Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

The Relative Pruning Power of Strong Stubborn Sets and Expansion Core

Martin Wehrle and **Malte Helmert**
University of Basel, Switzerland
{martin.wehrle,malte.helmert}@unibas.ch

Yusra Alkhazraji and **Robert Mattmüller**
University of Freiburg, Germany
{alkhazry,mattmuel}@informatik.uni-freiburg.de

Abstract

In the last years, pruning techniques based on partial order reduction have found increasing attention in the planning community. One recent result is that the expansion core method is a special case of the strong stubborn sets method proposed in model checking. However, it is still an open question if there exist *efficiently computable* strong stubborn sets with strictly higher pruning power than expansion core. In this paper, we prove that the pruning power of strong stubborn sets is strictly higher than the pruning power of expansion core even for a straight-forward instantiation of strong stubborn sets. This instantiation is as efficiently computable as expansion core. Hence, our theoretical results suggest that strong stubborn sets should be preferred to expansion core. Our empirical evaluation on all optimal benchmarks from the international planning competitions up to 2011 supports the theoretical results.

Introduction

Planning as heuristic search is a leading approach for optimal domain-independent planning. However, despite the success of heuristic search for optimal planning in practice, recent results show that there are significant and fundamental restrictions of this approach (Helmert and Röger 2008). Therefore, additional pruning techniques that work orthogonally to pure heuristic search are desirable for tackling the state explosion problem more effectively.

A recently explored pruning technique for planning is *partial order reduction*, which has been originally proposed for computer-aided verification (Valmari 1991; Godefroid 1996). This technique attempts to reduce the size of the generated state space by avoiding an unnecessary blow-up that is induced by interleaving independent operators. Valmari proposed the notion of (strong and weak) *stubborn sets* (1991), which select a subset of applicable transitions in every state that is sufficient to preserve completeness of the overall search algorithm. Recently, partial order reduction has also found increasing attention in the planning community. Chen and Yao (2009) proposed the expansion core method that follows ideas similar to stubborn sets.

Last year, Wehrle and Helmert (2012) showed that the strong stubborn sets method is a generalization of a corrected version of the expansion core method. More precisely, they proved that the pruning power of expansion core is the same as the pruning power of a particular strong stubborn set. However, although this result shows that *there exists* such a strong stubborn set, Wehrle and Helmert did not provide an explicit algorithm to actually compute it. Furthermore, as algorithms for the computation of strong stubborn sets have several choice points that can be instantiated in many different ways, the question remained open whether there are (efficiently computable) strong stubborn sets with *higher* pruning power than expansion core. Alkhazraji et al. (2012) empirically answer this question by demonstrating that this is the case in standard planning domains.

In this paper, we prove that the pruning power of strong stubborn sets is strictly higher than the pruning power of (an optimized version of) expansion core even for a straightforward and efficiently computable instantiation of strong stubborn sets. More precisely, we show that for every reachable state s in a given planning task, every operator that is pruned in s by expansion core is also pruned by this instantiation of strong stubborn sets. Furthermore, we show that there are cases where the pruning power of strong stubborn sets is exponentially higher than the pruning power of expansion core. To obtain these insights, we first present a succinct formulation of expansion core that allows for a deeper understanding of *what* is actually computed by the expansion core method compared to strong stubborn sets. This formulation in turn prepares the ground for a declarative description that shows *why* expansion core is less powerful than strong stubborn sets. We have implemented the instantiation of strong stubborn sets and expansion core in the Fast Downward planning system (Helmert 2006) and evaluated these techniques on all optimal IPC benchmarks until 2011. Our empirical evaluation confirms our theoretical results and shows that partial order reduction can significantly improve the performance of state-of-the-art planning algorithms.

Preliminaries

We consider domain-independent SAS$^+$ planning. In this setting, we are given a finite set of *finite-domain state variables* $\mathcal{V}$ to describe the possible states of the world. More formally, every variable $v \in \mathcal{V}$ has a finite domain $\mathcal{D}(v)$, and a state of the world is defined as a total assignment of the variables in $\mathcal{V}$ which maps each variable to a value in its

domain. A partial state is an assignment from a subset of $\mathcal{V}$ to values in their domains. For a partial state s, we denote the set of variables for which s is defined with $vars(s)$.

Definition 1 (Planning task). A *planning task* is defined as a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ consisting of a finite set of finite-domain state variables $\mathcal{V}$, a finite set of *operators* $\mathcal{O}$, an initial state s_0, and a partial goal state $s_\star$. An operator $o \in \mathcal{O}$ consists of a *precondition* $pre(o)$ and an *effect* $\mathit{eff}(o)$, where $pre(o)$ and $\mathit{eff}(o)$ are partial states.

We write $s[v]$ for the value of variable v in state s. Furthermore, we define the precondition and effect variables of an operator o as $prevars(o) = vars(pre(o))$ and $\mathit{effvars}(o) = vars(\mathit{eff}(o))$. We say that an operator o *reads* a variable v if $v \in prevars(o)$ and that it *modifies* v if $v \in \mathit{effvars}(o)$. A variable v is called *goal-related* if $v \in vars(s_\star)$. For the rest of the paper, we assume without loss of generality that operators have non-empty effects, i.e., $\mathit{effvars}(o) \neq \emptyset$ for all operators $o \in \mathcal{O}$.

An operator o is *applicable* in state s if $pre(o)[v] = s[v]$ for all $v \in prevars(o)$. In this case, the *successor state* $o(s)$ is defined as the state obtained from s by setting the values of the effect variables in $\mathit{effvars}(o)$ to their values in $\mathit{eff}(o)$ and keeping the values of the other variables. Moreover, for a variable v and a state s, we say that an operator o is *v-applicable* in s if o does not have a violated precondition on v in s, i.e., either $v \notin prevars(o)$, or $s[v] = pre(o)[v]$. A *solution* for a planning task is defined as a sequence of operators $o_1, \ldots, o_n$ such that $o_n(\ldots(o_1(s_0))\ldots)$ is defined and leads to a state that includes the goal (i.e., $o_n(\ldots(o_1(s_0))\ldots)[v] = s_\star[v]$ for all goal-relevant variables v).

Furthermore, we need the notion of *domain transition graphs (DTGs)*. For a planning task $\langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ and a variable $v \in \mathcal{V}$, the domain transition graph $DTG(v)$ for v is defined as the directed graph whose vertices are the values $\mathcal{D}(v)$ in the domain of v and which has an edge from d to d' if there is an operator $o \in \mathcal{O}$ such that $\mathit{eff}(o)[v] = d'$, and either $pre(o)[v] = d$ or $v \notin prevars(o)$. Note that for technical reasons (and in contrast to the standard definition), we allow self-loops. We observe that o induces an edge in $DTG(v)$ if and only if $v \in \mathit{effvars}(o)$.

We also need several notions of operator dependency. We say that an operator o *disables* an operator o' if there is a variable $v \in \mathit{effvars}(o) \cap prevars(o')$ such that $\mathit{eff}(o)[v] \neq pre(o')[v]$. We say that o and o' have *conflicting effects* if there is a variable $v \in \mathit{effvars}(o) \cap \mathit{effvars}(o')$ such that $\mathit{eff}(o)[v] \neq \mathit{eff}(o')[v]$. Moreover, we say that operators o and o' *interfere* if o disables o', or o' disables o, or o and o' have conflicting effects.

Finally, we will use the notion of *active operators*. Operators that are not active in a state s are not needed to reach a goal state from s and can be ignored in s without losing completeness or optimality. Following the original work on expansion core (Chen and Yao 2009), we use domain transition graphs to establish this property.

Definition 2 (Active operators). Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task and let s be a state. $Act(s)$ is the set of *active* operators $o \in \mathcal{O}$ that satisfy the following conditions:

1. For every variable $v \in prevars(o)$, there is a path in $DTG(v)$ from $s[v]$ to $pre(o)[v]$, and also from $pre(o)[v]$ to the goal value $s_\star[v]$ if v is goal-related.
2. For every goal-related variable $v \in \mathit{effvars}(o)$, there is a path in $DTG(v)$ from $\mathit{eff}(o)[v]$ to the goal value $s_\star[v]$.

We remark that Def. 2 strengthens the definition of operators that are *DTGactive* as defined by Wehrle and Helmert (2012). The difference between Def. 2 and DTGactive operators is the additional path criterion for goal-related variables in the first condition of Def. 2. For readers familiar with the notion of projected problems, we observe that an operator o is active in a state s iff for all $v \in prevars(o) \cup \mathit{effvars}(o)$, the projection of the problem onto v has a plan from (the projection of) s that includes operator o. Clearly, $Act(s)$ can be identified efficiently for a given state s, and operators that are not in $Act(s)$ need not be considered in s (i.e., can be treated as nonexistent when reasoning about s).

Stubborn Sets

Stubborn sets were introduced by Valmari (1991) and later studied by Godefroid (1996). Recently, the definition of strong stubborn sets has been adapted to the planning setting (Wehrle and Helmert 2012; Alkhazraji et al. 2012).

To define strong stubborn sets, we need some more terminology. A *disjunctive action landmark* in state s is a set of operators such that all operator sequences that lead from s to some goal state contain some operator in the set. A *necessary enabling set* for o in state s is a set of operators such that all operator sequences that lead from s to some goal state and include o contain some operator in the set before the first occurrence of o.

Definition 3 (Strong stubborn set). Let $\langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task, and let s be a state. A set of operators $T_s \subseteq \mathcal{O}$ is a *strong stubborn set* in s if the following conditions hold:

1. For all operators $o \in T_s$ not applicable in s, T_s contains a necessary enabling set in s for o.
2. For all operators $o \in T_s$ applicable in s, T_s contains all operators o' that are active in s and interfere with o.
3. T_s contains a disjunctive action landmark in s.

Strong stubborn sets can be used for pruning: in each state s, it suffices to apply the applicable operators of a strong stubborn set in s (instead of applying *all* applicable operators). Intuitively, this can be considered as a commitment towards making progress on a certain subgoal in the next step of the plan. This commitment is safe in the sense that at least one permutation of each plan remains in the search space. Therefore, pruning with strong stubborn sets maintains completeness and optimality of a search algorithm like A* (Alkhazraji et al. 2012).

Alkhazraji et al. also present a simple fixed-point algorithm for computing strong stubborn sets, adapting an earlier algorithm presented by Godefroid (1996). The algorithm leaves certain design choices, such as the choice of necessary enabling sets, open. We will later show how these design choices can be resolved in such a way that the resulting algorithm strictly dominates the expansion core method, which we describe next.

Expansion Core

The expansion core method (EC) is a state-space pruning technique introduced in the planning community (Chen and Yao 2009; Xu et al. 2011). The original algorithm contained an error and was recently corrected (Wehrle and Helmert 2012). In the following, we introduce this corrected version of EC as a basis for our analysis.

Our presentation of the algorithm is significantly simpler than the original presentation by Chen and Yao. This simplification is obtained by only considering active operators. Chen and Yao already use most of the concepts underlying active operators, but their presentation exploits these concepts only in certain contexts and only in limited ways. By focusing on active operators everywhere, several complex side conditions in the original definitions can be dropped.[1]

We first introduce the notion of potential preconditions.

Definition 4 (Potential precondition). Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task, let $v, v' \in \mathcal{V}$ be variables, let s be a state. Then v is a *potential precondition* of v' in s if there is $o \in Act(s)$ such that o reads v, o is v-applicable in s and o modifies v'.

In the original work on expansion core (Chen and Yao 2009; Xu et al. 2011), the definition of potential preconditions contains a path criterion to rule out some (unnecessary) operators. More precisely, using the notation of Def. 4, the original definition of potential precondition additionally requires the existence of a path in $DTG(v')$ (to the goal vertex if v' is goal-related) from $s[v']$ that includes an edge that is induced by o. In Def. 4, this criterion is already captured because we have restricted the problem to active operators.

In addition to potential preconditions, EC relies on the notion of potential dependents.

Definition 5 (Potential dependent). Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task, let $v, v' \in \mathcal{V}$ be variables, let s be a state. Then v is a *potential dependent* of v' in s if there is $o \in Act(s)$ such that o reads v', o modifies v, and o is v-applicable in s.

The original definition of potential dependents additionally contains the constraint that there must exist a path in $DTG(v')$ from $s[v']$ (to the goal vertex if v' is goal-related) that contains $pre(o)[v']$. Again, in Def. 5 this constraint is captured by the restriction to active operators in s. Furthermore, the original definition requires o to induce an edge in $DTG(v)$ with source vertex $s[v]$. In Def. 5, we formulated this constraint equivalently via v-applicability of o, which will be more convenient for the following investigations in this paper.

The simplified formulation of EC highlights the similarity between potential preconditions and potential dependents. It shows that these two are almost dual notions, but with some subtle difference regarding the exact conditions related to the applicability of o that are required in the two cases. To complete the discussion of EC, we need one more definition.

Definition 6 (Potential dependency graph). Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task, let s be a state. The *potential dependency graph* $PDG(s)$ in s is defined as the directed graph $\langle V, E \rangle$ with $V = \mathcal{V}$ and edge set $E \subseteq V \times V$ with $\langle v, v' \rangle \in E$ iff $v \neq v'$ and v is a potential precondition of v', or v is a potential dependent of v', or there is $o \in Act(s)$ with $\{v, v'\} \subseteq \mathit{effvars}(o)$.

The EC pruning algorithm uses the potential dependency graph to identify operators that need not be applied. Firstly, for a given non-goal state s, EC selects a goal-related variable v with $s[v] \neq s_\star[v]$. Secondly, based on v, EC computes a *dependency closure*. The dependency closure is defined as the minimal set $dc(s)$ of variables that contains v such that there is no edge in $PDG(s)$ from a variable in $dc(s)$ to a variable not in $dc(s)$. Thirdly, $EC(s)$ is defined as the set of operators that modify a variable in $dc(s)$. Finally, the EC pruning algorithm works by only applying the applicable operators in $EC(s)$, ignoring operators not in $EC(s)$.

The Pruning Power of Strong Stubborn Sets

In this section, we investigate the relative pruning power of strong stubborn sets and EC. For this purpose, we present strong stubborn sets and EC in a declarative way as sets of *rules*. In order to show the strict dominance of strong stubborn sets over EC, we also present a third set of rules and show that these rules are strictly more powerful than EC and strictly less powerful than strong stubborn sets.

Expansion Core as Set of Rules

We reformulate the expansion core algorithm so that for every state s, $EC(s)$ is the result of applying a set of rules in s until a fixed point is reached. This transformation makes the algorithm easier to understand and reason about. Let $v_\star$ be a goal-related variable that is not set to its goal value in s. (In goal states, there is nothing to do.) The sets $dc(s)$ and $EC(s)$ are initially empty. In all rules, v and v' denote variables and o denotes an operator that is active in s.

Rule EC1 (initialization). Add $v_\star$ to $dc(s)$.

Rule EC2 (potential preconditions). If $v \in dc(s)$, o reads v, o is v-applicable in s and o modifies v': add v' to $dc(s)$.

Rule EC3 (potential dependents). If $v \in dc(s)$, o modifies v, o reads v' and o is v-applicable in s: add v' to $dc(s)$.

Rule EC4 (conjunctive effects). If $v \in dc(s)$ and o modifies v and v': add v' to $dc(s)$.

Rule EC5 (expansion core). If $v \in dc(s)$ and o modifies v: add o to $EC(s)$.

Due to the monotonicity of the rules, the order of rule application does not matter and a unique fixed point is always reached. It is not hard to verify that the resulting set $EC(s)$ is equal to the result of the previous formulation of the expansion core algorithm based on potential dependency graphs if the same goal-related variable $v_\star$ is chosen for the initialization.

[1] As a consequence of only considering active operators, the algorithm we present here is slightly more powerful than the original one, but in most cases (such as when all operators are active) they behave identically. The purpose of the modification is to simplify presentation, not to increase pruning power.

Operator-based Expansion Core

We now introduce *operator-based expansion core* (*OBEC*), a variant of expansion core that directly computes the set of operators $OBEC(s)$ in the expansion core of state s without explicitly computing a variable set $dc(s)$. This EC variant provides a convenient intermediate step between EC and strong stubborn sets, which are operator-based.

Again, $v_\star$ is a goal-related variable that is not set to its goal value in s. The set $OBEC(s)$ is initially empty. In all rules, $\bar{v}$ and $\bar{v}'$ denote variables and $\bar{o}$ and $\bar{o}'$ denote operators active in s.[2]

The OBEC rules are closely related to the EC rules, and apart from one special case discussed below, $EC(s)$ and $OBEC(s)$ are identical for all states s, assuming that the same goal variable $v_\star$ is chosen for initializing both variants.

Rule OBEC1 (initialization). If $\mathit{eff}(\bar{o})[v_\star] = s_\star[v_\star]$: add $\bar{o}$ to $OBEC(s)$.

Rule OBEC2 (potential preconditions). If $\bar{o} \in OBEC(s)$, $\bar{o}$ modifies $\bar{v}$, $\bar{o}'$ reads $\bar{v}$ and $\bar{o}'$ is $\bar{v}$-applicable in s: add $\bar{o}'$ to $OBEC(s)$.

Rule OBEC3 (potential dependents). If $\bar{o} \in OBEC(s)$, $\bar{o}$ modifies $\bar{v}$, $\bar{o}$ is $\bar{v}$-applicable in s, $\bar{o}$ reads $\bar{v}'$ and $\bar{o}'$ modifies $\bar{v}'$: add $\bar{o}'$ to $OBEC(s)$.

Rule OBEC4 (common effects). If $\bar{o} \in OBEC(s)$, $\bar{o}$ modifies $\bar{v}$ and $\bar{o}'$ modifies $\bar{v}$: add $\bar{o}'$ to $OBEC(s)$.

We now compare the behaviour of EC and OBEC. Before we can state the main result, we need one more definition: a state variable is *trivial* in s if it is not modified by any active operator in s. Trivial variables cannot change their value in any state reachable from s.

Theorem 1.

1. *For all states s, we have $OBEC(s) \subseteq EC(s)$. In other words, the pruning power of OBEC is always at least as large as the pruning power of EC.*
2. *If no variables are trivial in s, then $EC(s) = OBEC(s)$. In other words, EC and OBEC have the same pruning power under this assumption.*

Proof: We prove part 1 by induction over the application of the OBEC rules. Let $OBEC_0(s), \ldots, OBEC_n(s)$ denote the value of $OBEC(s)$ over a sequence of n applications of rules OBEC1–4. We show that $OBEC_i(s) \subseteq EC(s)$ for all $0 \leq i \leq n$, which implies $OBEC(s) \subseteq EC(s)$.

Observation (*): because rule EC5 is the only one that adds operators to $EC(s)$, we know that for each operator $o \in EC(s)$, one of the variables it modifies must be contained in $dc(s)$. With rule EC4, *all* variables it modifies must be contained in $dc(s)$.

To start the induction, we have $OBEC_0(s) = \emptyset \subseteq EC(s)$. Now we assume $OBEC_i(s) \subseteq EC(s)$ and show that $OBEC_{i+1}(s) \subseteq EC(s)$ by distinguishing which of the OBEC rules is applied in the $(i+1)$-th step:

1. **OBEC1** is applied. All operators added to $OBEC_{i+1}(s)$ are in $EC(s)$ due to EC1.
2. **OBEC2** is applied with parameters $\bar{v}$, $\bar{o}$, $\bar{o}'$. We must show $\bar{o}' \in EC(s)$. From OBEC2's conditions, $\bar{o}$ modifies $\bar{v}$ and $\bar{o} \in OBEC_i(s)$. By induction, $\bar{o} \in EC(s)$. From (*) we get $\bar{v} \in dc(s)$. Then EC2 is applicable with $v = \bar{v}$ and $o = \bar{o}'$: any variable modified by $\bar{o}'$ must be in $dc(s)$. From EC5, we get $\bar{o}' \in EC(s)$.
3. **OBEC3** is applied with parameters $\bar{v}$, $\bar{v}'$, $\bar{o}$, $\bar{o}'$. We must show $\bar{o}' \in EC(s)$. We get $\bar{v} \in dc(s)$ by the same reasoning as in the previous case. Then EC3 is applicable with $v = \bar{v}$, $v' = \bar{v}'$ and $o = \bar{o}$, showing $\bar{v}' \in dc(s)$. From EC5, we get $\bar{o}' \in EC(s)$ because $\bar{o}'$ modifies $\bar{v}'$.
4. **OBEC4** is applied with parameters $\bar{v}$, $\bar{o}$ and $\bar{o}'$. We must show $\bar{o}' \in EC(s)$. From OBEC4's conditions, $\bar{o}$ modifies $\bar{v}$ and $\bar{o} \in OBEC_i(s)$. By induction, $\bar{o} \in EC(s)$. From (*) we get $\bar{v} \in dc(s)$. From EC5, we get $\bar{o}' \in EC(s)$ because $\bar{o}'$ modifies $\bar{v}$.

For part 2, we already know $OBEC(s) \subseteq EC(s)$ from part 1, so we need to show $EC(s) \subseteq OBEC(s)$ if no variables are trivial in s. For a sequence $dc_0(s), \ldots, dc_n(s)$ of values taken on by $dc(s)$ while applying rules EC1–4, we show by induction that $OBEC(s)$ contains all operators modifying a variable in $dc_i(s)$ for all $0 \leq i \leq n$. With rule EC5, this establishes $EC(s) \subseteq OBEC(s)$. The claim is clearly true for $i = 0$, so we proceed with the inductive step:

1. **EC1** is applied. We must show that all operators modifying $v_\star$ are contained in $OBEC(s)$. By assumption, $v_\star$ is not trivial, so at least one active operator modifying $v_\star$ exists. The definition of active operators then implies that at least one operator setting $v_\star$ to its goal values exists. Hence at least one operator modifying $v_\star$ is in $OBEC(s)$ because of OBEC1, which implies that all other operators modifying $v_\star$ are also in $OBEC(s)$ because of OBEC4.
2. **EC2** is applied with parameters v, v', o. We must show that all operators modifying v' are contained in $OBEC(s)$. From the conditions of EC2, we get $v \in dc_i(s)$. Hence, inductively, $o_v \in OBEC(s)$ for some arbitrary operator o_v modifying v. (Such an operator exists because by assumption v is not trivial.) Then OBEC2 is applicable with $\bar{o} = o_v$, $\bar{v} = v$ and $\bar{o}' = o$, which shows $o \in OBEC(s)$. Because o modifies v', all other operators that modify v' are in $OBEC(s)$ by rule OBEC4.
3. **EC3** is applied with parameters v, v', o. We must show that all operators modifying v' are contained in $OBEC(s)$. Let o' be such an operator. From the conditions of EC3, we get $v \in dc_i(s)$ and o modifies v. Hence, inductively, $o \in OBEC(s)$. Then OBEC3 is applicable with $\bar{v} = v$, $\bar{v}' = v'$, $\bar{o} = o$ and $\bar{o}' = o'$, which shows $o' \in OBEC(s)$.
4. **EC4** is applied with parameters v, v', o. We must show that all operators modifying v' are contained in $OBEC(s)$. From the conditions of EC4, we get $v \in dc_i(s)$ and o modifies v. Hence, inductively, $o \in OBEC(s)$. From the conditions of EC4, o also modifies v', and thus all other operators modifying v' are also contained in $OBEC(s)$ by rule OBEC4.

■

[2] The bars are used so that parameters of EC rules and OBEC rules can be more easily distinguished in the proof of Theorem 1.

Given Theorem 1, a natural question is whether there exist cases where OBEC offers strictly more pruning power than EC. (Of course, the theorem implies that any such example must include trivial variables.) The answer is that such examples indeed exist: we can construct a family of planning tasks of size $\Theta(n)$ where $\Theta(4^n)$ states are reachable from the initial state when pruning based on EC, but only $\Theta(2^n)$ states are reachable when pruning based on OBEC. For space reasons and because this result is not important to the main contribution of this section, we refer to a technical report for a proof (Wehrle et al. 2013).

Strong Stubborn Sets as Set of Rules

In the following, we show that OBEC is still strictly less powerful than a particular instantiation of strong stubborn sets, which we call SSS-EC. This instantiation is presented as a set of rules in the same fashion as the rules for EC and OBEC. The rules are modelled closely along the requirements for strong stubborn sets in Def. 3, with the choices for disjunctive action landmarks and necessary enabling sets resolved in such a way that dominance over OBEC is achieved.

As previously, $v_\star$ is a goal-related variable that is not set to its goal value in s. The rules iteratively compute a strong stubborn set $SSS(s)$, which is initially empty. In all rules, o and o' denote operators active in s.

Rule SSS1 (initialization). If $\mathit{eff}(o)[v_\star] = s_\star[v_\star]$: add o to $SSS(s)$.

Rule SSS2 (conflicting effects). If $o \in SSS(s)$, o is applicable in s and o and o' have conflicting effects: add o' to $SSS(s)$.

Rule SSS3 (disabling operators). If $o \in SSS(s)$, o is applicable in s and o' disables o: add o' to $SSS(s)$.

Rule SSS4 (disabled operators). If $o \in SSS(s)$, o is applicable in s and o disables o': add o' to $SSS(s)$.

Rule SSS5 (inapplicable operators). If $o \in SSS(s)$ and o is not applicable in s, pick a variable v such that o is not v-applicable and add all operators o' with $\mathit{eff}(o')[v] = \mathit{pre}(o)[v]$. Variables for which o is not v-applicable are called *violated variables*. Variable v is picked from the set of violated variables according to the following rules:

- If there exists a violated variable such that an applicable operator that modifies it (no matter which value it sets) is already contained in $SSS(s)$, then pick such a variable.
- Otherwise, if there exists a violated variable that is modified by o, then pick such a variable v.
- Otherwise, pick an arbitrary violated variable.

Unlike the previous rulesets, the SSS rules contain a history dependency because different cases can apply in rule SSS5 depending on which operators are already contained in $SSS(s)$. This means that $SSS(s)$ is not well-defined in the sense that different orders of triggering the rules can lead to different operator sets. This does not affect the results we will prove, as these do not make any assumptions about the order in which rules are triggered and hence apply to all possible orders. In particular, we will prove that the pruning power of SSS dominates that of OBEC irrespective of order.

Before we compare the OBEC and SSS rules, we observe that the set $SSS(s)$ computed by the SSS rules is a strong stubborn set. Rule SSS1 ensures that $SSS(s)$ contains a disjunctive action landmark (condition 3 of Def. 3), rules SSS2–4 add the required interfering operators (condition 2 of Def. 3), and rule SSS5 adds the required necessary enabling sets (condition 1 of Def. 3).

It would be nice if we could now show a result directly analogous to Theorem 1, showing that $SSS(s) \subseteq OBEC(s)$ for all states s. Unfortunately, this is not generally the case. In fact, we can give a much stronger kind of counterexample: there exist planning tasks for which neither $OBEC(s)$ nor any subset of it is a strong stubborn set, no matter how the choices of disjunctive action landmarks and necessary enabling sets in Def. 3 are resolved.[3]

However, we are still able to show that SSS dominates OBEC in terms of *pruning power*: while the above discussion implies that $SSS(s)$ may contain operators that are not contained in $OBEC(s)$, we can show that all such operators must be inapplicable in s. Therefore, the set of operators considered when using SSS for pruning is always a subset of the operators considered when using OBEC.

We prove this result in two steps. First, we introduce a modified version of the SSS ruleset, called SSS', and argue that SSS and SSS' compute the same set of *applicable* operators. Then, we show that the set of operators computed by the SSS' ruleset is always a subset of $OBEC(s)$.

The SSS' rules are identical to the SSS rules except that SSS4 is replaced by the following rule:

Rule SSS4' (disabled operators, modified version). If $o \in SSS(s)$, o is applicable in s and o disables o', let the *disabled variables* be all variables v such that o modifies v, o' reads v, and $\mathit{eff}(o)[v] \neq \mathit{pre}(o')[v]$. Consider two cases:

1. If o' is v-applicable for at least one disabled variable v: add o' to $SSS(s)$.
2. Otherwise, apply rule SSS5 to o' (i.e., execute SSS5 with o' as the parameter as if o' were contained in $SSS(s)$).

We now show that the original and modified ruleset have the same pruning behaviour.

Proposition 1. *For all states s, the set of applicable operators in $SSS(s)$ is the same for the SSS ruleset (SSS1–5) and modified SSS ruleset (SSS1–3, SSS4', SSS5).*

Proof: We need to compare rule SSS4 to rule SSS4' in the context of the other rules. The two rules are very similar. They have the same precondition, and in cases where case 1. of rule SSS4' applies, they also have the same effect. This leaves situations where case 2. applies. In such situations, o' is not v-applicable in s for any disabled variable. There must be at least one disabled variable (since o disables o'), which implies that o' is inapplicable in s.

Therefore, an execution of SSS4' in the modified ruleset can be simulated by an execution of SSS4 followed by SSS5

[3] The counterexample is somewhat technical and not necessary for the main results of this paper. Therefore, we again refer to a technical report for its presentation (Wehrle et al. 2013).

(applied to o') in the original ruleset. The only difference between the two scenarios is that in one case o' is added to $SSS(s)$ and in the other case it is not. However, this has no further consequences: adding inapplicable operators like o' to $SSS(s)$ cannot trigger any rule other than SSS5, and SSS5 is triggered in both cases. ∎

We can now prove the dominance result for SSS and OBEC.

Theorem 2.

1. *For all states s, we have $SSS(s) \subseteq OBEC(s)$ when using the modified ruleset for SSS.*
2. *For all states s, the set of applicable operators in $SSS(s)$ under the modified or original ruleset is a subset of the set of applicable operators in $OBEC(s)$.*
   In other words, the pruning power of SSS is always at least as large as the pruning power of OBEC.

Proof: We prove part 1 by induction over the application of the modified SSS rules. In several cases, we will say that a certain SSS rule is "equivalent to" or "a special case" of some OBEC rules. By this we mean that the OBEC rule can (at least) add the same operators to $OBEC(s)$ that the SSS rules adds to $SSS(s)$ under the same (or weaker) conditions.

Let $SSS_0(s), \ldots, SSS_n(s)$ denote the value of $SSS(s)$ over a sequence of n applications of the modified SSS rules. We show that $SSS_i(s) \subseteq OBEC(s)$ for all $0 \leq i \leq n$, which implies $SSS(s) \subseteq OBEC(s)$.

To start the induction, we have $SSS_0(s) = \emptyset \subseteq OBEC(s)$. Now we assume $SSS_i(s) \subseteq OBEC(s)$ and show that $SSS_{i+1}(s) \subseteq OBEC(s)$ by distinguishing which of the modified SSS rules is applied in the $(i+1)$-th step:

1. **SSS1** is applied. This rule is identical to OBEC1.
2. **SSS2** is applied with parameters o and o'. This rule is a special case of OBEC4 with $\bar{o} = o$, $\bar{o}' = o'$ and $\bar{v}$ any variable on which o and o' conflict.
3. **SSS3** is applied with parameters o and o'. This rule is a special case of OBEC3 with $\bar{o} = o$, $\bar{o}' = o'$, $\bar{v}'$ any variable on which o' disables o and $\bar{v}$ any variable modified by o. (Such a variable must exist because we require operators to have non-empty effects.)
4. **SSS4'** is applied with parameters o and o'. If the first case of the rule applies and v is the selected disabled variable, this is a special case of OBEC2 with $\bar{o} = o$, $\bar{o}' = o'$ and $\bar{v} = v$. If the second case of the rule applies, then SSS4' behaves like SSS5 with parameter o'.

   Because we are in the second case of SSS4', o' is not v-applicable for any disabled variable, and hence all disabled variables are violated variables in SSS5. Moreover, an applicable operator that modifies these variables is already contained in $SSS(s)$: namely o. Therefore, the first case of SSS5 applies. This means that SSS5 adds to $SSS_i(s)$ only operators that modify some variable v that is also modified by o. With $o \in SSS_i(s)$ and the induction hypothesis, we have $o \in OBEC(s)$, and hence with OBEC5 all other variables modifying v are also contained in $OBEC(s)$.
5. **SSS5** is applied with parameter o. We distinguish the three cases of SSS5:
   - In the first case, $SSS_i(s)$ and hence (by induction hypothesis) $OBEC(s)$ contain an operator that modifies v. Because of OBEC4, $OBEC(s)$ then contains *all* operators that modify v, including the necessary enabling set that SSS5 adds.
   - In the second case, SSS5 only adds operators that share an effect variable (v) with o. Because $o \in SSS_i(s)$, by induction hypothesis $o \in OBEC(s)$. Then all operators that share an affect variable with o are contained in $OBEC(s)$ by rule OBEC4.
   - In the third case, o is v-applicable for all variables v it modifies (otherwise the second case would trigger). Let v be a variable modified by o (one must exist because operators have non-empty effects). Let v' be the violated variables picked by SSS5 (so o reads v'), and let o' be an operator added by SSS5 (so o' modifies v'). We must show $o' \in OBEC(s)$. Because $o \in SSS_i(s)$, by induction hypothesis $o \in OBEC(s)$, and OBEC3 applies with $\bar{o} = o$, $\bar{o}' = o'$, $\bar{v} = v$ and $\bar{v}' = v'$. OBEC3 adds o', concluding the proof.

Part 2 follows directly from part 1 and Proposition 1. ∎

We have now almost completed the comparison of the expansion core and strong stubborn set methods. We know that EC is dominated by OBEC (even with exponential differences in pruning power in some cases involving trivial variables), and we know that OBEC is dominated by SSS. To complete the discussion, we show an exponential separation between OBEC and SSS.

Example 1. *For $n > 0$, let $\Pi_n = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task with the following components:*

- $\mathcal{V} = \{a_1, \ldots, a_n, g\}$
- $\mathcal{O} = \{o_1, \ldots, o_n, \overline{o_1}, \ldots, \overline{o_n}, o_\mathrm{g}\}$
- $pre(o_i) = \{a_i \mapsto 0\}$, $\mathit{eff}(o_i) = \{a_i \mapsto 1\}$ *for* $1 \leq i \leq n$
- $pre(\overline{o_i}) = \{a_i \mapsto 1\}$, $\mathit{eff}(\overline{o_i}) = \{a_i \mapsto 0\}$ *for* $1 \leq i \leq n$
- $pre(o_\mathrm{g}) = \{a_1 \mapsto 1, \ldots, a_n \mapsto 1\}$, $\mathit{eff}(o_\mathrm{g}) = \{g \mapsto 1\}$
- $s_0 = \{a_1 \mapsto 0, \ldots, a_n \mapsto 0, g \mapsto 0\}$
- $s_\star = \{g \mapsto 1\}$

The number of reachable states of Π_n is $2^n + 1$.[4]

Expansion core, in either the EC or OBEC variants, performs no pruning on this task, so the number of reachable states remains $2^n + 1$.

When pruning based on strong stubborn sets using the SSS rules, the number of reachable states is reduced to $2n + 1$.

Proof: It is easy to see that there are $2^n + 1$ reachable states: if $s[g] = 0$, the other variables can take on arbitrary values, which gives 2^n states. The additional state is the state where $s[v] = 1$ for all state variables, which is the state reached after applying s_g.

[4]We assume that no successor states are generated for goal states. Without this assumption, the number of reachable states is 2^{n+1} because all states s with $s[g] = 1$ can be reached once a goal state has been reached.

Expansion core. We show that $OBEC(s)$ contains all operators for all states s. We have $o_{\mathrm{g}} \in OBEC(s)$ by rule OBEC1. All other operators are then added by rule OBEC3 with $\bar{o} = o_{\mathrm{g}}$ and $\bar{v} = g$. (Note that all other operators modify some variable read by o_{g}.)

Strong stubborn sets. We first consider states s where $s[a_i] = 0$ for at least one variable a_i. In such states $SSS(s)$ only contains the operators $o_i, \overline{o_i}, o_{\mathrm{g}}$ for exactly one such a_i: $SSS(s)$ is equal to $\{o_{\mathrm{g}}\}$ after the initialization (rule SSS1), then o_i is added to $SSS(s)$ as a necessary enabling set (rule SSS5, third case, violated variable a_i), and finally, $\overline{o_i}$ is added to $SSS(s)$ because it has a conflicting effect with o_i (rule SSS2).

Of these operators, only o_i is applicable, so only one successor is generated for state s. In particular, $SSS(s)$ never contains an applicable operator $\overline{o_i}$ that sets a variable back from 1 to 0 in this case.

This leaves the case of states s where $s[a_i] = 1$ for all variables a_i. There are two such states, of which one is a goal state and does not generate successors. The other state is $\tilde{s} = \{a_1 \mapsto 1, \ldots, a_n \mapsto 1, g \mapsto 0\}$. $SSS(\tilde{s})$ contains all operators: the stubborn set is again initialized to $\{o_{\mathrm{g}}\}$, but this time o_{g} is applicable. By rule SSS3, we add all operators $\overline{o_i}$ because they disable o_{g}. These in turn bring the operators o_i into the set by rule SSS3 because o_i disables $\overline{o_i}$.

Therefore, all $n + 1$ successors of $\tilde{s}$ are generated. However, for the successor state s_i reached via $\overline{o_i}$, the previous case applies again, and hence the only transition from s_i that is not pruned is the one that takes us back to $\tilde{s}$.

In summary, there are $2n+1$ reachable states: $n+1$ states on the generated path from s_0 to $\tilde{s}$ (including s_0 and $\tilde{s}$), the goal state reached from $\tilde{s}$, and $n-1$ additional states reached from $\tilde{s}$ via operators of the form $\overline{a_i}$. (Note that one of these operators takes $\tilde{s}$ back to the state from which it was generated, so we do not count that state again.) ∎

With this example, we can conclude our theoretical comparison of expansion core and strong stubborn sets by stating the main result.

Theorem 3. *The instantiation of the strong stubborn set method using rules SSS1–5 strictly dominates the expansion core method in terms of pruning power.*

Moreover, there exist families of planning tasks where the number of explored states under the expansion core method is exponential in the number of explored states under the strong stubborn set method.

Proof: Follows directly from Theorem 1, Theorem 2 and Example 1. ∎

The theoretical results suggest that the strong stubborn set method should be preferred over the expansion core method if it can be implemented with comparable or lower overhead.

In the next section, we experimentally compare the two approaches and evaluate their usefulness compared to search algorithms that do not perform partial order reduction.

Experiments

So far, our comparison of strong stubborn sets and expansion core has been purely theoretical. To provide an experimental comparison, we implemented both algorithms in the Fast Downward planning system (Helmert 2006) and combined them with an A* search based on the landmark-cut heuristic (Helmert and Domshlak 2009). Throughout this section, we refer to the strong stubborn set implementation as SSS-EC, to highlight the fact that other variations of strong stubborn sets are conceivable and that this particular variation was designed in such as way that it achieves dominance over expansion core (EC). Considerable care was taken to make sure that SSS-EC and EC were efficiently implemented. As a baseline, we used A* search with the landmark-cut heuristic and no partial order reduction.

	Coverage			Nodes generated		
Domain (problems)	A*	+EC	+SSS-EC	A*	+EC	+SSS-EC
PARCPRINTER-08 (30)	18	±0	**+12**	2431852	100%	**<1%**
PARCPRINTER-OPT11 (20)	13	±0	**+7**	2431225	100%	**<1%**
WOODWK-OPT08 (30)	16	+6	**+12**	5444225	18%	**<1%**
WOODWK-OPT11 (20)	11	+4	**+8**	5443428	18%	**<1%**
SATELLITE (36)	7	±0	**+3**	1744479	64%	**5%**
AIRPORT (50)	26	±0	**+2**	897061	100%	**29%**
OPENSTACKS-OPT08 (30)	18	−2	**+2**	34336295	100%	**53%**
OPENSTACKS-OPT11 (20)	13	−2	**+2**	34209201	100%	**53%**
ROVERS (40)	7	±0	**+1**	1278006	99%	**57%**
ELEVATORS-OPT11 (20)	**17**	−1	−1	15882032	100%	**66%**
ELEVATORS-OPT08 (30)	20	−1	**+1**	16431397	100%	**67%**
PIPESWORLD-TK (50)	**9**	−1	**±0**	594317	100%	**97%**
SCANALYZER-08 (30)	**14**	**±0**	−1	**7879172**	**100%**	**100%**
SCANALYZER-OPT11 (20)	**11**	**±0**	−1	**7879044**	**100%**	**100%**
FREECELL (80)	**15**	−2	−1	**9472652**	**100%**	**100%**
GRIPPER (20)	**7**	−1	**±0**	**10807891**	**100%**	**100%**
MPRIME (35)	**22**	−1	**±0**	**953737**	**100%**	**100%**
TRUCKS (30)	**10**	**±0**	−1	**11678937**	**100%**	**100%**
REMAINING DOMAINS (805)	**488**	**±0**	**±0**	189005201	100%	**87%**
OVERALL (1396)	742	−1	**+45**	358800152	97%	**76%**

Table 1: Comparison of plain A*, A* with EC, and A* with SSS-EC, all guided by the landmark-cut heuristic. Coverage and nodes +EC, +SSS-EC are expressed relative to plain A*.

We evaluated our implementation on all optimal planning instances from the international planning competitions (IPCs) up to 2011 that are supported by the landmark-cut heuristic, which does not support conditional effects or axioms. All experiments were conducted on AMD Opteron CPUs model 2384 (2.7 GHz) with a time limit of 30 minutes and a memory limit of 2 GB per instance.

The overall results are given in Table 1, with those domains aggregated under "remaining domains" where all three configurations solve the same set of instances. The first and striking result is the disparity in performance between EC and SSS-EC. EC does not solve more benchmarks than the baseline – in fact, overall coverage reduces by 1, from 742 tasks to 741 solved tasks. In contrast to this, SSS-EC solves 45 additional tasks compared to the baseline, a very noticeable improvement considering the typically exponential runtime growth rate of optimal planning systems on benchmarks of scaling size.

SSS-EC solves more tasks than the baseline in 10 domains and generates fewer nodes in 26 out of 44 domains (most of

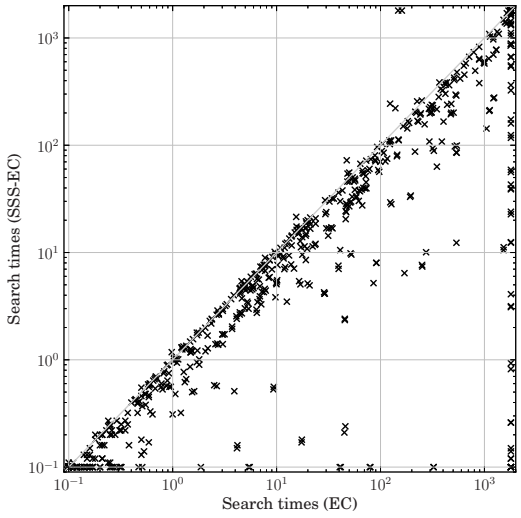

Figure 1: Comparison of search times (in seconds) of EC and SSS-EC across all domains.

which are subsumed in the "remaining domains" row of Table 1). However, there are also five domains where SSS-EC solves one less task than the baseline due to the overhead of computing the stubborn sets. In two of these five cases, SSS-EC runs out memory precomputing the interference relation. In the other three cases, no pruning is possible and the overhead for computing the stubborn sets (a factor of 1.01, 1.5 and 2 in these three cases) pushes runtime beyond the timeout when using SSS-EC.

The reduction in the total number of generated nodes (computed per domain as the sum of node generations over all commonly solved instances) when using SSS-EC can be substantial, with more than 99% of the node generations eliminated in the PARCPRINTER and WOODWORKING domains, 95% of the node generations eliminated in the SATELLITE domain, and several other domains where the number of node generations is almost or more than cut in half. By contrast, EC only substantially reduces the number of node generations in WOODWORKING and SATELLITE.

To provide a quantitative overview of the runtime of the two partial order reduction approaches, Figure 1 shows a scatter plot comparing SSS-EC to EC on all instances in the benchmark set. Clearly, the lower numbers of node generations with SSS-EC lead to faster search on average.

Finally, Table 2 shows details for some of the hardest solved IPC 2011 instances in domains where EC and SSS-EC differ in coverage. This includes OPENSTACKS, PARCPRINTER and WOODWORKING, where SSS-EC performed considerably better than EC, and SCANALYZER, one of the few domains where EC performed better. Focusing on the runtime results in the rightmost columns, SSS-EC is typically 5–6 times as fast as EC in OPENSTACKS, 10–100 times as fast in WOODWORKING and more than 1000 times as fast in PARCPRINTER. Speed is comparable in SCANALYZER, yet SSS-EC solves one task less, presumably due to unfavourable tie-breaking in the last search layer.

	Nodes generated		Search time	
Problem	A*+EC	A*+SSS-EC	A*+EC	A*+SSS-EC
OPENSTACKS-OPT11-P03	276461	**153030**	52.44	**9.57**
OPENSTACKS-OPT11-P06	351555	**197041**	195.67	**33.96**
OPENSTACKS-OPT11-P07	15433	**11437**	127.18	**28.24**
OPENSTACKS-OPT11-P08	1474905	**953425**	1111.89	**208.73**
OPENSTACKS-OPT11-P09	489990	**257104**	541.24	**85.04**
OPENSTACKS-OPT11-P10	319768	**204616**	1239.98	**278.08**
OPENSTACKS-OPT11-P11	—	**1326093**	—	**880.19**
OPENSTACKS-OPT11-P12	—	**700053**	—	**540.7**
OPENSTACKS-OPT11-P13	—	**615457**	—	**661.42**
OPENSTACKS-OPT11-P14	26981	**17785**	542.51	**99.36**
OPENSTACKS-OPT11-P15	—	**1182321**	—	**1715.75**
PARCPRINTER-OPT11-P10	257153	**27**	41.48	**0.1**
PARCPRINTER-OPT11-P12	1630974	**159**	324.05	**0.1**
PARCPRINTER-OPT11-P16	—	**88**	—	**0.12**
PARCPRINTER-OPT11-P17	—	**575**	—	**0.26**
PARCPRINTER-OPT11-P18	—	**1435**	—	**0.93**
PARCPRINTER-OPT11-P19	—	**3798**	—	**3.16**
SCANALYZER-OPT11-P06	**7446600**	**7446600**	1587.66	**1407.81**
SCANALYZER-OPT11-P07	**0**	**0**	**139.0**	221.45
SCANALYZER-OPT11-P09	**0**	**0**	**15.55**	17.45
SCANALYZER-OPT11-P10	**0**	**0**	**50.0**	55.6
SCANALYZER-OPT11-P11	**3800**	—	**160.68**	—
WOODWK-OPT11-P03	135627	**1250**	17.39	**0.17**
WOODWK-OPT11-P07	326266	**1709**	44.91	**0.21**
WOODWK-OPT11-P11	—	**60571**	—	**12.34**
WOODWK-OPT11-P12	6094668	**43836**	1527.22	**10.55**
WOODWK-OPT11-P14	—	**80326**	—	**23.98**
WOODWK-OPT11-P15	322557	**9657**	250.77	**7.65**
WOODWK-OPT11-P16	69752	**3577**	45.36	**2.41**
WOODWK-OPT11-P18	—	**113825**	—	**58.97**
WOODWK-OPT11-P19	107266	**11418**	90.81	**8.01**
WOODWK-OPT11-P20	—	**43227**	—	**39.59**

Table 2: Detailed results on some hard benchmarks. The "nodes generated" columns do not count nodes in the last f layer to eliminate the effect of tie-breaking in the search.

Conclusion

We provided a theoretical and experimental comparison of strong stubborn sets, a partial order reduction method proposed in the model checking community, to the expansion core algorithm, a partial order reduction method proposed for classical planning.

Extending previous work (Wehrle and Helmert 2012) that non-constructively showed that the expansion core approach is a special case of strong stubborn sets without establishing strict dominance, we provided a constructive description of a stubborn-set method that strictly dominates the expansion core algorithm, leading to exponentially smaller search spaces in some cases. The core of our result is a novel, considerably simpler formulation of EC and an operator-based variant of EC in terms of derivation rules.

We also showed that the dominance relationship between stubborn sets and EC is not limited to the theoretical realm: experimentally, the stubborn-set approach offers substantial performance improvements over a state-of-the-art baseline planner, while the expansion core algorithm does not.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Safe Pruning in Optimal State-Space Search (SPOSSS)" and by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems".

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In De Raedt, L.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1659–1664.

Godefroid, P. 1996. *Partial-Order Methods for the Verification of Concurrent Systems — An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer-Verlag.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In Fox, D., and Gomes, C. P., eds., *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, 944–949. AAAI Press.

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

Valmari, A. 1991. Stubborn sets for reduced state space generation. In Rozenberg, G., ed., *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (APN 1989)*, volume 483 of *LNCS*, 491–515. Springer-Verlag.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 297–305. AAAI Press.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core: Additional proofs. Technical Report CS-2013-002, University of Basel.

Xu, Y.; Chen, Y.; Lu, Q.; and Huang, R. 2011. Theory and algorithms for partial order based reduction in planning. *CoRR* abs/1106.5427.