Automatic Planning
14. Partial-Order Reduction
Which Should I Do First, the Right Shoe or the Left Shoe?

Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2017/2018

Introduction
Act Prune
S3 Ingredients
S3 Theory
S3 Practice
STRIPS
Conclusion
References

1. Introduction
2. Action-Pruning Functions
3. Strong Stubborn Sets: Ingredients
4. Strong Stubborn Sets: Theory
5. Strong Stubborn Sets: Practice
6. What about STRIPS?
7. Conclusion

The Pitfalls of Optimal Heuristic Search

Jörg Hoffmann

The Gripper benchmark:
 Carry \( n \) balls from \( L \) to \( R \)

Proposition. Let \( \Pi_n \) be the Gripper task with \( n \) balls. Then \( N^1(\Pi_n) \) grows exponentially in \( n \).

Proof sketch.

→ In other words: What’s killing us here are plan permutations.
Pruning Methods

To the rescue: Optimality-preserving pruning methods.

- **State Pruning**: Reduces search effort by cross-state comparisons. Prunes states whose exploration can be shown to be unnecessary.
- **Action Pruning**: Reduces search effort by analyzing applicable actions. Prunes actions whose exploration can be shown to be unnecessary.

We cover one such pruning method: **Partial-order reduction** via action pruning.

Partial-Order Reduction (POR) Methods: Overview

- Partial-order reduction (POR) methods identify, and prune, permutable parts of the search space.

- They do so via action pruning (cf. slide 6).

There are different kinds of POR methods:

- **Transition-reduction methods**: Prune applicable actions while preserving the reachable state space.
  - Sleep Sets, Ample Sets, etc. Not considered here (useful mainly in depth-first search algorithms).
- **State-reduction methods**: Prune applicable actions while preserving at least one optimal solution.
  - Strong Stubborn Sets (S3).

Our Agenda for This Chapter

- **Action-Pruning Functions**: We define and briefly analyze what an action-pruning function is, and when such pruning is safe.
- **Strong Stubborn Sets: Ingredients**: The strong stubborn sets technique (and POR more generally) relies on a number of basic concepts, that we introduce here.
- **Strong Stubborn Sets: Theory**: We define what a strong stubborn set is, and we prove safety as an action-pruning function.
- **Strong Stubborn Sets: Practice**: We consider how to operationalize the definition.
- **What about STRIPS?**: In the above, our definitions are agnostic to STRIPS/FDR where it doesn’t matter; where it does matter, we use FDR. Here we explain that very little changes for STRIPS.
Jörg Hoffmann  Automatic Planning  Chapter 14: Partial-Order Reduction  10/40

Safe Action-Pruning Functions

Definition (Safe $\rho$). Let $\Pi$ be a planning task with state space $\Theta = (S, L, c, T, I, S^G)$, and let $\rho$ be an action-pruning function for $\Pi$. We say that $\rho$ is safe if, for all $s \in S$, the cost of an optimal solution for $s$ in $\Theta_\rho$ equals $h^*(s)$.

→ A safe action-pruning function $\rho$ preserves optimality.

Proposition. Let $\Pi$ be a planning task with states $S$, and let $\rho$ be an action-pruning function for $\Pi$. If, for every solvable non-goal $s \in S$, $\rho(s)$ contains at least one action starting a shortest optimal plan for $s$, then $\rho$ is safe.

Proof. By induction on the length $n$ of a shortest optimal plan for $s$. Base case $n = 1$: Direct from definition. Inductive case $n \rightarrow n + 1$: The first action $a$ of a shortest optimal plan for $s$ is preserved. Say the transition is $s \xrightarrow{a} s'$. Then the shortest optimal plan for $s'$ is shorter than that for $s$, so the claim follows by induction hypothesis.

→ Why “shortest”? We may bother you with an exercise.

→ What about unsolvable $s$?

Jörg Hoffmann  Automatic Planning  Chapter 14: Partial-Order Reduction  12/40

Before We Begin . . .

Ingredients? Action dependencies.

→ How actions affect each other’s applicability and/or outcome state.

→ We define this semantically here. For practice, we will later define syntactic characterizations.

Illustrative example: “1/2-Log”

→ $V$: $\{\text{truck}_1, \text{truck}_2 : \{\text{A}, \text{B}, \text{C}, \text{D}\}; \text{pack}_1, \text{pack}_2 : \{\text{A}, \text{B}, \text{C}, \text{D}, \text{T}_1, \text{T}_2\}\}$

→ $I$: $\{\text{truck}_1, \text{truck}_2, \text{pack}_1, \text{pack}_2 = \text{A}; \text{G}; \text{pack}_1, \text{pack}_2 = \text{D}\}$

→ $A$: $\text{drive}(i, x, y)$ (for $x \neq y$ neighbors); $\text{pre} \text{truck}_i = x$, effect $\text{truck}_i = y$

→ $\text{load}(i, x)$: $\text{pre} \text{pack}_i = x \text{, truck}_i = x$, effect $\text{pack}_i = \text{T}_i$

→ $\text{unload}(i, x)$: $\text{pre} \text{pack}_i = \text{T}_i \text{, truck}_i = x$, effect $\text{pack}_i = x$

→ Note: Package $i$ load/unload only with truck $i$.

→ “1/2-Tele-Log”: $\text{teleport}(i, y)$: $\text{pre empty}$, effect $\text{truck}_i = y$

Jörg Hoffmann  Automatic Planning  Chapter 14: Partial-Order Reduction  15/40
**Introduction**

**Necessary Enabling Sets**

**Definition (Necessary Enabling Set).** Let $\Pi$ be a planning task with actions $A$, goal $G$, and states $S$. Given $a \in A$ and $s \in S$ where $a \notin A(s)$ (i.e., $\text{pre}_a \not\subseteq s$), a **necessary enabling set** for $a$ in $s$ is a set $A_{s \rightarrow a} \subseteq A$ of actions so that for every action sequence $\langle a_1, \ldots, a_n \rangle$ applicable in $s$, if $a_i = a$ then $\{a_1, \ldots, a_{i-1}\} \cap A_{s \rightarrow a} \neq \emptyset$.

Given $s \in S$ where $G \not\subseteq s$, a **necessary enabling set for $G$ in $s$** is a set $A_{s \rightarrow G} \subseteq A$ of actions so that for every action sequence $\langle a_1, \ldots, a_n \rangle$ applicable in $s$ that achieves $G$, $\{a_1, \ldots, a_n\} \cap A_{s \rightarrow G} \neq \emptyset$.

A necessary enabling set is a set of actions at least one of which must be applied to enable an action $a$ or the goal $G$.

**Example:** “1/2-Tele-Log”

- Example $s, a, A_{s \rightarrow a}$: $I; \{\text{drive}(1, B, C); \{\text{drive}(1, A, B), \text{teleport}(1, B)\}\}.$
- Example $s, A_{s \rightarrow G}$: $I; \{\text{unload}(1, D)\}$ or $\{\text{load}(1, A)\}$ or $\{\text{drive}(1, C, D), \text{teleport}(1, D)\}$.

---

**Conclusion**

**References**

---

**Act Prune**

**S3 Ingredients**

**S3 Theory**

**STRIPS**

**S3 Practice**

---

**Act Prune**

**S3 Ingredients**

**S3 Theory**

**STRIPS**

**S3 Practice**

---

**Act Prune**

**S3 Ingredients**

**S3 Theory**

**STRIPS**

**S3 Practice**

---

**Act Prune**

**S3 Ingredients**

**S3 Theory**

**STRIPS**

**S3 Practice**

---
**Strong Stubborn Sets: Intuition**

**Example:** “1/2-Log Small”

- \( V \): truck\(_1\), truck\(_2\), pack\(_1\), pack\(_2\).
- \( I \): As shown.
- \( G \): pack\(_1\) = B, pack\(_2\) = B.
- \( A \): drive\((i, x, y)\), load\((i, x)\), unload\((i, x)\).

**Definition (Strong Stubborn Sets).** Let \( \Pi \) be a planning task with actions \( A \), goal \( G \), and states \( S \). Let \( s \in S \) be a non-goal state. A strong stubborn set for \( s \) is a set \( A_{S3} \subseteq A \) of actions such that:

1. \( A_{S3} \) contains a necessary enabling set for \( G \) in \( s \);
2. For every \( a \in A_{S3} \setminus A[s] \), \( A_{S3} \) contains a necessary enabling set for \( a \) in \( s \);
3. For every \( a \in A_{S3} \cap A[s] \), \( A_{S3} \) contains all \( a' \in A \) that interfere with \( a \).

**Definition (S3 Pruning).** Let \( \Pi \) be a planning task with states \( S \). An action-pruning function \( \rho_{S3} \) for \( \Pi \) is called an S3 pruning function if, for every non-goal state \( s \in S \), there exists a strong stubborn set \( A_{S3} \) for \( s \) so that \( \rho_{S3}(s) = A(s) \cap A_{S3} \).

**Theorem (S3 Pruning Safety).** Let \( \Pi \) be a planning task, and let \( \rho_{S3} \) be an S3 pruning function. Then \( \rho_{S3} \) is safe.
Strong Stubborn Sets are Safe, ctd.

Questionnaire

Reminder: An S3 for $s$ is a set $A_{S3} \subseteq A$ of actions such that:

(i) $A_{S3}$ contains a necessary enabling set for $G$ in $s$;
(ii) For every $a \in A_{S3} \setminus A[s]$, $A_{S3}$ contains a necessary enabling set for $a$ in $s$; and
(iii) For every $a \in A_{S3} \cap A[s]$, $A_{S3}$ contains all $a' \in A$ that interfere with $a$.

Question!
Do strong stubborn sets have anything to do with commutative actions?

(A): Yes  (B): No

The S3 Definition as an Algorithm (compare slide 22)

input: Planning task $\Pi$, state $s$.
output: Strong stubborn set $S3$ for $s$.

(i) $S3 := A_{s} \rightarrow G$ /* a necessary enabling set for $G$ in $s$ */
Done := $\emptyset$ /* actions already processed */
while $S3 \not\subseteq$ Done do
  select $a \in S3 \setminus$ Done
  if $a \not\in A[s]$ then
    (ii) $S3 := S3 \cup A_{s} \rightarrow a$ /* a necessary enabling set for $a$ in $s$ */
  else
    (iii) $S3 := S3 \cup \{a' \mid a \text{ and } a' \text{ interfere}\}$
Done := Done $\cup\{a\}$
return $S3$

How to operationalize this?

- How to find the interfering actions?
- How to find the necessary enabling sets?

→ Syntactic approximation/characterization of these semantic definitions.
Interference: Syntactic Characterization, Part 1

Terminology: In an FDR task, say that partial assignments $p$ and $q$ agree if $p(v) = q(v)$ for all $v \in V[p] \cap V[q]$, and say that $p$ and $q$ disagree otherwise.

Reminder: $a_1$ disables $a_2$ in $s$ if both are applicable in $s$ but $a_2$ is no longer applicable after applying $a_1$.

Proposition. Let $\Pi = (V, A, c, I, G)$ be an FDR planning task with states $S$. Let $a_1, a_2 \in A$. Then there exists $s \in S$ s.t. $a_1$ disables $a_2$ in $s$ if and only if (i) $\text{pre}_{a_1}$ and $\text{pre}_{a_2}$ agree, and (ii) $\text{eff}_{a_1}$ and $\text{pre}_{a_2}$ disagree.

Done, because (reminder): $a_1$ and $a_2$ interfere if there exists $s \in S$ such that $a_1$ and $a_2$ either conflict in $s$, or one disables the other in $s$.

Interference: Syntactic Characterization, Part 2

Reminder: $a_1$ and $a_2$ conflict in $s$ iff they can be applied in both possible orders, but the outcome state differs depending on the order.

Proposition. Let $\Pi = (V, A, c, I, G)$ be an FDR planning task with states $S$. Let $a_1, a_2 \in A$. Then there exists $s \in S$ s.t. $a_1$ and $a_2$ conflict in $s$ if and only if (i) $\text{pre}_{a_1}$ and $\text{pre}_{a_2}$ agree, (ii) $\text{eff}_{a_1}$ and $\text{pre}_{a_2}$ agree, (iii) $\text{eff}_{a_2}$ and $\text{pre}_{a_1}$ agree, and (iv) $\text{eff}_{a_1}$ and $\text{eff}_{a_2}$ disagree.

Necessary Enabling Sets: Choosing an Open Subgoal

Reminder: $p \in \text{pre}_{a} \setminus s$ or $p \in G \setminus s$; $A' := \{ a' | p \in \text{eff}_{a'} \}$

$\rightarrow$ But which $p$ should we select?

Answer given by [Wehrle and Helmert (2014)]:

- Across the computation of $S3$ for different states, it is preferrable to select the same facts $p$ as much as possible.
- Static strategy: Fix an ordering over the FDR state variables (or, in STRIPS, over the facts), and always select the first $p$ in this order.
- Dynamic strategy: Where the choice depends on $s$ and the actions that have already been included into $S3$. For example, select $p$ minimizing the number of new actions added to $S3$.

BTW: Necessary enabling set = “disjunctive action landmark”

- A key concept we will introduce for landmark heuristics in Chapter 15.
- There, we will also see more advanced methods for finding such landmarks.
Summary

- Exponential blow-ups may occur in optimal search even with almost perfect heuristic functions $h$.
- Optimality-preserving pruning methods reduce search by means orthogonal to $h$, through state pruning or action pruning.
- Partial-order reduction (POR) is a family of action pruning methods targeting permutable parts of the search space, arising from commutative actions.
- Commutative actions occur frequently in planning: actions which neither interfere nor enable each other, and that can hence be applied in any order giving the same result.
- Strong stubborn sets (S3) is a POR technique that can reduce the reachable state space, avoiding the generation of states that would otherwise be reachable.
- A strong stubborn set $S3$ for a state $s$ contains a necessary enabling set for $G$, necessary enabling sets for $pre_a$ where $a \in S3 \setminus A[s]$, and interfering actions for $a \in S3 \cap A[s]$.

Reading

- About Partial Order Reduction in Planning and Computer Aided Verification [Wehrle and Helmert (2012)].

Available at:

Content: Introduces, to planning, two partial-order reduction methods originally defined for model-checking: stubborn sets and sleep sets. Discusses their relation with other pruning methods previously proposed in planning.
- **Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies** [Wehrle and Helmert (2014)].

Available at:

Content: More general definition of the strong stubborn sets technique, and empirical comparison of different strategies to find strong stubborn sets.