AI Planning

18. Partial-Order Reduction

Which Should I Do First, the Right Shoe or the Left Shoe?

Álvaro Torralba, Cosmina Croitoru

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

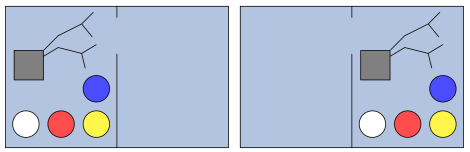
Agenda

The Pitfalls of Optimal Heuristic Search

The Pitfalls of Optimal Heuristic Search: Example Gripper

The Gripper benchmark:

Carry n balls from L to R



Proposition. *Let Π_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 3 different pruning methods:

1. Partial-order reduction: Action Pruning. → **This Chapter**
2. Dominance pruning: State Pruning. → **Next Chapter**
3. Symmetry reduction: State Pruning. → **Chapter 20**

The Right Shoe or the Left Shoe?

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). →**This Chapter**

Our Agenda for This Chapter

2. **Action-Pruning Functions:** We define and briefly analyze what an action-pruning function is, and when such pruning is safe.
3. **Strong Stubborn Sets: Ingredients:** The strong stubborn sets technique (and POR more generally) relies on a number of basic concepts, that we introduce here.
4. **Strong Stubborn Sets: Theory:** We define what a strong stubborn set is, and we prove safety as an action-pruning function.
5. **Strong Stubborn Sets: Practice:** We consider how to operationalize the definition.
6. **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.

Questionnaire

Action-Pruning Functions

Reminder: Given a task Π with actions A, and a state s, $A[s] := \{a \mid a \in A, pre_a \subseteq s\}$ denotes the actions applicable in s.

Definition (Action-Pruning Function). *Let Π be a planning task with states S. An action-pruning function for Π is a function $\rho : S \mapsto 2^A$ such that, for all $s \in S$, we have $\rho(s) \subseteq A[s]$.*

→ An action-pruning function ρ returns, for each state s, a subset $\rho(s)$ of the applicable actions.

Definition (Pruned State Space). *Let Π be a planning task, and let $\Theta = (S, L, c, T, I, S^G)$ be the state space of Π. Let ρ be an action-pruning function for Π. Then the pruned state space, denoted Θ_ρ, is defined like Θ but reducing T to those $s \xrightarrow{a} s'$ where $a \in \rho(s)$.*

→ The actions outside $\rho(s)$ are pruned.

Safe Action-Pruning Functions

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

→ A safe action-pruning function ρ preserves optimality.

Proposition. *Let Π be a planning task with states S, and let ρ be an action-pruning function for Π. If, for every solvable non-goal $s \in S$, $\rho(s)$ contains at least one action starting a shortest optimal plan for s, then ρ 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 $n \to n + 1$: 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.

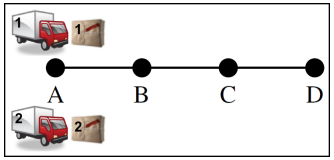
→ What about unsolvable s?

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: $truck_1, truck_2 : \{A, B, C, D\}$; $pack_1, pack_2 : \{A, B, C, D, T_1, T_2\}$.
- I: $truck_1, truck_2, pack_1, pack_2 = A$. G: $pack_1, pack_2 = D$.
- A: $drive(i, x, y)$ *(for $x \neq y$ neighbors)*: pre $truck_i = x$, effect $truck_i = y$
  $load(i, x)$: pre $pack_i = x, truck_i = x$, effect $pack_i = T_i$
  $unload(i, x)$: pre $pack_i = T_i, truck_i = x$, effect $pack_i = x$
  → **Note:** Package i load/unload only with truck i!
- "1/2-Tele-Log": $teleport(i, y)$: pre empty, effect $truck_i = y$

Action Pairs: Enabling, Disabling, Conflict

Definition (Action Pair Dependencies). *Let Π be a planning task with actions A and states S. Let $a_1 \neq a_2 \in A$ and $s \in S$. We say that:*

(i) a_1 *enables* a_2 *in* s *if* $a_1 \in A[s]$ *and* $a_2 \notin A[s]$*; but* $a_2 \in A[s[\![a_1]\!]]$.

(ii) a_1 *disables* a_2 *in* s *if* $a_1 \in A[s]$ *and* $a_2 \in A[s]$*; but* $a_2 \notin A[s[\![a_1]\!]]$.

(iii) a_1 *and* a_2 *conflict in* s *if* $a_1 \in A[s]$, $a_2 \in A[s]$, $a_1 \in A[s[\![a_2]\!]]$, *and* $a_2 \in A[s[\![a_1]\!]]$*; but* $s[\![\langle a_1, a_2\rangle]\!] \neq s[\![\langle a_2, a_1\rangle]\!]$.

Example: "1/2-Tele-Log"

(i) Example a_1, a_2, s?

(ii) Example a_1, a_2, s?

(iii) Example a_1, a_2, s?

Note:

Action Pairs: Interfering, Commutative

Definition (Action Pair Dependencies, ctd.). *Let Π be a planning task with actions A and states S. Let $a_1 \neq a_2 \in A$. We say that:*

(iv) 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.

(v) a_1 *and* a_2 *are commutative if they do not interfere, and neither enables the other in any* $s \in S$.

Example: "1/2-Tele-Log"

(iv) Example a_1, a_2: $drive(1, A, B)$ interferes with $load(1, A)$ and $teleport(1, B)$ interferes with $teleport(1, C)$, see previous slide.

(v) Example a_1, a_2?

Necessary Enabling Sets

Definition (Necessary Enabling Set). *Let Π 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., $pre_a \not\subseteq s$), a necessary enabling set for a in s is a set $A_{s \to^* a} \subseteq A$ of actions so that for every action sequence $\langle a_1, \ldots, a_n\rangle$ applicable in s, if $a_i = a$ then $\{a_1, \ldots, a_{i-1}\} \cap A_{s \to^* a} \neq \emptyset$.

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

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

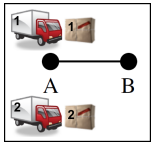
Example: "1/2-Tele-Log"

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

Questionnaire

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: $drive(i, x, y)$, $load(i, x)$, $unload(i, x)$.

Strong Stubborn Sets (S3)

Definition (Strong Stubborn Sets). *Let Π 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; and*
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 Π be a planning task with states S. An action-pruning function ρ_{S3} for Π 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 $\rho_{S3}(s) = A(s) \cap A_{S3}$.*

Strong Stubborn Sets are Safe

Reminder: ρ is safe if, for all states s, pruning using ρ preserves $h^*(s)$.

Theorem (S3 Pruning Safety). *Let Π be a planning task, and let ρ_{S3} be an S3 pruning function. Then ρ_{S3} is safe.*

Strong Stubborn Sets are Safe, ctd.

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 Π, state s.
output: Strong stubborn set $S3$ for s.

(i) $S3 := A_{s \to^* G}$ /* 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 $a \notin A[s]$ **then**
    (ii) $S3 := S3 \cup A_{s \to^* a}$ /* a necessary enabling set for a in s */
  else
    (iii) $S3 := S3 \cup \{a' \mid a$ and a' interfere$\}$
  $Done := Done \cup \{a\}$
return $S3$

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

The S3 Definition as an Algorithm (compare slide 22)

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

(i) $S3 := A_{s \to^* G}$ /* 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 $a \notin A[s]$ **then**
    (ii) $S3 := S3 \cup A_{s \to^* a}$ /* a necessary enabling set for a in s */
  else
    (iii) $S3 := S3 \cup \{a' \mid a$ and a' interfere$\}$
  $Done := Done \cup \{a\}$
return $S3$

How to operationalize this?

1. How to find the interfering actions?
2. How to find the necessary enabling sets?

→ Syntactic approximation/characterization of these semantic definitions.

Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 18: Partial-Order Reduction 28/40

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) pre_{a_1} and pre_{a_2} agree, and (ii) eff_{a_1} and pre_{a_2} disagree.*

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) pre_{a_1} and pre_{a_2} agree, (ii) eff_{a_1} and pre_{a_2} agree, (iii) eff_{a_2} and pre_{a_1} agree, and (iv) eff_{a_1} and eff_{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.

Necessary Enabling Sets: Syntactic Characterization?

Necessary Enabling Sets: Choosing an Open Subgoal

Reminder: $p \in pre_a \setminus s$ or $p \in G \setminus s$; $A' := \{a' \mid p \in eff_{a'}\}$

$\rightarrow$ 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.

Necessary Enabling Sets: The Choice Makes a Difference!

Strong Stubborn Sets in STRIPS

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:

  `http://ai.cs.unibas.ch/papers/wehrle-helmert-icaps2012.pdf`

  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.

Reading, ctd.

- *Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies* [Wehrle and Helmert (2014)].

  Available at:

  `http://ai.cs.unibas.ch/papers/wehrle-helmert-icaps2014.pdf`

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

References I

Martin Wehrle and Malte Helmert. About partial order reduction in planning and computer aided verification. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

Martin Wehrle and Malte Helmert. Efficient stubborn sets: Generalized algorithms and selection strategies. In Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml, editors, *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press, 2014.