Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Mutual Exclusion State Invariants in Classical Planning

Doctoral Thesis

Daniel Fišer

Ph.D. programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering
Supervisor: Ing. Anotnín Komenda, Ph.D.
Co-Supervisor: RNDr. Lukáš Chrpa, Ph.D.

Prague, May 2020

Acknowledgments

I would like to thank Jan Faigl for helping me with my return to the university after my short intermission in industry, and I would like to thank Michal Pěchouček for welcoming me in Department of Computer Science and AI Center. I want to thank my advisor Antonín Komenda for introducing me to the beautiful area of classical planning and for all the invaluable discussions we had during my studies. Without his support and help, this work would not be possible. I thank my colleagues and co-authors Michal Štolba, Álvaro Torralba, Alexander Shleyfman, Rostislav Horčík, and Lukáš Chrpa for helping me better understand the intricacies of planning. I thank my colleague and friend Vojtěch Vonásek for always reminding me that scientific research suppose to be, above all, joyful endeavour. I thank my friend Petr Pošvic for his never ending support and his willingness to pretend that my occasional rambling on about planning problems is interesting. Lastly, I want to thank my friends Pepa, Monča, Danuška, Mikula, Pupa, Ivka, and Ivanka for their support and love, and for making my life happier.

Abstract

The solution to a classical planning problem is a sequence of operators leading from the initial state to one of the goal states. Classical planning problems are described in a domain-independent manner, so automatic extraction of structural information can provide useful guidance for planners. In this work, we focus on the structural information in a form of mutual exclusion state invariants, namely mutexes and mutex groups. Mutex describes a set of facts that cannot co-occur in the same reachable state, and mutex group describes a set of facts out of which at most one can be part of any reachable state.

In this thesis, we discuss basic properties of mutexes and mutex groups and we show that the relation between them can be described in terms of graph cliques. We analyze the complexity of inference of mutex groups and we prove that the inference of the maximum sized mutex group is as hard as determining the existence of a plan (PSPACE-Complete).

Based on these findings, we introduce a new subclass of mutex groups that we call fact-alternating mutex groups (fam-groups). We provide an in-depth analysis of the structure of fam-groups and we show that fam-groups can be used for detection of unreachable operators and operators that can produce only dead-end state. We prove that the inference of the maximum sized fam-group is NP-Complete and we introduce an inference algorithm based on repeated solving of integer linear program that is complete with respect to all maximal fam-groups. Furthermore, we propose an algorithm for removing operators and facts that cannot be part of any plan.

We formalize the inference of mutex groups in the lifted (PDDL) representation of planning problems and we prove that the most commonly used translator from the Fast Downward (FD) planning system actually infers fam-groups. We show that the algorithm for removing operators and facts using fam-groups can be utilized on the lifted level during the translation from PDDL to the ground representation (STRIPS). Moreover, we propose an improved inference algorithm for lifted fam-groups that produces a richer set of fam-groups than the FD translator.

As another example of applicability of mutual exclusion state invariants, we propose an improvement of well-known potential heuristics. We show that the mutex-based disambiguations of the goal and preconditions of operators leads to a less constrained formulation of potential heuristics yielding higher admissible estimates. Also, we propose several new optimization functions for potential heuristics where we use mutexes to more accurately estimate the number of reachable states containing certain sets of facts.

In the last part of the thesis, we introduce the notion of operator mutex as a set of operators that cannot co-occur in the same (strongly) optimal plan. We propose four different methods for inference of operator mutexes and we show how operator mutexes can be combined with structural symmetries to remove redundant operators.

All of the aforementioned is accompanied with a comprehensive experimental evaluation on the standard benchmark set.

Keywords: classical planning, invariant, mutex, mutex group

Anotace

Řešením problémů v klasickém plánování je sekvence operátorů vedoucí z iniciálního stavu do jednoho z cílových stavů. Problémy v klasickém plánování jsou popsány doménově-nezávisle, takže automatická extrakce strukturálních informací může výrazně napomoci hledání řešení. Zde se zaměříme na strukturální informace ve formě stavových invariant vzájemného vyloučení, konkrétně na tzv. mutexy a mutex grupy. Mutex popisuje množinu faktů, které se nemohou společně vyskytnou v dosažitelném stavu, a mutex grupa popisuje množinu faktů, z nichž maximálně jeden může patřit do dosažitelného stavu.

V této práci prodiskutujeme základní vlastnosti mutexů a mutex grup a ukážeme, že vztah mezi nimi lze popsat přes kliky v grafu. Zanalyzujeme výpočetní složitost odvozování mutex grup a dokážeme, že odvození mutex grup sestávající z maximálního možného počtu faktů je stejně těžké jako rozhodování o existenci plánu (PSPACE-úplné).

Na základě toho pak zavedeme novou podtřídu mutex grup, kterou nazveme fact-alternating mutex grupy (fam-grupy). Následná analýza fam-grup ukáže, že lze fam-grupy použít pro detekci nedosažitelných operátorů a operátorů, které mohou vést jen do dead-end stavů. Dokážeme, že odvozování fam-grup je, na rozdíl od obecných mutex grup, NP-úplné a představíme algoritmus pro odvozování fam-grup založený na opakovaném řešení celočíselného lineárního programu, který je kompletní vzhledem k maximálním fam-grupám. Dále navrhneme algoritmus pro odebírání operátorů a faktů, které nemohou být součástí žádného plánu.

Formalizujeme odvozování mutex grup v tzv. liftované (PDDL) reprezentaci a dokážeme, že nejčastěji používaný překladač z plánovacího systému Fast Downward (FD) ve skutečnosti odvozuje fam-grupy. Ukážeme, že algoritmus pro odebírání operátoru a faktů s pomocí fam-grup lze aplikovat v liftované podobě během překladu z PDDL do tzv. ground reprezentace (STRIPS). Dále navrhneme vylepšení stávajícího algoritmu pro odvozování liftovaných fam-grup, který produkuje bohatší množinu fam-grup než překladač z FD.

Jako další příklad použití mutexů a mutex grup pak navrhneme vylepšení dobře známých potenciálních heuristik. Ukážeme, že na mutexech založené zjednoznačnění cíle a podmínek operátorů vede na méně omezenou formulaci potenciálních heuristik, což následně vede na vyšší přípustné odhady. Dále pak navrhneme několik nových optimalizačních funkcí pro potenciální heuristiky, které využívají mutexy k přesnějšímu odhadu počtu dosažitelných stavů obsahující určité podmnožiny faktů.

V poslední části práce zavedeme tzv. operátor mutexy jako množiny operátorů, jež se nemohou společně objevit ve stejném (silně) optimálním plánu. Navrhneme čtyři odvozovací metody a ukážeme jak je lze zkombinovat se strukturálními symetriemi tak, abychom byli schopni odstranit nadbytečné operátory.

Vše z výše uvedeného je doplněno komplexní experimentální evaluací na standardním plánovacím data setu.

Klíčová slova: klasické plánování, invariant, mutex, mutex grupa

Contents

Chapter 1

Introduction

Classical planning deals with the problem of finding a sequence of actions (sometimes also called operators) with discrete and deterministic effects leading from the initial state to one of the goal states in a fully observable world. Among problems that can be tackled by classical planning are logistic problems coordinating the transportation of goods, organic synthesis problems looking for sequences of reactions producing the target molecules, cybersecurity problems modelling actions of adversary agents, playing games like FreeCell or Spider solitaire, or solving various combinatioral puzzles. Classical planning problems are described in the domain-independent manner, so automatic extraction of a structural information is vital for solving these problems. This thesis is focused on a structural information in a form of state invariants which are certain intrinsic properties of a planning task that hold in all states reachable from the initial state, in particular on the *mutual exclusion* state invariants.

A mutual exclusion (mutex) invariant states that a certain set of facts cannot be part of any reachable state (Bonet & Geffner, 2001). Mutexes have many applications in classical planning. For example, operators with preconditions containing a mutex cannot be reached by any sequence of operators applied on the initial state and therefore they can be safely removed from the planning task (e.g., Alcázar & Torralba, 2015; Fišer & Komenda, 2018; Fišer et al., 2019). This can lead to substantially smaller planning tasks that are easier to solve.

When a planner finds a solution to the planning task, the solution itself certifies that the planning task is solvable. However, certifying that the planning task is unsolvable requires a different approach. Eriksson et al. (2017, 2018) tackles this problem by constructing a certificate that describes the reachable part of the state space while showing that the goal does not belong there. Mutexes are directly applicable here because they describe the complement to the reachable part of the state space and thus useful for the construction of unsolvability certificates.

In regression planning, i.e., planning in the backward direction from a goal towards the initial state, mutexes can be used to extend preconditions of operators with facts that must be part of the states where the operators are applicable even though these facts are not explicitly stated in their preconditions, and similarly can be extended the goal specification. This process is called disambiguation (Alcázar et al., 2013). It can improve heuristic estimates because it allows to consider facts that would be otherwise ignored, and it reduces the number of considered states, because more dead-end states are recognized.

In the SAT-based planning, the planning task is solved by fixing the length of plans (or the span of parallel plans) and reformulating the problem as a SAT instance (e.g., Rintanen et al., 2006; Rintanen, 2012; Chen et al., 2007, 2009; Huang et al., 2012). Mutexes are important for improving efficiency of the encoding, because they can be encoded as constraints which in turn helps the solver to identify unreachable states.

Closely related to mutexes are state invariants called *mutex groups*. A mutex group is a set of facts out of which maximally one can be part of any reachable state. Classical planning problems are usually described in the Planning Domain Definition Language (PDDL) (McDermott, 2000), which is a schematic language that allows to use objects, predicates, and (parametrized) actions. Most planners, however, require a non-schematic (ground) representation such as STRIPS (Fikes & Nilsson, 1971) or finite domain representation (FDR or SAS$^+$) (Bäckström & Nebel, 1995). So most planners need to translate the input PDDL planning task into STRIPS (where states are represented as sets of facts) or FDR (where states are assignments to variables) before they can start to solve the problems. Mutex groups consist of facts that cannot occur together in any reachable state, so they represent a crucial piece of structural information needed to determine how to construct variables in FDR (Helmert, 2009; Fišer & Komenda, 2018; Fišer, 2020).

A richer set of mutex groups usually leads to a more concise (and information-rich) representation, which in turn can have profound effect on other parts of a planning process that utilize FDR variables. For example, abstraction heuristics such as merge-and-shrink heuristics (Helmert et al., 2014; Sievers et al., 2014) or pattern databases (Culberson & Schaeffer, 1996; Edelkamp, 2001) start with atomic projections into individual FDR variables and then proceed with the construction of abstractions using synchronized products and further abstractions. Having large FDR variables consisting of many facts can substantially speed-up construction of abstractions, because even the atomic projections cover a large portion of facts. And knowing which facts cannot occur together in any state can help to identify which parts of the state space are not reachable, which can lead to more informed and accurate heuristics.

In this work, we focus mainly on the theoretical analysis of mutex groups and the application of mutex groups that could be beneficial for most planning techniques.

Theoretical Analysis of Mutex Groups We start with the analysis of mutexes and mutex groups in general and we will show that the relation between mutexes and mutex groups can be described in terms of cliques in graphs. Then we will show that the inference of mutex groups in general is as hard as planning (PSPACE-Complete). This, on one hand, means that mutex groups have a prospect of providing an information-rich structural information about planning tasks. On the other hand, this also means it might be very hard to derive mutex groups from the description of a planning task to such extent that it might be better to try to solve the planning task without wasting any time on the inference of mutex groups in the first place. The next logical step is to look for asymptotically "easier" subclasses of mutex groups, because it might help us better understand what tools it is reasonable to use for the inference and what type of structural information we can expect to find.

Fact-Alternating Mutex Groups We found a certain polynomially verifiable subclass of mutex groups, which we named fact-alternating mutex groups (fam-groups). Fact-alternating mutex groups are defined over the input description of the planning task rather than over all reachable states. This definition immediately provides a simple polynomial verification procedure, because we can check whether a certain set of facts is a fam-group by comparing that set against all operators and the initial state. Moreover, we show that the inference of the maximum sized fam-group is only NP-Complete, i.e., fam-groups are indeed a special case of mutex groups that is "easier".

This also leads to a straightforward formulation of an integer linear program (ILP) whose solution corresponds to a fam-group in the given planning task. The fixpoint algorithm that is able to provide a complete set of all (maximal) fam-groups is then only a simple variation on the said formulation in the ILP. One benefit of the inference algorithm we propose is that it can be easily modified into an anytime algorithm that is able to produce a set of fam-groups up to some predefined size or until some time limit is reached. This might be useful in practical applications.

The reason we have chosen the name "fact-alternating" is its main property: The facts from a fam-group can change from one to another from state to state through application of an operator (i.e., they alternate between each other), but once we reach a state that does not contain any fact from the fam-group, none of the following states will contain any fact from the same fam-group either. This simple property allows us to detect operators that cannot be part of any plan and thus can be removed from the planning task. Therefore, we are able to reduce the size of planning tasks using fam-groups as a preprocessing step, which is potentially beneficial for most planning techniques.

To better understand the structure of fam-groups and given the tight relationship between mutexes and mutex groups, we focused our attention on the description of mutexes that fam-groups consist of. We found that the well-known h^2 heuristic (Haslum & Geffner, 2000) produces a set of mutex pairs (i.e., mutexes consisting of exactly two facts) that is a superset of mutex pairs obtained from the decomposition of fam-groups. This means that we can find the building blocks of fam-groups in a polynomial number of steps, which nicely encloses our theoretical analysis, because it connects NP-Completeness of fam-groups with the relation between mutexes and mutex groups in terms of graph cliques.

Lifted Fact-Alternating Mutex Groups In contrast to our approach, the most commonly used algorithm for inference of mutex groups from the Fast Downward planner (Helmert, 2009) works not on the ground (STRIPS, FDR) representation, but on the lifted (PDDL) representation of planning tasks. We analyzed Helmert's approach and we found out that these lifted mutex groups are, in fact, lifted fam-groups, i.e., when grounded into STRIPS they form fam-groups. We prove this formally and we propose several improvements of Helmert's algorithm that allow to find even richer set of lifted fam-groups. Then, building on our previous findings, we propose an improvement of the grounding process (i.e., the process of translation from PDDL to STRIPS) that utilize properties of fam-groups so that we are able to reduce the planning task during the grounding. That is, we are able to detect that certain operators cannot be part of any plan even before the whole STRIPS representation is constructed from the input PDDL description.

Disambiguation and Potential Heuristics Consider the following example. Suppose we have a planning task with facts A, B, C and D, and we know that every reachable state must contain exactly one of A, B, or C (i.e., {A, B, C} is a special case of "exactly-one" mutex group). Furthermore, suppose that we also know that every reachable state containing D cannot contain B or C (i.e., {D, B} and {D, C} are mutexes). From this, we can easily infer that every state containing D must also contain A.

This process of extending a partial information about states is called disambiguation. It was described by Alcázar et al. (2013) in the context of regression planning, and it proved to be a crucial part of one of the strongest pruning techniques used in classical planning that is based on the h^2 heuristic (Alcázar & Torralba, 2015).

We use and extend the notion of disambiguation in the context of potential heuristics (Pommerening et al., 2015a; Seipp et al., 2015). We show that utilization of mutexes can significantly improve heuristic estimates provided by different types of potential heuristics. This demonstrates that studying mutual exclusion between facts pays off not only in the context of pruning of planning tasks as a preprocessing step (which may be beneficial for all planning techniques), but also in some specific planning techniques—in this case, in the construction of a heuristic function.

Operator Mutexes In the last part of this thesis, we move our attention to a mutual exclusion between operators. We introduce a new concept of a strong operator mutex (op-mutex) describing a set of operators that cannot be part of the same strongly optimal plan (i.e., the shortest of all optimal plans). We propose several inference methods based on well-known planning techniques and we experimentally verify that op-mutexes can be found in a sizable number of domains from the standard benchmark set. Lastly, we combine the notion of structural symmetries (e.g., Fox & Long, 1999; Pochter et al., 2011; Shleyfman et al., 2015), previously used in different parts of classical planning, with op-mutexes in order to prove which operators can be safely removed from the planning task so that at least one strongly optimal plan is preserved.

1.1 Outline and Contributions

The thesis is structured as follows.

In Chapter 2, we discuss previous work related to the inference of state invariants and in Chapter 3 we lay down a necessary technical background for this work and we describe a running example used throughout this thesis.

In Chapter 4, we describe the basic properties of mutexes and mutex groups and we show that the relation between them can be described in terms of cliques in graphs. We analyze the complexity of inference of mutex groups and we prove that the inference of the maximum sized mutex group is PSPACE-Complete, i.e., as hard as deciding whether a planning task has a plan.

In Chapter 5, we introduce a new subclass of mutex groups called *fact-alternating mutex groups* (fam-groups) which are defined not over the reachable states but over the input planning task. We show that fam-groups can be used for the detection of not only unreachable states and operators, but also dead-end states and operators that can produce only dead-end states. We provide an in-depth analysis of the complexity of inference of fam-groups and we show that the inference of the maximum sized fam-group is only NP-Complete (in contrast to the PSPACE-Completeness of mutex groups). Furthermore, we provide an algorithm for the inference of fam-groups based on a repeated solving of Integer Linear Program that is complete with respect to maximal fam-groups.

In Chapter 6, we show how to use fam-groups to remove facts and operators that cannot be part of any plan. We perform a detailed experimental evaluation of the pruning algorithm and its consequent impact on the number of solved task on the standard set of benchmarks.

In Chapter 8, we show that potential heuristics (Pommerening et al., 2015a; Seipp et al., 2015) can be made significantly more accurate if mutexes are utilized. Since mutexes tell us what is not reachable, we use this information to describe more precisely preconditions of operators and the goal specification than it is explicitly stated in the input description of the planning task. This often leads to higher heuristic estimates of potential heuristics and consequently to the higher number of solved tasks.

In Chapter 9, we extend the notion of mutexes into operators. We say that a set of operators is an operator mutex if they cannot be part of the same (strongly optimal) plan. We introduce several different methods for the inference of operator mutexes, and we utilize structural symmetries (e.g., Fox & Long, 1999; Pochter et al., 2011; Shleyfman et al., 2015) to find out which operators can be removed from the planning task so that at least one (strongly optimal) plan is preserved.

Finally, we conclude with Chapter 10.

1.2 Relation to Published Work

Most of this thesis is adapted from the following publications:

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

   This is the core publication that deals with mutexes, mutex groups, and fact-alternating mutex groups. Chapters 2 to 6 are adapted from this paper.

2. Fišer, D. & Komenda, A. (2018). Fact-alternating mutex groups for classical planning (extended abstract). In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, (pp. 5603–5607).

   This is a considerably shortened version of the previous paper published in the journal track of the conference IJCAI.

3. Fišer, D. & Torralba, Á. & Shleyfman, A. (2019). Operator mutexes and symmetries for simplifying planning tasks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, (pp. 7586–7593).

   This paper introduces the notion of operator mutexes. Chapter 9 is adapted from this paper.

4. Fišer, D. (2020). Lifted fact-alternating mutex groups and pruned grounding of classical planning problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, (Accepted).

   This paper deals with lifted fact-alternating mutex groups. Chapter 7 is adapted from this paper.

5. Fišer, D. & Horčík, R. & Komenda, A. (2020). Strengthening potential heuristics with mutexes and disambiguations. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, (Accepted).

   This paper describes the notion of multi-fact disambiguation and describes several improvements of potential heuristics. Chapter 8 is adapted from this paper.

Chapter 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 & 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 & 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 & Long (1998) and further extended by Cresswell et al. (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 & 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 h^{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. h^{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 h^{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 (h^{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.

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. Bernardini et al. (2018) proposed an extension of the Helmert's algorithm into temporal planning.

Chapter 3

Background

3.1 Technical Background

A classical planning task consists of a single fully described initial state, a set of deterministic operators that can alter state of the world, and a description of the goal. Formally, we can describe classical planning tasks using the STRIPS formalism (Fikes & Nilsson, 1971).

Definition 3.1. A **STRIPS planning task** Π is specified by a tuple $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, where $\mathcal{F} = \{f_1, \dots, f_n\}$ is a set of facts, and $\mathcal{O} = \{o_1, \dots, o_m\}$ is a set of grounded operators. A **state** $s \subseteq \mathcal{F}$ is a set of facts, $s_I \subseteq \mathcal{F}$ is an **initial state** and $s_G \subseteq \mathcal{F}$ is a **goal** specification.

An **operator** o is a tuple $o = \langle \mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o), \mathrm{c}(o) \rangle$, 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, and $\mathrm{c}(o) \in \mathbb{R}^+_0$ is a cost of the operator. All operators are well-formed, i.e., $\mathrm{add}(o) \cap \mathrm{del}(o) = \emptyset$ and $\mathrm{pre}(o) \cap \mathrm{add}(o) = \emptyset$. An operator o is **applicable** in a state s if $\mathrm{pre}(o) \subseteq s$. The **resulting state** of applying an applicable operator o in a state s is the state $o[\![s]\!] = (s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o)$. A state s is a **goal state** iff $s_G \subseteq s$.

A sequence of operators $\pi = \langle o_1, \dots, o_n \rangle$ is applicable in a state s_0 if there are states $s_1, \dots, s_n$ such that o_i is applicable in s_{i-1} and $s_i = o_i[\![s_{i-1}]\!]$ for $i \in \{1, \dots, n\}$. The resulting state of this application is $\pi[\![s_0]\!] = s_n$ and $\mathrm{c}(\pi) = \sum_{i=1}^{n} \mathrm{c}(o_i)$ denotes the cost of this sequence of operators. By $|\pi|$ we denote the length of the sequence. A sequence of operators π is called an **s-plan** iff π is applicable in a state s and $\pi[\![s]\!]$ is a goal state. An s-plan π is called **optimal** if its cost is minimal among all s-plans and it is called **strongly optimal** if it is an optimal plan and it contains the minimum number of operators among optimal plans. s_I-plan is called simply a **plan**.

Let $F \subseteq \mathcal{F}$ denote a set of facts, and let $f \in \mathcal{F}$ denote a fact. F (f) is **reachable from state** s if there exists an operator sequence π such that π is applicable in s and $F \subseteq \pi[\![s]\!]$ ($f \in \pi[\![s]\!]$). F (f) reachable from s_I is called simply reachable. F (f) that is not reachable (from s) is called **unreachable** (from s). F (f) is **relaxed reachable from state** s if there exists an operator sequence $\pi = \langle o_1, \dots, o_n \rangle$ such that $F \subseteq \pi'[\![s]\!]$ ($f \in \pi'[\![s]\!]$) where $\pi' = \langle o'_1, \dots, o'_n \rangle$ and $o'_i = \langle \mathrm{pre}(o_i), \mathrm{add}(o_i), \emptyset, \mathrm{c}(o_i) \rangle$ for every $i \in \{1, \dots, n\}$. F (f) relaxed reachable from s_I is called simply relaxed reachable. F (f) that is not relaxed reachable (from s) is called **relaxed unreachable** (from s). An operator o is (relaxed) reachable (from state s) iff $\mathrm{pre}(o)$ is (relaxed) reachable (from s). The set of all reachable states is denoted by $\mathcal{R}_\Pi$. A state s is a **dead-end state** iff $s_G \not\subseteq s$ and there is no s-plan.

We will also use some auxiliary notation. Given a set of facts $X \subseteq \mathcal{F}$ and a number k, $\binom{X}{k}$ denotes a set of all subsets of X of size exactly k, i.e., $\binom{X}{k} = \{F \mid F \subseteq X, |F| = k\}$

Most of this thesis uses the STRIPS formalism, but we will also use and refer to the finite domain representation (FDR) of planning tasks (Bäckström & Nebel, 1995), where state are represented as assignments to variables with finite domains.

Definition 3.2. An **FDR planning task** $\Pi^{\mathcal{V}}$ is specified by a tuple $\Pi^{\mathcal{V}} = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$. $\mathcal{V}$ is a finite set of **variables**, each variable $V \in \mathcal{V}$ has a finite **domain** $\mathrm{dom}(V)$. A **fact** $\langle V, v \rangle$ is a pair of a variable $V \in \mathcal{V}$ and one of its values $v \in \mathrm{dom}(V)$. The set of all facts is denoted by $\mathcal{F} = \{\langle V, v \rangle \mid V \in \mathcal{V}, v \in \mathrm{dom}(V)\}$, and the set of facts of the variable V is denoted by $\mathcal{F}_V = \{\langle V, v \rangle \mid v \in \mathrm{dom}(V)\}$. A **partial state** p is a variable assignment over some variables $\mathrm{vars}(p) \subseteq \mathcal{V}$. We write $p[V]$ for the value assigned to the variable $V \in \mathrm{vars}(p)$ in the partial state p. We also identify p with the set of facts contained in p, i.e., $p = \{\langle V, p[V] \rangle \mid V \in \mathrm{vars}(p)\}$. A partial state s is a **state** if $\mathrm{vars}(s) = \mathcal{V}$. I is an **initial state**. G is a partial state called **goal**, and a state s is a **goal state** iff $G \subseteq s$. Let p, t be partial states. We say that t **extends** p if $p \subseteq t$.

$\mathcal{O}$ is a finite set of **operators**, each operator $o \in \mathcal{O}$ has a precondition $\mathrm{pre}(o)$ and effect $\mathrm{eff}(o)$, which are partial states over $\mathcal{V}$, and a cost $\mathrm{c}(o) \in \mathbb{R}_0^+$. An operator o is **applicable** in a state s iff $\mathrm{pre}(o) \subseteq s$. The **resulting state** of applying an applicable operator o in a state s is another state $o[\![s]\!]$ such that $o[\![s]\!][V] = \mathrm{eff}(o)[V]$ for every $V \in \mathrm{vars}(\mathrm{eff}(o))$, and $o[\![s]\!][V] = s[V]$ for every $V \in \mathcal{V} \setminus \mathrm{vars}(\mathrm{eff}(o))$.

A sequence of operators, ((strongly) optimal) (s-)plan, dead-end state, and (relaxed) (un)reachability is defined analogously to STRIPS. The set of all reachable states is denoted by $\mathcal{R}_{\Pi^{\mathcal{V}}}$.

Note that the we re-use the same symbol for the set of facts ($\mathcal{F}$) as is used for the STRIPS representation, because it describes, essentially, the same concept, and the same symbol for the set of operators ($\mathcal{O}$) or for the application of an operator. However, it will always be clear from the context to which formalism we refer to.

We also use a standard definition of a graph and a graph clique.

Definition 3.3. An undirected simple **graph** G is a tuple $G = \langle V, E \rangle$, where V denotes a set of vertices and E denotes a set of edges such that each edge $\{v_i, v_j\} \in E$ connects two different vertices ($v_i \neq v_j$) and there are no two edges connecting the same vertices. A non-empty set $C \subseteq V$ of vertices of G forms a **clique** if each vertex of C is connected by an edge to every other vertex of C. A clique that is not a subset of any other clique is called a **maximal clique**. A **maximum clique** in G is a clique such that there is no other clique consisting of more vertices.

3.2 Running Example

Consider the following simple example of the `gorilla-feeding` planning task depicted in Figure 3.1. 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

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_I): `(at b)`, `(hungry)`

Goal (s_G): `(fed)`

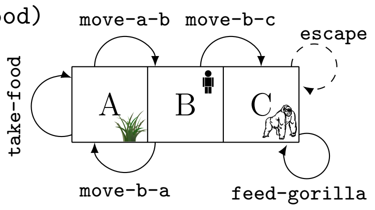

Figure 3.1: The `gorilla-feeding` planning task.

or it was fed, and `(carry-food)` specifies whether the zookeeper carries the food for the gorilla.

The operators in Figure 3.1 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 pre(o) = {`(at c)`, `(hungry)`, `(carry-food)`}, one add effect add(o) = {`(fed)`}, and two delete effects 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_I = {`(at b)`, `(hungry)`} meaning that the zookeeper starts at square `B` and the gorilla is hungry. The goal s_G = {`(fed)`} is to feed the gorilla.

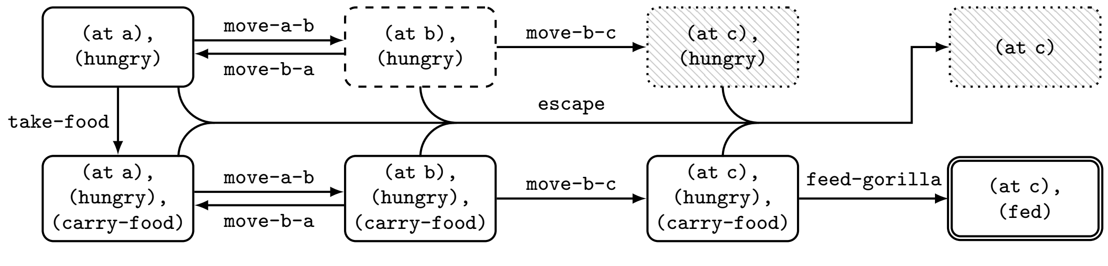

Figure 3.2: Reachable states and transitions between reachable states in the `gorilla-feeding` planning task.

All eight reachable states of the planning task are depicted in Figure 3.2 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 3.2 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.

Chapter 4

Mutex and Mutex Group

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. A closely related to mutexes is another type of state invariant called mutex group. A mutex group is a set of facts out of which at most one is true in any reachable state, i.e., a mutex group consists of facts that are pairwise mutually exclusive.

In this chapter, we discuss basic properties of mutexes and mutex groups and the relation between them (Section 4.1). Furthermore, we provide a complexity analysis showing that the inference of the maximum sized mutex group (and thus also a complete set of mutex pairs) is PSPACE-Complete (Section 4.2).

For this chapter, let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a STRIPS planning task.

4.1 Relation Between Mutexes and Mutex Groups

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.

Definition 4.1. A **mutex** $M \subseteq \mathcal{F}$ is a set of facts such that for every reachable state $s \in \mathcal{R}_\Pi$ it holds that $M \not\subseteq s$.

It is easy to see that every superset of a mutex is also mutex, but a subset of mutex is not necessarily a mutex. For example looking at the reachable states of our running example (Figure 3.2), {`(at b)`, `(hungry)`, `(fed)`} is a mutex because there is no reachable state containing all these facts, but {`(at b)`, `(hungry)`} is not a mutex.

Proposition 4.2. *If $M \subseteq \mathcal{F}$ is a mutex, then every $N \subseteq \mathcal{F}$ such that $N \supseteq M$ is a mutex.*

Proof. It follows directly from Definition 4.1. □

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. And a maximal mutex group is a mutex group that cannot be extended by any fact and remain a mutex group.

Definition 4.3. A **mutex group** $M \subseteq \mathcal{F}$ is a set of facts such that for every reachable state $s \in \mathcal{R}_\Pi$ 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**.

An example of a mutex group in our running example planning task is the mutex group $\{$`(at a)`, `(at b)`, `(at c)`$\}$ describing that the zookeeper must be at one of the positions. Another, less obvious, examples of mutex groups are $\{$`(at a)`, `(at b)`, `(fed)`$\}$ or $\{$`(carry-food)`, `(fed)`$\}$.

In contrast to mutexes, every subset of a mutex group is also a mutex group. And every mutex group consisting of at least two facts is a mutex.

Proposition 4.4. *If $M \subseteq \mathcal{F}$ is a mutex group, then every $N \subseteq M$ is a mutex group.*

Proof. It follows directly from Definition 4.3. □

Proposition 4.5. *If $M \subseteq \mathcal{F}$ such that $|M| \geq 2$ is a mutex group, then M is a mutex.*

Proof. It follows directly from Definition 4.3 and Definition 4.1. □

Where the both notions coincide is on mutexes containing exactly two facts, which we call **mutex pairs**. That is, a mutex pair is both a mutex and a mutex group at the same time, because if two facts form a mutex then every reachable state can contain at most one fact from this mutex, which is exactly the definition of a mutex group. And if two facts form a mutex group then it is a mutex by Proposition 4.5.

Proposition 4.6. *Let $M \subseteq \mathcal{F}$, $|M| = 2$, denote a pair of facts. M is a mutex iff M is a mutex group.*

Proof. If M is a mutex, then by Definition 4.1 it holds for every reachable $s \in \mathcal{R}_\Pi$ that $M \not\subseteq s$. And since $|M| = 2$ it follows that $|M \cap s| \leq 1$. The other direction follows from Proposition 4.5. □

Since every subset of a mutex group is also a mutex group, every mutex group can be decomposed into a set of mutex pairs by enumerating all pairs facts from the mutex group. Now we show that we can also compose a mutex group from a set of mutex pairs. In other words, we show that a set of facts is a mutex group if and only if every pair of facts from the mutex group is a mutex pair.

Proposition 4.7. *Let $M \subseteq \mathcal{F}$ denote a set of facts such that $|M| \geq 2$. M is a mutex group iff every $P \in \binom{M}{2}$ is a mutex group (mutex).*

Proof. From left to right follows from Proposition 4.4 (and Proposition 4.6). To prove the other direction by contradiction, let us assume that every $P \in \binom{M}{2}$ 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 $P \in \binom{M}{2}$ is a mutex group because $\{f_1, f_2\}$ belongs to $\binom{M}{2}$ by definition. □

Proposition 4.7 describes a close relationship between mutex pairs and mutex groups. The next question is how to actually construct mutex groups from mutex pairs. In the following, we show that we can construct a graph from a given set of mutex pairs where each vertex corresponds to a fact and each edge corresponds to a mutex pair, and then every clique in this graph corresponds to a mutex group.

Definition 4.8. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a STRIPS planning task, and let $\mathcal{M} \subseteq 2^{\mathcal{F}}$ denote a set of mutex pairs. A **mutex pair graph** for $\mathcal{M}$, denoted by $\mathrm{MPG}_{\mathcal{M}}$, is a simple undirected graph $\mathrm{MPG}_{\mathcal{M}} = \langle V, E \rangle$, where each vertex corresponds to a fact, i.e., $V = \{v_f \mid f \in \mathcal{F}\}$, and each edge corresponds to a mutex pair, i.e., $E = \{\{v_f, v_{f'}\} \mid \{f, f'\} \in \mathcal{M}\}$.

Proposition 4.9. *Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a STRIPS planning task, let $\mathcal{M} \subseteq 2^{\mathcal{F}}$ denote a set of mutex pairs, let $\mathrm{MPG}_{\mathcal{M}} = \langle V, E \rangle$ denote a mutex pair graph for $\mathcal{M}$, and let $C = \{v_{f_1}, \dots, v_{f_n}\} \subseteq V$ denote a set of vertices of $\mathrm{MPG}_{\mathcal{M}}$. If C is a clique, then $M = \{f_1, \dots, f_n\}$ is a mutex group in Π.*

Proof. If C is a clique, then every $\{f, f'\} \in M$, $f \neq f'$, is a mutex by Definition 4.8. So it follows from Proposition 4.7 that M is a mutex group. □

Proposition 4.9 describes how can be mutex groups inferred for STRIPS planning tasks with any algorithm that produces mutex pairs. The most commonly used method for inference of mutexes is the h^m heuristic (Haslum & Geffner, 2000), usually only for $m = 2$. So to find mutex groups, we can run h^2 to produce a set of mutex pairs $\mathcal{M}$, then we can construct a mutex pair graph $\mathrm{MPG}_{\mathcal{M}}$ and finally every clique in $\mathrm{MPG}_{\mathcal{M}}$ corresponds to a mutex group. Note that inference of cliques in graphs is NP-Complete problem (Karp, 1972) so even with a given set of mutex pairs it is still a hard problem to find mutex groups.

4.2 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.

Definition 4.10. Let $\mathcal{M}$ denote a set of all mutex groups. M is a **maximum mutex group** iff $M \in \mathcal{M}$ and $|M| \geq |N|$ for every $N \in \mathcal{M}$.

Definition 4.11. Maximum-Mutex-Group **decision problem**: Given a planning task Π and an integer k, does Π contain a mutex 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 4.1, but not the other way around. Maximum-Mutex-Group is a decision problem corresponding to the task of finding a maximum mutex group.

Definition 4.12. Plan-Exist **decision problem**: Given a planning task Π, determine the existence of a solution.

Now, we show (Theorem 4.17) that the Maximum-Mutex-Group is PSPACE-Complete in the following way. First, we prove that it is PSPACE-Hard (Proposition 4.15) using a polynomial reduction from Plan-Exist which is known to be PSPACE-Complete (Bylander, 1994). Then, we will present a PSPACE algorithm (Algorithm 4.1) solving the Maximum-Mutex-Group problem which leads to the conclusion that the Maximum-Mutex-Group is PSPACE-Complete. Moreover, we will show that a similar reasoning can be used to prove that the inference of a complete set of mutex pairs is also PSPACE-Complete.

Definition 4.13. Given a planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, $\Pi^M = \langle \mathcal{F}^M, \mathcal{O}^M, s_I, s_G \rangle$ 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 $\text{pre}(o^{\text{sat}}) = s_G$, $\text{del}(o^{\text{sat}}) = \{\}$, and $\text{add}(o^{\text{sat}}) = \mathcal{F}^M$.

Lemma 4.14. *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 $s_G \subseteq s$. 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} ($|N \cap s^{\text{sat}}| \geq 2$). 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 $s_G \subseteq s$, 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 $\bot$ appears in Π^M only in s^{sat}, therefore, any mutex group in Π^M can be extended by $\bot$ 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 $\bot$, 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 4.15. 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 4.14 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. □

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 4.1 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 4.6). 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 4.1 uses Plan-Exist to prove whether each pair of facts is a mutex group

Algorithm 4.1: MAXIMUM-MUTEX-GROUP

```
Input: A constant $k$, planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$
Output: "Yes" or "No"
1 Construct a complete graph
    $G = \langle V = \{v_f \mid f \in \mathcal{F}\}, E = \{\{v_f, v_{f'}\} \mid \{f, f'\} \subseteq \mathcal{F}, f \neq f'\}$;
2 for each $f, f' \in \mathcal{F}$ such that $f \neq f'$ do
3 |   if there exists a plan for $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, \{f, f'\} \rangle$ then /* PLAN-EXIST */
4 |   |   $E \leftarrow E \setminus \{v_f, v_{f'}\}$;
5 if $G$ contains a clique of size at least $k$ then return "Yes"; /* MAXIMUM-CLIQUE */
6 else return "No";
```

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 4.16. Theorem 4.17 just joins Proposition 4.15 (MAXIMUM-MUTEX-GROUP is PSPACE-Hard) and Lemma 4.16 (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.

Lemma 4.16. *Algorithm 4.1 decides* MAXIMUM-MUTEX-GROUP *using a polynomial amount of space in the size of the input.*

Proof. Algorithm 4.1 starts with a complete graph constructed from the facts as its vertices. 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 4.6 that having all mutex pairs 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 5), 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 vertex is a maximum clique as well.) □

Theorem 4.17. MAXIMUM-MUTEX-GROUP *is* PSPACE-*Complete.*

Proof. The MAXIMUM-MUTEX-GROUP is PSPACE-Hard (Proposition 4.15) and it also belongs to PSPACE (Lemma 4.16), therefore, the MAXIMUM-MUTEX-GROUP is PSPACE-Complete. □

Now we have shown that the inference of maximum mutex group in general is as hard as planning (PSPACE-Complete). Previously, we have shown that there is a close relationship between mutex pairs and mutex groups (Proposition 4.7). We can decompose a mutex group into a set of mutex pairs and, conversely, if we are given a set of mutex pairs,

Algorithm 4.2: MAXIMUM-SET-OF-MUTEX-PAIRS

```
Input: A constant $k$, planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$
Output: "Yes" or "No"
1 $n \leftarrow \binom{|\mathcal{F}|}{2}$ /* The number of all pairs of facts in the problem. */
2 for each $f, f' \in \mathcal{F}$ such that $f \neq f'$ do
3     if there exists a plan for $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, \{f, f'\} \rangle$ then /* PLAN-EXIST */
4         $n \leftarrow n - 1$;
5 if $n \geq k$ then return "Yes";
6 else return "No";
```

we can construct a mutex group from them by looking for cliques in the corresponding mutex pair graph (Proposition 4.9), which is NP-Complete. From this, it follows that the inference of mutex pairs is also as hard as planning.

Definition 4.18. MAXIMUM-SET-OF-MUTEX-PAIRS **decision problem**: Given a planning task Π and an integer k, does Π contain a at least k mutex pairs?

We prove that MAXIMUM-SET-OF-MUTEX-PAIRS is PSPACE-Complete in two steps. First, a small modification of Algorithm 4.1, depicted in Algorithm 4.2, shows that MAXIMUM-SET-OF-MUTEX-PAIRS is in PSPACE, because we can repeatedly solve the problem for each pair of facts as goals and count the number of pairs of facts that is not reachable, i.e., count the number of mutex pairs. Second, we can re-use the construction of Π^M (Definition 4.13) and Lemma 4.14 to show that MAXIMUM-SET-OF-MUTEX-PAIRS is PSPACE-Hard, because if the maximum mutex group in the problem consists of at most one fact, then such a problem does not contain any mutex pairs.

Theorem 4.19. MAXIMUM-SET-OF-MUTEX-PAIRS *is* PSPACE-*Complete.*

Proof. Algorithm 4.2 solves MAXIMUM-SET-OF-MUTEX-PAIRS within the polynomial space, because it repeatedly uses PLAN-EXIST (which is PSPACE-Complete) on the input planning task to decide whether each pair of facts is reachable and it counts the number of the unreachable pairs, i.e., the number of mutex pairs. Therefore, MAXIMUM-SET-OF-MUTEX-PAIRS belongs to PSPACE.

To show PSPACE-hardness, we reduce PLAN-EXIST to MAXIMUM-SET-OF-MUTEX-PAIRS. As we have shown in the proof of Proposition 4.15, we can translate Π to Π^M (Definition 4.13) in a polynomial number of steps. It follows from Lemma 4.14 and Proposition 4.7 that we can decide PLAN-EXIST by deciding MAXIMUM-SET-OF-MUTEX-PAIRS with $k = 1$ in the following way: If there are no mutex pairs in Π^M, then the maximum mutex group consists of at most one fact and therefore Π has a solution. If there is at least one mutex pair in Π^M, then Π^M must have a maximum mutex group of size at least 2, therefore Π does not have a solution. □

4.3 Summary

This chapter was focused on the relation between mutexes and mutex groups, and on the complexity analysis of mutex groups. We have shown that every mutex group can be decomposed into a set of mutex pairs, and that, given a set of mutex pairs, we can construct mutex groups consisting of those mutex pairs by looking for cliques in mutex pair graphs. Furthermore, we have shown that the inference of mutex groups is as hard as planning (PSPACE-Complete) and also that the same holds for the inference of mutex pairs. This paints a rather dim picture, because if we want to use mutex groups to help us solve planning tasks, we have to use an inference machinery as hard as planning in the worst case. However, in the next chapter we show that not all hope is lost. We formalize a subclass of mutex groups of which inference is NP-Complete and we show that this subclass of mutex groups is both information-rich and we can infer them in a reasonable time in practice.

Chapter 5

Fact-Alternating Mutex Group

In this chapter, we move away from the general mutex groups towards a more specific subclass. We introduce a more restricted form of mutex groups, called fact-alternating mutex groups (fam-groups), defined over the description of planning task rather than over the reachability of states (Section 5.1). We show that the inference of this subclass of mutex groups is only NP-Complete (Section 5.2). Since fam-groups are mutex groups, they can be decomposed into mutex pairs as well. To pinpoint the complexity of fam-groups more precisely, we investigate the relation between fam-groups and the h^m heuristic (Haslum & Geffner, 2000). We show that mutex pairs obtained from the decomposition of fam-groups are always a subset of mutex pairs obtained from the h^2 heuristic, but not necessarily the other way around (Section 5.3).

Lastly, we propose a novel inference algorithm that is complete with respect to all maximal fam-groups (Section 5.4) and we provide an in-depth experimental evaluation of the inferred fam-groups (Section 5.5). We compare fam-groups to the mutex groups inferred with the most commonly used algorithm from the Fast Downward planner (Helmert, 2009) and to the mutex groups constructed from the mutex pairs obtained from the h^2 heuristic. The evaluation shows that fam-groups are indeed information-rich state invariants, which we demonstrate on a use-case where we use fam-groups for construction of variables in the finite domain representation resulting in the increased number solved planning tasks on the standard benchmark set.

For this chapter, let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a STRIPS planning task.

5.1 Structure of Fact-Alternating Mutex Group

The definition of fact-alternating mutex groups (fam-groups) is based on the applicability of the operators rather than on a relation to all reachable states which makes it less complex than the general mutex group (as will be demonstrated in Section 5.2).

Definition 5.1. A **fact-alternating mutex group** (fam-group) $M \subseteq \mathcal{F}$ is a set of facts such that $|M \cap s_I| \leq 1$ and $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ for every operator $o \in \mathcal{O}$. 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 5.2. *Every fact-alternating mutex group is a mutex group.*

Proof. (By induction) The first part $|M \cap s_I| \leq 1$ 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 $\mathcal{R}_\Pi$, 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.

Formally, we first prove in Proposition 5.3 that if a state s does not contain any fact from a fam-group M, then all facts from M are relaxed unreachable from s. This also means that there is no point in looking for fam-groups that are disjoint from the initial state, because these facts can be easily detected as unreachable by a simple relaxed reachable analysis. Therefore we can safely use a more restricted constraint on the initial state $|M \cap s_I| = 1$. Second, we prove in Proposition 5.4 the fact-alternating nature of fam-groups, i.e., we show that if a state s is disjoint with a fam-group M, then all facts from M are unreachable from s.

Proposition 5.3. *Let M denote a fam-group and let s denote a state. If $M \cap s = \emptyset$, then every $f \in M$ is relaxed unreachable from s.*

Proof. (By contradiction) If M is relaxed reachable from s, then there exist two states s', s'' and an operator o such that s' is relaxed reachable from s, and $s' \cap M = \emptyset$, and $\mathrm{pre}(o) \subseteq s'$, and $s'' = s' \cup \mathrm{add}(o)$, and $s'' \cap M \neq \emptyset$. Since $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ and $\mathrm{pre}(o) \cap M \subseteq s' \cap M = \emptyset$, we can conclude that $s'' \cap M = \emptyset$. □

Proposition 5.4. *Let M denote a fam-group and let s denote a state. If $|M \cap s| = 0$, then every $f \in M$ is unreachable from s.*

Proof. It follows directly from Proposition 5.3, because it is well-known that if a fact is relaxed unreachable, then it is unreachable. □

Interestingly, we can use Proposition 5.4 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 ($|M \cap s_G| \geq 1$) 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 5.4 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 5.4.

Corollary 5.5. *Let $M \subseteq \mathcal{F}$ denote a set of facts and let s denote a state. If M is a fam-group and $|M \cap s_G| \geq 1$ and $|M \cap s| = 0$, then s is a dead-end state.*

Proof. From Proposition 5.4 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 $|M \cap s_G| \geq 1$, it follows that $s_G \not\subseteq \pi[s]$ 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 5.6. *Let $M \subseteq \mathcal{F}$ denote a set of facts, let s denote a state and let $o \in \mathcal{O}$ denote an operator applicable in s. If M is a fam-group and $|M \cap s_G| \geq 1$ and $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \geq 1$ and $|M \cap \mathrm{add}(o)| = 0$, 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 $|M \cap s_G| \geq 1$ and Corollary 5.5 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 5.1. The maximal mutex groups and maximal fam-groups are marked with a plus sign. As we have shown in Proposition 4.4 every subset of a mutex group is also a mutex group. However, an 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

[1]This is a special case of disambiguation proposed by Alcázar et al. (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 5.1: 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.

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 a singleton 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 5.7. *Let $f \in \mathcal{F}$ denote a single fact. $\{f\}$ is a fam-group iff $f \notin \mathrm{add}(o)$ for every operator $o \in \mathcal{O}$.*

Proof. First, we prove the direction from left to right by contradiction. Assuming there exists an operator o such that $f \in \mathrm{add}(o)$, also $f \in \mathrm{pre}(o)$ must hold, because the inequality $|\{f\} \cap \mathrm{add}(o)| \leq |\{f\} \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ must hold. This is in contradiction with the assumption $\mathrm{add}(o) \cap \mathrm{pre}(o) = \emptyset$ (Definition 3.1). Similarly, to prove the other direction by contradiction, we assume that $\{f\}$ is not a fam-group. Since $|\{f\} \cap s_I| \leq 1$ always holds, the inequality $|\{f\} \cap \mathrm{add}(o)| > |\{f\} \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ must hold. Therefore, $|\{f\} \cap \mathrm{add}(o)| \geq 1$, therefore, $\{f\} \in \mathrm{add}(o)$, 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 5.6, 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 5.6 is not limited to maximal fam-groups.

5.2 Complexity Analysis

The complexity analysis of fam-groups follows the complexity analysis from Section 4.2 in that we investigate the complexity of finding a maximum sized fam-group.

Definition 5.8. Let $\mathcal{M}$ denote a set of all fam-groups. M is a **maximum fam-group** iff $M \in \mathcal{M}$ and $|M| \geq |N|$ for every $N \in \mathcal{M}$.

Definition 5.9. MAXIMUM-FAM-GROUP **decision problem**: Given a planning task Π and an integer k, does Π contain a fam-group of size at least k?

To prove that MAXIMUM-FAM-GROUP is NP-Hard, we use a reduction from the decision problem corresponding to finding a maximum sized clique in a graph.

Definition 5.10. MAXIMUM-CLIQUE **decision problem**: Given a graph G and an integer k, does G contain a clique of size at least k?

Definition 5.11. Given an undirected simple graph $G = \langle V, E \rangle$, $\Pi^G = \langle \mathcal{F}^G, \mathcal{O}^G, s_I^G, \emptyset \rangle$ denotes a planning task where $\mathcal{F}^G = \{f_v \mid v \in V\} \cup \{\top\}$, $s_I^G = \{\top\}$, and $\mathcal{O}^G = \{o_{v,v'} \mid \{v, v'\} \subseteq V, v \neq v', \{v, v'\} \notin E\}$ with $\mathrm{pre}(o_{v,v'}) = \{\top\}$, $\mathrm{del}(o_{v,v'}) = \{\top\}$, and $\mathrm{add}(o_{v,v'}) = \{f_v, f_{v'}\}$.

For a given set of vertices $C \subseteq V$, $\mathcal{F}^G(C) = \{f_v \mid v \in C\}$ denotes the corresponding set of facts in Π^G.

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 5.11) 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 5.14). 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 (Lemma 5.15). This allows us to show that the maximum number of both maximal mutex groups and maximal fam-groups is the same as the maximum number of maximal cliques in a graph, and we express this number exactly (Proposition 5.16).

The main idea behind the way Π^G is constructed from G, is the following. Consider a complete graph. In such a graph, all vertices 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 vertices. 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 effects correspond to the pair of vertices 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 5.12 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 $\mathcal{F}^G(C) \cup \{\top\}$ 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 5.13, we prove an even stronger statement saying that not only all maximum fam-groups, but all maximal fam-groups contain $\top$. Therefore, Lemma 5.12 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 5.14 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 5.12. *C is a clique in G iff $M = \mathcal{F}^G(C) \cup \{\top\}$ 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 $s_I^G = \{\top\} \subseteq M$ 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

$o' \in \mathcal{O}^G$ such that $|M \cap \mathrm{add}(o')| > 1$. Since $\top$ is not part of any add effect and all add effects contain exactly two facts, it must hold that $\mathrm{add}(o') \subseteq \mathcal{F}^G(C)$. This is in contradiction with the assumption that C is a clique because all add effects are created only from the pairs of vertices that are not joined by an edge and there is no such pair of vertices 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 $M = \mathcal{F}^G(C) \cup \{\top\}$ is a fam-group and C is not a clique. If C is not a clique then there exist $v, v' \in C$ such that v and v' are not connected by an edge in G. So it follows that there exists an operator $o \in \mathcal{O}^G$ such that $\mathrm{add}(o) = \{f_v, f_{v'}\}$ 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. $\square$

Lemma 5.13. *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 $s_I^G = \{\top\}$, then surely $\left|M \cap s_I^G\right| \leq 1$. 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 $|N \cap \mathrm{add}(o)| = |N \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 0$, therefore, $|M \cap \mathrm{add}(o)| = 0 \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| = 1$, therefore, M is a fam-group. Finally, since every fam-group can be extended by $\top$, then surely every maximal fam-group must contain $\top$. $\square$

Theorem 5.14. 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 vertices in G. From Lemma 5.13, it follows that a maximum fam-group must contain $\top$ and then it follows from Lemma 5.12 that C is a maximum clique in G iff $M = \mathcal{F}^(C) \cup \{\top\}$ 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 5.1. 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. $\square$

The following lemma states that the maximum number of maximal fam-groups is exponential in the number of facts. The proof 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.

Lemma 5.15. *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 5.12 and Lemma 5.13 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 vertices (namely $c \cdot 3^{n/3}$ where n is number of vertices 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. □

It follows from Proposition 4.6 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 5.16.

Proposition 5.16. *Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ 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:*

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

Proof. It follows from Proposition 4.6 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 Lemma 5.15 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). □

5.3 h^2-mutexes and Fact-Alternating Mutex Groups

On one hand, we have shown that the inference of mutex groups in general case is PSPACE-Complete, that every mutex group can be decomposed into a set of mutex pairs, and that the inference of mutex pairs is also PSPACE-Complete. On the other hand, we have shown that the inference of fact-alternating mutex groups is only NP-Complete, and since every fam-group is a mutex group, it can also be decomposed into a set of mutex pairs. So, in this section, we answer the question how hard it is to infer mutex pairs that form fam-groups.

The family of h^m heuristics (Haslum & Geffner, 2000) is the most commonly used method for the inference of mutexes. The complexity of the computation of h^m heuristics grows exponentially with m, but it is polynomial for a fixed m. The h^1 heuristic corresponds to the h^{max} heuristic and it can infer only unreachable facts, the h^2 heuristic can infer also mutex pairs, the h^3 also mutexes of size three and so on. In this section, we show 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 a mutex pair inferred by h^2 (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 4.1), 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 5.4, Corollary 5.5 and Corollary 5.6. The importance of fam-groups, in particular its ability to detect operators that can produce only dead-end states, is demonstrated in Chapter 6.

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 5.17. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ 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 $\psi_{init} = \{\phi_c \mid c \subseteq s_I, |c| \leq 2\}$, 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 3.1), 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 $\mathrm{add}(o)$ and $\mathrm{del}(o)$, create a new operator $\omega_{o,g}$ with: $\mathrm{pre}(\omega_{o,g}) = \{\phi_c \mid c \subseteq (\mathrm{pre}(o) \cup g), |c| \leq 2\}$, $\mathrm{add}(\omega_{o,g}) = \{\phi_c \mid c \subseteq (\mathrm{add}(o) \cup g), c \cap \mathrm{add}(o) \neq \emptyset, |c| \leq 2\}$, $\mathrm{del}(\omega_{o,g}) = \emptyset$. 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 5.20 which is preceded by two auxiliary lemmas.

Lemma 5.18. *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. $\square$

Lemma 5.19. *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 $|\Sigma_M \cap \mathrm{add}(\omega)| \leq |\Sigma_M \cap \mathrm{pre}(\omega)|$.*

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

Now we prove $|\Sigma_M \cap \mathrm{add}(\omega)| \leq |\Sigma_M \cap \mathrm{pre}(\omega)|$ 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 $\mathrm{pre}^2(\omega) = \{\phi_c \mid \phi_c \in \mathrm{pre}(\omega), |c| = 2\}$, and $\mathrm{add}^2(\omega) = \{\phi_c \mid \phi_c \in \mathrm{add}(\omega), |c| = 2\}$. Since Σ_M contains only facts ϕ_c where $|c| = 2$, the inequality $|\Sigma_M \cap \mathrm{add}(\omega)| \leq |\Sigma_M \cap \mathrm{pre}(\omega)|$ holds iff $\left|\Sigma_M \cap \mathrm{add}^2(\omega)\right| \leq |\Sigma_M \cap \mathrm{pre}^2(\omega)|$ holds.

From Definition 5.17 it follows that for every operator $o \in \mathcal{O}$: $\mathrm{pre}^2(\omega_{o,\emptyset}) = \Sigma_{\mathrm{pre}(o)}$ and $\mathrm{add}^2(\omega_{o,\emptyset}) = \Sigma_{\mathrm{add}(o)}$, therefore we need to prove that $\left|\Sigma_M \cap \Sigma_{\mathrm{add}(o)}\right| \leq \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)}\right|$, which can be rewritten as $\left|\Sigma_{M \cap \mathrm{add}(o)}\right| \leq \left|\Sigma_{M \cap \mathrm{pre}(o)}\right|$ (Lemma 5.18), 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 $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,f})\right| \leq |\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})|$, because $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,\emptyset})\right| \leq |\Sigma_M \cap \mathrm{pre}^2(\omega_{o,\emptyset})|$.

(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 $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,f})\right| \leq |\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})|$.

(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 $|\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})| = \left|\Sigma_M \cap (\Sigma_{\mathrm{pre}(o)\setminus\{f\}} \cup \Gamma_{\mathrm{pre}(o),f})\right| = \left|\Sigma_M \cap \Sigma_{\mathrm{pre}(o)\setminus\{f\}}\right| + \left|\Sigma_M \cap \Gamma_{\mathrm{pre}(o),f}\right|$ ($\Sigma_{\mathrm{pre}(o)\setminus\{f\}}$ 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 5.17). 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 $\left|\Sigma_M \cap \mathrm{add}^2(\omega_{o,f})\right| \leq |\Sigma_M \cap \mathrm{pre}^2(\omega_{o,f})|$. $\square$

Theorem 5.20. *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 5.19 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 $\text{pre}(\omega) \subseteq \psi$ and, therefore, $|\Sigma_M \cap \text{pre}(\omega)| = 0$. So it follows from Lemma 5.19 that also $|\Sigma_M \cap \text{add}(\omega)| = 0$ because $|\Sigma_M \cap \text{add}(\omega)| \leq |\Sigma_M \cap \text{pre}(\omega)|$. Finally, since $\omega[\psi] = (\psi \setminus \text{del}(\omega)) \cup \text{add}(\omega)$ 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 5.20 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.

5.4 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 5.1) 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:

$$\sum_{f_i \in s_I} x_i \leq 1, \tag{5.1}$$

$$\forall o \in \mathcal{O} : \sum_{f_i \in \text{add}(o)} x_i \leq \sum_{f_i \in \text{del}(o) \cap \text{pre}(o)} x_i. \tag{5.2}$$

Equation (5.1) 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 5.1 ($|M \cap s_I| \leq 1$). Equation (5.2) corresponds to the second part of the definition ($|M \cap \text{add}(o)| \leq |M \cap \text{pre}(o) \cap \text{del}(o)|$) and it ensures that the mutex property is preserved by all operators.

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

$$\sum_{f_i \notin M} x_i \geq 1. \tag{5.3}$$

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.

Algorithm 5.1: Inference of fact-alternating mutex groups using ILP.

```
  Input: Planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$
  Output: A set of fam-groups $\mathcal{M}$
1 Initialize ILP with constraints according to Equations (5.1) and (5.2);
2 Set objective function of ILP to maximize $\sum_{f_i \in \mathcal{F}} x_i$;
3 Solve ILP and save the resulting fam-group into $M$;
4 while $|M| \geq 1$ do
5     Add $M$ to the output set $\mathcal{M}$;
6     Add constraint according to Equation (5.3) using $M$;
7     $M \leftarrow \emptyset$;
8     Solve ILP and if a solution was found, save the resulting fam-group into $M$;
```

The whole fam-group inferring algorithm is encapsulated in Algorithm 5.1. First, the ILP constraints are constructed according to Equations (5.1) and (5.2), which ensures that the 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 (5.3). 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 5.21. *Algorithm 5.1 is complete with respect to the maximal fact-alternating mutex groups.*

Proof. To prove Theorem 5.21 by contradiction, let us assume that Algorithm 5.1 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 5.1. Such a fam-group must satisfy the constraints expressed by Equations (5.1) and (5.2) and it must contain, at least, one fact that is not part of any fam-group produced by Algorithm 5.1. This is not violated by the ILP constraints in the last cycle of Algorithm 5.1 because they consist of Equations (5.1) and (5.2) and a set of Equation (5.3) 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 5.1 could not terminate, thus, such an M does not exist and Algorithm 5.1 is complete with respect to maximal fact-alternating mutex groups. $\square$

Note that Equation (5.3) can be used for the exclusion of any set of facts. This means that Algorithm 5.1 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 5.1. 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 after some time limit, because the algorithm produces one correct fam-group per cycle.

5.5 Experimental Evaluation

All algorithms experimentally evaluated in this section were implemented[2] in C. The experiments ran on a cluster of computing nodes with Intel Xeon Scalable Gold 6146 processors. The memory limit was set to 8 GB RAM, and the time limit was set to one hour for all compared variants and all related computations (such as mutex group cover number described in Section 5.5.2) together. The algorithms were evaluated on all domains from the optimal and satisficing deterministic tracks of the International Planning Competition (IPC) from 1998 to 2018 that do not contain any conditional effects after grounding.

The algorithm for inference of fam-groups (Algorithm 5.1) 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 `fam`. It is compared to two different methods for inferring mutex groups. One is the inference algorithm implemented in the Fast Downward's preprocessor (Helmert, 2009), abbreviated by `fd`. 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 5.3.

The presented algorithms are experimentally evaluated in several different ways. The algorithms are compared in terms of mutex pairs (Section 5.5.1), because the decomposition of the inferred mutex groups into a set of mutex pairs allows us to compare the algorithms without considering the differences in the shapes and sizes of the mutex groups. In Section 5.5.2, the algorithms are compared with respect to the mutex groups as they were inferred. In Section 5.5.3, the running times of the inference algorithms are compared and a faster version of `fam` is introduced. Lastly, utilization of mutex groups in a translation to finite domain representation is evaluated in Section 5.5.4.

5.5.1 Comparison 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 pairs. For example, $\mathtt{h}^2$ is able to produce mutex pairs only, but `fam` 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 `fam` 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 using some algorithm for enumerating maximal cliques, it must be taken into account that the reconstruction

[2]https://gitlab.com/danfis/cpddl

domain	**#mutex pairs**			**ratio to h^2**	
	fd	fam	h^2	fd	fam
agricola (34)	10 170	49 795	**141 244**	0.07	**0.35**
airport (3)	506	624	**3 445**	0.15	**0.18**
barman (74)	14 979	137 229	**152 059**	0.10	**0.90**
blocks (35)	59 766	59 766	**109 327**	**0.55**	**0.55**
caldera (40)	17 402	**37 956**	**37 956**	0.46	**1.00**
cavediving (40)	12 608	136 602	**183 020**	0.07	**0.75**
childsnack (40)	**9 347**	**9 347**	**9 347**	**1.00**	**1.00**
cybersec (30)	9 876	841 402	**2 396 394**	0.00	**0.35**
data-network (40)	**15 379**	**15 379**	**15 379**	**1.00**	**1.00**
depot (22)	208 628	209 896	**328 039**	0.64	**0.64**
driverlog (20)	**90 681**	**90 681**	**90 681**	**1.00**	**1.00**
elevators (100)	**714 069**	**714 069**	**714 069**	**1.00**	**1.00**
floortile (80)	**112 847**	**112 847**	**112 847**	**1.00**	**1.00**
freecell (80)	72 702	166 902	**199 996**	0.36	**0.83**
ged (33)	442 256	442 256	**450 165**	**0.98**	**0.98**
gripper (20)	**16 480**	**16 480**	**16 480**	**1.00**	**1.00**
hiking (40)	**7 371**	**7 371**	**7 371**	**1.00**	**1.00**
logistics (62)	**7 985 312**	**7 985 312**	**7 985 312**	**1.00**	**1.00**
maintenance (25)	0	**6**	**6**	0.00	**1.00**
movie (30)	**0**	**0**	**0**	–	–
mprime (35)	160 606	161 234	**258 806**	0.62	**0.62**
mystery (30)	126 381	134 529	**1 563 029**	0.08	**0.09**
nomystery (40)	590 997	592 797	**716 283**	0.83	**0.83**
openstacks (167)	815 020	815 020	**1 210 002**	**0.67**	**0.67**
organic-synthesis (10)	0	1 087	**6 163**	0.00	**0.18**
parcprinter (70)	63 249	130 761	**239 933**	0.26	**0.54**
parking (80)	2 806 194	2 806 194	**4 177 265**	**0.67**	**0.67**
pathways (30)	7 948	7 948	**38 396**	**0.21**	**0.21**
pegsol (70)	40 812	42 985	**53 564**	0.76	**0.80**
pipesworld-notankage (38)	41 000	52 608	**370 776**	0.11	**0.14**
pipesworld-tankage (38)	122 005	195 190	**404 819**	0.30	**0.48**
psr-small (50)	3 016	3 016	**5 836**	**0.52**	**0.52**
rovers (40)	496 910	496 915	**498 302**	1.00	**1.00**
satellite (36)	1 844 272	**1 845 356**	**1 845 356**	1.00	**1.00**
scanalyzer (70)	120 324	127 980	**128 172**	0.94	**1.00**
snake (31)	51 023	85 197	**95 024**	0.54	**0.90**
sokoban (100)	557 841	557 917	**640 479**	0.87	**0.87**
spider (36)	384 459	1 298 267	**4 679 270**	0.08	**0.28**
storage (30)	419 589	419 589	**497 428**	**0.84**	**0.84**
termes (40)	**15 034**	**15 034**	**15 034**	**1.00**	**1.00**
tetris (37)	116 296	25 591 456	**96 318 063**	0.00	**0.27**
thoughtful (20)	82 331	547 087	**819 046**	0.10	**0.67**
tidybot (35)	694	192 567	**420 195**	0.00	**0.46**
tpp (30)	45 234	47 083	**3 107 772**	0.01	**0.02**
transport (140)	8 827 475	**10 254 463**	**10 254 463**	0.86	**1.00**
trucks (16)	6 184	288 803	**346 455**	0.02	**0.83**
visitall (80)	**110 261 682**	**110 261 682**	**110 261 682**	**1.00**	**1.00**
woodworking (100)	42 063	44 376	**94 767**	0.44	**0.47**
zenotravel (20)	**39 147**	**39 147**	**39 147**	**1.00**	**1.00**
overall (2367)	137 888 165	168 090 208	**252 058 664**	0.55	**0.67**

Table 5.2: Left: Sums of numbers of inferred mutex pairs per domain and overall. Right: Ratios to h^2 per domain and overall. Maximums are highlighted.

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 fam were decomposed into mutex pairs (h^2 generates mutex pairs, so decomposition was not necessary), the results are shown in Table 5.2. Since we already know that h^2 dominates fam in terms of mutex pairs (Theorem 5.20), Table 5.2 shows also ratios to h^2. fam inferred only about two thirds of all h^2-mutexes overall, but in almost one third of the domains (16 out of 49) fam found all h^2-mutexes, in 20 domains fam found at least 90% of all h^2-mutexes and in 26 domains, i.e., in more than a half of all domains, fam found 80% of all h^2-mutexes or more. Moreover, there are few outliers, like the domains tpp and mystery, where fam found only a very small fraction of h^2-mutexes, which skews the overall picture.

Figure 5.1 shows the comparison as scatter plots where each point corresponds to an individual problem from the dataset. The middle plot comparing fam and h^2 shows that

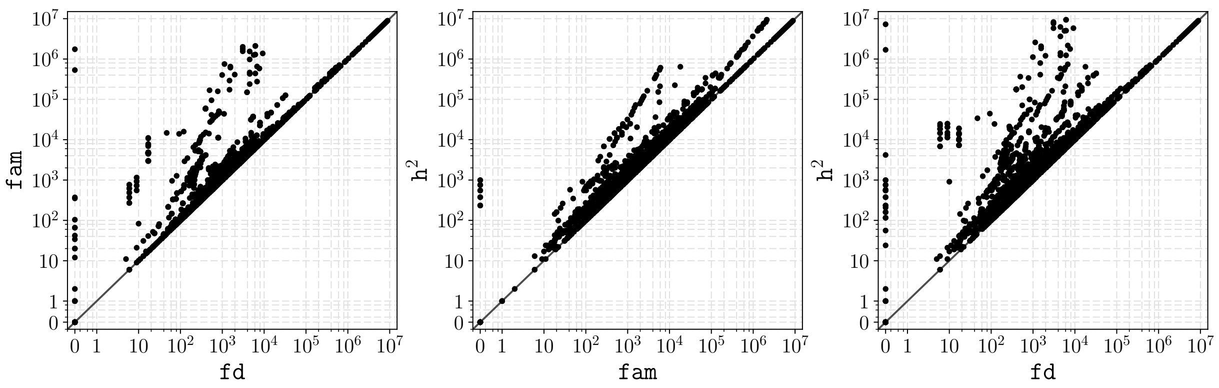

Figure 5.1: Comparison of the number of inferred mutex pairs in each problem as scatter plots with the logarithmic scale and added zero.

these two methods are very close to each other. The left scatter plot in Figure 5.1 shows that `fam` produces at least as many mutex pairs as `fd` in every single planning task (and we will explain in more detail why in Chapter 7). The relative difference between `fam` and `fd` is much higher than between `fam` 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.

5.5.2 Comparison of Mutex Groups

In Section 5.5.1, we provided an analysis of the algorithms for inference of mutex groups in terms of mutex pairs that were obtained by decomposition of the 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 for which larger mutex groups are desirable. Other possible applications were discussed in Section 4.1 and Section 5.1 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.

In this section, we compare the inferred mutex groups produced by the algorithms `fd` and `fam`. 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 &

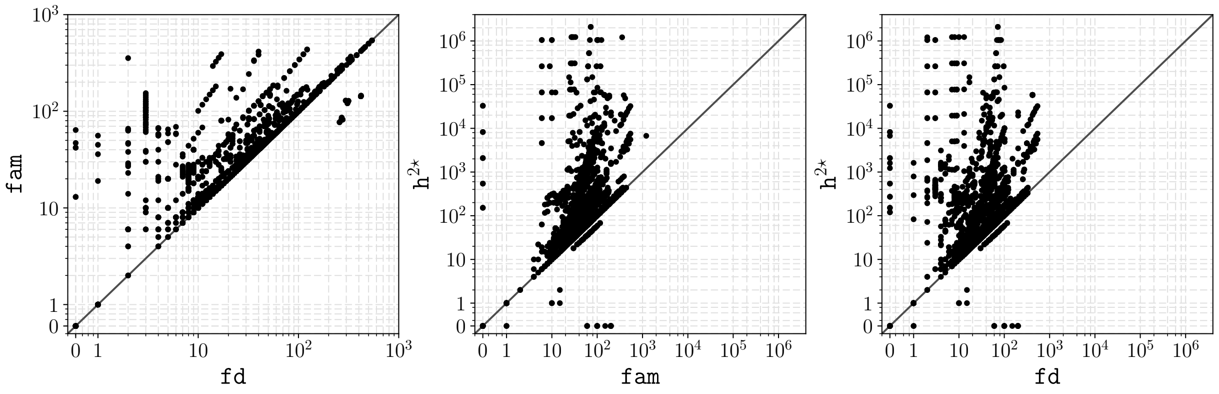

Figure 5.2: Comparison of the number of inferred mutex groups in each problem as scatter plots with the logarithmic scale and added zero.

domain	**#mutex groups**			**cover number**			**avg. rel. cover**		
	`fd`	`fam`	$\mathtt{h}^{2\star}$	`fd`	`fam`	$\mathtt{h}^{2\star}$	`fd`	`fam`	$\mathtt{h}^{2\star}$
agricola (34)	311	886	**6 965 285**	4 325	2 178	**2 144**	0.71	0.36	**0.35**
airport (3)	53	87	**263 011**	206	204	**110**	0.76	0.75	**0.41**
barman (74)	1 153	2 793	**16 160**	16 914	3 140	**3 053**	0.88	0.19	**0.18**
blocks (35)	709	709	**2 413**	**709**	**709**	**709**	**0.18**	**0.18**	**0.18**
caldera (40)	2 380	**7 312**	**7 312**	14 755	**11 797**	**11 797**	0.83	**0.71**	**0.71**
cavediving (40)	368	1 640	**13 438**	7 492	1 866	**1 826**	0.73	0.22	**0.22**
childsnack (40)	**2 596**	**2 596**	1 536	**3 126**	**3 126**	**3 126**	**0.49**	**0.49**	**0.49**
cybersec (30)	10 266	3 636	**740 692**	19 864	8 252	**6 750**	0.94	0.39	**0.32**
data-network (40)	**4 379**	**4 379**	**4 379**	**8 256**	**8 256**	**8 256**	**0.53**	**0.53**	**0.53**
depot (22)	1 029	1 029	**145 865**	**807**	**807**	**807**	**0.12**	**0.12**	**0.12**
driverlog (20)	**375**	**375**	**375**	**375**	**375**	**375**	**0.13**	**0.13**	**0.13**
elevators (100)	**2 401**	**2 401**	**2 401**	**2 401**	**2 401**	**2 401**	**0.09**	**0.09**	**0.09**
floortile (80)	**2 613**	**2 613**	**2 613**	**2 432**	**2 432**	**2 432**	**0.20**	**0.20**	**0.20**
freecell (80)	5 432	5 432	**24 026**	**5 432**	**5 432**	**5 432**	**0.30**	**0.30**	**0.30**
ged (33)	1 398	1 398	**8 274 638**	**866**	**866**	**866**	**0.09**	**0.09**	**0.09**
gripper (20)	**520**	**520**	**520**	**520**	**520**	**520**	**0.28**	**0.28**	**0.28**
hiking (40)	**522**	**522**	**522**	**522**	**522**	**522**	**0.20**	**0.20**	**0.20**
logistics (62)	**2 223**	**2 223**	**2 223**	**2 223**	**2 223**	**2 223**	**0.09**	**0.09**	**0.09**
maintenance (25)	**3 090**	**3 090**	6	14 200	**14 194**	**14 194**	1.00	**0.99**	**0.99**
movie (30)	**0**	**0**	**0**	**210**	**210**	**210**	**1.00**	**1.00**	**1.00**
mprime (35)	1 373	1 376	**1 439**	**1 373**	**1 373**	**1 373**	**0.10**	**0.10**	**0.10**
mystery (30)	1 042	1 102	**74 478**	**1 042**	**1 042**	**1 042**	**0.17**	**0.17**	**0.17**
nomystery (40)	400	440	**8 580**	780	**440**	**440**	0.08	**0.05**	**0.05**
openstacks (167)	20 852	20 852	**397 841**	21 847	21 847	**20 852**	0.36	0.36	**0.33**
organic-synthesis (10)	12	1 796	**1 220 440**	408	236	**134**	1.00	0.61	**0.39**
parcprinter (70)	2 345	4 492	**16 343**	12 615	**3 971**	**3 971**	0.61	**0.24**	**0.24**
parking (80)	4 795	4 795	**224 779**	**4 795**	**4 795**	**4 795**	**0.07**	**0.07**	**0.07**
pathways (30)	1 440	1 440	**31 888**	**9 268**	**9 268**	**9 268**	**0.81**	**0.81**	**0.81**
pegsol (70)	2 354	2 577	**120 699**	**2 354**	**2 354**	**2 354**	**0.34**	**0.34**	**0.34**
pipesworld-notankage (38)	384	480	**5 685 948**	8 498	7 130	**702**	0.75	0.67	**0.09**
pipesworld-tankage (38)	2 112	2 913	**5 674 152**	3 083	2 405	**1 952**	0.27	0.23	**0.18**
psr-small (50)	835	835	**1 885**	1 252	1 252	**1 103**	0.52	0.52	**0.46**
rovers (40)	1 936	1 938	**3 325**	**8 706**	**8 706**	**8 706**	**0.43**	**0.43**	**0.43**
satellite (36)	256	**508**	**508**	12 516	**11 950**	**11 950**	0.49	**0.45**	**0.45**
scanalyzer (70)	1 408	1 580	**2 572**	**1 580**	**1 580**	**1 580**	**0.18**	**0.18**	**0.18**
snake (31)	93	3 114	**14 285**	7 691	3 641	**3 083**	0.73	0.35	**0.29**
sokoban (100)	4 991	4 999	**8 094**	**4 891**	**4 891**	**4 891**	**0.20**	**0.20**	**0.20**
spider (36)	1 320	5 495	**568 034**	17 534	7 496	**4 850**	0.43	0.20	**0.13**
storage (30)	1 140	1 140	**234 778**	2 560	2 560	**1 140**	0.24	0.24	**0.17**
termes (40)	**690**	**690**	**690**	**690**	**690**	**690**	**0.17**	**0.17**	**0.17**
tetris (37)	118	1 834	**36 696**	68 302	**1 544**	**1 544**	0.91	**0.03**	**0.03**
thoughtful (20)	854	1 416	**68 781**	5 797	4 502	**3 727**	0.55	0.42	**0.34**
tidybot (35)	292	684	**9 868**	17 721	11 321	**5 940**	0.98	0.63	**0.33**
tpp (30)	3 683	4 064	**5 018**	**3 683**	**3 683**	**3 683**	**0.28**	**0.28**	**0.28**
transport (140)	2 102	**2 534**	**2 534**	27 273	**2 534**	**2 534**	0.25	**0.05**	**0.05**
trucks (16)	188	2 502	**5 182**	7 564	**330**	**330**	0.87	**0.05**	**0.05**
visitall (80)	**80**	**80**	**80**	**82 538**	**82 538**	**82 538**	**0.50**	**0.50**	**0.50**
woodworking (100)	4 590	6 544	**14 980**	16 922	16 886	**10 001**	0.58	0.57	**0.37**
zenotravel (20)	**335**	**335**	**335**	**335**	**335**	**335**	**0.13**	**0.13**	**0.13**
overall (2367)	103 838	126 196	**30 901 647**	459 253	290 810	**263 261**	0.41	0.30	**0.27**

Table 5.3: Left: Sums of the number of inferred mutex groups, maximums highlighted. Middle: Sums of mutex group cover numbers, minimums highlighted. Right: Average of $C/|\mathcal{F}|$ within each domain, where C is a mutex group cover number and $|\mathcal{F}|$ is a number of facts; the overall row is average over all problems from all domains; minimums highlighted.

Kerbosch, 1973; Tomita et al., 2006; Cazals & Karande, 2008) for construction of maximal mutex groups from the mutex pairs generated by $\mathtt{h}^2$ (as described in Proposition 4.9). This modified algorithm is denoted by $\mathtt{h}^{2\star}$.

The sums of a number of inferred mutex groups by all three algorithms are listed in Table 5.3 (left side), the maximal values are highlighted. Figure 5.2 shows per-problem comparison as scatter plots. Every mutex group that was generated by `fd` was also generated by `fam` or it was a subset of some mutex group generated by `fam`. Note that there are cases in which `fd` reports more mutex groups than `fam` (or `fam` more than $\mathtt{h}^{2\star}$), but in these cases the mutex groups from `fd` are subsets of some mutex group found by `fam` (more on that below).

Since a set of h^2-mutexes is always a superset of decompositions of all fam-groups (Theorem 5.20), $\mathtt{h}^{2\star}$ must always generate a richer set of mutex groups than `fam` (and, thus,

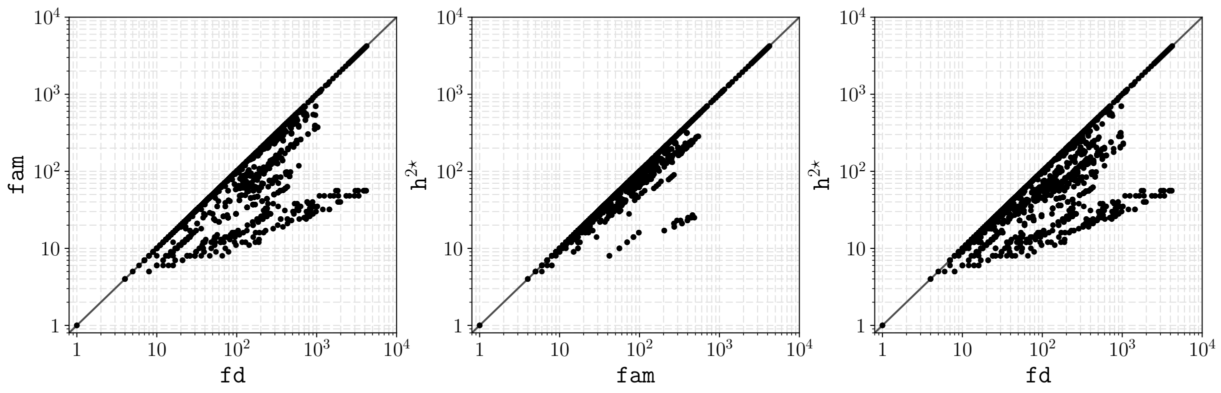

Figure 5.3: Comparison of the mutex group cover numbers for each problem as scatter plots with the logarithmic scale.

also than `fd`). Many of the mutex groups inferred by $\mathtt{h}^{2\star}$ are supersets of `fam` 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 $\mathtt{h}^{2\star}$ (30 901 647) and `fam` (126 196) 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 5.16). This aspect of the comparison is clearly visible if we compare the middle scatter plot in Figure 5.2, showing the number of mutex groups, with the middle scatter plot in Figure 5.1, 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 `fam` into mutex pairs and then use an algorithm for enumerating maximal cliques for the construction of new mutex groups. This approach may generate more mutex groups than is the original number of fam-groups, but the resulting mutex groups are not necessarily fam-groups.

Comparison based on the number of inferred mutex groups, however, does not give us the full picture, because, as mentioned before, the numbers do not reflect on the fact that one method can find multiple mutex groups that are all subset of some mutex group found by other method. 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 vertices 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 $\{f_1, f_2\}$, `fam` 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 `fam` 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 5.3 (middle). We also show cover numbers as a ratio to the number of facts, i.e., the smaller number means that less mutex groups are needed to cover all facts relatively to the size each problem. Table 5.3 (right) shows averages within each domain and the average over all problems. The table shows that fam-groups are actually very close to the mutex groups constructed from h^2-mutexes (0.3 versus 0.27 overall), but there are also domains h^2-mutexes provide much richer information (e.g., airport, pipesworld-notankage, or thoughtful). The difference between `fd` and `fam` is much more profound in both absolute and relative numbers.

The left scatter plot in Figure 5.3 clearly shows that the cover number of `fam` 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 about 1.58 times higher than that of `fam` even though `fam` inferred only about 20% more mutex groups. This result suggests that if `fam` is used as a replacement for `fd` in a translation from PDDL to FDR, the overall memory footprint of the planner will be reduced.

5.5.3 Comparison of Running Times

Table 5.4 shows the sum and average of running times in seconds of implemented algorithms in each domain and overall (0.00 means that the running time is below 10 milliseconds). The results show that `fd` is almost 2 times faster than $\mathtt{h}^2$ and more than 120 times faster than `fam`, and the difference is even more profound without considering the cybersec domain. The running time of `fd` almost never exceeds one second (note that we re-implemented the inference algorithm in C instead of using the original implementation in Python). `fam` is more than an order of magnitude slower than $\mathtt{h}^2$ which is an expected result considering that $\mathtt{h}^2$ is a polynomial algorithm, but `fam` requires to solve the ILP repeatedly.

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 inference of h^2-mutexes in all domains needed about 60% more time than the inference of the maximal mutex groups from these h^2-mutexes ($\mathtt{h}^{2\star}$) even though the inference of h^2-mutexes is polynomial in time, but the inference of the maximal mutex groups is NP-Hard.

In comparison to $\mathtt{h}^{2\star}$, `fam` requires almost fifty times more time. Considering that both $\mathtt{h}^{2\star}$ and `fam` are NP-Hard (both are implemented using exponential algorithms) and that $\mathtt{h}^{2\star}$ always produces supersets of `fam`, we suppose that `fam` 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 5.4, `fam` 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 `fam` (in all cases), we have implemented a combination of `fam` and `fd` that we denote by $\mathtt{fam}_{\mathtt{fd}}$ (this is also justified by the theory laid out in Chapter 7). The algorithm $\mathtt{fam}_{\mathtt{fd}}$ first infers mutex groups by `fd` and then runs `fam` initialized by these mutex groups (see Section 5.4 and specifically Equation (5.3)), i.e., `fam` spends its computational time only on the mutex groups that were not already inferred by `fd`. The resulting running time of $\mathtt{fam}_{\mathtt{fd}}$ over all domains is only about 43% of the running

domain	**sum**					**avg**				
	`fd`	h^2	$h^{2\star}$	`fam`	$\texttt{fam}_{\texttt{fd}}$	`fd`	h^2	$h^{2\star}$	`fam`	$\texttt{fam}_{\texttt{fd}}$
agricola (34)	**0.04**	11.57	38.84	141.29	87.59	**0.00**	0.34	1.14	4.16	2.58
airport (3)	0.00	**0.00**	0.24	0.68	0.37	0.00	**0.00**	0.08	0.23	0.12
barman (74)	**0.03**	0.70	0.77	110.98	87.21	**0.00**	0.01	0.01	1.50	1.18
blocks (35)	**0.04**	0.08	0.17	10.00	0.42	**0.00**	0.00	0.00	0.29	0.01
caldera (40)	**3.95**	100.39	100.46	7 549.06	5 237.19	**0.10**	2.51	2.51	188.73	130.93
cavediving (40)	**0.00**	1.43	1.56	240.90	233.10	**0.00**	0.04	0.04	6.02	5.83
childsnack (40)	**0.02**	0.56	0.58	350.95	3.03	**0.00**	0.01	0.01	8.77	0.08
cybersec (30)	695.57	**1.43**	20.28	701.01	1 731.11	23.19	**0.05**	0.68	23.37	57.70
data-network (40)	**0.02**	1.68	1.71	646.31	4.66	**0.00**	0.04	0.04	16.16	0.12
depot (22)	**0.01**	0.87	1.83	191.42	48.15	**0.00**	0.04	0.08	8.70	2.19
driverlog (20)	**0.01**	0.55	0.59	6.57	0.36	**0.00**	0.03	0.03	0.33	0.02
elevators (100)	**0.01**	15.19	15.50	981.13	66.65	**0.00**	0.15	0.15	9.81	0.67
floortile (80)	**0.02**	0.14	0.19	24.19	1.39	**0.00**	0.00	0.00	0.30	0.02
freecell (80)	**0.05**	4.13	4.32	1 120.16	1 252.24	**0.00**	0.05	0.05	14.00	15.65
ged (33)	**0.27**	1.18	12.69	996.55	36.26	**0.01**	0.04	0.38	30.20	1.10
gripper (20)	0.01	**0.01**	0.01	3.95	0.24	0.00	**0.00**	0.00	0.20	0.01
hiking (40)	**0.01**	0.32	0.33	9.85	0.69	**0.00**	0.01	0.01	0.25	0.02
logistics (62)	**0.03**	47.20	54.57	970.83	19.21	**0.00**	0.76	0.88	15.66	0.31
maintenance (25)	0.16	**0.15**	0.16	335.74	4.29	0.01	**0.01**	0.01	13.43	0.17
movie (30)	0.01	**0.00**	**0.00**	0.04	0.04	0.00	**0.00**	**0.00**	0.00	0.00
mprime (35)	**0.01**	6.75	6.90	470.81	6.04	**0.00**	0.19	0.20	13.45	0.17
mystery (30)	**0.00**	2.89	5.45	222.97	9.96	**0.00**	0.10	0.18	7.43	0.33
nomystery (40)	**0.04**	1.15	1.45	16.63	2.76	**0.00**	0.03	0.04	0.42	0.07
openstacks (167)	**0.55**	38.28	38.90	21 831.98	66.84	**0.00**	0.23	0.23	130.73	0.40
organic-synthesis (10)	**0.00**	0.05	1.30	632.08	622.30	**0.00**	0.01	0.13	63.21	62.23
parcprinter (70)	**0.05**	0.23	0.48	324.27	526.67	**0.00**	0.00	0.01	4.63	7.52
parking (80)	**0.03**	52.80	71.05	9 128.27	48.36	**0.00**	0.66	0.89	114.10	0.60
pathways (30)	**0.02**	0.34	0.36	23.69	0.64	**0.00**	0.01	0.01	0.79	0.02
pegsol (70)	**0.00**	0.02	0.20	21.32	4.79	**0.00**	0.00	0.00	0.30	0.07
pipesworld-notankage (38)	**0.00**	0.69	12.58	44.17	6.51	**0.00**	0.02	0.33	1.16	0.17
pipesworld-tankage (38)	**0.03**	3.32	14.71	2 791.67	332.76	**0.00**	0.09	0.39	73.47	8.76
psr-small (50)	1.62	**0.01**	0.01	1.91	0.23	0.03	**0.00**	0.00	0.04	0.00
rovers (40)	**0.24**	8.03	8.29	225.22	3.63	**0.01**	0.20	0.21	5.63	0.09
satellite (36)	**0.01**	110.49	111.79	545.06	289.91	**0.00**	3.07	3.11	15.14	8.05
scanalyzer (70)	**0.00**	4.20	4.30	2 326.46	1 259.31	**0.00**	0.06	0.06	33.24	17.99
snake (31)	**0.01**	6.54	6.60	11 301.19	11 084.62	**0.00**	0.21	0.21	364.55	357.57
sokoban (100)	**0.03**	1.22	1.59	242.88	5.55	**0.00**	0.01	0.02	2.43	0.06
spider (36)	**0.05**	70.40	87.89	11 173.51	8 197.01	**0.00**	1.96	2.44	310.38	227.69
storage (30)	**0.01**	3.31	5.89	213.85	2.89	**0.00**	0.11	0.20	7.13	0.10
termes (40)	**0.00**	0.18	0.19	2.51	0.17	**0.00**	0.00	0.00	0.06	0.00
tetris (37)	**0.01**	4.30	396.24	3 984.38	4 028.13	**0.00**	0.12	10.71	107.69	108.87
thoughtful (20)	**0.01**	5.32	13.79	963.22	1 093.57	**0.00**	0.27	0.69	48.16	54.68
tidybot (35)	**0.04**	10.45	10.64	207.85	162.59	**0.00**	0.30	0.30	5.94	4.65
tpp (30)	**0.02**	3.12	6.79	4 131.65	298.42	**0.00**	0.10	0.23	137.72	9.95
transport (140)	**0.03**	320.63	327.73	2 147.32	180.52	**0.00**	2.29	2.34	15.34	1.29
trucks (16)	**0.00**	0.51	0.86	2 573.49	2 232.62	**0.00**	0.03	0.05	160.84	139.54
visitall (80)	**0.03**	534.31	878.67	129.28	0.70	**0.00**	6.68	10.98	1.62	0.01
woodworking (100)	**0.09**	1.71	1.78	1 150.76	569.43	**0.00**	0.02	0.02	11.51	5.69
zenotravel (20)	**0.00**	0.89	0.90	14.76	0.66	**0.00**	0.04	0.05	0.74	0.03
overall (2367)	**703.18**	1 379.72	2 272.17	91 234.79	39 851.79	**0.30**	0.58	0.96	38.54	16.84
overall \ cybersec (2337)	**7.61**	1 378.29	2 251.89	90 533.78	38 120.69	**0.00**	0.59	0.96	38.74	16.31

Table 5.4: Left: Sums of running times in seconds in each domain and overall, minimums highlighted. Right: Averages of running times within each domain and over all problems, minimums highlighted.

time of `fam`. However, there are cases in which $\texttt{fam}_{\texttt{fd}}$ is slower than `fam` (e.g., cybersec, freecell, parcprinter, tetris, or thoughtful). The reason in most cases is that `fd` generates mutex groups that are not maximal. Therefore, the ILP corresponding to $\texttt{fam}_{\texttt{fd}}$ contains constraints that are actually useless, because it still needs to find maximal fam-groups that are superset of those expressed in the constraints.

5.5.4 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 the 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

domain	**#fdr variables**			**avg rel. to $\lvert\mathcal{F}\rvert$**		
	`fd`	`fam`	$h^{2\star}$	`fd`	`fam`	$h^{2\star}$
agricola (34)	**4 325**	2 178	2 169	0.71	0.36	**0.36**
airport (3)	**208**	**208**	202	0.77	0.77	**0.75**
barman (74)	**16 914**	3 140	3 519	0.88	**0.19**	0.22
blocks (35)	**709**	**709**	**709**	**0.18**	**0.18**	**0.18**
caldera (40)	**14 755**	11 797	11 797	0.83	**0.71**	**0.71**
cavediving (40)	**7 588**	1 962	2 030	0.74	**0.23**	0.24
childsnack (40)	**3 126**	**3 126**	**3 126**	**0.49**	**0.49**	**0.49**
cybersec (30)	**19 864**	8 252	7 408	0.94	0.39	**0.35**
data-network (40)	**8 256**	**8 256**	**8 256**	**0.53**	**0.53**	**0.53**
depot (22)	**1 029**	**1 029**	**1 029**	**0.15**	**0.15**	**0.15**
driverlog (20)	**375**	**375**	**375**	**0.13**	**0.13**	**0.13**
elevators (100)	**2 401**	**2 401**	**2 401**	**0.09**	**0.09**	**0.09**
floortile (80)	**2 613**	**2 613**	**2 613**	**0.22**	**0.22**	**0.22**
freecell (80)	**5 432**	**5 432**	**5 432**	**0.30**	**0.30**	**0.30**
ged (33)	**866**	**866**	**866**	**0.09**	**0.09**	**0.09**
gripper (20)	**520**	**520**	**520**	**0.28**	**0.28**	**0.28**
hiking (40)	**522**	**522**	**522**	**0.20**	**0.20**	**0.20**
logistics (62)	**2 223**	**2 223**	**2 223**	**0.09**	**0.09**	**0.09**
maintenance (25)	**14 200**	14 194	14 194	1.00	**0.99**	**0.99**
movie (30)	**210**	**210**	**210**	**1.00**	**1.00**	**1.00**
mprime (35)	1 373	1 375	**1 390**	**0.10**	0.10	0.10
mystery (30)	1 042	1 058	**1 093**	**0.17**	0.17	0.18
nomystery (40)	**780**	440	440	0.08	**0.05**	**0.05**
openstacks (167)	**21 847**	**21 847**	**21 847**	**0.36**	**0.36**	**0.36**
organic-synthesis (10)	**408**	274	210	1.00	0.68	**0.59**
parcprinter (70)	**12 615**	4 141	4 159	0.61	**0.25**	0.25
parking (80)	6 681	6 681	**7 503**	**0.10**	**0.10**	0.11
pathways (30)	**9 268**	**9 268**	**9 268**	**0.81**	**0.81**	**0.81**
pegsol (70)	2 354	2 371	**2 373**	**0.34**	0.34	0.34
pipesworld-notankage (38)	**8 498**	7 130	1 660	0.75	0.67	**0.27**
pipesworld-tankage (38)	**3 103**	2 460	2 458	0.27	0.24	**0.24**
psr-small (50)	**1 252**	**1 252**	**1 252**	**0.52**	**0.52**	**0.52**
rovers (40)	8 706	**8 708**	**8 708**	**0.43**	0.43	0.43
satellite (36)	**12 516**	11 950	11 950	0.49	**0.45**	**0.45**
scanalyzer (70)	**1 580**	**1 580**	**1 580**	**0.18**	**0.18**	**0.18**
snake (31)	**7 691**	3 672	3 673	0.73	**0.35**	0.35
sokoban (100)	**5 480**	5 476	5 477	0.22	**0.22**	0.22
spider (36)	**17 534**	7 949	6 897	0.43	0.21	**0.19**
storage (30)	**2 800**	**2 800**	**2 800**	**0.27**	**0.27**	**0.27**
termes (40)	**690**	**690**	**690**	**0.17**	**0.17**	**0.17**
tetris (37)	**68 302**	1 834	1 834	0.91	**0.04**	**0.04**
thoughtful (20)	**5 797**	4 505	3 871	0.55	0.42	**0.36**
tidybot (35)	**17 721**	11 322	11 057	0.98	0.63	**0.61**
tpp (30)	3 683	**3 883**	3 714	**0.28**	0.31	0.30
transport (140)	**27 273**	2 534	2 534	0.25	**0.05**	**0.05**
trucks (16)	**7 564**	330	330	0.87	**0.05**	**0.05**
visitall (80)	**82 538**	**82 538**	**82 538**	**0.50**	**0.50**	**0.50**
woodworking (100)	**16 922**	16 886	11 419	0.58	0.57	**0.42**
zenotravel (20)	**335**	**335**	**335**	**0.13**	**0.13**	**0.13**
overall (2367)	**462 489**	295 302	282 661	0.41	0.30	**0.29**

Table 5.5: Left: Sums of the number FDR variables, maximums highlighted. Right: Average of $V/\lvert\mathcal{F}\rvert$ within each domain, where V is a number of FDR variables and $\lvert\mathcal{F}\rvert$ is a number of facts; the overall row is average over all tasks from all domains; minimums highlighted.

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.

Table 5.5 shows the number of created variables per domain and overall for the al-

domain	`lmc`			`pot-all`			`ms`		
	`fd`	`h`^{2*}	`fam`$_{\texttt{fd}}$	`fd`	`h`^{2*}	`fam`$_{\texttt{fd}}$	`fd`	`h`^{2*}	`fam`$_{\texttt{fd}}$
agricola18 (20)	0	0	0	1	1	1	3	6	**10**
airport04 (50)	**29**	3	24	**23**	3	21	**16**	3	15
blocks00 (35)	28	28	28	28	**29**	28	21	21	21
caldera18 (20)	8	**9**	**9**	8	8	8	9	**10**	**10**
depot02 (22)	7	7	7	7	**10**	7	7	**11**	7
freecell00 (80)	15	15	15	40	**65**	64	20	**61**	**61**
mprime98 (35)	24	**25**	24	24	24	24	24	24	24
mystery98 (30)	17	17	17	17	17	17	**17**	16	**17**
nomystery11 (20)	15	**17**	15	14	14	14	20	20	20
organic-synthesis18 (20)	**10**	8	7	**10**	8	7	**9**	8	7
parcprinter08 (30)	19	19	19	**22**	20	20	**27**	26	**27**
parcprinter11 (20)	14	14	14	16	16	16	**20**	19	**20**
parking11 (20)	4	4	4	**8**	6	**8**	**8**	1	**8**
parking14 (20)	4	4	4	**8**	5	**8**	**8**	0	**8**
petri-net-alignment18 (20)	**9**	8	0	**7**	4	0	**7**	0	0
pipesworld-notankage04 (50)	18	18	18	20	20	20	21	**24**	22
pipesworld-tankage04 (50)	13	13	13	17	17	**18**	15	**16**	**16**
psr-small04 (50)	48	48	48	50	50	50	**50**	49	**50**
rovers06 (40)	7	7	7	5	5	5	6	**7**	6
scanalyzer08 (30)	16	**17**	**17**	13	13	13	13	13	13
scanalyzer11 (20)	13	**14**	**14**	10	10	10	10	10	10
snake18 (20)	7	7	7	**14**	12	13	7	10	**15**
sokoban08 (30)	30	30	30	28	28	28	**30**	29	**30**
spider18 (20)	11	11	11	12	**14**	13	6	10	**12**
tetris14 (17)	6	**9**	**9**	13	**16**	**16**	2	**12**	11
tidybot11 (20)	**14**	6	**14**	**14**	6	**14**	8	1	**13**
tidybot14 (20)	9	8	**10**	**10**	8	**10**	0	0	**3**
tpp06 (30)	7	7	7	6	6	6	**9**	**9**	8
transport11 (20)	**7**	6	6	6	6	6	6	6	6
trucks06 (30)	8	8	8	7	**8**	**8**	5	**7**	**7**
woodworking08 (30)	19	**20**	19	13	**14**	13	**17**	15	**17**
woodworking11 (20)	13	**14**	13	8	**9**	8	**11**	10	**11**
Σ (1697)	**840**	812	829	868	861	**883**	826	848	**899**
Σ \ petri-net-align18 (1677)	**831**	804	829	861	857	**883**	819	848	**899**

Table 5.6: Comparison of different algorithms used for the inference of FDR variables. Number of solved tasks in each domain and overall, only the domains with a difference are shown, maximums are highlighted.

gorithms `fd`, `h`^{2*}, and `fam`. The right side of Table 5.5 shows averages of a number of variables over the number of facts, i.e., it is directly comparable to the right side of Table 5.3 where we show averages of mutex group cover numbers over the number of facts. The numbers are very similar to the mutex group cover numbers listed in Table 5.3, which means that the greedy algorithm used in Fast Downward can actually generate a number of variables very close to the possible minimum.

`fd` is clearly dominated by both `h`^{2*} and `fam`, which was expected considering the experimental results from the previous sections. The results for `h`^{2*} and `fam` are very similar which also corresponds to the results in the previous sections. Note that there are domains (e.g., barman, cavediving, or parking) in which `fam` creates less variables than `h`^{2*} even though the mutex group cover number for `h`^{2*} must always be lower than for `fam`. This is clearly an effect of tie breaking in the greedy algorithm and the unnecessarily large number of mutex groups inferred by `h`^{2*} 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 sharing many common facts.

We have also compared the algorithms in terms of the number of solved planning tasks in the optimal track. We used the Fast Downward planner (Helmert, 2006) with A* and the following admissible heuristics: the LM-Cut (`lmc`) heuristic (Helmert & Domshlak, 2009), the potential (`pot-all`) heuristic optimized for all syntactic states (Seipp et al., 2015), and the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy and non-greedy bi-simulation shrink strategy (Helmert et al., 2014; Sievers et al., 2016). The maximal allowed time for the whole planning process (including preprocessor and search) was set to 30 minutes and the maximal memory limit was set to 8 GB. For this experiment, we used $\texttt{fam}_{\texttt{fd}}$ instead of `fam`, 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 5.6. First of, note that $\texttt{fam}_{\texttt{fd}}$ did not solve any task in the petri-net-alignment domain, because the inference of fam-groups in this domain consumed the whole time limit of 30 minutes. For `lmc`, the number of solved tasks is almost identical for all three variants except for the aforementioned petri-net-alignment domain and also airport, freecell, and tidybot domains. In airport and tidybot domains, $\texttt{h}^2$ solved considerably less tasks than the other two variants, because no pruning was used and the resulting task had a huge number of overlapping mutex groups of which inference consumed all allotted memory. The freecell domain contains more tasks than any other domain, so the change of coverage in this domain affects the overall results more than any other domain. The results for `pot-all` are also very similar for all variants. The main reason why $\texttt{fam}_{\texttt{fd}}$ solved more tasks than `fd` is due to the freecell domain.

The difference in the encoding of FDR variables had the biggest impact on the merge-and-shrink heuristic (`ms`). The reason is that the more facts are covered by a single mutex groups, the bigger and more informative are the resulting FDR variables and therefore also the atomic projections (which form a starting point for the merge-and-shrink heuristic). On one hand, this leads to the faster construction of the heuristic (because fewer variables means less merge steps). On the other hand, more informative FDR variables produce more accurate heuristic. The difference between $\texttt{fam}_{\texttt{fd}}$ and $\texttt{h}^2$ is mainly due to the tidybot and parking domains. The tidybot was problematic for $\texttt{h}^2$ for the same reasons as before—without pruning, the inference of mutex groups consumes all memory. In the parking domains, the problem was the greedy selection of mutex groups used for the construction of FDR variables. $\texttt{h}^2$ found a richer set of mutex groups than $\texttt{fam}_{\texttt{fd}}$, but the resulting number of FDR variables was often higher with $\texttt{h}^2$, i.e., more informative mutex groups did not translate to more informative FDR variables. This suggest that it would be beneficial to improve the merge-and-shrink heuristic so that it can use mutex groups directly instead of FDR variables constructed from the mutex groups.

5.6 Summary

In this chapter, we have introduced a new 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 a 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. This means that mutex groups constructed from h^2-mutexes are supersets of fam-groups, but they does not necessarily have the properties of fam-groups relating to dead-end states.

The algorithm for the inference of fam-groups was compared to 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. Our algorithm generated a richer set of mutex groups in most of the tested domains and in the rest of the domains 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.

Chapter 6

Pruning Tasks with Fact-Alternating Mutex Groups

As we have mentioned before, mutual exclusion invariants can be used to remove facts and operators that cannot be part of any plan—we call this process *pruning of planning tasks*. Almost all planning techniques can benefit from pruning, because it reduces the size of planning tasks and allows the selected planning technique to concentrate on the features of planning tasks that are useful for finding the plan. We have previously shown, that fam-groups have a property that allows to use them for the detection of not only unreachable operators, but also operators that can reach only dead-end states. In this chapter, we propose a simple fixpoint pruning algorithm that utilizes this property and we experimentally evaluate how beneficial the resulting pruning is for heuristic search planners.

6.1 Pruning Algorithm

One obvious way to use mutexes and mutex groups to remove unreachable facts and operators is to check preconditions of operators. If we find an operator with the precondition containing more than one fact from some mutex group, then we can remove such operator, because it cannot be reached from the initial state. Fact-alternating mutex groups provide another method to remove 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 6.1.

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 vertices and an edge $f_1 \rightarrow f_2$ connecting every pair of facts f_1 and f_2 if $f_1 \in \text{pre}(o)$ and $f_2 \in \text{add}(o) \cup \text{del}(o)$ 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 6.1 shows how irrelevant facts can be found without actually constructing a causal graph.

Algorithm 6.1: Pruning of a planning task using inferred fam-groups.

```
Input: STRIPS planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$
Output: Planning task $\Pi^\star = \langle \mathcal{F}^\star, \mathcal{O}^\star, s_I^\star, s_G \rangle$, set of fam-groups $\mathcal{M}$
1  $\mathcal{F}^\star \leftarrow \mathcal{F}$, $\mathcal{O}^\star \leftarrow \mathcal{O}$, $s_I^\star \leftarrow s_I$;
2  do
3      Remove facts returned by IrrelevantFacts($\mathcal{F}^\star, \mathcal{O}^\star, s_G$) from $\mathcal{F}^\star$, $s_I^\star$ and all
         operators in $\mathcal{O}^\star$;
4      Use Algorithm 5.1 with $\Pi^\star$ and store the inferred fam-groups into $\mathcal{M}$;
5      for each $o \in \mathcal{O}^\star$ and $M \in \mathcal{M}$ do
6          if $|\mathrm{pre}(o) \cap M| \geq 2$ or $|\mathrm{add}(o) \cap M| \geq 2$ then Remove $o$ from $\mathcal{O}^\star$;
7      for each $o \in \mathcal{O}^\star$ and $M \in \mathcal{M}$ such that $|M \cap s_G| \geq 1$ do
8          if $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \geq 1$ and $|M \cap \mathrm{add}(o)| = 0$ then Remove $o$ from $\mathcal{O}^\star$;
9  while change in $\mathcal{F}^\star$ or $\mathcal{O}^\star$ occurred;

10 function IrrelevantFacts($\mathcal{F}, \mathcal{O}, s_G$)
11     $U \leftarrow s_G$;
12     $R \leftarrow \emptyset$;
13     while $|U| > 0$ do
14         $f \leftarrow$ choose any fact from $U$;
15         $U \leftarrow U \setminus \{f\}$;
16         if $f \notin R$ then
17             $R \leftarrow R \cup \{f\}$;
18             for each $o \in \mathcal{O}$ such that $f \in (\mathrm{add}(o) \cup \mathrm{del}(o))$ do
19                 $U \leftarrow U \cup \{g \mid g \in \mathrm{pre}(o),\ g \notin R\}$;
20     return $\mathcal{F} \setminus R$;
```

The inferred fam-groups (line 4) are used for the removal of unreachable operators (lines 5–6) which are those that cannot be applied in any reachable state as already discussed in Section 4.1. Finally, fam-groups containing a fact from the goal specification are used for removal of dead-end operators (lines 7–8) according to Corollary 5.6.

Algorithm 6.1 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 7–8) cannot be used, because Corollary 5.6 does not generally hold for any type of mutex group.

6.2 Experimental Evaluation

For the evaluation of Algorithm 6.1, we used the same cluster as for the experiments from Section 5.5 with 8 GB and 30 minutes memory and time limit, respectively. All tested algorithms were implemented[1] in C 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 Fast Downward planner (Helmert, 2006) with the same admissible heuristics as in Section 5.5.4, i.e., the LM-Cut (`lmc`) heuristic (Helmert & Domshlak, 2009), the potential (`pot-all`) heuristic optimized for all syntactic states (Seipp et al., 2015), and the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy and non-greedy bi-simulation shrink strategy (Helmert et al., 2014; Sievers et al., 2016). We used only domains from

[1]https://gitlab.com/danfis/cpddl

domain	**perc. of removed operators**					**dead-end**		**perc. of removed facts**				
	`fd`	$\mathtt{h}^{2\star}$	$\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	`fd`	$\mathtt{h}^{2\star}$	$\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$
agricola18 (20)	0.00	**63.57**	14.36	14.36	**63.57**	0.00	0.00	0.00	**4.35**	3.80	3.80	**4.35**
airport04 (21)	0.00	55.48	10.76	17.46	**62.56**	6.74	**6.91**	0.00	46.46	46.02	47.04	**50.78**
barman11 (20)	16.09	26.92	16.09	43.19	**53.14**	**27.10**	26.21	0.00	0.00	0.00	**18.76**	**18.76**
barman14 (14)	15.53	25.58	15.53	44.48	**53.97**	**28.95**	28.40	0.00	0.00	0.00	**21.26**	**21.26**
caldera18 (20)	0.00	**9.68**	**9.68**	**9.68**	**9.68**	0.00	0.00	0.00	**34.26**	**34.26**	**34.26**	**34.26**
cavediving14 (20)	0.00	**0.27**	0.00	0.00	**0.27**	0.00	0.00	0.00	**2.27**	**2.27**	**2.27**	**2.27**
depot02 (22)	4.12	**24.26**	4.12	4.12	**24.26**	0.00	0.00	**2.49**	**2.49**	**2.49**	**2.49**	**2.49**
driverlog02 (20)	0.00	**2.04**	**2.04**	**2.04**	**2.04**	0.00	0.00	0.00	**2.48**	**2.48**	**2.48**	**2.48**
floortile11 (20)	0.00	0.00	0.00	**22.96**	**22.96**	22.96	22.96	0.00	**20.07**	**20.07**	**20.07**	**20.07**
floortile14 (20)	0.00	0.00	0.00	**22.83**	**22.83**	22.83	22.83	0.00	**22.54**	**22.54**	**22.54**	**22.54**
freecell00 (80)	0.01	**0.06**	0.02	0.02	**0.06**	0.00	0.00	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
logistics00 (28)	0.00	**7.92**	**7.92**	**7.92**	**7.92**	0.00	0.00	0.00	**8.12**	**8.12**	**8.12**	**8.12**
logistics98 (34)	0.00	**19.74**	**19.74**	**19.74**	**19.74**	0.00	0.00	0.00	**23.16**	**23.16**	**23.16**	**23.16**
maintenance14 (5)	0.00	6.25	6.25	**21.53**	**21.53**	15.28	15.28	0.00	**6.92**	**6.92**	**6.92**	**6.92**
mprime98 (35)	0.00	**11.50**	0.00	0.00	**11.50**	0.00	0.00	0.00	**0.69**	0.00	0.00	**0.69**
mystery98 (30)	0.00	**36.89**	0.02	0.02	**36.89**	0.00	0.00	0.00	**33.20**	2.47	2.47	**33.20**
nomystery11 (20)	0.00	**23.25**	0.00	0.00	**23.25**	0.00	0.00	0.00	**6.18**	1.13	1.13	**6.18**
organic-synth18 (8)	0.00	93.73	88.72	93.13	**95.68**	**2.72**	1.90	0.00	53.35	49.09	61.89	**69.21**
parcprinter08 (30)	0.00	3.23	3.23	63.77	**68.66**	54.47	**55.74**	0.00	35.71	35.71	**51.80**	**51.80**
parcprinter11 (20)	0.00	5.55	5.55	62.75	**67.39**	50.69	**51.61**	0.00	32.34	32.34	**51.11**	**51.11**
parking11 (20)	3.57	5.81	3.57	**7.13**	**7.13**	**3.57**	1.32	0.00	**3.35**	**3.35**	**3.35**	**3.35**
parking14 (20)	3.78	6.15	3.78	**7.56**	**7.56**	**3.78**	1.41	0.00	**3.53**	**3.53**	**3.53**	**3.53**
pathways06 (30)	0.00	**2.30**	**2.30**	**2.30**	**2.30**	0.00	0.00	0.00	**26.70**	**26.70**	**26.70**	**26.70**
pegsol08 (30)	0.00	13.47	8.98	11.97	**16.29**	**3.68**	2.45	0.00	5.27	4.83	8.15	**8.56**
pegsol11 (20)	0.00	5.43	1.32	5.68	**10.27**	**3.54**	3.32	0.00	0.10	0.10	**1.90**	**1.90**
pipesw-notank04 (44)	0.98	**6.38**	0.98	0.98	**6.38**	0.00	0.00	0.89	**3.62**	0.89	0.89	**3.62**
pipesw-tank04 (50)	4.26	**5.26**	5.25	5.25	**5.26**	0.00	0.00	0.53	**5.80**	**5.80**	**5.80**	**5.80**
psr-small04 (50)	0.00	**16.62**	15.49	15.49	**16.62**	0.00	0.00	0.00	**29.94**	27.91	27.91	**29.94**
rovers06 (40)	0.00	**23.41**	**23.41**	**23.41**	**23.41**	0.00	0.00	0.00	**29.08**	**29.08**	**29.08**	**29.08**
satellite02 (36)	0.00	**2.19**	**2.19**	**2.19**	**2.19**	0.00	0.00	0.00	**24.71**	**24.71**	**24.71**	**24.71**
scanalyzer08 (30)	0.41	**35.28**	35.27	35.27	**35.28**	0.00	0.00	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
scanalyzer11 (20)	0.72	**33.05**	33.05	33.05	**33.05**	0.00	0.00	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
snake18 (20)	0.00	**11.59**	11.41	11.41	**11.59**	0.00	0.00	0.00	**11.02**	**11.02**	**11.02**	**11.02**
sokoban08 (30)	0.00	0.28	0.00	0.02	**0.29**	**0.02**	0.01	0.00	**16.78**	16.75	16.75	**16.78**
sokoban11 (20)	0.00	0.36	0.00	0.03	**0.38**	**0.03**	0.01	0.00	**15.96**	15.91	15.91	**15.96**
spider18 (16)	0.00	15.01	5.48	5.48	**15.02**	**0.86**	0.01	0.00	**4.52**	1.43	1.43	**4.52**
tetris14 (17)	0.00	**93.85**	**93.85**	**93.85**	**93.85**	0.00	0.00	0.00	**71.94**	**71.94**	**71.94**	**71.94**
tidybot11 (20)	0.00	**42.89**	0.17	0.17	**42.89**	0.00	0.00	0.00	**39.56**	29.70	29.70	**39.56**
tidybot14 (20)	0.00	**36.68**	0.15	0.15	**36.68**	0.00	0.00	0.00	**37.02**	30.03	30.03	**37.02**
tpp06 (30)	0.00	41.69	20.13	40.26	**61.82**	**20.14**	20.13	0.00	19.11	12.89	14.82	**21.04**
trucks06 (30)	0.00	1.32	0.00	89.10	**89.32**	**89.10**	88.01	0.00	**89.45**	**89.45**	**89.45**	**89.45**
woodworking08 (30)	0.00	44.99	0.24	8.64	**52.27**	**8.06**	6.80	0.00	10.44	10.31	11.51	**11.71**
woodworking11 (20)	0.00	46.11	0.20	7.95	**52.66**	**7.42**	5.98	0.00	9.96	9.88	11.08	**11.34**
zenotravel02 (20)	0.00	**0.21**	**0.21**	**0.21**	**0.21**	0.00	0.00	0.00	**1.22**	**1.22**	**1.22**	**1.22**
overall (1625)	0.60	15.47	9.50	12.94	**18.80**	**3.45**	3.32	0.15	22.56	21.19	21.80	**23.23**

Table 6.1: Percentage of removed operators and facts for tasks in which all methods finished within the time and memory limits, only the domains with a difference are shown. Left: Percentage of removed operators per domain and over all domains, maximums highlighted. Middle: Percentage of removed operators that were detected as dead-end operators, maximums highlighted if there is a difference. Right: Percentage of removed facts per domain and over all domains, maximums highlighted.

the optimal track of IPC. We compare our pruning algorithm with `fd` and $\mathtt{h}^2$ (Section 5.5) in two ways. First in Section 6.2.1, we use only forward pruning where we discover superfluous facts and operators only in progression from the initial state towards goals. Then in Section 6.2.2, we use the technique proposed by Alcázar & Torralba (2015) that alternates between pruning in the forward direction (progression) and in the backward direction (regression, i.e., it starts in the goal and proceeds towards the initial state).

6.2.1 Forward Pruning of Planning Tasks

Although Algorithm 6.1 is formulated specifically for `fam`, we also described how the algorithm can be altered to include different inference algorithms. Algorithm 6.1 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

domain	lmc					pot-all					ms				
	`fd`	$\mathtt{h}^{2\star}$	$\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	`fd`	$\mathtt{h}^{2\star}$	$\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	`fd`	$\mathtt{h}^{2\star}$	$\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$
agricola18 (20)	0	0	0	0	0	1	1	**2**	**2**	1	3	6	**10**	**10**	7
airport04 (50)	**29**	28	28	23	21	23	**26**	25	22	20	16	**19**	**19**	**19**	**19**
barman11 (20)	4	4	4	4	**8**	8	8	8	8	8	8	8	8	8	8
barman14 (14)	0	0	0	0	**2**	3	3	3	3	3	3	3	3	3	3
caldera18 (20)	8	**12**	**12**	**12**	**12**	8	**12**	**12**	**12**	**12**	9	**12**	**12**	**12**	**12**
depot02 (22)	7	7	7	7	7	7	**11**	7	7	**11**	7	**11**	7	7	**11**
floortile11 (20)	8	8	8	8	8	6	6	6	6	6	4	6	6	**7**	**7**
floortile14 (20)	8	8	8	8	8	3	**5**	**5**	**5**	**5**	2	2	2	**5**	**5**
freecell00 (80)	15	15	15	15	15	40	**65**	64	64	**65**	20	**61**	**61**	**61**	**61**
logistics00 (28)	20	20	20	20	20	17	**19**	**19**	**19**	**19**	20	20	20	20	20
mprime98 (35)	24	**25**	24	24	**25**	24	24	24	24	24	24	24	24	24	24
mystery98 (30)	17	**19**	17	17	**19**	17	**19**	17	17	**19**	17	**18**	17	17	**18**
nomystery11 (20)	15	**17**	16	15	**17**	14	14	14	14	14	20	20	20	20	20
organic-synth18 (20)	**10**	**10**	9	9	8	**10**	**10**	9	9	8	9	**10**	9	9	8
parcprinter08 (30)	19	19	19	**22**	**22**	22	22	22	**27**	**27**	**27**	25	**27**	**27**	**27**
parcprinter11 (20)	14	14	14	**17**	**17**	16	18	18	**19**	**19**	**20**	18	**20**	**20**	**20**
parking11 (20)	4	4	4	4	4	**8**	6	**8**	**8**	**8**	**8**	0	**8**	**8**	**8**
parking14 (20)	4	4	4	4	4	**8**	5	**8**	**8**	**8**	**8**	0	**8**	**8**	**8**
petri-net-align18 (20)	**9**	7	0	0	0	**7**	3	0	0	0	**7**	0	0	0	0
pipesw-notank04 (50)	18	18	18	18	18	20	20	20	20	**21**	22	**24**	22	22	22
pipesw-tank04 (50)	13	13	13	13	13	17	17	**18**	**18**	17	15	**16**	**16**	**16**	**16**
psr-small04 (50)	48	**49**	**49**	**49**	**49**	50	50	50	50	50	50	50	50	50	50
rovers06 (40)	7	**9**	**9**	**9**	**9**	5	**7**	**7**	**7**	**7**	6	**8**	**8**	**8**	**8**
satellite02 (36)	7	7	7	7	7	5	**6**	**6**	**6**	**6**	6	**7**	**7**	**7**	**7**
scanalyzer08 (30)	16	**17**	**17**	**17**	**17**	13	13	13	13	13	13	13	13	13	13
scanalyzer11 (20)	13	**14**	**14**	**14**	**14**	10	10	10	10	10	10	10	10	10	10
snake18 (20)	7	7	7	7	7	**14**	13	13	13	13	7	14	**15**	**15**	**15**
spider18 (20)	**11**	**11**	10	10	**11**	12	**14**	11	11	13	6	**12**	11	11	**12**
tetris14 (17)	6	**9**	**9**	**9**	**9**	13	**16**	**16**	**16**	**16**	2	**11**	**11**	**11**	**11**
tidybot11 (20)	14	**17**	14	14	**17**	14	14	14	14	14	8	**16**	13	13	**16**
tidybot14 (20)	10	**13**	10	10	**13**	10	10	10	10	10	0	**9**	3	3	**9**
transport11 (20)	**7**	6	6	6	6	6	6	6	6	6	6	6	6	6	6
trucks06 (30)	8	10	10	**13**	**13**	7	12	12	**14**	**14**	5	**9**	**9**	**9**	**9**
visitall11 (20)	11	**12**	**12**	**12**	**12**	16	**17**	**17**	**17**	**17**	9	9	9	9	9
visitall14 (20)	5	**6**	**6**	**6**	**6**	13	13	13	13	13	3	**4**	**4**	**4**	**4**
woodworking08 (30)	19	21	19	**23**	**23**	13	**14**	13	13	**14**	17	16	17	**20**	**20**
woodworking11 (20)	13	15	13	**16**	15	8	**9**	8	8	**9**	11	10	11	**14**	**14**
Σ (1697)	841	868	845	855	**869**	868	918	908	913	**920**	827	906	915	925	**936**
Σ \ petri-net-al18 (1677)	832	861	845	855	**869**	861	915	908	913	**920**	820	906	915	925	**936**

Table 6.2: Comparison of different algorithms used for pruning of the planning tasks and the inference FDR variables. Number of solved tasks in each domain and overall, only the domains with a difference are shown, maximums are highlighted.

not utilized in this configuration because the properties required (Corollary 5.6) for this operation are not proven for the mutex groups inferred by `fd` (the relation between `fd` and `fam` is resolved in Chapter 7).

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 6.1 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 by $\mathtt{h}^{2\star}$ do not have the required properties (see Section 5.3). In the case of fam-groups, we use the $\mathtt{fam}_{\mathtt{fd}}$ variant of the algorithm because it is less time demanding than `fam` in most cases.

To demonstrate the impact of the removal of dead-end operators, we have also evaluated Algorithm 6.1 without the removal of dead-end operators (i.e., with lines 7 and 8 of Algorithm 6.1 disabled) denoted by $\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$. We also tried the combination of $\mathtt{h}^{2\star}$ and $\mathtt{fam}_{\mathtt{fd}}$ where we use h^2 to prune unreachable operators (i.e., in the place of lines 5 and 6 of Algorithm 6.1), but we also prune dead-end operators with $\mathtt{fam}_{\mathtt{fd}}$ and use fam-groups for the creation of FDR variables. The combination of $\mathtt{h}^{2\star}$ and $\mathtt{fam}_{\mathtt{fd}}$ is denoted by $\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$.

Table 6.1 shows a relative pruning power of each evaluated variant as a percentage of operators and facts that were removed from the grounded PDDL. Note that the number of removed dead-end operators for $\mathtt{fam}_{\mathtt{fd}}$ does not necessarily equal to the difference between the number of removed operators by $\mathtt{fam}^{\mathtt{de}}_{\mathtt{fd}}$ and $\mathtt{fam}_{\mathtt{fd}}$. The reason is that the pruning algorithm runs in cycles and the removal of some dead-end operators can cause the removal of more operators in the next cycle because a different set of mutex groups is inferred. Furthermore, $\texttt{fam}_{\texttt{fd}}$ report more removed dead-end operators than $\texttt{fam}_{\texttt{fd}}^{\texttt{h}^2}$ in some cases, because the pruning using h^2 precedes the pruning of dead-end operators.

Considering the relationship between h^2-mutexes and fam-groups, $\texttt{h}^{2\star}$ must always remove a superset of operators of those removed by $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$, but the same does not hold for $\texttt{fam}_{\texttt{fd}}$ because of its ability to detect dead-end operators. Similarly, considering the results presented in Section 5.5.1, where it was shown that `fd` produced a subset of `fam` mutex groups, the operators removed by `fd` must be a subset of the operators removed by $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$, $\texttt{fam}_{\texttt{fd}}$ and, thus, also by $\texttt{h}^{2\star}$. Finally, $\texttt{fam}_{\texttt{fd}}^{\texttt{h}^2}$ necessarily removes a superset of operators removed by both $\texttt{h}^{2\star}$ and $\texttt{fam}_{\texttt{fd}}$, because it combines both of these methods.

The worst performance, in terms of the removed operators, was shown by `fd`, which was expected given the results presented in the previous sections. The most interesting results were measured in airport, barman, floortile, maintenance, parcprinter, and trucks domains, where the detection of dead-end operators significantly increased the number of removed operators in contrast to $\texttt{h}^{2\star}$. 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 6.2. Note, that in the petri-net-alignment domain, no task was solved by $\texttt{fam}_{\texttt{fd}}$, $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$, or $\texttt{fam}_{\texttt{fd}}^{\texttt{h}^2}$, because the inference of fam-groups consumed all allocated time. The lowest coverage overall was recorded for the planner with `fd` for all tested heuristics. This indicates that the shorter time spent in the 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 memory requirements and more operators in the planning tasks result in wider branching during a state space exploration and less informed heuristics.

Comparison of $\texttt{fam}_{\texttt{fd}}$ and $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$ clearly shows that pruning of dead-end operators pays off: $\texttt{fam}_{\texttt{fd}}$ has higher (or equal) coverage than $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$ in all domains for all heuristics except for `lmc` and `pot-all` in the airport domain and for `lmc` in the nomystery domain. In both cases, the reason is that the fixpoint pruning of dead-end operators took too much time. The increase of coverage by $\texttt{fam}_{\texttt{fd}}$ in comparison to $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$ (and by $\texttt{fam}_{\texttt{fd}}^{\texttt{h}^2}$ in comparison to both $\texttt{fam}_{\texttt{fd}}$ and $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$) corresponds to more effective pruning because all these methods use the same set of fam-groups for the construction of FDR variables.

For `lmc` and `pot-all`, $\texttt{h}^{2\star}$ solved more tasks than $\texttt{fam}_{\texttt{fd}}^{\texttt{de}}$ and $\texttt{fam}_{\texttt{fd}}$ overall, but note that the results for `pot-all` are almost identical with $\texttt{fam}_{\texttt{fd}}$ and $\texttt{h}^{2\star}$. For `ms`, however, $\texttt{fam}_{\texttt{fd}}$ solved significantly more tasks than $\texttt{h}^{2\star}$ because of more concise encoding of FDR variables, which resulted in more informed abstractions computed faster as was demonstrated in Section 5.5.4 also. The best results were recorded for $\texttt{fam}_{\texttt{fd}}^{\texttt{h}^2}$ which managed to reproduce the best coverage of $\texttt{h}^{2\star}$ and $\texttt{fam}_{\texttt{fd}}$ in most domains. So it seems that the increased computational time of $\texttt{fam}_{\texttt{fd}}^{\texttt{h}^2}$ rarely impacted the number of solved task negatively.

6.2.2 Forward and Backward Pruning

In the previous section, we have experimentally evaluated the pruning of planning tasks using mutex groups inferred in progression. In this section, we evaluate the pruning technique proposed by Alcázar & Torralba (2015) that alternates between pruning in progression and regression using h^2. 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

domain	**perc. of removed operators**					**perc. of removed facts**				
	`fd`	$\texttt{fam}_{\texttt{fd}}$	$\texttt{fam}^{\texttt{h}^2}_{\texttt{fd}}$	$\texttt{at}_{\texttt{fd}}$	$\texttt{at}_{\texttt{fam}}$	`fd`	$\texttt{fam}_{\texttt{fd}}$	$\texttt{fam}^{\texttt{h}^2}_{\texttt{fd}}$	$\texttt{at}_{\texttt{fd}}$	$\texttt{at}_{\texttt{fam}}$
agricola18 (20)	0.00	14.36	**63.57**	**63.57**	**63.57**	0.00	3.80	**4.35**	**4.35**	**4.35**
airport04 (21)	0.00	17.46	62.56	**77.12**	**77.12**	0.00	47.04	50.78	**62.80**	**62.80**
barman11 (20)	16.09	43.19	53.14	26.92	**53.97**	0.00	18.76	18.76	0.00	**19.48**
barman14 (14)	15.53	44.48	53.97	25.58	**55.46**	0.00	21.26	21.26	0.00	**22.59**
caldera18 (20)	0.00	9.68	9.68	**67.56**	**67.56**	0.00	34.26	34.26	**38.20**	**38.20**
cavediving14 (20)	0.00	0.00	**0.27**	**0.27**	**0.27**	0.00	**2.27**	**2.27**	**2.27**	**2.27**
depot02 (22)	4.12	4.12	**24.26**	**24.26**	**24.26**	**2.49**	**2.49**	**2.49**	**2.49**	**2.49**
driverlog02 (20)	0.00	**2.04**	**2.04**	**2.04**	**2.04**	0.00	**2.48**	**2.48**	**2.48**	**2.48**
floortile11 (20)	0.00	22.96	22.96	**37.88**	**37.88**	0.00	**20.07**	**20.07**	**20.07**	**20.07**
floortile14 (20)	0.00	22.83	22.83	**37.96**	**37.96**	0.00	**22.54**	**22.54**	**22.54**	**22.54**
freecell00 (80)	0.01	0.02	**0.06**	**0.06**	**0.06**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
logistics00 (28)	0.00	**7.92**	**7.92**	**7.92**	**7.92**	0.00	**8.12**	**8.12**	**8.12**	**8.12**
logistics98 (34)	0.00	**19.74**	**19.74**	**19.74**	**19.74**	0.00	**23.16**	**23.16**	**23.16**	**23.16**
maintenance14 (5)	0.00	**21.53**	**21.53**	6.25	**21.53**	0.00	**6.92**	**6.92**	**6.92**	**6.92**
mprime98 (35)	0.00	0.00	**11.50**	**11.50**	**11.50**	0.00	0.00	**0.69**	**0.69**	**0.69**
mystery98 (30)	0.00	0.02	36.89	38.85	**39.41**	0.00	2.47	33.20	38.08	**38.63**
nomystery11 (20)	0.00	0.00	23.25	**23.26**	**23.26**	0.00	1.13	6.18	**6.25**	**6.25**
organic-synth18 (8)	0.00	93.13	95.68	**98.67**	**98.67**	0.00	61.89	69.21	**71.04**	**71.04**
parcprinter08 (30)	0.00	63.77	68.66	56.61	**71.92**	0.00	**51.80**	**51.80**	51.78	**51.80**
parcprinter11 (20)	0.00	62.75	67.39	57.85	**70.49**	0.00	**51.11**	**51.11**	**51.11**	**51.11**
parking11 (20)	3.57	**7.13**	**7.13**	**7.13**	**7.13**	0.00	**3.35**	**3.35**	**3.35**	**3.35**
parking14 (20)	3.78	**7.56**	**7.56**	**7.56**	**7.56**	0.00	**3.53**	**3.53**	**3.53**	**3.53**
pathways06 (30)	0.00	**2.30**	**2.30**	**2.30**	**2.30**	0.00	**26.70**	**26.70**	**26.70**	**26.70**
pegsol08 (30)	0.00	11.97	16.29	17.00	**18.41**	0.00	8.15	8.56	9.28	**10.34**
pegsol11 (20)	0.00	5.68	10.27	5.43	**11.30**	0.00	1.90	1.90	0.10	**2.70**
pipesw-notank04 (50)	0.60	0.60	**5.51**	**5.51**	**5.51**	0.62	0.62	**3.29**	**3.29**	**3.29**
pipesw-tank04 (50)	4.26	5.25	5.26	**5.29**	**5.29**	0.53	5.80	5.80	**5.85**	**5.85**
psr-small04 (50)	0.00	15.49	**16.62**	**16.62**	**16.62**	0.00	27.91	**29.94**	**29.94**	**29.94**
rovers06 (40)	0.00	**23.41**	**23.41**	**23.41**	**23.41**	0.00	**29.08**	**29.08**	**29.08**	**29.08**
satellite02 (36)	0.00	**2.19**	**2.19**	**2.19**	**2.19**	0.00	**24.71**	**24.71**	**24.71**	**24.71**
scanalyzer08 (30)	0.41	35.27	**35.28**	**35.28**	**35.28**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
scanalyzer11 (20)	0.72	33.05	**33.05**	**33.05**	**33.05**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
snake18 (20)	0.00	11.41	11.59	11.59	**11.88**	0.00	**11.02**	**11.02**	**11.02**	**11.02**
sokoban08 (30)	0.00	0.02	0.29	**22.89**	**22.89**	0.00	16.75	16.78	**25.84**	**25.84**
sokoban11 (20)	0.00	0.03	0.38	**26.67**	**26.67**	0.00	15.91	15.96	**26.87**	**26.87**
spider18 (16)	0.00	5.48	**15.02**	**15.02**	**15.02**	0.00	1.43	**4.52**	**4.52**	**4.52**
tetris14 (14)	0.00	**92.65**	**92.65**	**92.65**	**92.65**	0.00	**68.12**	**68.12**	**68.12**	**68.12**
tidybot11 (20)	0.00	0.17	**42.89**	**42.89**	**42.89**	0.00	29.70	**39.56**	**39.56**	**39.56**
tidybot14 (20)	0.00	0.15	**36.68**	**36.68**	**36.68**	0.00	30.03	**37.02**	**37.02**	**37.02**
tpp06 (30)	0.00	40.26	**61.82**	**61.82**	**61.82**	0.00	14.82	**21.04**	**21.04**	**21.04**
trucks06 (30)	0.00	89.10	**89.32**	23.51	**89.32**	0.00	**89.45**	**89.45**	**89.45**	**89.45**
woodworking08 (30)	0.00	8.64	52.27	**52.76**	**52.76**	0.00	11.51	11.71	**12.19**	**12.19**
woodworking11 (20)	0.00	7.95	52.66	**53.13**	**53.13**	0.00	11.08	11.34	**11.76**	**11.76**
zenotravel02 (20)	0.00	**0.21**	**0.21**	**0.21**	**0.21**	0.00	**1.22**	**1.22**	**1.22**	**1.22**
overall (1628)	0.60	12.45	18.33	16.88	**19.00**	0.15	20.96	22.42	22.83	**23.02**

Table 6.3: Percentage of removed operators and facts for tasks in which all methods finished within the time and memory limits, only the domains with a difference are shown. Left: Percentage of removed operators per domain and over all domains, maximums highlighted. Right: Percentage of removed facts per domain and over all domains, maximums highlighted.

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. 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 & Torralba (2015) works with tasks 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 $\texttt{fam}_{\texttt{fd}}$ will be denoted as $\texttt{at}_{\texttt{fam}}$. These two variants are compared with `fd` as a baseline and with our algorithm $\texttt{fam}_{\texttt{fd}}$ and $\texttt{fam}^{\texttt{h}^2}_{\texttt{fd}}$.

Table 6.3 shows the percentage of removed operators and facts. (Note that the numbers for `fd`, $\texttt{fam}_{\texttt{fd}}$, and $\texttt{fam}^{\texttt{h}^2}_{\texttt{fd}}$ may be a little bit different than in Table 6.1, because we count only the tasks in which all variants finished within the time and memory limits.) Overall, there is only a small difference between $\texttt{fam}^{\texttt{h}^2}_{\texttt{fd}}$ and the variants that use h^2 in regression

domain	lmc					pot-all					ms				
	$\mathtt{fd}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	$\mathtt{at}_{\mathtt{fd}}$	$\mathtt{at}_{\mathtt{fam}}$	$\mathtt{fd}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	$\mathtt{at}_{\mathtt{fd}}$	$\mathtt{at}_{\mathtt{fam}}$	$\mathtt{fd}$	$\mathtt{fam}_{\mathtt{fd}}$	$\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$	$\mathtt{at}_{\mathtt{fd}}$	$\mathtt{at}_{\mathtt{fam}}$
agricola18 (20)	0	0	0	0	0	1	**2**	1	1	1	3	**10**	7	3	7
airport04 (50)	**29**	23	21	**29**	28	23	22	20	**30**	27	16	19	19	**22**	21
barman11 (20)	4	4	**8**	4	**8**	8	8	8	8	8	8	8	8	8	8
barman14 (14)	0	0	**2**	0	**2**	3	3	3	3	3	3	3	3	3	3
caldera18 (20)	8	**12**	**12**	**12**	**12**	8	**12**	**12**	**12**	**12**	9	**12**	**12**	**12**	**12**
depot02 (22)	7	7	7	7	7	7	7	**11**	**11**	**11**	7	7	**11**	**11**	**11**
floortile11 (20)	8	8	8	**14**	**14**	6	6	6	**8**	**8**	4	7	7	**8**	**8**
floortile14 (20)	8	8	8	**20**	**20**	3	5	5	**8**	**8**	2	5	5	**8**	**8**
freecell00 (80)	15	15	15	15	15	40	64	**65**	41	**65**	20	**61**	**61**	20	**61**
logistics00 (28)	20	20	20	20	20	17	**19**	**19**	**19**	**19**	20	20	20	20	20
mprime98 (35)	24	24	**25**	**25**	**25**	24	24	24	24	24	24	24	24	24	24
mystery98 (30)	17	17	**19**	17	17	17	17	**19**	17	17	17	17	**18**	17	16
nomystery11 (20)	15	15	**17**	15	**17**	14	14	14	14	14	20	20	20	20	20
organic-synth18 (20)	**10**	9	8	**10**	9	**10**	9	8	**10**	9	9	9	8	**10**	9
parcprinter08 (30)	19	22	22	22	**23**	22	**27**	**27**	23	**27**	27	27	27	27	27
parcprinter11 (20)	14	17	17	17	**18**	16	**19**	**19**	18	**19**	20	20	20	20	20
petri-net-align18 (20)	**9**	0	0	**9**	0	**7**	0	0	**7**	0	**7**	0	0	**7**	0
pipesw-notank04 (50)	18	18	18	18	18	20	20	**21**	20	**21**	22	22	22	22	22
pipesw-tank04 (50)	13	13	13	13	13	17	**18**	17	17	**18**	15	**16**	**16**	15	**16**
psr-small04 (50)	48	**49**	**49**	**49**	**49**	50	50	50	50	50	50	50	50	50	50
rovers06 (40)	7	**9**	**9**	**9**	**9**	5	**7**	**7**	**7**	**7**	6	**8**	**8**	**8**	**8**
satellite02 (36)	7	7	7	7	7	5	**6**	**6**	**6**	**6**	6	**7**	**7**	**7**	**7**
scanalyzer08 (30)	16	**17**	**17**	**17**	**17**	13	13	13	13	13	13	13	13	13	13
scanalyzer11 (20)	13	**14**	**14**	**14**	**14**	10	10	10	10	10	10	10	10	10	10
snake18 (20)	7	7	7	7	7	**14**	13	13	**14**	13	7	**15**	**15**	6	**15**
sokoban08 (30)	30	30	30	30	30	28	28	28	**30**	**30**	30	30	30	30	30
spider18 (20)	**11**	10	**11**	**11**	**11**	12	11	13	**15**	13	6	11	**12**	6	**12**
tetris14 (17)	6	**9**	**9**	**9**	**9**	13	**16**	**16**	12	**16**	2	**11**	**11**	10	**11**
tidybot11 (20)	14	14	**17**	**17**	**17**	14	14	14	14	14	8	13	**16**	11	**16**
tidybot14 (20)	10	10	**13**	**13**	**13**	10	10	10	10	10	0	3	**9**	4	**9**
transport11 (20)	**7**	6	6	**7**	6	6	6	6	6	6	6	6	6	6	6
trucks06 (30)	8	**13**	**13**	12	**13**	7	**14**	**14**	12	**14**	5	**9**	**9**	**9**	**9**
visitall11 (20)	11	**12**	**12**	**12**	**12**	16	**17**	**17**	**17**	**17**	9	9	9	9	9
visitall14 (20)	5	**6**	**6**	**6**	**6**	13	13	13	13	13	3	**4**	**4**	**4**	**4**
woodworking08 (30)	19	23	23	**24**	23	13	13	**14**	**14**	**14**	17	**20**	**20**	**20**	**20**
woodworking11 (20)	13	**16**	15	**16**	15	8	8	**9**	**9**	**9**	11	**14**	**14**	**14**	**14**
Σ (1697)	841	855	869	**898**	895	868	913	920	911	**934**	827	925	936	879	**941**
Σ \ petri-net-al18 (1677)	832	855	869	889	**895**	861	913	920	904	**934**	820	925	936	872	**941**

Table 6.4: Comparison of different algorithms used for pruning of the planning tasks and the inference FDR variables. Number of solved tasks in each domain and overall, only the domains with a difference are shown, maximums are highlighted.

($\mathtt{at}_{\mathtt{fd}}$ and $\mathtt{at}_{\mathtt{fam}}$). However, there are domains in which the use of h^2 in regression makes a big difference. For example, both $\mathtt{at}_{\mathtt{fd}}$ and $\mathtt{at}_{\mathtt{fam}}$ pruned significantly more operators than $\mathtt{fam}_{\mathtt{fd}}$ in airport, caldera, floortile, or sokoban. But in the barman, parcprinter, pegsol, or trucks domains, $\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$ pruned more operators than $\mathtt{at}_{\mathtt{fd}}$, which demonstrates how important is to have information-rich mutex groups for computing disambiguation in the regression h^2.

Table 6.4 shows a number of solved tasks for `lmc`, `pot-all`, and `ms` (the results for `fd`, $\mathtt{fam}_{\mathtt{fd}}$, and $\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$ are the same as in Table 6.2, but we repeated them for easier comparison). As in the previous cases, variants using fam-groups were not able to solve any task in the petri-net-alignment domain, because the inference of fam-groups took the whole time limit.

For `lmc`, $\mathtt{at}_{\mathtt{fd}}$ achieved best results overall, but $\mathtt{at}_{\mathtt{fam}}$ is slightly better without the petri-net-alignment domain. The most difference between $\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$ and $\mathtt{at}_{\mathtt{fd}}$ is mainly due to the floortile domain which corresponds to the increased number of pruned operators (cf. Table 6.3). The results for $\mathtt{at}_{\mathtt{fd}}$ and $\mathtt{at}_{\mathtt{fam}}$ are very similar. The main difference is due to the barman domain which, again, corresponds to more effective pruning due to the use of fam-groups for disambiguation.

For `pot-all`, the difference between the compared methods roughly corresponds to the results in pruning (cf. Table 6.3). The difference between $\mathtt{at}_{\mathtt{fd}}$ and $\mathtt{at}_{\mathtt{fam}}$ in the tetris and freecell domains corresponds to the different encoding of FDR variables due to a different set of mutex groups (cf. Table 5.6). Overall, $\mathtt{at}_{\mathtt{fam}}$ solved more tasks than $\mathtt{at}_{\mathtt{fd}}$, but most of the difference was measured in the freecell domain.

The results for the merge-and-shrink heuristic ($\mathtt{ms}$) show that having a concise encoding of FDR variables is as important as pruning for the abstraction heuristics. All variants using fam-groups solved more tasks than $\mathtt{at}_{\mathtt{fd}}$ even if we do not consider the freecell domain (where fam-groups-based methods solved 41 more tasks than both $\mathtt{fd}$ and $\mathtt{at}_{\mathtt{fd}}$). Pruning has, however, also a significant impact on the number of solved tasks. Table 6.4 shows a significant increase from $\mathtt{fd}$ to $\mathtt{at}_{\mathtt{fd}}$, and also less significant increase from $\mathtt{fam}^{\mathtt{h}^2}_{\mathtt{fd}}$ to $\mathtt{at}_{\mathtt{fam}}$. Overall, using fam-groups proved to be useful for the pruning with h^2 also in regression, because a rich set of mutex groups in the combination with disambiguation can increase pruning power similarly to the pruning of dead-end operators evaluated in the previous section.

6.3 Summary

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 three different state-of-the-art heuristics and the results indicate a substantial increase in the overall coverage in some of them. In particular, the ability of fam-groups to detect dead-end states proved to be crucial in the pruning of planning tasks and more concise encoding of the tasks in FDR positively impacted abstraction-based heuristics.

We also compared our algorithm with the state-of-the-art algorithm proposed by Alcázar & 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 & Torralba's method, because it improves the disambiguation process.

Chapter 7

Lifted Mutex Group

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

The inference of mutex groups is an integral part of the process of construction of a concise finite domain representation, because the inferred mutex groups allow us to group sets of (STRIPS) facts into (FDR) variables so that each fact is not encoded as a binary variable. Nowadays, the most commonly used translator from PDDL to FDR is the translator from the Fast-Downward planning system (Helmert, 2006) described by Helmert (2009).

In the previous chapter, we have shown that the inference of the maximum sized mutex group is PSPACE-Complete, but we have also shown that there is a certain subclass of mutex groups, called fact-alternating mutex groups (fam-groups), of which inference is NP-Complete. In this chapter, we focus on the inference of schematic mutex groups in the lifted representation, i.e., lifted mutex groups. We show that the lifted mutex groups described by Helmert (2009) are always fam-groups after grounding because of the constraints posed on their structure and not because of the proposed inference algorithm. We propose an improvement of the Helmert's inference algorithm that produces a richer set of lifted fam-groups. We also utilize the operator-pruning properties of fam-groups, described in Section 5.1 and Chapter 6, during the grounding phase and we show that operators can be pruned even before the PDDL planning problem is fully grounded.

7.1 PDDL and Grounding to STRIPS

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

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

Definition 7.1. A **normalized PDDL task** is a tuple $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$ where $\mathcal{B}$ is a non-empty set of **objects**, $\mathcal{T}$ is a non-empty set of **types** containing a default type denoted by $t_0 \in \mathcal{T}$, objects and types are associated by a total function $\mathcal{D} : \mathcal{T} \mapsto 2^{\mathcal{B}}$ such that $\mathcal{D}(t_0) = \mathcal{B}$ and for every pair of types $t_i, t_j \in \mathcal{T}$ it holds that $\mathcal{D}(t_i) \subseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \supseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \cap \mathcal{D}(t_j) = \emptyset$. $\mathcal{V}$ is a denumerable set of variable symbols, each variable $v \in \mathcal{V}$ has a type $\tau_{var}(v) \in \mathcal{T}$.

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

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

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

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

Definition 7.2. Given a set of variables $V \subseteq \mathcal{V}$, a **grounding** γ **restricted to** V is a function $\gamma : \mathcal{V} \cup \mathcal{B} \mapsto \mathcal{V} \cup \mathcal{B}$ such that $\gamma(v) \in \mathcal{D}(\tau_{var}(v))$ for every $v \in V$, and $\gamma(v) = v$ for every $v \in \mathcal{V} \setminus V$, and $\gamma(o) = o$ for every $o \in \mathcal{B}$, i.e., γ maps each variable $v \in V$ to an object from its corresponding domain $\mathcal{D}(\tau_{var}(v))$, and it is identity for everything else.

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

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

With the grounding defined, we can define the full grounding of a PDDL task as the STRIPS planning task (Definition 3.1), which is constructed by replacing all variables with all possible combinations of objects.

Definition 7.3. Given a normalized PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$, the **full grounding** of $\mathcal{P}$ is a STRIPS planning task $\Pi^{\mathrm{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ constructed as follows.

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

Note that the construction of add and delete effects keeps the operators well-formed according to the definition of the STRIPS planning task (Definition 3.1), i.e., the construction makes sure that $\mathrm{add}(o) \cap \mathrm{del}(o) = \mathrm{pre}(o) \cap \mathrm{add}(o) = \emptyset$ for every operator o.

The full grounding is not what would be used as a grounded representation of a PDDL task in practice. However, we will use it as a tool to prove that a certain lifted structure, if grounded, is a state invariant in the full grounding and therefore it is also a state invariant in the grounded representation obtained by a more constrained grounding.

7.2 Lifted Mutex Groups

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

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

Definition 7.4. An **invariant candidate** is a tuple $\nu = \langle \mathcal{V}^{\mathrm{fix}}[\nu], \mathcal{V}^{\mathrm{cnt}}[\nu], \mathrm{atoms}(\nu)\rangle$, where $\mathcal{V}^{\mathrm{fix}}[\nu] \subset \mathcal{V}$ is a finite set of **fixed variables** and $\mathcal{V}^{\mathrm{cnt}}[\nu] \subset \mathcal{V}$ is a finite set of **counted variables** such that $\mathcal{V}^{\mathrm{fix}}[\nu] \cap \mathcal{V}^{\mathrm{cnt}}[\nu] = \emptyset$, and $\mathrm{atoms}(\nu)$ is a finite set of atoms such that $\mathcal{V}[\mathrm{atoms}(\nu)] = \mathcal{V}^{\mathrm{fix}}[\nu] \cup \mathcal{V}^{\mathrm{cnt}}[\nu]$.

Definition 7.5. An **invariant grounding** is a tuple $\xi = \langle \gamma, G\rangle$, where $\gamma \in \mathcal{G}$ is a grounding and $G \subseteq \mathcal{G}$ is a set of groundings. For an invariant candidate ν, we define a set of all invariant groundings $\mathcal{H}_\nu = \{\langle \gamma, \mathcal{G}_{\mathcal{V}^{\mathrm{cnt}}[\nu]}\rangle \mid \gamma \in \mathcal{G}_{\mathcal{V}^{\mathrm{fix}}[\nu]}\}$. For an invariant candidate ν and a corresponding invariant grounding $\xi = \langle \gamma, G\rangle \in \mathcal{H}_\nu$, we define $\xi\langle\!\langle \nu \rangle\!\rangle = \gamma\langle\!\langle G\langle\!\langle \mathrm{atoms}(\nu)\rangle\!\rangle\rangle\!\rangle$.

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

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

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

Definition 7.6. Let $\Pi^{\mathrm{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G\rangle$ denote the full grounding of $\mathcal{P}$. An invariant candidate ν is a **lifted mutex group** (**lifted fam-group**) if for every invariant grounding $\xi \in \mathcal{H}_\nu$ it holds that $M = \{f_x \mid x \in \xi\langle\!\langle \nu \rangle\!\rangle\} \cap \mathcal{F}$ is a mutex group (fam-group) in $\Pi^{\mathrm{full}}_{\mathcal{P}}$.

Now we have defined lifted mutex groups and we know that if we find them on the lifted level, we can ground them and use them on the ground (STRIPS) level as mutex groups. In the following, we borrow the notions of *balance* and *weight* from Helmert (2009) to formulate sufficient conditions for an invariant candidate to be a lifted fam-group.

Definition 7.7. An invariant candidate ν is **balanced in** action $a \in \mathcal{A}$ if for every invariant grounding $\xi \in \mathcal{H}_\nu$ and every grounding $\gamma \in \mathcal{G}_a$, it holds that for every $\alpha \in \xi\langle\!\langle \nu \rangle\!\rangle \cap \mathrm{add}(\gamma\langle\!\langle a \rangle\!\rangle)$ there exists $\alpha' \in \xi\langle\!\langle \nu \rangle\!\rangle \cap \mathrm{pre}(\gamma\langle\!\langle a \rangle\!\rangle) \cap \mathrm{del}(\gamma\langle\!\langle a \rangle\!\rangle)$.

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

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

Definition 7.8. The **weight** of the invariant candidate ν is

$$\mathrm{weight}(\nu) = \max_{\xi \in \mathcal{H}_\nu, a \in \mathcal{A}, \gamma \in \mathcal{G}_a} \left| \mathrm{add}(\gamma\langle\!\langle a \rangle\!\rangle) \cap \xi\langle\!\langle \nu \rangle\!\rangle \right| .$$

The **init-weight** of the invariant candidate ν is

$$\text{i-weight}(\nu) = \max_{\xi \in \mathcal{H}_\nu} \left| \psi_I \cap \xi\langle\!\langle \nu \rangle\!\rangle \right| .$$

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

Theorem 7.9. *Let $\mathcal{P}$ denote a PDDL task, and let ν denote an invariant candidate. If* $\text{i-weight}(\nu) \leq 1$ *and* $\mathrm{weight}(\nu) \leq 1$ *and ν is balanced, then ν is a lifted fam-group.*

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

Note that the implementation of the tests for the weight and balance do not require to iterate over all possible groundings. The tests can run in polynomial time, because we can look for the renaming of the variables of actions and the invariant candidate. In the case of the weight test, we must iterate (in the worst case) over all pairs of atoms in all (lifted) add effects. And in the case of the balance test, we need to test the combination of every add effect with every precondition that is also a delete effect.

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

We improved upon Helmert's findings by showing that this kind of invariant is not only a (lifted) mutex group, but specifically a (lifted) fam-group. This means that, as we show in the next section, we can use the lifted fam-groups during the grounding process for removing operators that are either unreachable, or that can generate only dead-end states. Moreover, it also means that this kind of invariants always generates a subset of mutexes obtainable from the h^2 heuristic as we proved in Section 5.3.

7.3 Pruned Grounding

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

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

Definition 7.10. Given a normalized PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$ and a **pruning function** $\omega : \mathcal{G}\langle\!\langle \mathcal{A} \rangle\!\rangle \mapsto \{0, 1\}$, the **relaxed grounding** of $\mathcal{P}$ **pruned with** ω is a STRIPS planning task $\Pi^{\mathrm{relax}}_{\mathcal{P},\omega} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ constructed as follows.

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

Since we want to use lifted fam-groups for finding out which operators can be pruned, we need to make sure that (pruned) relaxed groundings preserve lifted fam-groups.

If we remove a set of operators and unreachable facts, then all mutex groups will be preserved, because it can only make less states reachable. However, fam-groups are not defined over the reachable state space, but over the input planning task, so we need to make sure that the conditions from Definition 5.1 hold.

Removing operators preserves also fam-groups, because the second condition in Definition 4.3 ($|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$) must hold for all operators, therefore it must also hold for the operators remaining after the removal. Removing unreachable facts can change the operators, but only their delete effects because removing unreachable facts from a precondition would mean that the corresponding operator is also unreachable. Therefore, $\mathrm{pre}(o) \cap \mathrm{del}(o)$ remains the same for all operators o, which is enough to show that fam-groups are preserved in any relaxed grounding pruned with any pruning function.

Theorem 7.11. *Let $\mathcal{P}$ denote a PDDL task, let ν denote a lifted fam-group, let ω denote a pruning function, and let $\Pi^{\text{relax}}_{\mathcal{P},\omega} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote the relaxed grounding of $\mathcal{P}$ pruned with ω. Then for every invariant grounding $\xi \in \mathcal{H}_\nu$ it holds that $M = \{f_x \mid x \in \xi\langle\!\langle \nu \rangle\!\rangle\} \cap \mathcal{F}$ is a fam-group in $\Pi^{\text{relax}}_{\mathcal{P},\omega}$.*

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

Now we introduce a novel pruning technique on the lifted level that uses lifted fam-groups to remove unreachable and dead-end operators during grounding, i.e., before the (relaxed or full) grounding is explicitly constructed. In Definition 7.12, we define a pruning function that uses lifted fam-groups (1.) to prune unreachable operators and (2.) to prune dead-end operators (recall Corollary 5.5). In Theorem 7.13, we show that with this pruning function, all plans are preserved.

Definition 7.12. Given a set of lifted fam-groups L, the **fam-group pruning function** ω_L is a pruning function such that for every $a \in \mathcal{G}\langle\!\langle \mathcal{A} \rangle\!\rangle$:

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

Theorem 7.13. *Let $\mathcal{P}$ denote a PDDL task, let $\Pi^{\text{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote the full grounding of $\mathcal{P}$, let L denote a set of lifted fam-groups, and let $\Pi^{\text{relax}}_{\mathcal{P},\omega_L} = \langle \mathcal{F}', \mathcal{O}', s_I, s_G \rangle$ denote the relaxed grounding of $\mathcal{P}$ pruned with the fam-group pruning function ω_L. And for a given sequence of operators $\pi = \langle o_1, \dots, o_n \rangle$, let $\pi|_{\mathcal{F}'} = \langle o'_1, \dots, o'_n \rangle$, where $\text{pre}(o'_i) = \text{pre}(o_i)$, $\text{add}(o'_i) = \text{add}(o_i)$, and $\text{del}(o'_i) = \text{del}(o_i) \cap \mathcal{F}'$ for every $i \in \{1, \dots, n\}$. If π is a plan in Π^{full}, then $\pi|_{\mathcal{F}'}$ is a plan in $\Pi^{\text{relax}}_{\mathcal{P},\omega_L}$. And if π' is a plan in $\Pi^{\text{relax}}_{\mathcal{P},\omega_L}$, then there exists a plan π in Π^{full} such that $\pi|_{\mathcal{F}'} = \pi'$.*

Proof Sketch. In Definition 7.12, 1. avoids grounding of unreachable operators, because they would be applicable only in states violating one of the fam-groups, i.e., in the unreachable states, and 2. avoids grounding of operators that can produce only dead-end states (Corollary 5.5 and Theorem 7.11), therefore they cannot be part of any plan. The rest of the reachable operators are the same in Π^{full} and $\Pi^{\text{relax}}_{\mathcal{P},\omega_L}$ except that operators in $\Pi^{\text{relax}}_{\mathcal{P},\omega_L}$ does not have relaxed unreachable facts in their delete effects. □

7.4 Inference of Lifted FAM-Groups

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

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

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

We improve this algorithm by the following:

(i) We allow any number of counted variables in all atoms.

(ii) We introduce new refinement techniques that allow us to refine also the candidates that fail weight test: the refinement of types, and the refinement of counted variables.

(iii) We introduce a new refinement technique for the proved lifted mutex groups so that the algorithm does not stop once an invariant candidate is proved to be a lifted fam-group, but it allows to construct supersets of the proved lifted fam-groups.

Another difference is that we infer directly lifted fam-groups instead of proving monotonicity invariants first. The reason is purely practical. Allowing any number of counted variables can generate a huge number of candidates, but restricting the candidates to those that has init-weight exactly one reduces the number significantly.

The main part of the algorithm is described as a pseudo-code in Algorithm 7.1. The remaining parts are described in the text below. As already mentioned, testing whether there exist groundings so that action atoms can be ground to the same atoms as invariant candidate's atoms can be done in a polynomial number of steps by renaming variables of the atoms. If there exists such a renaming, we say that these atoms can be unified.

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

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

Algorithm 7.1: Inference of lifted fam-groups.

```
Input: A PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$
Output: A set of lifted fam-groups $M$
1  $C \leftarrow \{\langle \emptyset, \{c_1, \dots, c_{\mathrm{ar}(p)}\}, \{p(c_1, \dots, c_{\mathrm{ar}(p)})\} \rangle \mid p \in \mathcal{P}\}$;
2  $M \leftarrow \{\}$;
3  while $|C| > 0$ do
4      $\nu \leftarrow$Pop($C$);
5      if i-weight($\nu$) $= 1$ then
6          if weight($\nu$) $\leq 1$ then
7              if $\nu$ is balanced then
8                  $M \leftarrow M \cup \{\nu\}$;
9                  $C \leftarrow C \cup$ RefineProved($\nu$);
10             else $C \leftarrow C \cup$ RefineUnbalanced($\nu$) ;
11         else $C \leftarrow C \cup$ RefineHeavyAction($\nu$) ;
12     else $C \leftarrow C \cup$ RefineHeavy($\nu$, $\psi_I$) ;
13 return $M$;
14 function RefineProved($\nu$)
15     $C \leftarrow \emptyset$;
16     for each $a \in \mathcal{A}$ s.t. both pre($a$) and del($a$) can be unified with some $\alpha \in$ atoms($\nu$),
         but add($a$) cannot be unified with atoms($\nu$) do
17         $C \leftarrow C \cup$ RefineExtend($\nu$, add($a$));
18     return $C$;
19 function RefineUnbalanced($\nu$)
20     Let $a$ be an action s.t. $\nu$ is not balanced in $a$;
21     Let $\beta \in$ add($a$) be the atom that can be unified with the corresponding atom $\alpha$
         from $\nu$;
22     $C \leftarrow$ Refine types of variables $\mathcal{V}[\alpha]$ so that $\beta$ cannot be unified with the refined $\alpha$;
23     return $C \cup$ RefineExtend($\nu$, del($a$));
24 function RefineExtend($\nu$, $X$)
25     $C \leftarrow \emptyset$;
26     for each $\delta \in X$ s.t. $\mathcal{P}[\delta] \notin \mathcal{P}[\text{atoms}(\nu)]$ do
27         $C \leftarrow C \cup$ Refine $\nu$ by extending it with an atom $\alpha$ of the predicate $\mathcal{P}[\delta]$ s.t. $\delta$
             can be unified with $\alpha$;
28     return $C$;
29 function RefineHeavyAction($\nu$)
30     Let $a \in \mathcal{A}$ be an action failing the weight test;
31     return RefineHeavy($\nu$, add($a$));
32 function RefineHeavy($\nu$, $X$)
33     Let $\beta_1, \beta_2, \beta_1 \neq \beta_2$, be two atoms from $X$ that can be unified with the
         corresponding atoms $\alpha_1, \alpha_2$ from $\nu$;
34     $C \leftarrow$ Refine types of variables $\mathcal{V}[\alpha_1]$ ($\mathcal{V}[\alpha_2]$) so that $\beta_1$ ($\beta_2$) cannot be unified with
         the refined $\alpha_1$ ($\alpha_2$);
35     if $\alpha_1 = \alpha_2$ then
36         $C \leftarrow C \cup$ Refine counted variables from $\mathcal{V}[\alpha_1]$ so that $\{\beta_1, \beta_2\}$ cannot be unified
             with the refined $\alpha_1$;
37     return $C$;
```

For example, assume we have a proved lifted mutex group $\{\mathtt{at}(v, c)\}$ with the fixed variable v and counted variable c (types are not important here). And let us assume that there is the action `load` with pre(`load`) $= \{\mathtt{at}(v_1, v_2)\}$, del(`load`) $= \{\mathtt{at}(v_1, v_2)\}$, and add(`load`) $= \{\mathtt{in}(v_1, v_3)\}$. This action removes the atom $\mathtt{at}(v_1, v_2)$, but does not add any other atom from the corresponding lifted mutex group $\{\mathtt{at}(v, c)\}$. So we can extend the lifted mutex group with an atom of the predicate `in`, which is added by the action `load`, resulting with the invariant candidate $\{\mathtt{at}(v, c), \mathtt{in}(v, c')\}$.

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

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

Refinement of Counted Variables This refinement simply changes counted variables into fixed variables. If the weight test fails because two atoms from some add effect (or the initial state) are covered by a single atom from the invariant candidate because of a counted variable, then changing the counted variable to the fixed variable fixes the weight test.

For example, consider an invariant candidate with a single atom $p(v_1, c_1)$ with a fixed variable v_1 and a counted variable c_1, and the add effect $\{p(w_1, w_2), p(w_1, w_3)\}$. Assuming all variables have the same type, the invariant candidate can be unified with both atoms from the add effect, because the invariant grounding can expand the counted variable c_1 into two different objects covering both w_2 and w_3. Changing c_1 to a fixed variable, however, prevents it, thus the weight test passes.

7.5 Experimental Evaluation

The grounding of PDDL tasks and the inference of lifted fam-groups was implemented[1] in C and experimentally evaluated on a cluster of computing nodes with Intel Xeon Scalable Gold 6146 processors. We implemented both Helmert's original algorithm, referred to as `H`, and our improved algorithm, referred to as `H+`. Both `H` and `H+` ran with the limit on the number of considered invariant candidates set to 10 000. We used domains from all IPCs from 1998 to 2018.

We compared the inferred (ground) fam-groups in terms of the mutex group cover number, i.e., the minimum number of mutex groups needed to cover all facts, and the number of fam-groups that were not found by other methods. We compare also to the complete inference method described in Section 5.4 (Algorithm 5.1), denoted by `F`, because it shows how close we are to the best possible results.

Table 7.1 shows results for problems from both optimal and satisficing track where the computation of fam-groups by all methods finished within 5 minutes time and 8 GB memory limit and also the mutex group cover number was computed within 1 hour. To

[1]https://gitlab.com/danfis/cpddl

domain	**avg. rel. cover**			**num. mutex groups**			
	H	H+	F	H≻H+	H+≻H	F≻H	F≻H+
agricola (40)	0.70	**0.47**	**0.35**	0	40	627	587
airport (20)	0.73	**0.72**	**0.72**	0	54	1 525	1 471
barman (74)	0.88	**0.19**	0.19	0	1 640	1 640	0
cavediving (40)	0.73	0.73	**0.22**	0	40	1 272	1 232
cybersec (30)	0.94	**0.72**	**0.39**	26	162 528	1 566	1 372
depot (22)	0.12	0.12	0.12	0	0	222	222
freecell (80)	0.30	0.30	0.30	0	0	4 632	4 632
maintenance (25)	1.00	1.00	**0.99**	0	0	6	6
mprime (35)	0.10	0.10	0.10	0	0	3	3
mystery (30)	0.17	0.17	0.17	0	0	60	60
nomystery (40)	0.08	**0.05**	0.05	0	40	40	0
organic-synthesis (9)	1.00	**0.83**	**0.62**	0	47	1 415	1 368
parcprinter (70)	0.61	0.61	**0.24**	0	0	4 034	4 034
pegsol (70)	0.34	0.34	0.34	0	0	223	223
pipesworld (86)	0.58	0.58	**0.50**	0	0	939	939
rovers (60)	0.48	0.48	0.48	0	0	4	4
satellite (36)	0.49	0.49	**0.45**	0	0	252	252
scanalyzer (70)	0.18	0.18	0.18	0	0	172	172
snake (19)	0.74	0.74	**0.34**	0	0	1 553	1 553
sokoban (100)	0.20	0.20	0.20	0	0	8	8
spider (26)	0.43	0.43	**0.21**	0	818	2 151	1 333
tetris (32)	0.90	**0.04**	**0.03**	0	136	1 400	1 264
thoughtful (20)	0.55	0.55	**0.42**	0	0	1 023	1 023
tidybot (60)	0.98	**0.63**	**0.63**	0	429	656	227
tpp (25)	0.30	0.30	0.30	0	0	237	237
transport (140)	0.25	**0.05**	0.05	0	432	432	0
trucks (12)	0.86	**0.06**	0.06	0	180	1 092	912
woodworking (100)	0.58	0.58	**0.57**	2	1 151	1 906	757
caldera (40)	0.83	0.83	-	0	920	-	-
citycar (40)	0.71	**0.09**	-	0	201	-	-
flashfill (20)	0.21	**0.11**	-	0	61 964	-	-
settlers (40)	0.96	**0.95**	-	0	140	-	-
overall w/o CE (2493)	0.42	**0.35**	**0.32**	28	167 535	29 090	23 891
overall with CE (2673)	0.43	**0.36**	-	28	230 760	-	-

Table 7.1: Left: the comparison of average C/F (the smaller is the better), where C is the mutex group cover number and F is the number of facts. F is in bold if it is strictly better than the other two, and H+ is in bold if it is strictly better than H. Right: for every $A{\succ}B$, the number of fam-groups found by A that are not a subset of any fam-group found by B. "overall w/o CE" counts only problems without conditional effects, and "overall with CE" counts all problems: averages over all counted problems on left and sums on right. H is Helmert's algorithm; H+ is our improvement; F is a complete algorithm for all maximal fam-groups.

deal with different sizes of problems, mutex group cover numbers are divided by the number of facts for each problem and averaged within each domain. The domains with conditional effects (under the middle horizontal line) do not contain results for F, because it is not clear how to find a complete set of fam-groups without compiling conditional effects away. Only domains with some difference are shown. Since F must always dominate both H and H+, the numbers in the F column are in bold if they are strictly better than the other two, and the numbers for H+ are in bold if they are strictly better than H.

The improvement by H+ in agricola, tetris, tidybot, and citycar was due to multiple counted variables used in a single atom. The type refinement helped in nomystery, organic-synthesis, and transport, where some predicates formed mutex groups only for certain sub-types of their arguments. In airport, barman, trucks, settlers, and woodworking, the combination of the type refinement and the extension of proved fam-groups provided a richer set of mutex groups. In 20 domains both H and H+ found the complete set of fam-groups, and in barman, nomystery, and transport only H+ found a complete set of fam-groups.

H found some fam-groups not found by H+ in cybersec because of the limit on the number of invariant candidates, and in woodworking because of the init-weight restriction

domain	mutex		+ dead-end		+ h^2 fw/bw	
	H	H+	H	H+	H	H+
agricola (20)	0.00	0.00	0.00	0.00	64.99	**65.13**
airport⋆ (45)	0.00	0.00	8.17	8.17	76.20	76.20
barman⋆ (74)	15.42	15.42	15.42	**45.95**	25.50	**57.83**
blocks (35)	10.82	10.82	10.82	10.82	10.82	10.82
citycar⋆ (40)	0.00	0.00	1.61	1.61	1.61	1.61
depot (22)	6.44	6.44	6.44	6.44	22.44	22.44
flashfill (18)	0.10	0.10	0.10	0.10	7.06	7.06
floortile⋆ (70)	0.00	0.00	22.79	22.79	35.43	35.43
freecell (80)	0.01	0.01	0.01	0.01	0.06	0.06
organic-synthesis⋆ (7)	0.00	**11.44**	14.14	**23.11**	90.85	90.85
parcprinter⋆ (30)	0.00	0.00	40.83	40.83	70.00	70.00
parking (80)	3.09	3.09	3.09	3.09	6.19	6.19
pipesworld (100)	4.53	4.53	4.53	4.53	8.46	8.46
scanalyzer (30)	1.34	1.34	1.34	1.34	28.12	28.12
spider (15)	0.00	**3.47**	0.00	**3.47**	16.61	16.61
trucks⋆ (30)	0.00	0.00	0.00	**82.14**	19.09	**82.94**
woodworking⋆ (30)	0.00	0.00	10.08	10.08	51.41	51.41
overall from above (726)	3.31	**3.49**	8.35	**15.01**	24.03	**29.97**
overall (1987)	1.21	**1.28**	3.05	**5.48**	14.20	**16.37**

Table 7.2: The average percentage of removed operators, overall is the average over all problems. Column "mutex": lifted fam-groups used for detection of unreachable operators (their preconditions are mutex); "+dead-end": additional pruning of dead-end operators; "+h^2 fw/bw": additional pruning using h^2 in forward and backward with fam-groups used for the disambiguation. Domains in which dead-end detection pruned additional operators are starred.

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

The time spent in the inference was under a millisecond in 2159 out of 2673 tested problems for H+ (2304 for H), between one millisecond and one second in 485 (340) problems, and more than a second for the remaining 29 (29) problems from the caldera, cybersec, and flashfill domains. F was orders of magnitude slower (cf. Table 5.4 from Section 5.5.3).

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

As Table 7.1 shows (non-zero values in the F≻H+ column), H+ is not complete with respect to all maximal (ground) fam-groups. The reason is that the algorithm allows at most one atom of each predicate in lifted fam-groups, and all variables are restricted to the types defined in the input PDDL. So, for example, suppose we have three objects l_1, l_2, l_3 of a type `loc`, there is no sub-type of `loc`, and the corresponding ground problem has a maximal (ground) fam-group $\{\mathtt{at}(l_1), \mathtt{at}(l_2)\}$. This fam-group cannot be found by H+, because the invariant candidate $\mathtt{at}(c{:}\mathtt{loc})$ (with a counted variable c) would correspond to $\{\mathtt{at}(l_1), \mathtt{at}(l_2), \mathtt{at}(l_3)\}$. And even if there was a sub-type for each l_i, then the invariant candidate $\{\mathtt{at}(v_1{:}\mathtt{l}_1), \mathtt{at}(v_2{:}\mathtt{l}_2)\}$ (with fixed variables v_1, v_2) is not constructed by H+, because it contains two atoms of the same predicate.

Next, we compared the pruning power of H, H+ and h^2 heuristic in forward and backward direction (Alcázar & Torralba, 2015) with fam-groups from H and H+ used for disambiguation. Table 7.2 shows the results for problems where all variants finished within the 5 minutes time limit. The table lists only the domains in which at least one operator was pruned during grounding or there was a difference between h^2 with `H` and with `H+`.

The pruning of dead-end operators during grounding had effect in 7 domains (difference between "mutex" and "+dead-end" columns). Overall, using `H` for pruned grounding removes more than 3% operators and using `H+` almost 5.5%. Moreover, inferring a richer set of fam-groups using `H+` provides more pruning power even for h^2: about 2.1% more operators are removed.

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

For the optimal track, we used $A^\star$ with the LM-Cut (`lmc`) heuristic (Helmert & Domshlak, 2009), the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy and non-greedy bi-simulation shrink strategy (Helmert et al., 2014; Sievers et al., 2016), the potential (`pot-all`) heuristic optimized for all syntactic states (Seipp et al., 2015), and two non-portfolio winners of the last IPC 2018, Complementary1 (`comp1`) (Franco et al., 2018), and Complementary2 (`comp2`) (Franco et al., 2017, 2018). The only difference for `lmc` was that `H+` solved three more problems in barman and one less in transport. For `pot-all`, `H+` solved three less problems in spider and two more in tetris.

The most interesting results were found with `ms`, `comp1`, and `comp2` planners shown in Table 7.3. The table also shows the results with the original translator from the Fast Downward planner (`fd`) which implements `H`. However, there are some differences in the grounding process (e.g., handling of negative preconditions and conditional effects, deduplication of operators, or ordering of the unification steps), so we do not think `fd` is directly comparable to `H` and `H+`, because it does not measure just the difference between different sets of lifted fam-groups.

The most noticeable increase in coverage was found in the tidybot domain, where `H` is not able to find mutex groups describing position of objects (`base-pos`, `cart-pos`, and `object-pos` predicates) because it requires multiple counted variables in the corresponding atoms and therefore these facts are encoded as binary variables in FDR. `H+`, however, finds these mutex groups, which results in less and bigger variables, and therefore the abstraction heuristics generate more informed abstractions faster than with `H`.

All planners from Table 7.3 use some variant of abstraction heuristic, where we expected to see the most difference because they depend on the complexity and the number of inferred mutex groups. However, the implementation of `ms` and pattern databases in FD uses FDR variables derived from fam-groups instead of fam-groups directly. We think this approach possibly disregards some useful information from the overlapping fam-groups that could improve the heuristic estimates.

Consider the following example. Say we have two mutex groups $\{a, b\}$ and $\{a, c\}$ in the STRIPS planning task. The translation to FDR would translate these two mutex groups to two FDR variables, v_1 and v_2, with domains $\mathrm{dom}(v_1) = \{a, b, t_1\}$ and $\mathrm{dom}(v_2) = \{c, t_2\}$, where t_1 and t_2 stand for special "none-of-those" values meaning that neither a or b is set in the case of t_1, and that c is not set in the case of t_2. Now note that:

(i) t_2 is a single value that stands for both "a is set" (which is already represented in

domain	ms			comp1			comp2		
	`fd`	`H`	`H+`	`fd`	`H`	`H+`	`fd`	`H`	`H+`
agricola18 (20)	3	3	3	**9**	8	8	6	7	**8**
airport04 (50)	24	24	24	27	27	27	27	**28**	27
barman11 (20)	8	8	8	8	8	8	8	8	**10**
barman14 (14)	3	3	3	3	3	**6**	3	4	**5**
caldera18 (20)	12	12	12	11	13	**14**	12	13	**15**
childsnack14 (20)	0	0	0	0	0	0	**1**	0	**1**
citycar14 (20)	14	**16**	**16**	10	15	**16**	13	**16**	**16**
data-network18 (20)	12	12	12	13	**14**	**14**	11	**13**	12
depot02 (22)	8	8	8	7	7	7	**8**	7	**8**
mprime98 (35)	24	24	24	**23**	22	22	**24**	**24**	23
mystery98 (30)	17	17	17	15	15	15	**16**	15	**16**
openstacks06 (30)	7	7	7	10	**12**	11	**11**	**11**	10
parcprinter08 (30)	23	23	23	**24**	**24**	23	23	23	**24**
parcprinter11 (20)	18	18	18	**18**	17	**18**	18	18	18
spider18 (20)	6	6	6	**12**	11	11	11	**12**	11
tetris14 (17)	11	11	11	11	11	**12**	13	13	13
tidybot11 (20)	9	9	**18**	18	18	**19**	17	17	**20**
tidybot14 (20)	2	2	**13**	14	14	**19**	13	14	**19**
tpp06 (30)	**7**	6	6	12	**13**	**13**	**15**	**15**	14
transport08 (30)	11	11	11	**14**	12	**14**	**14**	12	**14**
transport11 (20)	6	6	6	**10**	9	**10**	**10**	8	**10**
transport14 (20)	7	7	7	**9**	8	**9**	**9**	8	**9**
trucks06 (30)	9	**10**	**10**	**14**	13	13	10	12	**13**
visitall11 (20)	16	16	16	12	12	12	17	**18**	17
Σ from above (578)	257	259	**279**	304	306	**321**	310	316	**333**
blocks00 (35)	**28**	21	21	30	**31**	**31**	30	**31**	**31**
floortile14 (20)	8	8	8	17	17	17	**20**	19	19
freecell00 (80)	**21**	20	20	**32**	27	27	**31**	29	29
nurikabe18 (20)	12	12	12	**12**	11	11	**11**	10	10
openstacks14 (20)	3	3	3	**14**	13	13	14	14	14
organic-synthesis18 (20)	7	**10**	**10**	7	**10**	**10**	7	**10**	**10**
petri-net-alignment18 (20)	7	7	7	19	**20**	**20**	19	19	19
pipesworld-notankage04 (50)	20	**22**	**22**	17	**18**	**18**	25	25	25
pipesworld-tankage04 (50)	15	**16**	**16**	16	**17**	**17**	18	**19**	**19**
rovers06 (40)	7	7	7	**14**	13	13	13	13	13
woodworking11 (20)	13	13	13	**20**	19	19	19	**20**	**20**
Σ (1 757)	893	893	**913**	1 054	1 054	**1 069**	1 087	1 095	**1 112**

Table 7.3: The number of solved problems in the optimal track for selected planners. `fd`: FD with the original translator. The top part lists domains where `H` and `H+` differ.

the variable v_1) and "neither a or c is set".

(ii) Creating a synchronized product of atomic projections to these two variables will result in abstract states such as $\{a, c\}$, $\{a, t_2\}$, $\{t_1, t_2\}$, or $\{b, t_2\}$. But $\{a, c\}$ is actually a mutex and we could infer that directly from the mutex groups, but not from the FDR variables. In $\{a, t_2\}$, t_2 is implied by a because a is a mutex with c. And in $\{t_1, t_2\}$ and $\{b, t_2\}$, t_2 actually represents "neither a or c is set".

All this information that could be useful for abstraction-based heuristics is lost in the FDR variables unless we use also the mutexes from the original mutex groups. However, it is an open question whether utilizing this information in, for example, the merge-and-shrink heuristic would require some modifications in merge or shrink strategies.

7.6 Summary

Any translator from PDDL to FDR must, at some point, infer a set of mutex groups in order to create FDR variables. The most commonly used translator (Helmert, 2009) infers mutex groups on a lifted (PDDL) level and then grounds them as it grounds the task into STRIPS. We proved that these lifted mutex groups are lifted fam-groups, i.e., when grounded, they belong to the "easier" subclass of mutex groups introduced in Section 5.1.

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

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

Chapter 8

Strengthening Potential Heuristics

The most common approach to solving classical planning problems is a heuristic search. In this chapter, we focus on the family of admissible heuristics called *potential heuristics* (Pommerening et al., 2015a) that assign a numeric potential to each fact and the resulting heuristic value for a given state is computed as a sum of the potentials of the facts in the state. Pommerening et al. (2015a) showed that potential heuristics produce the same heuristic value for a given state as the state equation heuristic (van den Briel et al., 2007; Bonet, 2013) if they are optimized for each state individually, but, in practice, the potentials for all facts are found only once before the search starts and then re-used during the search in a very fast evaluation of states.

Since their introduction, potential heuristics are continuously studied. Seipp et al. (2015) introduced several new optimization functions that provide ways to select different sets of potentials in order to increase the heuristic values. New complexity measure for classical planning tasks, called the correlation complexity, was introduced by Seipp et al. (2016). It is based on a study of a dimensionality of features of potential heuristics, i.e., extending potentials from single facts to sets of facts. Pommerening et al. (2016) provided a detailed description of how to construct potential heuristics for high-dimensional features.

We propose to use mutexes (and mutex groups as means to construct variables in finite domain representation) to improve potential heuristics in two ways. First, we relax the constraints describing potential heuristics by using mutexes and so-called disambiguation (Alcázar et al., 2013) to infer which facts cannot be part of the goal states or states where operators are applied. This leads to higher (or at worst the same) heuristic values for all optimization functions, because the optimizer becomes less restricted in the search for the best potentials. Second, we use mutexes in new optimization functions aiming at more accurate estimations of the number of reachable states and thus improving average heuristic values over all reachable states.

8.1 Background

We consider the finite domain representation (FDR) of planning tasks (Bäckström & Nebel, 1995) (Definition 3.2), where states are represented as assignments to variables with finite domains.

Definition 8.1. Let $\Pi^{\mathcal{V}}$ denote an FDR planning task. A **heuristic** $h : \mathcal{R}_{\Pi^{\mathcal{V}}} \mapsto \mathbb{R} \cup \{\infty\}$ estimates the cost of optimal s-plans. The **optimal heuristic** $h^\star(s)$ maps each reachable state s to the cost of the optimal s-plan or to ∞ if s is a dead-end state. A heuristic h is called

(a) **admissible** iff $h(s) \leq h^\star(s)$ for every reachable state $s \in \mathcal{R}_{\Pi^\mathcal{V}}$;

(b) **goal-aware** iff $h(s) \leq 0$ for every reachable goal state s; and

(c) **consistent** iff $h(s) \leq h(o[\![s]\!]) + \mathrm{c}(o)$ for all reachable states $s \in \mathcal{R}_{\Pi^\mathcal{V}}$ and operators $o \in \mathcal{O}$ applicable in s.

It is well-known that goal-aware and consistent heuristics are also admissible. Note that we define heuristics over the reachable states (instead of all states) because we intend to use heuristics in the (forward) heuristic search and because we want to use state invariants describing the reachable state space for an improvement of the heuristic values. Note also that we allow negative heuristic values as is usual in the literature related to potential heuristics, because it was shown it may be beneficial and there is no reason for considering only non-negative heuristic values (Pommerening et al., 2015a).

In this chapter, we work with sets of mutexes. So, to simplify the notation we introduce the following notion of a mutex-set.

Definition 8.2. Let $\Pi^\mathcal{V}$ denote a planning task with variables $\mathcal{V}$ and facts $\mathcal{F}$. A set of sets of facts $\mathcal{M} \subseteq 2^\mathcal{F}$ is called a **mutex-set** if all of the following hold:

(a) every $M \in \mathcal{M}$ is a mutex; and

(b) for every $M \in \mathcal{M}$ and every $f \in \mathcal{F}$ it holds that $M \cup \{f\} \in \mathcal{M}$; and

(c) for every variable $V \in \mathcal{V}$ and every pair of facts $f, f' \in \mathcal{F}_V$, $f \neq f'$, it holds that $\{f, f'\} \in \mathcal{M}$.

In other words, a mutex-set is an upper set of a set of mutexes and it always contains all mutexes that can be inferred directly from the variables of the FDR representation. This allows us to write, for example, $s \in \mathcal{M}$ for a state s and a mutex-set $\mathcal{M}$ when we want to say that s contains a subset of facts that is a mutex. But, of course, in practice we keep the inferred mutexes as a set without explicitly constructing all supersets.

8.2 Disambiguation

When dealing with backward search, also known as regression, Alcázar et al. (2013) showed that mutexes can be used for extending partial states p with a value of some variable V not defined in p ($V \notin \mathrm{vars}(p)$), if all but one value is a mutex with p. Alcázar & Torralba (2015) re-used this idea for the pruning technique based on a fixpoint computation of h^2 heuristic in both forward and backward direction. They found out that this process is actually essential for the backward h^2 heuristic to be able to infer any useful information. We borrow and extend this notion, called disambiguation, in the following way.

Definition 8.3. Let $\Pi^\mathcal{V}$ denote a planning task with facts $\mathcal{F}$ and variables $\mathcal{V}$, let $V \in \mathcal{V}$ denote a variable, and let p denote a partial state. A set of facts $F \subseteq \mathcal{F}_V$ is called a **disambiguation of** V **for** p if for every reachable state $s \in \mathcal{R}_{\Pi^\mathcal{V}}$ such that $p \subseteq s$ it holds that $F \cap s \neq \emptyset$ (i.e., $\langle V, s[V] \rangle \in F$).

Algorithm 8.1: Single-fact fixpoint disambiguation.

```
Input: A planning task $\Pi^{\mathcal{V}}$ with variables $\mathcal{V}$ and facts $\mathcal{F}$, a partial state $p$, and a
       mutex-set $\mathcal{M}$.
Output: A partial state $p$ extended with disambiguations of size one.
1 do
2     $p' \leftarrow p$;
3     for each $V \in \mathcal{V}$ do
4         $D_V \leftarrow \mathcal{F}_V \setminus \mathcal{M}_p$;
5         if $|D_V| = 1$ then $p \leftarrow p \cup D_V$ ;
6         if $|D_V| = 0$ then return "$p$ is a mutex" ;
7 while $p' \neq p$;
```

Clearly, every $\mathcal{F}_V$ is a disambiguation of V for all possible partial states. Moreover, it follows directly from the definition that if some $F \subseteq \mathcal{F}_V$ is a disambiguation of V for some partial state p then every $f \in \mathcal{F}_V \setminus F$ must be a mutex with p, i.e., every state s such that $\{f\} \cup p \subseteq s$ is unreachable. Therefore, if F is the empty set, then any state s such that $p \subseteq s$ is unreachable. So, as previously noted by Alcázar et al. (2013), we can use empty disambiguations to prune unreachable operators (if a precondition of an operator extends p), or to prove unsolvability of the planning task (if G extends p). Moreover, if the disambiguation of V consist of exactly one fact, then the partial state p can be safely extended with that fact, because it is the only value that can be assigned to the variable V in any reachable state $s \supseteq p$.

Definition 8.4. Given a partial state p and a mutex-set $\mathcal{M}$, we define a set $\mathcal{M}_p = \{f \mid f \in \mathcal{F}, p \cup \{f\} \in \mathcal{M}\}$ as the set of facts which $\mathcal{M}$ entails to be mutex with p.

Note that $\mathcal{F}_V \setminus \mathcal{M}_p$ is a disambiguation of V for p.

Algorithm 8.1 encapsulates a use of disambiguation in a simple fixpoint algorithm. On line 4, the domain of each variable is reduced by removing facts that are mutex with the partial state p. On line 5, p is extended if there is only one possible fact that can appear in a reachable state containing p. On line 6, the algorithm reports p as a mutex if all facts from the variable's domain are mutex with p.

We extend the notion of disambiguation described by Alcázar et al. (2013) and Alcázar & Torralba (2015) in that we keep also the sets containing more than one fact and we show in the next section how these sets can be used for strengthening potential heuristics.

Algorithm 8.2 shows an improved algorithm that computes disambiguations of all variables for a given partial state p. The algorithm keeps track of facts that cannot be part of a reachable state extending p (the set A). On line 7 the disambiguations are restricted by removing these facts, and on line 8 the set A is expanded by those facts that cannot be part of any reachable state extending any of $p \cup \{f\}$ for $f \in D_V$.

Theorem 8.5. *Algorithm 8.2 always produces a set of disambiguations of all variables for the given partial state p.*

Proof. The algorithm always terminates, because the sets D_V, for every $V \in \mathcal{V}$, can only decrease in size in each cycle (line 7), therefore there is a fixpoint.

Now we show that in every step, every D_V is a disambiguation of V for p. D_V is initialized with $\mathcal{F}_V$ (line 1) which is a disambiguation of V for p by definition. Then D_V is changed only on line 7 by removing the set A. So it suffices to show that at every

Algorithm 8.2: Multi-fact fixpoint disambiguation.

```
Input: A planning task $\Pi^{\mathcal{V}}$ with variables $\mathcal{V}$ and facts $\mathcal{F}$, a partial state $p$, and a
       mutex-set $\mathcal{M}$.
Output: A set of disambiguations $\mathfrak{D}$ of all variables $\mathcal{V}$ for $p$.
1  $D_V \leftarrow \mathcal{F}_V$ for every $V \in \mathcal{V}$;
2  $A \leftarrow \mathcal{M}_p$ ;                          // A set of facts that are mutex with p
3  do
4      change ← False;
5      for each $V \in \mathcal{V}$ do
6          if $D_V \setminus A \neq D_V$ then
7              $D_V \leftarrow D_V \setminus A$;
8              $A \leftarrow A \cup \bigcap_{f \in D_V} \mathcal{M}_{p \cup \{f\}}$;
9              change ← True;
10 while change;
11 $\mathfrak{D} \leftarrow \{D_V \mid V \in \mathcal{V}\}$;
```

point, A contains only the facts that cannot occur in any reachable state s extending p, i.e., $A \cap s = \emptyset$. This is true for the initialization of A (line 2). A is updated on line 8 by adding the set $X = \bigcap_{f \in D_V} \mathcal{M}_{p \cup \{f\}}$. Let $f' \in X$. By definition of X the fact f' cannot occur in any reachable state extending any of partial states $p \cup \{f\}$ for $f \in D_V$. Since D_V is a disambiguation of V for p, every reachable state s extending p extends $p \cup \{f\}$ for some $f \in D_V$. Therefore $s \cap X = \emptyset$. □

Note that the mutex-set $\mathcal{M}$ contains all mutex pairs from all variables so for every variable $V \in \text{vars}(p)$ defined in the partial state p, D_V is set to $\{\langle V, p[V] \rangle\}$ in the first cycle D_V is changed on line 7. Also note that if one disambiguation is found out to be empty, and therefore p is proved to be mutex, then all disambiguations are gradually also set to empty sets.

8.3 Potential Heuristics

Potential heuristics, introduced by Pommerening et al. (2015a), assign a numerical value to each fact, and the heuristic value for a state s is then simply a sum of the potentials of all facts in s.

Definition 8.6. Let $\Pi^{\mathcal{V}}$ denote a planning task with facts $\mathcal{F}$. A **potential function** is a function $\mathtt{P} : \mathcal{F} \mapsto \mathbb{R}$. A **potential heuristic** for $\mathtt{P}$ maps each state $s \in \mathcal{R}_{\Pi^{\mathcal{V}}}$ to the sum of potentials of facts in s, i.e., $h^{\mathtt{P}}(s) = \sum_{f \in s} \mathtt{P}(f)$.

Pommerening et al. (2015a) described a set of inequalities that are sufficient conditions for the potential heuristic to be admissible, which can be formulated as the following theorem.[1]

Theorem 8.7. *Let* $\Pi^{\mathcal{V}} = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ *denote a planning task,* $\mathtt{P}$ *a potential function, and for every operator* $o \in \mathcal{O}$*, let* $\text{pre}^\star(o) = \{\langle V, \text{pre}(o)[V] \rangle \mid V \in \text{vars}(\text{pre}(o)) \cap \text{vars}(\text{eff}(o))\}$

[1]The original formulation uses planning tasks in the so-called Transition Normal Form, but the general case is described in the technical report (Pommerening et al., 2015b).

and $\mathrm{vars}^\star(o) = \mathrm{vars}(\mathrm{eff}(o)) \setminus \mathrm{vars}(\mathrm{pre}(o))$. *If*

$$\sum_{f \in G} \mathtt{P}(f) + \sum_{V \in \mathcal{V} \setminus \mathrm{vars}(G)} \max_{f \in \mathcal{F}_V} \mathtt{P}(f) \leq 0 \tag{8.1}$$

and for every operator $o \in \mathcal{O}$ *it holds that*

$$\sum_{f \in \mathrm{pre}^\star(o)} \mathtt{P}(f) + \sum_{V \in \mathrm{vars}^\star(o)} \max_{f \in \mathcal{F}_V} \mathtt{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) \leq \mathrm{c}(o), \tag{8.2}$$

then the potential heuristic for $\mathtt{P}$ *is admissible.*

Equation (8.1) makes sure that the sum of potentials is goal-aware, and Equation (8.2) ensures consistency of the potential heuristic. The theorem, however, does not tell us how to actually choose the potential function. Pommerening et al. (2015a) proposed to formulate the inequalities as constraints of a linear program (LP) and then a solution for any optimization function results in an admissible potential heuristic. The selection of optimization functions is discussed in the next section.

For now, move your attention to the maxima over all facts of the undefined variables in Equation (8.1) and (8.2). For the goal Equation (8.1), we do not know how the reachable goal states actually look like, so we prepare for the worst case by using the maximum potential over all facts of each undefined variable. Similarly for the operator Equation (8.2), we do not fully know the reachable states where the operator is applicable. However, we have demonstrated in the previous section that mutexes can be used for narrowing down the unknown parts. So, we can use disambiguations to generalize Theorem 8.7 by the following theorem.

Theorem 8.8. *Let* $\Pi^{\mathcal{V}} = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ *denote a planning task with facts* $\mathcal{F}$, *and let* $\mathtt{P}$ *denote a potential function, and*

(i) for every variable $V \in \mathcal{V}$, *let* $G_V \subseteq \mathcal{F}_V$ *denote a disambiguation of* V *for* G *s.t.* $|G_V| \geq 1$, *and*

(ii) for every operator $o \in \mathcal{O}$ *and every variable* $V \in \mathrm{vars}(\mathrm{eff}(o))$, *let* $E^o_V \subseteq \mathcal{F}_V$ *denote a disambiguation of* V *for* $\mathrm{pre}(o)$ *s.t.* $|E^o_V| \geq 1$.

If

$$\sum_{V \in \mathcal{V}} \max_{f \in G_V} \mathtt{P}(f) \leq 0 \tag{8.3}$$

and for every operator $o \in \mathcal{O}$ *it holds that*

$$\sum_{V \in \mathrm{vars}(\mathrm{eff}(o))} \max_{f \in E^o_V} \mathtt{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) \leq \mathrm{c}(o), \tag{8.4}$$

then the potential heuristic for $\mathtt{P}$ *is admissible.*

Proof. To show that the potential heuristic is goal-aware, we need to prove that for every reachable goal state s_G it holds that $\sum_{f \in s_G} \mathtt{P}(f) \leq 0$. Let $f = \langle V, v \rangle \in s_G$. From (i), we have $f \in G_V$. Since for every G_V and every $f \in G_V$ it holds that $\mathtt{P}(f) \leq \max_{f' \in G_V} \mathtt{P}(f')$, then from Equation (8.3) it follows that $\sum_{f \in s_G} \mathtt{P}(f) \leq \sum_{V \in \mathcal{V}} \max_{f \in G_V} \mathtt{P}(f) \leq 0$.

To show consistency, we need to prove that for every operator $o \in \mathcal{O}$ and every reachable state $s \in \mathcal{R}_{\Pi^\mathcal{V}}$ such that $\mathrm{pre}(o) \subseteq s$ it holds that $\sum_{f \in s} \mathtt{P}(f) - \sum_{f \in o[\![s]\!]} \mathtt{P}(f) \leq \mathrm{c}(o)$. Let $t = (s \cap o[\![s]\!]) \setminus \mathrm{eff}(o)$ be the part of s that is not affected by the operator o. Then clearly, $\sum_{f \in s} \mathtt{P}(f) - \sum_{f \in o[\![s]\!]} \mathtt{P}(f) = \sum_{f \in s \setminus t} \mathtt{P}(f) - \sum_{f \in o[\![s]\!] \setminus t} \mathtt{P}(f) = \sum_{f \in s \setminus t} \mathtt{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f)$, because $o[\![s]\!] \setminus t = \mathrm{eff}(o)$. Furthermore, since $\mathrm{vars}(s \setminus t) = \mathrm{vars}(\mathrm{eff}(o))$, it follows that every $f = \langle V, v \rangle \in s \setminus t$ belongs to E^o_V and consequently $\mathtt{P}(f) \leq \max_{f' \in E^o_V} \mathtt{P}(f')$. Therefore (by Equation (8.4)) $\sum_{f \in s \setminus t} \mathtt{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) \leq \sum_{V \in \mathrm{vars}(\mathrm{eff}(o))} \max_{f \in E^o_V} \mathtt{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) \leq \mathrm{c}(o)$. So $h^\mathtt{P}$ is goal-aware and consistent and therefore admissible. □

Note that the requirement on the non-empty disambiguations in (i) and (ii) is just to simplify the theorem, because for every empty disambiguation we could either remove the corresponding operator (if E^o_V is empty), or report the planning task unsolvable (if G_V is empty), as we already explained.

Clearly, Equation (8.3) generalizes Equation (8.1) because for every $V \in \mathrm{vars}(G)$ the singleton $\{\langle V, G[V] \rangle\}$ is a disambiguation of V for G, and for every $V \in \mathcal{V} \setminus \mathrm{vars}(G)$ the set $\mathcal{F}_V$ is a disambiguation of V for G. Thus Theorem 8.8 allows to use proper subsets of $\mathcal{F}_V$ for the variables undefined in G. Similarly, Equation (8.4) generalizes Equation (8.2) because we can use the same reasoning for the preconditions of operators.

8.3.1 Transition Normal Form

Originally, potential heuristics were formulated for planning tasks in the so-called Transition Normal Form (TNF) (Pommerening & Helmert, 2015). A planning task is in TNF if the goal is fully defined ($\mathrm{vars}(G) = \mathcal{V}$) and $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every operator o. Any planning task $\Pi^\mathcal{V} = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ can be compiled into TNF as follows:

- Add a fresh value U to the domain of every variable.
- For every variable $V \in \mathcal{V}$ and every fact $f \in \mathcal{F}_V$, $f \neq \langle V, U \rangle$, add a new *forgetting* operator o_f with $\mathrm{pre}(o_f) = \{f\}$ and $\mathrm{eff}(o_f) = \{\langle V, U \rangle\}$ and the cost $\mathrm{c}(o_f) = 0$.
- For every operator $o \in \mathcal{O}$ and every variable $V \in \mathcal{V}$:
  - If $V \in \mathrm{vars}(\mathrm{pre}(o))$ and $V \notin \mathrm{vars}(\mathrm{eff}(o))$, then add $\langle V, \mathrm{pre}(o)[V] \rangle$ to $\mathrm{eff}(o)$.
  - If $V \in \mathrm{vars}(\mathrm{eff}(o))$ and $V \notin \mathrm{vars}(\mathrm{pre}(o))$, then add $\langle V, U \rangle$ to $\mathrm{pre}(o)$.
- For every $V \in \mathcal{V} \setminus \mathrm{vars}(G)$ add $\langle V, U \rangle$ to G.

There is a clear correspondence between the compilation into TNF and the LP formulation of Equation (8.1) and (8.2) from Theorem 8.7 (Pommerening & Helmert, 2015). When translating inequalities (8.1) and (8.2) to the LP constraints, we (a) create the LP variable X_f for every fact $f \in \mathcal{F}$ (holding the potential $\mathtt{P}(f)$), and (b) to deal with the maxima, we create another auxiliary LP variable M_V for every variable $V \in \mathcal{V}$ and add the constraint $X_f \leq M_V$ for every $f \in \mathcal{F}_V$. Then we can rewrite Equation (8.1) to the constraint

$$\sum_{f \in G} X_f + \sum_{V \in \mathcal{V} \setminus \mathrm{vars}(G)} M_V \leq 0, \tag{8.5}$$

and similarly Equation (8.2) to constraints

$$\sum_{f \in \mathrm{pre}^\star(o)} X_f + \sum_{V \in \mathrm{vars}^\star(o)} M_V - \sum_{f \in \mathrm{eff}(o)} X_f \leq \mathrm{c}(o), \tag{8.6}$$

for every operator $o \in \mathcal{O}$, and look for the maximization over some optimization function. Compare Equation (8.5) and Equation (8.6) to the constraints resulting from the planning task compiled into TNF and you will find that we get exactly the same constraints where LP variables M_V correspond to the special value U added to every variable in TNF.

The generalization of Theorem 8.7 via disambiguations transposes also to the TNF. Instead of creating a single fresh value U for every variable, we create fresh values U_{G_V} and $U_{E_V^o}$ for every disambiguation G_V and E_V^o, respectively. Then we use these values in the same way U values are used in the original TNF formulation. There will be forgetting operators for every U_{G_V} and $U_{E_V^o}$ that go over the facts in their respective disambiguations rather than over all values from the corresponding domain. Instead of adding $\langle V, U\rangle$ into operators' preconditions, we add $\langle V, U_{E_V^o}\rangle$. And instead of adding $\langle V, U\rangle$ into the goal, we add $\langle V, U_{G_V}\rangle$.

Pommerening & Helmert (2015) showed that the compilation to TNF can produce a planning task twice the size of the original one, in the worst case. With disambiguations, the compilation to TNF can grow even more, but it is still polynomially bounded. Although we choose disambiguations from the powerset $2^{\mathcal{F}}$, the actual number of disambiguations is limited by the number of operators and the size of their preconditions. So, the number of U_{G_V} values can be at most $|\mathcal{V}|$, and the number of $U_{E_V^o}$ cannot be more than $|\mathcal{O}| \cdot |\mathcal{V}|$. The maximum number of forgetting operators then corresponds to these limits.

However, the resulting representation can also be smaller than the original TNF, because the disambiguations of size one can get rid of U values completely, and the disambiguations that are proper subsets of the corresponding $\mathcal{F}_V$ produce fewer forgetting operators. In fact, the experimental evaluation on the domains from International Planning Competitions (IPCs) shows that the representation with disambiguations is never bigger than without disambiguations.

8.4 Optimization Functions

When Pommerening et al. (2015a) introduced potential heuristics, they used the optimization function for the initial state

$$\mathrm{opt}_I = \sum_{f \in I} \mathtt{P}(f). \tag{8.7}$$

Maximization of opt_I subject to the constraints from Theorem 8.8 (or Theorem 8.7) yields the highest possible heuristic value for the initial state. However, maximization of opt_I does not provide an incentive for optimizing potentials of the facts that do not appear in the initial state (at least not directly). Of course, one could recompute potentials for each state reached during the search. That would always provide the best possible heuristic value but it would also be too costly from the computational point of view.

Seipp et al. (2015) studied different optimization functions. One of their main contributions is the "automatic diversification" algorithm for finding an ensemble of potential heuristics constructed from a set of states sampled by random walks. We will experimentally evaluate the effect of disambiguations on this variant. Another contribution was the introduction of a family of optimization functions aiming at maximizing the average heuristic value over all reachable states. And this is the obvious place where mutexes can be utilized.

8.4.1 All States Potentials

The perfect optimization function of which maximization yields the maximum average heuristic value over all reachable states is the weighted sum of the potentials over all reachable states:

$$\mathrm{opt}_{\mathcal{R}_{\Pi^{\mathcal{V}}}} = \frac{1}{|\mathcal{R}_{\Pi^{\mathcal{V}}}|} \sum_{s \in \mathcal{R}_{\Pi^{\mathcal{V}}}} \sum_{f \in s} \mathtt{P}(f). \tag{8.8}$$

By reordering summands in Equation (8.8), $\mathrm{opt}_{\mathcal{R}_{\Pi^{\mathcal{V}}}}$ can be written as $\sum_{f \in \mathcal{F}} \alpha_f \mathtt{P}(f)$ where the coefficient α_f is just the probability that a randomly chosen reachable state contains f. Consequently, by maximizing $\mathrm{opt}_{\mathcal{R}_{\Pi^{\mathcal{V}}}}$ we look for potentials such that the corresponding potential heuristic maximizes its expected value.

Listing all reachable states is, obviously, infeasible, and so is uniform sampling of reachable states. So, as an approximation, Seipp et al. (2015) proposed to adopt the approach of Haslum et al. (2007) and sample the states $S \subseteq \mathcal{R}_{\Pi^{\mathcal{V}}}$ by random walks starting from the initial state with a binomially distributed length of the walks centered around the double of the maximum heuristic value for the initial state, leading to the following optimization function:

$$\mathrm{opt}_{\hat{S}} = \frac{1}{|S|} \sum_{s \in S} \sum_{f \in s} \mathtt{P}(f). \tag{8.9}$$

Another proposed option was to count all syntactic states (i.e., all possible assignments to variables):

$$\mathrm{opt}_{\mathcal{S}} = \sum_{\langle V,v \rangle \in \mathcal{F}} \frac{1}{|\mathrm{dom}(V)|} \mathtt{P}(\langle V, v \rangle). \tag{8.10}$$

This approach assumes the uniform distribution of the values within their respective domains over all reachable states which is $1/|\mathrm{dom}(V)|$ for every fact $\langle V, v \rangle$.

We extend the idea of using all syntactic states by taking mutexes into account. Suppose we want to estimate the number of states containing a fact $f = \langle V, v \rangle \in \mathcal{F}$. The simplest estimate of the number of these (syntactic) states is to compute a product of sizes of all variables' domains except V, because the variable V is assumed to be already set to v: $\prod_{V' \in \mathcal{V} \setminus \{V\}} |\mathrm{dom}(V')|$. This estimate is actually an upper bound on the true number of reachable states containing f.

However, if we take mutexes into account, we could remove the facts that are mutex with f from all domains and compute the product of sizes of these reduced domains. The resulting estimate would necessarily be lower than (or equal to) the previous one. It would be an upper bound too, because we are removing only the facts that are certainly not part of the reachable states containing f. Therefore, we certainly get a better estimate. Moreover, we can extend this idea to partial states, i.e., instead of asking how many reachable states contain a single fact f, we can ask how many reachable states extend a partial state p.

For a given number $1 \leq k \leq |\mathcal{V}|$ and a partial state t we define a set $\mathcal{P}_k^t$ as the set of all partial states of size k extending t. Further recall that for a mutex-set $\mathcal{M}$ and a partial state p we defined the set $\mathcal{M}_p = \{f \mid f \in \mathcal{F}, p \cup \{f\} \in \mathcal{M}\}$.

Now we can estimate the number of reachable states extending a partial state p using a given mutex-set $\mathcal{M}$ by the product $\prod_{V \in \mathcal{V}} |\mathcal{F}_V \setminus \mathcal{M}_p|$. Note that if p is a mutex ($p \in \mathcal{M}$), then $\mathcal{M}_p$ contains all facts and the product is zero. Also, since $\mathcal{M}$ always contains mutex pairs from all variables, then $|\mathcal{F}_V \setminus \mathcal{M}_p| = 1$ for all $V \in \mathrm{vars}(p)$ if $p \notin \mathcal{M}$.

With all building blocks in place, we can define

$$\mathcal{C}^k_f(\mathcal{M}) = \sum_{p \in \mathcal{P}^{\{f\}}_k} \prod_{V \in \mathcal{V}} |\mathcal{F}_V \setminus \mathcal{M}_p| \tag{8.11}$$

as an estimation of the number of reachable states containing the fact f while considering mutex-set $\mathcal{M}$ and all partial states of size k. The corresponding optimization function is:

$$\mathrm{opt}^k_{\mathcal{M}} = \sum_{f=\langle V,v\rangle \in \mathcal{F}} \frac{\mathcal{C}^k_f(\mathcal{M})}{\sum_{f' \in \mathcal{F}_V} \mathcal{C}^k_{f'}(\mathcal{M})} \mathtt{P}(f). \tag{8.12}$$

That is, we choose a number $k \geq 1$ and for every fact f and every partial state p of size k containing f, we compute the estimation of the number of reachable states extending p (the inner product in Equation (8.11)). To get the estimate for a single fact f, we sum over the estimates for partial states p containing f (this is the number $\mathcal{C}^k_f(\mathcal{M})$). Finally, for every variable $V \in \mathcal{V}$ we normalize the collection of $\mathcal{C}^k_f(\mathcal{M})$ for $f \in \mathcal{F}_V$ so that it forms a probability distribution estimating the actual probability that a fact $f \in \mathcal{F}_V$ appears in a randomly chosen reachable state. In other words, instead of using the uniform distribution as in Equation (8.10), i.e., $1/|\mathrm{dom}(V)|$, we use mutexes and estimate the number of states step-by-step for all partial states of size k and then sum these counts to the final estimation.

8.4.2 Conditioned Ensemble of All States Potentials

Averaging sampled states $S \subseteq \mathcal{R}_{\Pi^{\mathcal{V}}}$, as in $\mathrm{opt}_{\hat{S}}$, is not the only way S can be used for a construction of a potential heuristic. Seipp et al. (2015) proposed to use an ensemble of potential heuristics, for example one potential heuristic per state from S, and then use the maximum heuristic value from all potential heuristic as the heuristic value for the given state.

The optimization for all reachable states $\mathrm{opt}_{\mathcal{R}_{\Pi^{\mathcal{V}}}}$ tackles the problem of finding the best potentials by maximizing the expected value of the sum of potentials for a randomly chosen reachable state. However, if we somehow divide the whole reachable state space into sets of states $S_1 \cup \ldots \cup S_n = \mathcal{R}_{\Pi^{\mathcal{V}}}$, then averaging over S_i with $\mathrm{opt}_{\hat{S}_i}$ would give us better heuristic values (at average) for each set S_i. Then we could use the ensemble of potential heuristics optimized for $\mathrm{opt}_{\hat{S}_i}$ for all $i = \{1, \ldots, n\}$ and the maximum of heuristic values for a given state over all these heuristics should give us a better resulting heuristic value. Intuitively, we can see this approach as being halfway between $\mathrm{opt}_{\mathcal{R}_{\Pi^{\mathcal{V}}}}$, and computing potentials for each individual state. Unfortunately, we do not know how to select the sets S_i or how to sample them efficiently. We can, however, re-use the approach to $\mathrm{opt}^k_{\mathcal{M}}$ in constructing a similar ensemble.

Taking the idea of using mutexes one step further, we can optimize for the maximum average potentials over the states extending a partial state t. In other words, we can fix the partial state t as a sort of selector for states and then use the same idea as for $\mathrm{opt}^k_{\mathcal{M}}$ except that we count only the states extending t. We start with a slight modification of Equation (8.11) by restricting the count to a given partial state t:

$$\mathcal{K}^k_f(\mathcal{M}, t) = \sum_{p \in \mathcal{P}^{t \cup \{f\}}_{|t|+k}} \prod_{V \in \mathcal{V}} |\mathcal{F}_V \setminus \mathcal{M}_p|\,. \tag{8.13}$$

In words, $\mathcal{K}^k_f$ differs from $\mathcal{C}^k_f$ in that $\mathcal{K}^k_f$ takes the partial states p of size $|t| + k$ extending $t \cup \{f\}$ instead of the partial states of size k containing f. We also tacitly assume that $t \cup \{f\}$ is a partial state, i.e., either $f \in t$ or the variable V corresponding to the fact f does not belong to $\mathrm{vars}(t)$. For the cases where $V \in \mathrm{vars}(t)$, we define $\mathcal{K}^k_f(\mathcal{M}, t) = 0$.

The optimization function for a fixed partial state t is a small modification of Equation (8.12) where we replace $\mathcal{C}^k_f$ with $\mathcal{K}^k_f$:

$$\mathrm{opt}^{t,k}_{\mathcal{M}} = \sum_{f=\langle V,v\rangle \in \mathcal{F}} \frac{\mathcal{K}^k_f(\mathcal{M}, t)}{\sum_{f' \in \mathcal{F}_V} \mathcal{K}^k_{f'}(\mathcal{M}, t)} \mathtt{P}(f). \tag{8.14}$$

The remaining question is how to select the partial states t on which to condition potential heuristics in the ensemble. Here, we evaluate only uniformly randomly sampled partial states of size 1 and 2. However, the question left for future research is whether we can use some sort of structural information, such as causal graphs or some kind of a relation between mutexes, for the selection of the best possible sets.

8.4.3 Adding Constraint on Initial State

As already mentioned, the disadvantage of optimizing for the initial state (opt_I) is that the optimization function does not provide an incentive to optimize the potentials that are not part of the initial state. But it should provide good heuristic values in the vicinity of the initial state, assuming the operators change only few facts at a time. Conversely, the optimization for all states, in all variants, including $\mathrm{opt}_{\mathcal{R}_\Pi \mathcal{V}}$, does not target any particular state and the resulting heuristic values highly depend on what the reachable state space actually looks like. For example, a huge number of goal states can unintentionally decrease the average heuristic values even though the goal-awareness of potential heuristics is explicitly enforced by a constraint and we actually want to push the heuristic values higher for all states but the goal states.

We can, however, overcome this behaviour at least partially by combining the optimization for the initial state and for all states. Let $h^{\mathtt{P}}_I$ denote the potential heuristic optimized for the initial state. Then using an optimization for all states (any discussed variant) and imposing the additional constraint

$$\sum_{f \in I} \mathtt{P}(f) = h^{\mathtt{P}}_I(I) \tag{8.15}$$

will force the optimizer to find potentials that will produce a high heuristic value for the initial state while maximizing average heuristic values for states containing facts that are not part of the initial state. So, during the search, we should get more accurate heuristic values from the beginning of the search and as we get farther from the initial state and closer to the goal, the potentials optimized for the average case should take over.

This, of course, requires to compute the potentials twice. The first time for the heuristic value $h^{\mathtt{P}}_I(I)$. And the second time for the all states potentials using Equation (8.15).

8.5 Experimental Evaluation

The evaluated methods were all implemented[2] in C and experimentally evaluated with the Fast Downward planner (Helmert, 2006) on a cluster of computing nodes with Intel Xeon

[2]https://gitlab.com/danfis/cpddl, branch icaps20-potentials

Scalable Gold 6146 processors and CPLEX solver v12.9. The time and memory limits were set to 30 minutes and 8 GB, respectively. Operators and facts are pruned with the h^2 heuristic in forward and backward direction (Alcázar & Torralba, 2015), and the translation from PDDL to FDR uses the inference of mutex groups proposed in Chapter 7. However, for the methods that require mutexes, we computed them again with the h^2 heuristic because we can use only the forward direction for mutexes as per Definition 4.1 and we wanted to account for the increased computational demand for these methods. We used all planning domains from International Planning Competitions (IPCs) from 1998 to 2018 excluding the ones containing conditional effects after translation (leaving 65 domains).

We refer to the compared variants of potential heuristics as follows:

- $\mathtt{N}$: the variant without disambiguation,
- $\mathtt{D}$: the multi-fact disambiguation (Algorithm 8.2),
- $\mathtt{D}_1$: the single-fact disambiguation (Algorithm 8.1),
- $\mathtt{Init}$: the optimization for the initial state (opt_I),
- $\mathtt{All}$: the optimization for all syntactic states ($\mathrm{opt}_{\mathcal{S}}$),
- $\mathtt{MaxIA}$: the maximization over $\mathtt{Init}$ and $\mathtt{All}$,
- $\mathtt{Div}_n$: the diversification algorithm with n samples,
- $\hat{\mathtt{S}}_n$: the optimization for n randomly sampled states ($\mathrm{opt}_{\hat{\mathcal{S}}}$),
- $\mathtt{S}_n$: the maximization over n potential heuristics each optimized for a randomly sampled state,
- $\mathtt{M}^k$: the optimization for all states considering inferred mutexes ($\mathrm{opt}^k_{\mathcal{M}}$),
- $\mathtt{K}^k_n$: the maximization over n potential heuristics each optimized for all states considering mutexes and conditioned on a randomly sampled fact f ($\mathrm{opt}^{\{f\},k}_{\mathcal{M}}$).
- $\mathtt{L}^k_n$: the same as $\mathtt{K}^k_n$ except partial states $\{f_1, f_2\}$ of size two are sampled ($\mathrm{opt}^{\{f_1,f_2\},k}_{\mathcal{M}}$).

If the additional constraint on the initial state was used (Section 8.4.3, Equation (8.15)), we append $+\mathtt{I}$. We fixed seeds for random number generators in order to get comparable results for the variants that use sampling. $\mathtt{Div}_n$ and $\hat{\mathtt{S}}_n$ are evaluated for $n = 1000$ to compare them to the results of Seipp et al. (2015).

In Figure 8.1, we show scatter plots of the heuristic values for the initial state when optimized for the initial state with and without disambiguation. The plots clearly show the advantage of utilizing multi-fact disambiguation in comparison to both single-fact disambiguation (right) and no disambiguation (left).

We tried to fit as many results comparing the use of disambiguations as possible into Table 8.1 to show the positive effect of disambiguations on the number of solved tasks. (We added the row with the sum without the freecell domain because this domain contains considerably more planning tasks than other domains which may skew the overall results.) Disambiguations decrease the coverage only in a few domains, but never overall. The most significant decrease can be found in the blocks domain for $\mathtt{Init}$ and freecell00 for $\mathtt{All}$,

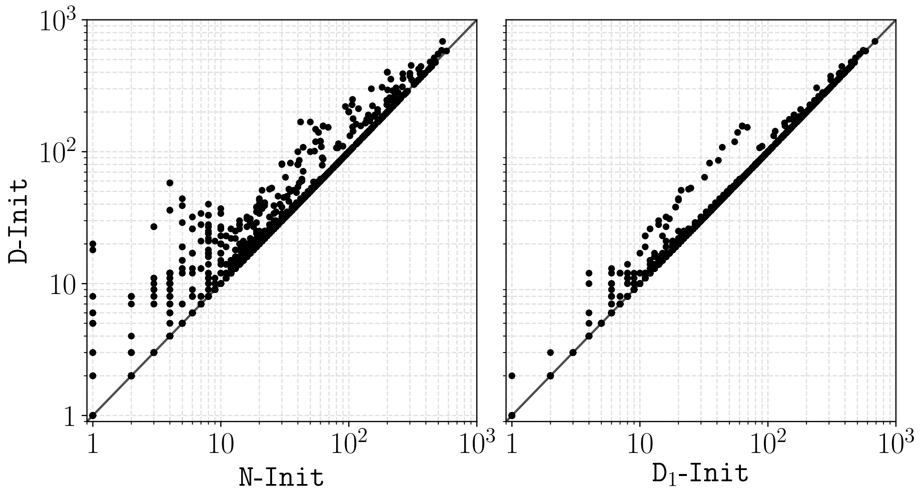

Figure 8.1: Heuristic values for the initial state with the zero heuristic values filtered out and the heuristic values in the parcprinter domain scaled by 10^{-4} to keep the plots compact.

but otherwise the decrease is just one or two fewer problem solved. Overall, the increase in coverage due to the (multi-fact) disambiguation ranges from 1.6% for $\mathtt{M}^2$ to 3.8% for `All+I` and it is two or more percent for all methods proposed by Seipp et al. (2015) (similar results are obtained also if freecell00 is not counted).

The only difference between $\mathtt{M}^1$ and $\mathtt{M}^2$ is in the parcprinter domains (in favor of $\mathtt{M}^1$) which suggests that increasing k increases the computational intensity but does not provide much more accurate estimates of the number of reachable states. Overall, both $\mathtt{M}^1$ and $\mathtt{M}^2$ have higher coverage than `All` (with or without disambiguations), but most of the difference is due to the freecell domain. If the freecell domain is filtered out, `All` solves more tasks. The average from a sample of 1000 states ($\hat{\mathtt{S}}_{1\mathrm{k}}$) also seems to provide better results (both overall and in per-domain comparison).

However, constructing ensembles of 100 or 50 potential heuristics with $\mathtt{K}^1_{100}$ and $\mathtt{K}^1_{50}$ results in a higher coverage than `All`, $\mathtt{M}^1$, $\mathtt{M}^2$, and $\hat{\mathtt{S}}_{1\mathrm{k}}$, which all optimize for the average state. The coverage is also higher than $\mathtt{Div}_{1\mathrm{k}}$ and $\mathtt{S}_{100}$ that maximize over an ensemble of potential heuristics as $\mathtt{K}^k_n$ does. We should note here that $\mathtt{K}^k_n$ conceptually sits between optimization for the average state and maximization over a sample of states, because $\mathtt{K}^k_n$ tries to do a little bit of both.

Table 8.2 compares the sampling based methods (with disambiguations) for different numbers of samples. The more samples are used in the ensemble the longer it takes to evaluate a state during the search and therefore the advantage of using potential heuristics diminishes. The sweet spot for $\mathtt{S}_n$ seems to be at 100 samples and it is at 50 samples for the methods using mutexes. It also seems that increasing k ($\mathtt{D\text{-}K}^1_n$ vs. $\mathtt{D\text{-}K}^2_n$, and $\mathtt{D\text{-}L}^1_n$ vs. $\mathtt{D\text{-}L}^2_n$) does not provide better heuristic values, and the same holds for increasing the size of sampled partial states ($\mathtt{D\text{-}K}^1_n$ vs. $\mathtt{D\text{-}L}^1_n$ and $\mathtt{D\text{-}L}^2_n$).

The best results were achieved with adding the constraint on the heuristic value of the initial state (Section 8.4.3, Equation (8.15)) in combination with disambiguations. In Table 8.1, for the "+I" columns, we added (a) "$\circ$" to indicate that the coverage in the respective domain is higher than for the corresponding variant without the constraint (without +I); (b) "+" to indicate a higher coverage than the `Init` variant; (c) "$\oplus$" if both of the previous cases hold at the same time; and (d) "$-$" to indicate that the coverage is smaller than either of the variants. There are very few domains in which the coverage

domain	Init			All			M^1		M^2		Div_{1k}		$\hat{S}_{1k}$		S_{100}		K^1_{100}		K^1_{50}	MaxIA		All+I		$\hat{S}_{1k}$+I		M^1+I	
	N	D_1	D	N	D_1	D	N	D	N	D	N	D	N	D	N	D	N	D	D	N	D	N	D	N	D	N	D
agricola18 (20)	1	1	**3**	1	1	**3**	1	1	1	1	1	**3**	1	**3**	1	**3**	2	**3**	**3**	1	**3**	1	**3**	1	**3**	1	◦**3**
airport04 (50)	29	32	32	30	31	31	30	31	30	31	30	31	30	28	27	30	27	29	30	30	33	+30	⊕**36**	+30	⊕33	+30	◦32
blocks00 (35)	**28**	21	21	**28**	21	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	21	**28**	**28**	**28**	**28**	**28**	**28**	+**28**	**28**	+**28**	**28**	+**28**
caldera18 (20)	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	10	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
depot02 (22)	7	9	8	11	11	11	11	11	11	11	10	11	11	11	11	10	11	**12**	**12**	11	11	+11	+11	+11	+11	+11	+11
freecell00 (80)	67	71	71	41	36	36	58	58	58	58	**73**	**73**	**73**	72	68	71	62	64	65	67	71	⊕72	⊕72	+**73**	⊕**73**	⊕69	◦69
ged14 (20)	15	15	15	15	15	15	15	15	15	15	17	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	15	15	15	15	+**19**	+**19**	15	15
hiking14 (20)	13	13	13	**14**	**14**	**14**	**14**	**14**	**14**	**14**	11	11	**14**	**14**	**14**	**14**	13	13	13	**14**	**14**	+**14**	+**14**	+**14**	+**14**	+**14**	+**14**
logistics00 (28)	11	11	11	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	16	16	**19**	**19**	**19**	**19**	**19**	+**19**	+**19**	+**19**	+**19**	+**19**	+**19**
logistics98 (35)	2	2	2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	3	3	**5**	**5**	**5**	**5**	**5**	+**5**	+**5**	+**5**	+**5**	+**5**	+**5**
mystery98 (30)	17	17	**18**	17	17	**18**	17	**18**	17	**18**	17	**18**	17	**18**	17	**18**	17	**18**	**18**	17	**18**	17	**18**	17	**18**	17	**18**
nomystery11 (20)	10	10	10	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	10	10	**14**	**14**	**14**	**14**	**14**	+**14**	+**14**	+**14**	+**14**	+**14**	+**14**
openstacks06 (30)	7	**14**	13	7	**14**	**14**	7	7	7	7	7	7	7	7	7	7	7	10	10	7	**14**	7	−13	7	◦13	7	◦13
openstacks08 (30)	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	21	**23**	**23**	**23**	22	**23**	**23**	**23**	**23**	**23**	**23**	−21	**23**	**23**	**23**
openstacks11 (20)	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	16	**18**	**18**	**18**	17	**18**	**18**	**18**	**18**	**18**	**18**	−16	**18**	**18**	**18**
openstacks14 (20)	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	0	**3**	**3**	**3**	1	2	2	**3**	**3**	**3**	**3**	−0	**3**	**3**	**3**
parcprinter08 (30)	27	27	27	21	20	20	24	25	23	23	17	17	17	16	16	16	**28**	**28**	**28**	**28**	27	◦25	⊕**28**	◦22	◦22	◦25	−25
parcprinter11 (20)	**20**	**20**	**20**	16	15	15	17	18	17	17	13	13	13	12	12	12	**20**	**20**	**20**	**20**	**20**	◦**20**	◦**20**	◦17	◦17	−17	−17
parking11 (20)	7	6	6	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	5	**8**	**8**	**8**	**8**	+**8**	+**8**	+**8**	+**8**	+**8**	+**8**
parking14 (20)	7	6	6	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	5	**8**	**8**	**8**	**8**	+**8**	+**8**	+**8**	+**8**	+**8**	+**8**
pegsol08 (30)	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	**30**	**30**	**30**	29	29	29	29	29	29	29	29
pegsol11 (20)	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	**20**	**20**	**20**	19	19	19	19	19	19	19	19
petr-net-align18 (20)	**13**	**13**	**13**	7	9	9	7	7	7	7	11	11	11	11	9	9	9	7	11	**13**	**13**	◦12	◦**13**	−11	−11	◦**13**	◦**13**
pipesworld-notank04 (50)	26	26	25	21	25	25	20	24	20	24	25	26	27	**30**	26	27	19	24	25	26	28	◦26	⊕**30**	+27	+29	◦26	⊕29
pipesworld-tank04 (50)	18	17	17	16	17	16	17	17	17	17	14	18	19	19	20	**21**	18	20	20	18	17	⊕19	⊕19	⊕20	⊕20	⊕19	⊕19
rovers06 (40)	6	6	6	7	**8**	**8**	7	**8**	7	**8**	7	**8**	7	**8**	6	6	7	**8**	**8**	7	**8**	+7	+**8**	+7	+**8**	+7	+**8**
scanalyzer08 (30)	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	12	12	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
scanalyzer11 (20)	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	9	**10**	**10**	9	9	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
snake18 (20)	13	15	15	14	14	14	12	12	12	12	10	10	13	**17**	12	15	11	11	12	14	15	⊕15	◦15	13	−15	⊕15	◦15
sokoban08 (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	29	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
spider18 (20)	14	13	15	12	14	14	13	13	13	13	12	14	15	**16**	13	14	13	13	13	15	15	⊕**16**	◦15	+15	−15	◦14	◦15
storage06 (30)	**16**	**16**	15	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	+**16**	**16**	+**16**	**16**	+**16**
termes18 (20)	12	12	12	**13**	**13**	**13**	**13**	**13**	**13**	**13**	12	12	12	12	12	12	**13**	**13**	**13**	**13**	**13**	−12	−12	12	12	−12	−12
tetris14 (17)	15	16	15	14	14	15	16	16	16	16	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	⊕**17**	⊕**17**	+**17**	+**17**	⊕**17**	⊕**17**
tidybot11 (20)	14	16	**18**	14	17	**18**	14	**18**	14	**18**	14	**18**	14	**18**	14	**18**	14	**18**	**18**	14	**18**	14	**18**	14	**18**	14	**18**
tidybot14 (20)	10	12	**14**	10	13	**14**	10	**14**	10	**14**	10	**14**	10	**14**	9	**14**	9	**14**	**14**	10	**14**	10	**14**	10	**14**	10	**14**
tpp06 (30)	6	6	6	6	6	6	6	6	6	6	**8**	**8**	7	7	6	6	6	6	6	7	7	⊕**8**	⊕**8**	⊕**8**	⊕**8**	⊕**8**	⊕**8**
trucks06 (30)	13	13	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	+**14**	**14**	+**14**	**14**	+**14**	**14**
visitall11 (20)	16	16	16	**17**	**17**	**17**	**17**	**17**	**17**	**17**	16	16	14	14	11	11	**17**	**17**	**17**	**17**	**17**	+**17**	+**17**	−14	−14	+**17**	+**17**
visitall14 (20)	12	12	12	13	13	13	13	13	13	13	11	11	9	9	5	5	**14**	**14**	**14**	13	13	+13	+13	−9	−9	+13	+13
woodworking08 (30)	12	12	12	14	14	15	12	12	12	12	16	**17**	15	16	13	15	13	12	12	14	15	+14	⊕**17**	+15	⊕**17**	⊕15	⊕**17**
woodworking11 (20)	7	7	7	9	9	10	7	7	7	7	11	**12**	10	11	8	10	8	7	7	9	10	+9	⊕**12**	+10	⊕**12**	⊕10	⊕**12**
zenotravel02 (20)	10	10	10	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	10	10	**11**	**11**	**11**	**11**	**11**	**11**	**11**	+**11**	+**11**	⊕**11**	⊕**11**	+**11**	+**11**
Σ (1697)	921	933	938	903	913	927	921	938	920	935	932	957	937	961	903	925	929	961	969	960	989	964	**1 001**	949	985	959	987
Σ w/o freecell00 (1617)	854	862	867	862	877	891	863	880	862	877	859	884	864	889	835	854	867	897	904	893	918	892	**929**	876	912	890	918

Table 8.1: Number of solved tasks for different heuristics. Only the domains with a difference in coverage are listed.

n	5	10	50	100	250	500	750	1 000	2 000
D-S$_n$	896	910	923	**925**	897	863	828	791	741
D-K$_n^1$	951	957	**969**	961	947	935	925	925	899
D-K$_n^2$	953	960	**967**	956	944	925	917	919	912
D-L$_n^1$	934	951	**965**	963	932	877	850	824	765
D-L$_n^2$	937	951	**962**	957	928	874	844	810	744

Table 8.2: Number of solved tasks for a different number of samples n.

	D-All+I	D-K$_{50}^1$	lmc	ms	flw	comp1	comp2	ppdbs	scrp
overall	1 001	969	911	895	816	1 046	1 091	1 090	1 112
w/o freecell00	929	904	896	874	762	1 019	1 059	1 056	1 040

Table 8.3: Number of solved tasks for a different heuristic search planners.

is smaller than for the variant without the added constraint or for `Init` (the cases with "–"): only 2 out of 65 for D-All+I, 4 for D-$\hat{\text{S}}_{1k}$+I, and only 3 for D-M^1+I. So, it indeed seems that the combination of the optimization for all states and for the initial state at the same time brings the best from both.

A similar approach to All+I is to compute two potential functions, for all states and for the initial state, separately and take the maximum as a heuristic value (denoted by MaxIA). The comparison between All+I and MaxIA shows that incorporating the initial state as a constraint (All+I) provides better results overall, but in some domains MaxIA solves more tasks than All+I. In parcprinter08, petri-net-alignment18, and termes18, N-MaxIA solves more tasks than N-All+I, and in openstacks06 D-MaxIA solves more tasks than D-All+I. In all other domains, All+I is at least as good as MaxIA.

To compare the results to other state-of-the-art heuristics, we also evaluated the LM-Cut (`lmc`) heuristic (Helmert & Domshlak, 2009), the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy and non-greedy bi-simulation shrink strategy (Helmert et al., 2014; Sievers et al., 2016), the flow (`flw`) heuristic (Bonet & van den Briel, 2014; Bonet, 2013), and the four best-performing non-portfolio planners from IPC 2018: Complementary1 (`comp1`) (Franco et al., 2018), and Complementary2 (`comp2`) (Franco et al., 2017, 2018) planners, the Planning-PDBs planner (`ppdbs`) (Moraru et al., 2018; Franco et al., 2017), and the Scorpion planner (`scrp`) (Seipp, 2018; Seipp & Helmert, 2018). The overall coverage is shown in Table 8.3.

Our best variant of the potential heuristic (D-All+I) solved 90 (33 without the freecell domain) more tasks than `lmc`, 106 (55) more than `ms`, and 185 (167) more than `flw`. And the best performing variant that uses mutexes for estimating the number of reachable states, D-K$_{50}^1$, solved 58 (8), 74 (30), and 153 (142) more tasks than `lmc`, `ms`, and `flw`, respectively. However, all variants of the potential heuristic were surpassed by all four evaluated planners from IPC 2018. D-All+I solved 45 (90), 90 (130), 89 (127), and 111 (111) fewer planning tasks than `comp1`, `comp2`, `ppdbs`, and `scrp`, respectively.

8.6 Summary

We showed that utilizing mutexes can significantly improve potential heuristics. Disambiguations in the goal and operator preconditions increase the overall coverage whenever they are used and they decrease the coverage only in a very small number of domains. It is also clear that the multi-fact disambiguation dominates the single-fact disambiguation for virtually no cost at all.

A more accurate estimation of the number of reachable states using mutexes results in a higher overall coverage than the optimization for all syntactic states, but mainly due to two domains only. However, a similar technique used for an ensemble of potential heuristics, each conditioned on a partial state, increases the number of solved tasks even if the partial states are sampled randomly. A better structural analysis, for example with causal graphs, could probably lead to even better selection of the partial states, but we leave it for future research.

The additional constraint on the heuristic value for the initial state improves the overall coverage whenever used. Its use is orthogonal to the disambiguation technique and it turned out that such constraint in combination with disambiguations and the optimization for all syntactic states results in the best performing variant of the potential heuristic. This variant outperforms the LM-Cut, merge-and-shrink, and flow heuristics solving 90, 106, and 185 more problems, respectively, on the standard benchmark set. However, it is still surpassed by all four best-performing non-portfolio planners from the last IPC 2018.

Chapter 9

Operator Mutex

In this chapter, we move from invariants describing mutual exclusion between facts to sets of operators that are mutually exclusive in the sense that applying one operator forbids to apply others in the shortest optimal plan. We say that a set of operators is a strong operator mutex (op-mutex) if no strongly optimal plan contains all of them. We introduce several methods to infer op-mutexes automatically from the description of the task, and show that they can be found in a sizable number of domains.

We combine op-mutexes with structural symmetries in order to identify operators that can be safely removed from the planning tasks. The notion of symmetries is not new to classical planning (Fox & Long, 1999). They can be used to reduce the size of the search space in heuristic search planners (Pochter et al., 2011; Domshlak et al., 2012), obtain more compact encodings in SAT-based planners (Rintanen, 2003), compute better heuristics (Domshlak et al., 2013; Sievers et al., 2015, 2017), transform the task (Riddle et al., 2015), or ground the task (Röger et al., 2018).

Here, we use symmetries to prove that certain operators can be safely removed from the planning task. We show that at least one optimal plan is preserved when removing some operators that are all op-mutex with other symmetric operators. In order to find out whether more operators can be removed, we then analyze which symmetries are preserved after removing a set of operators in this way. This leads to a fixpoint computation method that can remove a sizable number of operators in some domains.

9.1 Background

In this chapter, we use STRIPS formalism (Definition 3.1). Recall that a plan is called optimal if its cost is minimal among all plans, and it is called strongly optimal if it is optimal and it contains the minimum number of operators among all optimal plans. We denote the set of all strongly optimal plans in the STRIPS planning task Π by $\mathcal{P}_\Pi$.

Definition 9.1. A **labeled transition system** (LTS) is a tuple $\Theta = \langle \mathcal{S}, L, T, s_I, S_\star \rangle$, where $\mathcal{S}$ is a finite set of **states**, L is a finite set of **labels** with associated cost $\mathrm{c}(l) \in \mathbb{R}^+_0$ to each label $l \in L$, $T \subseteq \mathcal{S} \times L \times \mathcal{S}$ is a set of **transitions**, $s_I \in \mathcal{S}$ is the **initial state**, and $S_\star \subseteq \mathcal{S}$ is a set of **goal states**. We write $s_1 \xrightarrow{l} s_2$ to refer to a transition from s_1 to s_2 with the label l. A sequence of labels $\langle l_1, \dots, l_n \rangle$ is a **path** from s_0 to s_n in Θ if there exist $s_{i-1} \xrightarrow{l_i} s_i \in T$ for every $i \in \{1, \dots, n\}$. We say that s' is **reachable from** s if there is a path from s to s'.

The **state space** of a planning task Π is the LTS Θ_Π where $\mathcal{S} := \mathcal{R}_\Pi$, $s_I := s_I$, $s \in S_\star$ iff $s_G \subseteq s$, the labels L are the operators $\mathcal{O}$ with the given costs, and $s \xrightarrow{o} s'$ is a transition in T if $\text{pre}(o) \subseteq s$ and $o[\![s]\!] = s'$.

Definition 9.2. An **abstraction** α for a transition system Θ is a function mapping states $\mathcal{S}$ into a set of abstract states $\mathcal{S}^\alpha$. The **abstract transition system** Θ^α is defined as $\langle \mathcal{S}^\alpha, L, T^\alpha, s_I^\alpha, S_\star^\alpha \rangle$, where $\alpha(s) \xrightarrow{o} \alpha(s') \in T^\alpha$ iff $s \xrightarrow{o} s' \in T$, $s_I^\alpha = \alpha(s_I)$, and $S_\star^\alpha = \{\alpha(s) \mid s \in S_\star\}$.

Definition 9.3. A **projection** of the state space Θ_Π to the set of facts $F \subseteq \mathcal{F}$ is an abstract transition system $\Theta_\Pi^{\alpha_F}$ with the abstraction $\alpha_F(s) = s \cap F$.

For notational convenience, we will sometimes abuse the notation for sequences by referring to a sequence as a set. For a set of operators $O \subseteq \mathcal{O}$, we denote $\Pi \setminus O$ the planning task resulting from removing operators O, $\Pi \setminus O = \langle \mathcal{F}, \mathcal{O} \setminus O, s_I, s_G \rangle$.

9.2 Operator Mutexes and Redundancy

Our goal here is to eliminate operators from a planning task, if they can be proven unnecessary to find a strongly optimal plan. We say that a set of operators is redundant if removing them from the task preserves at least one strongly optimal plan.

Definition 9.4. Given the planning task Π, a set of operators $O \subseteq \mathcal{O}$ is **redundant** if $\mathcal{P}_\Pi \neq \emptyset$ implies $\mathcal{P}_{\Pi \setminus O} \cap \mathcal{P}_\Pi \neq \emptyset$.

The union of two redundant sets is not necessarily also a redundant set. However, we can merge two sets of redundant operators, if one is shown to be redundant in the task where the other has already been removed.

Proposition 9.5. *Let Π denote a planning task with a set of operators $\mathcal{O}$ and let $R_1, R_2 \subseteq \mathcal{O}$, $R_1 \cap R_2 = \emptyset$. If R_1 is redundant in Π and R_2 is redundant in $\Pi \setminus R_1$, then $R_1 \cup R_2$ is redundant in Π.*

Proof. $\mathcal{P}_\Pi \neq \emptyset$ implies $\mathcal{P}_{\Pi \setminus R_1} \cap \mathcal{P}_\Pi \neq \emptyset$, because R_1 is redundant in Π. And $\mathcal{P}_{\Pi \setminus R_1} \neq \emptyset$ implies $\mathcal{P}_{\Pi \setminus (R_1 \cup R_2)} \cap \mathcal{P}_{\Pi \setminus R_1} \neq \emptyset$, because R_1 is redundant in $\Pi \setminus R_1$. Therefore $\mathcal{P}_{\Pi \setminus (R_1 \cup R_2)} \subseteq \mathcal{P}_{\Pi \setminus R_1} \subseteq \mathcal{P}_\Pi$ and $\mathcal{P}_{\Pi \setminus (R_1 \cup R_2)} \neq \emptyset$. □

The notion of mutually exclusive facts that cannot be together in any reachable state has proven to be very useful to improve different types of planning algorithms. Here, we define strong operator mutexes as sets of operators that cannot be part of the same strongly optimal plan.

Definition 9.6. A **strong operator mutex** (op-mutex) $O \subseteq \mathcal{O}$ is a non-empty set of operators such that $O \not\subseteq \pi$ for every $\pi \in \mathcal{P}_\Pi$.

The mutual exclusion between operators with respect to strongly optimal plans means that every op-mutex always contains at least one operator that is redundant.

Proposition 9.7. *In every op-mutex $O \subseteq \mathcal{O}$, there is an operator $o \in O$ such that $\{o\}$ is redundant.*

Proof. Given $\pi \in \mathcal{P}_\Pi$, $O \not\subseteq \pi$ by Definition 9.6, thus there is an operator $o \in O$ such that $o \notin \pi$, therefore $\pi \in \mathcal{P}_{\Pi \setminus \{o\}}$. □

From now on, we will concentrate on the special case of op-mutexes that are formed by pairs of operators, since, as shown by Proposition 9.8, they can be used to identify redundant sets consisting of more than one operator.

Proposition 9.8. *Let Π denote a planning task, let $O_1, O_2 \subseteq \mathcal{O}$ such that $O_1 \cap O_2 = \emptyset$ and $\{o_1, o_2\}$ is an op-mutex for every $o_1 \in O_1$ and every $o_2 \in O_2$. Then O_1 or O_2 is redundant.*

Proof Sketch. If $o \notin \pi$ for every $o \in O_1$ and every $\pi \in \mathcal{P}_\Pi$, then $\mathcal{P}_{\Pi \setminus O_1} = \mathcal{P}_\Pi$. Otherwise there exists $o \in O_1$ s.t. $o \in \pi$, for some $\pi \in \mathcal{P}_\Pi$, and since $\{o, o'\}$ is an op-mutex for each $o' \in O_2$, $o' \notin \pi$. Thus, O_2 is redundant. □

Op-mutexes are useful for obtaining candidate sets of operators that may be redundant. However, op-mutexes are not sufficient to prove any such set redundant. Later, we will show how to combine op-mutexes with symmetries to identify which set of operators is actually redundant.

9.3 Inference of Operator Mutexes

In this section, we describe several methods for inference of op-mutexes based on well-known planning techniques.

9.3.1 Abstractions

Abstraction heuristics map the state space of a planning task into a smaller abstract state space and use the distance in the abstract state space as an admissible estimation. Abstractions can also be used to infer op-mutexes.

Theorem 9.9. *Let Π denote a planning task, and let $o_1, o_2 \in \mathcal{O}$, $o_1 \neq o_2$, denote operators in Π. If there exists an abstract transition system Θ^α_Π such that for every transition $s_1 \xrightarrow{o_1} s'_1$ and every transition $s_2 \xrightarrow{o_2} s'_2$, s_2 is not reachable from s'_1 and s_1 is not reachable from s'_2, then $\{o_1, o_2\}$ is an op-mutex.*

Proof Sketch. The theorem clearly holds for Θ_Π, since o_2 is never reachable from o_1 and vice versa. This means that o_1 and o_2 are never part of the same path, and therefore they cannot coexist in the same plan.

In their work, Helmert et al. (2007) showed that abstractions preserve all paths in the state space, thus if o_1 and o_2 are not reachable from one another in Θ^α_Π, then the same holds for Θ_Π. □

Theorem 9.9 shows how we can infer strong operator mutexes using any known method for computing abstractions of planning tasks, including pattern databases (Culberson & Schaeffer, 1996; Edelkamp, 2001), merge-and-shrink (Helmert et al., 2014; Sievers et al., 2014), or Cartesian abstractions (Seipp & Helmert, 2018). Analyzing what abstraction methods are best suited to find op-mutexes is left for future work. We will focus our evaluation only on projections to individual mutex groups.

To compute op-mutexes with this method one needs to check reachability between every pair of states in the abstract state space Θ^α_Π. However, this can be done more efficiently by considering instead a smaller abstract state space Θ^γ_Π, where $\gamma(s) = \gamma(s')$ if $\alpha(s)$ and $\alpha(s')$ belong to the same strongly connected component in Θ^α_Π. In other words, given an abstract LTS in which we want to look for op-mutexes, one can always apply a condensation of its strongly connected components. The set of inferred op-mutexes will not be affected because if $\alpha(s)$ and $\alpha(s')$ belong to the same strongly connected component, then they have the same set of reachable abstract states.

9.3.2 Operators-as-Facts Compilation

Here, we propose a compilation that allows us to use methods for inferring mutexes to infer op-mutexes. The idea is to transform the task by adding one artificial fact per operator, so that two operators are op-mutex if their corresponding artificial facts are mutex.

Definition 9.10. Given the planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, the **op-fact compilation** of Π is another planning task $\Pi_{\text{op}} = \langle \mathcal{F} \cup \mathcal{F}_{\text{op}}, \mathcal{O}_{\text{op}}, s_I, s_G \rangle$, where $\mathcal{F}_{\text{op}} = \{f_o \mid o \in \mathcal{O}\}$, and $\mathcal{O}_{\text{op}} = \{o^\diamond \mid o \in \mathcal{O}\}$ where each operator $o^\diamond$ is defined as $o^\diamond = \langle \text{pre}(o), \text{add}(o) \cup \{f_o\}, \text{del}(o), \text{c}(o) \rangle$.

Theorem 9.11. *Let Π_{op} denote the op-fact compilation of the planning task Π, and let $F = \{f_{o_1}, \dots, f_{o_n}\} \subseteq \mathcal{F}_{\text{op}}$. If F is a mutex in Π_{op}, then $\{o_1, \dots, o_n\}$ is an op-mutex in Π.*

Proof. Every $f_o \in \mathcal{F}_{\text{op}}$ appears only in the add effect of the operator $o^\diamond$ and otherwise the operators $o^\diamond$ and o are identical. Therefore a sequence of operators $\pi^\diamond = \langle o_1^\diamond, \dots, o_n^\diamond \rangle$ in Π_{op} is applicable in s_I iff the sequence $\pi = \langle o_1, \dots, o_n \rangle$ of the original task Π is applicable in s_I. Moreover, it holds that $\pi^\diamond[\![s_I]\!] = \pi[\![s_I]\!] \cup \{f_o \mid o^\diamond \in \pi^\diamond\}$. Assume, for contradiction, that $F = \{f_{o_1}, \dots, f_{o_n}\}$ is a mutex in Π_{op} and there is a path π in the task Π s.t. $\{o_1, \dots, o_n\} \subseteq \pi$. This immediately leads to a contradiction, because $F \subseteq \pi^\diamond[\![s_I]\!]$. □

The op-fact compilation allows us to infer op-mutexes by using known methods for inference of mutexes, including the well known h^m family of heuristics (Haslum & Geffner, 2000). However, we should stress that not every inference method is suitable for this type of compilation. For example, the inference of fact-alternating mutex groups (Section 5.1) by any inference method would fail to produce any mutex group (and thus any mutex) containing any fact from $\mathcal{F}_{\text{op}}$, because they appear only in add effects.

9.3.3 Critical-Path Heuristics

The alternative characterization of the h^m family of heuristics introduced by Haslum (2009) can also be adapted to compute op-mutexes directly, without considering artificial facts for every operator. Definition 9.12 differs from the definition provided by Haslum only in that the goal specification is empty, because we are only interested in computing the set of operators reachable from the initial state.

Definition 9.12. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a planning task. The planning task $\Pi^m = \langle \mathcal{F}^m, \mathcal{O}^m, I^m, \emptyset \rangle$ consists of a set of facts $\mathcal{F}^m = \{\phi_c \mid c \subseteq \mathcal{F}, |c| \leq m\}$, a set of operators $\mathcal{O}^m$, an initial state $I^m = \{\phi_c \mid c \subseteq s_I, |c| \leq m\}$, and an empty goal specification. For each operator $o \in \mathcal{O}$ and for each subset of facts $F \subseteq \mathcal{F}$ such that $|F| < m$ and F is disjoint with add(o) and del(o), the planning task Π^m contains an operator $\omega_{o,F} \in \mathcal{O}^m$ with: $\text{pre}(\omega_{o,F}) = \{\phi_c \mid c \subseteq (\text{pre}(o) \cup F), |c| \leq m\}$, $\text{add}(\omega_{o,F}) = \{\phi_c \mid c \subseteq (\text{add}(o) \cup F), c \cap \text{add}(o) \neq \emptyset, |c| \leq m\}$, $\text{del}(\omega_{o,F}) = \emptyset$.

To compute op-mutexes, we consider the compiled planning task for every operator o, that approximates which operators are reachable from the result of applying o in any reachable state.

Definition 9.13. Given the planning task $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, a set of mutexes $\mathcal{M} \subseteq 2^{\mathcal{F}}$ for Π, an operator $o \in \mathcal{O}$, and a natural number $m \geq 1$, the **m-$\mathcal{M}$-compilation of Π for o** is another planning task $\Pi^m_{\mathcal{M},o} = \langle \mathcal{F}^m, \mathcal{O}^m, I^m_{\mathcal{M},o}, \emptyset \rangle$, where $\mathcal{F}^m$ and $\mathcal{O}^m$ are the same as in Π^m, and $I^m_{\mathcal{M},o} = \{\phi_c \mid c \subseteq (o[\![\mathrm{pre}(o)]\!] \cup E_{\mathcal{M},o}), |c| \leq m, c \notin \mathcal{M}\}$, where $E_{\mathcal{M},o} = \{f \mid f \in (\mathcal{F} \setminus \mathrm{del}(o)), (\{f\} \cup \mathrm{pre}(o)) \notin \mathcal{M}^\star, (\{f\} \cup o[\![\mathrm{pre}(o)]\!]) \notin \mathcal{M}^\star\}$ and $\mathcal{M}^\star = \{M \cup N \mid M \in \mathcal{M}, N \subseteq \mathcal{F}\}$.

The m-$\mathcal{M}$-compilation for an operator o differs from Definition 9.12 only in the construction of the initial state. The initial state is constructed as an over-approximation of the union of all possible states resulting from the application of the operator o on some reachable state while taking into account given mutexes $\mathcal{M}$. So, for an empty $\mathcal{M}$, the initial state $I^m_{\mathcal{M},o}$ will contain all facts except those from the delete effect of o, because they can never be part of the state resulting from the application of o. But if we are given additional information in the form of mutexes, we can apply it to exclude from $I^m_{\mathcal{M},o}$ the facts that cannot be part of any state resulting from the application of o.

Theorem 9.14. *Let Π denote a planning task, $\mathcal{M} \subseteq 2^{\mathcal{F}}$ a set of mutexes for Π, and $o_1, o_2 \in \mathcal{O}$, $o_1 \neq o_2$, two reachable operators with corresponding m-$\mathcal{M}$-compilations $\Pi^m_{\mathcal{M},o_1}$ and $\Pi^m_{\mathcal{M},o_2}$. If $\omega_{o_2,\emptyset}$ is not reachable in $\Pi^m_{\mathcal{M},o_1}$ and $\omega_{o_1,\emptyset}$ is not reachable in $\Pi^m_{\mathcal{M},o_2}$, then $\{o_1, o_2\}$ is an op-mutex.*

Proof Sketch. Haslum (2009) showed that if $\omega_{o,\emptyset}$ is not reachable in Π^m, then o is not reachable in Π. So we need to show that for every reachable state $s \in \mathcal{R}_\Pi$ such that o_1 is applicable in s it holds that $\{\phi_c \mid c \subseteq o_1[\![s]\!], |c| \leq m\} \subseteq I^m_{\mathcal{M},o_1}$, because $\Pi^m_{\mathcal{M},o_1}$ is a delete-free problem and if o_2 is not reachable from the superset of all possible reachable states resulting from the application of the operator o_1, then o_2 cannot be reachable after any application of o_1 in Π (and the same for the reachability of o_1 from o_2). The rest follows directly from Theorem 9.9 with the identity abstraction $\alpha(s) = s$.

Let $s \in \mathcal{R}_\Pi$ denote a reachable state such that $\mathrm{pre}(o_1) \subseteq s$, and let $S = \{c \mid c \subseteq o_1[\![s]\!], |c| \leq m\}$. Since s is reachable, $o_1[\![s]\!]$ is reachable, therefore $S \cap \mathcal{M} = \emptyset$. Now it is enough to show that $o_1[\![s]\!] \subseteq (o_1[\![\mathrm{pre}(o_1)]\!] \cup E_{\mathcal{M},o_1})$. Since $o_1[\![s]\!] \cap \mathrm{del}(o_1) = \emptyset$ by definition and both s and $o_1[\![s]\!]$ are reachable, it follows that for every $f \in (o_1[\![s]\!] \setminus o_1[\![\mathrm{pre}(o_1)]\!])$ it holds that $f \cup \mathrm{pre}(o_1) \notin \mathcal{M}^\star$ and $f \cup o_1[\![\mathrm{pre}(o_1)]\!] \notin \mathcal{M}^\star$. □

9.3.4 Operators with Irreversible Add Effect

The last method is based on the observation that some domains contain facts that once added to a state, they cannot be subsequently deleted by any operator—they are in this sense irreversible. Any fact that does not appear in any delete effect is irreversible and operators that add the same irreversible facts (and nothing else) form pairwise an op-mutex.

Theorem 9.15. *Let $O \in \mathcal{O}$ be a set of operators with the same add effects, denoted $\mathrm{add}(O)$. If for all $o \in \mathcal{O}$, it holds that $\mathrm{add}(O) \cap \mathrm{del}(o) = \emptyset$, then for every $o_1, o_2 \in O$, $o_1 \neq o_2$, it holds that $\{o_1, o_2\}$ is an op-mutex.*

Proof Sketch. (By contradiction) Let $o_1, o_2 \in O$, $o_1 \neq o_2$, and let $\pi \in \mathcal{P}_\Pi$ s.t. $o_1, o_2 \in \pi$. Assume WLOG, that o_1 occurs before o_2 in π. There is a state s in the plan π where o_2 is applied. Note that since no operator deletes any fact $f \in \text{add}(o_1) = \text{add}(o_2)$ and o_1 has already occurred, then $\text{add}(o_2) \subseteq s$. This means that $o_2[\![s]\!] \subseteq s$, in contradiction to the strong optimality of the plan. □

The previous op-mutex inference methods produce op-mutexes that are not part of any plan. Theorem 9.15 can, however, provide op-mutexes that can be a part of some plan (or even an optimal plan), but cannot be part of any strongly optimal plan.

9.4 Symmetries

We adopt the definition of symmetry by Shleyfman et al. (2015) with additional stabilizer for the initial state as was proposed by Pochter et al. (2011).

Definition 9.16. A **plan preserving symmetry** (or **symmetry** for short) of the transition system $\Theta = \langle \mathcal{S}, L, T, s_I, S_\star \rangle$ is a permutation σ of $\mathcal{S} \cup L$ mapping states to states and labels to labels such that

- $s \xrightarrow{o} s' \in T$ iff $\sigma(s) \xrightarrow{\sigma(o)} \sigma(s') \in T$,
- $\text{c}(o) = \text{c}(\sigma(o))$,
- $s \in S_\star$ iff $\sigma(s) \in S_\star$, and
- $\sigma(s_I) = s_I$

for all states $s, s' \in \mathcal{S}$ and all labels $o \in L$. For notational convenience, we extend permutations σ to sequences $\sigma(\langle x_1, \dots, x_n \rangle) = \langle \sigma(x_1), \dots, \sigma(x_n) \rangle$ and sets $\sigma(\{x_1, \dots, x_n\}) = \{\sigma(x_1), \dots, \sigma(x_n)\}$.

Symmetries are closed under composition and inverse, and therefore form the automorphism group $\text{Aut}(\Theta)$ of the transition system.

The reason for stabilizing both the initial state and goals (i.e., mapping goals to goals and the initial state to itself) is that under this type of symmetry, symmetries preserve plans in the sense that for any plan, every symmetric sequence of actions is also a plan of the same length and cost.

Theorem 9.17. *Let Π denote a planning task and let σ be a symmetry for Θ_Π. For every plan π it holds that $\sigma(\pi)$ is also a plan, moreover $|\pi| = |\sigma(\pi)|$ and $\text{c}(\pi) = \text{c}(\sigma(\pi))$.*

Theorem 9.17 is adopted to our notation and definition of symmetry from the work of Shleyfman et al. (2015, Theorem 1). As a direct corollary of this theorem we have that $\sigma(\mathcal{P}_\Pi) = \mathcal{P}_\Pi$. This means that op-mutexes are invariant under symmetry.

Proposition 9.18. *Let Π denote a planning task, and let σ be a symmetry for Θ_Π. If O is an op-mutex so is $\sigma(O)$.*

Proof. (By contradiction) Suppose that $\sigma(O)$ is not an op-mutex, than there exist $\pi \in \mathcal{P}_\Pi$ s.t. $\sigma(O) \in \pi$. By the previous Theorem 9.17 we have that $\sigma^{-1}(\pi) \in \mathcal{P}_\Pi$. Therefore, $O = \sigma^{-1}(\sigma(O)) \in \sigma^{-1}(\pi)$, contradicting the fact that O is an op-mutex. □

Proposition 9.18 shows that we can use symmetries to generate more op-mutexes. However, this makes sense only if the input set of op-mutexes was obtained by some method that does not already explore the symmetric op-mutexes. As the work of Röger et al. (2018) suggests, Proposition 9.18 would not be useful for h^m heuristics with op-fact compilation, or our methods based on critical-path heuristics (Theorem 9.14). The same happens for op-mutexes obtained from searching over all (symmetric) operators with irreversible add effects (Theorem 9.15), for which further inference of op-mutexes using symmetries is not possible.

Abstractions, on the other hand, focus on a subset of the facts of the problem and the results can be extrapolated to other symmetric projections of the problem, e.g., if several mutex groups are symmetric, it may suffice to compute the set of op-mutexes for one of them and then extend the set of op-mutexes with symmetries.

In Proposition 9.8, we have shown how to find candidates for redundant sets of operators using op-mutex pairs. Symmetries allow us to identify which sets of operators are actually redundant.

Theorem 9.19. *Let Π denote a planning task, $O_1, O_2 \subseteq \mathcal{O}$ s.t. $O_1 \cap O_2 = \emptyset$, and $\{o_1, o_2\}$ is an op-mutex for every $o_1 \in O_1$ and every $o_2 \in O_2$. If for every $o_2 \in O_2$ there exists $\sigma \in \mathrm{Aut}(\Theta_\Pi)$ s.t. $\sigma(o_2) \in O_1$, then O_2 is redundant.*

Proof. This follows directly from the fact that if there is $o \in O_2$ and $\pi \in \mathcal{P}_\Pi$ s.t. $o \in \pi$, then $\sigma(o) \in \sigma(\pi) \in \mathcal{P}_\Pi$. Hence, since $\sigma(o) \in O_1$, O_2 is redundant by Proposition 9.8. □

In its simplest form, if two operators form an op-mutex and there exists a symmetry between these two operators, then one of them can be safely removed. Note that Theorem 9.19 is not restricted to a single symmetry. So, if we find a single operator o and a set of operators that are all symmetric to o and op-mutex with o, then we can safely remove such a set of operators. We can also choose to apply Theorem 9.19 to a single symmetry. If we find a symmetry σ and a set of operators O such that $\sigma(O) \cap O = \emptyset$ and op-mutexes form a complete bipartite graph between O and $\sigma(O)$, then we can safely remove either O or $\sigma(O)$.

9.5 Destroying and Preserving Symmetries

In this section, we describe the effect of removing operators on destroying and preserving symmetries. We say that a symmetry is destroyed by removing a certain set of operators if this symmetry is not valid for the reduced planning task. Conversely, a preserved symmetry is the one that still holds in the reduced planning task.

Definition 9.20. Given an LTS $\Theta = \langle \mathcal{S}, L, T, s_I, S_\star \rangle$ and a set of labels $K \subseteq L$, $\Theta \setminus K = \langle \mathcal{S}, L \setminus K, T_K, s_I, S_\star \rangle$ denotes Θ **reduced by** K, where $s \xrightarrow{l} t \in T_K$ iff $s \xrightarrow{l} t \in T$ and $l \notin K$.

Suppose that there is a label $l_1 \in L \setminus K$ and a transition $s \xrightarrow{l_1} t \in T$, and for some symmetry σ it holds that $\sigma(l_1) \in K$. Then σ is not a symmetry for $\Theta \setminus K$, since then there must exist $s' \xrightarrow{\sigma(l_1)} t' = \sigma(s \xrightarrow{l_1} t)$, but $\Theta \setminus K$ does not contain any such a transition. This is exactly the case for any redundant set obtained according to Theorem 9.19, because for any such redundant set K there is a symmetry that maps K to $L \setminus K$. Therefore removing K always destroys at least one symmetry.

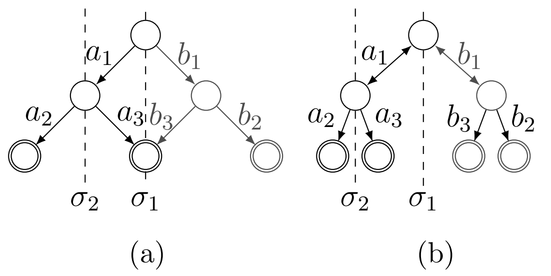

Figure 9.1: (a) Emergence of a new symmetry; (b) Preserving a symmetry by merging two redundant sets.

We can, however, preserve a symmetry if we find a redundant set consisting of operators covering both sides of the symmetry mapping.

Definition 9.21. Given an LTS Θ and a set of labels $K \subseteq L$, we say that $\sigma \in \mathrm{Aut}(\Theta)$ is **preserved for** K if $\sigma(K) = K$.

Let us show that the term is well defined.

Theorem 9.22. *Let $\sigma \in \mathrm{Aut}(\Theta)$ be preserved for K, as described above. Then σ is also a symmetry for $\Theta \setminus K$.*

Proof. Note that the transition system $\Theta \setminus K$ differs from Θ only on the sets of transitions T_K and labels $L \setminus K$. Thus, it suffices to show that σ preserves the structure on these sets. Since σ is a permutation of $\mathcal{S} \cup L$, and by definition it holds that $\sigma(\mathcal{S}) = \mathcal{S}$, then $\sigma(L) = L$. Thus, $\sigma(L \setminus K) = \sigma(L) \setminus \sigma(K) = L \setminus K$. This, in turn, implies that σ preserves the transitions in T_K, by definition of the reduced LTS. □

Removal of a redundant set of operators can also result in emergence of new symmetries that were not in the original task. This is illustrated in Figure 9.1a: σ_2 is not a symmetry in the LTS, but after removing operators b_i (which are redundant because all $\{a_i, b_i\}$ are symmetric and pairwise op-mutex), σ_2 becomes a symmetry for the resulting reduced LTS.

Figure 9.1b shows that a symmetry destroyed by removing redundant operators can be salvaged after subsequently removing another set of redundant operators. If we use symmetry σ_1 to remove b_3 (because $\{a_3, b_3\}$ is an op-mutex), σ_1 is destroyed. If we use afterwards the symmetry σ_2 to remove a_3 ($\{a_2, a_3\}$ is an op-mutex), then σ_1 becomes, again, a symmetry for the reduced planning task.

These observations motivate us to infer redundant operators using a fixpoint computation.

9.6 Inference of Redundant Operators

The algorithm for the inference of a redundant set of operators should aim at finding the largest set possible, because, ultimately, we want to use the redundant set for the simplification of the planning task. Redundant sets cannot be automatically merged, but as proven in Proposition 9.5, a fixpoint computation is possible.

Theorem 9.19 provides a way to identify redundant sets of operators, but we also already explained that the removal of such operators always destroys at least one symmetry that could be helpful for further inference steps. We propose an algorithm that is based

Algorithm 9.1: Fixpoint computation of a redundant set.

```
Input: Set of symmetries $\Sigma$, set of operators $\mathcal{O}$, set of op-mutex pairs $\mathcal{M}$
Output: Redundant set of operators $R$
1  $R \leftarrow \emptyset$;
2  do
3      $\Sigma^+ \leftarrow \{\sigma \in \Sigma \mid \sigma(R) = R\}$;
4      $\mathcal{R} \leftarrow \{$RedundantSet($\sigma$, $R$)$\mid \sigma \in \Sigma^+\}$;
5      $R' \leftarrow$ SelectRedundantSet ($\mathcal{R}$);
6      $R \leftarrow R \cup R'$;
7  while $|R'| > 0$;
8  function RedundantSet($\sigma$, $R$)
9      $S \leftarrow \emptyset$;
10     $C \leftarrow \{o \mid o \in \mathcal{O} \setminus R, o \neq \sigma(o), \{o, \sigma(o)\} \in \mathcal{M}\}$;
11     while $|C| > 0$ do
12         $o' \leftarrow$ SelectCandidate($C$, $R \cup S$);
13         $S \leftarrow S \cup \{o'\}$;
14         $C \leftarrow \{o \mid o \in C \setminus \{o'\}, \{o, \sigma(o')\} \in \mathcal{M}\}$;
15     return $S$;
```

on the assumption that as many symmetries as possible should be preserved in each step in order to increase the chance of finding more redundant operators in the following steps. Moreover, in a fixpoint computation, it can happen that a symmetry that is destroyed in one step can re-emerge again in one of the next steps (Theorem 9.22), so after removing a set of redundant operators we check which symmetries of the original planning task are preserved under the current set of redundant operators.

Since Theorem 9.19 requires to find a complete bipartite subgraph of op-mutexes and determining the maximal one is already NP-Hard (Garey & Johnson, 1979), we propose a greedy algorithm that gradually increases the size of the resulting redundant set in each step.

The pseudo-code in Algorithm 9.1 contains two selection steps that can use any rule and the algorithm will still remain sound. `SelectRedundantSet` on line 5 selects one of the redundant sets found on the previous line. In our implementation, we select the largest of those sets that preserves the most symmetries. `SelectCandidate` on line 12 selects one operator from the set of candidates C. We select the operator $o' \in C$ for which the set $R \cup S \cup \{o'\}$ preserves the most symmetries.

Proposition 9.23. *Algorithm 9.1 always returns a redundant set of operators.*

Proof Sketch. The main cycle (lines 2–7) computes a union of sets found by `RedundantSet` function in consecutively reduced planning tasks. Therefore what remains to show is that these sets are redundant (Proposition 9.5). To prove that `RedundantSet` always returns a redundant set, it is enough to show that in every cycle (a) $\{o, \sigma(o)\}$ is op-mutex for every $o \in C$; (b) $\{o, o'\}$ is op-mutex for every $o \in S$ and every $o' \in C$; (c) $S \cup \{o'\}$ is redundant for any $o' \in C$. (a) follows from line 10. (b) follows from line 14 and the symmetry of op-mutexes (Proposition 9.18). (c) follows from (a), (b), and Theorem 9.19 with sets $O_1 := \sigma(S)$ and $O_2 := S$ and a single symmetry σ. □

domain	op-mutex pairs in 5 min.				avg. time [s]		
	fam	Π^2_{op}	**2-h^2**	**iadd**	**fam**	Π^2_{op}	**2-h^2**
agricola18 (20)	(15) 59 862.8	(18) **425 278.3**	(0) 0.0	0.0	**29.1**	86.0	0.0
barman11 (20)	**19.8**	**19.8**	**19.8**	0.0	0.2	**0.0**	0.7
barman14 (14)	**20.2**	**20.2**	**20.2**	0.0	0.2	**0.0**	1.0
caldera18 (20)	0.0	235.6	**397.1**	0.0	41.7	**0.3**	31.2
cavediving14 (20)	938.4	**8 331.7**	(16) 1 068.1	140.8	**7.8**	9.1	49.8
childsnack14 (20)	**14 562.0**	**14 562.0**	**14 562.0**	213.7	**0.4**	0.8	37.5
floortile11 (20)	2.0	**3.1**	**3.1**	1.1	0.4	**0.0**	0.9
floortile14 (20)	0.9	**1.5**	**1.5**	0.4	0.2	**0.0**	0.2
hiking14 (20)	**0.1**	**0.1**	**0.1**	0.0	**0.1**	0.4	13.7
nomystery11 (20)	20 741.0	20 916.5	(19) **47 163.1**	0.0	9.4	**3.2**	56.8
nurikabe18 (14)	14 747.2	**19 313.1**	(11) 4 584.5	0.0	18.7	**4.6**	16.1
openstacks06 (30)	**3 957.6**	(27) 1 530.5	(23) 457.0	0.0	11.9	**6.7**	24.9
openstacks08 (30)	**340.3**	**340.3**	**340.3**	0.0	0.5	**0.1**	5.5
openstacks11 (20)	**178.9**	**178.9**	**178.9**	0.0	0.4	**0.1**	3.5
openstacks14 (20)	**1 078.0**	**1 078.0**	**1 078.0**	0.0	3.0	**0.8**	62.3
organic-synthesis18 (18)	(10) 0.3	(14) 27.0	(13) **209.6**	0.0	4.0	**0.0**	5.9
parcprinter08 (30)	**3.2**	**3.2**	**3.2**	0.0	0.9	**0.0**	0.1
parcprinter11 (20)	**2.2**	**2.2**	**2.2**	0.0	0.7	**0.0**	0.1
pathways06 (30)	**1 184.7**	**1 184.7**	**1 184.7**	1.1	1.0	**0.3**	16.3
pegsol08 (30)	1.7	1.7	**2.3**	0.0	0.2	**0.0**	0.1
pegsol11 (20)	0.5	0.5	**0.8**	0.0	0.3	**0.0**	0.1
petri-net-alignment18 (20)	(0) 0.0	**276.7**	**276.7**	0.0	0.0	**1.6**	133.1
pipesworld06 (50)	(33) 21.3	(41) **23.4**	(19) 18.4	0.0	59.8	**35.3**	54.2
rovers06 (40)	1 260.4	1 260.4	(27) 90.5	**3 928.4**	**6.4**	10.5	41.2
snake18 (20)	(13) 254.6	**1 096.2**	(0) 0.0	0.0	129.6	**36.1**	0.0
sokoban08 (30)	59.2	60.7	**61.5**	0.0	1.7	**0.1**	7.8
sokoban11 (20)	30.9	31.5	**32.0**	0.0	2.4	**0.1**	5.9
spider18 (20)	(12) 12 660.1	(19) **75 457.6**	(3) 187.8	0.0	**66.3**	73.2	126.5
tidybot11 (20)	1 234.9	**2 205.5**	(9) 12.2	0.0	**5.4**	19.9	30.3
tidybot14 (20)	2 428.7	**4 267.9**	(0) 0.0	0.0	**10.7**	38.2	0.0
tpp06 (30)	(27) 1 901.1	**3 782.2**	(20) 180.1	0.0	31.0	**4.3**	21.8
trucks06 (30)	1 684.4	1 693.3	**2 127.2**	1 498.9	0.6	**0.4**	28.6
woodworking08 (30)	48.9	49.7	**49.8**	0.0	0.5	**0.0**	1.2
woodworking11 (20)	29.3	30.0	**30.1**	0.0	0.5	**0.0**	1.0
overall (806)	(738) 139 255.6	(787) **583 264.2**	(644) 74 342.9	5 784.5	11.5	**9.4**	21.0

Table 9.1: Left: Number of inferred op-mutex pairs (in thousands) within a time limit of 5 minutes. If the method did not terminate in time, the number of successfully processed tasks is shown in parentheses. Maximums are highlighted, and overall is a sum. Right: Average time per instance in seconds with cut-off at 5 minutes. Minimums are highlighted, and overall is average over all instances.

9.7 Experimental Evaluation

The proposed methods were implemented[1] in C. Symmetries are computed using the BLISS library (Junttila & Kaski, 2007) on the Problem Description Graph (PDG) of the task (Pochter et al., 2011; Shleyfman et al., 2015). We used all IPC benchmarks 2006–2018 from the optimal track. Since our methods are described for STRIPS planning tasks without conditional effects, we compile them away (Nebel, 2000). We exclude all instances where the compilation of conditional effects or grounding of the task exceeded the time or memory limits. The experiments were performed on a cluster with Intel E5-2670 2.6 GHz processor with 8 GB memory limit for each task. We evaluated four methods for inference of op-mutexes:

- `fam` refers to the method based on abstractions (Theorem 9.9) used on projections to individual maximal fact-alternating mutex groups (fam-groups) (we used Algorithm 5.1 for the inference of fam-groups);
- Π^2_{op} refers to h^2 reachability analysis of the op-fact compilation (Theorem 9.11);
- `2-h`^{2} refers to running reachability on the 2-$\mathcal{M}$ compilation (Definition 9.13) with mutexes $\mathcal{M}$ obtained from the h^2 reachability with all operators (Theorem 9.14); and

[1] https://gitlab.com/danfis/cplan.git, branch aaai19

domain	**op-mutex pairs finished in 30 min.**			
	fam	Π^2_{op}	**2-h^2**	**iadd**
agricola18 (4)	3 478.2	21 065.9	**32 855.5**	0.0
barman11 (20)	**19.8**	**19.8**	**19.8**	0.0
barman14 (14)	**20.2**	**20.2**	**20.2**	0.0
caldera18 (20)	0.0	235.6	**397.1**	0.0
cavediving14 (16)	214.6	847.9	**1 068.1**	18.3
childsnack14 (20)	**14 562.0**	**14 562.0**	**14 562.0**	213.7
floortile11 (20)	2.0	**3.1**	**3.1**	1.1
floortile14 (20)	0.9	**1.5**	**1.5**	0.4
hiking14 (20)	**0.1**	**0.1**	**0.1**	0.0
nomystery11 (20)	20 741.0	20 916.5	**57 411.9**	0.0
nurikabe18 (14)	14 747.2	19 313.1	**32 476.7**	0.0
openstacks06 (25)	448.8	704.7	**958.8**	0.0
openstacks08 (30)	**340.3**	**340.3**	**340.3**	0.0
openstacks11 (20)	**178.9**	**178.9**	**178.9**	0.0
openstacks14 (20)	**1 078.0**	**1 078.0**	**1 078.0**	0.0
organic-synthesis18 (11)	0.3	3.6	**108.1**	0.0
parcprinter08 (30)	**3.2**	**3.2**	**3.2**	0.0
parcprinter11 (20)	**2.2**	**2.2**	**2.2**	0.0
pathways06 (30)	**1 184.7**	**1 184.7**	**1 184.7**	1.1
pegsol08 (30)	1.7	1.7	**2.3**	0.0
pegsol11 (20)	0.5	0.5	**0.8**	0.0
petri-net-alignment18 (0)	0.0	0.0	0.0	0.0
pipesworld06 (26)	20.7	**20.8**	**20.8**	0.0
rovers06 (31)	185.9	185.9	185.9	**477.5**
snake18 (10)	198.9	245.0	**255.6**	0.0
sokoban08 (30)	59.2	60.7	**61.5**	0.0
sokoban11 (20)	30.9	31.5	**32.0**	0.0
spider18 (6)	567.4	601.9	**636.0**	0.0
tidybot11 (12)	0.0	0.0	**35.0**	0.0
tidybot14 (8)	82.7	144.3	**244.9**	0.0
tpp06 (25)	1 056.1	1 056.1	**1 117.4**	0.0
trucks06 (30)	1 684.4	1 693.3	**2 127.2**	1 498.9
woodworking08 (30)	48.9	49.7	**49.8**	0.0
woodworking11 (20)	29.3	30.0	**30.1**	0.0
overall (672)	60 988.9	84 602.7	147 469.5	2 211.0

Table 9.2: Number of inferred op-mutex pairs (in thousands) in commonly solved tasks within a time limit of 30 minutes.

- `iadd` refers to looking for operators with a single irreversible add effect (Theorem 9.15).

Table 9.1 summarizes the results on the number of inferred op-mutex pairs (in thousands). We list only the domains in which at least one op-mutex pair was found. The results show that it is possible to infer a significant number of op-mutex pairs in many different domains (34 out of 83) even with a time limit of 5 minutes which is suitable for a preprocessing step. The average time of `iadd` is omitted because in all cases it was a fraction of a second. The most successful method was Π^2_{op} mainly because it offers a good trade-off between op-mutex pairs found and computational effort. The reason why `fam` was slower than Π^2_{op} on average is that in some cases the inference of fam-groups is slow, i.e., the inference of fam-groups is much slower then the inference of op-mutexes given the fam-groups. As expected, the slowest of all methods was `2-h`^{2}, because it requires to compute h^2 reachability for every operator.

In Table 9.2, we compare the strength of inference methods. It shows the number of op-mutex pairs found in instances where all methods successfully terminated within 30 minutes. In every case, the set of op-mutexes found by `fam` was a subset of those found by Π^2_{op}, and Π^2_{op} found a subset of `2-h`^{2}. Theoretical analysis of the dominance between these methods is left for future work. The results suggest that `2-h`^{2} can provide significantly better results than Π^2_{op}, but practical applicability requires a significant speed-up of the inference process (cf. Table 9.1). Promising lines of research, left for future work, seems to be the utilization of structural symmetries of h^2 previously studied by Röger et al. (2018), and some sort of clever selection of operators for which to construct m-$\mathcal{M}$-compilations.

The implementation of Algorithm 9.1 was experimentally evaluated as a standalone

domain					**h^2+de combined with**		
	h^2+de	**fam**	Π^2_{op}	**2-h^2**	**fam**	Π^2_{op}	**2-h^2**
barman11 (20)	**52.11**	4.71	24.48	20.65	**+3.60**	**+3.60**	**+3.60**
barman14 (14)	**53.43**	4.94	25.28	21.25	**+4.01**	**+4.01**	**+4.01**
caldera18 (19)	**39.34**	0.73	13.25	12.64	0.00	**+1.45**	+0.72
cavediving14 (5)	0.66	**1.15**	**1.44**	0.55	**+1.15**	**+1.15**	+0.55
childsnack14 (20)	0.00	**39.58**	**39.58**	**39.58**	+**39.58**	+**39.58**	+**39.62**
hiking14 (11)	0.00	**0.07**	**0.07**	**0.07**	**+0.07**	**+0.07**	**+0.07**
parcprinter08 (24)	**70.68**	28.61	28.61	28.61	0.00	0.00	0.00
parcprinter11 (17)	**70.15**	25.75	25.75	25.75	0.00	0.00	0.00
pathways06 (30)	**3.94**	2.22	2.26	2.26	**+3.73**	**+3.73**	**+3.73**
pegsol08 (9)	**13.57**	0.00	1.92	1.92	**+0.36**	**+0.36**	**+0.36**
pegsol11 (8)	**9.53**	1.08	3.51	3.45	**+0.20**	**+0.20**	**+0.20**
pipesworld06 (17)	**11.46**	0.04	4.04	4.04	**+0.04**	**+0.04**	**+0.04**
scanalyzer08 (11)	**3.19**	0.00	2.10	2.10	0.00	0.00	0.00
scanalyzer11 (8)	**3.17**	0.00	2.12	2.12	0.00	0.00	0.00
sokoban08 (6)	0.24	1.54	**1.71**	1.65	+1.54	**+1.60**	**+1.60**
sokoban11 (4)	0.35	1.86	**2.12**	2.04	+1.86	**+1.95**	**+1.95**
tpp06 (15)	**37.96**	1.49	19.97	18.97	**+1.79**	**+1.79**	+1.49
trucks06 (12)	**75.02**	0.00	0.71	0.71	**+7.87**	**+7.87**	**+7.87**
woodworking08 (24)	**53.47**	2.83	10.79	10.79	**+1.60**	**+1.60**	**+1.60**
woodworking11 (18)	**53.72**	2.73	10.17	10.17	**+1.83**	**+1.83**	**+1.83**
overall (292)	**53.72**	7.99	13.73	13.17	+4.29	**+4.39**	+4.32

Table 9.3: Average percentage of removed operators in each domain. The number in the parenthesis after the domain name is the number of instances in which all methods successfully terminated before the 15 minutes time limit and at least one operator was pruned by at least one method; the left part shows the average percentage of removed operators in those instances; and the right part shows the increase when `h`^{2}`+de` is combined with the methods using op-mutexes. The last row (overall) shows averages over all instances. Maximums are highlighted in bold.

preprocessing step and in combination with the pruning using forward/backward h^2 (Alcázar & Torralba, 2015) and the detection of dead-end operators using fam-groups (Algorithm 6.1) (we will refer to the combination of the two as `h`^{2}`+de`). Table 9.3 shows the average percentage of removed operators within each domain and over all evaluated instances. Only the instances in which all methods successfully finished within the 15 minutes time limit are listed. In all domains except cavediving, childsnack, hiking, and sokoban, `h`^{2}`+de` prunes more operators than any of our methods, but combining `h`^{2}`+de` with our methods further increases the number of removed operators as shown in the right part of the table.

Table 9.4 encapsulates the results from combining `h`^{2}`+de` with our pruning methods in absolute numbers. We set the baseline (the column `base`) as the number of operators after the grounding and `h`^{2}`+de` pruning. The remaining columns contain the number of additionally removed operators after a subsequent application of Algorithm 9.1 using the op-mutexes found by our methods and `h`^{2}`+de` pruning until a fixpoint is reached. We do not combine `fam`, Π^2_{op}, and `2-h`^{2} methods because of the observed dominance between them. We also exclude `iadd`, because by itself it managed to prune only a third of the operators of that of the remaining three methods, and if combined with any of the remaining three, the results were, surprisingly, worse, probably because of the greedy selection steps in Algorithm 9.1. The results show that op-mutexes can be combined with symmetries to simplify planning tasks in several domains. In some domains, the reduction in the number of operators is quite significant, especially in childsnack, agricola, trucks, and barman. Finding op-mutexes with Π^2_{op} achieves the best results in most cases.

We also tried the most promising Π^2_{op} pruning technique with the Fast Downward planner (Helmert, 2006). We used A* with the LM-Cut (`lmc`) heuristic (Helmert & Domshlak, 2009), the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy and non-greedy

domain	**base**	**fam**	Π^2_{op}	**2-h^2**
agricola18 (17)	251 306	(15) **−5 108**	(12) −3 242	(0) 0
barman11 (20)	7 408	**−554**	**−554**	**−554**
barman14 (14)	6 010	**−522**	**−522**	**−522**
caldera18 (20)	24 178	0	**−492**	−220
cavediving14 (20)	91 832	**−74**	**−74**	(16) −37
childsnack14 (20)	53 698	**−21 660**	**−21 660**	−21 624
hiking14 (20)	35 822	**−11**	**−11**	**−11**
organic-synthesis18 (13)	47 614	(12) −27	−432	(12) **−1 095**
pathways06 (30)	40 595	**−731**	**−731**	**−731**
pegsol08 (30)	4 392	**−6**	**−6**	**−6**
pegsol11 (20)	3 320	**−3**	**−3**	**−3**
pipesworld06 (42)	1 122 448	(37) −69	**−121**	(22) −27
sokoban08 (30)	12 637	−73	**−74**	**−74**
sokoban11 (20)	7 139	−45	**−46**	**−46**
tpp06 (30)	107 409	(25) −1 147	**−2 167**	(22) −429
trucks06 (21)	22 769	−4 673	−4 673	**−4 679**
woodworking08 (30)	12 535	**−400**	**−400**	**−400**
woodworking11 (20)	8 159	**−307**	**−307**	**−307**
Σ (417)	1 859 271	(404) −35 410	(412) **−35 515**	(367) −30 765

Table 9.4: Number of operators pruned over the baseline `h`^{2}`+de` within a time limit of 15 minutes. The number in parenthesis after the domain name is the number of instances in which at least one method successfully finished within the time limit. For methods that finished in less instances, their number of successfully finished instances is indicated in parenthesis.

bi-simulation shrink strategy (Helmert et al., 2014; Sievers et al., 2016), and the potential (`pot`) heuristic optimized for all syntactic states (Seipp et al., 2015). The time limit was set to 30 minutes for the whole planning process.

When we set the Π^2_{op} pruning phase to abstain from the inference of op-mutexes if no symmetries were found, the planners solved all planning tasks that were solved without Π^2_{op} except of one task in organic-synthesis for all three heuristics, and one task in scanalyzer for `lmc`. In these cases, the inference of op-mutexes took too long (organic-synthesis) or exceeded the memory limit (scanalyzer). However, despite the size of many instances is reduced by the pruning, this was not substantially reflected on the coverage. Only in the agricola domain, `pot` with Π^2_{op} was able to solve two more tasks, and `m&s` one more task. The reductions had also a negligible effect on the number of expanded states in most instances. In childsnack, which is the domain where more operators are pruned by our method, we measured about twice as many expanded states per second by `lmc` with Π^2_{op}. However, no planner solved any instance in this domain.

9.8 Summary

We introduced a new notion of strong operator mutexes (op-mutexes) as sets of operators that cannot be part of the same strongly optimal plan and proposed four different methods for inference of op-mutexes. We proved that every op-mutex contains at least one operator that can be safely removed from the planning task and we have experimentally evaluated that they can be found in a sizable amount of planning domains. Combining op-mutexes with structural symmetries provides further information about which operators can actually be removed, and we showed that there are some domains where removing operators yields a significant reduction in the size of the planning tasks. Even though our experiments show that this reduction does not translate into significant gains for heuristic search planners, this opens new avenues of research on how to leverage operator mutexes to simplify planning tasks.

Chapter 10

Conclusion

This thesis aims at theoretical analysis and a study of practical applicability of mutual exclusion state invariants in the context of classical planning. We analyzed state invariants called mutexes (describing facts that cannot co-occur in the same reachable state) and mutex groups (describing facts out of which at most one can be part of any reachable state). We showed that the relation between mutex pairs (mutexes consisting of exactly two facts) and mutex groups can be described in terms of cliques in graphs and we described a conceptually simple procedure that can be used to infer mutex groups given a set of mutex pairs. To describe the inference of mutex groups in terms of computational complexity, we proved that the inference of the maximum sized mutex groups is as hard as deciding the existence of a plan, i.e., it is PSPACE-Complete. This also means that the inference of a complete set of mutex pairs is also PSPACE-Complete.

Observing the hardness of the inference of mutex groups motivated the search for a provably "easier" subclass of mutex groups. We introduced the notion of fact-alternating mutex groups (fam-groups) that are defined over the input planning task rather than over the reachable states. We proved that every fam-group is a mutex group such that an operator can switch one fact for another from the same fam-group, or it can delete the fact, but once we reach a state s that does not contain any fact from the fam-group, all other states reachable from s cannot contain any fact from that fam-group. In this sense, facts from a fam-group alternate between each other on the path from the initial state and once facts from the fam-group disappear from the state they cannot reappear again in any following state. We showed that this property of fam-groups can be used to determine not only unreachable operators (as is possible with the general mutex groups), but also operators that can reach only dead-end states (dead-end operators).

We proved that the inference of the maximum sized fam-group is NP-Complete and we proposed and evaluated the inference algorithm based on repeated solving of integer linear program that is complete with respect to all maximal fam-groups. The evaluation showed that fam-groups are information-rich invariants dominating state-of-the-art inference algorithm proposed by Helmert (2009). Furthermore, we demonstrated that using fam-groups for the construction of Finite Domain Representation (FDR) from the STRIPS representation results in a more concise representation which in turn has a potential to increase the number of solved tasks.

To pinpoint the complexity of fam-groups more precisely, we investigated the relation between fam-groups and the h^m heuristic (Haslum & Geffner, 2000). We proved that every pair of facts from every fam-group is a mutex pair obtained from the h^2 heuristic, but not necessarily the other way around. This result connects the NP-Completeness of fam-groups and relation between mutex pairs and mutex groups in terms of graph cliques, because h^2 can be computed in a polynomial number of steps.

The ability of fam-groups to detect dead-end operators was utilized in a novel pruning algorithm which removes operators and facts that cannot be part of any plan. We showed that this pruning algorithm substantially reduces planning tasks from some domains which results in more solved tasks. We also showed that fam-groups can improve the state-of-the-art h^2-based pruning technique proposed by Alcázar & Torralba (2015) that uses reachability analysis in both progression and regression.

The experimental results showing that fam-groups dominate mutex groups inferred by the algorithm proposed by Helmert (2009) motivated a further investigation into the relation between Helmert's mutex groups and fam-groups. Helmert's algorithm works on the lifted representation (PDDL). We formalized the inference of mutex groups in the lifted representation and we proved that every mutex group inferred by the Helmert's algorithm is in fact fam-group after grounding. Moreover, we proposed an improvement of the Helmert's algorithm which allows to generate a richer set of lifted fam-groups. We also applied our previous findings regarding the pruning of planning tasks using fam-groups to the lifted level. We proposed a pruning algorithm on the lifted level that utilizes the fact that the inferred mutex groups are lifted fam-groups. This allows us to disregard unreachable and dead-end operators even before the ground representation is generated from the lifted (PDDL) input. Experimental evaluation then verified that this leads to an increased number of solved tasks on the standard benchmark set.

Mutual exclusion invariants turned out to be useful also in the context of potential heuristics (Pommerening et al., 2015a; Seipp et al., 2015). We used the notion of disambiguation (Alcázar et al., 2013), originally applied in the context of regression planning, to show how mutex groups (in the form of FDR variables) and mutexes can be combined in order to improve heuristic estimates of potential heuristics with significant positive impact on the number of solved tasks. Moreover, we showed that mutexes can be used for a more accurate approximation of the number of reachable states which in turn can be used for a construction of optimization functions for potential heuristics.

In the last part of the thesis, we extended the notion of mutual exclusion onto operators. We defined strong operator mutex (op-mutex) as a set of operators such that there is no strongly optimal plan where operators from the op-mutex co-occur. We described four different methods for the inference of op-mutexes based on well-known planning techniques and we experimentally verified that op-mutexes can be found in many different domains. Unfortunately, it turns out that op-mutexes do not tell us directly which operators can be safely removed from the planning task. So we combined op-mutexes with structural symmetries (e.g., Fox & Long, 1999; Pochter et al., 2011; Shleyfman et al., 2015), previously studied in different contexts, and we showed how to use them to find out which operators can be removed so that at least one strongly optimal plan is preserved. In contrast to the previous method for reducing planning tasks, this method is able to remove even the operators that are reachable, because we make sure that if the planning task is solvable, then at least one strongly optimal plan remains in the resulting planning task.

Appendix A

Publications

Below are listed all author's publications. Each related publication is acompanied with the percentage contribution of the author as agreed by all co-authors. Number of citations is based on Web of Science (WoS), Scopus, and Google Scholar (GS). Journal articles also contain information about the Impact Factor (IF) and 5-year Impact Factor (IF5y) by Thomson Reuters, and the CiteScore (CS) by Scopus.

Related Publications

This section lists author's publications related to the topic of this thesis.

Journals with impact factor:

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

   80% contribution, IF 1.820, IF5y 2.448, CS 3.23, citations: 1 in WoS, 1 in Scopus, 4 in GS

Conferences A* (CORE) (indexed by WoS):

2. **Fišer, D.** & Torralba, Á. & Shleyfman, A. (2019). Operator mutexes and symmetries for simplifying planning tasks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, (pp. 7586–7593).

   60% contribution, citations: 0 in WoS, 2 in GS

Conferences A* (CORE):

3. **Fišer, D.** & Horčík, R. & Komenda, A. (2020). Strengthening potential heuristics with mutexes and disambiguations. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, (Accepted).

   60% contribution, citations: 0 in GS

4. **Fišer, D.** (2020). Lifted fact-alternating mutex groups and pruned grounding of classical planning problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, (Accepted).

   100% contribution, citations: 2 in GS

5. **Fišer, D.** & Komenda, A. (2018). Fact-alternating mutex groups for classical planning (extended abstract). In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, (pp. 5603–5607).

   80% contribution, citations: 0 in Scopus

Unrelated Publications

The following papers were published during author's studies, but they are not covered by this thesis.

Journals with impact factor:

6. **Fišer, D.** & Faigl, J. & Kulich, M. (2013). Growing neural gas efficiently. *Neurocomputing*, 104, 72–82.

   IF 4.072, IF5y 3.824, CS 5.00, citations: 17 in WoS, 17 in Scopus, 29 in GS

Conferences A* (CORE) (indexed by WoS):

7. Štolba, M. & **Fišer, D.** & Komenda, A. (2016). Potential heuristics for multi-agent planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*, (pp. 308–316).

   citations: 2 in WoS, 6 in Scopus, 10 in GS

8. Štolba, M. & **Fišer, D.** & Komenda, A. (2015). Admissible landmark heuristic for multi-agent planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, (pp. 211–219).

   citations: 2 in WoS, 11 in Scopus, 16 in GS

Conferences A* (CORE):

9. Štolba, M. & **Fišer, D.** & Komenda, A. (2019). Privacy leakage of search-based multi-agent planning algorithms. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*, (pp. 482–490).

   citations: 2 in GS

Others (indexed in WoS):

10. Vonásek, V. & **Fišer, D.** & Košnar, K. & Přeučil, L. (2014). A light-weight robot simulator for modular robotics. In *Proceedings of the 1st International Workshop on Modeling and Simulation for Autonomous Systems (MESAS)*, (pp. 206–216).

    citations: 3 in WoS, 4 in Scopus, 8 in GS

11. Krajník, T. & Vonásek, V. & **Fišer, D.** & Faigl, J. (2011). AR-Drone as a platform for robotic research and education. In *Proceedings of the International Conference on Research and Education in Robotics (EUROBOT 2011)*, (pp. 172–186).

    citations: 96 in WoS, 133 in Scopus, 314 in GS

Others:

12. Štolba, M. & Urbanovská, M. & **Fišer, D.** & Komenda, A. (2019). A general approach to distributed and privacy-preserving heuristic computation. In *Proceedings of the Agents and Artificial Intelligence - 11th International Conference, ICAART, Revised Selected Papers*, (pp. 55–71)

    citations: 0 in Scopus

13. Štolba, M. & Urbanovská, M. & **Fišer, D.** & Komenda, A. (2019). Cost partitioning for multi-agent planning. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence*, (pp. 44–49)

    citations: 1 in Scopus, 1 in GS

14. **Fišer, D.** & Komenda, A. (2018). Concise finite-domain representations for factored MA-PDDL planning tasks In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence*, (pp. 306–313)

    citations: 1 in GS

15. Štolba, M. & **Fišer, D.** & Komenda, A. (2015). Comparison of RPG-based FF and DTG-based FF distributed heuristics. In *Proceedings of the Distributed and Multi-Agent Planning (DMAP-15)*, (pp. 77–82).

    citations: 5 in GS

16. **Fišer, D.** & Štolba, M. & Komenda, A. (2015). MAPlan. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, (pp. 8–10)

    citations: 13 in GS

17. Vonásek, V. & Kulich, M. & Krajník, T. & Saska, M. & **Fišer, D.** & Petrík, V. & Přeučil, L. (2012). Techniques for modelling simulation environments for modular robotics. FAC Proceedings Volumes, 45, 210–215.

    citations: 6 in GS

18. Saska, M. & Vonásek, V. & Kulich, M. & **Fišer, D.** & Krajník, T. & Přeučil, L. (2011). Bringing reality to evolution of modular robots: bio-inspired techniques for building a simulation environment in the SYMBRION project. In: *Proceedings of the Reconfigurable Modular Robotics: Challenges of Mechatronic and Bio-Chemo-Hybrid Systems*, (pp. 1–6).

    citations: 4 in GS

Bibliography

Alcázar, V., Borrajo, D., Fernández, S., & Fuentetaja, R. (2013). Revisiting regression in planning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, (pp. 2254–2260).

Alcázar, V. & Torralba, Á. (2015). A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, (pp. 2–6).

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

Bernardini, S., Fagnani, F., & Smith, D. E. (2018). Extracting mutual exclusion invariants from lifted temporal planning domains. *Artificial Intelligence*, *258*, 1–65.

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'13)*, (pp. 2268–2274).

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

Bonet, B. & van den Briel, M. (2014). Flow-based heuristics for optimal planning: Landmarks and merges. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, (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. *Artificial Intelligence*, *69*(1–2), 165–204.

Cazals, F. & Karande, C. (2008). A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, *407*(1-3), 564–568.

Chen, Y., Huang, R., Xing, Z., & Zhang, W. (2009). Long-distance mutual exclusion for planning. *Artificial Intelligence*, *173*(2), 365–391.

Chen, Y., Xing, Z., & Zhang, W. (2007). Long-distance mutual exclusion for propositional planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, (pp. 1840–1845).

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*.

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

Domshlak, C., Katz, M., & Shleyfman, A. (2012). Enhanced symmetry breaking in cost-optimal planning as forward search. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, (pp. 343–347).

Domshlak, C., Katz, M., & Shleyfman, A. (2013). Symmetry breaking: Satisficing planning and landmark heuristics. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, (pp. 298–302).

Edelkamp, S. (2001). Planning with pattern databases. In *Proceedings of the 6th European Conference on Planning (ECP'21)*, (pp. 13–24).

Eriksson, S., Röger, G., & Helmert, M. (2017). Unsolvability certificates for classical planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, (pp. 88–97).

Eriksson, S., Röger, G., & Helmert, M. (2018). A proof system for unsolvable planning tasks. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, (pp. 65–73).

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

Fišer, D. (2020). Lifted fact-alternating mutex groups and pruned grounding of classical planning problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*. Accepted.

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

Fišer, D., Torralba, Á., & Shleyfman, A. (2019). Operator mutexes and symmetries for simplifying planning tasks. In *Proceedings of the 33nd AAAI Conference on Artificial Intelligence (AAAI'19)*, (pp. 7586–7593).

Fox, M. & Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, *9*, 367–421.

Fox, M. & Long, D. (1999). The detection and exploitation of symmetry in planning problems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, (pp. 956–961).

Franco, S., Lelis, L. H. S., & Barley, M. (2018). The Complementary2 planner in IPC 2018. In *IPC 2018 planner abstracts*, (pp. 32–36).

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

Franco, S., Torralba, A., Lelis, L. H., & Barley, M. (2017). On creating complementary pattern databases. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, (pp. 4302–4309).

Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman.

Gerevini, A. & Schubert, L. (1998). Inferring state-constraints for domain independent planning. In *Proceedings of the 15th National Conference of the American Association for Artificial Intelligence (AAAI'98)*, (pp. 905–912).

Gerevini, A. & Schubert, L. (2000). Inferring state constraints in DISCOPLAN: Some new results. In *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI'00)*, (pp. 761–767).

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'09)*, (pp. 354–357).

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, (pp. 1007–1012).

Haslum, P. & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, (pp. 140–149).

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

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

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'09)*, (pp. 162–169).

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, (pp. 176–183).

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

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

Huang, R., Chen, Y., & Zhang, W. (2012). SAS+ planning as satisfiability. *Journal of Artificial Intelligence Research*, *43*, 293–328.

Junttila, T. & Kaski, P. (2007). Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, (pp. 135–149).

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

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

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

Moraru, I., Edelkamp, S., Martinez, M., & Franco, S. (2018). Planning-pdbs planner. In *IPC 2018 planner abstracts*, (pp. 69–73).

Mukherji, P. & Schubert, L. K. (2005). Discovering planning invariants as anomalies in state descriptions. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, (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).

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, *12*, 271–315.

Pochter, N., Zohar, A., & Rosenschein, J. S. (2011). Exploiting problem symmetries in state-based planners. In *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*.

Pommerening, F. & Helmert, M. (2015). A normal form for classical planning tasks. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, (pp. 188–192).

Pommerening, F., Helmert, M., & Bonet, B. (2016). Higher-dimensional potential heuristics for optimal classical planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, (pp. 3636–3643).

Pommerening, F., Helmert, M., Röger, G., & Seipp, J. (2015a). From non-negative to general operator cost partitioning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, (pp. 3335–3341).

Pommerening, F., Helmert, M., Röger, G., & Seipp, J. (2015b). From non-negative to general operator cost partitioning: Proof details. Technical Report CS-2014-005, University of Basel, Department of Mathematics and Computer Science.

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

Riddle, P. J., Barley, M. W., Franco, S., & Douglas, J. (2015). Automated transformation of PDDL representations. In *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, (pp. 214–215).

Rintanen, J. (2000). An iterative algorithm for synthesizing invariants. In *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI'00)*, (pp. 806–811).

Rintanen, J. (2003). Symmetry reduction for SAT representations of transition systems. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, (pp. 32–41).

Rintanen, J. (2008). Regression for classical and nondeterministic planning. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, (pp. 568–572).

Rintanen, J. (2012). Planning as satisfiability: Heuristics. *Artificial Intelligence*, *193*, 45–86.

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

Röger, G., Sievers, S., & Katz, M. (2018). Symmetry-based task reduction for relaxed reachability analysis. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, (pp. 208–217).

Seipp, J. (2018). Fast downward scorpion. In *IPC 2018 planner abstracts*, (pp. 77–79).

Seipp, J. & Helmert, M. (2018). Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, *62*, 535–577.

Seipp, J., Pommerening, F., & Helmert, M. (2015). New optimization functions for potential heuristics. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, (pp. 193–201).

Seipp, J., Pommerening, F., Röger, G., & Helmert, M. (2016). Correlation complexity of classical planning domains. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, (pp. 3242–3250).

Shleyfman, A., Katz, M., Helmert, M., Sievers, S., & Wehrle, M. (2015). Heuristics and symmetries in classical planning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, (pp. 3371–3377).

Sievers, S., Wehrle, M., & Helmert, M. (2014). Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, (pp. 2358–2366).

Sievers, S., Wehrle, M., & Helmert, M. (2016). An analysis of merge strategies for merge-and-shrink heuristics. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*, (pp. 294–298).

Sievers, S., Wehrle, M., Helmert, M., & Katz, M. (2017). Strengthening canonical pattern databases with structural symmetries. In *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*, (pp. 91–99).

Sievers, S., Wehrle, M., Helmert, M., Shleyfman, A., & Katz, M. (2015). Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, (pp. 3378–3385).

Tomita, E., Tanaka, A., & Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, *363*(1), 28–42.

van den Briel, M., Benton, J., Kambhampati, S., & Vossen, T. (2007). An LP-based heuristic for optimal planning. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, (pp. 651–665).