A Reminder about the Importance of Computing and Exploiting Invariants in Planning

Vidal Alcázar
vidal.alcazar.saiz@gmail.com
Universidad Carlos III de Madrid, Madrid, Spain

Álvaro Torralba
torralba@cs.uni-saarland.de
Saarland University, Saarbrücken, Germany

Abstract

Throughout the years, extensive work has pointed out how important computing and exploiting invariants is in planning. However, no recent studies about their empirical impact regarding their ability to simplify and/or complete the model have been done. In particular, an analysis about the impact of invariants computed in regression from the goals is severely lacking, despite the existence of previous attempts to use this kind of invariants in different planning settings. In this work we focus on the ability of invariants to simplify the planning task in a preprocessing step. Our results show that this simplification significantly improves the performance of different optimal and satisficing planners.

Introduction

The invariants of a planning problem are relationships between the entities of the task that are true in the definition of the initial state and the goal condition (if the problem has a solution) and whose truth is preserved by the application of the actions of the problem. The importance of invariants has been highlighted by many authors throughout the years. First, different methods to compute invariants have been proposed (Gerevini and Schubert 1998; Fox and Long 1998; Rintanen 2000; 2008; Helmert 2009); second, the benefits of exploiting the information provided by invariants have proven to be essential in many planning paradigms. Examples of successful uses of invariants are partial-order planning (Penberthy and Weld 1992), planning graphs (Blum and Furst 1997), backward search (Bonet and Geffner 2001), planning as SAT (Chen, Xing, and Zhang 2007; Huang, Chen, and Zhang 2012) and multi-valued formulations of planning (Edelkamp and Helmert 1999; Helmert 2009).

However, and despite the substantial amount of publications dedicated to the study of invariants, some areas remain unclear. In particular, the computation of binary static mutexes using a reachability analysis in P^m (Haslum 2009), arguably the most common form of invariant computation, and the impact of a fixpoint computation of invariants, have not been empirically studied in isolation. In addition, some major aspects, like the direction of the computation of the invariants, appear scattered in the literature. Hence, in this work we analyze some aspects of the computation of invariants, focusing on important details of the implementation and the direction of the computation. The aim of this work is to propose the use of invariants as a general method to simplify the model of the planning task (Haslum 2007), orthogonal to the techniques used to solve it. The empirical results obtained show a noticeable improvement in coverage for very different planners with little to no drawbacks.

Background

Classical planning tasks are formalized as a STRIPS problem (Fikes and Nilsson 1971) or a multi-valued problem. The two most common multi-valued formalizations are SAS$^+$ (Bäckström and Nebel 1995) and the Finite-Domain Representation (Helmert 2006). In this work we will use SAS$^+$, although we will make reference to the STRIPS formalization whenever it is relevant. A planning task in SAS$^+$ is defined as a tuple Π=($\mathcal{V}$,s_0,$s_\star$,$\mathcal{O}$). $\mathcal{V}$ is a set of state variables, and every variable $v \in \mathcal{V}$ has an associated extended domain $D_v^+ = D_v \cup \{\mathbf{u}\}$ composed of the regular domain of each variable, D_v, and the undefined value $\mathbf{u}$ (used to denote unknown values). The value of a variable $v \in \mathcal{V}$ in a given state s, also known as *fluent*, is defined as $s[v]$. s_0 is the initial state. $s_\star$ is the (commonly partial) state that defines the goals. $\mathcal{O}$ is a set of operators (actions), where each operator is a tuple $o = (pre(o), post(o), prev(o))$, where $pre(o), post(o), prev(o)$ represent the *pre-, post-* and *prevail-conditions* respectively. A solution plan $(a_1, a_2, \ldots, a_n)$ where $a_i \in \mathcal{O}$ is related to a sequence of states $(s_0, s_1, s_2, \ldots, s_n)$ *s.t.* $s_\star \subseteq s_n$ and s_i results from executing the action a_i in s_{i-1}, $\forall i = 1..n$.

State invariants and Spurious States/Actions

A state invariant is a logical formula over the fluents of a state that must hold in every state that may belong to a solution path. We will work with two types of state invariants: relations of mutual exclusivity (mutex) and *"exactly-1"* invariant groups.

Definition 1. *(Mutex) A set of fluents $M = \{f_1, \ldots, f_m\}$ is a set of mutually exclusive fluents of size m (mutex of size m) if there is no state $s \subseteq S$ that may belong to a solution path such that $M \subseteq s$.*

This differs from the original definition of mutex by Bonet and Geffner (2001) in that it does not depend on s_0. While the original definition is suitable to identify states unreachable from s_0, it does not contemplate other cases, as unreachable states from $s_\star$ (which would correspond to forward dead ends) or spurious states detected using other types of invariants.

The most common method to find mutexes is the h^m heuristic (Bonet and Geffner 2001). h^m performs a reachability analysis in P^m (Haslum 2009), a semi-relaxed version of the original problem in which atoms are actually sets of m fluents. If the value of h^{max} of an atom in P^m is infinite (it is unreachable in P^m) then that atom is a mutex of size m. The time needed to compute h^m grows exponentially with m, so we will consider only mutexes of size two.

Definition 2. *(*"exactly-1" *invariant group) An* "exactly-1" *invariant group is a set of fluents $I_g = \{f_1, \ldots, f_n\}$ such that $\forall p_m = \{f_i, f_j\} | f_i, f_j \in I_g \wedge f_i \neq f_j$, p_m is mutex, and $(f_1 \vee \ldots \vee f_n)$ is a state invariant.*

All the variables $v \in \mathcal{V}$ are *"exactly-1"* invariant groups, but not all the *"exactly-1"* invariant groups of the problem need to be variables. Groups of fluents such that all the elements are pairwise mutex can be completed with an additional fluent $\langle$*none of those*$\rangle$ to turn them into *"exactly-1"* invariant groups (Helmert 2009).

Definition 3. *(Spurious state/action) A spurious state is a state that cannot be part of a solution path. A spurious action is an action that cannot be part of a solution plan.*[1]

A state that violates a state invariant is spurious and hence can be safely pruned. An action whose preconditions or effects violate a state invariant is also spurious and can also be safely discarded.

Implementation Details of h^2

Due to its simplicity and efficiency, h^2 is the most popular invariant computation method. There exist generalizations of h^2 (Rintanen 2008), but for classical planning h^2 is still widely used. In spite of this, there are important details that affect its behavior.

Negated Atoms h^2's original definition is based on STRIPS. This does not represent explicitly the negation of the propositions, which may be problematic, *e.g.*: in the *Matching-Blocksworld* domain, a block may become non-solid, in which case no other block can be stacked on top of it. Solid blocks are represented by the predicate *(solid ?x - block)*, so a non-solid block is identified by having the corresponding proposition as false. Because of this, a mutex like {*on a b*), ¬(*solid b)*} will not be found by h^2, with a potentially high detrimental impact. Yet, not all the propositions need to have their negation explicitly represented.

Theorem 1. *Only the propositions that do not belong to an* "exactly-1" *invariant group must have their negation explicitly represented to ensure that all spurious states detectable with binary mutexes can be found.*

Proof by contradiction. Let p, q be propositions such that $(\neg p \wedge q)$ is a mutex. Let $I_g = \{p, f_1, \ldots, f_n\}$ be an *"exactly-1"* invariant group such that $p \in I_g$. $\neg p \implies (f_1 \vee \ldots \vee f_n)$, so whenever $(\neg p \wedge q)$ is true then $((q \wedge f_1) \vee \ldots \vee (q \wedge f_n))$ is true. Therefore, $\{(q \wedge f_1), \ldots, (q \wedge f_n)\}$ are mutexes too and whenever $(\neg p \wedge q)$ is true, at least one of those mutexes is violated. □

In SAS$^+$, fluents that do not belong to an *"exactly-1"* invariant group will be grouped with a $\langle$*none of those*$\rangle$ fluent. Therefore, the negation of the fluents of a SAS$^+$ task does not need to be explicitly represented, as all the fluents will be part of an *"exactly-1"* invariant group.

Unreachable Actions Every action that has not been applied in h^2 is spurious, as it is unreachable in P^2 and necessarily contains mutex preconditions. Spurious actions will not be applicable forward, but they may be applicable backwards or be otherwise used in other parts of the planner, like the heuristic computation.

Encoding Extra Information in the Actions Other invariant computation methods use the invariants found to enrich the description of the actions and find new invariants with a fixpoint computation (Gerevini and Schubert 1998; Rintanen 2000; 2008). This is applicable to h^2 too. A general way to do this is to *disambiguate* the actions (Alcázar et al. 2013). This consists of creating a CSP in which the *"exactly-1"* invariant groups are the variables, the fluents that belong to them are their domain, and the mutexes are the constraints. Departing from a partial state, constraint propagation is enforced. If the domain of some variable becomes empty, the partial state is spurious. If the domain of some variable has a single valid value, it can be encoded explicitly in the partial state. This can be done with both the effects and preconditions of the actions, and generalizes over other methods such as *e-deletion* (Vidal and Geffner 2005). If the preconditions or effects of an operator are inconsistent with the state invariants, it can be removed. This includes all the actions unreachable in h^2, but it may detect more.

State Invariants in Regression

Current invariant computation methods work in progression starting from s_0 and inferring new information forward. However, this can be also done from $s_\star$ in regression, inferring new information backward: imagine a problem with a truck, with fuel encoded in a discrete fashion using a sequence of fluents. The truck must arrive to its destination, and moving from one location to an adjacent one uses one unit of fuel. By regressing over $s_\star$, one can easily infer that at the locations at distance 1 of the destination the truck must have at least one unit of fluent, at the locations at distance 2, 2 units, and so on. This implication can be encoded with a set of mutexes. For example, being at a location at distance 2 of the destination and having 1 unit of fuel is a mutex.

Forward search planners can prune spurious states that violate mutexes computed backward, *e.g.*: if a forward search planner finds a mutex "*truck at some location*" and "*fuel*

[1] Again, this does not depend on s_0 for the aforementioned reasons.

equals n", that state can be pruned, as the truck will not be able to make it to its destination. This is the reason why Definition 1 does not depend exclusively on s_0, as mutexes can be computed in either direction. Invariants computed in regression can be used normally in any planning paradigm, but only the invariants computed in the opposite direction matter when pruning states during state space search.

Theorem 2. *An invariant computed in a given direction cannot be violated by a state generated in that direction.*

Proof sketch. The invariants of the relaxation of a problem are a subset of the invariants of the original problem. Invariants hold through the application of actions. Hence, all the invariants respected in a given direction in a relaxed problem will hold in the same direction in the original one. □

Backward invariants can be computed using a reversed formulation R(Π) of the original problem Π, initially proposed by Massey (1999) and later revisited by Petterson (2005) and Suda (2013). This reversed problem is computable in polynomial time and space, and every solution plan of R(Π) corresponds univocally to a solution plan of Π. s_0 in R(Π), which corresponds to $s_\star$ in Π, is commonly a partial state though. In order to obtain as much information as possible from R(Π), it is advisable to disambiguate s_0 in R(Π) with information from forward invariants.

Regarding the actual computation of invariants in regression, this is not a novel idea. Petterson already used R(Π) to compute a reversed planning graph, which yields the set of static binary mutexes obtained by backward h^2 when it levels off. Also, Haslum (2008) explicitly implemented h^2 on R(Π) to use it as an admissible heuristic for a forward heuristic planner. Nevertheless, neither author explored this line as a general way to obtain invariants, nor reported the amount of mutexes found or the number of spurious actions discarded. Performing h^2 in regression dominates a backwards reachability analysis (which is nothing but h^{max} in regression) and has the same properties as h^2 in progression.

Since in this work we use SAS$^+$, a few points must be clarified. First, we do not encode the negation of fluents explicitly. Second, the reversed operators are substantially simpler to obtain: the preconditions of the action become the postconditions and *vice versa*, while the prevail conditions remain the same. In the computation of h^2 backwards, though, we have to deal with undefined preconditions, *i.e.* a variable $v \in \mathcal{V}$ and an action $o \in \mathcal{O}$ such that $pre(o)[v] = \{\mathbf{u}\}$ and $post(o)[v] \neq \{\mathbf{u}\}$. In this case, any value of D_v should be considered as a potential postcondition in regression, which corresponds to an *add* effect of the action in h^2. However, by disambiguating the action some values of D_v may be discarded, obtaining a tighter set of *add* effects (for instance, if a fluent $p \in D_v$ is mutex with another fluent p' such that $p' \in pre(o)$, then p does not need to be an *add* effect in regression). Note that these potential postconditions in regression cannot be used to infer *delete* effects straightforwardly by adding as a *delete* any fluent mutex with them: unless a fluent is mutex with *all* the potential preconditions of D_v, it cannot be added as a *delete* (following the former example, if $D_v = \{p, q, r\}$ and p was discarded, a fluent p'' can only be added as a *delete* effect if p'' is mutex with *both* q and r).

Algorithm 1: Fixpoint computation of invariants.

```
fw ← True, updatedFW ← True, updatedBw ← True
while updatedFW ∨ updatedBW do
    if fw ∧ updatedBW then
        updatedFW ⟵ computeH2(fw) ∨
        disambiguateActions()
    if ¬ fw ∧ updatedFW then
        updatedBW ⟵ computeH2(fw) ∨
        disambiguateActions()
    fw ⟵ (¬ fw)
```

The computation of invariants in both directions can be combined to allow inferring more information. For instance, it is possible that after a backward computation of h^2, additional mutexes can be found forward by computing h^2 with the additional backward invariants. Algorithm 1 shows the fixpoint procedure employed.

Experiments

We have implemented the described techniques as part of Fast Downward's preprocessor (Helmert 2006). To assess their usefulness, we have first analyzed their capability to simplify the benchmark tasks of the International Planning Competition (IPC) for optimal planning. Table 1 shows the sum of original facts (F) and actions (O) among all the problems of the domain, the geometric mean of the ratio of remaining facts (% F) and actions (% O) after the simplification, the sum of forward mutexes barring those encoded by $\mathcal{V}$ (M_{fw}), the geometric mean of the ratio of forward mutexes that were not extracted from *"exactly-1"* invariant groups (% M_{fw}), the number of backward mutexes (M_{bw}), the maximum number of times that mutexes were computed forward or backward until no new invariants could be found (i) and the maximum time necessary to compute the invariants (t(s)) among all the problems of the domain. In the latter, if in at least one problem the invariant computation surpassed the imposed limit of 300 seconds, we report the number of problems in which this occurs instead.

A significant simplification is obtained in several domains. More than a fourth of the facts could be discarded in three domains (Airport, Parcprinter and Tidybot), and more than three fourths of the actions in Airport and Tidybot, with several other domains like Trucks, Scanalyzer, NoMystery or Floortile greatly benefiting from action pruning too. Some of these domains are STRIPS versions of ADL formulations, which indicates that a naive compilation may produce a high number of irrelevant actions and facts, but in domains like Floortile and NoMystery this is due to their structure.

In many domains h^2 finds a sizable amount of new mutexes, and in 15 domains backward mutexes are found, with a positive correlation between the number of backward mutexes and pruned actions in most cases. Only in three domains the computation of invariants is not possible under 300 seconds, and often in tasks far from the reach of current

Domain	#	F	% F	O	% O	M_{fw}	% M_{fw}	M_{bw}	i	t (s)
Airport	(50)	157592	0.62	144963	0.23	7711748	0.96	130480	7	> (13)
Barman	(20)	4604	1.00	13264	0.86	11569	0.96	0	2	0.22
Blocksworld	(35)	4826	1.00	7490	1.00	28694	0.07	0	2	0.12
Depot	(22)	9423	1.00	68894	0.83	104628	0.21	0	2	16.56
Driverlog	(20)	6007	1.00	53494	1.00	715	0.00	0	2	5.86
Elevators08	(30)	3360	1.00	18520	1.00	0	–	0	2	0.06
Elevators11	(20)	2097	1.00	11450	1.00	0	–	0	2	0.04
Floortile11	(20)	3578	0.82	9188	0.62	3766	0.00	4758	4	0.22
Freecell	(80)	23419	1.00	1071066	1.00	267475	0.83	37	3	5.76
Grid	(5)	3373	1.00	38808	1.00	1770	0.80	0	2	13.5
Gripper	(20)	2380	1.00	3720	1.00	2300	0.00	0	2	0.02
Logistics00	(28)	3429	1.00	6972	1.00	0	–	0	2	0.06
Logistics98	(35)	82687	1.00	501186	1.00	0	–	0	2	> (1)
Miconic	(150)	13950	1.00	189100	1.00	0	–	0	2	0.14
Mprime	(35)	17796	0.99	567960	0.90	13558	1.00	0	2	36.22
Mystery	(30)	13066	0.94	217800	0.77	22891	1.00	838	5	38.9
NoMystery11	(20)	4434	0.94	72522	0.62	26802	1.00	364	4	0.7
Openstacks08	(30)	3540	1.00	28480	1.00	3138	1.00	0	2	0.14
Openstacks11	(20)	2360	1.00	17320	1.00	2050	1.00	0	2	0.1
Openstacks06	(30)	10634	1.00	213470	1.00	177579	1.00	0	2	114.96
Parcprinter08	(30)	6139	0.75	9066	0.33	57862	1.00	8483	5	0.48
Parcprinter11	(20)	3993	0.72	5096	0.32	35205	1.00	5054	5	0.2
Parking11	(20)	11020	0.97	241740	0.96	195470	0.55	340	3	13.28
Pathways-nn	(30)	13119	1.00	40595	1.00	20585	1.00	219	3	1.38
Pegsol08	(30)	2920	0.86	5346	0.70	2164	1.00	152	7	0.02
Pegsol11	(20)	2000	1.00	3700	0.94	1559	1.00	0	2	0.02
Pipesw-notan	(50)	44594	0.97	187388	0.92	556322	1.00	0	2	81.86
Pipesw-tan	(50)	28027	0.98	1135917	0.98	278986	0.79	14	3	127.44
Psrsmall	(50)	2158	1.00	14546	0.98	3486	1.00	0	2	0.22
Rovers	(40)	29324	1.00	231653	1.00	637	1.00	0	2	34.68
Satellite	(36)	30479	1.00	3709130	1.00	1156	1.00	0	2	> (3)
Scanalyzer08	(30)	4680	1.00	1145836	0.57	25776	1.00	0	2	13.66
Scanalyzer11	(20)	3088	1.00	631288	0.69	16768	1.00	0	2	13.64
Sokoban08	(30)	8518	0.79	12674	0.76	102842	0.43	3454	7	4.78
Sokoban11	(20)	5306	0.78	7166	0.76	87257	0.51	996	7	4.8
Tidybot11	(20)	11476	0.69	384018	0.15	87905	1.00	0	2	21.76
TPP	(30)	18807	0.88	281351	0.55	2088	1.00	0	3	25.22
Transport08	(30)	6800	1.00	105888	1.00	0	–	0	2	1.92
Transport11	(20)	2886	1.00	35216	1.00	0	–	0	2	0.2
Trucks	(30)	6961	0.95	442262	0.54	30234	0.35	25219	3	39.86
Visitall	(20)	2516	1.00	3520	1.00	753	1.00	0	2	0.1
Woodw08	(30)	5677	0.96	27835	0.49	13244	0.91	36	4	0.5
Woodw11	(20)	3805	0.96	18175	0.48	9005	0.91	26	4	0.3
Zenotravel	(20)	4518	1.00	140433	1.00	0	–	0	2	12.8

Table 1: Results of the invariant computation.

state-of-the-art optimal planners.

We also did an ablation study disabling some features of the preprocessor. Disambiguation used to prune spurious actions proved to be essential in Airport, Floortile, Parcprinter, TPP and Woodworking, but disabling the encoding of invariants in the actions via disambiguation had very little impact. The fixpoint alternation of the forward and backward direction is of most importance in Parcprinter, though it also helps in other domains such as Airport and Sokoban. Disabling the backward h^2 computation has a major impact in domains with an important amount of backward mutexes.

Now we check whether simplifying the problem helps current planners to solve more tasks. Table 2 shows the coverage of four planners, two optimal planners run on the optimal benchmark - Fast Downward with blind search and the lm-cut heuristic (Helmert and Domshlak 2009) - and two satisficing planners on the satisficing benchmark - FD is Fast Downward with lazy greedy best-first search, preferred operators and the FF heuristic (Hoffmann and Nebel 2001), and LAMA is the first iteration of LAMA (Richter and Westphal 2010). Only the domains for which a change in coverage was observed are displayed. As shown by the results, there is a significant improvement in all four planners. The optimal planners solve strictly more tasks, and most of the tasks lost by FD are due to different tie breakings. This result is very significative, as it proves that an orthogonal preprocessing step helps planners of a very different nature.

We also tried a version that explicitly checked if backward invariants were violated when generating states forward. Blind search benefited from this, solving six more tasks in Floortile, one more in Sokoban08 and one more in Trucks-strips, and reducing the number of expanded nodes in other domains. The results remained unchanged for the heuristic planners, though. This is because once some actions are pruned thanks to the backward h^2, the reachability analysis done by these planners suffices to detect states that violate backward mutexes as forward dead-ends.

		Optimal				Satisficing			
		Blind		LM-cut		FD		LAMA	
Domain	#	-	h^2	-	h^2	-	h^2	-	h^2
Airport	50	22	+5	28	+1	37	+2	35	+3
Barman	20	4	0	4	0	8	-1	20	0
Depot	22	4	0	7	0	18	0	21	+1
Floortile-11	20	2	+6	7	+7	7	+13	6	+14
Freecell	80	20	0	15	0	80	-1	80	0
Mystery	30	15	0	17	0	16	+2	19	0
Nomystery-11	20	8	0	14	0	10	-1	13	0
Parcprinter-08	30	10	+11	18	+4	21	+9	27	+3
Parcprinter-11	20	6	+10	13	+4	3	+15	14	+6
Pegsol-08	30	27	0	28	+1	30	0	30	0
Pegsol-11	20	17	0	18	+1	20	0	20	0
Pipes-notank	50	17	0	17	0	44	-2	43	+1
Satellite	36	6	0	7	0	36	-1	36	-1
Sokoban-08	30	22	+5	30	0	28	0	29	0
Sokoban-11	20	19	+1	20	0	18	0	19	0
Tidybot-11	20	12	0	14	+3	15	+2	16	+3
Tpp	30	6	0	6	+1	30	0	30	0
Trucks-strips	30	6	+1	10	0	20	-2	15	0
Woodworking-08	30	8	+1	17	+5	30	0	30	0
Woodworking-11	20	3	+1	12	+3	20	0	20	0
Others	788	299	0	445	0	647	0	773	0
$\sum$	1396	533	+41	747	+30	1138	+35	1296	+30

Table 2: Coverage of different planners. In domains with different sets of problems, the corresponding set was used.

Conclusions

In this work we have analyzed several unclear aspects of invariant computation. Additionally, we have done an experimental analysis regarding implementation details not covered by previous works, with a particular emphasis on the computation of invariants in regression. Results show that it is very advantageous to exploit invariants in a preprocessing step to simplify the planning task, even if those invariants are not explicitly used during search.

We leave as future work exploring other invariant computation methods in regression and testing empirically the impact of backward invariants in other paradigms. The potential of completing the model using invariants is also interesting, *e.g.*: disambiguating undefined preconditions can positively affect LP-based heuristics (Pommerening et al. 2014) by turning actions that *sometimes* consume or produce an atom into actions that *always* do so. We also plan to have a closer look at the interaction with complementary methods to simplify the planning task such as irrelevant-action pruning (Scholz 2004; Haslum, Helmert, and Jonsson 2013).

Acknowledgements

This work is partially supported by the EU FP7 Programme under grant agreement no. 295261 (MEALS).

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2254–2260.

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

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

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

Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1840–1845.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, 135–147.

Fikes, R., and Nilsson, N. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 608–620.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in tim. *Journal of Artificial Intelligence Research (JAIR)* 9(1):367–421.

Gerevini, A., and Schubert, L. 1998. Inferring state constraints for domain-independent planning. In *National Conference on Artificial Intelligence (AAAI)*, 905–912.

Haslum, P.; Helmert, M.; and Jonsson, A. 2013. Safe, strong, and tractable relevance analysis for planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 317–321.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1898–1903.

Haslum, P. 2008. Additive and reversed relaxed reachability heuristics revisited. *Proceedings of the 6th International Planning Competition*.

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

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

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

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

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

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

Massey, B. 1999. *Directions In Planning: Understanding The Flow Of Time In Planning*. Ph.D. Dissertation, Computational Intelligence Research Laboratory, University of Oregon.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Principles of Knowledge Representation and Reasoning*, 103–114.

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

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

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

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In *AAAI Conference on Artificial Intelligence (AAAI)*, 806–811.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *European Conference on Artificial Intelligence (ECAI)*, 568–571.

Scholz, U. 2004. *Reducing Planning Problems by Path Reduction*. Ph.D. Dissertation, Technische Universität Darmstadt.

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

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *International Conference on Principles and Practice of Constraint Programming*, volume 3709 of *LNCS*, 682–696.