Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Recursive Polynomial Reductions for Classical Planning

Jan Tožička, Jan Jakubův, Martin Svatoš, Antonín Komenda
{jan.tozicka, jan.jakubuv, martin.svatos, antonin.komenda}@agents.fel.cvut.cz
Agent Technology Center, FEE, CTU in Prague
Technická 2, 166 27 Praha 6, Czech Republic

Abstract

Reducing accidental complexity in planning problems is a well-established method for increasing efficiency of classical planning. Removal of superfluous facts and actions, and problem transformation by recursive macro actions are representatives of such methods working directly on input planning problems. Despite of its general applicability and thorough theoretical analysis, there is only a sparse amount of experimental results.

In this paper, we adopt selected reduction methods from literature and amend them with a generalization-based reduction scheme and auxiliary reductions. We show that all presented reductions are polynomial in time to the size of an input problem. All reductions applied in a recursive manner produce only safe (solution preserving) abstractions of the problem, and they can implicitly represent exponentially long plans in a compact form. Experimentally, we validate efficiency of the presented reductions on the IPC benchmark set and show average 24% reduction over all problems. Additionally, we experimentally analyze the trade-off between increase of coverage and decrease of the plan quality.

Introduction

Classical planning problems are hard in general. Precisely, the unbounded decision variant is PSPACE-complete for grounded models as shown by (Bylander 1994). However, some of the commonly used benchmark problems are easy, that is polynomial in time to the size of the problem definition (Helmert 2003) and some special cases are even very easy—linear in time (Slaney and Thiébaux 2001). Appropriate encoding of such problems is of high importance for planning techniques which depend on hidden structures in the planning problems. This "superfluous hardness" was denoted by (Haslum 2007) as *accidental complexity*, a term borrowed from software engineering to denote removable (as opposed to intrinsic) complexity in programs. Additionally, his work showed how the complexity can be partially reduced and how the reductions help to downsize some of planning benchmarks.

The most prevailing approach to classical planning is currently state-space search with automatically derived heuristics. Still, the most successful planners rely on preliminary reduction of the planning problem by reachability analysis, e.g., in (Hoffmann and Nebel 2001; Helmert 2006), that is removal of a subset of actions which can be proven to be inapplicable during the search. Reachability analysis can dramatically reduce the size of the planning problem thus increase efficiency of the following search. Problem reduction by reachability analysis is one of transition system reduction techniques (Haslum 2007; Chen and Yao 2009; Coles and Coles 2010; Nissim, Apsel, and Brafman 2012), which are related to reductions by macro actions (describing sequences of actions) identified from the planning problem (Jonsson 2009; Bäckström, Jonsson, and Jonsson 2012). Although these approaches were recently shadowed by the tremendous advances in domain-independent heuristics, we argue they can be beneficial for classical planning, especially as they can be used with any (black-box) planner.

Most of the problem reduction methods in literature are based on graph representation of possible changes of state facts by actions—domain transition graphs (Helmert 2006)—and causal influences among actions in the planning domain—causal graphs (Knoblock 1994; Bacchus and Yang 1994). Both graphs represent mentioned hidden structures in the problems, technically projections of the planning problems. We define presented reductions over complete description of a planning problem, therefore we do not "project out" any information. As (Pochter, Zohar, and Rosenschein 2011), we represent the planning problem as a graph, however with different semantics. In our description, preconditions and effects represent edges, not vertices and we do not explicitly comprise planning variables as vertices, but we use vertices for variable and value pairs.

We build on selected transition system reductions inspired by (Haslum 2007) and by (Coles and Coles 2010), and add a new action generalization scheme towards stronger problem reduction. This additional scheme reduces a pattern irreducible by (Haslum 2007) or by (Coles and Coles 2010) as it generalizes two actions by partial lifting. In contrast to (Haslum 2007), where some of the reductions can be exponential, we use only polynomial reduction schemes in planning problem size. Despite this, our reduction schemes can implicitly represent exponentially long plans in a compact form (i.e., polynomially), as shown by (Bäckström and Jonsson 2012) for recursive macros. That is we combine principles of pure reduction (e.g., removing superfluous variables or actions) without any influence on the resulting plan with selected macro-like reductions (e.g., combining several actions into one) supplemented by action reconstruction rules used for reconstruction of the resulting plan. Similarly as (Haslum 2007), we praise benefits of possibility to use a black-box planner, therefore our reduction schemes are offline (in contrast to e.g., partial order reduction techniques (Wehrle and Helmert 2012) which work online during the search). In contrast to (Coles and Coles 2010) and to (Wehrle et al. 2015), we do not target optimality preserving reductions, only correctness and completeness, i.e., safe abstractions in terms of (Hoffmann, Kissmann, and Torralba 2014). We show the proposed set of reduction rules favorably affects planning efficiency on an exhaustive set of contemporary planning benchmarks.

Formalism

Classical planning problems are often described by STRIPS (Fikes and Nilsson 1971) where possible system states are described by *facts*, and applicable *actions* are described in terms of precondition and effect formulas. In this work, we use *Finite Domain Representation* (FDR) based on Multi-Valued Planning Task (Helmert 2006) and SAS$^+$ (Bäckström 1992) of planning problems, which is a widely used equivalent of STRIPS. In FDR, system states are described in terms of multi-valued variables, and actions are described by precondition and effects on variable values.

An FDR *planning problem* Π is a quadruple $\Pi = \langle V, A, I, G\rangle$, where V is a finite set of *domain variables*, A is a finite set of *actions*, I is the *initial state*, and G is the *goal condition*. Every finite-domain variable $v \in V$ is associated with a finite domain $\mathsf{dom}(v)$ of possible values. We consider pair-wise disjoint domains. An *assignment* is a function from $V_0 \subseteq V$ such that every variable $v \in V_0$ is assigned a value from its domain $\mathsf{dom}(v)$. An *action a* is a pair of assignments called in turn the *precondition* and the *effect*. A *state s* is an assignment of all variables V. Finally, the *initial state I* is a state, and the *goal condition G* is an assignment.

Assignment t_0 *agrees* with assignment t_1 on $V_0 \subseteq V$ when $t_0(v) = t_1(v)$ for all $v \in V_0$, provided both t_0 and t_1 are defined on V_0. Assignment t_0 *agrees* with assignment t_1 when t_0 agrees with t_1 on all variables relevant to t_0. Action $a = \langle pre, \mathit{eff}\rangle$ is *applicable* in state s iff pre agrees with s. A *state progression* function $\gamma(s_0, a)$ is defined whenever a is applicable in s_0 and it yields the state $s_1 = \gamma(s_0, a)$ which agrees with eff which agrees with s_0 on variables not specified in eff. For an FDR problem $\Pi = \langle V, A, I, G\rangle$, a *plan π* is a finite sequence of actions from A. The state progression function can be iteratively extended to $\gamma^\star(s_0, \pi)$ defined on plans instead of actions. Plan π is a *solution* of Π when $\gamma^\star(I, \pi)$ is defined and agrees with G. Let $\mathsf{sol}(\Pi)$ denote all solutions of problem Π.

We find it convenient to describe reductions of planning problems in terms of *full transition graphs* (FTG). The full transition graph FTG(Π) of problem Π is a directed graph whose nodes are actions and all legal variable-value pairs. For every action $a = \langle pre, \mathit{eff}\rangle$ from Π, graph FTG(Π) contains a *precondition edge* ($\langle v, x\rangle \rightarrow a$) whenever $pre(v) = x$, and it contains an *effect edge* ($a \rightarrow \langle v, y\rangle$) whenever $\mathit{eff}(v) = y$. Furthermore, FTG contains an *implicit precondition edge* ($\langle v, x\rangle \dashrightarrow a$) for every value x of variable v whenever $\mathit{eff}(v)$ is defined but $pre(v)$ is not defined. A *prevail precondition* edge is a precondition edge ($\langle v, x\rangle \rightarrow a$) such that there is no edge ($a \rightarrow \langle v, y\rangle$) for any y.

Moreover FTG contains two additional special actions *initial* ($a_I = \langle \emptyset, I\rangle$) and *goal* ($a_G = \langle G, \emptyset\rangle$) actions. These two actions can be used by any reduction instead of any other action if it is not stated otherwise. The graph FTG(Π) is thus uniquely determined by problem Π.

In an FTG, as opposed to *causal graphs* or *domain transition graphs*, no information is abstracted. Hence FTG just provides an equivalent description of an FDR problem which allows us to easily describe and implement all the reductions.

Reduction Preliminaries

In this section we provide a formal ground for description of *polynomial* FDR *reductions*. We formally define properties of polynomial reductions. We prove basic results concerning recursive application of arbitrary polynomial reductions. Namely, that recursive application always terminates (Lemma 1), and that the result of a recursive application can itself be computed in polynomial time (Theorem 1).

An FDR reduction is characterized by preservation of solvability and by instance size reduction. The instance size covers not only variable and action count but also the sizes of involved variable domains and number of preconditions, effects and goal variables:

$$|\Pi| = |V| + |\text{vertices}(\text{FTG}(\Pi))| + |\text{edges}(\text{FTG}(\Pi))|$$

Definition 1. *An* FDR reduction *ρ is a partial function from* FDR *problems to* FDR *problems such that*

1. *Π is solvable if and only if $\rho(\Pi)$ is solvable, and*
2. *it reduces the instance size, that is, $|\rho(\Pi)| < |\Pi|$, and*
3. *there is a* solution extension *function $\overleftarrow{\rho}(\pi)$ such that $\overleftarrow{\rho}(\pi) \in \mathsf{sol}(\Pi)$ for every solution π of $\rho(\Pi)$.*

The problem is *irreducible* (with respect to ρ) if $\rho(\Pi)$ is not defined.

A direct consequence of the above definition is that a composition, or a recursive application, of several reductions is itself a reduction. Hence it is solution preserving. We define polynomial reductions with respect both to problem reduction and solution extension.

Definition 2. FDR *reduction ρ is* polynomial *when*

1. *$\rho(\Pi)$ can be computed in time polynomial in $|\Pi|$, and*
2. *$\overleftarrow{\rho}(\pi)$ can be computed in time polynomial in $|\pi|$.*

We suppose that the computation of $\rho(\Pi)$ terminates in polynomial time even when ρ is not applicable to Π. That is to say, the applicability of ρ to Π has to be decidable in time polynomial in $|\Pi|$. For convenience, we define the *reduction relation* $\rightarrow$ as follows.

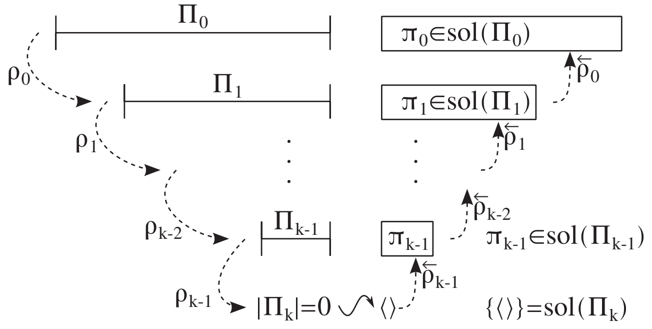
Π_0
$\pi_0 \in \mathrm{sol}(\Pi_0)$
ρ_0
Π_1
$\pi_1 \in \mathrm{sol}(\Pi_1)$
$\overleftarrow{\rho}_0$
ρ_1
$\overleftarrow{\rho}_1$
$\overleftarrow{\rho}_{k-2}$
ρ_{k-2}
Π_{k-1}
π_{k-1}
$\pi_{k-1} \in \mathrm{sol}(\Pi_{k-1})$
$\overleftarrow{\rho}_{k-1}$
ρ_{k-1}
$|\Pi_k|=0$
$\langle\rangle$
$\{\langle\rangle\}=\mathrm{sol}(\Pi_k)$

Figure 1: Recursive application of FDR reductions and backward solution extension. Input problem Π_0 is reduced either to an irreducible or a trivially solved problem Π_k. A solution of Π_k is then extended to a solution of the input problem Π_0.

Definition 3. *Write $\Pi_0 \to \Pi_1$ when there is a polynomial reduction ρ such that $\rho(\Pi_0) = \Pi_1$. Let $\twoheadrightarrow$ be iterative application of set of reductions $\rho_0, \ldots, \rho_n$ defined by following procedure* [1]*:*

```
repeat
    for $i \leftarrow 0$ to $n$ do
        repeat
            $\Pi \leftarrow \rho_i(\Pi)$;
        until $\Pi$ has not changed;
    end
until $\Pi$ has not changed;
```

Different algorithms are possible. Their results can differ because, in general, application of one reduction can disable application of another reduction.

The reduction relation represents an arbitrary polynomial reduction application. All the results from this section are, however, also valid for an arbitrary reduction collection. Transitive closure $\twoheadrightarrow$ represents recursive reduction application. It suggests a natural method to find a solution of a planning problem by reductions and backward extensions depicted in Figure 1. We prove that recursive reductions always terminate.

Lemma 1. *The reduction relation $\to$ is terminating, that is, there is no infinite chain of reductions $\Pi_0 \to \Pi_1 \to \cdots$.*

Proof. Follows directly from the fact that reductions reduce instance size (Def. 1) which is lower bounded by 0. Moreover, the length of that chain is limited by $|\Pi_0|$. $\square$

When arbitrary polynomial reductions are applied recursively, the resulting problem is always computed in polynomial time. The following theorem formally states the claim.

Theorem 1. *When $\Pi_0 \twoheadrightarrow \Pi_1$ then Π_1 can be computed from Π_0 in time polynomial in $|\Pi_0|$.*

[1]Here, we suppose that $\rho_i(\Pi) = \Pi$ whenever $\rho_i(\Pi)$ is not defined.

Proof. From the proof of Lemma 1 follows that (1) every reduction application can be computed in time polynomial in $|\Pi_0|$, and (2) the count of reductions in the sequence is limited by $|\Pi_0|$. $\square$

In contrast, a backward recursive plan extension does not necessarily yield a polynomial function. The backward extension can be exponential in the size of a solution of Π_1, even in the instance size $|\Pi_0|$. This happens naturally for FDR problems whose solution sizes are exponential in the instance size but it can happen even for problems with polynomial solutions. We can only say that the plan extension is polynomial in the size of the extended solution of Π_0 as long as there is no extension step that decreases the length of the solution, which is the case of presented reductions.

Reductions

The previous section described polynomial reductions in general. In this section we formally describe and prove correctness of three specific reductions. The first one, *Generalize Action* (ρ_{ga}), is novel, to the best of our knowledge, not similar to any previously published reduction schemes. The other two can be used to achieve similar results as (Haslum 2007) and (Junghanns and Schaeffer 2001). We proceed by a discussion of related literature and conclude by a case study taken from (Haslum 2007).

To describe reductions, we use a renaming operator which changes the value of a variable in an assignment. Formally, *renaming* $[t]_{x\to y}$ of value x to value y in assignment t is the assignment defined below.

$$[t]_{x\to y}(v) = \begin{cases} y & \text{iff } t(v) = x \\ t(v) & \text{otherwise} \end{cases}$$

Similarly, we define the *delete variable* operator $[t]_{-v}$ which simply removes variable v from the domain of assignment t, making the value $t(v)$ undefined. And with the same notation the *delete value* operator $[t]_{-x}$ which removes a single value x from the domain of the related variable, which means that if $t(v) = x$ then it makes $t(v)$ undefined. The operators are naturally extended to actions (applying the operator to the precondition and to the effect) and to action sets (element-wise).

Technically, the following definitions provide partial function schemes which can be turned into functions by selecting specific values for unbound parameters.

Reduction: Generalize Action (ρ_{ga})

Suppose a problem with variable v with a two-valued domain $\mathsf{dom}(v) = \{x, y\}$. Let us have two actions which differ only in that the first requires value x and the second requires value y as a prevail precondition. In this case, the action can be generalized into one without any precondition on the value of variable v. The backward solution expansion changes the generalized action in a plan to one of the original actions based on the current value of variable v. The reduction, demonstrated in Figure 2, can be generalized for n actions and an n-valued domain.

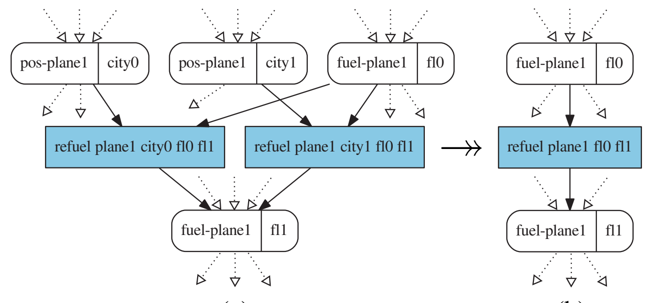

Figure 2: A novel *Generalize Action* (ρ_{ga}) reduction demonstrated on a part of the FTG of a Zenotravel benchmark problem. Round boxes represent variable-value pairs (facts), rectangles represent actions. Dotted edges represent possible connections to other actions. Our reduction can recognize that two refuel actions can be safely merged into one omitting the information about position (the refuel action can be executed in both cities). Hence pattern (a) can be reduced to a simpler case (b) decreasing action count.

Definition 4. *Reduction ρ_{ga} is applicable to problem Π when there is variable v with $\mathsf{dom}(v) = \{x, y\}$ and two actions a_1 and a_2, such that* FTG(Π) *contains*

1. *a prevail precondition edge ($\langle v, x\rangle \to a_1$), and*
2. *a prevail precondition edge ($\langle v, y\rangle \to a_2$), and*
3. *it holds that $[a_1]_{x\to y} = a_2$.*

Reduction ρ_{ga} applied to $\Pi = \langle V, A, I, G\rangle$ yields

$$\rho_{\mathsf{ga}}(\Pi) = \langle V, (A \setminus \{a_1, a_2\}) \cup \{[a_1]_{-v}\}, I, G\rangle$$

where variable domains are as in Π.

Theorem 2. *Reduction ρ_{ga} is a polynomial* FDR *reduction.*

Proof. Let Π be given, and let x, y, v, a_1, a_2 be as in Def. 4. Let $a = [a_1]_{-v}$. To avoid possible action confusion caused by value renaming, let us suppose that actions are assigned unique ids which are preserved by the reduction, and that plans are sequences of these ids.

Suppose we have a solution $\pi \in \mathsf{sol}(\Pi)$. The solution π' of $\rho_{\mathsf{ga}}(\Pi)$ can be obtained from π by replacing all the occurrences of actions a_1 and a_2 by a in π. The other way round, when a solution π' of $\rho_{\mathsf{ga}}(\Pi)$ is given, we can simulate plan π' in states of problem Π. When action a should be executed in state s, we change it to a_1 when $s(v) = x$, or to a_2 when $s(v) = y$. This procedure yields a solution π of Π and describes the plan extension function $\overleftarrow{\rho_{\mathsf{ga}}}(\pi')$.

Reduction ρ_{ga} reduces an instance size because it removes one action. Finally, the reduction is clearly polynomial because its applicability, application, and plan extension can be computed in time linear to the input size. □

Reduction: Merge Values (ρ_{mv})

Reduction ρ_{mv} looks for two actions which, without other side effects, switch between two different values of a variable. Then, it merges both values and it removes the actions.

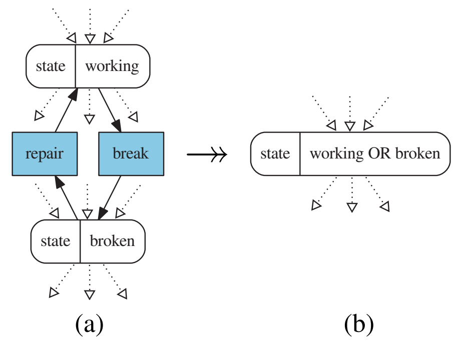

Figure 3: Example of ρ_{mv} reduction.

Figure 3 demonstrates this by reducing case (a) to (b). Solution expansion is achieved by inserting one of the removed action where required.

This reduction is closely related to (Haslum 2007, Theorem 1) which achieve similar results using abstraction by deleting a variable which has a strongly connected *free domain transition graph* (Haslum 2007). A formal definition and its correctness follow.

Definition 5. *Reduction ρ_{mv} is applicable to problem Π when there are values x, y of variable v, and actions a_1, a_2, such that* FTG(Π) *contains*

1. *edges ($\langle v, x\rangle \to a_1 \to \langle v, y\rangle$), and*
2. *edges ($\langle v, y\rangle \to a_2 \to \langle v, x\rangle$), and*
3. *there are no other edges ingoing/outgoing a_1 and a_2.*

Reduction ρ_{mv} applied to $\Pi = \langle V, A, I, G\rangle$ yields

$$\rho_{\mathsf{mv}}(\Pi) = \langle V, [A \setminus \{a_1, a_2\}]_{x\to y}, [I]_{x\to y}, [G]_{x\to y}\rangle$$

where variable domains are as in Π, except for the $\mathsf{dom}(v)$ from which the value x is removed.

Theorem 3. *Reduction ρ_{mv} is a polynomial* FDR *reduction.*

Proof. Similarly to the proof of Theorem 2. The solution extension is done again by a simulation of plan π' in states of problem Π. When some action a without preconditions fulfilled in the simulated state is encountered, it must be the case that variable v has value x but value y is required (or vice versa). Then we can insert action a_1 or a_2 prior to a to change the value of v appropriately and continue with the simulation. □

There are cases reducible by (Haslum 2007) but not by ρ_{mv}, and vice versa. ρ_{mv} can be applied when only some of the variable values are strongly connected, which is not possible by (Haslum 2007, Theorem 1). For practical reasons, the reduction ρ_{mv} does not reduce all possible connected components (composed of cycles of size larger than two) in a free DTG, which do not often appear in the benchmark problems anyway. Generalized implementation of ρ_{mv} towards full free DTG reduction would keep the polynomial guarantees and is left for future work.

Reduction: Tunnel Macro (ρ_{tm})

The reduction is related to *tunnel macros* from (Junghanns and Schaeffer 2001) and to *generalized tunnel macros* from (Haslum 2007, Theorem 2), which in general can lead to an exponential number of added tunnel macros, therefore does not necessarily reduce instance size nor is necessarily polynomial. Our restricted case assures that the number of added actions will be limited and thus the reduction will follow the condition 2 of the Def. 1. First, we restrict the number of the actions in the tunnel macro to 2. Second, we allow the reduction only if the number of tunnel macros is not larger than the number of the source actions in the problem Π.

The reduction looks for a variable value which is consumed by actions with no side effects. Then, for each combination of an action producing the value and an action switching it to another value, the reduction creates a pair action representing the tunnel macro. Provided that the number of the pairs does not exceed the number of the source actions, the reduction is used and the original actions and the value are removed. Solution expansion is achieved by splitting the tunnel macros to an ordered sequence of the actions in the pair. The formal definition of the restricted tunnel macro reduction follows:

Definition 6. *Reduction ρ_{tm} is applicable to problem Π when there are values x, $y_1, \ldots, y_n$ of variable v, and two sets of actions B, C for which $|B| + |C| \geq |B||C|$ holds, such that* $\mathrm{FTG}(\Pi)$ *contains*

1. *edges $b_i \rightarrow \langle v, x\rangle \rightarrow c_j \rightarrow \langle v, y_j\rangle$ for all $b_i \in B, c_j \in C$,*
2. *$\forall j : x \neq y_j$ (x is always a non-prevail precondition),*
3. *there are no other edges ingoing/outgoing $c_j \in C$, and*
4. *there are no other edges ingoing $\langle v, x\rangle$, and*
5. *at least one of following is true:*
   (a) *there are no other edges outgoing $\langle v, x\rangle$, or*
   (b) *$|C| = 1$ and there are no other edges ingoing $\langle v, y_1\rangle$.*

Reduction ρ_{tm} applied to $\Pi = \langle V, A, I, G\rangle$ yields

$$\rho_{\mathsf{tm}}(\Pi) = \begin{cases} (5a) & \langle V, [A_{\mathsf{tm}} \setminus B]_{-x}, [I]_{-x}, [G]_{-x}\rangle \\ (5b) & \langle V, [A_{\mathsf{tm}}]_{x \to y_1}, [I]_{x \to y_1}, [G]_{x \to y_1}\rangle \end{cases}$$

where $A_{\mathsf{tm}} = (A \setminus C) \cup B \times C$, where each combined action from the Cartesian product $B \times C$ merges separately preconditions and effects, and variable domains are as in Π. In the case of (5a) x value can be removed from $\mathsf{dom}(v)$ as it is not used anymore.

Theorem 4. *Reduction ρ_{tm} is a polynomial* FDR *reduction.*

Proof. The solution extension is simply done by replacing added macro actions by a pair of original actions. The reduction replaces all pairs of actions in the particular order producing and consuming $\langle v, x\rangle$ (only a tractable number of combinations is used), which cannot be interleaved by other actions, by one combined action with join preconditions and effects. The condition for non-prevail preconditions obviates cases with repeated c_j actions and, as the value $\langle v, x\rangle$ is removed after adding the $B \times C$ combinations, the inconsistencies in preconditions and effects are removed.

Reduction ρ_{tm} reduces an instance size, as the reduction removes value $\langle v, x\rangle$ and is used only for sets of actions B and C such that their Cartesian product is equal or smaller than sum of original numbers of actions in both action sets B and C. $\square$

The rather restrictive condition on the size of the actions sets $|B| + |C| \geq |B||C|$ can be in practice relaxed to $|B| + |C| \geq l$, where $l \leq |B||C|$ is a reduced number of needed tunnel macros for sets B and C. In some cases, several tunnel macros can be removed as far as they are superfluous because of empty effects. For example, when action c_j is inverse to b_i action, for some j and i. Such combinations, unnecessary in the reduced problem, are increasing $|B||C|$ and can obviate use of the reduction.

Case Study: Binary Counter

We demonstrate an implicit polynomial representation of exponential plans on an simple example of a *Binary Counter* taken from (Haslum 2007). A Binary Counter with n bits is represented by an FDR problem with n variables $\mathsf{b}_1,\ldots,\mathsf{b}_n$ taking values from $\{0, 1\}$ domains. Variables represent different bits, and actions inc_i (for $0 < i \leq n$) are designed to increase the value stored by the bits by 1. Let $\mathsf{inc}_1 = \langle \mathsf{b}_1 = 0, \mathsf{b}_1 = 1\rangle$ and $\mathsf{inc}_i = \langle pre_i, \mathit{eff}_i\rangle$ for $i > 1$ is defined as follows.

$$\begin{array}{llll} pre_i(\mathsf{b}_i) = 0 & pre_i(\mathsf{b}_{i-1}) = 1 & \cdots & pre_i(\mathsf{b}_1) = 1 \\ \mathit{eff}_i(\mathsf{b}_i) = 1 & \mathit{eff}_i(\mathsf{b}_{i-1}) = 0 & \cdots & \mathit{eff}_i(\mathsf{b}_1) = 0 \end{array}$$

Hence to change all the bits from 0's to 1's requires a plan with $2^n - 1$ actions, that is, exponential in the size of this FDR representation.

Using n applications of *tunnel macro* (ρ_{tm}) reduction, we can reduce the Binary Counter FDR problem to a trivially solved problem. These reductions iteratively use the condition *(5b)* of the definition of ρ_{tm}. In the first step, the C contains only action inc_1 which is the only action setting the last bit to 1. After merging this action with all actions setting this bit to 0 and after changing the respective variable initial value to 1, we can see that the value of this variable never changes and thus we can ignore it. Then we proceed with the next bits in the same manner until only one variable with a single value remains. This variable value is both in the initial state and required by goal condition, thus empty plan solves this reduced problem.

Suppose, we add decreasing actions dec_i, dual to inc_i, to increase the plan-space size. Even with this additional hassle, we can achieve the same result by n applications of *Merge Values* (ρ_{mv}) reduction. Technically, we need also a trivial "clean up" reduction which removes variables with a one-valued domain (see ρ_{rv} in the next section).

The trivially solved problem obtained by the reductions, together with a solution expansion method, can be seen as an implicit polynomial representation of an exponential plan. This representation can itself be computed in polynomial time. Furthermore, the solution can be extended in an iterative way, yielding next plan action in time polynomial to the length of the current solution part.

Size	Reduction [s]	Extension [s]	LAMA [s]
4 bits	< 0.1	< 0.1	0.1
8 bits	< 0.1	< 0.1	0.1
12 bits	< 0.1	< 0.1	0.3
16 bits	< 0.1	< 0.1	4.6
20 bits	< 0.1	0.2	111.8
24 bits	< 0.1	10.2	—
28 bits	< 0.1	—	—
32 bits	< 0.1	—	—

Table 1: Experimental results for a Binary Counter.

Table 1 shows experimental results for the Binary Counter problem (with dec_i operators) for varied number of bits. *Reduction* is the time required for problem reduction, *extension* is the time of backward solution expansion, and LAMA is the planning time of the state-of-the-art classical planner LAMA (Richter and Westphal 2010) (without any reductions). As expected, classical planner requires exponential time to find the solution. On the other hand, the time needed for the polynomial reduction phase is hardly measurable. Even though the expansion time grows exponentially, it is still substantially smaller than classical planner time.

The same effect could be achieved by a blind search with duplicate detection, however still as the plans are exponentially long the search will not end in polynomial time (similarly as *extension* in Table 1). In case of the reductions ρ_{tm} and ρ_{mv} used on the problem, the exponential plan will be compactly represented and as the reductions are polynomial such representation will be found in polynomial time (*reduction* in Table 1).

Auxiliary Reductions

This section provides informal description of other polynomial reductions implemented in order to undertake experiments described in the next section. *Merge with Initial State* (ρ_{mi}) reduction is related to inevitable actions (Coles and Coles 2010, Definition 3) applicable in the initial state. The remaining reductions are basically "clean up" reductions which simplify the problem after applications of previous reductions. This can allow application of another reductions.

Several reductions use *mutexes* – pairs of values of different variables that cannot be simultaneously assigned in one state. We keep detected mutexes together with FTG to avoid repeating calculations and update them when necessary. In our implementation, we use simple method for mutex detection: two values of different variables are sure to be mutexed if every operator setting one value always deletes the other and when both these values are not in I.

Some of these reductions are already implemented in preprocessing phase of many planners. Nevertheless, we do not apply them on the original problem only, but also on reduced problems which significantly boosts their effect.

Merge with Initial State (ρ_{mi}) reduction is applicable when G does not agree with I, that is, when the problem is not trivially solved and when the problem contains a unique action a executable in the initial state I. If a is executed only once, which is tested using simple sufficient but not necessary condition that it consumes a value that is never produced but by I, we can apply this action to I and remove it from the problem.

Merge Equivalent Actions (ρ_{ma}) reduction allows to merge two actions with equal set of incoming and outgoing edges in an FTG. During the solution extension, the merged action is changed back to any of the original actions.

Remove Variable (ρ_{rv}) reduction simply removes from the problem all variables with a one-valued domain. This value has to be set in the initial state and can not be changed. All possible references to the removed variables are removed from action preconditions as well.

Remove Unreachable Values (ρ_{ru}) reduction detects unreachable variable values using delete-relaxation reachability analysis. Detected unreachable values are removed from the problem.

Remove Unreachable Operators (ρ_{ro}) reduction deletes unreachable operators whose preconditions are in a mutex.

Remove Dead Ends (ρ_{rd}) reduction removes from the problem value x of variable v such that $\langle v, x\rangle$ has no outgoing edges in the FTG, and either $G(v)$ is undefined or $G(v) \neq x$, and $I(v) \neq x$. An exception is the case when there is an action a and another value $x \neq y \in \mathsf{dom}(v)$ such that the FTG contains edges $(\langle v, y\rangle \rightarrow a \rightarrow \langle v, x\rangle)$. Removing value x might lead to a wrong repeated application of a in the reduced problem.

Ground Operator Preconditions (ρ_{gv}) grounds free preconditions if their values can be determined. More specifically it tries all possible values and if all but one value is in mutex with some other precondition then this only possible value is used.

Ground Simple Operator (ρ_{gr}) removes implicit preconditions $\langle v, .\rangle \dashrightarrow a$ of operator with only one effect $a \rightarrow \langle v, x\rangle$. If variable v contains only two values, i.e. $|\mathsf{dom}(v)| = \{x, y\}$, then we can replace both implicit preconditions by a single precondition $\langle v, y\rangle \rightarrow a$, because the precondition $\langle v, x\rangle \rightarrow a$ would lead to an action with no effect.

Experiments

We evaluate the impact of the presented polynomial reductions on contemporary IPC benchmarks from the deterministic track. Each experiment run on E5420 2.5GHz processor, 2GB RAM with a 5 min limit. The benchmarks contain total 660 problems from 44 different domains. We start by analysis of reducibility of the benchmark problems. Then we evaluate the impact of reductions on classical heuristics used by greedy best-first search, concentrating on total number of solved problems (coverage) and runtime. Finally, we present the impact of the reductions on the LAMA planner featuring combination of planning techniques as multi-heuristic search, preferred operators, and others. All presented experiments use the Fast Downward planning system[2].

Table 2 shows average instance size reductions of benchmark problems. We can see that problems in 5 domains were

[2]http://www.fast-downward.org/

Domains	Mostly used reductions	Reduction
Completely Reduced		
gripper	$\rho_{tm}(376)$, $\rho_{ga}(89)$, $\rho_{mv}(23)$	100 %
logistics98	$\rho_{mv}(3055)$, $\rho_{rv}(34)$	100 %
logistics00	$\rho_{mv}(114)$, $\rho_{rv}(4)$	100 %
miconic	$\rho_{gr}(31)$, $\rho_{mv}(30)$, $\rho_{rv}(16)$	100 %
zenotravel	$\rho_{mv}(196)$, $\rho_{ga}(53)$, $\rho_{rv}(6)$	100 %
Mostly Reduced >50%		
rovers	$\rho_{ma}(463)$, $\rho_{mv}(348)$, $\rho_{rv}(189)$	95.5 %
satellite	$\rho_{mv}(435)$, $\rho_{gr}(413)$, $\rho_{rv}(202)$	94.0 %
parcprinter11	$\rho_{ga}(265)$, $\rho_{rv}(203)$, $\rho_{gv}(113)$	60.9 %
parcprinter08	$\rho_{ga}(134)$, $\rho_{rv}(83)$, $\rho_{gv}(37)$	53.7 %
Slightly Reduced 10 – 50%		
tpp	$\rho_{ma}(1506)$, $\rho_{mv}(30)$, $\rho_{rv}(5)$	46.7 %
tidybot11	$\rho_{gv}(954)$, $\rho_{rv}(47)$	44.0 %
woodworking11	$\rho_{gv}(3800)$, $\rho_{ma}(830)$, $\rho_{ga}(81)$	36.1 %
driverlog	$\rho_{mv}(91)$, $\rho_{ga}(24)$, $\rho_{gv}(12)$	27.0 %
floortile11	$\rho_{gv}(106)$, $\rho_{mv}(2)$, $\rho_{rv}(2)$	26.4 %
woodworking08	$\rho_{gv}(1840)$, $\rho_{ma}(170)$, $\rho_{ga}(45)$	23.1 %
airport	$\rho_{gv}(1503)$, $\rho_{rv}(24)$	11.6 %
Almost None Reduction <10%		
blocks, depot, elevators08, elevators11, freecell, openstacks, parking11, psr-small, scanalyzer08, scanalyzer11, sokoban08, sokoban11, storage, transport08, transport11, trucks, visitall11		
No Reductions		
barman11, mystery, no-mprime, no-mystery, nomystery11, openstacks08, openstacks11, pegsol08, pegsol11, pipesworld-notankage, pipesworld-tankage		
Overall Average Reductions		24.2 %

Table 2: Reducibility of benchmark problems.

completely reduced[3]. Problems of four domains were significantly simplified and the average problem reduction was 24.2%. Numbers in parentheses show number of applications of mostly used reductions averaged per problem. In most cases, the reduction is calculated within a fraction of a second and only for the biggest instances it takes few seconds, which is always a fraction of time required by *translate* step of the Fast Downward system.

Figure 4 shows relative importance of reductions in different domains. The importance is calculated as relative decrease in reduction when the respective reduction is disabled. The merge values reduction ρ_{mv} seems to be the most useful of the reductions but even other reductions help in several domains. The graph also shows that the auxiliary reductions do more than just cleaning after applications of main reductions and in some cases they significantly reduce the problem on their own. In these cases, dead end ρ_{rd} and reachability analyses ρ_{ru}, ρ_{ro} helped the most.

Table 3 shows an impact of the reductions on the total coverage of three selected planners without and with (columns $\twoheadrightarrow$) reductions together with the ratio of costs of created solutions[4]. For evaluations, we used greedy best-first search

[3]Solution of completely reduced problem is an empty plan which can be directly extended to the solution of the original problem and thus no execution of the classical planner is required.

[4]Action costs are not directly addressed by the reductions. Newly added action simply copies the cost of a removed action.

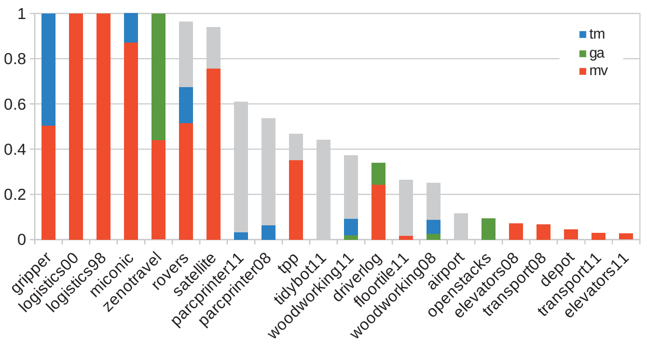

Figure 4: Relative importance of reductions. Gray area represents auxiliary reductions.

with (1) h^{max} heuristic and (2) h^{add} heuristic (Bonet and Geffner 1999), and (3) h^{FF} heuristic (Hoffmann and Nebel 2001). We can see that reductions help all heuristics to increase the coverage while the average decrease in solution quality is reasonable. In few domains, the decrease in solution quality is significant, the worst results are for *Zenotravel* domain (solution cost is up to 3.5 times higher). The generated solutions contain many sequences of flying around a city to consume fuel and then refuelling to certain level which is required to fly to next city. Nevertheless, these inefficiencies could be detected and removed in many cases by simple linear checking of repeating states in the solution which indicate action loops. On the other hand, we can see that the solution found on a reduced problem has been occasionally extended to a solution shorter than that found by the heuristics search itself.

Figure 5 compares the runtimes. For each problem, a point is drawn at the position corresponding to the runtime without reductions (x-coordinate) and the runtime with reductions (y-coordinate). Hence problems below the diagonal were solved faster with reductions. Notable are the problems at the right edge which were solved with reductions but not without. For all the problems, the reduction/expansion times are included in the runtime. In theory, a decreased instance size promises a runtime decrease. As the evaluation shows, in practice, it is not always the case.

Reductions with Multi-Heuristic Search

Table 4 shows how reductions affect the performance of a state-of-the-art planner LAMA (Richter and Westphal 2010). We can see that reductions overall help LAMA planner even though in many domains the coverage decreases. In most cases, it is caused by the Ground Operator Preconditions ρ_{gv} reduction which seems to hurt the landmark heuristics used by LAMA. We can observe a correlation between the domains in Table 4 and a smaller reduction ratio: 11 out of 17 domains are *Almost None Reduction* cases. The reason for that is that both LAMA and LAMA with reductions were able to solve all instances of at least *Slightly Reduced* domains not mentioned in Table 4. It seems that it is caused by heuristics which LAMA combines. They seem to help in

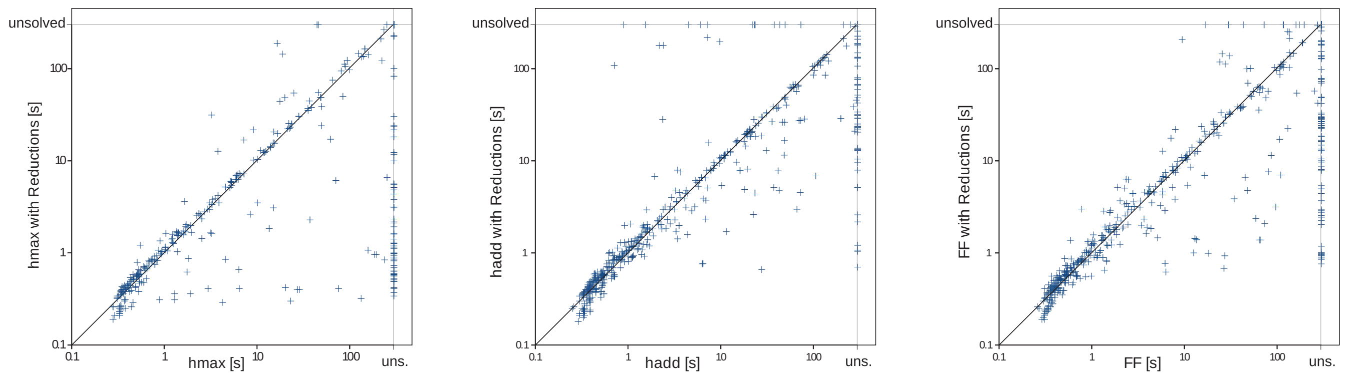

Figure 5: Runtime evaluation (logarithmic axes) of greedy best-first search with different heuristics (h^{max}, h^{add}, h^{FF}) with and without reductions with action costs. Problems with unit costs produce similar results.

	Normal Action Costs									Unit Action Costs								
Domains	h^{max}	$\twoheadrightarrow$	%	h^{add}	$\twoheadrightarrow$	%	h^{FF}	$\twoheadrightarrow$	%	h^{max}	$\twoheadrightarrow$	%	h^{add}	$\twoheadrightarrow$	%	h^{FF}	$\twoheadrightarrow$	%
gripper	6	**15**	1.35	**15**	**15**	1.04	**15**	**15**	1.34	6	**15**	1.35	**15**	**15**	1.35	**15**	**15**	1.04
logistics00	12	**15**	1.19	**15**	**15**	1.21	**15**	**15**	1.34	12	**15**	1.19	**15**	**15**	1.19	**15**	**15**	1.21
logistics98	3	**15**	1.32	12	**15**	1.36	11	**15**	1.40	3	**15**	1.32	12	**15**	1.32	11	**15**	1.36
miconic	6	**15**	1.82	**15**	**15**	**2.29**	**15**	**15**	**2.40**	6	**15**	1.82	**15**	**15**	1.82	**15**	**15**	**2.29**
zenotravel	8	**15**	**3.00**	**15**	**15**	**3.45**	**15**	**15**	**3.39**	8	**15**	**3.00**	**15**	**15**	1.06	**15**	**15**	**3.45**
rovers	2	7	1.50	9	**15**	1.93	9	**15**	1.86	2	7	1.50	9	**15**	1.08	9	**15**	1.93
satellite	2	5	1.72	12	**15**	**2.43**	10	**15**	**2.40**	2	5	1.72	12	**15**	0.83	10	**15**	**2.43**
parcprinter11	1	5	1.00	10	12	1.00	5	**14**	1.02	5	9	1.00	**15**	**15**	1.50	**15**	**15**	1.06
parcprinter08	8	9	1.00	11	14	1.01	11	**15**	1.02	9	13	1.00	**15**	**15**	1.00	**15**	**15**	1.00
tpp	4	4	1.04	9	**14**	1.17	8	**14**	1.30	4	4	1.04	9	**14**	1.20	8	**14**	1.17
tidybot11	3	4	1.00	10	**13**	1.00	9	10	1.01	3	4	1.00	10	**13**	1.03	10	10	1.00
woodworking11	1	1	0.93	9	10	1.04	9	**11**	1.06	1	1	1.00	2	2	0.99	**3**	**3**	1.00
driverlog	10	11	1.31	13	**15**	1.11	13	14	1.19	10	11	1.31	13	**15**	1.31	13	14	1.11
floortile11	**9**	8	1.63	4	7	1.42	6	7	1.43	6	**9**	1.04	3	6	0.83	3	6	0.98
woodworking08	5	6	1.02	12	**14**	1.09	13	**14**	1.08	3	3	1.00	6	**7**	1.09	6	**7**	1.00
airport	8	8	1.00	11	7	1.00	9	**10**	0.97	**11**	7	1.00	11	7	1.00	10	9	1.00
Total (660)	262	317	1.19	418	452	1.24	407	**475**	1.26	257	318	1.16	434	484	1.24	463	**511**	1.23

Table 3: Impact of polynomial reductions on the coverage and on solution quality (column %) of greedy best-first search with different heuristics (h^{max}, h^{add}, h^{FF}) with normal action costs and with unit action costs (most reduced domains).

some situations we solve by reductions (which also partially holds for h^{add} and h^{FF}).

Figure 6 evaluates the impact of reductions on runtime of LAMA planner. Most of the problems notably above the diagonal are from the *sokoban* domain where we can see also decrease in coverage.

Conclusions and Future Work

Besides reformulation of polynomial reduction rules from literature, we proposed a reduction scheme for action generalization and several auxiliary reductions. On a classical example of Binary Counter, we showed that a class of problems with exponentially long plans can be solved by our polynomial reductions as well. On contemporary standard benchmark domains, we have experimentally shown that the reduction rules used in recursive manner improves efficiency of planners using different heuristics and search schemes.

Recently, planners based on a general purpose SAT solver often outperform state-space search based planners (Rintanen 2012). We suppose that during the translation of planning problem into the SAT problem or during the SAT solving itself, similar operations to presented reductions are performed and thus the improvement in coverage is expected to be lower than in the case of state-space search. The verification of this hypothesis and more detailed analysis of the relation between reductions and SAT solver operations are also part of our future work.

During analysis of the reduced problems, we have noticed that in some problems, the reductions can reveal symmetric and (reasonably) decoupled sub-problems. Further analysis and utilization of techniques as (Shleyfman et al. 2015; Wehrle et al. 2015) and (Gnad and Hoffmann 2015) on (partially) reduced problems are left for future work.

Acknowledgments

This research was supported by the Czech Science Foundation (no. 13-22125S and 15-20433Y) and by the Ministry of Education of the Czech Republic within the SGS project (no. SGS13/211/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Re-

Domains	LAMA	$\twoheadrightarrow$	%
airport	**9**	8	1.00
depot	14	**15**	1.07
floortile11	2	**5**	0.91
freecell	13	**14**	1.07
parcprinter08	12	**14**	0.96
parcprinter11	10	**14**	1.00
satellite	13	**14**	**2.24**
scanalyzer08	**13**	12	1.00
scanalyzer11	**13**	12	1.00
sokoban08	**14**	11	1.05
sokoban11	**13**	10	1.04
storage	**10**	9	1.09
tidybot11	**12**	10	0.96
transport08	14	**15**	**2.45**
transport11	8	**15**	**2.56**
trucks	6	**7**	0.94
visitall11	4	**5**	1.00
Total (660)	554	**564**	1.31

Table 4: Impact of polynomial reductions on the coverage of LAMA planner (domains where the coverage differs).

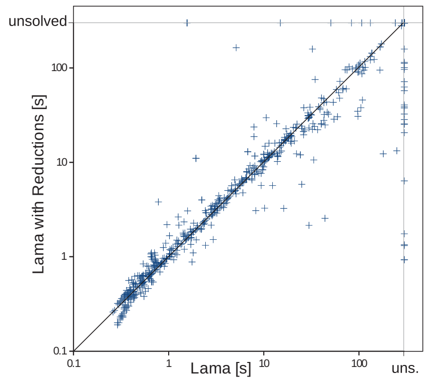

Figure 6: Runtime evaluation (logarithmic axes) of LAMA with and without reductions.

search, Development, and Innovations” (LM2010005), is greatly appreciated.

References

Bacchus, F., and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *AI* 71(1):43–100.

Bäckström, C., and Jonsson, P. 2012. Algorithms and limits for compact plan representations. *JAIR* 44:141–177.

Bäckström, C.; Jonsson, A.; and Jonsson, P. 2012. Macros, reactive plans and compact representations. In *Proc. of ECAI’12*, 85–90.

Bäckström, C. 1992. Equivalence and tractability results for SAS$^+$ planning. In *Proc. of KR’92*, 126–137.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proc. of ECP’99*, 360–372.

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

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *Proc. of IJCAI’09*, 1659–1664.

Coles, A., and Coles, A. 2010. Completeness-preserving pruning for optimal planning. In *Proc. of ECAI’10*, 965–966.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. of IJCAI’71*, 608–620.

Gnad, D., and Hoffmann, J. 2015. Beating LM-Cut with h$^{\max}$ (sometimes): Fork-decoupled state space search. In *Proc. of ICAPS’15*, 88–96.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *Proc. of IJCAI’07*, 1898–1903.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *AI* 143(2):219–262.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

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

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. Distance? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In *Proc. of ECAI’14*, 441–446.

Jonsson, A. 2009. The role of macros in tractable planning. *JAIR* 36:471–511.

Junghanns, A., and Schaeffer, J. 2001. Sokoban: Enhancing general single-agent search methods using domain knowledge. *AI* 129(1-2):219–251.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *AI* 68(2):243–302.

Nissim, R.; Apsel, U.; and Brafman, R. 2012. Tunneling and decomposition-based state reduction for optimal planning. In *Proc. of ECAI’12*, 624–629.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In *Proc. of AAAI’11*, 1004–1009.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39(1):127–177.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *AI* 193:45–86.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proc. of AAAI’15*, 3371–3377.

Slaney, J. K., and Thiébaux, S. 2001. Blocks world revisited. *AI* 125(1-2):119–153.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In *Proc. of ICAPS’12*, 297–305.

Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In *Proc. of IJCAI’15*, 1712–1718.