Journal of Artificial Intelligence Research 61 (2018) 475-521 Submitted 09/16; published 03/18

Fact-Alternating Mutex Groups for Classical Planning

Daniel Fišer DANFIS@DANFIS.CZ
Antonín Komenda ANTONIN.KOMENDA@FEL.CVUT.CZ
Department of Computer Science,
Faculty of Electrical Engineering,
Czech Technical University in Prague,
Prague 6, Czech Republic 166 27

Abstract

Mutex groups are defined in the context of STRIPS planning as sets of facts out of which, maximally, one can be true in any state reachable from the initial state. The importance of computing and exploiting mutex groups was repeatedly pointed out in many studies. However, the theoretical analysis of mutex groups is sparse in current literature. This work provides a complexity analysis showing that inference of mutex groups is as hard as planning itself (PSPACE-Complete) and it also shows a tight relationship between mutex groups and graph cliques. This result motivates us to propose a new type of mutex group called a fact-alternating mutex group (fam-group) of which inference is NP-Complete. Moreover, we introduce an algorithm for the inference of fam-groups based on integer linear programming that is complete with respect to the maximal fam-groups and we demonstrate how beneficial fam-groups can be in the translation of planning tasks into finite domain representation. Finally, we show that fam-groups can be used for the detection of dead-end states and we propose a simple algorithm for the pruning of operators and facts as a preprocessing step that takes advantage of the properties of fam-groups. The experimental evaluation of the pruning algorithm shows a substantial increase in a number of solved tasks in domains from the optimal deterministic track of the last two planning competitions (IPC 2011 and 2014).

1. Introduction

State invariants in domain-independent planning are certain intrinsic properties of a particular planning task that hold in all states reachable from the initial state. State invariants (as well as other types of invariants) tell something about the internal structure of the problem. This revealed structure can be further utilized in the process of solving the task. State invariants can, for example, be used to design heuristic functions that can better guide search algorithms. They can be used to prune the search space within which a plan is searched for or even to reformulate the original problem to some more simple form as a preprocessing step.

A mutual exclusion (mutex) invariant states that certain facts cannot be true at the same time in any reachable state. This type of state invariant is especially interesting for the purpose of translating planning tasks to a finite-domain representation (Helmert, 2009) or for the construction of heuristic functions based on reachability analysis such as the h^m heuristics (Haslum & Geffner, 2000). In this work, we are particularly interested in the inference of state invariants called mutex groups consisting of facts that are pairwise mutually exclusive. Therefore, a mutex group states that any reachable state can contain at most one fact from the mutex group.

The most straightforward application of mutex groups is in the translation to finite domain representation (FDR, or SAS$^+$) (Bäckström & Nebel, 1995; Edelkamp & Helmert, 1999; Helmert, 2009). Given a set of mutex groups, the FDR can be constructed by creating variables from those mutex groups that cover all facts. A special value "none of those" can be added to some variables, if needed, to cover a situation where none of the facts from the invariant is present in the state. Heuristic functions based on domain transition graphs (DTG) (Helmert, 2006; Torreño, Onaindia, & Sapena, 2014) could also benefit from mutex groups. These heuristics can be constructed either from FDR or directly from the set of all inferred mutex groups.

State invariants (including mutex invariants) are also critical for improving the performance of SAT planners (Kautz & Selman, 1992; Sideris & Dimopoulos, 2010). SAT planners are based on a formulation of planning tasks as a problem of satisfiability of logical formulas. State invariants expressed as a logical formula often significantly prune the search space and, therefore, improve efficiency of the solvers.

Exploration of the state space in a symbolic search with Binary Decision Diagrams (BDDs) is not carried out through the expansion of single states but rather by construction of BDDs representing sets of states, which potentially provides an exponential saving in memory consumption. State invariants are useful in the symbolic search for two reasons. First, for the construction of FDR as the basis for smaller BDDs (Kissmann & Edelkamp, 2011). Second, state invariants encoded as BDDs can be used for the pruning of unreachable states during search and also during the preprocessing of the planning task for pruning operators that generate dead-end states (Torralba & Alcázar, 2013). The connection between state invariants and dead-end states was recently studied by Lipovetzky, Muise, and Geffner (2016).

This work is aimed mainly at the analysis and inference of mutex groups in the context of STRIPS planning (Fikes & Nilsson, 1971). We introduce a new type of mutex group called the fact-alternating mutex group and we discuss its relation to the general mutex group and to the mutexes inferred by the heuristic h^m (Haslum & Geffner, 2000). We also discuss, in detail, the properties of fact-alternating mutex groups, in particular their connection to dead-end states.

We provide a complexity analysis showing that the inference of the maximum sized mutex group is PSPACE-Complete whereas inferring the maximum sized fact-alternating mutex group is NP-Complete. The complexity analysis leads to a novel inference algorithm that is complete with respect to maximal fact-alternating mutex groups. The algorithm is based on a direct translation of the definition of fact-alternating mutex groups into the constraints of an integer linear program (ILP). The solution of the ILP provides only one invariant at a time, so the ILP is refined in a cycle in such a way that all invariants are eventually discovered.

The experimental evaluation of the inference algorithm includes the comparison to two state-of-the-art algorithms, h^m and Helmert's algorithm (2009) used in Fast Downward's translator from PDDL to finite domain representation. Moreover, we introduce an algorithm for the pruning of operators and facts as an example of the applicability of the fact-alternating mutex groups in solving planning tasks. We show that taking advantage of mutex invariants in a preprocessing stage (in particular the ability of the fact-alternating mutex groups to detect operators that can produce only dead-end states) can lead to a substantial increase in the coverage of the solved problems.

The paper is organized as follows. A list of a related work is laid out in Section 2 where different types of invariants, as well as different approaches to the inference of invariants, are discussed. After establishing a background for this work (Section 3), we formally define two types of mutex groups and we discuss their properties and relationships between each other (Section 4 and 5). The complexity analysis is provided in Section 6, followed by a description of the relationship between mutexes generated by $\mathrm{h^m}$ and fact-alternating mutex groups in Section 7. In Section 8, we describe a novel inference algorithm and we prove it is complete with respect to the maximal fact-alternating mutex groups. In Section 9, we propose a pruning algorithm utilizing the inferred fact-alternating mutex groups. The experimental results are discussed in Section 10 and we conclude with Section 11.

2. Related Work

State invariants are formulas that are true in every state of a planning task reachable from the initial state by the application of a sequence of operators. In this section, we provide a brief discussion of different approaches to the inference of state invariants related to the approach presented in this work.

One of the first approaches to the inference of state invariants was the DISCOPLAN system proposed by Gerevini and Schubert (1998, 2000). The algorithm uses a *guess, check, and repair* approach for generating invariants. Invariants are first hypothesized from the definitions of the operators. The consecutive steps involve verification that the invariants still hold in all reachable states and the unverified invariants are refined to form new invariants that are then in turn verified again. The refinements are based on sets of candidate supplementary conditions called “excuses” that are extracted during the verification phase. These “excuses” are extracted through analysis considering all operators. The analysis allows the algorithm to make more informed choices in the consequent refinement than the “excuses” that would be derived only from the first operator violating the invariant. However, this comes with an increased computational burden as noticed by Helmert (2009). The algorithm is able to generate a wide range of different types of state invariants (or state constraints as they are called by Gerevini and Schubert) such as implicative constraints of the form $\phi \Rightarrow \psi$ stating that every state satisfying formulae ϕ has to satisfy ψ also, static constraints providing type information about predicates, or *xor* constraints providing information about the mutual exclusion of two literals given some additional conditions.

Type Inference Module (TIM) proposed by Fox and Long (1998) and further extended by Cresswell, Fox, and Long (2002) takes the domain description possibly without any information about types and infers (or enriches) a type structure from the functional relationships in the domain. State invariants can be extracted from the way in which the inferred types are partitioned.

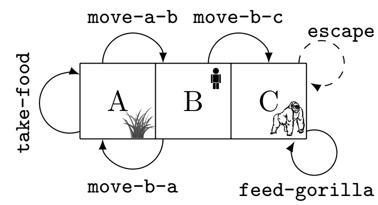
Rintanen (2000) proposed an iterative algorithm for generating state invariants. The algorithm uses a guess, check, and repair approach and it is polynomial in time due to restrictions on the form and length of the invariants. The procedure starts with the identification of the initial set of candidate invariants corresponding to the grounded facts in the initial state. In the following steps, the initial set of candidates is expanded with new invariants that are created by expanding invariant candidates from the previous step using grounded operators. The invariant candidates that do not preserve their invariant property are rejected and new candidates that are weaker in the sense that they hold in more states than the original ones are created. An interesting property of this algorithm is that it considers all invariant candidates during the creation of new ones instead of expanding one invariant at a time.

Mukherji and Schubert (2005, 2006) proposed a completely different approach. Instead of analyzing operators of the planning task, state invariants are inferred from one or more reachable states. The set of reachable states can be obtained by random walks through state space or by an exhaustive search with a bounded depth. State invariants are then inferred by an any-time algorithm employing a data analysis of the provided reachable states. The resulting invariants are not guaranteed to be correct in the sense that they do not have to hold in all reachable states besides those provided, but the authors suggest that some other algorithm, such that of Rintanen (2000), can be used for the quick verification of the correctness of the invariants produced.

A generalization of the $\mathrm{h}^{\mathrm{max}}$ heuristic to a family of h^{m} heuristics (Haslum & Geffner, 2000; Haslum, 2009; Alcázar & Torralba, 2015) offers another method for the generation of invariants. $\mathrm{h}^{\mathrm{max}}$ is a widely known and a well understood admissible heuristic for STRIPS planning. The heuristic value is computed on a relaxed reachability graph as a cost of the most costly fact from a conjunction of reachable facts. The heuristic works with single facts, but it can be generalized to consider a conjunction of at most m facts instead. h^1 would then be equal to $\mathrm{h}^{\mathrm{max}}$, h^2 would build the reachability graph with single facts and pairs of facts, h^3 would add triplets of facts also, and h^{m} would consider conjunctions of at most m facts. This heuristic is not bound by h^+ and is even equal to the optimal heuristic for sufficiently large m. Unfortunately, the cost of the computation increases exponentially in m.

The important property of h^{m} related to inference of invariants is its ability to provide a set of fact conjunctions that are not reachable from the initial state. The facts that do not appear in the reachability graph of h^1 ($\mathrm{h}^{\mathrm{max}}$) cannot affect the planning procedure. The same can be said about the unreachable conjunctions of m facts in the case of h^{m}. For example, an unreachable pair of facts in case of h^2 can be interpreted as an invariant stating that both facts from the pair cannot hold at the same time. Similarly, an unreachable triplet of facts in case of h^3 corresponds to an invariant stating that there is no reachable state that contains all three facts at the same time. Therefore, the h^{m} heuristic is able to find mutex invariants of a cardinality up to m.

The state invariants inferred by the algorithm introduced by Rintanen (2008) have a form of a disjunction of facts possibly with negations. The algorithm employs regression operators and satisfiability tests to check whether the clauses form invariants. Each clause initially consists of a single fact or a negation of a fact holding in the initial state. The clause that is not approved as an invariant is replaced by a set of weaker clauses each containing one additional fact (or its negation). Rintanen's algorithm is able to produce invariants in a more general form than h^{m} invariants, because an h^{m} invariant consisting of m facts corresponds to the disjunction $\neg f_1 \vee ... \vee \neg f_m$. Moreover, it was proven that the algorithm produces a superset of h^{m} invariants, therefore, it is a generalization of the h^{m} mutexes.



Facts ($\mathcal{F}$): `(at a)`, `(at b)`, `(at c)`, `(hungry)`, `(fed)`, `(carry-food)`

Operators ($\mathcal{O}$):

`move-a-b`:	`(at a)` $\mapsto$ `(at b)`, $\neg$`(at a)`
`move-b-a`:	`(at b)` $\mapsto$ `(at a)`, $\neg$`(at b)`
`move-b-c`:	`(at b)` $\mapsto$ `(at c)`, $\neg$`(at b)`
`take-food`:	`(at a)`, `(hungry)` $\mapsto$ `(carry-food)`
`feed-gorilla`:	`(at c)`, `(hungry)`, `(carry-food)` $\mapsto$ `(fed)`, $\neg$`(hungry)`, $\neg$`(carry-food)`
`escape`:	`(hungry)` $\mapsto$ `(at c)`, $\neg$`(at a)`, $\neg$`(at b)`, $\neg$`(hungry)`, $\neg$`(carry-food)`

Initial state (s_{init}): `(at b)`, `(hungry)`

Goal (s_{goal}): `(fed)`

Figure 1: The `gorilla-feeding` planning task.

An algorithm for translating PDDL planning tasks into a concise finite domain representation (FDR) was proposed by Helmert (2009). The construction of the FDR is based on identifying invariants in the form of mutex groups. A mutex group states that, at most, one of the invariant facts can be present in any reachable state. The invariants are generated using a guess, check, and repair procedure running on the lifted PDDL domain. The procedure is initialized with small invariants containing only a single atom. The following step is proving the invariants through the identification of so called *threats*. A threat emerges whenever there is an operator that has either two or more instances of invariant atoms in its add effects or the instances in the add effects are not compensated by the same number of instances in the delete effects. The threatened invariants are then either discarded or refined by adding more atoms that could compensate the invariant in the delete effects. The invariants that are not threatened are clearly invariants. The resulting invariants in a lifted form are grounded to a set of facts and they are used in this final form for construction of variables in FDR.

3. Background

Definition 1. A STRIPS **planning task** Π is specified by a tuple $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, where $\mathcal{F} = \{f_1, ..., f_n\}$ is a set of facts, and $\mathcal{O} = \{o_1, ..., o_m\}$ is a set of grounded operators. A **state** $s \subseteq \mathcal{F}$ is a set of facts, $s_{init} \subseteq \mathcal{F}$ is an **initial state** and $s_{goal} \subseteq \mathcal{F}$ is a **goal** specification. An **operator** o is a triple $o = \langle \mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o) \rangle$, 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 $s_{goal} \subseteq s$.

A sequence of operators $\pi = \langle o_1, ..., o_n \rangle$ is applicable in a state s_0 if there are states $s_1, ..., s_n$ such that o_i is applicable in s_{i-1} and $s_i = o_i[s_{i-1}]$ for $1 \leq i \leq n$. The resulting state of this application is $\pi[s_0] = s_n$. A state s is called a **reachable state** if there exists an applicable operator sequence π such that $\pi[s_{init}] = s$. A set of all reachable states is denoted by $\mathcal{R}$. An operator o is called a **reachable operator** iff it is applicable in some

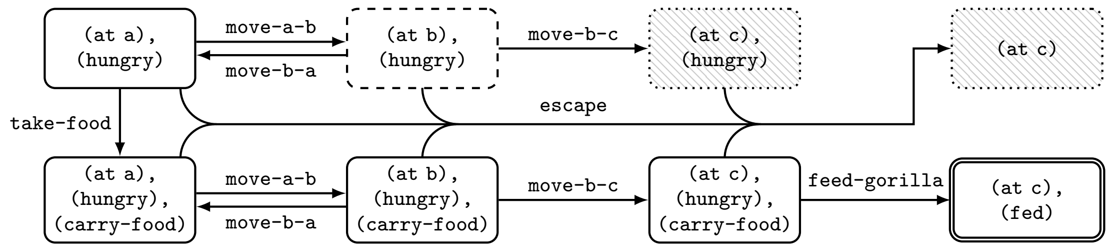


Figure 2: Reachable states and transitions between reachable states in the `gorilla-feeding` planning task.

reachable state. A state s is called a **dead-end state** iff $s_{goal} \not\subseteq s$ and there does not exist any applicable operator sequence π such that $s_{goal} \subseteq \pi[s]$.

Consider the following simple example of the `gorilla-feeding` planning task depicted in Figure 1. The planning task describes a zookeeper whose job is to feed a gorilla. The zookeeper can move between adjacent squares, he can take some food from the stock, carry it to the gorilla and feed it if the gorilla is hungry. The gorilla can escape the zoo if it is hungry.

The planning task is described using six facts: `(at a)`, `(at b)`, and `(at c)` specify a position of the zookeeper, `(hungry)` and `(fed)` denote whether the gorilla is hungry or it was fed, and `(carry-food)` specifies whether the zookeeper carries the food for the gorilla.

The operators in Figure 1 are described using simplified notation where preconditions are placed on the left hand side of the arrow symbol and the effects on the right hand side. The delete effects are listed with the $\neg$ symbol in front of them and the add effects are listed without it. So for example, the operator `feed-gorilla` has three preconditions $\text{pre}(o) = \{$`(at c)`, `(hungry)`, `(carry-food)`$\}$, one add effect $\text{add}(o) = \{$`(fed)`$\}$, and two delete effects $\text{del}(o) = \{$`(hungry)`, `(carry-food)`$\}$. The planning task contains three operators for moving between adjacent squares (`move from-square to-square`), one operator for taking food from the square that contains the food stock (`take-food`), one operator for feeding the gorilla (`feed-gorilla`) that can be applied only on a square where the gorilla is and only when the zookeeper carries the food with him, and one operator corresponding to the gorilla escaping from the zoo (`escape`), which results in the zookeeper being punished by moving into the gorilla's cage. The initial state is set to $s_{init} = \{$`(at b)`, `(hungry)`$\}$ meaning that the zookeeper starts at square `B` and the gorilla is hungry. The goal $s_{goal} = \{$`(fed)`$\}$ is to feed the gorilla.

All eight reachable states of the planning task are depicted in Figure 2 along with all possible transitions between the states. The initial state is marked with the dashed box, the goal state is depicted in a double border box, and the dead-end states are indicated by gray background. Figure 2 shows that the zookeeper can move between adjacent squares which is reflected in the current state as the exchange between `(at ...)` facts. Once the zookeeper takes food from the stock, the current state is extended by the fact `(carry-food)`. And once the gorilla is fed, the gorilla is not hungry anymore and the zookeeper does not carry the food. The effects of operators `move-b-c`, `take-food`, `feed-gorilla`, and `escape` cannot be reversed, i.e., once they are used, it is not possible to come back to the previous state by any sequence of operators.

Note that `move-b-c` can result in a dead-end state if the zookeeper does not carry food and `escape` always results in a dead-end state. These two drawbacks could be fixed, but the `gorilla-feeding` planning task will be used as a running example on which we will demonstrate different types of mutex groups and this enables us to keep the example planning task very brief but with the ability to demonstrate the differences.

4. Mutex Groups

Definition 2. A **mutex** $M \subseteq \mathcal{F}$ is a set of facts such that for every reachable state $s \in \mathcal{R}$ it holds that $M \not\subseteq s$.

Definition 3. A **mutex group** $M \subseteq \mathcal{F}$ is a set of facts such that for every reachable state $s \in \mathcal{R}$ it holds that $|M \cap s| \leq 1$. A mutex group that is not a subset of any other mutex group is called a **maximal mutex group**.

A mutex and a mutex group are both defined as invariants with respect to all states reachable from the initial state by a sequence of operators. A mutex invariant states that certain facts cannot be part of the same reachable state at the same time. So for example, a mutex $\{f_1, f_2, f_3\}$ states that there is no reachable state containing all three facts, but a reachable state containing $\{f_1, f_2\}$, or $\{f_1, f_3\}$, or $\{f_2, f_3\}$ can still exist.

A mutex group is defined as a set of facts out of which, maximally, one can be true in any reachable state, i.e., the facts from a mutex group are pairwise mutex. This is a very broad definition that makes it hard to design a computationally feasible algorithm that would be able to produce all instances of mutex groups. Therefore, we introduce a definition of a more restricted form of mutex groups, namely the fact-alternating mutex group. The definition is based on the applicability of the operators which makes it less complex than the general mutex group as will be demonstrated in Section 6.

Definition 4. A **fact-alternating mutex group** (fam-group) $M \subseteq \mathcal{F}$ is a set of facts such that $|M \cap s_{init}| \leq 1$ and $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ for every operator $o \in \mathcal{O}$. A fam-group that is not a subset of any other fam-group is called a **maximal fact-alternating mutex group** (maximal fam-group).

Proposition 5. *Every fact-alternating mutex group is a mutex group.*

Proof. (By induction) The first part $|M \cap s_{init}| \leq 1$ ensures a mutex group property of M with respect to the initial state. Let s denote a state such that $|M \cap s| \leq 1$, i.e., the mutex group property holds with respect to s. Now, we need to make sure that the mutex group property also holds for every state that is a resulting state from the application of an applicable operator o on s, i.e., for all $o \in \mathcal{O}$ such that $\mathrm{pre}(o) \subseteq s$ the inequality $|M \cap o[s]| \leq 1$ holds. Since $|M \cap s| \leq 1$ and $\mathrm{pre}(o) \subseteq s$ it follows that $|M \cap \mathrm{pre}(o)| \leq 1$ and, furthermore, $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \leq 1$. This means that three cases must be investigated. First, if $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 0$, then $|M \cap \mathrm{add}(o)| = 0$ which means that no additional fact from M can be added to the resulting state and, thus, $|M \cap o[s]| \leq |M \cap s| \leq 1$. Second, if $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 1$ and $|M \cap \mathrm{add}(o)| = 0$, then the same holds. Third, if $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 1$ and $|M \cap \mathrm{add}(o)| = 1$, then $|M \cap \mathrm{pre}(o)| = 1$, thus, $|M \cap s| = 1$ (because $\mathrm{pre}(o) \subseteq s$), so it follows that $M \cap \mathrm{pre}(o) \cap \mathrm{del}(o) = M \cap s \subseteq M \cap \mathrm{del}(o)$. This means that $|M \cap (s \setminus \mathrm{del}(o))| = 0$, so it follows that $|M \cap o[s]| = |M \cap ((s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o))| = 1$, i.e., the mutex group property is preserved in the third case as well. Finally, since the mutex group is defined for reachable states $\mathcal{R}$, every fact-alternating mutex group must be a mutex group. □

The name fact-alternating mutex group was chosen to stress its interesting property, which lies in the mechanism by which facts from a fact-alternating mutex group appear and disappear in particular states after the application of the operators. Consider some fam-group M and some state s that does not contain any fact from M ($M \cap s = \emptyset$). Now we can ask whether any following state $\pi[s]$ can contain any fact from M. The answer is that it cannot because any operator o applicable in s that could add a new fact from M to the following state $o[s]$ would need to have a fact from M in its precondition ($M \cap \mathrm{pre}(o) \neq \emptyset$) which is in contradiction with the assumption that s contains no fact from M. So it follows that facts from each particular fact-alternating mutex group alternate between each other as new states are created and once the facts disappear from the state they cannot ever reappear again in any following state. This is formally proven in the following Proposition 6.

Proposition 6. *Let M denote a fact-alternating mutex group and let s denote a state. If $|M \cap s| = 0$, then for every operator sequence π applicable in s it holds that $|M \cap \pi[s]| = 0$.*

Proof. *(By induction)* The assumption provides a base case. Now let assume that $|M \cap t| = 0$ for some state $t = \pi[s]$ reachable from s and we will show that $|M \cap o[t]| = 0$ for any operator o applicable in t. Since o is applicable in t then $\mathrm{pre}(o) \subseteq t$, and since $|M \cap t| = 0$ then $|M \cap \mathrm{pre}(o)| = 0$, therefore, also $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 0$. So it follows that also $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \geq |M \cap \mathrm{add}(o)| = 0$ because M is a fam-group. Finally, this means that $|M \cap o[t]| = 0$ because o could not add any fact from M into $o[t]$. □

It follows from Proposition 6 that given a fam-group that has an empty intersection with an initial state, none of that the facts the fam-group consists of can ever appear in any reachable state. This observation is not particularly helpful by itself, because these unreachable facts can be detected by a simple reachability analysis. However, if we are interested only in fam-groups that contain reachable facts, Proposition 6 shows that we can safely use a more restricted constraint on the initial state $|M \cap s_{init}| = 1$.

More interestingly, we can use Proposition 6 for the detection of dead-end states. A dead-end state is a state from which it is impossible to reach any goal state by a sequence of applied operators. Consider a fam-group M having a non-empty intersection with the goal ($|M \cap s_{goal}| \geq 1$) and a reachable state s that does not contain any fact from M ($|M \cap s| = 0$). Such a state must be clearly a dead-end state, because it follows from Proposition 6 that all states reachable from s, including the goal states, cannot contain any fact from M, which is formally proven in the following simple corollary of Proposition 6.

Corollary 7. *Let $M \subseteq \mathcal{F}$ denote a set of facts and let s denote a state. If M is a fam-group and $|M \cap s_{goal}| \geq 1$ and $|M \cap s| = 0$, then s is a dead-end state.*

Proof. From Proposition 6 and $|M \cap s| = 0$ it follows that for every operator sequence π applicable in s it holds that $|M \cap \pi[s]| = 0$. Therefore since $|M \cap s_{goal}| \geq 1$, it follows that $s_{goal} \not\subseteq \pi[s]$ which concludes the proof. □

The operators that have more than one fact from some mutex group (and, therefore, also from some fam-group) in its preconditions cannot be applicable in any reachable state. Similarly, the operators with add effects containing more than one fact from some mutex group (fam-group) are also unreachable, because the resulting state would be in contradiction with the mutex group (fam-group).[1] Such operators can be safely removed from the planning task. These two simple rules are not limited to the fact-alternating mutex groups, but they can be used with any type of mutex group.

However, fact-alternating mutex groups provide one additional method for pruning superfluous operators. Consider a fam-group M having a non-empty intersection with the goal and an operator o that does not have any fact from M in its add effects, but it has a non-empty intersection with its preconditions, delete effects, and the fam-group M. The resulting state of the application of the operator o would not contain any fact from the fam-group M. Therefore, such a state would be a dead-end state for the reasons already explained. This means that the operator o can be safely removed from the planning task because it can only produce dead-end states. In other words, the states resulting from the application of the operator are not useful in finding a plan and, therefore, the operator itself is not useful too. This is formally proven in the following corollary.

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 $|M \cap s_{goal}| \geq 1$ and $|M \cap pre(o) \cap del(o)| \geq 1$ and $|M \cap add(o)| = 0$, then $o[s]$ is a dead-end state.*

Proof. $|M \cap s| \leq 1$ because s is a reachable state, and $\mathrm{pre}(o) \subseteq s$ because o is applicable in s. Moreover, since $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \geq 1$, it holds that $|M \cap s| = 1$ and, thus, $M \cap s = M \cap \mathrm{pre}(o) = M \cap \mathrm{del}(o) \neq \emptyset$. Therefore, $|M \cap o[s]| = 0$ because $|M \cap \mathrm{add}(o)| = 0$ and $o[s] = (s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o)$. And finally from $|M \cap s_{goal}| \geq 1$ and Corollary 7 it follows that $o[s]$ is a dead-end state. □

A list of selected mutex groups and fam-groups in the example planning task is shown in Table 1. The maximal mutex groups and maximal fam-groups are marked with a plus sign. The simple corollary of the definition of the mutex group is that every subset of any mutex group is also a mutex group. But the interesting property of fam-groups is that not every subset of this type of mutex group is also a fam-group, the reason is its strict definition. This also means that even though it is always safe to consider a single fact to be a mutex group this does not hold for fam-groups. For example, `(at a)` is not a fam-group because the operator `move-b-a` has `(at a)` as its add effect, but it is not balanced by a delete effect and it cannot be because operators are not allowed to have the same facts in its add and delete effects. On the other hand, `(hungry)` is a fam-group because it is not listed as an add effect of any operator. This observation can be generalized and we can say that any single fact is a fam-group if and only if it does not appear in any add effect, which we formally prove in the following proposition.

Proposition 9. *Let $f_1 \in \mathcal{F}$ denote a single fact. $\{f_1\}$ is a fam-group iff $f_1 \notin add(o)$ for every operator $o \in \mathcal{O}$.*

1. This is a special case of disambiguation proposed by Alcázar, Borrajo, Fernández, and Fuentetaja (2013).

	mutex group	fam-group
{`(at a)`}	✓	✗
{`(hungry)`}	✓	✓
{`(carry-food)`, `(fed)`}	✓+	✗
{`(at a)`, `(at b)`}	✓	✓+
{`(at b)`, `(at c)`}	✓	✗
{`(at a)`, `(at b)`, `(fed)`}	✓+	✗
{`(at a)`, `(at b)`, `(at c)`}	✓+	✗
{`(hungry)`, `(fed)`}	✓+	✓+
{`(hungry)`, `(carry-food)`}	✗	✗

Table 1: A list of selected mutex groups and fam-groups in the `gorilla-feeding` planning task. Maximal mutex groups and maximal fam-groups are marked with a plus sign.

Proof. First, we prove the direction from left to right by contradiction. Assuming there exists an operator o such that $f_1 \in \mathrm{add}(o)$, also $f_1 \in \mathrm{pre}(o)$ must hold, because the inequality $|\{f_1\} \cap \mathrm{add}(o)| \leq |\{f_1\} \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ must hold. This is in contradiction with the assumption $\mathrm{add}(o) \cap \mathrm{pre}(o) = \emptyset$ (Definition 1). Similarly, to prove the other direction by contradiction, we assume that $\{f_1\}$ is not a fam-group. Since $|\{f_1\} \cap s_{init}| \leq 1$ always holds, the inequality $|\{f_1\} \cap \mathrm{add}(o)| > |\{f_1\} \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ must hold. Therefore, $|\{f_1\} \cap \mathrm{add}(o)| \geq 1$, therefore, $\{f_1\} \in \mathrm{add}(o)$, which is contradiction. □

The facts `(carry-food)` and `(fed)` do not form a fam-group because of the operator `take-food` which adds the `(carry-food)` fact, but does not delete the `(fed)` fact. This is exactly the type of mutex group that is not covered by the fact-alternating mutex group because the facts from this mutex group appear in the state seemingly from nothing, i.e., the facts do not alternate between each other, but their appearance is conditional on some other fact that is not part of the mutex group.

The maximal mutex groups and maximal fam-groups are those that cannot be extended by any fact and still remain mutex groups and fam-groups, respectively. Therefore all other mutex groups or fam-groups are already contained within the maximal ones, but we need to be careful while considering classification of the subsets of the maximal mutex groups or fam-groups. It is obviously true that every subset of a maximal mutex group is also a mutex group. Nevertheless, not every subset of a maximal fam-group is also a fam-group.

For example, {`(at a)`, `(at b)`, `(at c)`} is a maximal mutex group and, therefore, {`(at a)`, `(at b)`} and {`(at b)`, `(at c)`} are mutex groups. However, {`(at a)`, `(at b)`} is a maximal fam-group, but {`(at a)`} is not a fam-group as discussed above. Moreover, {`(at a)`, `(at b)`, `(at c)`} is not a fam-group, because of the operator `escape`, which adds `(at c)` without balancing it by `(at a)` or `(at b)`. But even if we remove the operator `escape` and, therefore, {`(at a)`, `(at b)`, `(at c)`} becomes a maximal fam-group, its subset {`(at b)`, `(at c)`} still would not be a fam-group. The set {`(hungry)`, `(fed)`} is both a maximal mutex group and a maximal fam-group, although {`(fed)`} is not a fam-group, but {`(hungry)`} is a fam-group.

The example planning task also demonstrates how fam-groups can be useful in dealing with dead-end states. The operator `escape` always produces a dead-end state. According to Corollary 8, the fam-group {`(hungry)`, `(fed)`} can be used to remove this operator, because `(fed)` is a part of the goal specification and the operator removes `(hungry)` but does not add `(fed)`. In other words, $\{$`(hungry)`, `(fed)`$\}$ is a fam-group and `(fed)` must be a part of every goal state, therefore, the facts `(hungry)` and `(fed)` must alternate between each other in all states between the initial state and the goal state. Therefore, since the resulting state of application of `escape` does not contain any of those two facts, the operator `escape` cannot be part of any operator sequence leading from the initial state to a goal state. Note also that Corollary 8 is not limited to maximal fam-groups.

5. Mutex Groups in Regression

Until now, we considered mutex groups as state invariants that hold in all states reachable in progression, i.e., they hold in the initial state s_{init} and all states that can be reached by consecutive application of operators in a forward direction. However, we can also consider state invariants in regression, i.e., starting from the goal specification s_{goal} proceeding backwards towards the initial state s_{init}. More precisely, planning in regression starts in the goal specification s_{goal}, an operator o is applicable in regression in a state s^{reg} iff $\mathrm{del}(o) \cap s^{\mathrm{reg}} = \emptyset$, and the resulting state of the application in regression is a new regression state $(s^{\mathrm{reg}} \setminus \mathrm{add}(o)) \cup \mathrm{pre}(o)$. A state in regression corresponds to a set of states in progression and, therefore, a goal state in regression is any state s^{reg} such that $s^{\mathrm{reg}} \subseteq s_{init}$. Every plan in regression corresponds to a reversed plan in progression, thus, we can think of regression search as solving a reversed planning task. Nevertheless, we should stress that search in progression and regression are not both always equally suitable for all domains (Massey, 1999).

Similarly to state invariants in progression, we can define state invariants in regression as logical formulas that hold in all reachable regression states, and analogously for mutex groups. Unfortunately, defining fam-groups in regression would require more changes due to the different way the operators are used in regression. However, we can also use a formulation of a dual (reversed) planning task of the original planning task, initially proposed by Massey (1999) and later revisited by Pettersson (2005) and Suda (2013).

Definition 10. Given a planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, its corresponding **dual planning task** $\Pi^{\mathcal{D}}$ is the planning task $\Pi^{\mathcal{D}} = \langle \mathcal{F}, \mathcal{O}^{\mathcal{D}}, \overline{s_{goal}}, \overline{s_{init}} \rangle$, where $\mathcal{F}$ is a set of facts from Π, $\overline{s_{goal}} = \mathcal{F} \setminus s_{goal}$, and $\overline{s_{init}} = \mathcal{F} \setminus s_{init}$. Operators $\mathcal{O}^{\mathcal{D}} = \{o^{\mathcal{D}} \mid o \in \mathcal{O}\}$ are constructed from $\mathcal{O}$ such that $\mathrm{pre}(o^{\mathcal{D}}) = \mathrm{del}(o)$, $\mathrm{add}(o^{\mathcal{D}}) = \mathrm{add}(o)$, and $\mathrm{del}(o^{\mathcal{D}}) = \mathrm{pre}(o)$ for every $o \in \mathcal{O}$.

A dual planning task is an ordinary planning task where the initial state and the goal specification correspond to complements of the goal specification and the initial state of the original problem, respectively, and operators have exchanged preconditions and delete effects. The search in a dual planning task corresponds to the search in regression in the original planning task. Therefore, every plan in Π is a reversed plan in $\Pi^{\mathcal{D}}$, and also a dead-end state in Π is an unreachable state in $\Pi^{\mathcal{D}}$. It should be obvious that as $\Pi^{\mathcal{D}}$ is a dual planning task to Π, Π is a dual planning task to $\Pi^{\mathcal{D}}$, meaning that a plan in $\Pi^{\mathcal{D}}$ is a reversed plan in Π, and a dead-end state in $\Pi^{\mathcal{D}}$ is an unreachable state in Π. It also follows that (fact-alternating) mutex groups in the dual planning task can be inferred the same way as in progression, because we can always construct a dual planning task and infer (fact-alternating) mutex groups there.

Facts ($\mathcal{F}$): `(at a)`, `(at b)`, `(at c)`, `(hungry)`, `(fed)` `(carry-food)`

Operators ($\mathcal{O}^{\mathcal{D}}$):
`move-a-b`$^{\mathcal{D}}$: `(at a)` $\mapsto$ `(at b)`, $\neg$`(at a)`
`move-b-a`$^{\mathcal{D}}$: `(at b)` $\mapsto$ `(at a)`, $\neg$`(at b)`
`move-b-c`$^{\mathcal{D}}$: `(at b)` $\mapsto$ `(at c)`, $\neg$`(at b)`
`take-food`$^{\mathcal{D}}$: $\emptyset \mapsto$ `(carry-food)`, $\neg$`(at a)`, $\neg$`(hungry)`
`feed-gorilla`$^{\mathcal{D}}$: `(hungry)`, `(carry-food)` $\mapsto$ `(fed)`, $\neg$`(at c)`, $\neg$`(hungry)`, $\neg$`(carry-food)`
`escape`$^{\mathcal{D}}$: `(at a)`, `(at b)`, `(hungry)`, `(carry-food)` $\mapsto$ `(at c)`, $\neg$`(hungry)`

Initial state ($\overline{s_{goal}}$): `(at a)`, `(at b)`, `(at c)`, `(hungry)`, `(carry-food)`

Goal ($\overline{s_{init}}$): `(at a)`, `(at c)`, `(carry-food)` `(fed)`

Figure 3: The `gorilla-feeding` dual planning task.

The example in Figure 3 shows the `gorilla-feeding` (from Figure 1) dual planning task. The dual operators for moving between adjacent squares are the same as in the original planning task, because the preconditions and delete effects are exactly the same in those operators. The three remaining operators differ from their counterparts in the original problem. The dual initial state $\overline{s_{goal}}$ contains all the facts except `(fed)`, which means that any dual mutex group can contain, at most, two facts and all non-trivial dual mutex groups must contain the fact `(fed)`. As it turns out, the only non-trivial mutex group (and the only fam-group as well) in the dual `gorilla-feeding` problem is $\{$`(hungry)`, `(fed)`$\}$ (which is also a mutex group and a fam-group in Π).

Note also that the operator `escape` cannot be detected as a dead-end operator by a simple reachability analysis, because all its preconditions are reachable in the dual planning task. However, fam-group $\{$`(hungry)`, `(fed)`$\}$ can be used for the removal using Corollary 8, similarly as it can be used in progression.

6. Complexity Analysis

The complexity analysis of the mutex group structure we propose is based on an analysis of complexity classes of decision problems corresponding to the problems of finding the largest possible mutex groups. We will show that such a decision problem for mutex groups is asymptotically harder than for fam-groups even though, in general, there can be exponentially many mutex groups of both types.

Definition 11. Let $\mathcal{M}$ denote a set of all mutex groups (fam-groups). M is a **maximum mutex group** (**maximum fam-group**) iff $M \in \mathcal{M}$ and $|M| \geq |N|$ for every $N \in \mathcal{M}$.

Definition 12. Maximum-Mutex-Group (Maximum-FAM-Group) decision problem: Given a planning task Π and an integer k, does Π contain a mutex group (fam-group) of size at least k?

A maximum mutex group is a mutex group that has the maximum possible number of facts in the corresponding planning task, i.e., the maximum mutex groups are the largest mutex groups in the number of facts they consist of. It should be clear that every maximum mutex group is also a maximal mutex group by Definition 3 (but not the other way around) and the same holds for fam-groups. Maximum-Mutex-Group is a decision problem corresponding to the task of finding a maximum mutex group. Similarly, Maximum-FAM-Group is a decision problem corresponding to finding a maximum fam-group. In Section 6.1 we prove that the Maximum-FAM-Group is NP-Complete and in Section 6.2 we will show that Maximum-Mutex-Group is PSPACE-Complete.

6.1 Maximum-FAM-Group is NP-Complete

Definition 13. An undirected simple **graph** G is a tuple $G = \langle N, E\rangle$, where N denotes a set of nodes and E denotes a set of edges such that each edge $\{n_i, n_j\} \in E$ connects two different nodes ($n_i \neq n_j$) and there are no two edges connecting the same nodes. A non-empty set C of nodes of G forms a **clique** if each node of C is connected by an edge to every other node of C. A clique that is not a subset of any other clique is called a **maximal clique**. A **maximum clique** is a clique such that there is no other clique consisting of more nodes.

Definition 14. Maximum-Clique decision problem: Given a graph G and an integer k, does G contain a clique of size at least k?

Definition 15. Given a graph $G = \langle N, E\rangle$, $\Pi^G = \langle \mathcal{F}^G, \mathcal{O}^G, s^G_{init}, \emptyset\rangle$ denotes a planning task where $\mathcal{F}^G = N \cup \{\top\}$, $\top \notin N$, $s^G_{init} = \{\top\}$, and $\mathcal{O}^G = \{o_{n_1,n_2} \mid \{n_1, n_2\} \subseteq N, n_1 \neq n_2, \{n_1, n_2\} \notin E\}$ with $\text{pre}(o_{n_1,n_2}) = \{\top\}$, $\text{del}(o_{n_1,n_2}) = \{\top\}$, and $\text{add}(o_{n_1,n_2}) = \{n_1, n_2\}$.[2]

The Maximum-Clique problem is a well known NP-Complete decision problem (Karp, 1972), which we use to show that the Maximum-FAM-Group is NP-Hard. The reduction from Maximum-Clique is made by translating a graph G into a planning task Π^G (Definition 15) in a polynomial time. After the translation, it is shown that Maximum-Clique for G can be solved by solving Maximum-FAM-Group for Π^G. In other words, we show that the fam-group decision problem is at least as hard as some NP-Complete problem, in this case Maximum-Clique.

Proving that the Maximum-FAM-Group belongs to NP is much easier, because the definition of fam-groups provides a verification algorithm running in polynomial time, which concludes the proof that the Maximum-FAM-Group is NP-Complete (Theorem 18). Moreover, it follows from the polynomial reduction, as we propose it, that the maximum possible number of maximal fam-groups is exponential in the number of facts of the corresponding planning task (Proposition 19).

The main idea behind the way Π^G is constructed from G, is the following: Consider a complete graph. In such a graph, all nodes form one maximal clique together and by gradual removal of the edges from the graph, the original clique is divided into more cliques consisting of a smaller number of nodes. Similarly, consider a planning task having only one fact in the initial state and without any operator. In such a planning task, all facts form one maximal fam-group together. This fam-group can be divided into smaller ones by adding new operators having facts that should not be part of the same fam-group, into their add effects, without balancing them by delete effects and preconditions. So following this idea, the algorithm constructs the resulting planning task in such a way that the operators' add

2. We slightly abuse the notation here and in the rest of this section to simplify the notation.

effects correspond to the pair of nodes in the original graph that are not connected by any edge. Therefore, they cannot be in the same clique and the operators make sure that they cannot also be in the same fam-group. The additional auxiliary fact $\top$ is added to make sure that all operators are applicable in the initial state, thus, effects of the operators are reachable.

The proof that the Maximum-FAM-Group is NP-Complete starts with some auxiliary lemmas. Lemma 16 shows that we can translate a graph G into the corresponding planning task Π^G and all cliques (including the maximum ones) are preserved during the translation in the form of fam-groups. More precisely, it is shown that if C is a clique in G, then the corresponding fam-group in Π^G can be constructed as $C \cup \{\top\}$ and also that every fam-group in Π^G containing $\top$ corresponds to a clique in the original graph G.

The remaining piece of the proof of correctness of the polynomial reduction from the Maximum-Clique problem is to show that there are no maximum fam-groups that do not contain $\top$, i.e., we must show that if we find a maximum fam-group, then we can reconstruct a maximum clique in the original graph G from it and, therefore, the Maximum-Clique problem can be solved by solving the the Maximum-FAM-Group. In Lemma 17, we prove an even stronger statement saying that not only all maximum fam-groups, but all maximal fam-groups contain $\top$. Therefore, Lemma 16 can be safely used to prove that the polynomial reduction from the Maximum-Clique problem is correct, thus, the Maximum-FAM-Group is NP-Hard.

The main contribution of this section is formulated in Theorem 18 stating that the Maximum-FAM-Group is NP-Complete. Once we have proven that the decision problem is NP-Hard then it easily follows that it must be NP-Complete because any fam-group can be verified in a polynomial number of steps by checking the initial state and all operators.

Lemma 16. *C is a clique in G iff $M = C \cup \{\top\}$ is a fam-group in Π^G.*

Proof. To prove the direction from left to right by contradiction, let us assume that C is a clique and M is not a fam-group. Since $s^G_{init} = \{\top\} \subseteq M$ and $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 1$ for every operator $o \in \mathcal{O}^G$ (because $\mathrm{pre}(o) = \mathrm{del}(o) = \{\top\}$) there must exist an operator $o^\star \in \mathcal{O}^G$ such that $|M \cap \mathrm{add}(o^\star)| > 1$. Since $\top$ is not part of any add effect and all add effects contain exactly two facts, it must hold that $\mathrm{add}(o^\star) \subseteq C$. This is in contradiction with the assumption that C is a clique because all add effects are created only from the pairs of nodes that are not joined by an edge and there is no such pair of nodes in C by definition. Therefore, if C is a clique, then M is a fam-group.

To prove the other direction, also by contradiction, let us assume that $M = C \cup \{\top\}$ is a fam-group and C is not a clique. If C is not a clique then there exist $n_1, n_2 \in C$ such that n_1 and n_2 are not connected by an edge in G. So it follows that there exists an operator $o \in \mathcal{O}^G$ such that $\mathrm{add}(o) = \{n_1, n_2\}$ and $\mathrm{pre}(o) = \mathrm{del}(o) = \{\top\}$, therefore, $|M \cap \mathrm{add}(o)| = 2 > |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 1$. This is in contradiction with the assumption that M is a fam-group, therefore, if M is a fam-group then C is a clique. □

Lemma 17. *For every maximal fam-group M in Π^G it holds that $\top \in M$.*

Proof. Let N denote a fam-group such that $\top \notin N$. Now we prove that $M = \{\top\} \cup N$ is also a fam-group. Since $s^G_{init} = \{\top\}$, then surely $\left|M \cap s^G_{init}\right| \leq 1$. For every operator $o \in \mathcal{O}^G$ it holds that $\mathrm{pre}(o) = \mathrm{del}(o) = \{\top\}$ and $\top \notin \mathrm{add}(o)$ and $|N \cap \mathrm{add}(o)| \leq |N \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$.

So, it follows that $|N \cap \text{add}(o)| = |N \cap \text{pre}(o) \cap \text{del}(o)| = 0$, therefore, $|M \cap \text{add}(o)| = 0 \leq |M \cap \text{pre}(o) \cap \text{del}(o)| = 1$, therefore, M is a fam-group. Finally, since every fam-group can be extended by $\top$, then surely every maximal fam-group must contain $\top$. □

Theorem 18. *Maximum-FAM-Group is NP-Complete.*

Proof. To show that the Maximum-FAM-Group is NP-Hard, we will reduce Maximum-Clique to the Maximum-FAM-Group. Clearly, any graph G can be translated into a planning task Π^G in polynomial time, namely $O(n^2)$, where n is the number of nodes in G. From Lemma 17, it follows that a maximum fam-group must contain $\top$ and then it follows from Lemma 16 that C is a maximum clique in G iff $M = C \cup \{\top\}$ is a maximum fam-group in Π^G. Therefore, the Maximum-FAM-Group is NP-Hard.

What remains is to show that the Maximum-FAM-Group is in NP. It is easy to see that given a set of facts it can be verified as a fam-group by checking the initial state and all operators according to Definition 4. The verification procedure runs in a polynomial number of steps in a number of facts and operators. Therefore, the Maximum-FAM-Group is NP-Complete. □

Proposition 19. *The maximum possible number of maximal fam-groups in a planning task* Π *is exponential in a number of facts.*

Proof. It follows from Lemma 16 and Lemma 17 that for every possible graph, it is possible to construct a planning task in which every maximal fam-group corresponds to some maximal clique and vice versa. The maximum possible number of maximal cliques in a graph is exponential in a number of nodes (namely $c \cdot 3^{n/3}$ where n is number of nodes and $c \in \{1, 4/3, 2\}$ depending on $n \bmod 3$) (Moon & Moser, 1965). This makes the lower bound exponential. The upper bound is the maximum number of subsets of $\mathcal{F}$, which is also exponential ($2^{|\mathcal{F}|}$). This makes the maximum possible number of maximal fam-groups exponential in a number of facts. □

6.2 Maximum-Mutex-Group is PSPACE-Complete

Definition 20. Plan-Exist decision problem: Given a planning task Π, determine the existence of a solution.

In this section, we will show that the Maximum-Mutex-Group is PSPACE-Complete (Theorem 26). First, it will be proven that it is PSPACE-Hard (Proposition 23) using a polynomial reduction from Plan-Exist which is known to be PSPACE-Complete (Bylander, 1994). Then, we will present a PSPACE algorithm (Algorithm 1) solving the Maximum-Mutex-Group problem which leads to the conclusion that the Maximum-Mutex-Group is PSPACE-Complete. Moreover, we will show that the maximum possible number of maximal mutex groups is exactly the same as the maximal fam-groups and we will express this number exactly.

Definition 21. Given a planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, $\Pi^M = \langle \mathcal{F}^M, \mathcal{O}^M, s_{init}, s_{goal} \rangle$ denotes a planning task such that $\mathcal{F}^M = \mathcal{F} \cup \{\bot\}$, $\bot \notin \mathcal{F}$, and $\mathcal{O}^M = \mathcal{O} \cup \{o^{\text{sat}}\}$, where $\text{pre}(o^{\text{sat}}) = s_{goal}$, $\text{del}(o^{\text{sat}}) = \{\}$, and $\text{add}(o^{\text{sat}}) = \mathcal{F}^M$.

Lemma 22. *Let M denote a maximum mutex group in Π^M. A solution of Π exists iff $|M| \leq 1$.*

Proof. A solution of Π exists iff there exists a reachable state s such that $s_{goal} \subseteq s$. If such s exists then o^{sat} is a reachable operator and, thus, its resulting state $s^{\text{sat}} = \mathcal{F}^M$ is also reachable. So it follows that every mutex group of Π^M consists of, at most, one fact because any mutex group N having more than one fact would violate the mutex group property on s^{sat} ($\left|N \cap s^{\text{sat}}\right| \geq 2$). Therefore, if a solution of Π exists, then $|M| \leq 1$.

To prove the other direction by contradiction, let us assume that we have a maximum mutex group M in Π^M such that $|M| \leq 1$ and Π has no solution. If Π has no solution, then there does not exist a reachable state s such that $s_{goal} \subseteq s$, which means that o^{sat} is not applicable in any reachable state, therefore, the state $s^{\text{sat}} = \mathcal{F}^M$ is not reachable. But the fact $\perp$ appears in Π^M only in s^{sat}, therefore, any mutex group in Π^M can be extended by $\perp$ and it still remains a mutex group. Finally, since a mutex group of size of at least one (a single fact is always a mutex group) exists in any planning task and since such a mutex group can be in Π^M extended by $\perp$, then it follows that a maximum mutex group M must have at least two facts ($|M| \geq 2$) which is in contradiction with the assumption that $|M| \leq 1$. Therefore, if $|M| \leq 1$, then Π has a solution. □

Proposition 23. *Maximum-Mutex-Group is PSPACE-Hard.*

Proof. We will reduce Plan-Exist to Maximum-Mutex-Group. Clearly, any planning task Π can be translated to a different planning task Π^M in polynomial time. It follows from Lemma 22 that we can determine whether Π has a solution by solving the Maximum-Mutex-Group problem on Π^M in the following way: If the maximum mutex group in Π^M has, at most, one fact, then the planning task Π has a solution. If the maximum mutex group in Π^M has more than one fact, then the planning task Π does not have a solution. Therefore, the Maximum-Mutex-Group is PSPACE-Hard. □

Algorithm 1: Maximum-Mutex-Group

Input: A constant k, planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$
Output: "Yes" or "No"

Construct a complete graph $G = \langle N = \mathcal{F}, E = \{p \mid p \subseteq \mathcal{F}, |p| = 2\}$;
for each $f_1, f_2 \in \mathcal{F}$ *such that* $f_1 \neq f_2$ **do**
  if *there exists a plan for* $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, \{f_1, f_2\} \rangle$ **then** /* Plan-Exist */
    $E \leftarrow E \setminus \{f_1, f_2\}$;
  end
end
if *G contains a clique of size at least k* **then return** "Yes"; /* Maximum-Clique */
else return "No";

Once we have proven that the Maximum-Mutex-Group is PSPACE-Hard, proving that it is also PSPACE-Complete requires to show that it is possible to decide the Maximum-Mutex-Group using a polynomial amount of space. Algorithm 1 shows a pseudocode for such a procedure. The main idea of the algorithm is that every set of facts M of size of at least two is a mutex group if and only if every pair of facts from M is also a mutex group (Proposition 24). In other words, if we are able to infer all mutex groups containing exactly two facts, we can always use these mutex pairs for the construction of all other mutex groups of size of at least two. The main cycle of Algorithm 1 uses Plan-Exist to prove whether each pair of facts is a mutex group or if it is not, i.e., whether the facts are part of some reachable state or not. The inferred mutex pairs are used for construction of a graph where each edge corresponds to one mutex pair. And finally, Maximum-Clique is used to infer a maximum mutex group. Such an algorithm clearly uses only a polynomial amount of space which is formally proven in Lemma 25. Theorem 26 just joins Proposition 23 (Maximum-Mutex-Group is PSPACE-Hard) and Lemma 25 (Maximum-Mutex-Group belongs to PSPACE) to formulate the main contribution of this section, i.e., the proof that the Maximum-Mutex-Group is PSPACE-Complete.

Proposition 24. *Let $M \subseteq \mathcal{F}$ denote a set of facts such that $|M| \geq 2$ and let $\mathcal{D}^M$ denote a set of all pairs of all facts from M, i.e., $\mathcal{D}^M = \{p \mid p \subseteq M, |p| = 2\}$. M is a mutex group iff every $P \in \mathcal{D}^M$ is a mutex group.*

Proof. It is easy to see that if M is a mutex group (i.e., $|s \cap M| \leq 1$ for every reachable state s) then every subset of M also must be a mutex group (i.e., for every $N \subseteq M$ it also holds that also $|s \cap N| \leq 1$ for every reachable state s).

To prove the other direction by contradiction, let us assume that every $P \in \mathcal{D}^M$ is a mutex group, but M is not a mutex group. If M is not a mutex group then there exists a reachable state s such that $|s \cap M| \geq 2$. This means that there must exist a pair of facts $\{f_1, f_2\}$ such that $f_1, f_2 \in M$ and $f_1, f_2 \in s$ which is in contradiction with the assumption that every $P \in \mathcal{D}^M$ is a mutex group because $\{f_1, f_2\}$ must belong to $\mathcal{D}^M$ by definition. □

Lemma 25. *Algorithm 1 decides Maximum-Mutex-Group using a polynomial amount of space in the size of the input.*

Proof. Algorithm 1 starts with a complete graph constructed from the facts as its nodes. Then, in $O(|\mathcal{F}|^2)$ steps, each pair of facts is checked whether they appear together in any reachable state (line 3). This is checked by deciding the Plan-Exist on the modified input planning task, where the original goal is replaced by a new goal consisting of the tested pair of facts. Plan-Exist is PSPACE-Complete, therefore this step requires at most a polynomial amount of space. If there exists a reachable state containing both facts, an edge connecting those two facts is removed from the graph. The edges remaining in the graph only connect those facts that never appear together in the same reachable state. Therefore every pair of facts connected by an edge is a mutex group.

It follows from Proposition 24 that having all mutex groups of size two is enough to construct any other mutex group which also covers the maximum mutex groups. Therefore, by deciding Maximum-Clique on the constructed graph with the same constant k (line 7), the Maximum-Mutex-Group is decided too. This also requires at most a polynomial amount of space, because Maximum-Clique is NP-Complete. (If the graph has no edges, any single fact is a maximum mutex group and any single node is a maximum clique as well.) □

Theorem 26. *Maximum-Mutex-Group is PSPACE-Complete.*

Proof. The Maximum-Mutex-Group is PSPACE-Hard (Proposition 23) and it also belongs to PSPACE (Lemma 25), therefore, the Maximum-Mutex-Group is PSPACE-Complete. □

Proposition 19 states that the maximum number of maximal fam-groups is exponential in a number of facts. The proof of Proposition 19 is based on the proposed procedure that can translate any graph into a planning task in such a way that every maximal fam-group corresponds to a maximal clique in the original graph. This enables us to enumerate the lower bound on the maximum possible number of maximal fam-groups as the maximum possible number of maximal cliques.

It follows from Proposition 24 that given a complete list of all mutex pairs, all maximal mutex groups can be constructed using an algorithm for listing all maximal cliques. This means that the maximum possible number of maximal mutex groups is exactly the same as the maximum possible number of maximal cliques. Furthermore, since the same number is the lower bound on the maximum possible number of maximal fam-groups and since every fam-group is also a mutex group, the maximum possible number of maximal mutex groups and maximal fam-groups are exactly the same, which we formally prove in Proposition 27.

Proposition 27. *Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$ denote a planning task and let $n = |\mathcal{F}|$ denote a number of facts in Π. The maximum possible number $\mu(n)$ and $\mu_{fa}(n)$ of maximal mutex groups and maximal fam-groups, respectively, for $n \geq 2$, is the following:*

$$\mu(n) = \mu_{fa}(n) = \begin{cases} 3^{n/3}, & \textit{if } n \bmod 3 = 0; \\ \frac{4}{3} \cdot 3^{n/3}, & \textit{if } n \bmod 3 = 1; \\ 2 \cdot 3^{n/3}, & \textit{if } n \bmod 3 = 2. \end{cases}$$

Proof. It follows from Proposition 24 that all maximal mutex groups can be constructed from a complete set of mutex pairs using an algorithm for the enumeration of all maximal cliques. Moreover, it is easy to see that given a set of facts, it is always possible to construct a planning task that would contain any combination of mutex pairs. It follows from the proof of Proposition 19 that the same holds for maximal fam-groups. This means that the maximum possible number of maximal mutex groups and maximal fam-groups in a planning task is exactly the same as the maximum possible number of maximal cliques in a graph (Moon & Moser, 1965). □

7. Relationship Between h^2 and Fact-Alternating Mutex Groups

The h^{m} heuristic (Haslum & Geffner, 2000) discussed in Section 2 is able to produce a set of mutex invariants. More specifically, the h^2 heuristic provides a method for inferring pairs of facts that cannot hold together in any reachable state. Such pairs of facts can be interpreted as both mutex invariants and mutex groups because if two facts cannot both be part of the same reachable state, then, at most, one of them can be a part of any reachable state, which is exactly the definition of a mutex group, i.e., every mutex pair is also a mutex group. This reasoning does not apply generally to the h^{m} heuristic because for $m \geq 3$ stating that a set of three or more facts is unreachable, does not necessarily mean that, at most, one of these facts can be part of the same reachable state. For example, if the h^3 heuristic provides a set of three facts $\{f_1, f_2, f_3\}$ that is not part of any reachable state (i.e., it is a mutex) then it could be the case that there are reachable states that contain both f_1 and f_2 but not f_3 (i.e., the set $\{f_1, f_2, f_3\}$ would not be a mutex group). Therefore, from the whole family of h^m heuristics only h^2 can be used for the inference of mutex groups by Definition 3. From now on, the mutex groups consisting of two facts (mutex pairs) generated by h^2 will be called h^2-mutexes.

Proposition 24, from the previous section, shows that any mutex group consisting of, at least, two facts can be decomposed into a set of mutex pairs and that the original mutex group can be always reconstructed back from this set of mutex pairs. Moreover, it follows from the proof of Lemma 25 that if we somehow obtain a complete set of mutex pairs, we can determine the maximum mutex group simply by invoking Maximum-Clique which is NP-Complete. However, finding the maximum mutex group is as hard as planning itself, therefore, it follows that inference of a complete set of mutex pairs is also as hard as finding a plan. So since h^2 runs in polynomial time we can conclude that this method is far from being complete with respect to mutex pairs.

Nevertheless, an interesting question is, what is the relationship between fam-groups and h^2-mutexes? In this section we will resolve this question by showing that h^2 always produces a (possibly non-strict) superset of decomposition of all fam-groups. More precisely, we will prove that any mutex pair that is a subset of a fam-group must be an h^2-mutex, but not the other way around. This also means that if we infer h^2-mutexes and use an algorithm for listing maximal cliques to join the h^2-mutexes into larger mutex groups (similarly as it is used in Algorithm 1), then the resulting mutex groups will be non-strict supersets of fam-groups. However, the mutex groups created from h^2-mutexes do not have the same properties as fam-groups described in Proposition 6, Corollary 7 and Corollary 8. The importance of fam-groups, in particular its ability to detect operators that can produce only dead-end states, will be demonstrated by the experimental results in Section 10.

The formal definition of an h^2-mutex below is based on an alternative characterization of the h^m heuristic using a modified planning task introduced by Haslum (2009). In our opinion, this formulation leads to a more straightforward line of proof than the original definition by a recursive equation and regression operators.

Definition 28. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$ denote a planning task. The planning task $\Pi^2 = \langle \Phi, \Omega, \psi_{init}, \{\} \rangle$ consists of a set of facts $\Phi = \{\phi_c \mid c \subseteq \mathcal{F}, |c| \leq 2\}$, a set of operators Ω, an initial state $\psi_{init} = \{\phi_c \mid c \subseteq s_{init}, |c| \leq 2\}$, and an empty goal specification. For each operator $o \in \mathcal{O}$, the planning task Π^2 contains an operator $\omega_{o,\emptyset} \in \Omega$ with

$\mathrm{pre}(\omega_{o,\emptyset}) = \{\phi_c \mid c \subseteq \mathrm{pre}(o), |c| \leq 2\}$,
$\mathrm{add}(\omega_{o,\emptyset}) = \{\phi_c \mid c \subseteq \mathrm{add}(o), |c| \leq 2\}$,
$\mathrm{del}(\omega_{o,\emptyset}) = \emptyset$,

and additionally, for each operator $o \in \mathcal{O}$ and each fact $f \in \mathcal{F}$ such that $f \notin \mathrm{add}(o) \cup \mathrm{del}(o)$, the planning task Π^2 contains an operator $\omega_{o,f} \in \Omega$ with

$\mathrm{pre}(\omega_{o,f}) = \mathrm{pre}(\omega_{o,\emptyset}) \cup \{\phi_{\{f\}}\} \cup \{\phi_{\{g,f\}} \mid g \in \mathrm{pre}(o), g \neq f\}$,
$\mathrm{add}(\omega_{o,f}) = \mathrm{add}(\omega_{o,\emptyset}) \cup \{\phi_{\{g,f\}} \mid g \in \mathrm{add}(o)\}$,
$\mathrm{del}(\omega_{o,f}) = \emptyset$.

Let Ψ denote a set of all reachable states in Π^2. A pair of facts $\{f_1, f_2\} \subseteq \mathcal{F}$ such that $f_1 \neq f_2$ is an **h^2-mutex** iff for every reachable state $\psi \in \Psi$, it holds that $\phi_{\{f_1,f_2\}} \notin \psi$.

The Π^2 planning task is an ordinary STRIPS planning task (Definition 1), but we have decided to use Greek letters to describe its parts to prevent confusion between the parts of the original planning task Π and the parts of Π^2 which is constructed from Π. Note also that contrary to the original formulation by Haslum, Π^2 has an empty goal specification. The reason is that we do not need a goal specification because we are not interested in the h^2 heuristic, but only in the h^2-mutexes resulting from the reachability of facts of the planning task.

Also note the difference between our construction of operators in Π^2 and how Haslum defined the operators. Haslum uses a single formula for preconditions and effects (Haslum, 2009, Definition 4). So in our notation, the definition would be the following. For each operator $o \in \mathcal{O}$ and for each subset of facts $g \subseteq \mathcal{F}$ such that $|g| \leq 1$ and g is disjoint with $\text{add}(o)$ and $\text{del}(o)$, create a new operator $\omega_{o,g}$ with: $\text{pre}(\omega_{o,g}) = \{\phi_c \mid c \subseteq (\text{pre}(o) \cup g), |c| \leq 2\}$, $\text{add}(\omega_{o,g}) = \{\phi_c \mid c \subseteq (\text{add}(o) \cup g), c \cap \text{add}(o) \neq \emptyset, |c| \leq 2\}$, $\text{del}(\omega_{o,g}) = \emptyset$. We, however, decided to split the definition of operators between those that are direct images of the original operators ($\omega_{o,\emptyset}$) and those that are extended by an additional fact in its preconditions and add effects ($\omega_{o,f}$). The reason is that it, in our opinion, considerably simplifies the proofs, because it is more obvious how $\omega_{o,\emptyset}$ differs from its extensions $\omega_{o,f}$.

The main result of this section, stating that every mutex pair that is part of a fam-group is an h^2-mutex, is stated in Theorem 31 which is preceded by two auxiliary lemmas.

Lemma 29. *Let $\Sigma_X = \{\phi_c \mid c \subseteq X, |c| = 2\}$, where $X \subseteq \mathcal{F}$, and let $A, B \subseteq \mathcal{F}$. $\Sigma_A \cap \Sigma_B = \Sigma_{A \cap B}$.*

Proof. (By contradiction) If $\Sigma_A \cap \Sigma_B \neq \Sigma_{A\cap B}$ then two cases must be investigated: *(i)* There exists $\phi_{\{f_1,f_2\}}$ such that $\phi_{\{f_1,f_2\}} \in \Sigma_A \cap \Sigma_B$ and $\phi_{\{f_1,f_2\}} \notin \Sigma_{A\cap B}$. So it follows that $f_1, f_2 \in A$ and $f_1, f_2 \in B$ and $f_1, f_2 \notin A \cap B$, which is contradiction. *(ii)* There exists $\phi_{\{f_1,f_2\}}$ such that $\phi_{\{f_1,f_2\}} \notin \Sigma_A \cap \Sigma_B$ and $\phi_{\{f_1,f_2\}} \in \Sigma_{A\cap B}$. So it follows that $f_1, f_2 \in A \cap B$, therefore $f_1, f_2 \in A$ and $f_1, f_2 \in B$, therefore $\phi_{\{f_1,f_2\}} \in \Sigma_A$ and $\phi_{\{f_1,f_2\}} \in \Sigma_B$, which is contradiction. □

Lemma 30. *Let $\Sigma_X = \{\phi_c \mid c \subseteq X, |c| = 2\}$, where $X \subseteq \mathcal{F}$, and let $M \subseteq \mathcal{F}$ denote a set of facts in Π such that $|M| \geq 2$. If M is a fam-group then $|\Sigma_M \cap \psi_{init}| = 0$ and for every operator $\omega \in \Omega$ it holds that $|\Sigma_M \cap \text{add}(\omega)| \leq |\Sigma_M \cap \text{pre}(\omega)|$.*

Proof. Since ψ_{init} is created from s_{init} and $|M \cap s_{init}| \leq 1$ then it is easy to see that $|\Sigma_M \cap \psi_{init}| = 0$ must hold.

Now we prove $|\Sigma_M \cap \text{add}(\omega)| \leq |\Sigma_M \cap \text{pre}(\omega)|$ separately for operators $\omega_{o,\emptyset}$ and for the rest of the operators. But first, we start with some preliminaries. It is easy to see that given a set of facts $A \subseteq \mathcal{F}$: $|\Sigma_A| = C_2(|A|)$, where $C_k(n) = \frac{n!}{k!(n-k)!}$ is a binomial coefficient for $n \geq k \geq 0$ and $C_k(n) = 0$ for $0 \leq n < k$. Let $\text{pre}^2(\omega) = \{\phi_c \mid \phi_c \in \text{pre}(\omega), |c| = 2\}$, and $\text{add}^2(\omega) = \{\phi_c \mid \phi_c \in \text{add}(\omega), |c| = 2\}$. Since Σ_M contains only facts ϕ_c where $|c| = 2$, the inequality $|\Sigma_M \cap \text{add}(\omega)| \leq |\Sigma_M \cap \text{pre}(\omega)|$ holds iff $\left|\Sigma_M \cap \text{add}^2(\omega)\right| \leq \left|\Sigma_M \cap \text{pre}^2(\omega)\right|$ holds.

From Definition 28 it follows that for every operator $o \in \mathcal{O}$: $\text{pre}^2(\omega_{o,\emptyset}) = \Sigma_{\text{pre}(o)}$ and $\text{add}^2(\omega_{o,\emptyset}) = \Sigma_{\text{add}(o)}$, therefore we need to prove that $\left|\Sigma_M \cap \Sigma_{\text{add}(o)}\right| \leq \left|\Sigma_M \cap \Sigma_{\text{pre}(o)}\right|$, which can be rewritten as $\left|\Sigma_{M\cap\text{add}(o)}\right| \leq \left|\Sigma_{M\cap\text{pre}(o)}\right|$ (Lemma 29), and further $C_2(|M \cap$

$\mathrm{add}(o)|) \leq C_2(|M \cap \mathrm{pre}(o)|)$. Since M is a fam-group, $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$, which implies $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o)|$, therefore inequality $C_2(|M \cap \mathrm{add}(o)|) \leq C_2(|M \cap \mathrm{pre}(o)|)$ holds, because $C_2(n)$ is an increasing function.

Let $\Gamma_{\mathrm{pre}(o),f} = \{\phi_{\{g,f\}} \mid g \in \mathrm{pre}(o), g \neq f\}$, and $\Gamma_{\mathrm{add}(o),f} = \{\phi_{\{g,f\}} \mid g \in \mathrm{add}(o)\}$. For the remaining operators $\omega_{o,f} \in \Omega \setminus \{\omega_{o,\emptyset} \mid o \in \mathcal{O}\}$ it holds that $\mathrm{pre}^2(\omega_{o,f}) = \Sigma_{\mathrm{pre}(o)} \cup \Gamma_{\mathrm{pre}(o),f}$ and $\mathrm{add}^2(\omega_{o,f}) = \Sigma_{\mathrm{add}(o)} \cup \Gamma_{\mathrm{add}(o),f}$. Now two cases need to be investigated.

(1) If $f \notin M$ then obviously $\Sigma_M \cap \Gamma_{\mathrm{add}(o),f} = \Sigma_M \cap \Gamma_{\mathrm{pre}(o),f} = \emptyset$, therefore $\Sigma_M \cap \mathrm{add}^2(\omega_{o,f}) = \Sigma_M \cap \mathrm{add}^2(\omega_{o,\emptyset})$ and $\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f}) = \Sigma_M \cap \mathrm{pre}^2(\omega_{o,\emptyset})$. So it follows that $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,f})\right| \leq \left|\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})\right|$, because $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,\emptyset})\right| \leq \left|\Sigma_M \cap \mathrm{pre}^2(\omega_{o,\emptyset})\right|$.

(2) If $f \in M$ then $\left|\Sigma_M \cap \Gamma_{\mathrm{add}(o),f}\right| = |M \cap \mathrm{add}(o)|$, because $f \notin \mathrm{add}(o)$ by definition. And $\left|\Sigma_M \cap (\Sigma_{\mathrm{add}(o)} \cup \Gamma_{\mathrm{add}(o),f})\right| = \left|\Sigma_M \cap \Sigma_{\mathrm{add}(o)}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{add}(o),f}\right|$, because $\Sigma_{\mathrm{add}(o)}$ and $\Gamma_{\mathrm{add}(o),f}$ are disjunct. Now two more cases need to be investigated.

(2.1) If $f \notin \mathrm{pre}(o)$ then $\left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right| = |M \cap \mathrm{pre}(o)|$, therefore $\left|\Sigma_M \cap \Gamma_{\mathrm{add}(o),f}\right| \leq \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$, because $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o)|$. Furthermore, since $f \notin \mathrm{pre}(o)$, $\left|\Sigma_M \cap (\Sigma_{\mathrm{pre}(o)} \cup \Gamma_{\mathrm{pre}(o),f})\right| = \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$, because $\Sigma_{\mathrm{pre}(o)}$ and $\Gamma_{\mathrm{pre}(o),f}$ are disjunct. So it follows from $\left|\Sigma_M \cap \Sigma_{\mathrm{add}(o)}\right| \leq \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)}\right|$ and $\left|\Sigma_M \cap \Gamma_{\mathrm{add}(o),f}\right| \leq \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$, that $\left|\Sigma_M \cap \Sigma_{\mathrm{add}(o)}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{add}(o),f}\right| \leq \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$, therefore $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,f})\right| \leq \left|\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})\right|$.

(2.2) If $f \in \mathrm{pre}(o)$ then $\mathrm{pre}^2(\omega_{o,f}) = \mathrm{pre}^2(\omega_{o,\emptyset}) = \Sigma_{\mathrm{pre}(o)} = \Sigma_{\mathrm{pre}(o)\setminus\{f\}} \cup \Gamma_{\mathrm{pre}(o),f}$, and $\left|\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})\right| = \left|\Sigma_M \cap (\Sigma_{\mathrm{pre}(o)\setminus\{f\}} \cup \Gamma_{\mathrm{pre}(o),f})\right| = \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)\setminus\{f\}}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$ ($\Sigma_{\mathrm{pre}(o)\setminus\{f\}}$ and $\Gamma_{\mathrm{pre}(o),f}$ are disjunct), and $\left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right| = |M \cap (\mathrm{pre}(o) \setminus \{f\})|$. Also $|M \cap \mathrm{add}(o)| \leq |M \cap (\mathrm{pre}(o) \setminus \{f\})|$, because $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ and $f \notin \mathrm{add}(o)$ and $f \notin \mathrm{del}(o)$ (Definition 28). Therefore similarly to *(2.1)*, $\left|\Sigma_M \cap \Sigma_{\mathrm{add}(o)}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{add}(o),f}\right| \leq \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)\setminus\{f\}}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$, therefore $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,f})\right| \leq \left|\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})\right|$. □

Theorem 31. *Let $M \subseteq \mathcal{F}$ denote a set of facts such that $|M| \geq 2$ and let $H = \{p \mid p \subseteq M, |p| = 2\}$. If M is a fam-group, then every $h \in H$ is an h^2-mutex.*

Proof. *(By induction)* Let Ψ denote a set of all reachable states in Π^2 and let $\Sigma_M = \{\phi_h \mid h \in H\}$. It follows from Lemma 30 that $|\Sigma_M \cap \psi_{init}| = 0$. Now, we need to prove that for any reachable state $\psi \in \Psi$ and every operator $\omega \in \Omega$ applicable in ψ it holds that if $|\Sigma_M \cap \psi| = 0$, then $|\Sigma_M \cap \omega[\psi]| = 0$. For every operator ω applicable in ψ it holds that $\mathrm{pre}(\omega) \subseteq \psi$ and, therefore, $|\Sigma_M \cap \mathrm{pre}(\omega)| = 0$. So it follows from Lemma 30 that also $|\Sigma_M \cap \mathrm{add}(\omega)| = 0$ because $|\Sigma_M \cap \mathrm{add}(\omega)| \leq |\Sigma_M \cap \mathrm{pre}(\omega)|$. Finally, since $\omega[\psi] = (\psi \setminus \mathrm{del}(\omega)) \cup \mathrm{add}(\omega)$ it must follow that $|\Sigma_M \cap \omega[\psi]| = 0$, therefore, there is no reachable state $\psi \in \Psi$ containing any fact from Σ_M which means that every $h \in H$ is an h^2-mutex. □

To show that the implication in the other direction than stated in Theorem 31 does not hold, i.e., that there can be an h^2-mutex that is not a subset of any fam-group, we can get back to our example planning task. The pair of facts {`(carry-food)`, `(fed)`} is an h^2-mutex, but this pair of facts cannot be part of any fam-group, because `(carry-food)` cannot be balanced in the operator `take-food` by any other fact since `take-food` has empty delete effects.

In Section 5, we described how mutex groups can be inferred in regression and in the same way can be inferred h^2-mutexes. For any planning task, the corresponding dual planning task (Definition 10) can be constructed and h^2-mutexes inferred there are, therefore, invariants with respect to reachable states in the dual planning tasks. As Theorem 31 describes the relationship between fam-groups and h^2-mutexes in progression, the theorem also holds for fam-groups and h^2-mutexes in the corresponding dual planning task.

8. Inference of Fact-Alternating Mutex Groups

In this section, we describe an algorithm for the inference of fam-groups. The main part of the algorithm consists of an integer linear program (ILP) based on the definition of fact-alternating mutex groups (Definition 4) rewritten into a set of constraints. The ILP is constructed in the following way.

Each variable x_i of the ILP corresponds to a fact $f_i \in \mathcal{F}$ from the planning task. Variables can acquire binary values 0 or 1 only, 0 meaning that the corresponding fact is not present in the fam-group and 1 meaning the corresponding fact is part of the fam-group. For example having three facts f_1, f_2, f_3, the corresponding ILP would consist of three binary variables x_1, x_2, x_3 and an assignment of the variables $x_1 = 1, x_2 = 0, x_3 = 1$ would mean that the fam-group M consists of facts f_1 and f_3 ($M = \{f_1, f_3\}$).

The definition of a fact-alternating mutex group can be rewritten into ILP constraints as follows:

$$\sum_{f_i \in s_{init}} x_i \leq 1, \tag{1}$$

$$\forall o \in \mathcal{O} : \sum_{f_i \in \mathrm{add}(o)} x_i \leq \sum_{f_i \in \mathrm{del}(o) \cap \mathrm{pre}(o)} x_i. \tag{2}$$

Equation (1) is a constraint saying that the initial state must have at most one common fact with the fam-group and corresponds to the first condition in Definition 4 ($|M \cap s_{init}| \leq 1$). Equation (2) corresponds to the second part of the definition and it ensures that the mutex property is preserved by all operators ($|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$).

The objective function of the ILP is to maximize $\sum_{f_i \in \mathcal{F}} x_i$. The maximization enforces the inference of a fam-group containing the maximum possible number of facts.

Unfortunately, the solution to this ILP is only one fam-group, so some mechanism enabling inference of all fact-alternating mutex groups is required. This drawback can be resolved by solving the ILP repeatedly, each time with added constraints that exclude already inferred fam-groups. Let M denote a known fam-group. Such a fam-group and all its subsets can be excluded from the ILP solution by adding the constraint

$$\sum_{f_i \notin M} x_i \geq 1. \tag{3}$$

The constraint forces the ILP solver to add a fact to the solution that is not present in the known fam-group M and, thus, excluding M and all its subsets. In other words, since we are not interested in the fam-group M and its subsets, we know that any other fam-group must contain a fact that is not a part of M.

The whole fam-group inferring algorithm is encapsulated in Algorithm 2. First, the ILP constraints are constructed according to Equations 1 and 2, which ensures that the

Algorithm 2: Inference of fact-alternating mutex groups using ILP.

Input: Planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$
Output: A set of fam-groups $\mathcal{M}$
Initialize ILP with constraints according to Equations 1 and 2;
Set objective function of ILP to maximize $\sum_{f_i \in \mathcal{F}} x_i$;
Solve ILP and save the resulting fam-group into M;
while $|M| \geq 1$ **do**
    Add M to the output set $\mathcal{M}$;
    Add constraint according to Equation (3) using M;
    $M \leftarrow \emptyset$;
    Solve ILP and if a solution was found, save the resulting fam-group into M;
end

solutions of the ILP will be fact-alternating mutex groups. Then, in turn, a maximal fam-group is inferred through the ILP solution and consequently removed from future solutions using the added constraint corresponding to Equation (3). The cycle continues until the inferred fam-groups consist of, at least, one fact. Since a maximal fam-group is produced at each step, arriving at smaller and smaller fam-groups means that the algorithm eventually terminates. The combination of the maximization and removal of the found fam-groups and all theirs subsets from the solutions in the following steps also ensures that every produced fam-group is unique and it is never a subset of any already found fam-group.

Theorem 32. *Algorithm 2 is complete with respect to the maximal fact-alternating mutex groups.*

Proof. To prove Theorem 32 by contradiction, let us assume that Algorithm 2 was terminated and it produced a set of fam-groups, and let us assume that there exists a fam-group M that is not a subset of any fam-group produced by Algorithm 2. Such a fam-group must satisfy the constraints expressed by Equations 1 and 2 and it must contain, at least, one fact that is not part of any fam-group produced by Algorithm 2. This is not violated by the ILP constraints in the last cycle of Algorithm 2 because they consist of Equations 1 and 2 and a set of Equation (3) constraints that force the next fam-group to include a fact that is not part of any fam-group found so far. This means that Algorithm 2 could not terminate, thus, such an M does not exist and Algorithm 2 is complete with respect to maximal fact-alternating mutex groups. $\square$

Note that Equation (3) can be used for the exclusion of any set of facts. This means that Algorithm 2 can be initialized with any set of mutex groups obtained by any other method. Therefore, if there is a faster but incomplete alternative available for the inference of fam-groups, the fam-groups inferred by that method can be used for the initialization of Algorithm 2. This could speed up the running time of the inference algorithm while preserving its completeness.

Furthermore, the algorithm can be easily altered to an any-time algorithm just by setting a limit on the number of cycles or by the premature stopping of the computation, because the algorithm produces one correct fam-group per cycle.

Algorithm 3: Pruning of a planning task using inferred fam-groups.

Input: Planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$
Output: A planning task $\Pi^\star = \langle \mathcal{F}^\star, \mathcal{O}^\star, s^\star_{init}, s_{goal} \rangle$, a set of fam-groups $\mathcal{M}$

$\mathcal{F}^\star \leftarrow \mathcal{F}$, $\mathcal{O}^\star \leftarrow \mathcal{O}$, $s^\star_{init} \leftarrow s_{init}$;
do
    Remove facts returned by `IrrelevantFacts(`$\mathcal{F}^\star, \mathcal{O}^\star, s_{goal}$`)` from $\mathcal{F}^\star$, $s^\star_{init}$ and all operators in $\mathcal{O}^\star$;
    Use Algorithm 2 with $\Pi^\star$ and store the inferred fam-groups into $\mathcal{M}$;
    for each $o \in \mathcal{O}^\star$ *and* $M \in \mathcal{M}$ **do**
        if $|pre(o) \cap M| \geq 2$ *or* $|add(o) \cap M| \geq 2$ **then** Remove o from $\mathcal{O}^\star$;
    end
    for each $o \in \mathcal{O}^\star$ *and* $M \in \mathcal{M}$ *such that* $|M \cap s_{goal}| \geq 1$ **do**
        if $|M \cap pre(o) \cap del(o)| \geq 1$ *and* $|M \cap add(o)| = 0$ **then** Remove o from $\mathcal{O}^\star$;
    end
while *change in* $\mathcal{F}^\star$ *or* $\mathcal{O}^\star$ *occurred*;

function `IrrelevantFacts(`$\mathcal{F}, \mathcal{O}, s_{goal}$`)`
    $U \leftarrow s_{goal}$;
    $R \leftarrow \emptyset$;
    while $|U| > 0$ **do**
        $f \leftarrow$ choose any fact from U;
        $U \leftarrow U \setminus \{f\}$;
        if $f \notin R$ **then**
            $R \leftarrow R \cup \{f\}$;
            for each $o \in \mathcal{O}$ *such that* $f \in (add(o) \cup del(o))$ **do**
                $U \leftarrow U \cup \{g \mid g \in \mathrm{pre}(o),\ g \notin R\}$;
            end
        end
    end
    return $\mathcal{F} \setminus R$;
end

9. Pruning of Planning Tasks

In Section 4, we have described how mutex groups and fam-groups can be used for the pruning of operators and facts in planning tasks as a preprocessing step. Any mutex group can be used for the removal of unreachable operators by checking its preconditions and add effects. Moreover, fam-groups can be used for the removal of superfluous operators that can produce dead-end states only (i.e., the removal of dead-end operators). The algorithm for the pruning of planning tasks we propose is encapsulated in Algorithm 3.

The algorithm repeatedly removes facts and operators until a fixpoint is reached when no more facts or operators can be removed. In each cycle, the irrelevant facts are detected and removed, then fam-groups are inferred and they are, in turn, used for the removal of unreachable operators and dead-end operators.

The detection and removal of irrelevant facts (line 3) follows the idea used by Helmert (2006) for removal of variables in finite domain representation of a planning task. First, a causal graph of facts is constructed, i.e., a directed graph with facts represented by nodes and an edge $f_1 \rightarrow f_2$ connecting every pair of facts f_1 and f_2 if $f_1 \in \mathrm{pre}(o)$ and $f_2 \in \mathrm{add}(o) \cup \mathrm{del}(o)$ for some operator o. Then, a fact in the causal graph, from which there is no path leading to a goal fact, is not useful for finding a plan because it takes no part in the applicability of the operators in reachable states. Such a fact can be safely removed from the planning task. The function `IrrelevantFacts` listed in Algorithm 3 shows how irrelevant facts can be found without actually constructing a causal graph.

The inferred fam-groups (line 4) are used for the removal of unreachable operators (lines 5–7) which are those that cannot be applied in any reachable state as already discussed in Section 4. Finally, fam-groups containing a fact from the goal specification are used for removal of dead-end operators (lines 8–10) according to Corollary 8.

Algorithm 3 can be also used with any other type of mutex group, i.e., line 4 can be replaced by any other algorithm that provides a set of mutex groups, but the removal of dead-end operators (lines 8–10) cannot be used, because Corollary 8 does not generally hold for any type of mutex group.

10. Experimental Results

All algorithms experimentally evaluated in this section were implemented in the Fast Downward's preprocessor[3] (Helmert, 2006) written in Python programming language. Algorithm 2 and the h^2 heuristic were implemented as C extensions for Python. The experiments were run on a computer with an Intel Xeon E5-4617 2.9GHz processor and a memory limit set to 8 GB RAM. The algorithms were evaluated on all domains from the optimal deterministic track of the International Planning Competition (IPC) 2011 and 2014 that do not contain any conditional effects after grounding (i.e., all except the Citycar domain from IPC 2014). The domains that contain negative preconditions were also included because all algorithms presented here work with them without change.

The algorithms proposed in this work are compared with two different methods for inferring mutex groups that were already discussed in Section 2. One is the inference algorithm implemented in the Fast Downward's preprocessor (Helmert, 2009) that will be abbreviated by `fd` from now on. Helmert's algorithm was used by most planners that participated in the last IPC 2014 in the deterministic track. Therefore, the comparison with `fd` is certainly relevant to the planning community. The time and memory limits of `fd` set in Fast Downward were disabled. The other state-of-the-art algorithm that we use for comparison is the $\mathtt{h}^2$ heuristic (Haslum & Geffner, 2000). The relationship between h^2-mutexes and fam-groups was already discussed in Section 7.

The algorithm for inference of fam-groups (Algorithm 2) was implemented using a CPLEX ILP solver (v12.6.1.0) running with default configuration in one thread. We will refer to this algorithm as `fa`.

The presented algorithms are experimentally evaluated in several different ways. The algorithms are compared in terms of mutex pairs (Section 10.1), because the decomposition of the inferred mutex groups into a set of mutex pairs allows us to compare the

3. https://github.com/danfis/fast-downward, branch jair-fa-mutex

algorithms without considering the differences in the shapes and sizes of the mutex groups. In Section 10.2, the algorithms are compared with respect to the mutex groups as they were inferred. In Section 10.3, the running times of the inference algorithms are compared and a faster version of `fa` is introduced. Utilization of mutex groups in a translation to finite domain representation is evaluated in Section 10.4. In Section 10.5, the algorithms are compared in the context of pruning of planning tasks (Algorithm 3) and the impact of pruning on the coverage over all tested domains is evaluated. And lasty, comparison with a state-of-the-art pruning method that uses inference of h^2-mutexes in both progression and regression is provided in Section 10.6.

10.1 Comparison in Terms of Mutex Pairs

Any mutex group can be decomposed into a set of mutex pairs by enumerating all pairs of facts the original mutex group consists of. Such a decomposition provides a common base for comparing the algorithms in terms of inferred mutex groups. For example, $\mathtt{h}^2$ is able to produce mutex pairs only, but `fa` is designed to produce maximal fam-groups. The pair decomposition provides a transparent method for comparing these two and all other algorithms for the inference of mutex groups.

On the other hand, this method of comparison clouds the fact that `fd` and `fa` both provide a richer structure than just a set of mutex pairs. Although it is always possible to reconstruct back any mutex group from its pair decomposition by using some algorithm for enumerating maximal cliques, it must be taken into account that the reconstruction alone is NP-Hard and it can generate an exponential number of mutex groups (i.e., possibly many more besides the original ones that were used for the decomposition). The significance of having richer mutex group sets than just pairs of facts is discussed in more depth in the following sections.

All mutex groups inferred by `fd`, and `fa` were decomposed into mutex pairs ($\mathtt{h}^2$ generates mutex pairs, so decomposition was not necessary). Table 2 shows the sums and ratios of a number of inferred mutex pairs per domain and overall by all tested algorithms, the maximum numbers are highlighted (the column #ps shows the number of tasks with the corresponding domain).

`fd` had the poorest performance with 1 340 085 inferred mutex pairs overall. The highest number of mutex pairs was inferred by $\mathtt{h}^2$ (12 919 402) and decomposition of mutex groups inferred by `fa` results in 5 612 542 mutex pairs overall. This seems as a huge lead for $\mathtt{h}^2$ before all other algorithms, but careful investigation of Table 2 shows us that most of the margin is disproportionately made in a single domain `tetris14`. The reason for this is that the domain `tetris14` models a grid of positions and several objects of different shapes that can occupy the grid. After grounding, the tasks from the domain `tetris14` contain a high number of facts that have strong restrictions on their co-occurrence in the grid, therefore they contain a high number of mutex pairs. In this particular case, $\mathtt{h}^2$ is much more effective in inferring these mutex pairs than any other algorithm, but `fa` is still much more effective than the widely used algorithm `fd`. Without the domain `tetris14` the numbers are 1 317 533, 1 644 252, and 1 922 128 for `fd`, `fa`, and $\mathtt{h}^2$, respectively. This shows that $\mathtt{h}^2$ has still the best performance, but with a much smaller margin than `fa`.

domain	#ps	#mutex pairs			ratio		
		fd	**$\mathbf{h^2}$**	**fa**	**fd**	**$\mathbf{h^2}$**	**fa**
childsnack14	20	3 194	3 194	3 194	1.00	1.00	1.00
elevators11	20	11 598	11 598	11 598	1.00	1.00	1.00
floortile11	20	28 366	28 366	28 366	1.00	1.00	1.00
floortile14	20	17 572	17 572	17 572	1.00	1.00	1.00
hiking14	20	2 505	2 505	2 505	1.00	1.00	1.00
transport11	20	20 344	20 344	20 344	1.00	1.00	1.00
transport14	20	114 168	114 168	114 168	1.00	1.00	1.00
visitall11	20	39 468	39 468	39 468	1.00	1.00	1.00
visitall14	20	227 268	227 268	227 268	1.00	1.00	1.00
barman11	20	1 816	**12 640**	11 012	0.14	1.00	0.87
barman14	20	1 792	**13 345**	11 645	0.13	1.00	0.87
cavediving14	20	6 304	**67 847**	61 614	0.09	1.00	0.91
ged14	20	48 452	**69 564**	68 326	0.70	1.00	0.98
maintenance14	20	0	**1 039**	**1 039**	0.00	1.00	1.00
nomystery11	20	232 677	**280 875**	232 677	0.83	1.00	0.83
openstacks11	20	5 890	**7 940**	5 890	0.74	1.00	0.74
openstacks14	20	16 085	**21 340**	16 085	0.75	1.00	0.75
parcprinter11	20	15 255	**50 162**	29 235	0.30	1.00	0.58
parking11	20	213 540	**312 550**	213 540	0.68	1.00	0.68
parking14	20	178 720	**260 880**	178 720	0.69	1.00	0.69
pegsol11	20	11 880	**13 571**	12 202	0.88	1.00	0.90
scanalyzer11	20	31 952	**33 488**	33 440	0.95	1.00	1.00
sokoban11	20	85 222	**89 519**	85 241	0.95	1.00	0.95
tetris14	20	22 552	**10 997 274**	3 968 290	0.00	1.00	0.36
tidybot11	20	240	**82 248**	**82 248**	0.00	1.00	1.00
tidybot14	20	240	**133 744**	**133 744**	0.00	1.00	1.00
woodworking11	20	2 985	**6 893**	3 111	0.43	1.00	0.45
Σ	540	1 340 085	**12 919 402**	5 612 542	0.10	1.00	0.43
$\Sigma \setminus$ tetris14	520	1 317 533	**1 922 128**	1 644 252	0.69	1.00	0.86

Table 2: Sum and ratio of the number of inferred mutex pairs.

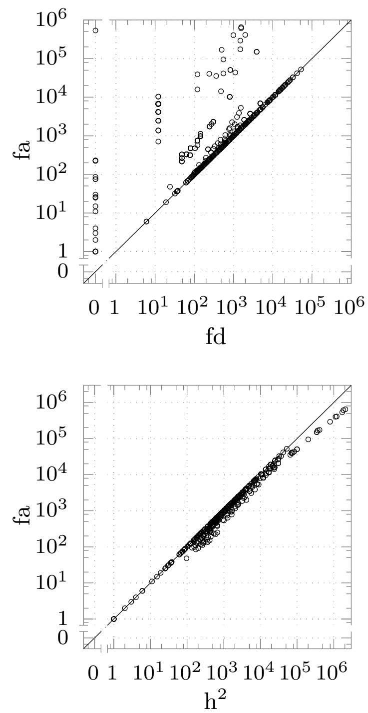


Figure 4: Comparison of the number of inferred mutex pairs in each planning task.

The ratios of a number of mutex pairs show that `fa` has very similar results as $\mathtt{h}^2$ in most domains except the aforementioned `tetris14` and also in `woodworking11`, `parcprinter11`, `parking11`, and `parking14` (although the difference is smaller). The small difference between $\mathtt{h}^2$ and `fa` can also be seen in the bottom scatter plot in Figure 4 showing that $\mathtt{h}^2$ and `fa` produce similar results in most planning tasks.

The top scatter plot in Figure 4 shows that `fa` produces at least as many mutex pairs as `fd` in every single planning task. The relative difference between `fa` and `fd` is much higher than between `fa` and $\mathtt{h}^2$ which is a promising result considering that `fd` unlike $\mathtt{h}^2$ is able to produce mutex groups consisting of more than two facts. An interesting result is that `fa` produced at least one mutex group in every tested planning task whereas `fd` did not generate any mutex group in 21 planning tasks (the whole `maintenance14` domain and one planning task from `tetris14`).

10.2 Comparison of Inferred Mutexes

In the previous section, we provided an analysis of the algorithms for inference of mutex groups in terms of mutex pairs that were obtained by decomposition of the actual inferred mutex groups. As mentioned before, this type of analysis disregards the fact that the mutex groups consisting of more than two facts can provide more useful information than those formed by just a pair of facts. A translation from PDDL to FDR is one of the applications

domain	**#ps**	**fd**	**h^{2*}**	**fa**	**fa≻fd**	**h^{2*}≻fd**	**h^{2*}≻fa**
childsnack14	20	618	618	618	0	0	0
elevators11	20	245	245	245	0	0	0
floortile11	20	624	624	624	0	0	0
floortile14	20	575	575	575	0	0	0
hiking14	20	229	229	229	0	0	0
transport11	20	217	217	217	0	0	0
transport14	20	206	206	206	0	0	0
visitall11	20	20	20	20	0	0	0
visitall14	20	20	20	20	0	0	0
barman11	20	208	**1 596**	504	20	20	20
barman14	20	206	**1 663**	498	20	20	20
cavediving14	20	184	**3 004**	800	20	20	20
ged14	20	555	**6 261**	555	20	20	20
maintenance14	20	0	**460**	**460**	20	20	0
nomystery11	20	190	**3 874**	190	0	20	20
openstacks11	20	800	**2 850**	800	0	20	20
openstacks14	20	730	**5 985**	730	0	20	20
parcprinter11	20	105	**3 426**	1 118	20	20	20
parking11	20	870	**16 590**	870	0	20	20
parking14	20	820	**13 860**	820	0	20	20
pegsol11	20	680	**67 617**	699	3	20	20
scanalyzer11	20	392	**680**	432	5	5	1
sokoban11	20	983	**1 403**	985	1	20	20
tetris14	20	52	**7 947**	676	20	20	20
tidybot11	20	60	**200**	**200**	20	20	0
tidybot14	20	60	**200**	**200**	20	20	0
woodworking11	20	632	**1 796**	721	13	20	20
Σ	540	10 281	**142 166**	14 012	202	345	281

Table 3: Number of inferred mutex groups.

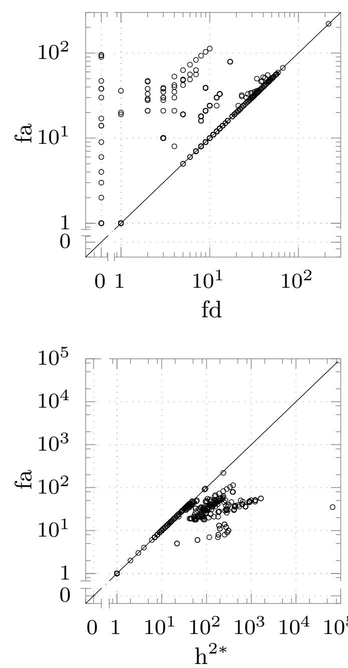


Figure 5: Comparison of the number of inferred mutex groups in each planning task.

for which larger mutex groups are desirable. Other possible applications were discussed in Section 4 and there are possibly more to be discovered given the tight relationship between the satisfiability of planning tasks and the inference of maximum mutex groups as described in Section 6.2.

In this section, we compare the inferred mutex groups as they are produced by the algorithms `fd` and `fa`. Since $\mathtt{h}^2$ can produce mutex pairs only, which we have compared in the previous section, we decided to use an algorithm for enumerating all maximal cliques (Bron & Kerbosch, 1973), implemented in the NetworkX Python library (Hagberg, Schult, & Swart, 2008), for construction of maximal mutex groups from the mutex pairs generated by $\mathtt{h}^2$. This modified algorithm will be denoted by $\mathtt{h}^{2\star}$.

The sums of a number of inferred mutex groups by all three algorithms are listed in Table 3, the maximal values are highlighted. The table also contains columns labeled as `a`≻`b` showing the number of planning tasks in which algorithm `a` generated a richer set of mutex groups than `b`, i.e., the number of planning tasks in which `a` inferred more mutex groups than `b` or the number of inferred mutex groups was the same but some mutex group inferred by `a` was a proper superset of some mutex group inferred by `b`. The latter happened only in a domain `ged14` where `fa` and `fd` generated the same number of mutex groups in every planning task, but the mutex groups inferred by `fa` contained more facts.

Every mutex group that was generated by `fd` was also generated by `fa` or it was a subset of some mutex group generated by `fa` (only in the case of `ged14`). In 202 out of 540 planning tasks, `fa` generated a richer set of mutex groups. These results are also well documented on the top scatter plot in Figure 5. All of the 3 731 mutex groups that were generated by `fa` on top of those generated by `fd` were not supersets of any mutex group generated by `fd`. In other words, the whole difference between the number of inferred mutex groups by `fa` and `fd` corresponds to the new mutex groups that are not just extensions of the mutex groups generated by `fd`.

Since a set of h^2-mutexes is always a superset of decompositions of all fam-groups (Theorem 31), $\mathtt{h}^{2\star}$ must always generate a richer set of mutex groups than `fa` (and, thus, also than `fd`) which is also reflected in Table 3. Many of the mutex groups inferred by $\mathtt{h}^{2\star}$ are supersets of `fa` mutex groups that differ in a couple of facts only. The main source of the big difference between the number of mutex groups inferred by $\mathtt{h}^{2\star}$ (142 166) and `fa` (14 012) is caused by the fact that the maximum possible number of mutex groups grows exponentially which follows from the tight relationship between mutex groups and graph cliques (Proposition 27). This aspect of the comparison is clearly visible if we compare the bottom scatter plot in Figure 5, showing the number of mutex groups, with the bottom scatter plot in Figure 4, depicting the number of mutex pairs. The comparison shows how the relatively small difference between the number of mutex pairs translates into a more substantial difference in the number of maximal mutex groups. Moreover, similarly to $\mathtt{h}^{2\star}$, we can decompose fam-groups inferred by `fa` into mutex pairs and then use an algorithm for enumerating maximal cliques for the construction of new mutex groups. This approach just generates the original fam-groups in most cases, but, for example, in the domain `tetris14`, the resulting number of mutex groups is 141 852 (which would, in sum, exceed the results for $\mathtt{h}^{2\star}$) for similar reasons why the number of mutex groups inferred by $\mathtt{h}^{2\star}$ is so high in comparison to `fa`.

For these reasons, we do not consider this type of analysis to be sufficient for a comparison of $\mathtt{h}^{2\star}$ with `fa` (or `fd`). Therefore, we have decided to borrow a well established concept from graph theory called the *clique cover number*. The clique cover number is the minimum number of cliques that cover all the nodes of a graph. Similarly to this, we use the *mutex group cover number* (or just *cover number* for short) as the minimum number of mutex groups that cover all facts of a planning task. None of the tested algorithms generates single facts as mutex groups, but since every single fact is a mutex group by definition, we add them artificially as if they were generated by the corresponding algorithm solely for the purpose of the computation of the mutex group cover number if we need them for covering the facts that are not covered by any other generated mutex group. To demonstrate a mutex group cover number on an example, consider a planning task with five facts $\{f_1, f_2, f_3, f_4, f_5\}$ and suppose that `fd` generates a single mutex group $\{f_1, f_2\}$, `fa` generates two mutex groups $\{f_1, f_2\}$ and $\{f_2, f_3, f_4\}$, and $\mathtt{h}^{2\star}$ generates $\{f_1, f_2, f_3, f_4, f_5\}$. In this case the cover number for `fd` would be 4, for `fa` it would be 3 and for $\mathtt{h}^{2\star}$ it would be only 1.

When comparing cover numbers, the smaller is better because the mutual exclusion between the facts can be described more concisely by a smaller number of mutex groups. The cover number can be also interpreted in the context of finite domain representation as the minimum number of variables that can be used for the full description of all reachable states given the inferred mutex groups by the corresponding algorithm.

The sum of the mutex group cover numbers for each domain and overall is listed in Table 4 and, as in the previous case, the table also includes the number of planning tasks in which one algorithm achieved a smaller cover number than the other one. The table

domain	**#ps**	**fd**	**$h^{2\star}$**	**fa**	**fa≻fd**	**$h^{2\star}$≻fd**	**$h^{2\star}$≻fa**
childsnack14	20	1 248	1 248	1 248	0	0	0
elevators11	20	245	245	245	0	0	0
floortile11	20	576	576	576	0	0	0
floortile14	20	535	535	535	0	0	0
hiking14	20	229	229	229	0	0	0
nomystery11	20	190	190	190	0	0	0
openstacks11	20	800	800	800	0	0	0
openstacks14	20	1 440	1 440	1 440	0	0	0
parking11	20	870	870	870	0	0	0
parking14	20	820	820	820	0	0	0
pegsol11	20	680	680	680	0	0	0
scanalyzer11	20	432	432	432	0	0	0
sokoban11	20	1 284	1 284	1 284	0	0	0
transport11	20	217	217	217	0	0	0
transport14	20	206	206	206	0	0	0
visitall11	20	1 030	1 030	1 030	0	0	0
visitall14	20	2 454	2 454	2 454	0	0	0
barman11	20	2 164	**555**	584	20	20	20
barman14	20	2 210	**555**	581	20	20	20
cavediving14	20	3 726	**913**	**913**	20	20	0
ged14	20	330	**328**	330	0	2	2
maintenance14	20	1 285	**894**	**894**	20	20	0
parcprinter11	20	2 487	**985**	**985**	20	20	0
tetris14	20	16 672	**560**	**560**	20	20	0
tidybot11	20	5 708	**2 732**	**2 732**	20	20	0
tidybot14	20	7 472	**3 514**	**3 514**	20	20	0
woodworking11	20	1 815	**1 290**	1 804	6	20	20
Σ	540	57 125	**25 582**	26 153	166	182	62

Table 4: Sum of mutex group cover numbers.

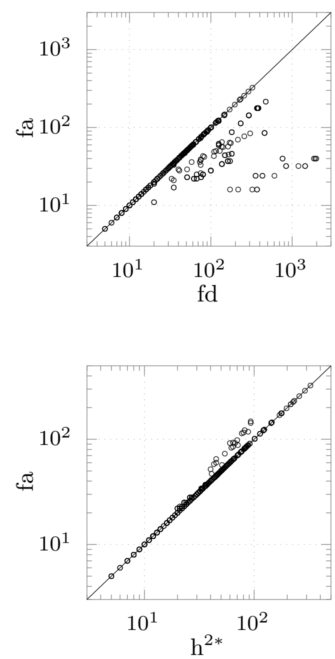


Figure 6: Comparison of mutex group cover numbers in each planning task.

shows that the cover number of `fa` was smaller than of `fd` in 166 planning tasks. If we compare these results with the results from Table 3, we can see that, in 36 planning tasks, `fa` generated more mutex groups than `fd`, but without decreasing the cover number (domains `pegsol11`, `scanalyzer11`, `sokoban11`, `ged14`, and 7 planning tasks from `woodworking11`).

The top scatter plot in Figure 6 clearly shows that the cover number of `fa` is smaller than (or the same as) `fd` in all planning tasks, which was expected given the comparison laid out above. Rather surprising is the fact that the overall cover number of `fd` is more than twice as much as that of `fa` even though `fa` inferred only 50% more mutex groups. This result suggests that if `fa` is used as a replacement for `fd` in a translation from PDDL to FDR, the overall memory footprint of the planner will be significantly reduced.

The difference between the overall mutex group cover number of $\mathtt{h}^{2\star}$ and `fa` is almost negligible. This also corresponds to the bottom scatter plot in Figure 6. $\mathtt{h}^{2\star}$ has a lower cover number than `fa` in 62 out of 540 planning tasks. This means that in a majority of the planning tasks in which $\mathtt{h}^{2\star}$ produced more mutex groups (namely 219), the cover number remained the same for both $\mathtt{h}^{2\star}$ and `fa`.

10.3 Comparison of Running Times

Table 5 shows the sums of running times in seconds of implemented algorithms in each domain and overall and Table 6 shows the minimal and maximal running times. The results show that `fd` is almost 20 times faster than $\mathtt{h}^2$ and more than 120 times faster than

domain	#ps	fd	h^2	$h^{2\star}$	fa	fa_{fd}
barman11	20	**0.54**	1.10	1.37	17.39	15.47
barman14	20	**0.74**	1.28	1.27	17.87	15.55
cavediving14	20	**0.47**	6.30	9.53	263.07	213.89
childsnack14	20	**0.29**	2.51	2.45	62.11	6.64
elevators11	20	**0.18**	0.59	0.79	6.09	3.31
floortile11	20	**0.42**	0.62	1.29	17.30	4.06
floortile14	20	**0.41**	**0.41**	0.66	12.62	3.08
ged14	20	5.91	**1.44**	3.61	109.67	123.71
hiking14	20	**0.70**	4.21	4.10	10.47	7.89
maintenance14	20	**0.07**	0.08	0.10	20.78	22.35
nomystery11	20	**1.23**	8.99	38.87	19.92	9.30
openstacks11	20	**0.55**	0.94	1.02	22.26	4.23
openstacks14	20	**1.07**	3.66	4.38	49.83	7.57
parcprinter11	20	**0.33**	0.62	2.44	100.57	168.33
parking11	20	**0.48**	22.79	63.36	410.20	25.72
parking14	20	**0.66**	17.34	46.95	312.28	22.41
pegsol11	20	**0.21**	0.35	2.15	18.28	3.41
scanalyzer11	20	**2.28**	64.34	64.39	885.05	437.14
sokoban11	20	**0.93**	1.39	6.15	67.90	5.48
tetris14	20	**2.18**	154.50	483 950.68	620.99	629.74
tidybot11	20	**2.35**	49.70	52.36	60.92	55.79
tidybot14	20	**2.93**	93.38	96.10	101.64	88.49
transport11	20	**0.23**	2.07	2.40	10.97	5.03
transport14	20	**0.35**	6.13	14.90	21.00	8.75
visitall11	20	**0.08**	0.35	1.45	0.88	2.72
visitall14	20	**0.27**	2.07	35.33	1.46	2.75
woodworking11	20	**0.87**	1.20	1.22	27.15	10.62
Σ	540	**26.73**	448.37	484 409.30	3 268.67	1 903.43
$\Sigma \setminus$ tetris14	520	**24.55**	293.87	458.63	2 647.68	1 273.69

Table 5: Sum of running times in seconds of inference algorithms.

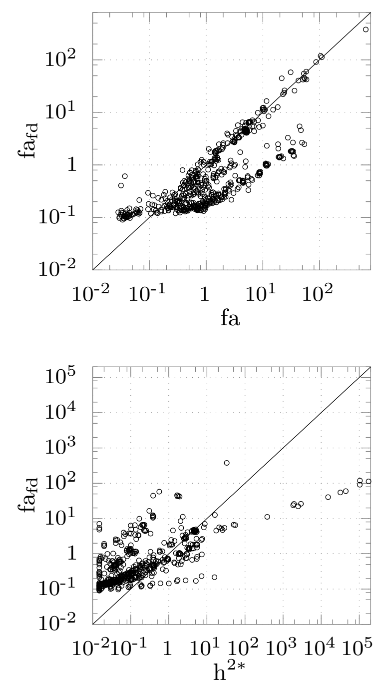


Figure 7: Comparison of running times in each planning task.

fa. The running time of fd never exceeds 2 seconds and the sum of running times over all planning tasks in the data set is only 26.73 seconds. The running time of h^2 does not exceed 10 seconds except for the domains scanalyzer11 and tetris14 and the overall running time does not exceed eight minutes. fa is more than seven times slower than h^2 which is an expected result considering that h^2 is a polynomial algorithm. The slowest inference of fa was measured in the scanalyzer11 domain (885.05 seconds) but most of this time was spent on a single planning task. The overall time that fa spent in all planning tasks combined amounts to almost 55 minutes which corresponds to an average of 6 seconds per planning task, but there is a big variance over the tested domains as Table 6 suggests. The difference between h^2 and fa was expected since fa requires to solve the ILP repeatedly, but h^2 runs in polynomial time. A little surprising is the fact that fd turned out to be the fastest of the tested algorithms by a huge margin, because fd has not guaranteed polynomial running time since it can generate an exponential number of mutex group candidates.

The overall running time of $h^{2\star}$ (Table 5) over all tested planning tasks amounts to more than 100 hours, but almost all of this time is spent in a single domain tetris14. The reason is that tetris14 contains a huge number of h^2-mutexes, which results in the construction of large densely connected graphs in which $h^{2\star}$ must find maximal cliques. Without considering tetris14, the sum of running times is only under eight minutes which is only about 1.5 times more than the inference of h^2-mutexes by h^2 (458.63 seconds for $h^{2\star}$ and 293.87 seconds for h^2). This is a surprising result, because it means that the inference of h^2-mutexes in all domains except tetris14 needed twice as much time as the inference of

domain	#ps	fd		h^2		$h^{2\star}$		fa		fa_{fd}	
		min	max	min	max	min	max	min	max	min	max
barman11	20	0.02	0.03	0.03	0.12	0.03	0.14	0.35	1.58	0.35	1.37
barman14	20	0.03	0.12	0.03	0.20	0.03	0.12	0.39	1.66	0.38	1.39
cavediving14	20	0.01	0.04	0.02	1.24	0.03	1.93	0.49	58.91	0.49	45.58
childsnack14	20	0.01	0.02	0.02	0.43	0.02	0.46	0.63	8.65	0.14	0.63
elevators11	20	0.01	0.01	0.02	0.05	0.02	0.11	0.16	0.54	0.13	0.28
floortile11	20	0.01	0.03	0.01	0.11	0.01	0.21	0.29	2.09	0.12	0.80
floortile14	20	0.02	0.03	0.01	0.03	0.02	0.05	0.33	1.53	0.13	0.19
ged14	20	0.21	0.74	0.01	0.12	0.02	0.39	1.41	10.47	1.65	12.36
hiking14	20	0.01	0.17	0.01	0.89	0.01	0.79	0.11	1.09	0.12	1.15
maintenance14	20	0.00	0.01	0.00	0.01	0.00	0.01	0.03	6.56	0.11	7.09
nomystery11	20	0.01	0.15	0.01	1.51	0.02	7.34	0.13	2.67	0.12	0.96
openstacks11	20	0.01	0.04	0.01	0.10	0.01	0.11	0.36	2.20	0.13	0.30
openstacks14	20	0.02	0.17	0.04	0.41	0.04	0.51	0.55	5.44	0.19	0.62
parcprinter11	20	0.01	0.02	0.01	0.10	0.01	0.56	0.41	31.04	0.50	57.93
parking11	20	0.02	0.04	0.21	2.84	0.38	7.54	4.44	54.30	0.48	2.65
parking14	20	0.02	0.15	0.21	1.92	0.40	5.53	4.39	34.00	0.42	2.67
pegsol11	20	0.01	0.01	0.01	0.02	0.02	1.64	0.69	3.26	0.13	0.62
scanalyzer11	20	0.02	1.32	0.01	44.87	0.02	33.11	0.23	654.01	0.19	379.87
sokoban11	20	0.03	0.28	0.01	0.48	0.02	3.04	0.41	35.72	0.14	1.48
tetris14	20	0.02	0.40	0.33	33.21	2.61	175 640.84	1.71	110.22	1.88	119.36
tidybot11	20	0.06	0.27	0.24	4.84	0.31	4.96	0.56	7.04	0.79	4.41
tidybot14	20	0.10	0.20	2.69	8.64	2.63	8.77	2.88	10.02	2.75	7.06
transport11	20	0.01	0.02	0.02	0.28	0.02	0.32	0.16	1.24	0.13	0.39
transport14	20	0.01	0.03	0.02	0.99	0.02	3.04	0.19	3.23	0.13	1.01
visitall11	20	0.00	0.01	0.00	0.06	0.00	0.32	0.03	0.07	0.09	0.61
visitall14	20	0.00	0.09	0.01	0.47	0.01	15.77	0.04	0.19	0.10	0.23
woodworking11	20	0.03	0.11	0.02	0.11	0.02	0.12	0.43	4.00	0.18	1.75
overall	540	0.00	1.32	0.00	44.87	0.00	175 640.84	0.03	654.01	0.09	379.87
overall \ tetris14	520	0.00	1.32	0.00	44.87	0.00	33.11	0.03	654.01	0.09	379.87

Table 6: Minimal and maximal running times in seconds of inference algorithms.

the maximal mutex groups from these h^2-mutexes even though the inference of h^2-mutexes is polynomial in time, but the inference of the maximal mutex groups is NP-Hard. Therefore, it seems that, in most cases, h^2-mutexes form relatively small or simple structures.

In comparison to $\mathtt{h}^{2\star}$, `fa` requires almost six times more time on all domains except `tetris14`. Considering that both $\mathtt{h}^{2\star}$ and `fa` are NP-Hard (both are implemented using exponential algorithms) and that $\mathtt{h}^{2\star}$ always produces supersets of `fa`, we suppose that `fa` can be implemented more efficiently either by using some appropriate heuristic for ILP, or by using some other type of formulation than ILP.

As suggested in Section 8, `fa` can be combined with a faster algorithm to increase its speed while preserving its completeness. Since the `fd` generated only subsets of mutex groups inferred by `fa` (in all cases), we have implemented a combination of `fa` and `fd` that we denote by $\mathtt{fa}_{\mathtt{fd}}$. The algorithm $\mathtt{fa}_{\mathtt{fd}}$ first infers mutex groups by `fd` and then runs `fa` initialized by these mutex groups (see Section 8 and specifically Equation (3)), i.e., `fa` spends its computational time only on the mutex groups that were not already inferred by `fd`. The resulting running time of $\mathtt{fa}_{\mathtt{fd}}$ over all domains is below 32 minutes which is only about 58% of the running time of `fa`. Without considering the domain `tetris14`, `fa` is more than two times slower than $\mathtt{fa}_{\mathtt{fd}}$ Therefore, the difference between $\mathtt{h}^{2\star}$ and $\mathtt{fa}_{\mathtt{fd}}$ is smaller than between $\mathtt{h}^{2\star}$ and `fa`. Thus, $\mathtt{fa}_{\mathtt{fd}}$ proved to be a significant improvement over `fa`.

domain	**#ps**	**fd**	**$h^{2\star}$**	**fa**	**fa$\succ$fd**	**fd$\succ$fa**	**$h^{2\star}\succ$fa**	**fa$\succ h^{2\star}$**	**$h^{2\star}\succ$fd**	**fd$\succ h^{2\star}$**
childsnack14	20	1 248	1 248	1 248	0	0	0	0	0	0
elevators11	20	245	245	245	0	0	0	0	0	0
floortile11	20	624	624	624	0	0	0	0	0	0
floortile14	20	575	575	575	0	0	0	0	0	0
ged14	20	330	330	330	0	0	0	0	0	0
hiking14	20	229	229	229	0	0	0	0	0	0
nomystery11	20	190	190	190	0	0	0	0	0	0
openstacks11	20	800	800	800	0	0	0	0	0	0
openstacks14	20	1 440	1 440	1 440	0	0	0	0	0	0
scanalyzer11	20	432	432	432	0	0	0	0	0	0
transport11	20	217	217	217	0	0	0	0	0	0
transport14	20	206	206	206	0	0	0	0	0	0
visitall11	20	1 010	1 010	1 010	0	0	0	0	0	0
visitall14	20	2 434	2 434	2 434	0	0	0	0	0	0
barman11	20	2 164	**555**	584	20	0	20	0	20	0
barman14	20	2 210	**555**	581	20	0	20	0	20	0
cavediving14	20	3 726	**913**	**913**	20	0	0	0	20	0
maintenance14	20	1 285	**536**	**536**	20	0	0	0	20	0
parcprinter11	20	2 467	1 026	**1 016**	20	0	0	10	20	0
parking11	20	**1 210**	1 340	**1 210**	0	0	0	20	0	20
parking14	20	**1 140**	1 260	**1 140**	0	0	0	20	0	20
pegsol11	20	**680**	683	683	0	3	0	0	0	3
sokoban11	20	1 066	**1 065**	**1 065**	1	0	0	0	1	0
tetris14	20	16 672	**676**	**676**	20	0	0	0	20	0
tidybot11	20	5 708	**2 732**	**2 732**	20	0	0	0	20	0
tidybot14	20	7 472	**3 514**	**3 514**	20	0	0	0	20	0
woodworking11	20	1 595	**1 134**	1 584	6	0	20	0	20	0
Σ	540	57 375	**25 969**	26 214	167	3	60	50	181	43

Table 7: Number of variables in FDR.

10.4 Translation to Finite Domain Representation

Now we shift our attention from the comparison of the tested algorithms in terms of inferred mutex groups towards the applicability of the algorithms in the actual planning process. One straightforward application of mutex groups is in the translation from PDDL to finite domain representation (FDR). The variables of FDR can be created from mutex groups such that each mutex group is used for the creation of one variable. Since, at most, one fact from a mutex group can be true in any state, each value of the corresponding variable represents one fact from the mutex group. If it is possible that a state does not contain any fact from the mutex group, the corresponding variable must contain one additional value "none of those".

The optimal allocation of variables in terms of the minimal number of variables is NP-Hard, as already mentioned above when we discussed mutex group cover numbers. (The minimal mutex group cover number is also the minimal possible number of variables in FDR.) The Fast Downward's preprocessor that we used for comparison creates variables from the inferred mutex groups in a greedy way. In each step, the mutex group containing the most facts that are not yet covered by any variable (breaking ties arbitrarily) is taken and a new variable is created from it. Moreover, the preprocessor also performs some basic pruning based on an inconsistent encoding of operators' preconditions and the resulting unreachability of facts within domain transition graphs of the corresponding variables. Therefore, the number of created variables can be actually lower than the computed mutex group cover number.

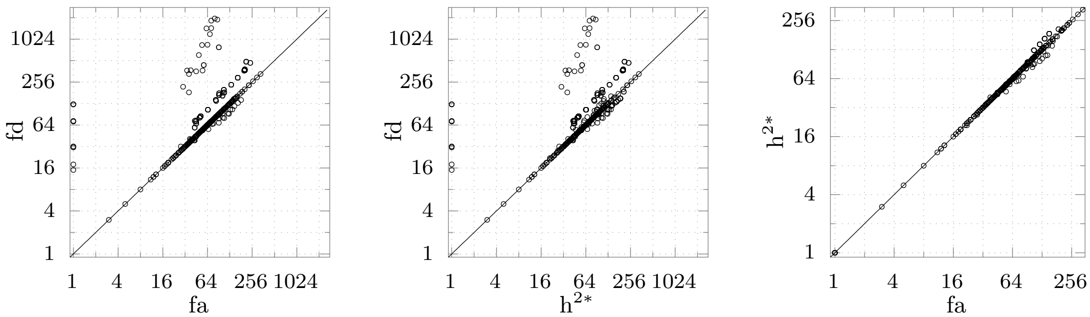


Figure 8: Comparison of the minimal number of bits required for storing a single state given the variable encoding in FDR.

Table 7 shows the number of created variables per domain and overall for the algorithms `fd`, `h`^{2*}, and `fa`. The numbers are very similar to the mutex group cover numbers listed in Table 4, which means that the greedy algorithm used in Fast Downward can actually generate a number of variables very close to the possible minimum.

Table 7 also shows a number of problems in which one algorithm generated less variables than the other one ($a \succ b$ means a generated less variables than b). `fd` is clearly dominated by both `h`^{2*} and `fa`, which was expected considering the experimental results from the previous sections. The results for `h`^{2*} and `fa` are very similar. `h`^{2*} creates less variables than `fa` in 60 problems, but `fa` creates less variables than `h`^{2*} in 50 problems even though the mutex group cover number for `h`^{2*} must always be lower than for `fa`. This is clearly an effect of tie breaking in the greedy algorithm. Rather surprising is the fact that `fd` dominates `h`^{2*} in 43 problems. The reason is, again, tie breaking, but it also shows that in some cases, `h`^{2*} generates unnecessarily large number of mutex groups that are pairwise complementary which confuses the preprocessor. However, it does not mean that the mutex groups cannot be useful in other parts of the planner. It just means that the particular greedy algorithm used in Fast Downward is not well equipped for rich sets of mutex groups having many common facts.

We have also tried a different selection algorithm used in the SymbA* planner (Torralba, Alcázar, Borrajo, Kissmann, & Edelkamp, 2014), where the mutex groups containing a fact that is not part of any other mutex group take precedence. Although the resulting number of variables was lower due to stronger pruning (54 626, 24 009, and 24 551 for `fd`, `h`^{2*}, and `fa`, respectively), the relative comparison between the methods was almost identical. The most noticeable difference was that in the `parking11` and `parking14` domains, the resulting variables were identical for all three methods, and that in `parcprinter11`, the variables were identical for `h`^{2*} and `fa`. So, toghether with some small changes in `pegsol11` and `sokoban11`, the dominance changed to 0, 4, and 3 for `fd`$\succ$`fa`, `fa`$\succ$`h`^{2*}, and `fd`$\succ$`h`^{2*}, respectively, the rest remained the same.

As mentioned in Section 10.2, the lower number of variables can lead to the lower memory consumption of the planner, because states in FDR are represented more compactly. Figure 8 shows scatter plots of the minimal number of bits required for storing a single state for each planning problem and inference algorithm. The number is computed as the sum of the number of bits required for storing each variable, e.g., if the planning task has three variables with 2, 6, and 12 values, then the minimal number of bits is computed as $1 + 3 + 4 = 8$ bits. Clearly, fa and $\mathtt{h}^{2\star}$ both generate more compact representations than fd in most cases. The number of bits required for a single state ranged from 1 to 332 bits for fa and $\mathtt{h}^{2\star}$, and from 3 to 1992 bits for fd.

We have also compared the algorithms in terms of coverage over all planning domains. We use the MAPlan[4] planner implementation of A* with the following admissible heuristics:

- lmc: LM-Cut heuristic (Helmert & Domshlak, 2009),
- flow: flow-based heuristic (Bonet, 2013; Bonet & van den Briel, 2014),
- flow-lmc: flow-based heuristic with added constraints corresponding to the landmarks obtained using LM-Cut (Bonet & van den Briel, 2014), and
- pot: potential heuristic (Pommerening, Helmert, Röger, & Seipp, 2015) with the maximization over all syntactic states (Seipp, Pommerening, & Helmert, 2015).

The maximal allowed time for the whole planning process (including preprocessor and search) was set to four hours and the maximal memory limit was set to 8 GB. For this experiment, we used $\mathtt{fa}_{\mathtt{fd}}$ instead of fa, because it is the faster variant as was demonstrated in the previous section.

Although we would like to filter out the influence of the pruning of operators and facts from the planning tasks, this is not entirely possible. Some operators simply cannot be translated into FDR because they have their preconditions or effects in conflict with some variable in FDR. For example, consider the mutex group $\{f_1, f_2\}$ that is used for the construction of a new variable, and an operator $f_1, f_2 \mapsto f_3$. Such an operator has preconditions that cannot be represented in the constructed FDR. It should be stressed that this behaviour is correct, because this operator cannot be used in any reachable state, which follows from the mutex group $\{f_1, f_2\}$. However, this side effect of using mutex groups for translation into FDR also needs to be taken into consideration, when the experimental results are evaluated.

The results are listed in Table 8. A more detailed investigation of the differences between the tested algorithms shows the following. The lower coverage of fd in the tetris14 domain in comparison to both $\mathtt{fa}_{\mathtt{fd}}$ and $\mathtt{h}^{2\star}$ is due to the considerable pruning of operators during the translation, which corresponds to the much lower number of variables (Table 7) produced by $\mathtt{fa}_{\mathtt{fd}}$ and $\mathtt{h}^{2\star}$. $\mathtt{h}^{2\star}$ has lower coverage than $\mathtt{fa}_{\mathtt{fd}}$ in tetris14 because the inference of mutex groups consumed the entire assigned time in some problems (compare with Table 5).

In the domains nomystery11, scanalyzer11, parking11, and woodworking11, the number of mutex groups differ. The differences in coverage for these domains are caused by more successful pruning by the methods that produced richer sets of mutex groups.

The results from the remaining domains that were caused by the different behavior of the heuristics depending on the FDR variable encoding are more interesting. The indeterminism of lmc in the construction of a justification graph presented itself only in one additional problem solved by fd in tidybot14. The different numbers for flow and flow-lmc heuristics

4. https://github.com/danfis/maplan, branch jair-fa-mutex

domain	#ps	lmc			flow			flow-lmc			pot		
		fd	**$h^{2\star}$**	**fa_{fd}**	**fd**	**$h^{2\star}$**	**fa_{fd}**	**fd**	**$h^{2\star}$**	**fa_{fd}**	**fd**	**$h^{2\star}$**	**fa_{fd}**
barman11	20	8	8	8	4	4	4	4	4	4	4	4	4
barman14	20	9	9	9	6	6	6	6	6	6	6	6	6
cavediving14	20	7	7	7	7	7	7	7	7	7	7	7	7
childsnack14	20	0	0	0	0	0	0	0	0	0	0	0	0
elevators11	20	18	18	18	12	12	12	18	18	18	13	13	13
floortile11	20	9	9	9	4	4	4	4	4	4	4	4	4
floortile14	20	8	8	8	2	2	2	2	2	2	2	2	2
ged14	20	19	19	19	15	15	15	15	15	15	15	15	15
hiking14	20	11	11	11	11	11	11	10	10	10	13	13	13
maintenance14	20	5	5	5	5	5	5	5	5	5	5	5	5
openstacks11	20	18	18	18	15	15	15	15	15	15	18	18	18
openstacks14	20	3	3	3	3	3	3	3	3	3	3	3	3
pegsol11	20	19	19	19	19	19	19	19	19	19	19	19	19
sokoban11	20	20	20	20	20	20	20	18	18	18	20	20	20
transport11	20	9	9	9	6	6	6	7	7	7	6	6	6
transport14	20	7	7	7	6	6	6	6	6	6	7	7	7
visitall11	20	11	11	11	17	17	17	19	19	19	16	16	16
visitall14	20	6	6	6	14	14	14	15	15	15	12	12	12
nomystery11	20	17	17	17	12	12	12	13	**14**	13	14	14	14
parcprinter11	20	17	17	17	**20**	16	19	**20**	16	16	6	**8**	**8**
parking11	20	5	5	5	3	3	3	1	**2**	1	5	5	5
parking14	20	5	5	5	3	3	3	0	**3**	0	5	5	5
scanalyzer11	20	14	14	14	13	**14**	**14**	13	**14**	**14**	10	10	10
tetris14	20	8	10	**11**	13	14	**17**	13	14	**17**	12	12	**13**
tidybot11	20	15	15	15	9	**11**	10	13	13	13	14	14	14
tidybot14	20	**12**	11	11	1	**2**	**2**	**7**	6	6	9	**10**	**10**
woodworking11	20	17	17	17	8	8	8	6	**8**	6	6	**7**	6
Σ	540	297	298	**299**	248	249	**254**	259	**263**	259	251	**255**	**255**
as %	100	55.0	55.2	**55.4**	45.9	46.1	**47.0**	48.0	**48.7**	48.0	46.5	**47.2**	**47.2**

Table 8: Coverage with different inference algorithms used for FDR.

in `parcprinter11`, `tidybot11`, `parking14`, and `tidybot14` domains are caused by the dependency of the linear program (LP) used for computing flow heuristics on the variable encoding in FDR. We are not aware of any literature regarding the influence of variable encoding on the computation of LP-based flow heuristics, so we cannot fully explain this behaviour. The LP-based flow heuristics use LP-relaxation of the integer variables used in the flow heuristics based on integer linear program (ILP) (Pommerening, Röger, Helmert, & Bonet, 2014) and we have experimentally verified that the actual values returned by the LP-based heuristic depends on the encoding of the problem. We have also experimentally verified that the values returned by the ILP-based flow heuristics is independent to the encoding (given the identical set of operators and the identical set of facts, i.e., either no pruning or an identical pruning is involved). The results show that a less compact representation (e.g., `parcprinter11`) can positively influence the flow heuristics, but how should the FDR variables be constructed so that flow heuristics can benefit from it, remains an open question.

Even though the memory size needed for storing states during the search is significantly reduced by $\mathtt{fa}_{\mathtt{fd}}$ and $\mathtt{h}^{2\star}$, the results do not suggest that it has a direct positive effect on the number of solved tasks. It seems that the inferred mutex groups used for translation to FDR have a more profound effect due to pruning (as a side effect of the translation) and the changed behavior of heuristic functions as discussed above.

domain	**#operators**				**#removed operators**				**dead-end**
	fd	$\mathbf{h^{2\star}}$	$\overline{\mathbf{fa}}_{\mathbf{fd}}$	$\mathbf{fa_{fd}}$	**fd**	$\mathbf{h^{2\star}}$	$\overline{\mathbf{fa}}_{\mathbf{fd}}$	$\mathbf{fa_{fd}}$	$\mathbf{fa_{fd}}$
childsnack14	53 698	53 698	53 698	53 698	0	0	0	0	0
elevators11	11 450	11 450	11 450	11 450	0	0	0	0	0
ged14	14 114	14 114	14 114	14 114	375	375	375	375	0
hiking14	55 878	55 878	55 878	55 878	0	0	0	0	0
openstacks11	17 320	17 320	17 320	17 320	0	0	0	0	0
openstacks14	55 820	55 820	55 820	55 820	0	0	0	0	0
tidybot11	384 018	384 018	384 018	384 018	0	0	0	0	0
tidybot14	599 642	599 642	599 642	599 642	0	0	0	0	0
transport11	35 216	35 216	35 216	35 216	0	0	0	0	0
transport14	81 058	81 058	81 058	81 058	0	0	0	0	0
visitall11	3 520	3 520	3 520	3 520	0	0	0	0	0
visitall14	8 912	8 912	8 912	8 912	0	0	0	0	0
barman11	13 264	11 552	13 264	**8 980**	2 544	4 256	2 544	**6 828**	4 284
barman14	13 610	11 930	13 610	**9 014**	2 592	4 272	2 592	**7 188**	4 596
cavediving14	92 078	**91 832**	92 078	92 078	0	**246**	0	0	0
floortile11	9 188	9 188	9 188	**7 078**	0	0	0	**2 110**	2 110
floortile14	6 544	6 544	6 544	**5 050**	0	0	0	**1 494**	1 494
maintenance14	984	417	417	**166**	111	678	678	**929**	390
nomystery11	72 522	**55 663**	72 522	72 522	0	**16 859**	0	0	0
parcprinter11	5 080	4 816	4 816	**1 932**	16	280	280	**3 164**	2 492
parking11	241 740	236 120	241 740	**232 800**	8 940	14 560	8 940	**17 880**	8 940
parking14	201 600	196 640	201 600	**193 680**	7 920	12 880	7 920	**15 840**	7 920
pegsol11	3 700	3 499	3 651	**3 490**	0	201	49	**210**	131
scanalyzer11	631 288	**425 680**	425 720	425 720	4 552	**210 160**	210 120	210 120	0
sokoban11	7 166	**7 140**	7 166	7 164	0	**26**	0	2	2
tetris14	252 952	**12 468**	**12 468**	**12 468**	0	**240 484**	**240 484**	**240 484**	0
woodworking11	18 175	**10 148**	18 141	16 709	0	**8 027**	34	1 466	1 306
Σ	2 890 537	**2 404 283**	2 443 571	2 409 497	27 050	**513 304**	474 016	508 090	33 665

Table 9: The number of operators after pruning, the number of removed operators and the number of removed dead-end operators.

10.5 Pruning of Planing Tasks

In Section 9, we proposed an algorithm for pruning planning tasks which uses inferred mutex groups for the removal of operators and facts. Although the algorithm is formulated specifically for `fa`, we also described how the algorithm can be altered to include different inference algorithms.

The proposed pruning algorithm (Algorithm 3) is experimentally evaluated as a part of the Fast Downward's preprocessor. The pruning algorithm is utilized right after the grounding of the PDDL planning task and the inferred mutex groups are further used for creation of the variables in FDR. For the solving of FDR planning tasks, we use the same MAPlan planner with the same admissible heuristics as in the previous section (`lmc`, `flow`, `flow-lmc`, and `pot`). The time and memory limits were also set to four hours and 8 GB, respectively. The results in Table 9 and Table 10 were measured without a time limit.

Algorithm 3 with `fd` used for inference runs in one cycle only, because `fd` infers mutex groups on the lifted PDDL task, therefore, the consecutive cycles cannot remove any additional operators or facts. The removal of the operators producing dead-end states (*dead-end operators*) is not utilized in this configuration because the properties required (Corollary 8) for this operation are not proven for the mutex groups inferred by `fd`. The algorithm $\mathtt{h}^{2\star}$ produces mutex groups useful for the creation of FDR variables, therefore, we also compare this algorithm. In contrast to `fd`, Algorithm 3 utilizing $\mathtt{h}^{2\star}$ runs until a fixpoint is reached because h^2-mutexes are inferred from the grounded operators. Operators producing dead-end states are not removed in this configuration either because the mutex groups generated

domain	#facts fd	#facts $\mathbf{h}^{2\star}$	#facts $\overline{\mathbf{fa}}_{\mathbf{fd}}$	#facts $\mathbf{fa}_{\mathbf{fd}}$	#variables fd	#variables $\mathbf{h}^{2\star}$	#variables $\overline{\mathbf{fa}}_{\mathbf{fd}}$	#variables $\mathbf{fa}_{\mathbf{fd}}$
childsnack14	3 674	3 674	3 674	3 674	1 248	1 248	1 248	1 248
elevators11	2 097	2 097	2 097	2 097	245	245	245	245
floortile11	2 966	2 966	2 966	2 966	624	624	624	624
floortile14	2 474	2 474	2 474	2 474	575	575	575	575
ged14	3 329	3 329	3 329	3 329	330	330	330	330
hiking14	1 104	1 104	1 104	1 104	229	229	229	229
openstacks11	2 360	2 360	2 360	2 360	800	800	800	800
openstacks14	4 280	4 280	4 280	4 280	1 440	1 440	1 440	1 440
scanalyzer11	3 088	3 088	3 088	3 088	432	432	432	432
transport11	2 886	2 886	2 886	2 886	217	217	217	217
transport14	5 544	5 544	5 544	5 544	206	206	206	206
visitall11	2 516	2 516	2 516	2 516	773	773	773	773
visitall14	6 910	6 910	6 910	6 910	2 258	2 258	2 258	2 258
barman11	4 604	2 847	2 876	**2 407**	2 164	**555**	584	584
barman14	4 692	2 891	2 917	**2 411**	2 210	**555**	581	581
cavediving14	8 368	**5 515**	**5 515**	**5 515**	3 654	**821**	**821**	**821**
maintenance14	2 496	1 065	1 065	**306**	1 248	525	525	**153**
nomystery11	4 404	**4 180**	4 404	4 404	190	190	190	190
parcprinter11	3 977	3 716	3 684	**2 494**	1 197	959	945	**624**
parking11	11 020	11 150	11 020	**10 680**	**1 210**	1 340	**1 210**	**1 210**
parking14	9 880	10 000	9 880	**9 560**	**1 140**	1 260	**1 140**	**1 140**
pegsol11	2 000	2 032	2 032	**1 998**	680	683	683	**676**
sokoban11	4 698	**4 690**	4 697	4 696	1 066	**1 060**	1 065	1 065
tetris14	34 728	**5 612**	**5 612**	**5 612**	16 672	**676**	**676**	**676**
tidybot11	11 476	**8 380**	**8 380**	**8 380**	5 708	**2 732**	**2 732**	**2 732**
tidybot14	15 004	**10 926**	**10 926**	**10 926**	7 472	**3 514**	**3 514**	**3 514**
woodworking11	3 707	**3 291**	3 690	3 625	1 489	**1 025**	1 476	1 454
Σ	164 282	119 523	119 926	**116 242**	55 477	25 272	25 519	**24 797**

Table 10: The number of variables and facts after pruning.

by $\mathtt{h}^{2\star}$ do not have the required properties (see Section 7). In the case of fam-groups, we use the $\mathtt{fa}_{\mathtt{fd}}$ variant of the algorithm because it is less time demanding than `fa`. To stress the impact of the removal of dead-end operators, we have also evaluated Algorithm 3 without the removal of this type of operator and this variant is denoted by $\overline{\mathtt{fa}}_{\mathtt{fd}}$.

Table 9 shows the number of operators that remained in the planning tasks after pruning, the number of operators that were removed from the grounded PDDL, and the number of removed dead-end operators (in the case of $\mathtt{fa}_{\mathtt{fd}}$). Note that the number of removed dead-end operators does not necessarily equal the difference between the number of removed operators by $\overline{\mathtt{fa}}_{\mathtt{fd}}$ and $\mathtt{fa}_{\mathtt{fd}}$. The reason is that the pruning algorithm runs in cycles and the removal of some dead-end operator can cause the removal of more operators in the next cycle because a different set of mutex groups is inferred. Considering the relationship between h^2-mutexes and fam-groups, $\mathtt{h}^{2\star}$ must always remove a superset of operators of those removed by $\overline{\mathtt{fa}}_{\mathtt{fd}}$, but the same does not hold for $\mathtt{fa}_{\mathtt{fd}}$ because of its ability to detect dead-end operators. Similarly, considering the results presented in Section 10.2, where it was shown that `fd` produced a subset of `fa` mutex groups, the operators removed by `fd` must be a subset of the operators removed by $\overline{\mathtt{fa}}_{\mathtt{fd}}$, $\mathtt{fa}_{\mathtt{fd}}$ and, thus, also by $\mathtt{h}^{2\star}$.

The poorest performance, in terms of the removed operators, was shown by `fd`, which was expected given the results presented in the previous sections. $\mathtt{h}^{2\star}$ managed to remove almost twenty times more operators than `fd` (513 304 vs. 27 050), but only around 8% more than $\overline{\mathtt{fa}}_{\mathtt{fd}}$, and around 1% more than $\mathtt{fa}_{\mathtt{fd}}$.

If we look at the number of facts that remained in the planning tasks after pruning (Table 10), we observe similar results. Everything that was said about the removed operators

domain	#ps	lmc				flow				flow-lmc				pot			
		fd	$h^{2\star}$	$\overline{\mathtt{fa}}_{\mathtt{fd}}$	$\mathtt{fa}_{\mathtt{fd}}$	fd	$h^{2\star}$	$\overline{\mathtt{fa}}_{\mathtt{fd}}$	$\mathtt{fa}_{\mathtt{fd}}$	fd	$h^{2\star}$	$\overline{\mathtt{fa}}_{\mathtt{fd}}$	$\mathtt{fa}_{\mathtt{fd}}$	fd	$h^{2\star}$	$\overline{\mathtt{fa}}_{\mathtt{fd}}$	$\mathtt{fa}_{\mathtt{fd}}$
cavediving14	20	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
childsnack14	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
elevators11	20	18	18	18	18	12	12	12	12	18	18	18	18	13	13	13	13
ged14	20	19	19	19	19	15	15	15	15	15	15	15	15	15	15	15	15
hiking14	20	11	11	11	11	11	11	11	11	10	10	10	10	13	13	13	13
maintenance14	20	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
openstacks11	20	18	18	18	18	15	15	15	15	15	15	15	15	18	18	18	18
openstacks14	20	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
pegsol11	20	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19
sokoban11	20	20	20	20	20	20	20	20	20	19	19	19	19	20	20	20	20
transport11	20	9	9	9	9	6	6	6	6	7	7	7	7	6	6	6	6
transport14	20	7	7	7	7	6	6	6	6	6	6	6	6	7	7	7	7
visitall11	20	11	11	11	11	17	17	17	17	19	19	19	19	16	16	16	16
visitall14	20	6	6	6	6	14	14	14	14	15	15	15	15	12	12	12	12
barman11	20	8	8	8	8	4	4	4	**8**	4	4	4	**8**	4	4	4	**8**
barman14	20	9	9	9	9	6	6	6	**9**	6	6	6	**9**	6	6	6	**9**
floortile11	20	9	9	9	9	4	4	4	**6**	4	4	4	**6**	4	4	4	**6**
floortile14	20	8	8	8	**9**	2	2	2	**5**	2	2	2	**5**	2	2	2	**5**
nomystery11	20	17	17	17	17	12	12	12	12	13	**14**	13	13	14	14	14	14
parcprinter11	20	17	17	17	**18**	**20**	18	19	**20**	**20**	17	16	**20**	6	13	13	**16**
parking11	20	5	5	5	5	3	3	3	**5**	0	3	0	**5**	5	4	5	**7**
parking14	20	5	5	5	5	3	3	3	**5**	0	3	0	**5**	5	5	5	**7**
scanalyzer11	20	14	14	14	14	13	**14**	**14**	**14**	13	**14**	**14**	**14**	10	10	10	10
tetris14	20	8	9	**11**	**11**	13	12	**17**	**17**	13	12	**17**	**17**	12	10	**13**	**13**
tidybot11	20	**16**	15	**16**	**16**	9	**12**	11	**12**	13	13	13	13	14	14	14	14
tidybot14	20	12	12	12	12	0	2	**3**	2	**7**	6	6	6	9	**10**	**10**	**10**
woodworking11	20	17	17	17	**19**	7	8	7	**9**	6	8	6	**20**	6	**8**	6	**8**
Σ	540	298	298	301	**305**	246	250	255	**274**	259	264	259	**299**	251	258	260	**281**
as %	100	55.2	55.2	55.7	**56.5**	45.6	46.3	47.2	**50.7**	48.0	48.9	48.0	**55.4**	46.5	47.8	48.1	**52.0**
30 min. time limit:																	
Σ	540	255	261	259	**264**	200	201	205	**219**	208	210	208	**238**	249	254	258	**279**
as %	100	47.2	48.3	48.0	**48.9**	37.0	37.2	38.0	**40.6**	38.5	38.9	38.5	**44.1**	46.1	47.0	47.8	**51.7**

Table 11: Coverage with different inference algorithms used for pruning.

holds also for the removed facts, i.e., `fd` removed a subset of those removed by $\overline{\mathtt{fa}}_{\mathtt{fd}}$, $\mathtt{fa}_{\mathtt{fd}}$, and $\mathtt{h}^{2\star}$; and $\overline{\mathtt{fa}}_{\mathtt{fd}}$ removed a subset of $\mathtt{h}^{2\star}$. On the other hand, $\mathtt{fa}_{\mathtt{fd}}$ removed a different set of facts than $\mathtt{h}^{2\star}$ because it also removed a different set of operators.

The results regarding the number of the variables that were created from the inferred mutex groups after pruning are also listed in Table 10. The lowest number was achieved by $\mathtt{fa}_{\mathtt{fd}}$, but the results for $\mathtt{fa}_{\mathtt{fd}}$, $\overline{\mathtt{fa}}_{\mathtt{fd}}$, and $\mathtt{h}^{2\star}$ are very similar. The difference between these methods was caused mainly due to the greedy algorithm used for the creation of variables in the Fast Downward's preprocessor described in the previous section. The main sources of the difference between $\mathtt{fa}_{\mathtt{fd}}$ and $\mathtt{h}^{2\star}$ were the domains `parcprinter11`, `parking11`, and `maintenance14` where $\mathtt{fa}_{\mathtt{fd}}$ pruned more facts than $\mathtt{h}^{2\star}$, whereas `fd` created more than twice as many variables than any other method. Note that these results correspond to the analysis based on the computation of the mutex group cover number presented in Section 10.2 (Table 4). The main question now is how these results translate into the number of solved tasks.

The coverage over all tested domains is reported in Table 11. The lowest coverage overall was recorded for the planner with `fd` for all tested heuristics. This indicates that the shorter time spent in inference and pruning by `fd` does not compensate for less informative mutex groups and more sparse pruning. The less concise representation of states results in higher

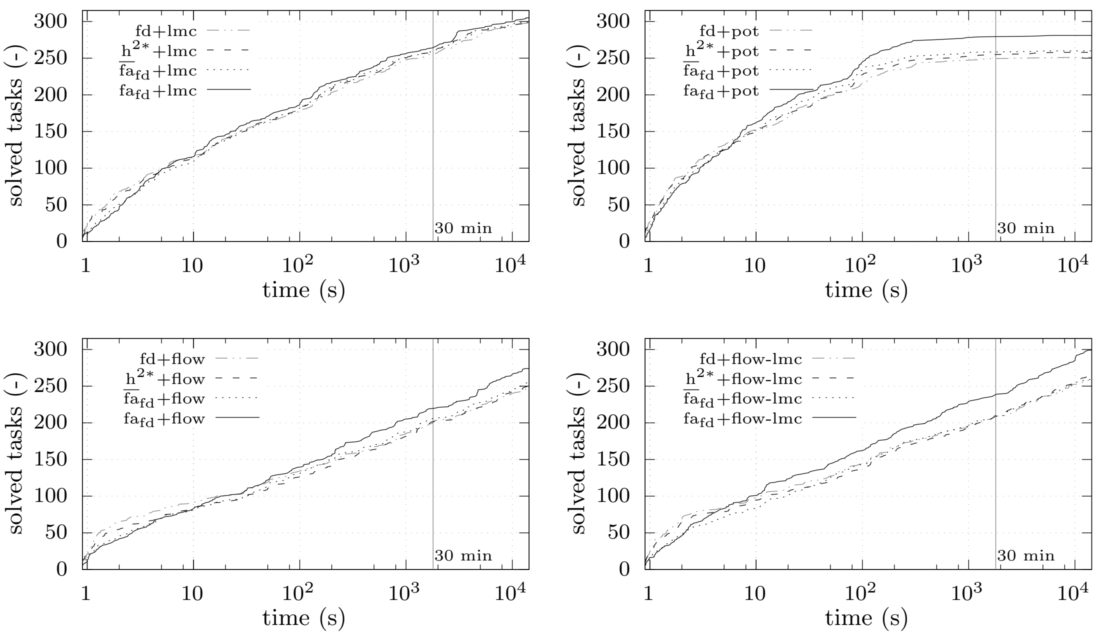


Figure 9: The number of solved tasks over time. The added vertical line marks 30 minutes.

memory requirements and more operators in the planning tasks result in wider branching during a state space exploration.

The coverage for $\mathtt{h}^{2\star}$ and $\overline{\mathtt{fa}}_{\mathtt{fd}}$ is almost the same for all heuristics. Judging by the quality of the translation process only, $\mathtt{h}^{2\star}$ should be strictly better than $\overline{\mathtt{fa}}_{\mathtt{fd}}$ because $\mathtt{h}^{2\star}$ always generates a superset of $\overline{\mathtt{fa}}_{\mathtt{fd}}$ mutex groups, thus, the number of operators should be smaller and states should be encoded by a smaller number of variables. If we look carefully at the results we can observe that $\mathtt{h}^{2\star}$ consistently shows a small coverage in domain `tetris14` (in comparison to $\overline{\mathtt{fa}}_{\mathtt{fd}}$ or $\mathtt{fa}_{\mathtt{fd}}$) which corresponds to the high running times measured for the $\mathtt{h}^{2\star}$ variant (Table 5). The time needed for $\mathtt{h}^{2\star}$ to complete in the case of `tetris14` domain clearly took its toll and it is reflected in the results. In the case of `flow`, the whole difference between $\mathtt{h}^{2\star}$ and $\overline{\mathtt{fa}}_{\mathtt{fd}}$ can be accounted for the `tetris14` domain, the difference in the case of `lmc` and `pot` is even higher than the overall difference, and in the case of `flow-lmc`, both $\mathtt{h}^{2\star}$ and $\overline{\mathtt{fa}}_{\mathtt{fd}}$ have the same overall coverage even though $\overline{\mathtt{fa}}_{\mathtt{fd}}$ solved six more tasks in `tetris14`. So we can conclude that $\mathtt{h}^{2\star}$ performed worse than $\overline{\mathtt{fa}}_{\mathtt{fd}}$ over all domains mainly because of the very slow inference in the `tetris14` domain.

The removal of dead-end operators and the concise finite domain representation in the case of $\mathtt{fa}_{\mathtt{fd}}$ led to the highest overall coverage for all tested heuristics. The planner with $\mathtt{fa}_{\mathtt{fd}}$ solved from 2.4% (`lmc`) to 7.2% (`flow-lmc`) more tasks than the planner with `fd` and from 1.7% to 7% more than the planner with $\mathtt{h}^{2\star}$. The increase in the number of solved tasks is substantial even if we consider shorter time limits. For the 30 minute time limit, the relative differences between inference algorithms is very similar as with four hour time limit (see the bottom rows in Table 11). The graphs in Figure 9 show the course of the number of solved tasks in time. For very short time limits, under ten seconds, $\mathtt{fa}_{\mathtt{fd}}$ achieves a smaller coverage than any other method because it is the slowest one (if we do not consider the `tetris14` domain), i.e., the inference of mutex groups, the pruning of the planning tasks and, therefore, the whole translation process, before the exploration of a state space starts, takes longer time than for other methods. But, at the very latest on the 100 second mark, $\texttt{fa}_{\texttt{fd}}$ takes the lead and the resulting coverage is consistently higher than with the other methods, indicating that fam-groups produced by $\texttt{fa}_{\texttt{fd}}$ are able to provide a concise representation of states and to substantially prune planning tasks.

10.6 Pruning in Regression

In the previous section, we have experimentally evaluated the pruning of planning tasks using mutex groups inferred in progression. As we have described in Section 5, state invariants can be inferred in regression as well. We have implemented the inference of fam-groups in dual planning tasks and used it for pruning in combination with pruning in progression. Unfortunately, it turned out that the fam-groups inferred in dual planning tasks are very weak in the tested domains. The only domains in which it was possible to infer some dual fam-groups, were `woodworking11`, `parcprinter11`, and `pegsol11`. But even in those domains, no operators were pruned besides those that were already pruned in progression. Therefore, fam-groups in regression could not help increase the coverage over the tested domains.

However, Alcázar and Torralba (2015) proposed an algorithm for pruning planning tasks that also uses state invariants inferred in regression. The algorithm alternates between pruning in progression and regression. In both directions, h^2-mutexes are inferred and used for disambiguation (Alcázar et al., 2013) of operators that helps to identify unreachable operators. Besides the detection of operators having preconditions in contradiction with some mutex group, disambiguation can also extend preconditions and effects with facts that must hold because all other options are in contradiction with the known mutex groups. The algorithm works with problems in FDR. So, for example, consider two variables v_1 and v_2, and an operator with a precondition where v_1 is set to some value f_1 and v_2 is not set. If all values of v_2, except some value f_2, are mutex with the value f_1, then the disambiguation of this operator sets the variable v_2 to the value f_2, because it is the only viable option.

Since the algorithm proposed by Alcázar and Torralba works with problems encoded in FDR, we experimentally evaluated two variants of this algorithm. The variant that uses mutex groups inferred by `fd` for construction of FDR will be denoted as $\texttt{at}_{\texttt{fd}}$, and the variant that uses $\texttt{fa}_{\texttt{fd}}$ will be denoted as $\texttt{at}_{\texttt{fa}}$. These two variants are compared with `fd` as a baseline and with our algorithm $\texttt{fa}_{\texttt{fd}}$. The same configuration was used as in the previous two sections.

Table 12 shows that overall number of removed operators is much higher for $\texttt{at}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$ than for `fd` and $\texttt{fa}_{\texttt{fd}}$. But most of the difference originates in domains `tidybot11` and `tidybot14` where more than 660 000 operators were removed by $\texttt{at}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$, whereas `fd` and $\texttt{fa}_{\texttt{fd}}$ removed no operators. In `nomystery11` and `cavediving14`, `fd` and $\texttt{fa}_{\texttt{fd}}$ also did not remove any operator, but $\texttt{at}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$ managed to remove some.

In `barman11`, `barman14`, and `pegsol11`, $\texttt{fa}_{\texttt{fd}}$ pruned more operators than $\texttt{at}_{\texttt{fd}}$, because of $\texttt{fa}_{\texttt{fd}}$'s ability to prune dead-end operators (Corollary 8), that cannot be replaced by simple disambiguation using h^2-mutexes in progression and regression. Once fam-groups are used for translation into FDR, disambiguation using the created variables helps to identify

domain	#operators				#removed operators			
	fd	**fa**$_{\mathbf{fd}}$	**at**$_{\mathbf{fd}}$	**at**$_{\mathbf{fa}}$	**fd**	**fa**$_{\mathbf{fd}}$	**at**$_{\mathbf{fd}}$	**at**$_{\mathbf{fa}}$
childsnack14	53 698	53 698	53 698	53 698	0	0	0	0
elevators11	11 450	11 450	11 450	11 450	0	0	0	0
ged14	14 114	14 114	14 114	14 114	375	375	375	375
hiking14	55 878	55 878	55 878	55 878	0	0	0	0
openstacks11	17 320	17 320	17 320	17 320	0	0	0	0
openstacks14	55 820	55 820	55 820	55 820	0	0	0	0
transport11	35 216	35 216	35 216	35 216	0	0	0	0
transport14	81 058	81 058	81 058	81 058	0	0	0	0
visitall11	3 520	3 520	3 520	3 520	0	0	0	0
visitall14	8 912	8 912	8 912	8 912	0	0	0	0
barman11	13 264	8 980	11 552	**7 276**	2 544	6 828	4 256	**8 532**
barman14	13 610	9 014	11 930	**7 244**	2 592	7 188	4 272	**8 958**
cavediving14	92 078	92 078	**91 832**	**91 832**	0	0	**246**	**246**
floortile11	9 188	7 078	**5 708**	**5 708**	0	2 110	**3 480**	**3 480**
floortile14	6 544	5 050	**4 060**	**4 060**	0	1 494	**2 484**	**2 484**
maintenance14	984	**166**	**166**	**166**	111	**929**	**929**	**929**
nomystery11	72 522	72 522	**55 654**	**55 654**	0	0	**16 868**	**16 868**
parcprinter11	5 080	1 932	**1 542**	**1 542**	16	3 164	**3 554**	**3 554**
parking11	241 740	**232 800**	**232 800**	**232 800**	8 940	**17 880**	**17 880**	**17 880**
parking14	201 600	**193 680**	**193 680**	**193 680**	7 920	**15 840**	**15 840**	**15 840**
pegsol11	3 700	3 490	3 499	**3 315**	0	210	201	**385**
scanalyzer11	631 288	425 720	**425 680**	**425 680**	4 552	210 120	**210 160**	**210 160**
sokoban11	7 166	7 164	**5 255**	**5 255**	0	2	**1 911**	**1 911**
tetris14	252 952	**12 468**	**12 468**	**12 468**	0	**240 484**	**240 484**	**240 484**
tidybot11	384 018	384 018	**114 963**	**114 963**	0	0	**269 055**	**269 055**
tidybot14	599 642	599 642	**202 432**	**202 432**	0	0	**397 210**	**397 210**
woodworking11	18 175	16 709	**8 927**	**8 927**	0	1 466	**9 248**	**9 248**
Σ	2 890 537	2 409 497	1 719 134	**1 709 988**	27 050	508 090	1 198 453	**1 207 599**

Table 12: The number of operators after pruning and the number of removed operators.

dead-end operators in `barman11` and `barman14`, where all dead-end operators removed by $\texttt{fa}_{\texttt{fd}}$ are also removed by $\texttt{at}_{\texttt{fa}}$. However, in `pegsol11`, $\texttt{fa}_{\texttt{fd}}$ still removes dead-end operators that are not detected by $\texttt{at}_{\texttt{fa}}$ (although there is a small number of them).

Table 13 shows that $\texttt{at}_{\texttt{fa}}$ had the highest overall coverage, followed by $\texttt{at}_{\texttt{fd}}$ and then by $\texttt{fa}_{\texttt{fd}}$. $\texttt{at}_{\texttt{fa}}$ solved from 1.5% (`pot`) to 3.8% (`lmc`) more tasks than $\texttt{fa}_{\texttt{fd}}$. $\texttt{at}_{\texttt{fd}}$ had almost the identical overall coverage as $\texttt{fa}_{\texttt{fd}}$ for `flow` and `pot` heuristics, but when `lmc` was involved, the overall coverage increased more significantly.

The differences in the number of solved tasks basically correspond to the number of pruned operators. The additional pruning of $\texttt{at}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$ was most significant in the domains `floortile11` and `floortile14`, and also in `tidybot11` and `tidybot14`. The majority of the tasks solved by $\texttt{at}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$ on top of those solved by $\texttt{fa}_{\texttt{fd}}$ are due to these domains. In the case of `flow`, `flow-lmc`, and `pot` heuristics, $\texttt{fa}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$ solved more tasks than $\texttt{at}_{\texttt{fd}}$ in the domains `barman11` and `barman14`. This also corresponds to the number of removed operators due to the ability of $\texttt{fa}_{\texttt{fd}}$ to detect dead-end operators. And in the case of $\texttt{at}_{\texttt{fa}}$, due to the disambiguation effect enabled by the variables created using fam-groups.

Figure 10 shows the course of the number of solved tasks in time per pruning algorithm and per heuristics. Similarly to the corresponding graphs in the previous section (Figure 9), the graphs show that when $\texttt{fa}_{\texttt{fd}}$ is involved (i.e., $\texttt{fa}_{\texttt{fd}}$ and $\texttt{at}_{\texttt{fa}}$), the number of solved tasks is lower than for `fd` (and $\texttt{at}_{\texttt{fd}}$) in the first tens of seconds only. The graphs also show a more significant gap between the $\texttt{fa}_{\texttt{fd}}$ and `at` methods whenever computation of the LM-

domain	**#ps**	**lmc**				**flow**				**flow-lmc**				**pot**			
		fd	**fa$_{\text{fd}}$**	**at$_{\text{fd}}$**	**at$_{\text{fa}}$**	**fd**	**fa$_{\text{fd}}$**	**at$_{\text{fd}}$**	**at$_{\text{fa}}$**	**fd**	**fa$_{\text{fd}}$**	**at$_{\text{fd}}$**	**at$_{\text{fa}}$**	**fd**	**fa$_{\text{fd}}$**	**at$_{\text{fd}}$**	**at$_{\text{fa}}$**
cavediving14	20	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
childsnack14	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
elevators11	20	18	18	18	18	12	12	12	12	18	18	18	18	13	13	13	13
ged14	20	19	19	19	19	15	15	15	15	15	15	15	15	15	15	15	15
hiking14	20	11	11	11	11	11	11	11	11	10	10	10	10	13	13	13	13
maintenance14	20	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
openstacks11	20	18	18	18	18	15	15	15	15	15	15	15	15	18	18	18	18
openstacks14	20	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
pegsol11	20	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19
transport11	20	9	9	9	9	6	6	6	6	7	7	7	7	6	6	6	6
transport14	20	7	7	7	7	6	6	6	6	6	6	6	6	7	7	7	7
visitall11	20	11	11	11	11	17	17	17	17	19	19	19	19	16	16	16	16
barman11	20	8	8	8	8	4	**8**	4	**8**	4	**8**	4	**8**	4	**8**	4	**8**
barman14	20	9	9	9	9	6	**9**	6	**9**	6	**9**	6	**9**	6	**9**	6	**9**
floortile11	20	9	9	**14**	**14**	4	6	**8**	**8**	4	6	**8**	**8**	4	6	**8**	**8**
floortile14	20	8	9	**20**	**20**	2	5	**8**	**8**	2	5	**8**	**8**	2	5	**8**	**8**
nomystery11	20	17	17	17	17	12	12	12	12	13	13	**14**	**14**	14	14	14	14
parcprinter11	20	17	**18**	**18**	**18**	20	20	20	20	20	20	20	20	6	16	**18**	**18**
parking11	20	5	5	5	5	3	**5**	**5**	**5**	0	**5**	**5**	**5**	5	**7**	**7**	**7**
parking14	20	5	5	5	5	3	**5**	**5**	**5**	0	**5**	**5**	**5**	5	**7**	**7**	**7**
scanalyzer11	20	14	14	14	14	13	**14**	**14**	**14**	13	**14**	**14**	**14**	10	10	10	10
sokoban11	20	20	20	20	20	20	20	20	20	19	19	**20**	**20**	20	20	20	20
tetris14	20	8	**11**	**11**	**11**	13	**17**	**17**	**17**	13	**17**	**17**	**17**	12	13	**14**	13
tidybot11	20	16	16	**18**	**18**	9	12	12	**13**	13	13	**17**	**17**	14	14	14	14
tidybot14	20	12	12	**14**	**14**	0	2	5	**6**	7	6	**12**	**12**	9	**10**	**10**	**10**
visitall14	20	6	6	6	6	14	14	**15**	**15**	15	15	15	15	12	12	12	12
woodworking11	20	17	**19**	**19**	**19**	7	**9**	**9**	**9**	6	**20**	**20**	**20**	6	8	**9**	**9**
Σ	540	298	305	**325**	**325**	246	274	276	**285**	259	299	309	**316**	251	281	283	**289**
as %	100	55.2	56.5	**60.2**	**60.2**	45.6	50.7	51.1	**52.8**	48.0	55.4	57.2	**58.5**	46.5	52.0	52.4	**53.5**
30 min. time limit:																	
Σ	540	255	264	289	**290**	200	219	228	**232**	208	238	262	**264**	249	279	282	**288**
as %	100	47.2	48.9	53.5	**53.7**	37.0	40.6	42.2	**43.0**	38.5	44.1	48.5	**48.9**	46.1	51.7	52.2	**53.3**

Table 13: Coverage with different inference algorithms used for pruning.

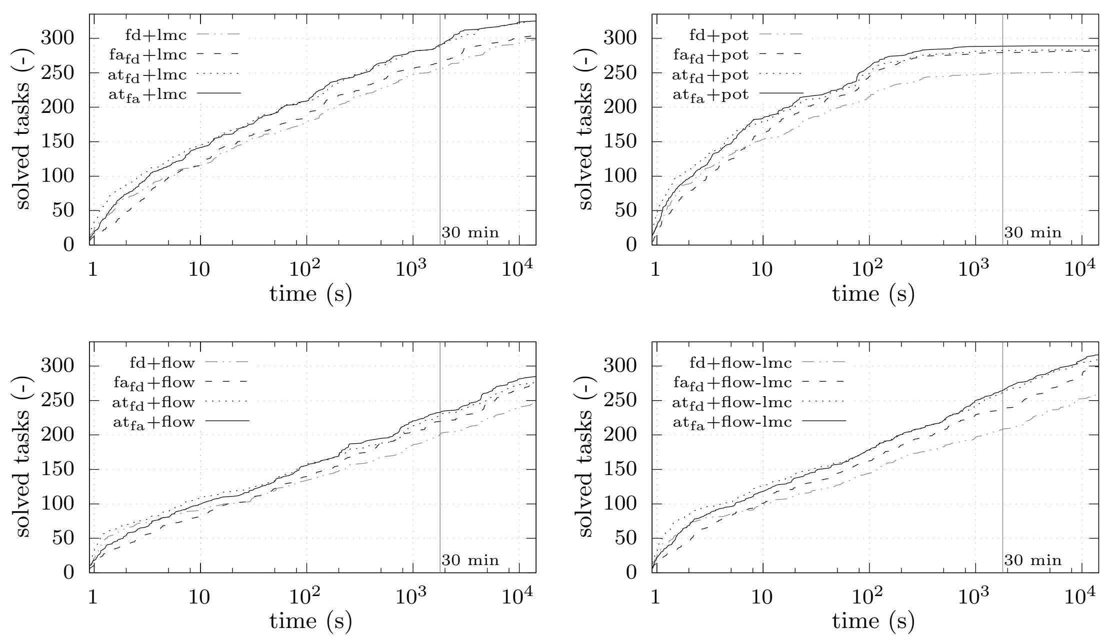


Figure 10: The number of solved tasks over time. The added vertical line marks 30 minutes.

Cut heuristics is involved. In the case of `flow` and `pot`, the course of all three methods $\mathtt{fa}_{\mathtt{fd}}$, $\mathtt{at}_{\mathtt{fd}}$, and $\mathtt{at}_{\mathtt{fa}}$ is very similar.

Overall, the results show that pruning both in progression and regression provides a significant increase in coverage over the methods using pruning only in progression. Regrettably, fam-groups turned out to be very weak invariants in regression and basically useless for pruning in regression (at least in the tested domains). However, fam-groups used together with the method proposed by Alcázar and Torralba (2015) considerably increase the number of solved tasks in some domains.

11. Conclusion

This paper is focused on the inference of a particular type of state invariants called mutex groups in the context of STRIPS planning. The complexity analysis, that we have provided, shows that the inference of the maximum sized mutex group is PSPACE-Complete, i.e., it is as hard as solving the planning problem itself. For this reason, we have introduced a new weaker type of mutex group called a fact-alternating mutex group (fam-group) and we have shown that the inference of the maximum sized fam-group is NP-Complete. This result allowed us to introduce a novel algorithm for inference of fam-groups based on integer linear programming that is complete with respect to maximal fam-groups.

The main property of the fam-group is that the facts from the fam-group alternate between each other in all states on a path leading from the initial state and once they disappear from any state they cannot reappear again in any consecutive state. This property provides a way to detect operators that can produce only dead-end states.

We have proven that the h^2 variant of h^{m} heuristics (Haslum & Geffner, 2000) generates mutex pairs (h^2-mutexes) that are a superset of a pair decomposition of fam-groups. However, in the experimental evaluation, our algorithm manifested comparable overall results in terms of inferred mutex groups.

The algorithm was also compared with the algorithm for the inference of mutex groups proposed by Helmert (2009) for the translation of planning tasks from PDDL to FDR that is widely used among the planning community. The comparison was performed on the planning tasks from the optimal deterministic track of the last two planning competitions (IPC 2011 and 2014). Our algorithm generated a richer set of mutex groups in almost half of the planning tasks, and in the rest of the tasks the generated set of mutex groups was identical. Therefore, our algorithm can provide a smaller state encoding in FDR than Helmert's algorithm, which was also experimentally verified.

As an example of applicability of fam-groups, we have proposed a pruning algorithm that removes facts and operators from a planning task if they are not useful for solving the task. The algorithm was evaluated with four different state-of-the-art heuristic functions and the results indicate a substantial increase in the overall coverage. In particular, the ability of fam-groups to detect dead-end states proved to be crucial in the pruning of planning tasks.

We also compared our algorithm with the state-of-the-art algorithm proposed by Alcázar and Torralba (2015) for pruning using h^2-mutexes inferred both in progression and regression. This algorithm achieves even better results than our algorithm, because of the h^2-mutexes inferred in regression, whereas fam-groups in regression turned out to be useless for pruning. However, we have shown that using fam-groups for the construction of variables in

FDR further increases the number of operators removed by Alcázar and Torralba's method, because it improves the disambiguation process.

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. 15-20433Y). Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

References

Alcázar, V., Borrajo, D., Fernández, S., & Fuentetaja, R. (2013). Revisiting regression in planning. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2254–2260.

Alcázar, V., & Torralba, Á. (2015). A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 2–6.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, *11*, 625–656.

Bonet, B. (2013). An admissible heuristic for SAS^+ planning obtained from the state equation. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2268–2274.

Bonet, B., & van den Briel, M. (2014). Flow-based heuristics for optimal planning: Landmarks and merges. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 47–55.

Bron, C., & Kerbosch, J. (1973). Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, *16*(9), 575–576.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artif. Intell.*, *69*(1-2), 165–204.

Cresswell, S., Fox, M., & Long, D. (2002). Extending TIM domain analysis to handle ADL constructs. In *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*.

Edelkamp, S., & Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In *Recent Advances in AI Planning, 5th European Conference on Planning (ECP)*, pp. 135–147.

Fikes, R., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, *2*(3/4), 189–208.

Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *J. Artif. Intell. Res. (JAIR)*, *9*, 367–421.

Gerevini, A., & Schubert, L. K. (1998). Inferring state constraints for domain-independent planning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI, IAAI)*, pp. 905–912.

Gerevini, A., & Schubert, L. K. (2000). Discovering state constraints in DISCOPLAN: some new results. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence (AAAI, IAAI)*, pp. 761–767.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15.

Haslum, P. (2009). $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 354–357.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 140–149.

Helmert, M. (2006). The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)*, *26*, 191–246.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, *173*(5–6), 503–535.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pp. 85–103.

Kautz, H. A., & Selman, B. (1992). Planning as satisfiability. In *Tenth European Conference on Artificial Intelligence (ECAI)*, pp. 359–363.

Kissmann, P., & Edelkamp, S. (2011). Improving cost-optimal domain-independent symbolic planning. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI)*, pp. 992–997.

Lipovetzky, N., Muise, C. J., & Geffner, H. (2016). Traps, invariants, and dead-ends. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 211–215.

Massey, B. (1999). *Directions in planning: Understanding the flow of time in planning*. Ph.D. thesis, University of Oregon.

Moon, J. W., & Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, *3*(1), 23–28.

Mukherji, P., & Schubert, L. K. (2005). Discovering planning invariants as anomalies in state descriptions. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 223–230.

Mukherji, P., & Schubert, L. K. (2006). State-based discovery and verification of propositional planning invariants. In *Proceedings of the 2006 International Conference on Artificial Intelligence (ICAI)*, pp. 465–471.

Pettersson, M. P. (2005). Reversed planning graphs for relevance heuristics in ai planning. In *Planning, Scheduling and Constraint Satisfaction: From Theory to Practice*, Vol. 117, pp. 29–38. IOS Press.

Pommerening, F., Helmert, M., Röger, G., & Seipp, J. (2015). From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)*, pp. 3335–3341.

Pommerening, F., Röger, G., Helmert, M., & Bonet, B. (2014). LP-based heuristics for cost-optimal planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*.

Rintanen, J. (2000). An iterative algorithm for synthesizing invariants. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence (AAAI, IAAI)*, pp. 806–811.

Rintanen, J. (2008). Regression for classical and nondeterministic planning. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, pp. 568–572.

Seipp, J., Pommerening, F., & Helmert, M. (2015). New optimization functions for potential heuristics. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 193–201.

Sideris, A., & Dimopoulos, Y. (2010). Constraint propagation in propositional planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 153–160.

Suda, M. (2013). Duality in STRIPS planning. *CoRR*, *abs/1304.0897*.

Torralba, Á., & Alcázar, V. (2013). Constrained symbolic search: On mutexes, BDD minimization and more. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SOCS)*, pp. 175–183.

Torralba, Á., Alcázar, V., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014). SymBA*: A symbolic bidirectional A* planner. In *International Planning Competition (IPC)*, pp. 105–108.

Torreño, A., Onaindia, E., & Sapena, O. (2014). FMAP: distributed cooperative multi-agent planning. *Appl. Intell.*, *41*(2), 606–626.