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

Optimal heuristic search: (in particular, optimal planning)

- Admissible h avoids the expansion of nodes where $g(n) + h(n) > h^*(I)$.
- Can be highly effective in practice.

$\rightarrow$ However: Sometimes even "almost perfect" heuristics are not good enough!

Definition (Almost Perfect Heuristic). *Let Π be a planning task with states S, and let h be an admissible heuristic for Π. We say that h is almost perfect if there exists $c \in \mathbb{R}_0^+$ such that, for all $s \in S$, $h^*(s) - h(s) \leq c$.*

$\rightarrow$ An almost-perfect h has at most constant error c.

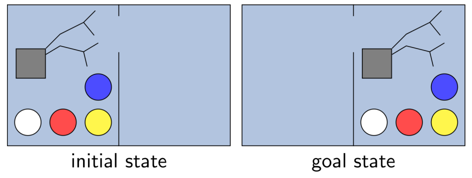
What about the search effort given such h?

- Define $N^c(\Pi) :=$ number of nodes where $g(n) + (h^*(n) - c) < h^*(I)$. Then A* using almost-perfect h will expand at least $N^c(\Pi)$ nodes.
- [Gaschnig (1977)]: If the state space is a tree, and there is only one goal state, then $N^c(\Pi)$ is linear in the length of the solution.
- [Helmert and Röger (2008)]: Even in the simplest standard planning benchmark domains, $N^c(\Pi)$ grows exponentially even for $c = 1$!

The Pitfalls of Optimal Heuristic Search: Example Gripper

The Gripper benchmark:

Carry n balls from L to R



Empirical results:

n	$h^*(I)$	$N^1(\Pi)$
04	11	125
06	17	925
08	23	5885
10	29	34301
12	35	188413
14	41	991229
16	47	5046269

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

Proof sketch. Consider the nodes n where $g(n) + h^*(n) = h^*(I)$, i.e., the nodes on an optimal plan. Obviously, every such n satisfies $g(n) + (h^*(n) - 1) < h^*(I)$, so counts towards $N^1(\Pi_n)$. In Gripper, every state with half of the balls at L and the other half at R lies on an optimal plan.

$\rightarrow$ 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?

Example: Say the task is to put on our **so**cks and **sh**oes.

⟨sockL, shoeL, sockR, shoeR⟩
⟨sockL, sockR, shoeL, shoeR⟩
⟨sockL, sockR, shoeR, shoeL⟩
⟨sockR, shoeR, sockL, shoeL⟩
⟨sockR, sockL, shoeL, shoeR⟩
⟨sockR, sockL, shoeR, shoeL⟩

$$\begin{tikzcd}
& & \text{sh}\,\bot \arrow[dr, "soR"] & & \\
& \text{so}\,\bot \arrow[ur, "shL"] \arrow[dr, "soR"] & & \text{sh so} \arrow[dr, "shR"] & \\
\bot\,\bot \arrow[ur, "soL"] \arrow[dr, "soR"'] & & \text{so so} \arrow[ur, "shL"] \arrow[dr, "shR"] & & \text{sh sh} \\
& \bot\,\text{so} \arrow[ur, "soL"] \arrow[dr, "shR"'] & & \text{so sh} \arrow[ur, "shL"'] & \\
& & \bot\,\text{sh} \arrow[ur, "soL"'] & &
\end{tikzcd}$$

Commutative actions:

- Actions that can be applied in any order, leading to the same result.
- E.g. here: sockL vs. sockR, shoeL vs. sockR, . . .
- E.g. Gripper: pickup(ball1,L) vs. pickup(ball2,L), . . .
- If an optimal plan π contains such actions, then any π' permuting these actions also is a plan. → Lots of states on optimal plans (cf. slide 5)!

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

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

- Variables V: $H_1, H_2, H_3 : \{Fine, Broken\}$; $P_1, P_2, P_3 : \{Home, Captured\}, W : \{Hungry, Happy\}\}$.
- Initial state I: $H_1, H_2, H_3 = Fine$, $P_1, P_2, P_3 = Home$, $W = Hungry$.
- Goal G: $W = Happy$.
- Actions A: $Blow(x)$: pre $H_x = Fine$, eff $H_x = Broken$
  $Capture(x)$: pre $P_x = Home$, $H_x = Broken$, eff $P_x = Captured$
  $Banquet$: pre $P_1 = P_2 = P_3 = Captured$, eff $W = Happy$

Question!

1. **What are optimal plans for this task?**
2. **What causes the state-space explosion?**
3. **What should POR do to avoid that explosion?**

1. $\langle Blow_1, Capture_1, Blow_2, Capture_2, Blow_3, Capture_3, Banquet\rangle$, $\langle Blow_2, Capture_2, Blow_1, Capture_1, Blow_3, Capture_3, Banquet\rangle$, $\langle Blow_1, Blow_2, Blow_3, Capture_1, Capture_2, Capture_3, Banquet\rangle$, ...
2. The order in which we blow the houses/capture the pigs does not matter. (aka socks/shoes: blow=sock, kidnap=shoe ...)
3. Prune all but one arbitrary fixed order for the blowing/kidnapping.

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.

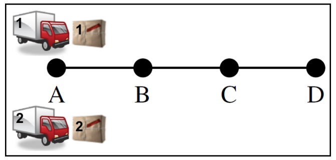
→ What about unsolvable s? ρ may be arbitrary on such states, even $\rho(s) = \emptyset$. (Indeed, $\rho(s) = \emptyset$ on dead-ends s would be ideal for the search.)

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? $a_1 = drive(1, A, B)$ enables $a_2 = drive(1, B, C)$ in $s = I$.

(ii) Example a_1, a_2, s? $drive(1, A, B)$ disables $load(1, A)$ in I.

(iii) Example a_1, a_2, s? $teleport(1, B)$ and $teleport(1, C)$ conflict in I.

Note: The exact state doesn't matter in these examples. (i) works for any s where $a_1 \in A[s]$. (ii) and (iii) work for any s where $a_1 \in A[s]$ and $a_2 \in A[s]$. We will get back to this in **Section S3 Practice**.

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? $load(1, A)$ and $load(2, D)$ are commutative.

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 $\langle a_1, \dots, a_n\rangle$ applicable in s, if $a_i = a$ then $\{a_1, \dots, a_{i-1}\} \cap A_{s \to^* a} \neq \emptyset$.

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 $\langle a_1, \dots, a_n\rangle$ applicable in s that achieves G, $\{a_1, \dots, a_n\} \cap A_{s \to^* G} \neq \emptyset$.

→ 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.

(**Necessary enabling set for G =** "disjunctive action landmark"(**Chapter 14**)

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

- Variables V: $H_1, H_2, H_3 : \{Fine, Broken\}$; $P_1, P_2, P_3 : \{Home, Captured\}, W : \{Hungry, Happy\}\}$.
- Initial state I: $H_1, H_2, H_3 = Fine$, $P_1, P_2, P_3 = Home$, $W = Hungry$.
- Goal G: $W = Happy$.
- Actions A: $Blow(x)$: pre $H_x = Fine$, eff $H_x = Broken$
  $Capture(x)$: pre $P_x = Home$, $H_x = Broken$, eff $P_x = Captured$
  $Banquet$: pre $P_1 = P_2 = P_3 = Captured$, eff $W = Happy$

Question!

Which are necessary enabling sets for G in I?

(A): $\{Blow(1), Blow(2), Blow(3)\}$	(B): $\{Blow(1), Blow(2)\}$
(C): $\{Blow(1)\}$	(D): $\emptyset$

→ (A): Yes, we need to blow the house of at least one pig.

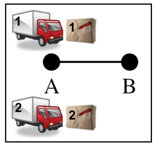
→ (B) and (C): Also yes. We actually need to do this for *every* pig, so every such action is a necessary enabling set on its own.

→ There often are many necessary enabling sets. An important question in practice is how to choose one. See **Section S3 Practice**.

→ (D): No. $\emptyset$ is a necessary enabling set for G in s iff s is unsolvable.

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

Observe: The order in which we transport the packages does not matter.

Idea: Focus on one of the two subgoals first! Say, $pack_1 = B$.

(i) Collect actions needed for our subgoal: $unload(1,B)$.
$\rightarrow$ *Make progress towards the part of G focused on.*

(ii) Collect actions needed for already collected actions: $load(1,A)$.
$\rightarrow$ *Chain backwards to actions applicable in the current state.*

(iii) Collect actions that interfere with already collected applicable actions: $drive(1,A,B)$, $unload(1,A)$.
$\rightarrow$ *Include non-permutable alternatives.*

$\rightarrow$ Other applicable actions (pertaining to truck/package 2) can be safely ignored: we can do this later, with the same effect on the state.

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:*

(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.*

Example: "1/2-Log Small" (cf. previous slide)

(i) $unload(1,B)$.

(ii) $load(1,A)$.

(iii) $drive(1,A,B)$, $unload(1,A)$.

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.*

Proof. It suffices to show (cf. slide 13) that, for every solvable non-goal state s, $\rho_{S3}(s)$ contains at least one action starting a shortest optimal plan for s.

Let s be such a state, and let A_{S3} be the strong stubborn set for s so that $\rho_{S3}(s) = A(s) \cap A_{S3}$. Let $\pi = \langle a_1, \dots, a_n \rangle$ be *any* plan for s. Then

(a) π shares an action with A_{S3}, i.e., $\{a_1, \dots, a_n\} \cap A_{S3} \neq \emptyset$.

This is because, by (i), A_{S3} contains a necessary enabling set for G in s.

Example: "1/2-Log Small" on $s :=$ initial state

- A_{S3}: $\{unload(1,B), load(1,A), drive(1,A,B), unload(1,A)\}$.
- π: $\langle load(2,A), drive(2,A,B), unload(2,B), load(1,A), drive(1,A,B), unload(1,B)\rangle$.
- (a) Shared due to (i): $unload(1,B)$.

Strong Stubborn Sets are Safe, ctd.

Reminder: (a) π shares an action with A_{S3}, i.e., $\{a_1, \dots, a_n\} \cap A_{S3} \neq \emptyset$.

Proof, ctd. Let now a_k be the first shared action, i.e., the one with smallest index in $\{a_1, \dots, a_n\} \cap A_{S3}$. Then

(b) a_k is applicable in s, i.e., $a_k \in A[s]$.

This is because, if $a_k \notin A[s]$, then by (ii) A_{S3} would contain a necessary enabling set $A_{s \rightarrow^* a}$ for a in s, and we would have $\{a_1, \dots, a_{k-1}\} \cap A_{s \rightarrow^* a} \neq \emptyset$ in contradiction to a_k being the first shared action.

Example: "1/2-Log Small" on $s :=$ initial state

- (b) First shared action a_k: $load(1,A)$.

Finally, given this we know that

(c) a_k does not interfere with any of $a_1, \dots, a_{k-1}$.

This is because, if a_k did interfere with a_i for $i < k$, then by (b) and (iii) we would have $a_i \in A_{S3}$, again in contradiction to a_k being the first shared action.

Example: "1/2-Log Small" on $s :=$ initial state

- (c) $load(1,A)$ does not interfere with $load(2,A)$, $drive(2,A,B)$, $unload(2,B)$.

Strong Stubborn Sets are Safe, ctd.

Reminder: (b) The first shared action a_k is applicable in s, i.e., $a_k \in A[s]$.
(c) a_k does not interfere with any of $a_1, \dots, a_{k-1}$.

What remains to be proven? That we can move a_k up front:

Lemma. *Let s be a state, and $\pi_k = \langle a_1, \dots, a_k \rangle$ an action sequence applicable in s where $a_k \in A[s]$. If a_k does not interfere with any of $a_1, \dots, a_{k-1}$, then $\pi'_k := \langle a_k, a_1, \dots, a_{k-1} \rangle$ is applicable in s and $s[\![\pi_k]\!] = s[\![\pi'_k]\!]$.*

Proof. Denote the states traversed by $\langle a_1, \dots, a_k \rangle$ with $s = s_1, s_2, \dots, s_k, s_{k+1}$.
Then $a_k \in A[s_i]$ for $1 \leq i \leq k-1$: $a_k \in A[s_1]$; so $a_k \in A[s_2]$ as a_1 does not disable a_k in s_1; iterate the argument.
Hence, for $1 \leq i \leq k-1$ we have $a_i \in A[s_i]$ and $a_k \in A[s_i]$. As a_k does not disable a_i in s_i, we also have $a_i \in A[s_i[\![a_k]\!]]$. As a_i and a_k do not conflict in s_i, we furthermore have $s_i[\![\langle a_i, a_k \rangle]\!] = s_i[\![\langle a_k, a_i \rangle]\!]$.
Hence we can move a_k to the front iteratively while preserving both the applicability and the outcome state of the action sequence, proving the lemma.

$\Rightarrow$ There exists a permutation of π that starts with a_k, an action contained in $A_{S3} \cap A[s]$ and thus in $\rho_{S3}(s)$. This proves the theorem.

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

$\rightarrow$ Directly, no: The notion of commutative actions is not needed for the definition of S3.

$\rightarrow$ Indirectly, yes: Commutative actions are ones that neither interfere with, nor enable, some action a. If a pertains to the subgoal the S3 focuses on (e.g. a is selected in (i)), then these actions (more generally: non-interfering actions not part of the necessary enabling set for a) are *not* included into A_{S3}.

The S3 Definition as an Algorithm (compare slide 22)

```
input: Planning task $\Pi$, state $s$.
output: Strong stubborn set $S3$ for $s$.
  (i) $S3 := A_{s \to^* G}$ /* 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 $S3$
```

Example: "1/2-Log Small"

- (i) $A_{s \to^* \{pack_1 = B, pack_2 = B\}} = \{unload(1, B)\}$.
- (ii) $A_{s \to^* unload(1,B)} = \{load(1, A)\}$.
- (iii) Interfering a' for $load(1, A)$: $drive(1, A, B)$; interfering a' for $drive(1, A, B)$: $unload(1, A)$.

The S3 Definition as an Algorithm (compare slide 22)

```
input: Planning task $\Pi$, state $s$.
output: Strong stubborn set $S3$ for $s$.
  (i) $S3 := A_{s \to^* G}$ /* 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?

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

Proof. From left to right: Given s as in reminder, clearly (i) and (ii) must hold: (i) as both actions are applicable, (ii) as a_2 is no longer applicable after applying a_1.

From right to left: Due to (i) and the definition of FDR, there exists a state where both actions are applicable. Let s be any such state. Due to (ii), after applying a_1, a_2 is no longer applicable, as we needed to show.

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.*

Proof. From left to right: Say we have s as in the reminder. As both actions are applicable in s, we have (i). As both orders are possible, there cannot be effects disvalidating preconditions so we have (ii) and (iii). As the outcome state differs, we have (iv).

From right to left: Due to (i) and the definition of FDR, there exists a state where both actions are applicable. Let s be any such state. Due to (ii) and (iii), both action orders are possible. Due to (iv), their outcome state differs as we needed to show.

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?

There exists no efficient syntactic characterization:

Theorem. *Let Π be a planning task with actions A, goal G, and states S. Given $a \in A$, $s \in S$ where $a \notin A(s)$, and $A' \subseteq A$, it is* **PSPACE***-complete to decide whether A' is a necessary enabling set for a in s.*

Given $s \in S$ where $G \not\subseteq s$ and $A' \subseteq A$, it is **PSPACE***-complete to decide whether A' is a necessary enabling set for G in s.*

Proof. Second part of claim: $A' := \emptyset$ is a necessary enabling set for G in the initial state iff Π is unsolvable (cf. slide 19). First part of claim: same when adding a new action a whose precondition is G.

So we approximate . . . (simple sufficient criterion)

Proposition. *Let Π be an FDR planning task with actions A, goal G, and states S. Given $a \in A$, $s \in S$ where $a \notin A(s)$, and $p \in pre_a \setminus s$. Then $A' := \{a' \mid p \in eff_{a'}\}$ is a necessary enabling set for a in s.*

Given $s \in S$ where $G \not\subseteq s$ and $p \in G \setminus s$. Then $A' := \{a' \mid p \in eff_{a'}\}$ is a necessary enabling set for G in s.

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 introduced for landmark heuristics in **Chapter 14**.
- There, we also saw more advanced methods for finding such landmarks.

Necessary Enabling Sets: The Choice Makes a Difference!

- Goal G: $W = Happy$.
- Actions A: $Blow(x)$: pre $H_x = Fine$, eff $H_x = Broken$
  $Capture(x)$: pre $P_x = Home$, $H_x = Broken$, eff $P_x = Captured$
  $Banquet$: pre $P_1 = P_2 = P_3 = Captured$, eff $W = Happy$

Question!

How many applicable actions are contained in an S3 for I?

(A): 3 (B): 2

(C): 1 (D): 0

→ (A) – (C): Yes, depends on your choice of necessary enabling sets. For example:

(C) $A_{I\rightarrow^*\{W=Happy\}} = \{Banquet\}$, $A_{I\rightarrow^* banquet} = \{Capture(1)\}$, $A_{I\rightarrow^* Capture(1)} = \{Blow(1)\}$.

(B) $A_{I\rightarrow^*\{W=Happy\}} = \{Banquet\}$, $A_{I\rightarrow^* banquet} = \{Capture(1), Capture(2)\}$, $A_{I\rightarrow^* Capture(1)} = \{Blow(1)\}$, $A_{I\rightarrow^* Capture(2)} = \{Blow(2)\}$.

(A) $A_{I\rightarrow^*\{W=Happy\}} = \{Banquet\}$, $A_{I\rightarrow^* banquet} = \{Capture(1), Capture(2), Capture(3)\}$, $A_{I\rightarrow^* Capture(1)} = \{Blow(1)\}$, $A_{I\rightarrow^* Capture(2)} = \{Blow(2)\}$, $A_{I\rightarrow^* Capture(3)} = \{Blow(3)\}$.

→ (D): No. $\emptyset$ is an S3 for s iff s is unsolvable (cf. slide 19).

Strong Stubborn Sets in STRIPS

Reminder: (slide 9)

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.

→ So, where does it matter? Only in the syntactic characterization of interference, and in the approximation of necessary enabling sets. Everything else applies, exactly as stated, to STRIPS as well.

Interference: (compare slides 29 and 30)

- There exists $s \in S$ s.t. a_1 disables a_2 in s if and only if $del_{a_1} \cap pre_{a_2} \neq \emptyset$.
- There exists $s \in S$ s.t. a_1 and a_2 conflict in s if and only if $del_{a_1} \cap pre_{a_2} = \emptyset$, $del_{a_2} \cap pre_{a_1} = \emptyset$, and either $del_{a_1} \cap add_{a_2} \neq \emptyset$ or $del_{a_2} \cap add_{a_1} \neq \emptyset$.

Necessary enabling sets: (compare slide 31)

- $p \in pre_a \setminus s$ or $p \in G \setminus s$; $A' := \{a' \mid p \in add_{a'}\}$.

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

John Gaschnig. Exactly how good are heuristics?: Toward a realistic predictive theory of best-first search. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI'77)*, pages 434–441, Cambridge, MA, August 1977. William Kaufmann.

Malte Helmert and Gabriele Röger. How good is almost perfect? In Dieter Fox and Carla Gomes, editors, *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, pages 944–949, Chicago, Illinois, USA, July 2008. AAAI Press.

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.