Sleep Sets Meet Duplicate Elimination

Yusra Alkhazraji
University of Freiburg, Germany
alkhazry@informatik.uni-freiburg.de

Martin Wehrle
University of Basel, Switzerland
martin.wehrle@unibas.ch

Abstract

The sleep sets technique is a path-dependent pruning method for state space search. In the past, the combination of sleep sets with graph search algorithms that perform duplicate elimination has often shown to be error-prone. In this paper, we provide the theoretical basis for the integration of sleep sets with common search algorithms in AI that perform duplicate elimination. Specifically, we investigate approaches to safely integrate sleep sets with *optimal* (best-first) search algorithms. Based on this theory, we provide an initial step towards integrating sleep sets within A* and additional state pruning techniques like strong stubborn sets. Our experiments show slight, yet consistent improvements on the number of generated search nodes across a large number of standard domains from the international planning competitions.

Introduction

State space search is a popular approach to solve search and planning tasks. To tackle the state explosion problem, techniques like move pruning (Burch and Holte 2012; Holte and Burch 2014) and sleep sets (Godefroid and Wolper 1992; Godefroid 1996; Wehrle and Helmert 2012; Holte, Alkhazraji, and Wehrle 2015) have been investigated. Both move pruning and sleep sets are path-dependent pruning techniques that preserve the reachability of all reachable states. Such techniques can be beneficial to reduce the number of repeated explorations of equal states in tree search algorithms like IDA*. In addition, previous work in computer aided verification (e.g., Godefroid 1996) showed that the combination with state reduction techniques can yield synergy effects when applied with search algorithms that perform duplicate elimination. Furthermore, for efficiency reasons, some form of duplicate elimination (like cycle detection) is often performed also for tree search algorithms. Overall, the question arises to which extent duplicate elimination and path-dependent pruning can safely be integrated.

Generally, integrating path-dependent techniques with graph search algorithms is non-trivial because subtle interactions may occur that can render search algorithms suboptimal or incomplete. For example, Holte (2013) has studied the interaction of duplicate elimination with move pruning, showing that rather strict requirements are needed for a safe integration. In contrast, in their original form, sleep sets have been proposed in several variants for depth-first search with duplicate elimination, which turned out to be incomplete later on, as shown by Koutny and Pietkiewicz-Koutny (1995) as well as by Bosnacki et al. (2009). However, although corrected versions have been proposed, sleep sets have hardly been investigated in combination with further search algorithms that perform duplicate elimination. In particular, the general question how they can be applied with *optimal* (best-first) graph search algorithms has not yet been answered so far. Although short counter-examples are usually preferred in the area of computer aided verification for the purpose of debugging faulty system models, approaches that guarantee optimality have mostly not been considered by the verification community. In contrast, optimal solutions are often desired for search and planning problems.

In this paper, we develop the theoretical basis for the integration of sleep sets with common search algorithms in AI that perform duplicate elimination. To prepare the ground for this integration, we provide a literature analysis of four main variants of sleep sets when combined with different forms of duplicate elimination and graph search algorithms. Based on this analysis, we provide the theoretical foundations on the combination of sleep sets with common best-first (optimal) search algorithms that perform duplicate elimination. Furthermore, as a first step towards the application of sleep sets for state pruning, we propose an integration of A* with sleep sets and *strong stubborn sets* for search and planning (e.g., Alkhazraji et al. 2012). Our implementation in the Fast Downward planning system (Helmert 2006) shows slight, yet consistent improvements regarding the size of the generated search space on a large class of domains from the international planning competitions.

Background

We consider search and planning problems formalized in the SAS$^+$ formalism (Bäckström and Nebel 1995), which is based on a finite set $\mathcal{V}$ of finite-domain state variables. A *partial state* is defined as an assignment from a subset of $\mathcal{V}$, denoted with $vars(s)$, to the corresponding domain of the variables in $vars(s)$. For a partial state s and variable $v \in vars(s)$, the value of v in s is denoted with $s[v]$. A *state* s is a partial state with $vars(s) = \mathcal{V}$. We assume a given *initial* state s_0 and a partial *goal* state $s_\star$. We will denote

partial states as sets of mappings from variables to values.

States can be transformed with *operators* $o = \langle pre(o), \mathit{eff}(o)\rangle$, where both the *precondition* $pre(o)$ and the *effect* $\mathit{eff}(o)$ are partial states. The set of operators is denoted with $\mathcal{O}$. An operator $o \in \mathcal{O}$ is *applicable* in a state s iff $pre(o)[v] = s[v]$ for all $v \in vars(pre(o))$, and applying an applicable operator o in s yields the successor state $s' = o[s]$ by changing the values in s of the variables in $\mathit{eff}(o)$ accordingly. The set of all applicable operators in state s is denoted with $app(s)$. Operators o have a non-negative cost $cost(o)$. If all $o \in \mathcal{O}$ have the same cost, the operators in $\mathcal{O}$ are called unit-cost operators. Sequences of operators $\sigma = o_1 \dots o_n$ that are sequentially applicable in the initial state s_0, i.e., if the state $\sigma(s_0) := o_n[\dots o_1[s_0]\dots]$ is defined, are called *paths*. The cost of σ is the sum of the costs of the operators in σ. The length of σ is the number of operators in σ, and denoted with $|\sigma|$. Our objective is to find a path to a *goal state*, i.e., a state that complies with $s_\star$. Paths that lead to goal states are called *solutions*.

Furthermore, we need the notion of *commutativity* of operators. We say that o and o' are commutative if $vars(\mathit{eff}(o)) \cap vars(pre(o')) = \emptyset$, $vars(\mathit{eff}(o')) \cap vars(pre(o)) = \emptyset$, and there exists no $v \in vars(\mathit{eff}(o)) \cap vars(\mathit{eff}(o'))$ such that $\mathit{eff}(o)[v] \neq \mathit{eff}(o')[v]$. We denote commutative operators o and o' with $o \bowtie o'$. To define sleep sets, we use the definition of Holte et al. (2015). Let $<_{ss}$ be a total order on the set of operators $\mathcal{O}$.

Definition 1. *For a path $\sigma = o_1 \dots o_n$, the* sleep set $ss(\sigma)$ *for σ is a set of operators that satisfies the following conditions: For $n = 0$, i.e., for the empty path ε, $ss(\varepsilon) := \emptyset$ (the empty set). For $n > 0$, $ss(\sigma) := \{o$ applicable in $\sigma(s_0) \mid (o_n \bowtie o)$ and $(o <_{ss} o_n$ or $o \in ss(o_1 \dots o_{n-1}))\}$.*

Sleep sets can be used as an operator pruning technique: for a state s reached by path σ, instead of applying *all* operators that are applicable in s, only apply the applicable operators in s that are *not* contained in $ss(\sigma)$.

Example 1. *Consider a planning problem with variables $\mathcal{V} = \{a, b\}$ with $dom(a) = dom(b) = \{0, 1\}$, initial state $s_0 = \{a \mapsto 0, b \mapsto 0\}$, and goal $s_\star = \{a \mapsto 1, b \mapsto 1\}$. The set of operators is given by $\mathcal{O} = \{o_1, o_2\}$, where $o_1 = \langle\{a \mapsto 0\}; \{a \mapsto 1\}\rangle$ and $o_2 = \langle\{b \mapsto 0\}; \{b \mapsto 1\}\rangle$, and $cost(o_1) = cost(o_2) = 1$. There are two solutions, namely the operator sequences $o_1 o_2$ and vice versa, $o_2 o_1$. The state space of the problem is depicted in Fig.1 on the left (where we shortly denote states $\{a \mapsto i, b \mapsto j\}$ as ij).*

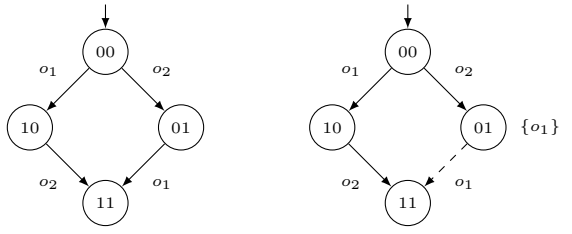

Figure 1: State spaces without and with sleep sets pruning

When sleep set pruning is applied with the operator ordering $o_1 <_{ss} o_2$, we observe that the sleep set of the path $\sigma = o_2$ is $\{o_1\}$, because o_1 is applicable in the state $s := \{a \mapsto 0, b \mapsto 1\}$, $o_1 \bowtie o_2$, and $o_1 <_{ss} o_2$. Hence sleep set pruning will not apply o_1 in s, indicated with the dashed arrow in Fig.1 on the right.

As the sleep sets pruning technique is path-dependent, it cannot be directly applied to algorithms that perform duplicate elimination, because different paths to a state s can cause different pruning decisions in s. To be able to describe algorithms that apply sleep sets in combination with duplicate elimination, we will use the notion of sleep sets defined for *several* paths $\sigma_1, \dots, \sigma_n$ that all generate the same state s. We will denote this set with $ss(\sigma_1, \dots, \sigma_n)$, which has the intended meaning to carry the information which operators to prune in s when s has been reached by $\sigma_1, \dots, \sigma_n$ in this particular order (i.e., σ_1 first, σ_n last). The formal definition of the semantics of $ss(\sigma_1, \dots, \sigma_n)$ will depend on the particular way sleep sets and duplicate elimination are integrated. We will come back to this point in the next section.

As a basis for our further investigations, we provide an analysis of several sleep set variants from the literature in combination with graph search algorithms.

A Literature Analysis on Sleep Sets

In the literature (Godefroid and Wolper 1992; Godefroid, Holzmann, and Pirottin 1993; 1995; Godefroid 1996; Holte, Alkhazraji, and Wehrle 2015), four main variants of sleep sets with duplicate elimination have been considered.

(A) Full duplicate elimination: Let s be a state first generated by path σ. If s is revisited by path σ', then s is immediately pruned as duplicate. Sleep sets are computed (only once per state) as in Def. 1, and $ss(\sigma, \sigma')$ does not need to be computed because s is pruned as duplicate in this case.

This variant has been applied in a first approach on sleep sets (Godefroid and Wolper 1992), used within depth-first search. However, as the sleep sets' pruning decisions are path-dependent, reaching a state s via different paths can cause different operators to be pruned in s when s is revisited. As a consequence, the above variant is incomplete, as shown by counter-examples by Koutny and Pietkiewicz-Koutny (1995)[1] and Bosnacki et al. (2009).

(B) States that are revisited are pruned as duplicates as in (A), but the definition of sleep sets is modified compared to Def. 1. In a nutshell, in contrast to (A), operators that close a cycle in the state space are treated in a special way for the sleep sets computation. To describe this modification in more detail, we must first provide some more technical details how sleep sets are typically *incrementally* computed. Firstly, (i) every time a successor path σo is generated from path σ, all operators from the sleep set $ss(\sigma)$ of the parent path σ that are commutative with o are propagated to $ss(\sigma o)$. Secondly, (ii) to accommodate for the ordering condition of sleep sets (i.e., for the "$o <_{ss} o_n$" part in Def. 1), after having generated the

[1] In the paper by Koutny and Pietkiewicz-Koutny (1995), a modified sleep set algorithm is considered, as discussed in (B). However, their counter-example applies to (A) as well.

successor path σo, o is (locally) included in $ss(\sigma)$ to be propagated to the further successor paths $\sigma o'$ of σ generated with operators o' with $o <_{ss} o'$.

To avoid operators to be propagated in case a cycle is closed, the second step (ii) is restricted to be performed only if σo does not yield a state that is already on the search stack (Godefroid, Holzmann, and Pirottin 1993). In other words, for a state s, assume that applying o in s closes a cycle, i.e., assume that the application of o in s yields a state that is on the search stack. Then o is excluded from the sleep sets of those successor states s' of s that are generated with operators o' with $o <_{ss} o'$.

However, the resulting algorithm is still incomplete, as shown by the counter-example by Koutny and Pietkiewicz-Koutny (1995). In an additional refinement, the first step (i) is refined such that cycle-closing operators are ruled out of sleep sets at the time a state is taken from the stack, i.e., before the recursive depth-first search call for the successors. The final algorithm (Godefroid, Holzmann, and Pirottin 1995) resolves the issue of incompleteness, by guaranteeing that every time o closes a cycle, o is not contained in the sleep set of the parent path.

(C) Let s be a state first reached by σ_1, and then revisited by paths $\sigma_2, \ldots, \sigma_n$. The sleep set $ss(\sigma_1, \ldots, \sigma_n)$ is inductively defined as $ss(\sigma_1, \ldots, \sigma_{n-1}) \cap ss(\sigma_n)$. Informally, the sleep set after exploring σ_n is updated according to $ss(\sigma_n)$ such that all operators are applied according to $ss(\sigma_n)$ that have been pruned according to $ss(\sigma_1, \ldots, \sigma_{n-1})$ in the corresponding state before. States are pruned as duplicates if $ss(\sigma_1, \ldots, \sigma_n) = \emptyset$.

Informally, the intersection of sleep sets is needed in this variant because the intersection ensures that the remaining operators in the updated sleep set can still be pruned (according to all the sleep sets computed for the state), corresponding to the pruning information obtained on all paths on which the state has been reached so far. This variant of updating sleep sets has been proposed by Godefroid (1996) – we will come back to it in the next section.

(D) Let s be a state revisited on a cycle, i.e., let s first be generated by path $\sigma = o_1 \ldots o_n$ and afterwards by path $\sigma' = o_1 \ldots o_n o_{n+1} \ldots o_{n+k}$ for $k \geq 1$. Then s is pruned as duplicate, and as in case (A), $ss(\sigma, \sigma')$ does not need to be computed. For all the remaining states reached by uncyclic paths $\sigma_1, \ldots, \sigma_n$, an entirely new sleep set is computed according to Def. 1, i.e., $ss(\sigma_1, \ldots, \sigma_n) := ss(\sigma_n)$.

This variant has been proposed for planning (Holte, Alkhazraji, and Wehrle 2015) within IDA* and cycle detection. Holte et al. provide a more general definition of sleep sets that allows for more pruning. However, they did not provide a proof that IDA* with sleep sets and cycle detection is completeness and optimality preserving. We will show that this is indeed the case.

In most of these related papers, there is no empirical experimental study of sleep sets in combination with duplicate elimination. As an exception, Holte et al. provide an experimental study, focusing on a comparison of the pruning power of sleep sets compared to their generalization.

Sleep Sets with Duplicate Elimination

We consider the combination of sleep sets with graph search algorithms, with a focus on optimal search algorithms. For our investigations, we need some more terminology. Following Holte et al. (2015), orderings $<_{\mathcal{O}}$ on the set $\mathcal{O}$ of operators induce a lexicographical ordering on the set of operators sequences: for operator sequences $\sigma = o_1 \ldots o_n$ and $\sigma' = o'_1 \ldots o'_m$, if $|\sigma| < |\sigma'|$, then $\sigma <_{\mathcal{O}} \sigma'$; if $|\sigma| = |\sigma'|$, then $\sigma <_{\mathcal{O}} \sigma'$ iff $o_i <_{\mathcal{O}} o'_i$, where i is the index such that $o_k = o'_k$ for all $1 \leq k \leq i-1$, and $o_i \neq o'_i$. For an ordering defined on $\mathcal{O}$, we will use the same symbol for the induced lexicographical ordering on operator sequences when the meaning is clear from the context.

For states s and s' such that s' is reachable from s, and for the given sleep sets ordering $<_{ss}$, let $\min(s, s')$ denote the least-cost operator sequence (among all operator sequences) that is applicable in s and leads to s' and that is minimal according to $<_{ss}$. Some of our investigations are based on the following theorem by Holte et al. (2015).

Theorem 1 (Holte et al., 2015)**.** *Let s, s' be states with s' reachable from s. Let $\min(s, s') := o_1 \ldots o_n$. Then $o_k \notin ss(o_1 \ldots o_{k-1})$ for all k with $1 \leq k \leq n$.*

In particular, the theorem states that for all states reachable from s_0, the path $\min(s_0, s)$ is preserved by sleep sets. This result in turn provides us with a sufficient criterion for complete and optimal graph search algorithm applied with sleep sets: if $\min(s_0, s)$ is preserved, then by Theorem 1 completeness and optimality are preserved as well.

For a search algorithm A that works on the operator set $\mathcal{O}$, we assume a total ordering $<_A$ on $\mathcal{O}$ in which A generates its successor states. Furthermore, if A is applied with sleep sets, we assume that $<_A$ and $<_{ss}$ are identical.

Breadth-First Search

We consider the combination of breadth-first search with variant (A) discussed in the last section: for state s and path σ that generates s for the first time, operators are pruned in s according to $ss(\sigma)$, and s is pruned when reached by other paths later on. We call this combination BFS^{ss}. The following theorem shows that BFS^{ss} inherits completeness and optimality from standard breadth-first search.

Theorem 2. *BFS^{ss} is complete. When applied with unit-cost operators, BFS^{ss} is optimal.*

The proof of Theorem 2 will rely on showing that BFS^{ss} preserves $\min(s_0, s)$ for all reachable states s when $<_A$ is equal to $<_{ss}$. For the proof, we first observe in the following lemma that prefixes of minimal paths are minimal as well.

Lemma 1. *Let s be a reachable state, and let $\min(s_0, s) = o_1 \ldots o_n$ be the minimal path from s_0 to s. Then for all $i \in \{1, \ldots, n-1\}$ and $s_i := o_i[\ldots o_1[s_0] \ldots]$: $\min(s_0, s_i) = o_1 \ldots o_i$ is the minimal path from s_0 to s_i.*

Proof. Observe that for paths σ and σ', if $\sigma <_{ss} \sigma'$, then $\sigma X <_{ss} \sigma' X$ for all operator sequences X. (*)

Consider $\sigma_i := o_1 \ldots o_i$ and $s_i := o_i[\ldots o_1[s_0] \ldots]$ for some $i \in \{1, \ldots, n-1\}$. First, we observe that $cost(\sigma_i)$ is minimal among all paths to s_i (otherwise, if there was a

cheaper path σ' to s_i, then $\sigma' o_{i+1} \dots o_n$ would be cheaper than $\min(s_0, s)$). Second, consider a path σ' to s_i with $cost(\sigma') = cost(\sigma_i)$. If $\sigma' <_{ss} \sigma_i$, then by (*) for $X = o_{i+1} \dots o_n$, $\sigma' X <_{ss} \sigma_i X = \min(s_0, s)$, which again would imply that $\min(s_0, s)$ is not the minimal path to s. □

Proof. (Theorem 2) Let s be a state reachable from s_0. Let $\min(s_0, s) = o_1 \dots o_n$, and $\sigma = o'_1 \dots o'_m$ be a path with $\sigma \neq \min(s_0, s)$ that reaches s. We show that standard breadth-first search using $<_A$ generates s with $\min(s_0, s)$ first, i.e., before it generates s with σ. To see this, consider the following cases:

1. $n < m$: $\min(s_0, s)$ is explored before σ because breadth-first search explores shorter paths before longer ones.
2. $n > m$ cannot occur because it would contradict the assumption that $\min(s_0, s)$ is minimal.
3. $n = m$: Let i be the left-most position where $\min(s_0, s)$ and σ differ, i.e., $o_i \neq o'_i$ and $o_j = o'_j$ for $j < i$. By assumption, $<_{ss}$ is equal to $<_A$, and $\min(s_0, s) <_{ss} \sigma$, hence $\min(s_0, s) <_A \sigma$ and $o_i <_A o'_i$. It follows that breadth-first search explores the path $o_1 \dots o_i$ before $o'_1 \dots o'_i$, and hence (by exploring states in a first-in first-out manner) also their completion $\min(s_0, s)$ before σ.

By Lemma 1, the prefixes of $\min(s_0, s)$ are minimal as well, hence it follows that all states $s_1, \dots, s_n$ generated on the path from s_0 to s are generated on the path $\min(s_0, s)$ first. It follows that $s_1, \dots, s_n$ are not pruned by breadth-first search as duplicate states. By Theorem 1, it follows that additionally computing sleep sets for these prefix paths that generate $s_1, \dots, s_n$, which yields BFS^{ss}, preserves $\min(s_0, s)$, showing the claim. □

As a general observation, to preserve optimality of graph search algorithms when applied with sleep sets, it is sufficient to guarantee that states on $\min(s_0, s)$ are not pruned as duplicates. The theorem shows that this is the case for breadth-first search even with full duplicate elimination according to variant (A), because the $\min(s_0, s)$ paths are generated first. In general, we do not have this property, e.g., with Dijkstra's algorithm or more generally, with A*. Therefore, sleep set updates will be needed.

A* Search

A* can be combined with sleep sets variant (C) described in the literature review section, by a reduction to the sleep set algorithm proposed by Godefroid (1996). In his monograph, Godefroid proposes this algorithm directly in combination with the *persistent sets* pruning technique. For ease of presentation, we will first discuss his algorithm and the adaptation to A* for the special case without persistent sets, which amounts to the "pure" combination of A* and sleep sets. We will come back to the combination with additional pruning techniques (based on strong stubborn sets, which is a variant of persistent sets) in the next section.

We do not give pseudo code of Godefroid's algorithm, but only provide a short description of the main points (for more details, the reader is referred to his monograph, Section 5.2). Godefroid uses a *stack* as open list, and stores expanded states (together with their associated sleep set) in a hash table as closed list. States that are generated and recognized as duplicates are handled by updating the associated sleep set: Consider a state s that has been generated by paths $\sigma_1, \dots, \sigma_n$, and is generated again by path σ. If s is contained in closed, then all operators in $ss(\sigma_1, \dots, \sigma_n) \setminus ss(\sigma)$ are additionally applied in s. In particular, states are pruned as duplicates only in case the corresponding sleep set has become empty.

We adapt Godefroid's algorithm to emulate A* combined with sleep sets, called A$^*_{ss}$ in the following. For simplicity, we assume consistent heuristics (for inconsistent heuristics, some more cases on state reopening need to be distinguished). In a nutshell, compared to Godefroid's algorithm, A$^*_{ss}$ differs in three main points: Firstly, although Godefroid's algorithm uses a stack as open list, the completeness proof that all states of the state space can still be generated (Theorem 5.4 in the monograph) does not rely on the stack behavior. Applying the algorithm with a priority queue retains the completeness property (we will formalize this claim below). Secondly, like A*, we additionally need to check in A$^*_{ss}$ for the goal condition when states are popped from open. These adaptations are trivial extensions. In addition, assume a state s that has been generated by paths $\sigma_1, \dots, \sigma_n$, and assume s is generated again by path σ. If s is contained in open and not yet in closed, then the sleep set of s in open is updated according to σ, i.e., $ss(\sigma_1, \dots, \sigma_n, \sigma) := ss(\sigma_1, \dots, \sigma_n) \cap ss(\sigma)$.

We will describe A$^*_{ss}$ in more detail in the following. To avoid confusion, we slightly extend the notation on sleep sets $ss(\sigma_1, \dots, \sigma_n)$ for states s reached on several paths $\sigma_1, \dots, \sigma_n$: To make clear which state we are talking about, we explicitly label the state s reached by path σ with σ^s. Accordingly, the sleep set of state s reached by paths $\sigma^s_1, \dots, \sigma^s_n$ in this order is denoted with $ss(\sigma^s_1, \dots, \sigma^s_n)$.

Compared to A*, A$^*_{ss}$ differs in the computation of successor states. Assuming a state s that is subject to expansion, A$^*_{ss}$ computes successor states of s as follows.

Computation of operators to be applied Instead of considering *all* applicable operators in s (like A*), the set of operators applied in s by A$^*_{ss}$ is defined as

$$app(s) \setminus ss(\sigma^s_1, \dots, \sigma^s_n),$$

where $\sigma^s_1, \dots, \sigma^s_n$ are the paths by which s has been generated at the time when s is expanded.

Operator application and sleep set updates A$^*_{ss}$ applies the operators in $app(s) \setminus ss(\sigma^s_1, \dots, \sigma^s_n)$ and computes (or updates, respectively) the corresponding sleep set of the successor states. The pseudo code of this expansion step, called EXPAND$(s, app(s), ss(\sigma^s_1, \dots, \sigma^s_n))$ in the following, is given in Fig. 2.

Assuming that σ^s is the path on which s has been reached last, EXPAND$(s, app(s), ss(\sigma^s_1, \dots, \sigma^s_n))$ computes the sleep set of the successor state s' reached on the path $\sigma^s o$ (Line 5–6). The sleep set of s' is updated according to variant (C) as described in the sleep sets literature analysis

```
1: function EXPAND(s, app(s), ss(σ^s_1, ..., σ^s_n))
2:     for o ∈ app(s) \ ss(σ^s_1, ..., σ^s_n) do
3:         s' ← o(s)
4:         σ^s ← minimal cost generating path of s
5:         X ← ss(σ^s_1, ..., σ^s_n) ∪ {o' | o' <_ss o ∧ o' ∈ app(s)}
6:         ss(σ^s o) ← {o' | o' ∈ X and o' ⋈ o}
7:         ss(σ^{s'}_1, ..., σ^{s'}_m, σ^s o) ← ss(σ^{s'}_1, ..., σ^{s'}_m) ∩ ss(σ^s o)
8:         if s' ∈ Closed then
9:             applicable_sleep ← ss(σ^{s'}_1, ..., σ^{s'}_m) \ ss(σ^s o)
10:            EXPAND (s', applicable_sleep, ∅)
11:        else
12:            n' ← make_node(s')
13:            Open ← Open ∪ {n'}
14:        end if
15:    end for
16: end function
```

Figure 2: Successor generation and sleep set updates of A^*_{ss}

section (Line 7). If s' is closed, then s' is further expanded by generating all successors that are not pruned according to the most recently computed sleep set (Line 8–10). Recall that $\sigma^{s'}_1, \ldots, \sigma^{s'}_m$ are the paths by which s' has been reached before reaching s' on $\sigma^s o$. At this point, we also observe that the particular function signature (which includes $app(s)$ and the sleep set of s) is convenient for the recursive call in Line 10. Finally, in Line 11–13, we cover the case where s' is either generated for the first time, or previously generated but not expanded yet, i.e. s' is already in open.

Theorem 3. *For admissible and consistent heuristics,* A^*_{ss} *is complete and optimal.*

Proof. The proof is a special case of the proof of Theorem 6, which shows the claim for A^*_{ss} with additional state pruning based on strong stubborn sets. □

IDA* With Cycle Detection

Sleep sets have already been applied with a limited form of duplicate elimination: Holte et al. (2015) combine IDA* with sleep sets and cycle detection as described in part (D) of the literature analysis section: States that are revisited on a cycle are pruned as duplicates, and for all other states s that are revisited, a sleep set is re-computed according to the last path that generated s. We call this algorithm IDA^*_{cyc}. Holte et al. did not provide a formal correctness proof in their paper that IDA^*_{cyc} preserves the completeness and optimality of IDA*. The following theorem shows that this is indeed the case. For an operator sequence X applicable in state s, we shortly denote the state obtained by applying X in s by $X[s]$.

Theorem 4. IDA^*_{cyc} *is complete. For admissible heuristics,* IDA^*_{cyc} *is optimal.*

The proof closely follows the structure of the proof by Holte and Burch (2014) that move pruning can safely be used with cycle detection.

Proof. Let s be a state that is reachable from s_0. We show that cycle detection does not eliminate $\min(s_0, s)$. Assume that cycle detection eliminates $\min(s_0, s)$. This means that $\min(s_0, s)$ must contain a cycle, i.e., $\min(s_0, s) = PCQ$ for operator sequences P, C, Q, with $|C| > 0$ and $PC[s_0] = P[s_0]$. This implies that PQ is a path from s_0 to s with $cost(PQ) \leq cost(\min(s_0, s))$ and $PQ <_{ss} \min(s_0, s)$, which contradicts the definition of $\min(s_0, s)$, hence showing that $\min(s_0, s)$ is not eliminated. From Theorem 1, it follows that $\min(s_0, s)$ is preserved by sleep sets as well. Together with the properties that IDA* is complete (and optimal for admissible heuristics), this shows the claim. □

IDA* With Heuristic Cutoffs

With the same reasoning of Holte and Burch (2014) that move pruning can safely be used with *heuristic cutoffs*, sleep sets can safely be applied with IDA* and heuristic cutoffs: For the cost C^* of a cheapest path to a goal state, heuristic cutoffs use a bound $B \geq C^*$, and prune all paths with strictly larger costs than B. Let IDA^*_{hc} denote IDA* with bound $B \geq C^*$ combined with sleep sets.

Theorem 5. IDA^*_{hc} *with admissible heuristics is complete and optimal.*

Proof. Let h be an admissible heuristic, and let s be a state that is reachable from s_0. We show that heuristic cutoffs do not eliminate $\min(s_0, s)$. Let operator sequence P be a prefix of $\min(s_0, s)$. As h is admissible, we have $cost(P) + h(P[s_0]) \leq cost(\min(s_0, s))$ since P is a prefix of $\min(s_0, s)$. As $cost(\min(s_0, s)) = C^*$, we have $cost(P) + h(P[s_0]) \leq C^*$. Heuristic cutoffs can only prune paths with costs strictly larger than C^*, hence P is not pruned. Since P has been chosen as an arbitrary prefix of $\min(s_0, s)$ (including $\min(s_0, s)$ itself), this shows that heuristic cutoffs do not prune $\min(s_0, s)$. □

Combining Sleep Sets With State Pruning

When applying sleep sets within a search algorithm, the set of generated states is equal to the set of generated states without sleep sets pruning. Hence, a natural question is whether sleep sets can be applied in graph search algorithms in combination with further pruning techniques that also prune states. Following Godefroid (1996), we provide an initial step of integrating sleep sets with *strong stubborn sets*, where we particularly investigate optimality of the remaining pruning technique.

Godefroid showed that sleep sets can safely be combined with *persistent sets* (1996), in the sense that the combined algorithm still preserves all deadlocks of the system. A persistent set in a state s is a subset of the applicable operators in s, where the applicable operators that are *not* included in the persistent set are pruned. *Strong stubborn sets* are a variant of persistent sets, which have originally been proposed for model checking Petri nets (Valmari 1989), and recently been applied for domain-independent planning as search (Alkhazraji et al. 2012; Wehrle et al. 2013; Wehrle and Helmert 2014). In contrast to sleep sets, strong stubborn sets take goal states into account for their pruning decision. In a nutshell, for a given state s, a strong stubborn set T_s for s contains i) a disjunctive action landmark in s, ii)

for all operators o in T_s that are applicable in s all interfering operators with o, and iii) for all operators o in T_s that are not applicable in s a set N of operators such that all plans starting in s that contain o also contain an operator $o' \in N$ before the first occurrence of o (such sets N are called *necessary enabling sets*). The set of applicable operators of T_s is a subset of the applicable operators in s. For the following investigations, it is not necessary to understand further details of strong stubborn sets and how they can be computed – the only important property that will be needed for our discussions and proofs is the following: When applying A^* with strong stubborn sets, then for every state s and for every solution π that starts in s, at least one permutation of π is preserved. In more detail, among all permutations of π's operators that lead from s to a goal state, there is at least one first operator of such a permutation that is not pruned in s.

Example 2. *Consider again the problem in Fig. 1, consisting of two independent operators o_1 and o_2, initial state* 00 *and goal state* 11*. The original state space is depicted on the left in Fig. 3. In contrast to sleep sets, strong stubborn sets can recognize that among the two solutions, only one needs to be explored, and in particular, one of the "intermediate" states* 10 *and* 01 *does not need to be generated. The resulting reduced state space is depicted in Fig. 3 on the right. Among the solutions $\pi_1 = o_1 o_2$ and $\pi_2 = o_2 o_1$, the first operator of π_1 is preserved in this example.*

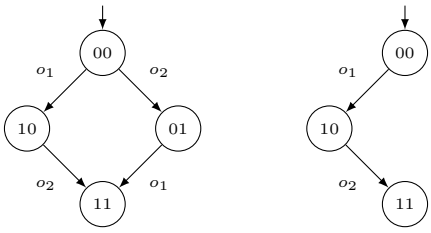

Figure 3: State spaces without and with strong stubborn sets

Along the lines of Godefroid, we show that strong stubborn sets can be combined with sleep sets within A^*_{ss} in a completeness and optimality preserving way. The resulting algorithm, called A^*_{ssssss} in the following, works exactly like A^*_{ss}, except that the set of operators applied in a state s is defined as the applicable operators from the set

$$T_s \setminus ss(\sigma^s_1, \ldots, \sigma^s_n)$$

instead of $app(s) \setminus ss(\sigma^s_1, \ldots, \sigma^s_n)$ (see Line 2 in Fig. 2), where T_s is a strong stubborn set, and $\sigma^s_1, \ldots, \sigma^s_n$ are the paths that have generated s.

The following notation and correctness proofs closely follow those given by Godefroid for deadlock detection in concurrent systems (1996). Let s_g be a goal state reachable from a state s, i.e., there is an operator sequence σ such that applying σ in s leads to s_g. The equivalence class of *permutation equivalent paths*, consisting of all paths $\bar{\sigma}$ that are permutations of the operators in σ such that $\bar{\sigma}$ still leads from s to s_g, is denoted with $[\sigma]^s$. Formally, $[\sigma]^s := \{\bar{\sigma} \mid \bar{\sigma}$ is a permutation of σ leading from s to a goal state$\}$.

For $[\sigma]^s$, we denote the set $\Sigma^{s,\sigma} := \{o^i_1 \mid \sigma \in [\sigma]^s, \sigma = o^i_1 o^i_2 \ldots o^i_{n_i}\}$ the *set of initial operators* of $[\sigma]^s$, i.e., the set that contains the first operators of all paths in $[\sigma]^s$.

Theorem 6. *Let s be a state, and let s_g be a goal state reachable from s via operator sequence σ (i.e., $\sigma(s) = s_g$). Let $\sigma_1, \ldots, \sigma_n$ be the paths explored by A^*_{ssssss} that generated s in this particular order before termination (i.e., s is generated by σ_1 first, and by σ_n last).*

If $\Sigma^{s,\sigma} \cap ss(\sigma_1, \ldots, \sigma_n) = \emptyset$, then there is a permutation $\bar{\sigma} \in [\sigma]^s$ that is preserved by A^*_{ssssss}.

Before giving the proof, let us discuss the claim and its implications in some more detail. Theorem 6 states that if the (updated) sleep set of a state s eventually does not contain any first operator of the sequences in $[\sigma]^s$, then at least one of these sequences is preserved. As discussed by Godefroid, this particularly implies the completeness of A^*_{ssssss} because the sleep set of the initial state is empty by definition. In addition, we observe that A^*_{ssssss} remains optimal because for all solutions, at least one permutation is preserved.

Proof. (Theorem 6) Consider the permutation equivalent paths $[\sigma]^s$ of σ, and the set of initial operators $\Sigma^{s,\sigma}$ of $[\sigma]^s$. We show by induction on the length of σ that at least one permutation sequence $\bar{\sigma} \in [\sigma]^s$ is preserved by A^*_{ssssss}. If $|\sigma| = 0$, the result is immediate.

If $|\sigma| > 0$, then there is an operator sequence of length $|\sigma|$ from s to s_g in the state space induced by A^* and strong stubborn sets. The proof will show that such an operator sequence to reach s_g still exists in the state space induced by A^*_{ssssss}.

First, we observe that there is $o \in \Sigma^{s,\sigma}$ that is applied by A^*_{ssssss} in s: To see this, consider the first sequence σ^s_k ($1 \leq k \leq n$) by which state s is generated such that $\Sigma^{s,\sigma} \cap ss(\sigma^s_1, \ldots, \sigma^s_k) = \emptyset$ (i.e., $\Sigma^{s,\sigma} \cap ss(\sigma^s_1, \ldots, \sigma^s_i) \neq \emptyset$ for $1 \leq i \leq k-1$). Such σ^s_k must exist because $\Sigma^{s,\sigma} \cap ss(\sigma^s_1, \ldots, \sigma^s_n) = \emptyset$ by assumption, and by definition, sleep sets can only reduce when a state is revisited.

Now consider the expansion process of s when s is reached by σ^s_k. Let o be the operator in $\Sigma^{s,\sigma}$ that is applied in s and is smallest among the remaining operators in $\Sigma^{s,\sigma}$ (according to $<_{ss}$) that have not yet been applied in s. Such an operator must exist because $\Sigma^{s,\sigma} \cap ss(\sigma^s_1, \ldots, \sigma^s_{k-1}) \neq \emptyset$. Let $s' := o[s]$. As $o \in \Sigma^{s,\sigma}$, the goal state s_g is reachable from s' with an operator sequence σ' with $|\sigma'| = |\sigma| - 1$.

Consider the paths $\rho_1, \ldots, \rho_t$ explored by A^*_{ssssss} that generate s'. To conclude the inductive proof argument, we will show (by contradiction) that $\Sigma^{s',\sigma'} \cap ss(\rho^{s'}_1, \ldots, \rho^{s'}_t) = \emptyset$. Assume $\Sigma^{s',\sigma'} \cap ss(\rho^{s'}_1, \ldots, \rho^{s'}_t) \neq \emptyset$. Then there exists an operator $\bar{o} \in \Sigma^{s',\sigma'}$ with $\bar{o} \in ss(\rho^{s'}_1, \ldots, \rho^{s'}_m)$ for all $1 \leq m \leq t$. In particular, $\bar{o} \in ss(\rho^{s'}_1, \ldots, (\sigma_k o)^{s'})$, which implies that o and $\bar{o}$ are commutative. It follows that $\bar{o}$ is applicable in s (because $\bar{o}$ is applicable in s', and $\bar{o}$ is not disabled by o), and furthermore, $\bar{o}$ is an initial operator of a permutation of σ which leads to s_g, i.e., $\bar{o} \in \Sigma^{s,\sigma}$. On the other hand, as $\bar{o} \in ss(\rho^{s'}_1, \ldots, (\sigma_k o)^{s'})$, it follows that $\bar{o} \in ss(\sigma^s_1, \ldots, \sigma^s_k)$ already (and $\bar{o}$ is propagated to $ss(\rho^{s'}_1, \ldots, (\sigma_k o)^{s'})$ afterwards), or $\bar{o}$ has been added to

$ss(\rho_1^{s'}, \ldots, (\sigma_k o)^{s'})$ after applying o in s (meaning that $\bar{o}$ is applied before o in s, i.e., $\bar{o} <_{ss} o$). However, both of these cases cannot happen: The former case is a contradiction to the fact that $\bar{o} \in \Sigma^{s,\sigma}$ (because $\Sigma^{s,\sigma} \cap ss(\sigma_1^s, \ldots, \sigma_k^s) = \emptyset$), and the latter case is a contradiction to the choice of o being the smallest operator according to $<_{ss}$. The induction continues from s' until s_g is reached. □

Corollary 1. A^*_{sssss} *inherits the completeness and optimality properties from* A^*_{ss}.

Proof. Completeness follows because the sleep set of the initial state is empty by definition. Optimality follows because for *every* solution π, a permutation of π is preserved, hence in particular for every optimal solution. □

Experimental Evaluation

We have implemented the combined approach in the Fast Downward planning system (Helmert 2006) in order to experimentally evaluate the impact of sleep sets combined with strong stubborn sets on the size of the generated state space. The experiments are performed on a cluster with Intel Xeon E5-2650v2 2.6 GHz CPUs, with a timeout of 30 minutes and a memory bound of 3 GB per run. We consider the STRIPS planning domains of the deterministic, sequential optimal track of the recent international planning competitions (IPC-08, IPC-11, and IPC-14), with an overall number of 33 domains and 461 problems.

Figure 4 shows an overview of the generated search nodes (i.e., the summed number each search node has been generated) per domain on the commonly solved problems. We compare A^* and strong stubborn sets (Alkhazraji et al. 2012), called A^*_{sss} in the following, with its sleep sets extension A^*_{sssss}, excluding the last f-layer to avoid tie-breaking issues. The pruning techniques can be applied with arbitrary heuristics – in our experiments, we have used the LM-cut heuristic (Helmert and Domshlak 2009), which is a state-of-the-art heuristic for optimal planning. The numbers of commonly solved problems are given in parenthesis after the domain names, best results are shown in bold.

As the overall picture, we observe a consistent improvement regarding the number of generated nodes per domain. Although the savings are slight, the results show that the node generations *can* be further reduced compared to pure strong stubborn set pruning. In addition to this "direct" combination, which has already been proposed by Godefroid, further research on such combinations could be beneficial to obtain further reductions.

To get a more detailed overview, we provide the number of generated nodes on a per-problem basis for the domains from the most recent competition (IPC-14) in Figure 5, again excluding the last f-layer to avoid tie-breaking issues. Domains in which the same problems are solved with the same number of node generations by A^*_{sss} and A^*_{sssss} are left out. Figure 5 lists the generated search nodes as well as the search time for all problems which at least one of A^*_{sss} and A^*_{sssss} has solved within our resource limits.

The results show that for most problems, the number of node generations for A^*_{sssss} is at most as high as for A^*_{sss}

Generated nodes	$\mathbf{A}^*_{sssss}$	$\mathbf{A}^*_{sss}$
barman-opt11-strips (4)	**22731591**	22882501
elevators-opt08-strips (22)	**38380306**	48873617
elevators-opt11-strips (18)	**36666539**	46248291
floortile-opt11-strips (7)	**29401436**	34912718
floortile-opt14-strips (6)	**52200140**	61804871
ged-opt14-strips (15)	**9064612**	**9064612**
hiking-opt14-strips (9)	**30638123**	30742124
nomystery-opt11-strips (14)	**387045**	410776
openstacks-opt08-strips (20)	**5777074**	6129805
openstacks-opt11-strips (15)	**5763904**	6116635
openstacks-opt14-strips (3)	**3866657**	4138032
parcprinter-08-strips (30)	**4877**	**4877**
parcprinter-opt11-strips (20)	**1884**	**1884**
parking-opt11-strips (2)	**555418**	560427
parking-opt14-strips (3)	**2253957**	2274968
pegsol-08-strips (28)	**54045223**	**54045223**
pegsol-opt11-strips (18)	**54408002**	**54408002**
scanalyzer-08-strips (12)	**13504754**	13942542
scanalyzer-opt11-strips (9)	**13504746**	13942534
sokoban-opt08-strips (29)	**38525983**	**38525983**
sokoban-opt11-strips (20)	**8310909**	**8310909**
tetris-opt14-strips (5)	**1150721**	1280023
tidybot-opt11-strips (14)	**299325**	308515
tidybot-opt14-strips (8)	**269184**	269891
transport-opt08-strips (11)	**302942**	426271
transport-opt11-strips (6)	**300508**	423268
transport-opt14-strips (6)	**2936311**	3396159
visitall-opt11-strips (11)	**23775034**	**23775034**
visitall-opt14-strips (5)	**2530507**	**2530507**
woodworking-opt08-strips (27)	**1722277**	2583855
woodworking-opt11-strips (19)	**976547**	1431694
Sum (427)	**454256536**	493766548

Figure 4: Sum of generated search nodes per domain on commonly solved problems (excluding the last f-layer)

(with the exception of three problems in the Hiking domain, see below for a discussion on this). Again, we observe that the node savings are mostly slight, yet some domains show that more additional pruning can be gained – for example, in the Transport domain, the number of generated nodes are roughly cut in half in the largest commonly solved problem ($\#14$). We remark that inconsistent heuristics (like LM-cut) can cause A^*_{sssss} to generate more nodes than A^*_{sss}. This presumably happens in the three Hiking problems where slightly more nodes are generated when sleep sets are applied in addition to stubborn set pruning.

We finally also remark that the savings in node generations do not pay off in terms of *coverage* (i.e., number of solved problems) for the considered "direct" combination of sleep sets and strong stubborn sets: The coverage of A^*_{sss} and A^*_{sssss} with LM-cut is equal in all of the 33 domains.

Conclusions

The paper sheds light on sleep sets combined with duplicate elimination. We have provided a literature analysis of sleep sets with duplicate elimination from computer aided verification and from artificial intelligence. Based on this analy-

Problem	Generated nodes A^*_{sssss}	Generated nodes A^*_{sss}	Search time A^*_{sssss}	Search time A^*_{sss}
floortile-opt14-strips				
p01-4-3-2	**1242815**	1510800	**32.11**	35.13
p01-4-4-2	**29688755**	35050715	**1090.50**	1186.70
p01-5-3-2	**4724922**	5612842	**153.86**	170.11
p02-5-3-2	**7206855**	8537779	**226.70**	245.06
p03-4-3-2	**1784070**	2141977	**44.69**	48.49
p03-5-3-2	**7552723**	8950758	**238.01**	257.55
hiking-opt14-strips				
ptesting-1-2-3	**1901**	**1901**	**0.01**	**0.01**
ptesting-1-2-4	**15294**	**15294**	**0.18**	**0.18**
ptesting-1-2-5	69869	**69749**	1.26	**1.23**
ptesting-1-2-7	634939	**634379**	21.81	**21.50**
ptesting-1-2-8	1496276	**1495364**	67.06	**66.22**
ptesting-2-2-3	**97479**	98151	2.99	**2.72**
ptesting-2-2-4	**4702120**	4710935	273.92	**265.36**
ptesting-2-3-4	**22911601**	22969357	1590.73	**1490.90**
ptesting-2-4-3	**708644**	746994	33.43	**28.86**
openstacks-opt14-strips				
p20_1	**1710721**	1764993	816.36	**803.53**
p20_2	**1939861**	2156773	608.38	**596.98**
p20_3	**216075**	216266	20.34	**19.83**
parking-opt14-strips				
p_12_7-01	**648718**	653952	439.45	**430.86**
p_12_7-02	**1305778**	1318576	868.17	**853.79**
p_12_7-03	**299461**	302440	216.37	**212.79**
tetris-opt14-strips				
p01-8	**599145**	645908	1530.18	**1135.26**
p02-4	**71**	140	0.09	**0.08**
p02-6	**496359**	568559	1208.25	**893.27**
p03-4	**6442**	7635	2.63	**2.18**
p05-6	**48704**	57781	36.93	**33.32**
tidybot-opt14-strips				
p02	**27104**	27128	540.48	**534.46**
p03	**31957**	32420	**532.31**	534.35
p04	**3202**	3203	53.44	**52.65**
p08	**9748**	9822	**152.59**	153.74
p11	**10775**	10817	104.06	**103.98**
p12	**153555**	153583	1545.73	**1526.25**
p13	**29865**	29938	**283.56**	283.98
p14	**2978**	2980	**50.51**	50.58
transport-opt14-strips				
p01	**1916**	3073	**0.14**	0.17
p02	**210118**	227350	**23.25**	24.41
p03	**215554**	268080	**67.49**	89.86
p07	**2310840**	2513397	236.03	**233.54**
p13	**10979**	19434	**4.51**	5.53
p14	**186904**	364825	**164.77**	230.93

Figure 5: Node generations (excluding the last f-layer) and search time on a per-problem basis for IPC-14 domains

sis, we have provided approaches to combine sleep sets with common optimal best-first graph search algorithms and with strong stubborn sets. For the future, the paper motivates the further investigation of sleep sets combined with state reduction techniques. As a proof of concept, the "direct" combination of sleep sets with strong stubborn sets has shown that it is possible to further reduce the number of node generations compared to strong stubborn sets. It will be interesting to investigate if sleep sets can be integrated more tightly with state pruning techniques, and if further (and stronger) synergy effects can be achieved.

Acknowledgments

We thank the anonymous reviewers for their comments, which helped improve the paper.

This work was partially supported by the Swiss National Science Foundation as part of the project "Automated Reformulation and Pruning in Factored State Spaces (ARAP)".

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.

Bosnacki, D.; Elkind, E.; Genest, B.; and Peled, D. 2009. On commutativity based edge lean search. *Annals of Mathematics and Artificial Intelligence* 56(2):187–210.

Burch, N., and Holte, R. C. 2012. Automatic move pruning revisited. In *Proc. SoCS 2012*.

Godefroid, P., and Wolper, P. 1992. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proc. CAV 1991*, 332–342. Springer-Verlag.

Godefroid, P.; Holzmann, G. J.; and Pirottin, D. 1993. State-space caching revisited. In *Proc. CAV 1992*, 178–191. Springer-Verlag.

Godefroid, P.; Holzmann, G.; and Pirottin, D. 1995. State-space caching revisited. *Form. Methods Syst. Des.* 7(3):227–241.

Godefroid, P. 1996. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag.

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. *Journal of Artificial Intelligence Research* 26:191–246.

Holte, R. C.; Alkhazraji, Y.; and Wehrle, M. 2015. A generalization of sleep sets based on operator sequence redundancy. In *Proc. AAAI 2015*, 3291–3297.

Holte, R. C., and Burch, N. 2014. Automatic move pruning for single-agent search. *AI Communications* 27(4):363–383.

Holte, R. C. 2013. Move pruning and duplicate detection. In *Proceedings of the 26th Canadian Conference on Artificial Intelligence*, volume 7884 of *Lecture Notes in Computer Science*, 40–51. Springer-Verlag.

Koutny, M., and Pietkiewicz-Koutny, M. 1995. On the sleep sets method for partial order verification of concurrent systems. Technical Report 495, Department of Computing Science, University of Newcastle upon Tyne.

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., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proc. ICAPS 2014*, 323–331.

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.