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

Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies

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

Abstract

Strong stubborn sets have recently been analyzed and successfully applied as a pruning technique for planning as heuristic search. Strong stubborn sets are defined declaratively as constraints over operator sets. We show how these constraints can be relaxed to offer more freedom in choosing stubborn sets while maintaining the correctness and optimality of the approach. In general, many operator sets satisfy the definition of stubborn sets. We study different strategies for selecting among these possibilities and show that existing approaches can be considerably improved by rather simple strategies, eliminating most of the overhead of the previous state of the art.

Introduction

Heuristic search is a leading approach for classical domain-independent planning. Recently, for optimal planning, pruning techniques based on partial order reduction have found increasing attention in the planning community. Partial order reduction mitigates the state explosion problem by reducing the unnecessary blow-up of the state space that is induced by independent (permutable) operators.

Originally, partial order reduction was proposed by Valmari (1989) in the area of computer aided verification in the form of strong stubborn sets. In the last years, techniques based on partial order reduction have also been investigated for domain-independent planning, including the expansion core method (Chen and Yao 2009) and a direct adaptation of Valmari's strong stubborn sets (Alkhazraji et al. 2012). Moreover, the theoretical relationships between different partial order reduction techniques have recently been investigated (Wehrle and Helmert 2012; Wehrle et al. 2013). For example, Wehrle et al. (2013) have shown that strong stubborn sets strictly dominate the expansion core method in terms of pruning power if the choice points of the strong stubborn set approach are resolved in a suitable way.

A general question in this context is how to automatically resolve these choice points in such a way that the resulting algorithm is *powerful* (offers significant pruning) and *efficient* (computes stubborn sets quickly). Recent results in the model checking community (Geldenhuys, Hansen, and Valmari 2009; Laarman et al. 2013) show that different strategies for computing stubborn sets vary widely in terms of the resulting pruning power. Recent results in planning (Wehrle et al. 2013) show that current algorithms for computing strong stubborn sets can offer significant improvements of total performance on a large benchmark suite. However, they also demonstrate a nontrivial performance penalty in cases where little or no pruning occurs, leading to fewer problems solved in 5 out of 44 benchmark domains. Ultimately, one seeks for strategies that combine significant pruning power with low computational overhead.

In this paper, we make two contributions towards this research goal. Firstly, we introduce and prove the correctness of a generalization of the strong stubborn set approach that allows us to reduce the sets of operators that need to be considered when verifying the constraints that characterize strong stubborn sets. The resulting sets are smaller than with the original definition, which makes them faster to compute and potentially stronger in terms of pruning power.

Secondly, we investigate different strategies for resolving the choice points in the computation of strong stubborn sets. In particular, we investigate strategies for computing *necessary enabling sets* and to approximate the *interference relation* for operator dependencies. We also briefly investigate the impact of *active operators*, which have been used as a pruning technique in addition to strong stubborn sets.

Our experiments show that rather simple strategies are sufficient for matching the pruning power of recent successful stubborn set approaches for domain-independent planning. In addition, the experiments show that the closer approximation of interfering operators afforded by our generalized definition of stubborn sets significantly reduces the overhead of stubborn set computation, almost completely eliminating the performance penalty in planning domains where little or no pruning occurs.

Preliminaries

We consider classical planning in a notational variant of the SAS$^+$ formalism (e.g., Bäckström and Nebel 1995), extended with nonnegative operator costs. States of the world are described with a finite set of state variables $\mathcal{V}$. Every variable $v \in \mathcal{V}$ has a finite domain $\mathcal{D}(v)$. A *partial state* s is a function defined on a subset of $\mathcal{V}$, denoted as $vars(s)$, which maps every variable $v \in vars(s)$ to an element of

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

its domain $\mathcal{D}(v)$. We write $s[v]$ for the value to which v is mapped. If a partial state is defined on all variables ($vars(s) = \mathcal{V}$), it is called a *state*.

Definition 1 (planning task)**.** *A* planning task *is a 4-tuple* $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$*, where* $\mathcal{V}$ *is a finite set of finite-domain variables,* $\mathcal{O}$ *is a finite set of* operators*,* s_0 *is a state called the* initial state*, and* $s_\star$ *is a partial state called the* goal*.*

Each operator o *has an associated partial state* $pre(o)$ *called its* precondition*, an associated partial state* $\mathit{eff}(o)$ *called its* effect*, and an associated nonnegative number* $cost(o) \in \mathbb{R}_0^+$ *called its* cost*.*

The *precondition variables* $prevars(o)$ of an operator o are the variables for which $pre(o)$ is defined, i.e., $prevars(o) = vars(pre(o))$. Similarly, the *effect variables* of o are defined as $\mathit{effvars}(o) = vars(\mathit{eff}(o))$.

A pair (v, d) with $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$ is called a *fact*. We follow the convention of considering functions as sets of pairs, so $(v, d) \in s$ and $s[v] = d$ are interchangeable notations. If f is a fact and o is an operator with $f \in \mathit{eff}(o)$, then o is called an *achiever* of f. Operators which achieve only one fact are called *unary*.

An operator o is *applicable* in a state s if $pre(o) \subseteq s$. In this case, the *successor state* $o(s)$ of s for operator o is defined as the state that is obtained by updating s according to $\mathit{eff}(o)$: $o(s)[v] = \mathit{eff}(o)[v]$ for all $v \in \mathit{effvars}(o)$ and $o(s)[v] = s[v]$ for all $v \notin \mathit{effvars}(o)$. If o is not applicable in s, the successor state $o(s)$ is undefined. We denote the set of applicable operators in s with $app(s)$.

Definition 2 (plan, solvable, optimal)**.** *Let* $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ *be a planning task and let* s *be one of its states. A* plan *or* solution *for* s *is a finite sequence* $\pi = o_1, \ldots, o_n$ *of operators of* Π *such that applying these operators to* s *in sequence leads to a state that includes the goal, i.e.,* $s_\star \subseteq o_n(o_{n-1}(\ldots(o_1(s))\ldots))$*. The* cost *of* π *is defined as* $\sum_{i=1}^{n} cost(o_i)$*.*

We call a state s solvable *if a plan for* s *exists and* unsolvable *otherwise. A plan for* s *is* optimal *if its cost is minimal among all plans for* s*. We write* $h^*(s)$ *to denote the cost of an optimal plan for* s*, setting* $h^*(s) = \infty$ *if* s *is unsolvable.*

In this paper we consider *optimal* planning, which is the problem of finding an optimal plan for the initial state of a planning task or proving that the initial state is unsolvable.

Successor Pruning

Strong stubborn sets are an example of a *state-based successor pruning* method, used within a state-space search algorithm such as A* (Hart, Nilsson, and Raphael 1968). In this section, we introduce a general notion of *successor pruning functions* and describe conditions under which they preserve the optimality of A*-style algorithms. In the following section, we will describe a generalization of strong stubborn sets that satisfies these conditions.

Definition 3 (successor pruning function)**.** *Let* Π *be a planning task with states* $\mathcal{S}$ *and operators* $\mathcal{O}$*. A* successor pruning function *for* Π *is a function* $succ : \mathcal{S} \to 2^{\mathcal{O}}$ *such that* $succ(s) \subseteq app(s)$ *for all* $s \in \mathcal{S}$*.*

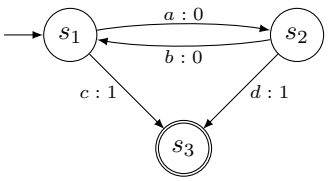

Figure 1: Example for unsafe pruning with zero-cost operators. Operators a and b start optimal plans in s_1 and s_2, but setting $succ(s_1) = \{a\}$ and $succ(s_2) = \{b\}$ makes the task unsolvable.

Successor pruning functions are used within search algorithms to modify the *state expansion* step: when generating the successor states of a given state s, instead of considering the complete set of applicable operators $app(s)$, we only consider the subset $succ(s)$. The other applicable operators are said to be *pruned*.

For optimal planning, we are interested in successor pruning functions which guarantee that optimal solutions for the planning task can still be found with this kind of pruning. We call such successor pruning functions *safe*.

Definition 4 (safe)**.** *Let* $succ$ *be a successor pruning function for a planning task* Π*. We say that* $succ$ *is* safe *if for every state* s*, the cost of an optimal solution for* s *is the same when using the pruned state space induced by* $succ$ *as when using the full state space induced by* app*.*

It is easy to see that general optimal search algorithms such as A* with an admissible heuristic remain optimal when using a safe successor pruning function: general search algorithms like A* treat the state space as a black box, so the known properties of A* apply, except that the algorithm uses the pruned state space instead of the original one. Therefore, if the initial state is solvable A* will find a solution which is optimal in the pruned state space. With a safe successor pruning function such a solution is also optimal in the full state space.

In general, it is of course difficult to determine if a given successor pruning function is safe. Therefore, we will try to find sufficient conditions for safety. As a first attempt to make the definition more operational, we might believe that a successor pruning function is safe if, in every solvable non-goal state, it includes at least one operator that starts an optimal plan from this state. Indeed, in the absence of zero-cost operators, this is a sufficient and necessary criterion for safety.

However, in the general case this criterion is not sufficient, as illustrated in Fig. 1. In state s_1, $\pi_1 = a, d$ is an optimal plan, and in state s_2, $\pi_2 = b, c$ is an optimal plan. However, if we use a successor pruning function based on the first operators of these plans (i.e., $succ(s_1) = \{a\}$ and $succ(s_2) = \{b\}$), it is no longer possible to reach the goal state s_3 from the other states. The underlying problem here is that applying zero-cost operators from an optimal plan does not necessarily make progress towards the goal. Fortunately, we can work around this problem by paying special attention to operators of cost 0.

Definition 5 (strongly optimal)**.** *Let Π be a planning task and s be one of its states. We say that a plan for s is* strongly optimal *if it is optimal for s and includes a minimal number of 0-cost state transitions among all optimal plans for s.*

Strongly optimal plans can be understood as plans that are optimal under a modified cost function where operators of original cost 0 incur a very small cost ε, chosen small enough to only affect tie-breaking between plans of the same original cost. It is easy to see that every operator in a strongly optimal plan "takes us closer to the goal" in a quantifiable way (either $h^*(s)$ or the number of required 0-cost transitions decreases at each step), from which we obtain the following result.

Proposition 1. *Let succ be a successor pruning function such that for every solvable non-goal state s, $succ(s)$ contains at least one operator which is the first operator in a strongly optimal plan for s. Then succ is safe.*

Generalized Strong Stubborn Sets

Strong stubborn sets have been defined in three recent papers on AI planning (Wehrle and Helmert 2012; Alkhazraji et al. 2012; Wehrle et al. 2013). All three definitions are subtly different, and the definition of *generalized strong stubborn sets* we present in this section generalizes all of them.

Intuitively, the previous definitions ensure that the following property holds: *for every plan π for the current state, there exists a permutation of π which is not pruned.* Our generalization only requires that in a solvable state, for *at least one strongly optimal plan*, there exists a permutation which is not pruned.

As a first step, we define *interfering operators*, which are related to the notion of permuting plans.

Definition 6 (interference)**.** *Let o_1 and o_2 be operators of a planning task Π, and let s be a state of Π. We say that o_1 and o_2* interfere *in s if they are both applicable in s, and*

- *o_1* disables *o_2 in s, i.e., $o_2 \notin app(o_1(s))$, or*
- *o_2 disables o_1, or*
- *o_1 and o_2* conflict *in s, i.e., $s_{12} = o_2(o_1(s))$ and $s_{21} = o_1(o_2(s))$ are both defined and differ: $s_{12} \neq s_{21}$.*

This definition follows our earlier work (Wehrle and Helmert 2012), except that the earlier definition is based on a global notion of interference rather than interference in a particular state. The other two cited papers define interference as a syntactic approximation of our definition. In particular, under their definitions, operators can interfere even if they are never jointly applicable (for example because of contradictory preconditions).

Next, we give a definition of *necessary enabling sets*, which are the second important ingredient for the computation of strong stubborn sets.

Definition 7 (necessary enabling set)**.** *Let Π be a planning task, let o be one of its operators, and let Seq be a set of operator sequences applicable in the initial state of Π.*

A necessary enabling set *(NES) for o and Seq is a set N of operators such that every operator sequence in Seq which includes o as one of its operators also includes some operator $o' \in N$ before the first occurrence of o.*

Necessary enabling sets are closely related to *disjunctive action landmarks* (e.g., Helmert and Domshlak 2009), which are sets of operators that contain at least one operator from every plan. Our definition of necessary enabling sets differs from previous ones by allowing to restrict attention to an arbitrary set of operator sequences. In previous definitions, *Seq* must either be the set of *all* applicable operator sequences (Alkhazraji et al. 2012) or all solutions (Wehrle and Helmert 2012; Wehrle et al. 2013).

We can now give the definition of generalized strong stubborn sets.

Definition 8 (generalized strong stubborn set)**.** *Let $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ be a planning task, and let s be a state of Π.*

A generalized strong stubborn set *(GSSS) in s is a set of operators $G \subseteq \mathcal{O}$ with the following properties.*

If s is an unsolvable state or a goal state, every set $G \subseteq \mathcal{O}$ is a GSSS.

If s is a solvable non-goal state, then G has an associated set of operators $E \subseteq \mathcal{O}$ (called the envelope *of G). Let $\Pi_s^E = (\mathcal{V}, E, s, s_\star)$, let Opt be the set of strongly optimal plans for Π_s^E, and let $\mathcal{S}_{Opt}$ be the set of states that are visited by at least one plan in Opt. The following conditions must be true for G to be a GSSS:*

1. *E includes all operators from at least one strongly optimal plan for s (in task Π).*
2. *G contains at least one operator from at least one strongly optimal plan for Π_s^E.*
3. *For every $o \in G$ that is not applicable in s, G contains a necessary enabling set for o and Opt.*
4. *For every $o \in G$ that is applicable in s, G contains all operators in E that interfere with o in any state $s' \in \mathcal{S}_{Opt}$.*

Note that computing a GSSS (or verifying that a given set is a GSSS) does not require complete knowledge of the sets *Opt* and $\mathcal{S}_{Opt}$: if conditions 1.–4. can be verified for an overapproximation of these sets, they must also hold for the actual sets.

Before showing that this definition gives rise to a safe pruning method, we briefly discuss how it generalizes earlier definitions of strong stubborn sets. The first difference of Def. 8 to previous definitions is the newly introduced notion of envelopes. An envelope is an operator set that is known to be sufficient in the sense that we can safely treat all operators outside the envelope as if they did not exist. Note that it is always possible to pick the set of all operators as the envelope, but tighter envelopes might permit more pruning. In particular, we hope that the notion of envelopes might help combine stubborn sets with other pruning techniques, for example based on symmetry elimination (e.g., Pochter, Zohar, and Rosenschein 2011).

Two of the earlier definitions (Wehrle and Helmert 2012; Wehrle et al. 2013) included the notion of *active operators*, which allow disregarding operators that are not part of any plan. Envelopes generalize this by only requiring us to consider operators from *one* plan that is strongly optimal (Condition 1). While this extension might appear obvious, care must be taken regarding the details: as seen in the previous section, it is of critical importance that we consider strongly optimal rather than merely optimal plans here.

Related to this, the second difference of Def. 8 to previous definitions is that Condition 2 only requires that the stubborn set contains one operator from *one* strongly optimal plan, while previous definitions require inclusion of a disjunctive action landmark, which means including one operator from *every* plan, including nonoptimal plans. While we do not exploit this generalization algorithmically in this paper, we think that it could be usefully integrated with the notion of *intended effects* by Karpas and Domshlak (2012).

The final difference to previous definitions is that Def. 8 considers necessary enabling sets and operator interference at a finer level of detail, restricting attention to strongly optimal plans and the states they visit. We will later see that using a more fine-grained notion of interference here than in the work of Wehrle et al. can lead to significant performance improvements in the computation of strong stubborn sets.

We conclude this section by showing that successor pruning based on generalized strong stubborn sets is safe.

Theorem 1. *Let succ be a successor pruning function defined as $succ(s) = G(s) \cap app(s)$, where $G(s)$ is a generalized strong stubborn set in s. Then succ is safe.*

Proof: Let s be a state and G be a GSSS in s. We show that if s is a solvable non-goal state, then G contains an operator which is the first operator in a strongly optimal plan for s. The claim then follows with Proposition 1.

Let E denote the envelope of G, and let Π_s^E, *Opt* and $\mathcal{S}_{Opt}$ be defined as in Def. 8. In the following, we refer to the four conditions of Def. 8 as C1–C4.

Every plan of Π_s^E (for its initial state s) is a plan for state s of Π because the two tasks only differ in their initial states and in the set of operators, and the operator set E of Π_s^E is a subset of the operator set $\mathcal{O}$ of Π. Further, because of C1 at least one strongly optimal plan for s in Π is also present in Π_s^E. It follows that strongly optimal plans for Π_s^E are also strongly optimal for state s in Π. Hence it is sufficient to show that G contains the first operator of some strongly optimal plan for Π_s^E.

Let $\pi = o_1, \ldots, o_n$ be a strongly optimal plan for Π_s^E of which at least one operator is contained in G. Such a plan must exist because of C2. Let $k \in \{1, \ldots, n\}$ be the minimal index for which $o_k \in G$.

We show by contradiction that o_k is applicable in s. Assume it were not applicable. Because $o_k \in G$, C3 guarantees that G contains a necessary enabling set for o_k and *Opt*. *Opt* contains the plan π, so by the definition of necessary enabling sets, G must contain one of the operators in π that occur before o_k. This is a contradiction to the choice of k. It follows that o_k is applicable in s.

Let $s^0, \ldots, s^n$ be the sequence of states visited by π: $s^0 = s$, and $s^i = o_i(s^{i-1})$ for all $1 \leq i \leq n$. Because π is strongly optimal, all these states are contained in $\mathcal{S}_{Opt}$.

It follows that o_k does not interfere with any of the operators $o_1, \ldots, o_{k-1}$ in any of the states s^j: if it did, then from C4 (with $o = o_k$), the interfering operators would be contained in G, again contradicting the minimality of k.

We now show that if o_k is not already the first operator in π, it can be shifted to the front of π. Consider the case where $k > 1$ (otherwise o_k is already at the front). We already know that o_k is applicable in s^0; also, o_1 is applicable in s^0 (or π would not be applicable in s). Because o_1 and o_k do not interfere in s^0, it follows that o_1 does not disable o_k, and hence o_k is also applicable in s^1. This argument can be repeated to show that o_k is applicable in all states s^j with $j < k$. In particular, it is applicable in s^{k-2}, the state right before the one in which it is applied in π. Therefore, in this state, o_{k-1} and o_k are both applicable and do not interfere, so they can be applied in either order, leading to the same state: $s^k = o_k(o_{k-1}(s^{k-2})) = o_{k-1}(o_k(s^{k-2}))$. Hence we can swap o_{k-1} and o_k in π and still have a valid plan.

This argument can be repeated to swap o_k to position $k-2$, $k-3$ and so on, until we end up with the plan $\pi' = o_k, o_1, \ldots, o_{k-1}, o_{k+1}, \ldots, o_n$. Because π' is a permutation of π, it has the same cost and same number of 0-cost operators as π and consists of the same set of operators, so it is also a strongly optimal plan for Π_s^E. Its first operator, o_k, belongs to G. We have found a strongly optimal plan for Π_s^E whose first operator is in G, concluding the proof. ∎

Strategies for Computing GSSSs

In this and the following sections, we investigate different methods for computing generalized strong stubborn sets that exploit the degrees of freedom given by the definition. To keep our discussion from becoming too lengthy, we will focus on the *strategies*, i.e., what is computed rather than how it is computed. Our implementations build on the strong stubborn set implementation by Wehrle et al. (2013).

We will vary three aspects of the algorithm: the envelope (Condition 1), the way that interference is approximated (Condition 4), and the choice of seed operators and necessary enabling sets (Conditions 2 and 3).

Envelope Strategies

We consider two strategies for choosing the envelope of a GSSS. The first strategy is to use the *full* envelope which includes all operators of the planning task. This requires no computation, but means that we might compute unnecessarily large stubborn sets because we need to add interfering operators that are not actually relevant, whose addition can in turn cause further operators to be added.

The second strategy is to use the set of *active* operators as the envelope. Restricting stubborn sets to active operators is not a new idea, but the impact of such a restriction has not been previously evaluated in isolation.

Previous papers give different definitions of active operators. Wehrle and Helmert (2012) define operators as active if they can be part of a plan from the current state. This is a quite informative notion, but intractable to compute. The definition of Wehrle et al. (2013) considers an operator active if it is active, according to the earlier definition, in all single-variable projections of the planning task. We adopt the latter definition in this paper.

Strategies for Approximating Interference

The exact notion of operator interference in our generalized strong stubborn set definition is intractable to compute, so practical approaches need to overapproximate it. Recent

work in model checking shows that fine approximations of interference can improve pruning power of stubborn set approaches (Geldenhuys, Hansen, and Valmari 2009), but does not suggest strategies that are at the same time powerful and efficiently computable.

We will consider two strategies. The first strategy is the purely *syntactic* approximation of operator interference that has been used in previous implementations of strong stubborn sets in planning (Alkhazraji et al. 2012; Wehrle et al. 2013). Under this definition, o_1 is considered to disable operator o_2 if there exists a variable $v \in \mathit{effvars}(o_1) \cap \mathit{prevars}(o_2)$ such that $\mathit{eff}(o_1)[v] \neq \mathit{pre}(o_2)[v]$, and o_1 and o_2 conflict if there exists a variable $v \in \mathit{effvars}(o_1) \cap \mathit{effvars}(o_2)$ with $\mathit{eff}(o_1)[v] \neq \mathit{eff}(o_2)[v]$. Operators interfere if they conflict or either disables the other; the notion is not state-dependent.

The second strategy additionally employs *mutex* reasoning: if operators o_1 and o_2 have mutually exclusive preconditions, they can never be simultaneously applicable and hence cannot interfere. Apart from this additional test, interference is computed as in the syntactic approach. In our implementation, we use the mutex information provided by the Fast Downward planner (Helmert 2009) to determine mutually exclusive preconditions.

Seed Operators and Necessary Enabling Sets

The remaining major choice points when computing generalized strong stubborn sets that we explore in this paper is the choice of *seed operators*, i.e., the operator(s) that are included in the stubborn set to satisfy Condition 2 of Def. 8, and the choice of necessary enabling set for inapplicable operators in Condition 3. These two choices are related: for seed operators, we need operators that can help us towards achieving currently unsatisfied goal conditions, while necessary enabling sets consist of operators that can help us towards achieving currently unsatisfied operator preconditions. Due to this similarity, we handle these choice points together.

All previous approaches for computing strong stubborn sets in planning resolve these choice points in a similar way: for Condition 2, they select a variable v with a defined goal value where the current state has a different value for v, and then include all achievers of this variable in the stubborn set. For Condition 3, they select a variable v for which the given operator has an unsatisfied precondition in the current state and then include all achievers of this precondition in the stubborn set.

We follow the same approach in this paper and investigate different strategies for deciding *which* unsatisfied goal/precondition variable to choose in cases where there are multiple options. Different strategies for resolving these choices have not yet been compared in previous work, but as we will see in the following, resolving these choices in a reasonable way can be rather critical for performance. Because we discuss this choice point more thoroughly than envelopes or interference relations, it deserves its own section, which comes next.

Strategies for Choosing Unsatisfied Conditions

In this section, we investigate various strategies for selecting from a set of unsatisfied conditions (either operator preconditions or goals). As discussed in the previous section, this is a subproblem that arises within strategies for determining seed operators or necessary enabling sets.

Roughly speaking, the choice of unsatisfied condition determines which subproblem the search algorithm should focus on next. For example, in a planning task with two unsatisfied and causally unrelated goals, the stubborn set will only contain operators that are relevant to the chosen goal.

Previous work in AI planning has not given this question much consideration. In the SSS-EC algorithm by Wehrle et al. (2013), unsatisfied conditions are chosen in such a way that dominance over the Expansion Core algorithm (Chen and Yao 2009) is guaranteed, but Expansion Core itself resolves choices of this kind arbitrarily. In the only other paper that reports experimental results for a stubborn set method (Alkhazraji et al. 2012), the question is not considered at all. A look at the implementations of both algorithms shows that (apart from the special considerations in SSS-EC related to the Expansion Core algorithm), when choosing between conditions on two variables v and v', the one that occurs earlier in the variable order of the underlying Fast Downward planner is preferred. The variable order in Fast Downward is in turn based on an approximate topological sorting of the causal graph of the task (Helmert 2006). In the following, we refer to these selection strategies as *SSS-EC* and *Fast Downward*.

Static Variable Orderings

Intuitively, it appears attractive to select unsatisfied conditions in such a way that the search algorithm does not "jump" between different goals and subgoals too often. Again considering the case of two causally unrelated goals, if the search algorithm selects the same unsatisfied goal in a child state as in its parent whenever possible, then the search algorithm will solve the two subtasks sequentially, which appears to be a very reasonable approach.

One simple way to ensure that the same condition is chosen in parent and child states most of the time is to define a fixed ordering between variables and prefer unsatisfied conditions on variables that are minimal according to this ordering. We call such a strategy a *static variable ordering strategy*. The Fast Downward strategy discussed above is a static variable ordering strategy.

In our experiments, we will compare the Fast Downward order to a natural baseline variable ordering called *static small*. Under the static small ordering, variables that appear in the effects of fewer operators are always preferred over variables that appear in the effects of more operators.

We now give a simple example illustrating how static variable orderings can lead to exponentially smaller state spaces than other strategies for choosing unsatisfied conditions.

Example 1. *Let* $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ *be a planning task with*

- $\mathcal{V} = \{v_1, \ldots, v_n\}$
- $\mathcal{O} = \{o_i^j \mid 1 \leq i \leq n, 1 \leq j \leq 3\}$

- $pre(o_i^1) = \{v_i \mapsto 0\}$, $\mathit{eff}(o_i^1) = \{v_i \mapsto 1\}$ *for* $1 \leq i \leq n$
- $pre(o_i^2) = \{v_i \mapsto 1\}$, $\mathit{eff}(o_i^2) = \{v_i \mapsto 0\}$ *for* $1 \leq i \leq n$
- $pre(o_i^3) = \{v_i \mapsto 1\}$, $\mathit{eff}(o_i^3) = \{v_i \mapsto 2\}$ *for* $1 \leq i \leq n$
- $cost(o) = 1$ *for all* $o \in \mathcal{O}$
- $s_0 = \{v_i \mapsto 0 \mid 1 \leq i \leq n\}$
- $s_\star = \{v_i \mapsto 2 \mid 1 \leq i \leq n\}$

We assume a GSSS algorithm that uses the full envelope. When expanding state s, the GSSS computation will select a variable v_i with $s[v_i] \neq 2$, and it will generate exactly the applicable operators that modify the selected variable v_i. (Note that o_i^2 and o_i^3 interfere in s if $s[v_i] = 1$.)

If the unsatisfied goal $v_i \mapsto 2$ is selected according to a static variable order, the size of the reachable state space is only $\Theta(n)$: only states along a single optimal solution are generated. (All states generated via operators o_i^2 are duplicates in this case.)

However, without a static order, it is possible that $\Omega(2^n)$ states are generated. To see this, let $\sigma_0, \ldots, \sigma_{2^n-1}$ be a Gray code sequence over $\{0,1\}^n$ with $\sigma_0 = (0, \ldots, 0)$, i.e., a sequence of n-tuples over $\{0,1\}$ where each possible n-tuple occurs exactly once and consecutive elements differ in exactly one position. (It is well-known that such sequences exist.) We identify such n-tuples with states of Π where $s[v_i] \in \{0,1\}$ for all $1 \leq i \leq n$. Consider a goal selection strategy which, in a state s which corresponds to the tuple σ_m with $m < 2^n - 1$ selects the unsatisfied goal $v_i \mapsto 2$ where i is the unique tuple position in which σ_m and σ_{m+1} differ. It is easy to verify that all 2^n states in the Gray code sequence are part of the reachable state space.

Relationship to Tunnel Macros

The idea to avoid jumping around unnecessarily in a state-space search has also motivated other pruning methods (not based on stubborn sets). One such method is *action tunneling*, a pruning technique that has originally been proposed for the Sokoban puzzle by Junghanns and Schaeffer (2001), generalized to domain-independent planning with unary tunneling operators by Coles and Coles (2010), and further generalized by Nissim, Apsel, and Brafman (2012). Here we follow the definition of Coles and Coles for unary operators, which is sufficient for the following discussion.

Definition 9 (tunnel macro, based on Coles & Coles 2010)**.** *Let $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ be a planning task, and let $o \in \mathcal{O}$ be an operator with $\mathit{eff}(o) = \{v \mapsto d\}$. Let $s_{min} := \{(v,d)\} \cup \{(v', d') \in pre(o) \mid v' \neq v\}$ (this is a set of facts guaranteed to hold in the successor state $o(s)$).*

Then o allows a tunnel *if $s_\star(v) \neq d$ and every operator o' with $pre(o')[v] = d$ is unary*[1]*, has an effect on v, and $pre(o') \subseteq s_{min}$. If o allows a tunnel, then only these operators o' may be applied after o.*

[1]Coles and Coles give a slightly more general definition that additionally allows o' to affect variables of *irrelevant resources*. However, as noted by Nissim, Apsel, and Brafman (2012), these are essentially variables that are useless and can be removed from the planning task.

The previous example suggests similarities between tunneling and stubborn set methods with static variable orders: in both cases, after setting variable v_i to the "intermediate" value 1, v_i must be modified again immediately afterwards. However, we will show that the two techniques are incomparable: either can lead to exponentially smaller reachable state spaces than those obtained by the other method.

For one direction of this argument, we again refer to Example 1. As discussed there, stubborn-set techniques with a static variable order only generate a reachable state space of size $\Theta(n)$. While tunneling can skip over states with $s[v_i] = 1$, it will generate all 2^n possible states where variables have values in $\{0, 2\}$.

The following example shows that there are also cases where the reachable state space of the stubborn set method (even with a static variable order) is larger than for tunneling by an exponential factor in the size of the planning task.

Example 2. *Let $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ be a planning task with*

- $\mathcal{V} = \{v_1, \ldots, v_n, g\}$
- $\mathcal{O} = \{o_i^j \mid 1 \leq i \leq n, 1 \leq j \leq 4\}$
- $pre(o_i^1) = \{v_i \mapsto 0\}$, $\mathit{eff}(o_i^1) = \{v_i \mapsto 1\}$
- $pre(o_i^2) = \{v_i \mapsto 1\}$, $\mathit{eff}(o_i^2) = \{v_i \mapsto 2\}$
- $pre(o_i^3) = \{v_i \mapsto 2\}$, $\mathit{eff}(o_i^3) = \{v_i \mapsto 3\}$
- $pre(o_i^4) = \{v_i \mapsto 3\}$, $\mathit{eff}(o_i^4) = \{g \mapsto 1\}$
- $cost(o) = 1$ *for all* $o \in \mathcal{O}$
- $s_0 = \{v \mapsto 0 \mid v \in \mathcal{V}\}$
- $s_\star = \{g \mapsto 1\}$

Strong stubborn sets obtain no pruning: in a non-goal state, every disjunctive action landmark includes either o_j^4 or (if $s[v_i] \neq 3$) one of the operators "on the way" to o_i^4. After adding necessary enabling sets for inapplicable operators, the stubborn set includes all applicable operators. Hence, states with all value combinations of all v_i are reachable, and the size of the reachable state space is $\Theta(4^n)$.

In contrast, with action tunneling, all applicable operators in s_0 induce a tunnel that avoids the exponential blow-up induced by "intermediate" states with variable values in $\{1,2\}$: in a state where $s[v_i] \in \{1,2\}$, this variable must be increased immediately afterwards, so at most one variable can hold a value in $\{1,2\}$ in a reachable state. The size of the reachable state space is therefore reduced to $\Theta(n2^n)$ (where the factor n is caused by the fact that one of the n variables can hold a value in $\{1,2\}$).

Note that the example assumes a *non-generalized* strong stubborn set approach where the computed stubborn set must be seeded by a disjunctive action landmark. Better pruning, on par with tunneling, can be obtained with the generalized approach by setting a tight envelope or seeding the stubborn set with a non-landmark to satisfy Condition 1 of the GSSS definition. However, in both cases the required knowledge that the GSSS constraints are satisfied would need to come from an additional source of information (such as symmetry reduction techniques).

We conclude from the two examples:

Proposition 2. *Strong stubborn sets and action tunneling are incomparable techniques in terms of pruning power.*

Dynamic Strategies

In contrast to a static variable order, dynamically choosing unsatisfied conditions in every state offers more flexibility – obviously, every static strategy can be trivially simulated with a dynamic approach.

Perhaps the simplest conceivable dynamic strategy is the *random* one, which always selects among unsatisfied conditions uniformly randomly. Following the previous discussion, we would expect this random strategy to jump between subgoals more than a static variable order would, and we expect that this would be detrimental to performance. We include the random strategy in our experimental evaluation to test this hypothesis and to serve as a baseline for more sophisticated approaches.

The SSS-EC strategy used by Wehrle et al. (2013) mentioned at the beginning of this section is also an example of a dynamic strategy. The precise choice is quite complicated and depends on the current state of the stubborn set computation (see Rule SSS5 in their paper). The reason for their complicated rule is that it partially mimics design aspects of the Expansion Core algorithm, as the objective of SSS-EC is to find a stubborn set method that dominates the Expansion Core method in terms of pruning power.

The last two strategies we consider are motivated by the idea that an intuitive measure for the quality of a stubborn set is its size: all other things being equal, it seems reasonable to prefer smaller stubborn sets because they are likely to contain fewer applicable operators, maximizing pruning.

The first of these strategies is called *dynamic small* and is a dynamic counterpart of the *static small* strategy discussed previously. Whenever we must select a fact from a set of unsatisfied conditions, we count the achievers for each fact that are not already part of the currently computed stubborn set. Facts minimizing this number are preferred.

The second strategy, which we call the *Laarman* strategy, comes from the model checking literature (Laarman et al. 2013). It takes into account that only *applicable* operators in the stubborn set affect the amount of pruning and hence counts inapplicable operators less heavily than applicable ones when selecting among unsatisfied conditions. In more detail, it assigns a *weight* to a necessary enabling set, defined as the sum of weights of the contained operators, where the weight of operator o is defined as

$$weight(o, s, G) := \begin{cases} 1, & \text{if } o \notin app(s) \text{ and } o \notin G \\ K, & \text{if } o \in app(s) \text{ and } o \notin G \\ 0, & \text{otherwise} \end{cases},$$

where G is the GSSS currently being computed and $K \geq 1$ is a parameter of the method. In our experiments, we use $K = 10$.[2] Note that it is reasonable to assign a non-zero weight to inapplicable operators because including them in the stubborn set can indirectly lead to the inclusion of further applicable operators.

[2]Laarman et al. do not give conclusive evidence regarding good parameter choices; in our experiments, we use $K = 10$ (we observed very similar coverage results for $K = 5$, $K = 20$, and $K = 100$, which differed at most by 1 compared to $K = 10$).

envelope/ interference	static orders		dynamic orders			
	FD	small	random	small	Laarman	SSS-EC
full/syntactic	*801*	801	759	789	789	798
full/mutex	808	806	778	796	795	**814**
active/syntactic	802	803	765	793	792	*800*
active/mutex	807	809	784	799	797	813
baseline (no successor pruning): 763						

Table 1: Coverage results overview. Best configuration in bold, previous algorithms from the literature (Alkhazraji et al. 2012; Wehrle et al. 2013) in italics.

Experimental Evaluation

We now experimentally evaluate the GSSS strategies presented in the previous two sections. To summarize, we vary the following three different aspects of the computation:

- envelope: *full* or *active*
- interference: *syntactic* or *mutex*
- unsatisfied condition choice: static strategies *FD* (Fast Downward) and *static small*; dynamic strategies *random*, *SSS-EC*, *dynamic small* and *Laarman*

We evaluate all 24 possible configurations, as well as a baseline algorithm without successor pruning, in the context of an A* search with the LM-Cut heuristic (Helmert and Domshlak 2009). The base implementation is identical to Wehrle et al. (2013), where in particular the interference relation of the operators is computed in a preprocessing step. The combination *full-syntactic-FD* is the algorithm by Alkhazraji et al. (2012), and *active-syntactic-SSS-EC* is the algorithm by Wehrle et al. (2013).

We used Intel Xeon E5-2660 CPUs (2.2 GHz) with a 30 minute timeout and 2 GB memory bound. We tested all tasks from all IPC 1998–2011 domains supported by the planner.

Overall Coverage. Table 1 shows an overview of *coverage*, i.e., total number of tasks solved within the given time and memory bounds. Coverage is a commonly used measure for evaluating optimal planners. For example, it was used at the previous two IPCs to rank optimal planners.

We remark that even small improvements in coverage are hard to achieve for optimal planners on this benchmark suite because tasks tend to increase in difficulty rapidly. For example, at IPC 2011, the planners ranked 2nd to 5th were separated by less than 2% in terms of coverage. (The winner was set off by a larger margin of 9–10%, but it was a portfolio system, while we compare individual algorithms.)

The table shows the following trends:

- Using the *active operator* envelope instead of the *full* envelopes is mildly beneficial in most cases. In 10 of the 12 settings for the other parameters, more tasks (between 1–6) are solved when using active operators, whereas the performance decreases in two settings (by 1). Unfortunately, a performance decrease in particular occurs in the configuration which is best overall. All in all, using active operators appears at best mildly beneficial.
- Using the *mutex-based* interference criterion instead of the syntactical criterion used in previous work leads to

substantial improvements of 5–19 tasks in all 12 combinations of the other parameters. This includes significant gains (13–16 tasks) for some of the best configurations. This technique clearly seems to be worth using.

- To choose between unsatisfied conditions, the *random* strategy is worst by a wide margin. For all four combinations of the other parameters, it solves 13–30 tasks fewer than the second-worst strategy, and in the combination with a full envelope and syntactic interference it is even worse than the baseline which does not perform any successor pruning (759 vs. 763 solved tasks).

  Among the other strategies, the ones using static variable orders (*FD* and *small*) and the dynamic strategy *SSS-EC* perform best, outperforming the ones that attempt to minimize the size of stubborn sets dynamically (*dynamic small* and *Laarman*).

- The new best approach, *full-mutex-SSS-EC* solves 814 tasks, compared to 801 (Alkhazraji et al. 2012) or 800 (Wehrle et al. 2013) for the previous state of the art and to 763 without successor pruning.

As a baseline, we also report experimental results for blind search, which solves 510 tasks without successor pruning. Applying *SSS-EC* with the full envelope yields a coverage of 524 and 550 (with syntactic and mutex-based interference, respectively), whereas *SSS-EC* with the active operator envelope yields a coverage of 526 and 552 (syntactic/mutex). Overall, we observe similar trends as with LM-Cut: *SSS-EC* is beneficial, active operators are mildly beneficial, whereas the mutex-based operator interference increases coverage more significantly.

Amount of Pruning. To get some estimate of the amount of pruning obtained by each of the methods, Table 2 shows the number of generated states for each algorithm. Results are reported as the geometric average over all tasks solved by all approaches, including the non-pruning baseline. Note that the true advantage of the pruning methods is larger because there are many cases in which pruning is immensely beneficial but which do not show up in the averages because they cannot be solved by the baseline.

The contribution of active operator envelopes is again marginal. Interestingly, the same holds for mutex-based interference, which significantly improves coverage. The experimental data reveals that the tighter interference relation can reduce the overhead of the stubborn set computation rather than cause more pruning. A closer look at the results reveals the following reasons. First, applying the mutex-based interference relation can reduce the size of the stubborn sets, especially with respect to the number of inapplicable operators. We observe a particularly large reduction in the Logistics00 domain: for example, the size of the strong stubborn sets in the initial states is roughly cut in half in the largest instances 15-0 and 15-1 (158 vs. 83 operators). An additional reason for the reduced overhead (on a more technical level) is the reduced number of operator processing steps during the fixed-point computation of *SSS-EC*. In particular, the *SSS-EC* implementation requires processing the operators disabled by an operator already contained in the set, and fewer dependencies in turn require fewer such processing steps. Finally, with fewer dependencies the algorithm can afford to precompute the operator interference relation more often within the memory bounds, allowing us to solve the additional instances in the Scanalyzer domain. Overall, we observe that applying the mutex-based interference makes an efficient implementation of strong stubborn sets easier to achieve.

envelope/	**static orders**		**dynamic orders**			
interference	FD	small	random	small	Laarman	SSS-EC
full/syntactic	*10252*	10022	13088	8917	9252	9261
full/mutex	9521	9333	12075	8995	9333	9261
active/syntactic	10099	9976	12816	**8912**	9252	*9252*
active/mutex	9403	9308	11594	8992	9330	9252
baseline (no successor pruning): 15840						

Table 2: Generated states (geometric mean on commonly solved tasks). Best configurations in bold, previous algorithms (Alkhazraji et al. 2012; Wehrle et al. 2013) in italics.

		full		active	
coverage	baseline	syntactic	mutex	syntactic	mutex
airport (50)	28	**29**	**29**	**29**	**29**
logistics00 (28)	20	20	**21**	20	**21**
openstacks-opt08 (30)	19	19	**22**	20	**22**
openstacks-opt11 (20)	14	14	**17**	15	**17**
parcprinter-08 (30)	18	**30**	**30**	**30**	**30**
parcprinter-opt11 (20)	13	**20**	**20**	**20**	**20**
parking-opt11 (20)	**3**	2	2	2	2
rovers (40)	7	**10**	**10**	**10**	**10**
satellite (36)	7	11	**12**	11	**12**
scanalyzer-08 (30)	**15**	12	**15**	12	**15**
scanalyzer-opt11 (20)	**12**	9	**12**	9	**12**
sokoban-opt08 (30)	**30**	29	**30**	29	**30**
visitall-opt11 (20)	**11**	10	**11**	10	10
woodworking-opt08 (30)	17	**27**	**27**	**27**	**27**
woodworking-opt11 (20)	12	**19**	**19**	**19**	**19**
Sum (424)	226	261	**277**	263	276
Overall sum (1396)	763	798	**814**	800	813

Table 3: Coverage details for *SSS-EC*. Best results in bold. Domains where all algorithms perform identically are not shown individually, but included in the overall sum.

Comparing strategies for selecting unsatisfied conditions, we see that the dynamic *small* approach, which attempts to minimize the stubborn set, obtains most pruning, followed by *Laarman* and *SSS-EC*. The inferior coverage of *small* and *Laarman* compared to *FD* and *SSS-EC* can be explained by the (considerable) overhead incurred by dynamically counting how many new operators would need to be added to the stubborn set at every stage. The random strategy offers much less pruning than all other methods, even though it still offers substantial pruning compared to the baseline.

Per-Domain Coverage. We conclude our discussion of experimental results with a look at per-domain coverage for the *SSS-EC* strategy (Table 3). We focus on three configurations: *baseline* (no successor pruning), *active/syntactic* (the state-of-the-art algorithm by Wehrle et al., 2013), and *full/mutex* (the best strategy introduced in this paper).

The new approach solves 51 more tasks than the baseline (spread over 10 domains) and 14 more tasks than the previous state of the art (in 8 domains). The new approach actually dominates the previous state of the art on a *per-task* basis: every task solved by the Wehrle et al. approach is solved by the new method. A similar dominance result over the baseline without successor pruning is *almost* obtained: there is a single task in the *parking-opt11* domain that can be solved by the baseline, but not by the stubborn set approach. A look at the detailed experimental results reveals that no pruning occurs in this domain and the baseline requires more than 1791 seconds to solve this task. With an overall timeout of 1800 seconds, it is clear that even a very modest overhead of the stubborn set approach is enough to leave this task unsolved.

Conclusions

We provided a generalization and unification of earlier definitions for strong stubborn sets and performed the first comparison of different strategies for computing stubborn sets. Our experiments show that the performance of existing approaches can be improved with rather simple methods like the mutex-based interference criterion.

For the future, it would be interesting to exploit the additional freedom in our generalized definition to combine stubborn-set approaches with other pruning methods. Tunnel macros are particularly promising in this regard because we have shown that there are cases where they offer exponentially more pruning than stubborn sets (and vice versa).

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)".

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In *Proc. ECAI 2012*, 891–892.

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

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

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

Geldenhuys, J.; Hansen, H.; and Valmari, A. 2009. Exploring the scope for partial order reduction. In *Proc. ATVA 2009*, 39–53.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AIJ* 173:503–535.

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

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proc. ICAPS 2012*, 92–100.

Laarman, A.; Pater, E.; van de Pol, J.; and Weber, M. 2013. Guard-based partial-order reduction. In *Proc. SPIN 2013*, 227–245.

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

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

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proc. APN 1989*, 491–515.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In *Proc. ICAPS 2012*.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In *Proc. ICAPS 2013*, 251–259.