planners in hard-to-ground domains, but it also has a minimal additional cost on the IPC domains. Interestingly, it also improved the performance of planners in a few domains, including the 2023 IPC domain Slitherlink, for LAMA and Maidu. Moreover, the results show our approach is quite competitive with Powerlifted, the state-of-the-art planner, in hard-to-ground domains. This experiment confirms that we have met our goals to a great extent.

Conclusion

We have addressed three different aspects of the action-splitting approach of Areces et al. (2014) that addresses planning problems in which straightforward grounding of schematic actions produces impractically many ground actions. First, we have proposed an improvement on the dependency graph by encoding problem invariants into it and eliminating some superfluous edges. Also, we have suggested an additional criterion to choose a more effective topological ordering of the dependency graph. Then, we have provided a more meaningful and accurate objective function described by a priority list of features. Finally, we have proposed to utilize a variant of the Monte Carlo search algorithm, which is more suitable for our specific problem.

Our results demonstrate that the resulting action-splitting method clearly improves on Powerlifted on hard-to-ground domains, but also generally avoids performance deterioration on other domains. Overall, combining our method with state-of-the-art planners clearly outperforms existing planners on a mix of hard-to-ground and standard domains. This is important when it is not known a priori whether domains to be solved are hard-to-ground or not.

References

Areces, C. E.; Bustos, F.; Dominguez, M.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In *ICAPS 2014. Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, 11–19. AAAI Press.

		LAMA				Maidu				BFWS			
	#	Base	EAS_g	EAS_{obj}	EAS_M	Base	EAS_g	EAS_{obj}	EAS_M	Base	EAS_g	EAS_{obj}	EAS_M
Pipesworld	50	17	20	27	37	33	34	23	37	10	32	47	50
GED	156	156	156	156	156	156	156	156	156	156	156	156	156
$Synthesis_{Alkene}$	18	18	18	18	18	18	18	18	18	6	18	18	18
$Synthesis_{MIT}$	18	16	17	17	18	15	18	18	18	3	18	17	18
$Synthesis_{original}$	20	5	10	12	11	2	11	12	11	0	6	15	16
Total	262	212	221	230	240	224	237	227	240	175	230	253	258

Table 1: Coverage of hard-to-ground domains to evaluate different components of our proposed approach. The Base column represents the coverage achieved by the Areces et al. approach with $\gamma = 0$ as the baseline, EAS_g illustrates the results when we incorporate the dependency graph improvement, EAS_{obj} presents the coverage when we also utilize the proposed objective function, and EAS_M specifies the result when we use all improvements including our Monte Carlo search.

		LAMA				Maidu				BFWS				Powerlifted
	#		γ_0	$\gamma_{0.8}$	EAS		γ_0	$\gamma_{0.8}$	EAS		γ_0	$\gamma_{0.8}$	EAS	
Pipesworld	50	18	17	13	37	18	33	21	37	25	10	27	**50**	48
GED	156	**156**	**156**	153	**156**	**156**	**156**	**156**	**156**	0	**156**	**156**	**156**	**156**
$Synthesis_{Alkene}$	18	15	**18**	14	**18**	15	**18**	14	**18**	**18**	6	14	**18**	**18**
$Synthesis_{MIT}$	18	2	16	1	**18**	2	15	1	**18**	2	3	1	**18**	**18**
$Synthesis_{original}$	20	1	5	0	11	1	2	0	11	1	0	0	**16**	15
Agricola	20	12	10	13	12	15	14	**17**	14	11	10	10	11	10
Barman	20	**20**	**20**	**20**	**20**	**20**	18	**20**	**20**	4	0	19	4	**20**
Childsnack	20	6	0	6	6	18	16	**19**	**19**	0	0	0	0	6
Data Network	20	13	0	13	13	**16**	1	**16**	**16**	11	1	10	11	10
Floortile	20	2	0	2	2	**20**	1	**20**	**20**	0	0	0	0	1
Folding	20	11	0	11	12	11	0	11	**13**	**13**	0	**13**	8	12
$GED_{IPC'14}$	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	0	**20**	**20**	**20**	**20**
Hiking	20	**20**	12	**20**	**20**	**20**	16	**20**	**20**	13	2	11	13	19
Labyrinth	20	1	8	1	4	0	4	0	4	**20**	1	**20**	18	0
Nurikabe	20	10	0	0	11	**19**	0	0	18	15	0	0	15	0
Openstacks	20	**20**	8	11	**20**	**20**	3	8	**20**	**20**	0	0	**20**	0
$Synthesis_{IPC'18}$	20	3	10	1	14	3	8	1	15	3	0	1	**19**	17
Parking	20	**20**	0	**20**	**20**	**20**	**20**	**20**	**20**	0	0	0	0	**20**
Quantum Layout	20	**20**	17	**20**	**20**	**20**	**20**	**20**	**20**	**20**	17	**20**	**20**	**20**
Ricochet Robots	20	14	0	14	13	**17**	0	**17**	**17**	13	0	12	13	0
Rubiks Cube	20	**20**	0	0	**20**	**20**	0	0	**20**	6	0	0	6	0
Slitherlink	20	0	0	0	**6**	0	0	0	5	3	1	3	2	2
Snake	20	5	6	5	7	15	9	14	16	19	13	**20**	16	16
Spider	20	16	7	9	16	17	8	9	16	**20**	5	15	15	0
Termes	20	**16**	5	**16**	**16**	13	3	14	13	9	3	9	9	0
Tetris	20	16	0	16	16	**20**	0	19	19	0	0	0	0	0
Thoughtful	20	15	5	8	11	**20**	5	12	**20**	**20**	4	19	**20**	**20**
Transport	20	**16**	0	**16**	**16**	**16**	0	15	15	13	3	13	13	14
Visitall	20	**20**	4	4	**20**	**20**	4	4	**20**	**20**	4	4	**20**	**20**
Total	742	508	344	427	575	572	394	488	**640**	299	259	417	531	482

Table 2: Coverage of hard-to-ground and IPC benchmarks for LAMA, Maidu, and BFWS planners with different splitting approaches, in addition to Powerlifted. For the first three planners, the first column specifies the coverage when there is no splitting, the second column is for splitting with the Areces et al. approach when $\gamma = 0$, the third column shows the result for $\gamma = 0.8$, and the last column describes the coverage of our approach. The best overall coverages are highlighted in boldface, and the best coverages of a planner in different splitting approaches are shown in regular black font while the rest are in gray.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Corrêa, A. B.; Frances, G.; Hecher, M.; Longo, D. M.; and Seipp, J. 2023a. The powerlifted planning system in the IPC 2023. *IPC2023–Classical Tracks*.

Corrêa, A. B.; Frances, G.; Hecher, M.; Longo, D. M.; and Seipp, J. 2023b. Scorpion Maidu: width search in the Scorpion planning system. *IPC2023–Classical Tracks*.

Corrêa, A. B.; Frances, G.; Pommerening, F.; and Helmert, M. 2021. Delete-relaxation heuristics for lifted classical planning. In *ICAPS 2021. Proceedings of the 31st International Conference on Automated Planning and Scheduling*, 94–102. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2020. Lifted successor generation using query optimization techniques. In *ICAPS 2020. Proceedings of the 30th International Conference on Automated Planning and Scheduling*, 80–89. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2022. The FF heuristic for lifted classical planning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI-17)*, 9716–9723. AAAI Press.

Ernst, M.; Millstein, T.; and Weld, D. S. 1997. Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 1169–1176. Morgan Kaufmann Publishers.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Haslum, P.; et al. 2011. Computing genome edit distances using domain-independent planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 45–51.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5): 503–535.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Lipovetzky, N.; and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*. AAAI Press.

Matloob, R.; and Soutchanski, M. 2016. Exploring organic synthesis with state-of-the-art planning techniques. In *Proceedings SPARK Workshop*, 52–61.

Milidiú, R. L.; dos Santos Liporace, F.; and de Lucena, C. 2003. Pipesworld: planning pipeline transportation of petroleum derivatives. *ICAPS'03 - Workshop on the competition: Impact, organization, evaluation, benchmarks*.

Penberthy, J. S.; and Weld, D. S. 1992. UCPOP: a sound, complete, partial order planner for ADL. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, 103–114. Morgan Kaufmann Publishers.

Richter, S.; and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Ridder, B.; and Fox, M. 2014. Heuristic evaluation based on lifted relaxed planning graphs. In *ICAPS 2014. Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, volume 24, 244–252.

Rintanen, J. 2017. Schematic Invariants by reduction to ground invariants. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, 3644–3650. AAAI Press.

Robinson, N.; Gretton, C.; Pham, D.-N.; and Sattar, A. 2008. A compact and efficient SAT encoding for planning. In *ICAPS 2008. Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*. AAAI Press.

Wichlacz, J.; Höller, D.; and Hoffmann, J. 2022. Landmark heuristics for lifted classical planning. In *IJCAI 2022, Proceedings of the 31st International Joint Conference on Artificial Intelligence*, 4665–4671. AAAI Press.