On an Attempt at Casting Orbit Search as a Task Transformation

Daniel Gnad[1], **David Speck**[1,2]

[1]Linköping University, Sweden
[2]University of Basel, Switzerland
⟨daniel.gnad, david.speck⟩@liu.se

Abstract

State-space reduction techniques are a powerful means to enhancing the performance of search-based planners. Recent work has shown that decoupled search, which compactly represents sets of states, can be cast as a task reformulation. Concretely, it is possible to exactly simulate the behavior and reduction of (non-optimal) decoupled search by applying a transformation to the input task and running regular search on the transformed task. In this work, we investigate if the same approach is feasible for symmetry breaking, in particular orbit space search (OSS). One of the challenges in this endeavor is that OSS dynamically computes a *canonical representative* of every state generated during search. To represent this computation as a task transformation, however, we need to fix the procedure at transformation time. This leads to reduced pruning potential, which we discuss in detail and verify empirically. We also discuss an approach that can fully simulate OSS, at the cost of vastly blowing-up the task size.

Introduction

Classical planning involves finding a path between states in a compactly described state-space. State-space reduction techniques are powerful tools for improving the performance of planners by addressing issues such as accidental complexity that may arise from task modeling (Haslum 2007). Established reduction techniques can be broadly divided into two groups: pruning methods remove states or transitions, such as symmetry breaking or partial order reduction, while compression techniques, such as symbolic or decoupled search, represent the state space more compactly.

In domain-specific settings, state-space reduction techniques are often realized by reformulating the task at hand; prominent examples are solving the Rubik's Cube (Korf 1997) or finding algorithms for matrix multiplication (Fawzi et al. 2022; Speck et al. 2023). In domain-independent settings, however, the most common approach to realize powerful state-space reduction techniques is through specialized algorithms and implementations (Gnad and Hoffmann 2018; Torralba et al. 2017; Torralba and Sievers 2019; Speck 2022), which often limits the transfer of other planning techniques or novel advances to these specialized approaches. Therefore, a natural question arises: Can modern domain-independent state-space reduction techniques be described by means of task reformulation? A recent successful example of this is the work of Speck and Gnad (2024), which showed that decoupled search can be embodied by searching on a transformed task, allowing the full toolbox of planning techniques to be used along with decoupled search.

In this paper, we analyze the possibility and demonstrate methods for implementing symmetry breaking through task reformulation. That is, given a planning task, we create a new task such that standard search algorithms on the new task mimic specialized symmetry-breaking algorithms, such as orbit space search (Domshlak, Katz, and Shleyfman 2015), applied to the original task. While symmetry-breaking methods can theoretically and practically reduce the state space of forward search exponentially (Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012), and this theoretical guarantee carries over to our task reformulation, certain properties of the transformation we present seem to hinder the practical ability to achieve the pruning power of specialized algorithms. We believe that these observations, combined with insights into why this pruning power is not effectively translated by the current method of encoding symmetry breaking in a planning task, can inspire new ideas and help overcome the identified obstacles.

Background

In this section, we introduce the necessary background for our work by defining classical planning and orbit space search.

Classical Planning

We adopt a common definition of planning in finite-domain representation (FDR) (Bäckström and Nebel 1995; Helmert 2009) without axioms and derived variables. A FDR planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{V}$ denotes a set of *variables*, $\mathcal{O}$ denotes a set of *operators*, $\mathcal{I}$ denotes the *initial state*, and $\mathcal{G}$ denotes the partial *goal state*. Each variable $v \in \mathcal{V}$ has a finite *domain* D_v, an assignment $\langle v, d \rangle$ to a variable is called a *fact*. A *partial state* is a consistent assignment to some variables in $\mathcal{V}$. A *state* is a complete and consistent assignment to all variables in $\mathcal{V}$. For a partial state p we denote the subset of variables defined in p by $V(p) \subseteq \mathcal{V}$. Furthermore, we write $s(v) = x$ for the assignment of v to x made in a (partial) state s. By $\mathcal{S}$ we refer to the set of all states.

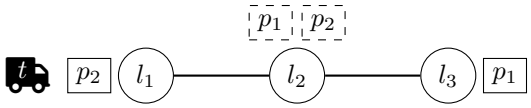

Figure 1: Illustration of the initial state (solid packages) and goal (dashed packages) of the running example.

Planning operators encode the transitions between states. Each operator $o \in \mathcal{O}$ is a triplet $\langle pre(o), \mathit{eff}(o), \mathit{ceff}(o)\rangle$. The *precondition* $pre(o)$ and *effect* $\mathit{eff}(o)$ are partial states, and $\mathit{ceff}(o)$ is a set of *conditional effects* $(cond \rhd \langle v, x\rangle)$, where $cond$ is a partial state and $\langle v, x\rangle$ is a fact.[1] Every operator $o \in \mathcal{O}$ is associated with a non-negative cost, $cost : \mathcal{O} \to \mathbb{R}_0^+$.

An operator $o \in \mathcal{O}$ is *applicable* in a state $s \in \mathcal{S}$ if $pre(o) \subseteq s$. The result of applying the operator o to a state s is a state $t = s[\![o]\!]$, where $t(v) = x$ for all $\langle v, x\rangle \in \mathit{eff}(o)$, $t(v) = x$ for all $(cond \rhd \langle v, x\rangle) \in \mathit{ceff}(o)$ with $cond \subseteq s$, and $t(v) = s(v)$ for all variables that do not have such effects.

The solution to a FDR task is a *plan*, i.e., a sequence of operators π iteratively applicable in $\mathcal{I}$ and ending in a goal state s_G such that $\mathcal{G} \subseteq s_G$. The plan is *optimal* if its summed-up cost, denoted by $cost(\pi)$, is minimal.

A SAS$^+$ planning task is a simplified version of a FDR planning task that does not include conditional effects (Bäckström and Nebel 1995). Formally, a SAS$^+$ planning task is a FDR planning task $\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{O}\rangle$ where for each operator $o = \langle pre(o), \mathit{eff}(o), \mathit{ceff}(o)\rangle \in \mathcal{O}$, it holds that $\mathit{ceff}(o) = \emptyset$. To simplify the notation, we sometimes exclude the empty components of a SAS$^+$ operator by representing it as $o = \langle pre(o), \mathit{eff}(o)\rangle$.

Moreover, we say that such an operator $o \in \mathcal{O}$ *affects* a variable $v \in \mathcal{V}$ if it has an effect on it, formally $v \in V(\mathit{eff}(a))$. Furthermore, by $post(o) = \mathit{eff}(o) \cup \{\langle v, d\rangle \in pre(o) \mid v \notin V(\mathit{eff}(o))\}$, we refer to the partial state, called the *postcondition*, that is always true after applying a SAS$^+$ operator $o \in \mathcal{O}$. Finally, $V(o) = V(post(o))$ refers to the variables of the postcondition of a SAS$^+$ operator o, which are also the variables of the precondition and effect of o.

In the remainder of this paper, we will use the following running example in the form of a SAS$^+$ planning task.

Example 1 (Running Example)**.** *Let us consider a simple logistics scenario with three locations connected in a line, l_1, l_2, l_3, along with two packages, p_1 and p_2, and one truck t. These are represented by the variables $\mathcal{V} = \{t, p_1, p_2\}$, with domains: $D_t = \{l_1, l_2, l_3\}$ and $D_{p_1} = D_{p_2} = \{t, l_1, l_2, l_3\}$. Initially, the truck and the package p_2 are at position l_1, modeled as $\mathcal{I}(t) = \mathcal{I}(p_2) = l_1$, while the package p_1 is initially at l_3, i.e., $\mathcal{I}(p_1) = l_3$. The goal is to transport both packages to l_2, formally $\mathcal{G} = \{\langle p_1, l_2\rangle, \langle p_2, l_2\rangle\}$. Both the initial and goal states are illustrated in Figure 1.*

[1] We assume well-formed effects, meaning that multiple conditional effects assigning different values to the same variable cannot trigger in the same state, and unconditional effects do not assign different values to variables than the conditional ones.

There are three types of operators in this example: drive *operators that drive the truck between locations,* load *operators responsible for loading a package onto the truck, and* unload *operators for unloading a package from the truck. All operators have unit cost of* 1*. Formally, we have a* drive *operator for any $i, j \in \{1, 2, 3\}$ with $|i - j| = 1$:*

- $\mathrm{drive}(l_i, l_j) = \langle\{\langle t, l_i\rangle\}, \{\langle t, l_j\rangle\}\rangle$

And we have the following load and unload operators for any $p \in \{p_1, p_2\}$ and $l \in \{l_1, l_2, l_3\}$:

- $\mathrm{load}(p, l) = \langle\{\langle t, l\rangle, \langle p, l\rangle\}, \{\langle p, t\rangle\}\rangle$
- $\mathrm{unload}(p, l) = \langle\{\langle t, l\rangle, \langle p, t\rangle\}, \{\langle p, l\rangle\}\rangle$

In Example 1, the number of states grows exponentially with the number of packages. Symmetry breaking in the form of orbit space search can significantly reduce this blow-up of the search.

Orbit Space Search

Symmetry breaking considers equivalence classes of symmetrical states in the search space, and allows for using representative states of each equivalence class. Shleyfman et al. (2015) introduced the notion of *structural symmetries*, which capture previously proposed concepts of symmetry breaking for classical planning. These symmetries relabel the factored representation of a given SAS$^+$ planning task. Operators are mapped to operators, variables to variables, and values to values (preserving the variable/value pairs structure). This relabeling induces an automorphism of the state space. We follow the definition of structural symmetries from Wehrle et al. (2015).

Definition 1 (Structural Symmetry)**.** *For a SAS$^+$ task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, G\rangle$, let P denote the set of all its facts. A* structural symmetry *is a permutation $\sigma : P \cup \mathcal{O} \to P \cup \mathcal{O}$ such that:*

1. *$\sigma(P) = P$, where $P := \{\{\langle v, d\rangle \mid d \in D_v\} \mid v \in \mathcal{V}\}$.*
2. *$\sigma(\mathcal{O}) = \mathcal{O}$, and, for all $o \in \mathcal{O}$, $\sigma(pre(o)) = pre(\sigma(o))$, $\sigma(\mathit{eff}(o)) = \mathit{eff}(\sigma(o))$, and $cost(\sigma(o)) = cost(o)$.*
3. *$\sigma(G) = G$.*

Here, for a set X we define $\sigma(X) := \{\sigma(x) \mid x \in X\}$.

A set of structural symmetries Σ for a planning task Π induces a subgroup Γ of the automorphism group of the state space of Π. This, in turn defines an equivalence relation over the states $\mathcal{S}$ of Π, where we say that a state s is *symmetric* to another state s' iff there exists an automorphism $\sigma \in \Gamma$ such that $\sigma(s) = s'$. The composition of two structural symmetries of Π is again a structural symmetry for Π. Similar to operators, we say that a structural symmetry *affects* a variable $v \in \mathcal{V}$ if there exists a value $d \in D_v$ such that $\sigma(\langle v, d\rangle) \neq \langle v, d\rangle$. If a variable v is not affected by a structural symmetry σ, then σ is the identity mapping over the domain D_v. By $V(\sigma)$ we denote the set of variables affected by σ. For details on how the structural symmetries of a planning planning task are computed we refer the reader to the literature (Pochter, Zohar, and Rosenschein 2011).

Forward search algorithms with symmetry elimination in their purest form aim to not consider all states $s \in \mathcal{S}$, but only a single representative state from the equivalence class of s. These equivalence classes are called *orbits* and are

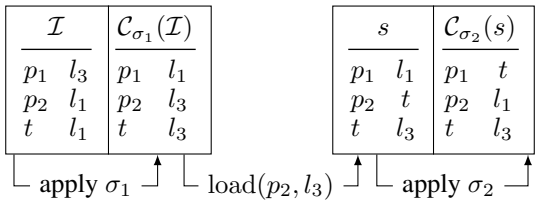

Figure 2: Illustration of a transition of orbit space search for the running example.

usually represented by one of its member states which is called the *canonical* state. A* with symmetry elimination, for example, is similar to its vanilla version without symmetry elimination (Hart, Nilsson, and Raphael 1968). However, it explores all applicable operators given a canonical state s and prunes the resulting successor states if another representative of their orbit has already been encountered during the search. Due to the properties of structural symmetries, this reduced state transition graph is guaranteed to still contain a shortest path from s to a goal state. Unfortunately, determining if two states are symmetric is NP-hard (Luks 1993). To overcome this, one can perform symmetry elimination by computing an approximated canonical representative with an incomplete ad-hoc procedure that is not guaranteed to detect all symmetries (Pochter, Zohar, and Rosenschein 2011). In this work, we consider the *orbit space search* (OSS) algorithm introduced by Domshlak, Katz, and Shleyfman (2015). The idea of OSS is to replace each state by its approximated canonical representative, resulting in a search over the state transition graph induced by the (approximated) canonical states.

The procedure that computes the canonical representative of a state minimizes the lexicographic representation of the state. For this, we assume an order on the variables, as well as on their domains. With this, a hill-climbing search is invoked that applies a structural symmetry to a state s whenever the output state is lexicographically smaller than s. The search stops once it hits a local minimum. For a state s, we denote its canonical state by $\mathcal{C}(s)$ or $\mathcal{C}_\sigma(s)$ to indicate the structural symmetry σ used to permute s into $\mathcal{C}_\sigma(s)$, i.e., the composition of structural symmetries used during the hill-climbing process to obtain the canonical form.

Example 2. *We illustrate a transition of orbit space search for our running example in Figure 2. There are several structural symmetries in our running example. For example, we can swap the positions l_1 and l_3 for all variables, but this operation must be applied to all of them simultaneously. We denote this symmetry by σ_1. Another structural symmetry σ_2 swaps the positions of the two packages. Let us consider $\langle p_1, p_2, t\rangle$ as the order of the variables, $\langle t, l_1, l_2, l_3\rangle$ as the order of the domains of the packages, and $\langle l_1, l_2, l_3\rangle$ as the order of the domain of the truck. More precisely, p_1 is our most significant variable, and t is the lowest value in its domain. In other words, we prioritize the value t over l_1, l_2, and l_3 for variable p_1 when determining a canonical state.*

Starting a search on our running example, first we compute the canonical representative $\mathcal{C}_{\sigma_1}(\mathcal{I})$ of the initial state. The canonical $\mathcal{C}_{\sigma_1}(\mathcal{I})$ of the initial state is shown in the left of Figure 2. $\mathcal{C}_{\sigma_1}(\mathcal{I})$ results from $\mathcal{I}$ by swapping the values l_1 and l_3 for all variables, and is considered lexicographically smaller than $\mathcal{I}$, since the value of the most significant variable p_1 is lower. In the canonical state, suppose we apply operator load(p_2, l_3). *The resulting state s and its canonical $\mathcal{C}_{\sigma_2}(s)$ are shown in the right of the figure. Here, σ_2 was applied, swapping the position of the two packages without touching the truck.*

Note that neither $\mathcal{I}$ nor s are ever considered for expansion by orbit space search. The initial state is permuted before starting the search, and for the newly generated state s only the canonical state $\mathcal{C}_\sigma(s)$ is further considered.

Orbit Search as a Task Transformation

In this section, we describe and exemplify our task transformation. It takes as input a SAS$^+$ planning task and a set of structural symmetries and generates a FDR task in which the symmetries are directly encoded in the effects of its operators. We then discuss the strengths but also the weaknesses of our transformation and possible ways to overcome them, which may result in a significant increase in task size.

Task Transformation

We define a task transformation called oss that transforms a given SAS$^+$ planning task Π and a set of structural symmetries Σ of Π into a FDR planning task Π^{oss}. Running a vanilla search algorithm on Π^{oss} will then correspond to running the equivalent orbit space search algorithm on the original task Π.

Our main observation is that we can model the standard two-step approach of orbit space search, i. e., (1) apply operator o to a state s, then (2) find a structural symmetry σ that transforms $s' = s[\![o]\!]$ into its canonical $\mathcal{C}_\sigma(s')$, directly as conditional effects of a transformed operator o^{oss}.

Assume an easy case where $V(o) = \mathcal{V}$ for an operator $o \in \mathcal{O}$. Here, no matter in which state s the operator o is applied, we know exactly what the successor state s' will look like, namely $s' = post(o) = \mathit{eff}(o) \cup \{\langle v, d\rangle \in pre(o) \mid v \notin V(\mathit{eff}(o))\}$. We can now precompute the structural symmetry that permutes s' to its canonical $\mathcal{C}(s') = \mathcal{C}(post(o))$, and adapt o's effects to be exactly $\mathcal{C}(s')$. Thereby, we merge the two aforementioned steps into one, by applying o, we directly compute the canonical successor state.

The same is possible also if $V(o) \subset \mathcal{V}$ if we simply skip variables that are not touched by the operator. More formally, we compute the canonical of the postcondition $post(o)$ by the same hill-climbing process as before, but if a structural symmetry σ permutes a fact $\langle v, d\rangle$ to another one $\langle v', d'\rangle$ where $v \notin V(o)$, then we ignore this when checking if $\sigma(post(o))$ is lexicographically smaller than $post(o)$. We must not, however, ignore such variables when adapting o's effects, because if we commit to the symmetry that obtains the canonical, then we have to apply it fully to all variables it affects. For these variables, we encode the application of σ as conditional effects. Formally, our transformation is defined as follows.

Definition 2 (Symmetry Transformation)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ *be SAS*$^+$ *planning task and* Σ *be a set of structural symmetries for* Π*. We define the* symmetry transformation oss *as a function that produces a new FDR planning task* $oss(\Pi, \Sigma) = \Pi^{oss} = \langle \mathcal{V}, \mathcal{O}^{oss}, \mathcal{C}(\mathcal{I}), \mathcal{G} \rangle$*, where* $\mathcal{C}(\mathcal{I})$ *is the canonical initial state, and the permuted operators are* $\mathcal{O}^{oss} := \{o^{oss} \mid o \in \mathcal{O}\}$*, where:*

- $pre(o^{oss}) = pre(o)$,
- $cost(o^{oss}) = cost(o)$,
- $\mathit{eff}(o^{oss}) = \sigma(\mathit{eff}(o))$*, and*
- $\mathit{ceff}(o^{oss}) = \bigcup_{v \in V(\sigma) \setminus V(o), d \in D_v} \{(\{\langle v, d \rangle\} \rhd \sigma(\langle v, d \rangle))\}$,

where the structural symmetry σ *is obtained by computing the canonical state* $\mathcal{C}_\sigma(post(o))$.

In words, the initial state of the transformed task is the canonical of the initial state of Π. For every operator $o \in \mathcal{O}$, we compute the structural symmetry σ that produces the canonical partial state of o's postcondition $post(o)$. The transformation keeps the original preconditions and cost, permutes the effects according to σ, and adds conditional effects that properly permute the variables affected by σ, but not by o.

For a transition $s \rightarrow^o s'$ in the original task and a structural symmetry σ obtained by computing $\mathcal{C}_\sigma(post(o))$, it is possible to see that first applying the operator o to s and then transforming the successor state s' with σ to its canonical form results in the same state as applying the operator o^{oss} to the state s. In other words, $\sigma(s[\![o]\!]) = s[\![o^{oss}]\!]$. Thus, this task transformation is equivalent to a particular instance of orbit space search, where the canonical state of successor states is computed based on the partial state $post(o)$ of the applied operator o.

Discussion

Like described in the previous section, our symmetry task transformation indeed instantiates a form of orbit space search. This means that no specialized algorithms, or adaptations to existing search implementations, are needed to for symmetry breaking in classical planning. Instead, a task formulation performed as a preprocessing step is sufficient. This not only simplifies the application of orbit search, but also makes it more efficient, as the hill-climbing search that computes the canonical is no longer needed. In our evaluation, we will show that there are domains in which this runtime advantage is significant. This comes with the drawback of a more complex task with a potentially large number of conditional effects. Although the number of conditional effects is only linear in the number of variables in $V(\sigma) \setminus V(o)$ and their domain size, the size of the task representation can grow significantly, by up to several orders of magnitude, because very likely *all* operators will have such effects.

The main issue of our transformation, however, is that in most planning tasks it will only be able to approximate the state-pruning capabilities of orbit search. That is because our transformation is static, fixed at transformation time, while orbit search dynamically computes the minimal representative of an orbit. On tasks where every operator affects all variables, our transformation does exactly the same as orbit search. On most planning domains, this is not the case, though. Suppose an operator o is applicable in two different states s and s', yielding two different successor states t and t', respectively. Then our transformation will apply the same structural symmetry σ, determined by the postcondition $post(o)$ of the operator o, to both successor states t and t'. That is, t is mapped to $\sigma(t)$ and t' is mapped to $\sigma(t')$, where the hill-climbing procedure of orbit space search may find another structural symmetry or even two different structural symmetries for the two successor states t and t', respectively, yielding lexicographically smaller states of the respective orbits. The higher the number of variables that are affected by some structural symmetry, but not by an operator, the higher the loss in potential pruning power of our transformation.

This hints at a potential fix. If the variables that are not part of the postcondition of an operator are problematic, then one could introduce a form of context splitting that uses a different structural symmetry depending on the partial state $p \subset s$ induced by $V' = \mathcal{V} \setminus V(o)$ in which an operator o is applied. This context splitting can be encoded as conditional effects of o, with conditions that take the different assignments to V'. While this will allow to exactly capture the dynamic nature of orbit search, it will lead to an increase in the transformed task size that is exponential in $|V'|$. We do not pursue the idea of context splitting further in this work, but consider it an interesting direction for future work.

Experiments

We implemented our symmetry task transformation in the Fast Downward 23.06 framework (FD) (Helmert 2006). Our experiments were conducted on a cluster of Intel Xeon Gold 6130 CPUs using Downward Lab 8.0 (Seipp et al. 2017), with runtime and memory limits of 5 min and 3 GiB, on all STRIPS instances from the optimal sequential tracks of the International Planning Competitions 1998–2023. For our own configurations, we impose the 5-min runtime limit on the entire process, i. e., transformation *and* search. We adopted the task transformation interface of Speck and Gnad (2024) for our own transformation, such that we can (1) run a search directly on the transformed task, or (2) write the transformed task to disk in (grounded) PDDL or FD's own `*.sas` format.

In the following, we evaluate our approach by performing search directly on the transformed task with blind search, i. e., FD's A* search with the blind heuristic, to focus on the state-space pruning capabilities of our transformation in comparison to orbit space search. Working on the original task, we denote default explicit search by **ES** and orbit space search by **OSS**.

We compare three different variants of our encoding, attempting to address the issue of variables not defined by an operator when computing its structural symmetry. The symmetry transformation described in Definition 2, which ignores variables not affected by an operator is called **STn** (no replacement for missing values). We also test two variants in which, when computing the structural symmetry of an operator o, we extend the postcondition $post(o)$ to a full state by

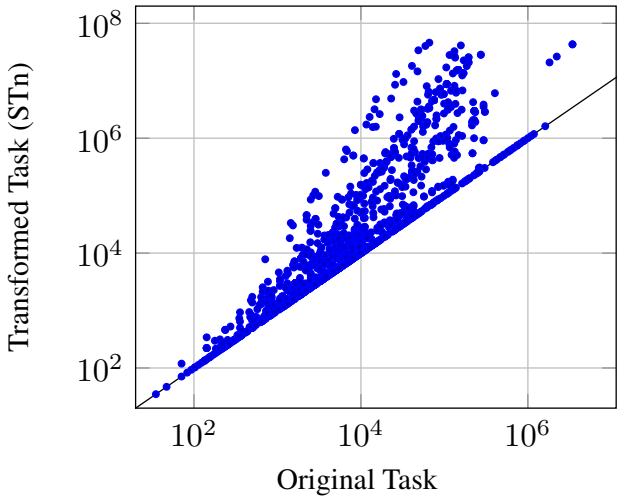

Figure 3: The plot shows a per-instance comparison of the original task size (x-axis) and the size of the transformed task (y-axis) of our STn variant.

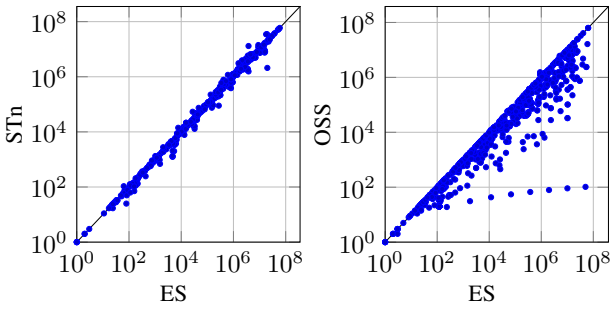

Figure 4: The plots show a per-instance comparison of the number of expanded states until the last f-layer for regular search (ES) vs. our task transformation (STn) on the left and orbit search (OSS) on the right.

setting the values of variables not in $V(o)$ to those of the initial state, denoted by **STi**, or to those of a random state, denoted by **STr**, using the same state for all operators. In other words, we determine the structural symmetry with respect to the successor state resulting from applying o to either the initial state or a specific randomly chosen state.

Transformation statistics. The transformation takes negligible time in most cases, with a maximum runtime of 23 seconds; the arithmetic mean is 0.4 seconds. In Figure 3 we compare the size of the original task to the size after the transformation (STn). The size of a task is measured as the encoding size in the same way as this is done by FD's translator component (Helmert 2009). The transformation can lead to a significant increase in the encoding size, up to more than one order of magnitude. The majority of instances only sees a moderate increase, though.

Planning performance. In Figure 4 we show the search-space size reduction achieved by our approach (left) and orbit space search (right). Quite obviously, our transformation does not achieve a significant reduction on the benchmark set. Where OSS reduces the search space by up to six orders of magnitude, the reduction achieved by our transformation is never larger than one order of magnitude, and there are more instances (compared to OSS), in which the search-space size increases.

Domain	#	ES	OSS	STn	STi	STr
agricola	20	0	**5**	0	**5**	1
airport	42	**13**	**13**	**13**	12	12
barman-11	20	4	**8**	4	4	4
barman-14	14	0	**3**	0	0	0
childsnack-14	20	0	**6**	0	4	0
depot	22	4	**5**	4	4	4
driverlog	20	**7**	**7**	**7**	6	**7**
elevators-08	20	**7**	**7**	6	**7**	**7**
elevators-11	13	**6**	**6**	5	**6**	5
gripper	20	8	**20**	8	9	7
hiking-14	20	11	**14**	11	11	10
miconic	141	41	41	**45**	**45**	44
mprime	35	19	**20**	19	14	14
mystery	28	**15**	**15**	**15**	10	10
nomystery-11	20	8	**9**	8	8	8
openstacks-08	30	22	**25**	22	**25**	23
openstacks-11	20	17	**19**	17	**19**	18
openstacks-14	14	2	**4**	2	**4**	2
organic-split	20	10	**11**	10	10	9
pegsol-08	30	27	**28**	27	**28**	**28**
pegsol-11	20	17	**18**	17	**18**	17
petri-net	20	**4**	2	**4**	**4**	3
pipesworld-noT	50	16	**20**	16	11	11
pipesworld-T	50	12	**17**	10	9	8
psr-small	32	31	**32**	**32**	**32**	31
satellite	36	5	5	**6**	5	5
scanalyzer-08	26	8	**10**	7	8	7
scanalyzer-11	18	7	**9**	6	7	6
sokoban-08	27	19	**23**	19	16	17
sokoban-11	19	**18**	**18**	**18**	15	15
storage	29	13	**15**	13	13	13
tetris-14	17	**9**	**9**	8	5	5
tidybot-11	7	**6**	**6**	**6**	5	5
tidybot-14	3	**2**	**2**	**2**	1	**2**
transport-14	20	**7**	**7**	**7**	6	4
trucks-strips	29	**5**	**5**	**5**	4	**5**
woodworking-08	22	2	2	2	**3**	2
woodworking-11	16	1	1	1	**2**	1
zenotravel	19	**7**	**7**	6	6	6
others	397	**107**	**107**	**107**	**107**	**107**
Sum	**1426**	517	**581**	515	508	483

Table 1: Coverage results of our approach (rightmost three columns) vs. the baseline (leftmost two columns).

In Table 1 we show coverage results (number of solved instances). OSS has a significant advantage over standard explicit-state search, gaining 64 instances overall, distributed over many domains. Comparing the variants of our

transformation, we observe quite large differences, overall and on a per-domain basis. We remind the reader that the only difference between the variants is *which* structural symmetry is encoded in the operator effects. The variant that simply ignores variables not affected by the operator does best overall, but does still worse than the ES baseline. Looking at individual domains, we see, though, that our variants sometimes achieve the same gains over ES as OSS does. This effect is visible for STi, for example, in agricola, openstacks, and pegsol.

In miconic, satellite, and woodworking, the results are actually encouraging. Here, STn and STi outperform both baselines, especially in miconic. In that latter domain, a closer look reveals that OSS does not achieve a strong search-space reduction. While the reduction of our transformation is even smaller, we see a clear runtime advantage. This indicates that if we can keep the increase in task size at bay when applying the idea of context splitting described earlier, we may be able to make search on the transformed task competitive with OSS.

Conclusion

We introduced a novel task transformation that can simulate specific forms of orbit space search, a state-of-the-art approach to symmetry breaking in classical planning. The transformation can be performed very efficiently, but can lead to a significant increase in the task representation size. Nevertheless, we observe empirically that, on domains where our transformation achieves a roughly similar reduction as native orbit space search, our approach has a clear runtime advantage. More concretely, it looks like the state-generation time during search can be accelerated when encoding orbit search as a transformation instead of computing canonical states dynamically during search.

A major drawback of our transformation is that, on most domains, it does not even get close to the search-space reduction achieved by the native implementation of orbit search. However, since the presented task transformation approach is more efficient during search, it may be possible in the future to achieve performance competitive with native orbit space search if one can improve the pruning power while keeping the increase in task size in check.

Acknowledgements

This work was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at the National Supercomputer Centre at Linköping University partially funded by the Swedish Research Council through grant agreement no. 2022-06725. David Speck was funded by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11(4): 625–655.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 343–347. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry Breaking in Deterministic Planning as Forward Search: Orbit Space Search Algorithm. Technical Report IS/IE-2015-03, Technion.

Fawzi, A.; Balog, M.; Huang, A.; Hubert, T.; Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Ruiz, F. J. R.; Schrittwieser, J.; Swirszcz, G.; Silver, D.; Hassabis, D.; and Kohli, P. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53.

Gnad, D.; and Hoffmann, J. 2018. Star-Topology Decoupled State Space Search. *Artificial Intelligence*, 257: 24–60.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Haslum, P. 2007. Reducing Accidental Complexity in Planning Problems. In Veloso, M. M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1898–1903.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Korf, R. E. 1997. Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, 700–705. AAAI Press.

Luks, E. M. 1993. Permutation Groups and Polynomial-Time Computation. In *Groups and Computation*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 139–175. American Mathematical Society.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting Problem Symmetries in State-Based Planners. In Burgard, W.; and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 1004–1009. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and Symmetries in Classical Planning. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Speck, D. 2022. *Symbolic Search for Optimal Planning with Expressive Extensions*. Ph.D. thesis, University of Freiburg.

Speck, D.; and Gnad, D. 2024. Decoupled Search for the Masses: A Novel Task Transformation for Classical Planning. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. AAAI Press.

Speck, D.; Höft, P.; Gnad, D.; and Seipp, J. 2023. Finding Matrix Multiplication Algorithms with Classical Planning. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 411–416. AAAI Press.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence*, 242: 52–79.

Torralba, Á.; and Sievers, S. 2019. Merge-and-Shrink Task Reformulation for Classical Planning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5644–5652. IJCAI.

Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Integrating Partial Order Reduction and Symmetry Elimination for Cost-Optimal Classical Planning. In Yang, Q.; and Wooldridge, M., eds., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1712–1718. AAAI Press.