The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems

Daniel Fišer
Czech Technical University in Prague,
Faculty of Electrical Engineering,
Prague, Czech Republic
danfis@danfis.cz

Abstract

In this paper, we focus on the inference of mutex groups in the lifted (PDDL) representation. We formalize the inference and prove that the most commonly used translator from the Fast Downward (FD) planning system infers a certain subclass of mutex groups, called fact-alternating mutex groups (fam-groups). Based on that, we show that the previously proposed fam-groups-based pruning techniques for the STRIPS representation can be utilized during the grounding process with lifted fam-groups, i.e., before the full STRIPS representation is known. Furthermore, we propose an improved inference algorithm for lifted fam-groups that produces a richer set of fam-groups than the FD translator and we demonstrate a positive impact on the number of pruned operators and overall coverage.

Introduction

Although classical planning problems are often described in the (schematic) Planning Domain Definition Language (PDDL) (McDermott 2000), i.e., in the lifted representation, most planners operate with a (non-schematic) ground representation such as STRIPS (Fikes and Nilsson 1971) or the finite domain representation (FDR or SAS$^+$) (Bäckström and Nebel 1995). These planners need to employ a translation process, called grounding, that transforms the lifted representation (PDDL) into STRIPS. The subsequent transformation from STRIPS, where states are described as sets of facts, into FDR, where states are assignments to a finite set of variables, requires an additional step of inference of mutex groups.

Mutex groups are sets of facts of which at most one is present in every reachable state. They are state invariants needed for the construction of a concise FDR representation, because they allow us to group sets of (STRIPS) facts into (FDR) variables so we do not need to encode each fact as a binary variable. This, however, is not the only application of mutex groups, because they provide a more general information about the structure of the planning task that can be used in pruning (Alcázar et al. 2013; Alcázar and Torralba 2015; Fišer and Komenda 2018), merge&shrink heuristics (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014), pattern databases (Culberson and Schaeffer 1996; Edelkamp 2001), SAT-based planning (Rintanen, Heljanko, and Niemelä 2006), or computing upper bounds on plan lengths (Abdulaziz, Gretton, and Norrish 2017).

It was shown that the inference of the maximum-sized mutex group is PSPACE-Complete, but there also exists a subclass of mutex groups, called fact-alternating mutex groups (fam-groups), defined over the input problem rather than over the whole reachable state space, of which inference is NP-Complete (Fišer and Komenda 2018).

In this paper, we focus on the inference of schematic mutex groups in the lifted representation. We show that the lifted mutex groups described by Helmert (2009) are always fam-groups after grounding because of the constraints posed on their structure and not because of the proposed inference algorithm. We propose an improvement of the Helmert's inference algorithm that produces a richer set of lifted fam-groups. Based on the previous results on operator pruning using fam-groups in STRIPS (Fišer and Komenda 2018), we introduce an operator pruning technique utilized during the grounding phase that uses the inferred lifted fam-groups.

PDDL and STRIPS

We consider the normalized non-numeric, non-temporal PDDL tasks without conditional effects and negative preconditions, and with all formulas being conjunctions of atoms (represented as sets of atoms). Since we will ground PDDL into STRIPS, we also split effects of PDDL actions into add effects (positive literals) and delete effects (negative literals) directly in the definition below to simplify the presentation.

In contrast to the normalization of PDDL tasks described by Helmert (2009), we do not support axioms (derived predicates) and we keep and utilize PDDL types. We also disregard conditional effects, but our implementation supports the full fragment of PDDL that is used in deterministic tracks of International Planning Competitions (IPCs).

Definition 1. A **normalized PDDL task** is a tuple $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$ where $\mathcal{B}$ is a non-empty set of **objects**, $\mathcal{T}$ is a non-empty set of **types** containing a default type denoted by $t_0 \in \mathcal{T}$, objects and types are associated by

a total function $\mathcal{D} : \mathcal{T} \mapsto 2^{\mathcal{B}}$ such that $\mathcal{D}(t_0) = \mathcal{B}$ and for every pair of types $t_i, t_j \in \mathcal{T}$ it holds that $\mathcal{D}(t_i) \subseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \supseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \cap \mathcal{D}(t_j) = \emptyset$. $\mathcal{V}$ is a denumerable set of variable symbols, each variable $v \in \mathcal{V}$ has a type $\tau_{var}(v) \in \mathcal{T}$.

$\mathcal{P}$ is a set of **predicate symbols**, each predicate $p \in \mathcal{P}$ has **arity** $\mathrm{ar}(p) \in \mathbb{N}$ and an associated type $\tau_{pred}(p, i) \in \mathcal{T}$ for every $i \in \{1, ..., \mathrm{ar}(p)\}$. An **atom** is of the form $p(s_1, \ldots, s_n)$, where $p \in \mathcal{P}$ is a predicate symbol, $n = \mathrm{ar}(p)$ is the arity of p, and each s_i is either an object $o \in \mathcal{D}(\tau_{pred}(p, i))$, or a variable $v \in \mathcal{V}$ with $\mathcal{D}(\tau_{var}(v)) \subseteq \mathcal{D}(\tau_{pred}(p, i))$. For a given atom $\alpha = p(s_1, \ldots, s_n)$, $\mathcal{V}[\alpha] \subset \mathcal{V}$ denotes a set of variables appearing in the atom, i.e., $\mathcal{V}[\alpha] = \{s_1, \ldots, s_n\} \cap \mathcal{V}$, and $\mathcal{P}[\alpha] = p$ denotes the predicate of α. Given a set of atoms X, we define $\mathcal{V}[X] = \bigcup_{x \in X} \mathcal{V}[x]$ and $\mathcal{P}[X] = \bigcup_{x \in X} \mathcal{P}[x]$. A **ground atom** is an atom α such that $\mathcal{V}[\alpha] = \emptyset$.

An **action** $a \in \mathcal{A}$ is a tuple $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$ where $\mathrm{pre}(a)$, $\mathrm{add}(a)$ and $\mathrm{del}(a)$ are sets of atoms, called **preconditions**, **add effects**, and **delete effects**, respectively. By $\mathcal{V}[a] = \mathcal{V}[\mathrm{pre}(a) \cup \mathrm{add}(a) \cup \mathrm{del}(a)]$ we denote a set of variables appearing in the action. For every pair of actions $a_i, a_j \in \mathcal{A}$, $a_i \neq a_j$, it holds that $\mathcal{V}[a_i] \cap \mathcal{V}[a_j] = \emptyset$. A **ground action** is an action a such that $\mathcal{V}[a] = \emptyset$.

ψ_I and ψ_G are sets of ground atoms, called **initial state** and **goal**, respectively.

Note that the type t_0 corresponds to the default PDDL type "object". The following definition describes the process of grounding, i.e., replacing all variables with objects of the corresponding type.

Definition 2. Given a set of variables $V \subseteq \mathcal{V}$, a **grounding** γ **restricted to** V is a function $\gamma : \mathcal{V} \cup \mathcal{B} \mapsto \mathcal{V} \cup \mathcal{B}$ such that $\gamma(v) \in \mathcal{D}(\tau_{var}(v))$ for every $v \in V$, and $\gamma(v) = v$ for every $v \in \mathcal{V} \setminus V$, and $\gamma(o) = o$ for every $o \in \mathcal{B}$, i.e., γ maps each variable $v \in V$ to an object from its corresponding domain $\mathcal{D}(\tau_{var}(v))$, and it is identity for everything else.

For an atom $\alpha = p(s_1, \ldots, s_n)$, by $\gamma[\![\alpha]\!]$ we denote the atom $p(\gamma(s_1), \ldots, \gamma(s_n))$. For a set of atoms X, we define $\gamma[\![X]\!] = \{\gamma[\![\alpha]\!] \mid \alpha \in X\}$. For an action $a \in \mathcal{A}$, we define $\gamma[\![a]\!] = \langle \gamma[\![\mathrm{pre}(a)]\!], \gamma[\![\mathrm{add}(a)]\!], \gamma[\![\mathrm{del}(a)]\!] \rangle$.

A set of all groundings is denoted by $\mathcal{G}$, and a set of all groundings restricted to $V \subset \mathcal{V}$ is denoted by $\mathcal{G}_V$. For a set of groundings $G \subseteq \mathcal{G}$ and an atom or an action x we define $G[\![x]\!] = \{\gamma[\![x]\!] \mid \gamma \in G\}$. For a set of groundings $G \subseteq \mathcal{G}$ and a set of atoms X we define $G[\![X]\!] = \bigcup_{\gamma \in G} \gamma[\![X]\!]$. For an action $a \in \mathcal{A}$, $\mathcal{G}_a$ denotes a shorthand for $\mathcal{G}_{\mathcal{V}[a]}$.

With the grounding defined, we can define the STRIPS planning task and the full grounding of a PDDL task, which is constructed by replacing all variables with all possible combinations of objects.

Definition 3. A STRIPS **planning task** Π is specified by a tuple $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, where $\mathcal{F} = \{f_1, \ldots, f_n\}$ is a set of facts, and $\mathcal{O} = \{o_1, \ldots, o_m\}$ is a set of operators. A **state** $s \subseteq \mathcal{F}$ is a set of facts, $s_I \subseteq \mathcal{F}$ is an **initial state** and $s_G \subseteq \mathcal{F}$ is a **goal** specification. An **operator** o is a tuple $o = \langle \mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o) \rangle$, where $\mathrm{pre}(o) \subseteq \mathcal{F}$ is a set of preconditions of the operator o, and $\mathrm{add}(o) \subseteq \mathcal{F}$ and $\mathrm{del}(o) \subseteq \mathcal{F}$ are sets of add and delete effects, respectively. All operators are well-formed, i.e., $\mathrm{add}(o) \cap \mathrm{del}(o) = \emptyset$ and $\mathrm{pre}(o) \cap \mathrm{add}(o) = \emptyset$. An operator o is **applicable** in a state s if $\mathrm{pre}(o) \subseteq s$. The **resulting state** of applying an applicable operator o in a state s is the state $o[s] = (s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o)$. A state s is a **goal state** iff $s_G \subseteq s$.

A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is applicable in a state s_0 if there are states $s_1, \ldots, s_n$ such that o_i is applicable in s_{i-1} and $s_i = o_i[s_{i-1}]$ for $i \in \{1, \ldots, n\}$. The resulting state of this application is $\pi[s_0] = s_n$. π is called a **plan** iff $\pi[s_I] \supseteq s_G$,

A set of facts $F \subseteq \mathcal{F}$ is **reachable** if there exists an operator sequence π such that $F \subseteq \pi[s_I]$. A set of facts $F \subseteq \mathcal{F}$ is **relaxed reachable** if there exists an operator sequence $\pi = \langle o_1, \ldots, o_n \rangle$ such that $F \subseteq \pi'[s_I]$ where $\pi' = \langle o'_1, \ldots, o'_n \rangle$ and $o'_i = \langle \mathrm{pre}(o_i), \mathrm{add}(o_i), \emptyset \rangle$ for every $i \in \{1, \ldots, n\}$. An operator o is (relaxed) reachable iff $\mathrm{pre}(o)$ is (relaxed) reachable. A state s is a **dead-end state** iff $s_G \not\subseteq s$ and there is no applicable operator sequence π such that $s_G \subseteq \pi[s]$.

Definition 4. Given a normalized PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$, the **full grounding** of $\mathcal{P}$ is a STRIPS planning task $\Pi^{\mathrm{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ constructed as follows.

Let $A = \bigcup_{a \in \mathcal{A}} \mathcal{G}_a[\![a]\!]$, and $X = \psi_I \cup \psi_G \cup \bigcup_{a \in A} (\mathrm{pre}(a) \cup \mathrm{add}(a) \cup \mathrm{del}(a))$. Then $\mathcal{F} := \{f_x \mid x \in X\}$, $s_I := \{f_x \mid x \in \psi_I\}$, $s_G := \{f_x \mid x \in \psi_G\}$, and $\mathcal{O} := \{o_a \mid a \in A\}$ with $\mathrm{pre}(o_a) = \{f_x \mid x \in \mathrm{pre}(a)\}$, $\mathrm{add}(o_a) = \{f_x \mid x \in \mathrm{add}(a)\} \setminus \mathrm{pre}(o_a)$, and $\mathrm{del}(o_a) = \{f_x \mid x \in \mathrm{del}(a)\} \setminus \{f_x \mid x \in \mathrm{add}(a)\}$.

The full grounding is not what would be used as a grounded representation of a PDDL task in practice. However, we will use it as a tool to prove that a certain lifted structure, if grounded, is a state invariant in the full grounding and therefore it is also a state invariant in the grounded representation obtained by a more constrained grounding. In the following, we formally define mutex groups and fact-alternating mutex groups (fam-groups), and we show (Theorem 6) that there is no point in looking for fam-groups that are disjoint with the initial state, because these facts can be easily pruned with the relaxed reachability. Therefore, it is reasonable for all algorithms for the inference of fam-groups to search only for the fam-groups that cover exactly one fact from the initial state (instead of at most one).

Definition 5. Given a STRIPS planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, a set of facts $M \subseteq \mathcal{F}$ is a **mutex group** if $|M \cap s| \leq 1$ for every state s reachable from s_I, and $M \subseteq \mathcal{F}$ is a **fact-alternating mutex group** (fam-group) if $|M \cap s_I| \leq 1$ and $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ for every operator $o \in \mathcal{O}$.

Theorem 6. *Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a STRIPS planning task. If $M \subseteq \mathcal{F}$ is a fam-group such that $M \cap s_I = \emptyset$, then M is not relaxed reachable in Π.*

Proof. (By contradiction) If M is relaxed reachable in Π, then there exist two states s, s' and an operator $o \in \mathcal{O}$ such that s is relaxed reachable in Π, and $s \cap M = \emptyset$, and $\mathrm{pre}(o) \subseteq s$, and $s' = s \cup \mathrm{add}(o)$, and $s' \cap M \neq \emptyset$. Since

$|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ and $\mathrm{pre}(o) \cap M \subseteq s \cap M = \emptyset$, we can conclude that $s' \cap M = \emptyset$. □

Lifted Mutex Groups

A lifted mutex group is a structure defined on the lifted (PDDL) level, that generates mutex groups through the grounding process. When describing the translation from PDDL to FDR, Helmert (2009) proposed an algorithm for the inference of lifted mutex groups that are used for a construction of FDR variables after grounding. In this section, we formalize lifted mutex groups and we show that the lifted mutex groups proposed by Helmert (2009) are in fact lifted fam-groups, i.e., when they are grounded they always form fam-groups in STRIPS.

We start with introducing an *invariant candidate* and *invariant grounding* that together provide a way to generate sets of facts in the corresponding ground (STRIPS) representation. Then we say that the invariant candidate is a lifted mutex group if all generated sets of facts are mutex groups. Finally, we provide a way to prove on the lifted level that an invariant candidate is a lifted fam-group.

Definition 7. An **invariant candidate** is a tuple $\nu = \langle \mathcal{V}^{\mathrm{fix}}[\nu], \mathcal{V}^{\mathrm{cnt}}[\nu], \mathrm{atoms}(\nu) \rangle$, where $\mathcal{V}^{\mathrm{fix}}[\nu] \subset \mathcal{V}$ is a finite set of **fixed variables** and $\mathcal{V}^{\mathrm{cnt}}[\nu] \subset \mathcal{V}$ is a finite set of **counted variables** such that $\mathcal{V}^{\mathrm{fix}}[\nu] \cap \mathcal{V}^{\mathrm{cnt}}[\nu] = \emptyset$, and $\mathrm{atoms}(\nu)$ is a finite set of atoms such that $\mathcal{V}[\mathrm{atoms}(\nu)] = \mathcal{V}^{\mathrm{fix}}[\nu] \cup \mathcal{V}^{\mathrm{cnt}}[\nu]$.

Definition 8. An **invariant grounding** is a tuple $\xi = \langle \gamma, G \rangle$, where $\gamma \in \mathcal{G}$ is a grounding and $G \subseteq \mathcal{G}$ is a set of groundings. For an invariant candidate ν, we define a set of all invariant groundings $\mathcal{H}_\nu = \{\langle \gamma, \mathcal{G}_{\mathcal{V}^{\mathrm{cnt}}[\nu]} \rangle \mid \gamma \in \mathcal{G}_{\mathcal{V}^{\mathrm{fix}}[\nu]}\}$. For an invariant candidate ν and a corresponding invariant grounding $\xi = \langle \gamma, G \rangle \in \mathcal{H}_\nu$, we define $\xi[\![\nu]\!] = \gamma[\![G[\![\mathrm{atoms}(\nu)]\!]]\!]$.

Intuitively, replacing the fixed variables with different combinations of objects generates different sets of ground atoms, whereas replacing the counted variables generates the ground atoms within each set.

For example, let $\mathrm{at}(v_1\colon \mathrm{vehicle}, c_1\colon \mathrm{location})$ describe an invariant candidate ν consisting of a single atom of a predicate at with arity 2, where the first argument, v_1, is a fixed variable with the type $\mathrm{vehicle}$ and the second argument, c_1, is a counted variable with the type $\mathrm{location}$. If we have two objects, t_1 and t_2, of the type $\mathrm{vehicle}$ and two objects, loc_1 and loc_2, of the type $\mathrm{location}$, then applying all invariant groundings from $\mathcal{H}_\nu$ generates the following two sets of ground atoms, $\{\mathrm{at}(\mathrm{t}_1, \mathrm{loc}_1), \mathrm{at}(\mathrm{t}_1, \mathrm{loc}_2)\}$ and $\{\mathrm{at}(\mathrm{t}_2, \mathrm{loc}_1), \mathrm{at}(\mathrm{t}_2, \mathrm{loc}_2)\}$.

For the invariant candidate $\mathrm{at}(c_1\colon \mathrm{vehicle}, c_2\colon \mathrm{location})$ with both variables counted and the same objects, the invariant groundings generate a single set of ground atoms, $\{\mathrm{at}(\mathrm{t}_1, \mathrm{loc}_1), \mathrm{at}(\mathrm{t}_1, \mathrm{loc}_2), \mathrm{at}(\mathrm{t}_2, \mathrm{loc}_1), \mathrm{at}(\mathrm{t}_2, \mathrm{loc}_2)\}$.

Definition 9. Let $\Pi^{\mathrm{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote the full grounding of $\mathcal{P}$. An invariant candidate ν is a **lifted mutex group** (**lifted fam-group**) if for every invariant grounding $\xi \in \mathcal{H}_\nu$ it holds that $M = \{f_x \mid x \in \xi[\![\nu]\!]\} \cap \mathcal{F}$ is a mutex group (fam-group) in $\Pi^{\mathrm{full}}_{\mathcal{P}}$.

Now we have defined lifted mutex groups and we know that if we find them on the lifted level, we can ground them and use them on the ground (STRIPS) level as mutex groups.

In the following, we borrow the notions of *balance* and *weight* from Helmert (2009) to formulate sufficient conditions for an invariant candidate to be a lifted fam-group.

Definition 10. An invariant candidate ν is **balanced in** an action $a \in \mathcal{A}$ if for every invariant grounding $\xi \in \mathcal{H}_\nu$ and every grounding $\gamma \in \mathcal{G}_a$, it holds that for every $\alpha \in \xi[\![\nu]\!] \cap \mathrm{add}(\gamma[\![a]\!])$ there exists $\alpha' \in \xi[\![\nu]\!] \cap \mathrm{pre}(\gamma[\![a]\!]) \cap \mathrm{del}(\gamma[\![a]\!])$.

An invariant candidate ν is **balanced** if it is balanced in every action $a \in \mathcal{A}$.

Note that the notion of balance, as we use it, considers each add effect in isolation. That is, it may happen that two different ground atoms from the add effect can be both balanced by the same ground atom from the precondition and delete effect, and we would still call such invariant candidate balanced. That is why we use the notion of weight to limit the number of ground atoms that can appear in the add effect.

Definition 11. The **weight** of the invariant candidate ν is $\mathrm{weight}(\nu) = \max_{\xi \in \mathcal{H}_\nu, a \in \mathcal{A}, \gamma \in \mathcal{G}_a} |\mathrm{add}(\gamma[\![a]\!]) \cap \xi[\![\nu]\!]|$.

The **init-weight** of the invariant candidate ν is $\text{i-weight}(\nu) = \max_{\xi \in \mathcal{H}_\nu} |\psi_I \cap \xi[\![\nu]\!]|$.

The weight of an invariant candidate is the maximum number of ground atoms of the invariant candidate that can appear in the add effect of any ground action (and similarly for the initial state). The invariant candidate is balanced in an action, if having a ground atom in action's add effect implies having another ground atom in its precondition and delete effect. Therefore, if the weight is at most one and the invariant candidate is balanced, then no ground action can increase the number of ground atoms in a state. And if we combine this with the condition that at most one ground atom is present in the initial state, then we must conclude that the invariant candidate is a lifted fam-group (and therefore also a lifted mutex group).

Theorem 12. *Let ν denote an invariant candidate. If* $\text{i-weight}(\nu) \leq 1$ *and* $\mathrm{weight}(\nu) \leq 1$ *and ν is balanced, then ν is a lifted fam-group.*

Proof. Let $\Pi^{\mathrm{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote the full grounding of $\mathcal{P}$, and let $\mathcal{M} = \{X_\xi \mid \xi \in \mathcal{H}_\nu\}$, where $X_\xi = \{f_x \mid x \in \xi[\![\nu]\!]\}$. $|M \cap s_I| \leq 1$ for every $M \in \mathcal{M}$ follows directly from $\text{i-weight}(\nu) \leq 1$. From $\mathrm{weight}(\nu) \leq 1$ it follows that $|M \cap \mathrm{add}(o)| \leq 1$ for every $M \in \mathcal{M}$ and every $o \in \mathcal{O}$. Since ν is balanced, then for every operator $o \in \mathcal{O}$ and every $M \in \mathcal{M}$ such that $|M \cap \mathrm{add}(o)| = 1$ it holds that $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \geq 1$, therefore $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ for every $M \in \mathcal{M}$ and every $o \in \mathcal{O}$. □

Note that the implementation of the tests for the weight and balance do not require to iterate over all possible groundings. The tests can run in polynomial time, because we can look for the renaming of the variables of actions and the invariant candidate. In the case of the weight test, we must iterate (in the worst case) over all pairs of atoms in all

(lifted) add effects. And in the case of the balance test, we need to test the combination of every add effect with every precondition that is also a delete effect.

The details are described by Helmert (2009). In fact, we use the same reasoning as Helmert: What he calls a *monotonicity invariant* corresponds here to the invariant candidate ν that is balanced and with $\text{weight}(\nu) \leq 1$, i.e., the invariant that does not increase the number of atoms in a state. Helmert's inference algorithm first looks for monotonicity invariants without considering the initial state, and only after the problem is grounded, the monotonicity invariants are grounded and checked against the initial state to form mutex groups.

We improved upon Helmert's findings by showing that this kind of invariant is not only a (lifted) mutex group, but specifically a (lifted) fam-group. This means that, as we show in the next section, we can use the lifted fam-groups during the grounding process for removing operators that are either unreachable, or that can generate only dead-end states. Moreover, it also means that this kind of invariants always generates a subset of mutexes obtainable from the h^2 heuristic (Fišer and Komenda 2018).

Pruned Grounding

In this section, we move from the full grounding of PDDL tasks to the grounding that uses relaxed reachability and utilizes pruning of operators that are either unreachable or can lead only to dead-end states (dead-end operators).

We start with the definition of a *relaxed grounding* as a grounding where we keep only relaxed reachable operators and facts, and we extend this notion with a pruning of operators using a *pruning function* that maps the ground actions to 1 if they are to be removed (or skipped during grounding), and to 0 otherwise.

Definition 13. Given a normalized PDDL task $\Pi = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$ and a **pruning function** $\omega : \mathcal{G}[\![\mathcal{A}]\!] \mapsto \{0, 1\}$, the **relaxed grounding** of Π **pruned with** ω is a STRIPS planning task $\Pi^{\text{relax}}_{\mathcal{P},\omega} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ constructed as follows.

Let $L_0 = \psi_I$, $G_0 = \emptyset$, $A_0 = \emptyset$, and for every $i \geq 1$ let $G_i = \bigcup_{a \in \mathcal{A}} G_{i,a}$ denote a set of groundings such that $G_{i,a} = \{\gamma \mid \gamma \in \mathcal{G}_a, \text{pre}(\gamma[\![a]\!]) \subseteq L_{i-1}, \omega(\gamma[\![a]\!]) = 0\}$, and let $A_i = \bigcup_{a \in \mathcal{A}} G_{i,a}[\![a]\!]$ and let $L_i = L_{i-1} \cup \bigcup_{a \in A_i} \text{add}(a)$. Finally let $k \geq 0$ denote the smallest number such that $L_k = L_{k+1}$. Then $\mathcal{F} := \{f_x \mid x \in L_k \cup \psi_G\}$, $s_I := \{f_x \mid x \in \psi_I\}$, $s_G := \{f_x \mid x \in \psi_G\}$, and $\mathcal{O} := \{o_a \mid a \in A_k\}$ with $\text{pre}(o_a) = \{f_x \mid x \in \text{pre}(a)\}$, $\text{add}(o_a) = \{f_x \mid x \in \text{add}(a)\} \setminus \text{pre}(o_a)$, and $\text{del}(o_a) = \{f_x \mid x \in \text{del}(a) \cap L_k\} \setminus \{f_x \mid x \in \text{add}(a)\}$.

Since we want to use lifted fam-groups for finding out which operators can be pruned, we need to make sure that (pruned) relaxed groundings preserve lifted fam-groups. If we remove a set of operators and unreachable facts, then all mutex groups will be preserved, because it can only make less states reachable. However, fam-groups are not defined over the reachable state space, but over the input planning task, so we need to make sure that the conditions from Definition 5 hold.

Removing operators preserves also fam-groups, because the second condition in Definition 5 must hold for all operators, therefore it must also hold for the operators remaining after the removal. Removing unreachable facts can change the operators, but only their delete effects because removing unreachable facts from a precondition would mean that the corresponding operator is also unreachable. Therefore, $\text{pre}(o) \cap \text{del}(o)$ remains the same for all operators o, which is enough to show that fam-groups are preserved in any relaxed grounding pruned with any pruning function.

Theorem 14. *Let $\mathcal{P}$ denote a PDDL task, let ν denote a lifted fam-group, let ω denote a pruning function, and let $\Pi^{\text{relax}}_{\mathcal{P},\omega} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote the relaxed grounding of $\mathcal{P}$ pruned with ω. Then for every invariant grounding $\xi \in \mathcal{H}_\nu$ it holds that $M = \{f_x \mid x \in \xi[\![\nu]\!]\} \cap \mathcal{F}$ is a fam-group in $\Pi^{\text{relax}}_{\mathcal{P},\omega}$.*

Proof. Let $\Pi^{\text{full}}_{\mathcal{P}} = \langle \mathcal{F}', \mathcal{O}', s_I, s_G \rangle$ denote the full grounding of $\mathcal{P}$ and let $M' = \{f_x \mid x \in \xi[\![\nu]\!]\} \cap \mathcal{F}'$ denote the corresponding fam-group in $\Pi^{\text{full}}_{\mathcal{P}}$. Clearly $\mathcal{F} \subseteq \mathcal{F}'$, and $M \subseteq M'$, and every $f \in \mathcal{F}$ is relaxed reachable in $\Pi^{\text{relax}}_{\mathcal{P},\omega}$ or $f \in s_G$, and for every operator $o \in \mathcal{O}$ it holds that o is relaxed reachable in $\Pi^{\text{relax}}_{\mathcal{P},\omega}$, and therefore there exists $o' \in \mathcal{O}'$ such that $\text{pre}(o) = \text{pre}(o')$, $\text{add}(o) = \text{add}(o')$, and $\text{del}(o) = \text{del}(o') \cap \mathcal{F}$. Therefore $M' \cap \text{add}(o') = M \cap \text{add}(o)$ and $M' \cap \text{pre}(o') = M \cap \text{pre}(o)$ and thus also $M' \cap \text{pre}(o') \cap \text{del}(o') = M \cap \text{pre}(o) \cap \text{del}(o)$ because $\text{pre}(o) = \text{pre}(o') \subseteq \mathcal{F}$. Therefore $|M \cap \text{add}(o)| \leq |M \cap \text{pre}(o) \cap \text{del}(o)|$ holds. Finally, since both $\Pi^{\text{full}}_{\mathcal{P}}$ and $\Pi^{\text{relax}}_{\mathcal{P},\omega}$ have the same initial state and $M \subseteq M'$, then $|M \cap s_I| \leq 1$ holds. □

Now we introduce a novel pruning technique on the lifted level that uses lifted fam-groups to remove unreachable and dead-end operators during grounding, i.e., before the (relaxed or full) grounding is explicitly constructed. In Definition 16, we define a pruning function that uses lifted fam-groups (1.) to prune unreachable operators and (2.) to prune dead-end operators (Proposition 15). In Theorem 17, we show that with this pruning function, all plans are preserved.

Proposition 15. *(Fišer and Komenda 2018, Corollary 8) Let $M \subseteq \mathcal{F}$ denote a set of facts, let s denote a state and let $o \in \mathcal{O}$ denote an operator applicable in s. If M is a fam-group and $|M \cap s_G| \geq 1$ and $|M \cap \text{pre}(o) \cap \text{del}(o)| \geq 1$ and $|M \cap \text{add}(o)| = 0$, then $o[s]$ is a dead-end state.*

Definition 16. Given a set of lifted fam-groups L, the **fam-group pruning function** ω_L is a pruning function such that for every $a \in \mathcal{G}[\![\mathcal{A}]\!]$:

1. $\omega_L(a) = 1$ if there exist $\nu \in L$ and $\xi \in \mathcal{H}_\nu$ such that $|\text{pre}(a) \cap \xi[\![\nu]\!]| \geq 2$;
2. $\omega_L(a) = 1$ if there exist $\nu \in L$ and $\xi \in \mathcal{H}_\nu$ such that $|\text{del}(a) \cap \text{pre}(a) \cap \xi[\![\nu]\!]| = 1$ and $|\text{add}(a) \cap \xi[\![\nu]\!]| = 0$ and $|\psi_G \cap \xi[\![\nu]\!]| \geq 1$;
3. $\omega_L(a) = 0$ otherwise.

Theorem 17. *Let $\mathcal{P}$ denote a PDDL task, let $\Pi^{\text{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote the full grounding of $\mathcal{P}$, let L denote a set of lifted fam-groups, and let $\Pi^{\text{relax}}_{\mathcal{P},\omega_L} = \langle \mathcal{F}', \mathcal{O}', s_I, s_G \rangle$*

denote the relaxed grounding of $\mathcal{P}$ pruned with the fam-group pruning function ω_L. And for a given sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$, let $\pi|_{\mathcal{F}'} = \langle o'_1, \ldots, o'_n \rangle$, where $\mathrm{pre}(o'_i) = \mathrm{pre}(o_i)$, $\mathrm{add}(o'_i) = \mathrm{add}(o_i)$, and $\mathrm{del}(o'_i) = \mathrm{del}(o_i) \cap \mathcal{F}'$ for every $i \in \{1, \ldots, n\}$. If π is a plan in Π^{full}, then $\pi|_{\mathcal{F}'}$ is a plan in $\Pi^{\mathrm{relax}}_{\mathcal{P},\omega_L}$. And if π' is a plan in $\Pi^{\mathrm{relax}}_{\mathcal{P},\omega_L}$, then there exists a plan π in Π^{full} such that $\pi|_{\mathcal{F}'} = \pi'$.

Proof Sketch. In Definition 16, 1. avoids grounding of unreachable operators, because they would be applicable only in states violating one of the fam-groups, i.e., in the unreachable states, and 2. avoids grounding of operators that can produce only dead-end states (Proposition 15 and Theorem 14), therefore they cannot be part of any plan. The rest of the reachable operators are the same in Π^{full} and $\Pi^{\mathrm{relax}}_{\mathcal{P},\omega_L}$ except that operators in $\Pi^{\mathrm{relax}}_{\mathcal{P},\omega_L}$ does not have relaxed unreachable facts in their delete effects. □

Inference of Lifted FAM-Groups

In this section, we introduce an extension of the algorithm proposed by Helmert (2009). In a nutshell, Helmert's algorithm is a "guess, check, and repair" algorithm that maintains a set of invariant candidates and, in each cycle, one candidate is tested against all actions. If the candidate is balanced in all actions and the weight is at most 1, then it is proved to be a monotonicity invariant and, after grounding, checked against the initial state whether it forms a mutex group. If the candidate has the weight larger than 1, then the candidate is thrown away. And finally, if the candidate is not balanced, it is refined in such a way the balance test passes, and the refined candidate is put back into the set of candidates.

The initial candidates are created from all predicates. For each predicate p, $\mathrm{ar}(p) + 1$ candidates are created so that one candidate has all variables fixed and the remaining $\mathrm{ar}(p)$ candidates have one of the arguments set to a counted variable and the rest to fixed variables.

The refinement of the candidate is done by taking the first action a in which the balance test fails and the candidate is extended with an atom that covers one of the atoms in the intersection of the precondition and delete effect so that the add effect is balanced by the refined candidate. The newly added atom is always of a predicate that is not yet part of the candidate, and the atom can contain at most one counted variable.

We improve this algorithm by the following. (i) We allow any number of counted variables in all atoms. (ii) We introduce new refinement techniques that allow us to refine also the candidates that fail weight test: the refinement of types, and the refinement of counted variables. (iii) We introduce a new refinement technique for the proved lifted mutex groups so that the algorithm does not stop once an invariant candidate is proved to be a lifted fam-group, but it allows to construct supersets of the proved lifted fam-groups. Another difference is that we infer directly lifted fam-groups instead of proving monotonicity invariants first. The reason is purely practical. Allowing any number of counted variables can generate a huge number of candidates, but restricting the candidates to those that has init-weight exactly one reduces the number significantly.

Algorithm 1: Inference algorithm.

Input: A PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$
Output: A set of lifted fam-groups M

$C \leftarrow \{\langle \emptyset, \{c_1, \ldots, c_{\mathrm{ar}(p)}\}, \{p(c_1, \ldots, c_{\mathrm{ar}(p)}\}\rangle \mid p \in \mathcal{P}\}$;
$M \leftarrow \{\}$;
while $|C| > 0$ **do**
  $\nu \leftarrow$ `Pop(`C`)`;
  if i-weight$(\nu) = 1$ **then**
    if weight$(\nu) \leq 1$ **then**
      if ν *is balanced* **then**
        $M \leftarrow M \cup \{\nu\}$;
        $C \leftarrow C \cup$ `RefineProved(`ν`)`;
      else $C \leftarrow C \cup$ `RefineUnbalanced(`ν`)` ;
    else $C \leftarrow C \cup$ `RefineHeavyAction(`ν`)` ;
  else $C \leftarrow C \cup$ `RefineHeavy(`ν, ψ_I`)` ;
return M;
function `RefineProved(`ν`)`
  $C \leftarrow \emptyset$;
  for each $a \in \mathcal{A}$ *s.t. both* $\mathrm{pre}(a)$ *and* $\mathrm{del}(a)$ *can be unified with some* $\alpha \in \mathrm{atoms}(\nu)$*, but* $\mathrm{add}(a)$ *cannot be unified with* $\mathrm{atoms}(\nu)$ **do**
    $C \leftarrow C \cup$ `RefineExtend(`ν, $\mathrm{add}(a)$`)`;
  return C;
function `RefineUnbalanced(`ν`)`
  Let a be an action s.t. ν is not balanced in a;
  Let $\beta \in \mathrm{add}(a)$ be the atom that can be unified with the corresponding atom α from ν;
  $C \leftarrow$ Refine types of variables $\mathcal{V}[\alpha]$ so that β cannot be unified with the refined α;
  return $C \cup$ `RefineExtend(`ν, $\mathrm{del}(a)$`)` ;
function `RefineExtend(`ν, X`)`
  $C \leftarrow \emptyset$;
  for each $\delta \in X$ *s.t.* $\mathcal{P}[\delta] \notin \mathcal{P}[\mathrm{atoms}(\nu)]$ **do**
    $C \leftarrow C \cup$ Refine ν by extending it with an atom α of the predicate $\mathcal{P}[\delta]$ s.t. δ can be unified with α;
  return C;
function `RefineHeavyAction(`ν`)`
  Let $a \in \mathcal{A}$ be an action failing the weight test;
  return `RefineHeavy(`ν, $\mathrm{add}(a)$`)` ;
function `RefineHeavy(`ν, X`)`
  Let $\beta_1, \beta_2, \beta_1 \neq \beta_2$, be two atoms from X that can be unified with the corresponding atoms α_1, α_2 from ν;
  $C \leftarrow$ Refine types of variables $\mathcal{V}[\alpha_1]$ ($\mathcal{V}[\alpha_2]$) so that β_1 (β_2) cannot be unified with the refined α_1 (α_2);
  if $\alpha_1 = \alpha_2$ **then**
    $C \leftarrow C \cup$ Refine counted variables from $\mathcal{V}[\alpha_1]$ so that $\{\beta_1, \beta_2\}$ cannot be unified with the refined α_1;
  return C;

The main part of the algorithm is described as a pseudocode in Algorithm 1. The remaining parts are described in the text below. As already mentioned, testing whether there exist groundings so that action atoms can be ground to the same atoms as invariant candidate's atoms can be done in a polynomial number of steps by renaming variables of the atoms. If there exists such a renaming, we say that these atoms can be unified.

Initial Invariant Candidates For each predicate $p \in$

$\mathcal{P}$, one invariant candidate consisting of a single atom $p(c_1, \ldots, c_{\mathrm{ar}(p)})$ with all variables being counted variables is created. Such candidates almost always fail the weight test, but we have the refinement of the counted variables, descried below, that allows to change the candidate in such a way, that the weight test is passed in the following cycles.

Refinement by Extension This type of refinement is the same as is used by Helmert for recovering from a failed balance test, but we use it also for the refinement of the proved lifted fam-groups. When we prove a lifted fam-group ν, the algorithm looks for actions that cannot add any atom from ν, but it deletes something from ν. For these actions, we try to extend ν with atoms from the add effect, because adding such atoms cannot violate balance in these actions.

Refinement of Variable Types Consider the atom $p(v_1\colon \mathrm{t}_1)$ with variable v_1 of type t_1 and the atom $p(v_2\colon \mathrm{t}_2)$ with variable v_2 of type t_2 that is a sub-type of t_1, i.e., $\mathcal{D}(\mathrm{t}_2) \subset \mathcal{D}(\mathrm{t}_1)$. Clearly, $p(v_1\colon \mathrm{t}_1)$ can be unified with $p(v_2\colon \mathrm{t}_2)$ because t_2 is a sub-type of t_1. However, if we change the type t_1 to some other sub-type t_3 such that $\mathcal{D}(\mathrm{t}_3) \subset \mathcal{D}(\mathrm{t}_1)$ and $\mathcal{D}(\mathrm{t}_3) \cap \mathcal{D}(\mathrm{t}_2) = \emptyset$, then $p(v_1\colon \mathrm{t}_3)$ can no longer be unified with $p(v_2\colon \mathrm{t}_2)$ and the change is valid in the sense that the predicate p must accept variables (objects) of type t_3.

This idea of type refinement is used by our inference algorithm to fix the weight test. If any candidate's atom can be unified with some atom from an add effect or the initial state, and the candidate's atom has, in some argument, more general type than the other atom, then we can decrease the weight by applying the change of types described above.

Refinement of Counted Variables This refinement simply changes counted variables into fixed variables. If the weight test fails because two atoms from some add effect (or the initial state) are covered by a single atom from the invariant candidate because of a counted variable, then changing the counted variable to the fixed variable fixes the weight test. For example, consider an invariant candidate with a single atom $p(v_1, c_1)$ with a fixed variable v_1 and a counted variable c_1, and the add effect $\{p(w_1, w_2), p(w_1, w_3)\}$. Assuming all variables have the same type, the invariant candidate can be unified with both atoms from the add effect, because the invariant grounding can expand the counted variable c_1 into two different objects covering both w_2 and w_3. Changing c_1 to a fixed variable, however, prevents it, thus the weight test passes.

Experimental Results

The grounding of PDDL tasks and the inference of lifted fam-groups was implemented[1] in C and experimentally evaluated on a cluster of computing nodes with Intel Xeon Scalable Gold 6146 processors. We implemented both Helmert's original algorithm, referred to as `H`, and our improved algorithm, referred to as `H+`. Both `H` and `H+` ran with the limit on the number of considered invariant candidates set to 10 000. We used domains from all IPCs from 2006 to 2018.

We compared the inferred (ground) fam-groups in terms of the mutex group cover number, i.e., the minimum number of mutex groups needed to cover all facts, and the number of fam-groups that were not found by other methods. Since Fišer and Komenda (2018) introduced a complete algorithm for inference of maximal fam-groups (based on repeated solving of Integer Linear Program), we compare also to this method, denoted by `F`, because it shows how close we are to the best possible results.

domain	#ps	avg. cover number			num. mutex groups			
		H	H+	F	H ≻ H+	H+ ≻ H	F ≻ H	F ≻ H+
agricola	40	0.70	**0.47**	**0.35**	0	40	627	587
barman	74	0.88	**0.19**	0.19	0	1 640	1 640	0
cavediving	40	0.73	0.73	**0.22**	0	40	1 272	1 232
cybersec	30	0.94	**0.72**	**0.39**	26	162 528	1 566	1 372
maintenance	25	1.00	1.00	**0.99**	0	0	6	6
nomystery	40	0.08	**0.05**	0.05	0	40	40	0
organic-synthesis	9	1.00	**0.83**	**0.62**	0	47	1 415	1 368
parcprinter	70	0.61	0.61	**0.24**	0	0	4 034	4 034
pegsol	70	0.34	0.34	0.34	0	0	223	223
rovers	40	0.43	0.43	0.43	0	0	2	2
scanalyzer	70	0.18	0.18	0.18	0	0	172	172
snake	19	0.74	0.74	**0.34**	0	0	1 553	1 553
sokoban	100	0.20	0.20	0.20	0	0	8	8
spider	26	0.43	0.43	**0.21**	0	818	2 151	1 333
tetris	32	0.90	**0.04**	**0.03**	0	136	1 400	1 264
thoughtful	20	0.55	0.55	**0.42**	0	0	1 023	1 023
tidybot	60	0.98	**0.63**	**0.63**	0	429	656	227
tpp	25	0.30	0.30	0.30	0	0	237	237
transport	140	0.25	**0.05**	0.05	0	432	432	0
trucks	12	0.86	**0.06**	0.06	0	180	1 092	912
woodworking	100	0.58	0.58	**0.57**	2	1 151	1 906	757
caldera	40	0.83	0.83	-	0	920	-	-
citycar	40	0.71	**0.09**	-	0	201	-	-
flashfill	20	0.21	**0.11**	-	0	61 964	-	-
settlers	40	0.96	**0.95**	-	0	140	-	-
overall w/o CE	1777	0.42	**0.34**	**0.29**	28	167 481	21 455	16 310
overall with CE	1957	0.44	**0.35**	-	28	230 706	-	-

Table 1: Left: the comparison of average C/F (the smaller is the better), where C is the mutex group cover number and F is the number of facts. Right: for every $A \succ B$, the number of fam-groups found by A that are not a subset of any fam-group found by B. "overall w/o CE" counts only problems without conditional effects, and "overall with CE" counts all problems: averages over all counted problems on left and sums on right. `H` is Helmert's algorithm; `H+` is our improvement; `F` is a complete alg. for all maximal fam-groups.

Table 1 shows results for problems where the computation of fam-groups by all methods finished within 5 minutes time and 8 GB memory limit and also the mutex group cover number was computed within 1 hour. To deal with different sizes of problems, mutex group cover numbers are divided by the number of facts for each problem and averaged within each domain. The domains with conditional effects (under the first horizontal line) do not contain results for `F`, because it is not clear how to find a complete set of fam-groups without compiling conditional effects away. Only domains with some difference are shown. Since `F` must always dominate both `H` and `H+`, the numbers in the `F` column are in bold if they are strictly better than the other two, and the numbers for `H+` are in bold if they are strictly better than `H`.

The improvement by `H+` in agricola, tetris, tidybot, and citycar was due to multiple counted variables used in a single atom. The type refinement helped in nomystery, organic-synthesis, and transport, where some predicates formed mutex groups only for certain sub-types of their arguments. In barman, trucks, settlers, and woodworking, the combination of the type refinement and the extension of proved fam-groups provided a richer set of mutex groups. In 13 domains both `H` and `H+` found the complete set of fam-groups, and in

[1] https://gitlab.com/danfis/cpddl.git, branch aaai20

domain	#ps	mutex		+ dead-end		+ h^2 fw/bw	
		H	H+	H	H+	H	H+
agricola	20	0.00	0.00	0.00	0.00	64.99	**65.13**
barman*	74	15.42	15.42	15.42	**45.95**	25.50	**57.83**
citycar*	40	0.00	0.00	1.61	1.61	1.61	1.61
flashfill	18	0.10	0.10	0.10	0.10	7.06	7.06
floortile*	70	0.00	0.00	22.79	22.79	35.43	35.43
organic-synthesis*	7	0.00	**11.44**	14.14	**23.11**	90.85	90.85
parcprinter*	30	0.00	0.00	40.83	40.83	70.00	70.00
parking	80	3.09	3.09	3.09	3.09	6.19	6.19
scanalyzer	30	1.34	1.34	1.34	1.34	28.12	28.12
spider	15	0.00	**3.47**	0.00	**3.47**	16.61	16.61
trucks*	30	0.00	0.00	0.00	**82.14**	19.09	**82.94**
woodworking*	30	0.00	0.00	10.08	10.08	51.41	51.41
overall from above	444	3.22	**3.52**	10.62	**21.52**	27.70	**37.40**
overall	1247	1.15	**1.25**	3.78	**7.66**	17.83	**21.28**

Table 2: The average percentage of removed operators, overall is the average over all problems. Column "mutex": lifted fam-groups used for detection of unreachable operators (their preconditions are mutex); "+dead-end": additional pruning of dead-end operators; "+h^2 fw/bw": additional pruning using h^2 in forward and backward with fam-groups used for the disambiguation. Domains in which dead-end detection pruned additional operators are starred.

domain	#ps	m&s			comp1			comp2		
		fd	H	H+	fd	H	H+	fd	H	H+
agricola	20	3	3	3	**9**	8	8	6	7	**9**
barman	34	11	11	11	12	11	**14**	11	12	**15**
caldera	20	12	12	12	11	13	**14**	12	13	**15**
citycar	20	14	**16**	**16**	10	15	**16**	13	**16**	**16**
settlers	20	9	9	9	8	**9**	8	9	9	9
tetris	17	11	11	11	11	11	**12**	13	13	13
tidybot	40	11	11	**31**	32	32	**38**	31	30	**39**
transport	70	24	24	24	**33**	29	**33**	**33**	29	**33**
trucks	30	9	**10**	**10**	**14**	13	13	10	12	**13**
Σ from above	271	104	107	**127**	140	141	**156**	138	141	**162**
childsnack	20	0	0	0	0	0	0	1	**2**	**2**
data-network	20	12	12	12	13	**14**	**14**	**13**	12	12
nurikabe	20	12	12	12	**12**	11	11	**11**	10	10
organic-synth	20	7	**10**	**10**	7	**10**	**10**	7	**10**	**10**
parcprinter	50	41	41	41	43	43	43	41	**43**	**43**
petri-net-align	20	7	7	7	19	**20**	**20**	19	19	19
rovers	40	7	7	7	**14**	13	13	13	13	13
spider	20	6	6	6	**12**	11	11	11	**12**	**12**
storage	30	15	15	15	**16**	15	15	15	15	15
tpp	30	**7**	6	6	12	**13**	**13**	15	15	15
woodworking	50	31	31	31	**47**	46	46	47	**48**	**48**
Σ	1 166	593	598	**618**	725	727	**742**	745	754	**775**

Table 3: The number of solved problems in the optimal track for selected planners. fd: FD with the original translator. The top part lists domains where H and H+ differ.

barman, nomystery, and transport only H+ found a complete set of fam-groups.

H found some fam-groups not found by H+ in cybersec because of the limit on the number of invariant candidates, and in woodworking because of the init-weight restriction we use in H+. If we did not set the limit on the number of invariant candidates and we created one auxiliary type per each object, H+ would dominate H in all domains, but the time spent in the inference would significantly increase. Note that the numbers of fam-groups found by one method and not the other may sometimes be misleading (especially for cybersec and flashfill), because there may be many overlapping fam-groups. So for example, finding fam-groups $\{A, B\}$, $\{B, C\}$ is reported as two fam-groups even if there is a maximal fam-group $\{A, B, C\}$ not found by that method.

The time spent in the inference was under a millisecond in 1490 out of 1957 tested problems for H+ (1640 for H), between one millisecond and one second in 418 (284) problems, and more than a second for the remaining 49 (33) problems from the caldera, cybersec, and flashfill domains. F was orders of magnitude slower (as previously reported by Fišer and Komenda (2018)).

The limit on the number of candidates (10 000) was reached only in the cybersec domain for H, and in the caldera, cybersec, flashfill, and organic-synthesis domains for H+. So, increasing the limit could provide more fam-groups only in these domains, but it would also require more time to process all candidates.

As Table 1 shows (non-zero values in the F $\succ$ H+ column), H+ is not complete with respect to all maximal (ground) fam-groups. The reason is that the algorithm allows at most one atom of each predicate in lifted fam-groups, and all variables are restricted to the types defined in the input PDDL. So, for example, suppose we have three objects l_1, l_2, l_3 of a type loc, there is no sub-type of loc, and the corresponding ground problem has a maximal (ground) fam-group $\{\mathrm{at}(l_1), \mathrm{at}(l_2)\}$. This fam-group cannot be found by H+, because the invariant candidate $\mathrm{at}(c\colon \mathrm{loc})$ (with a counted variable c) would correspond to $\{\mathrm{at}(l_1), \mathrm{at}(l_2), \mathrm{at}(l_3)\}$. And even if there was a sub-type for each l_i, then the invariant candidate $\{\mathrm{at}(v_1\colon \mathrm{l}_1), \mathrm{at}(v_2\colon \mathrm{l}_2)\}$ (with fixed variables v_1, v_2) is not constructed by H+, because it contains two atoms of the same predicate.

Next, we compared the pruning power of H, H+ and h^2 heuristic in forward and backward direction (Alcázar and Torralba 2015) with fam-groups from H and H+ used for disambiguation. Table 2 shows the results for problems where all variants finished within the 5 minutes time limit. The table lists only the domains in which at least one operator was pruned during grounding or there was a difference between h^2 with H and with H+.

The pruning of dead-end operators during grounding had effect in 7 domains (difference between "mutex" and "+dead-end" columns). Overall, using H for pruned grounding removes more than 3.7% operators and using H+ about twice as that (7.6%). Moreover, inferring a richer set of fam-groups using H+ provides more pruning power even for h^2: about 3.5% more operators are removed.

Lastly, we measured a coverage with the Fast Downward planner (FD) (Helmert 2006), where we switched the original translator from PDDL to FDR with our implementation, used h^2 fw/bw preprocessor, and set the time limit to 30 minutes and the memory limit to 8 GB. For the satisficing track, we evaluated LAMA-11 (Richter and Westphal 2010) and FF (Hoffmann and Nebel 2001) planners, but we did not find a significant difference between H and H+. LAMA-11 with H+ solved two more problems overall, FF with H solved one more problem overall, and the most notable difference was that FF with H+ solved 5 more problems in citycar.

For the optimal track, we used A* with the LM-Cut (lmc) heuristic (Helmert and Domshlak 2009), the merge-and-shrink (m&s) heuristic with SCC-DFP merge strategy and non-greedy bisimulation shrink strategy (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2016), the potential (pot) heuristic optimized for all syntactic states (Seipp, Pommerening, and Helmert 2015), and two non-portfolio win-

ners of the last IPC 2018, Complementary1 (comp1) (Franco et al. 2018), and Complementary2 (comp2) (Franco et al. 2017; Franco, Lelis, and Barley 2018). The only difference for lmc was that `H+` solved three more problems in barman and one less in transport. For pot, `H+` solved three less problems in spider and two more in tetris.

The most interesting results were found with m&s and comp1/2 planners shown in Table 3. The table also shows the results with the original translator from FD (`fd`) which implements `H`. However, there are some differences in the grounding process (e.g., handling of negative preconditions and conditional effects, deduplication of operators, ordering of the unification steps, ...), so we do not think `fd` is directly comparable to `H` and `H+`, because it does not measure just the difference between different sets of lifted fam-groups.

All these planners use some variant of abstraction heuristics, where we expected to see the most difference because they depend on the complexity and the number of inferred mutex groups. However, the implementation of m&s and pattern databases in FD uses FDR variables derived from fam-groups instead of fam-groups directly. We think this approach possibly disregards some useful information from the overlapping fam-groups that could improve the heuristic estimates.

Conclusion

Any translator from PDDL to FDR must, at some point, infer a set of mutex groups in order to create FDR variables. The most commonly used translator (Helmert 2009) infers mutex groups on a lifted (PDDL) level and then grounds them as it grounds the task into STRIPS. We proved that these lifted mutex groups are lifted fam-groups, i.e., they are a certain subclass of mutex groups previously described by Fišer and Komenda (2018).

Moreover, we showed how to use lifted fam-groups to reduce the number of operators during grounding by utilizing the ability of fam-groups to determine unreachable and dead-end operators. The experimental evaluation on IPC domains confirmed that operators are pruned in a sizable number of problems.

Finally, we proposed an extension of the Helmert's (2009) algorithm that produces a richer set of lifted fam-groups, which in turn increased the number of removed operators during grounding and the overall number of solved tasks for the heuristic search with abstraction heuristics.

Acknowledgements

The work was supported by the Czech Science Foundation (grant no. 18-24965Y). The experimental evaluation was supported by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

References

Abdulaziz, M.; Gretton, C.; and Norrish, M. 2017. A state-space acyclicity property for exponentially tighter plan length bounds. In *Proc. ICAPS'17*, 2–10.

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. ICAPS'15*, 2–6.

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In *Proc. IJCAI'13*, 2254–2260.

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

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Canadian Conference on AI*, volume 1081 of *Lecture Notes in Computer Science*, 402–416. Springer.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. ECP'01*, 13–24.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Fišer, D., and Komenda, A. 2018. Fact-alternating mutex groups for classical planning. *Journal of Artificial Intelligence Research* 61:475–521.

Franco, S.; Torralba, A.; Lelis, L. H.; and Barley, M. 2017. On creating complementary pattern databases. In *Proc. IJCAI'17*, 4302–4309.

Franco, S.; Lelis, L. H. S.; Barley, M.; Edelkamp, S.; Martinez, M.; and Moraru, I. 2018. The Complementary1 planner in IPC 2018. In *IPC 2018 planner abstracts*, 28–31.

Franco, S.; Lelis, L. H. S.; and Barley, M. 2018. The Complementary2 planner in IPC 2018. In *IPC 2018 planner abstracts*, 32–36.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS'09*, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3):16.1–16.63.

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.

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

McDermott, D. 2000. The 1998 AI planning systems competition. *The AI Magazine* 21(2):35–55.

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

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In *Proc. ICAPS'15*, 193–201.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proc. AAAI'14*, 2358–2366.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In *Proc. ICAPS'16*, 294–298.