# Artificial Intelligence

## 14. Planning, Part II: Algorithms

How to *Solve* Arbitrary Search Problems

Jana Koehler Álvaro Torralba

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summer Term 2019

Thanks to Prof. Hoffmann for slide sources

## Agenda

## Reminder: Our Agenda for This Topic

→ Our treatment of the topic "Planning" consists of Chapters 13 and 14.

- **Chapter 13:** Background, planning languages, complexity.

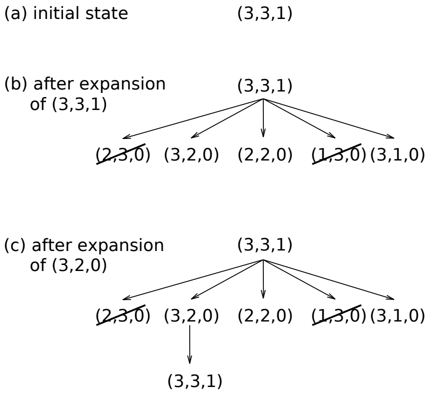
  → Sets up the framework. Computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions (see next).

- **This Chapter:** How to automatically generate a heuristic function, given planning language input?

  → Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

## Reminder: Search

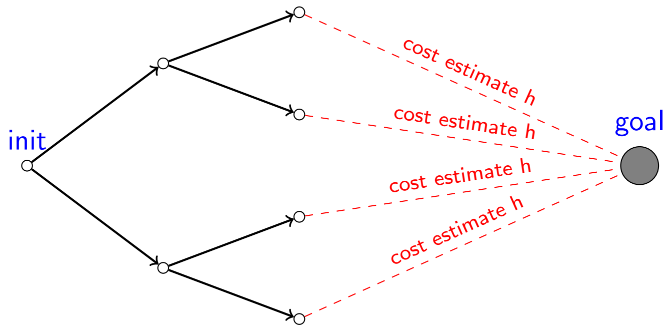
→ Starting at initial state, produce all successor states step by step:



→ In planning, this is referred to as forward search, or forward state-space search.

## Search in the State Space?

## Reminder: Informed Search



→ Heuristic function $h$ estimates the cost of an optimal path from a state $s$ to the goal; search prefers to expand states $s$ with small $h(s)$.

**Live Demo vs. Breadth-First Search:**

`http://qiao.github.io/PathFinding.js/visual/`

## Reminder: Heuristic Functions

**Definition (Heuristic Function).** *Let $\Pi$ be a planning task with states $S$. A heuristic function, short heuristic, for $\Pi$ is a function $h : S \mapsto \mathbb{N}_0^+ \cup \{\infty\}$ so that $h(s) = 0$ whenever $s$ is a goal state.*

$\rightarrow$ Exactly like our definition from **Chapter 5**. Except, because we assume unit costs here, we use $\mathbb{N}_0^+$ instead of $\mathbb{R}_0^+$.

**Definition ($h^*$, Admissibility).** *Let $\Pi$ be a planning task with states $S$. The perfect heuristic $h^*$ assigns every $s \in S$ the length of a shortest path from $s$ to a goal state, or $\infty$ if no such path exists. A heuristic function $h$ for $\Pi$ is admissible if, for all $s \in S$, we have*

$\rightarrow$ Exactly like our definition from **Chapter 5**, except for path *length* instead of path *cost* (cf. above).

$\rightarrow$ In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for $s$. Some algorithms guarantee to lower-bound $h^*(s)$.

## Reminder: Greedy Best-First Search and $A^*$

Duplicate elimination omitted for simplicity:

```
function Greedy Best-First Search [$A^*$](problem) returns a solution, or failure
  node ← a node $n$ with $n$.state=problem.InitialState
  frontier ← a priority queue ordered by ascending $h$ [$g + h$], only element $n$
  loop do
      if Empty?(frontier) then return failure
      $n$ ← Pop(frontier)
      if problem.GoalTest($n$.State) then return Solution($n$)
      for each action $a$ in problem.Actions($n$.State) do
          $n'$ ← ChildNode(problem,$n$,$a$)
          Insert($n'$, $h(n')$ [$g(n') + h(n')$], frontier)
```

$\rightarrow$ Is Greedy Best-First Search optimal?

$\rightarrow$ Is $A^*$ optimal?

## Our Agenda for This Chapter

- **How to Relax:** How to relax a problem?

  → Basic principle for generating heuristic functions.

- **The Delete Relaxation:** How to relax a planning problem?

  → The delete relaxation is the most successful method for the *automatic* generation of heuristic functions. It is a key ingredient to almost all IPC winners of the last decade. It relaxes STRIPS planning tasks by ignoring the delete lists.

- **The $h^+$ Heuristic:** What is the resulting heuristic function?

  → $h^+$ is the "ideal" delete relaxation heuristic.

- **Approximating $h^+$:** How to actually compute a heuristic?

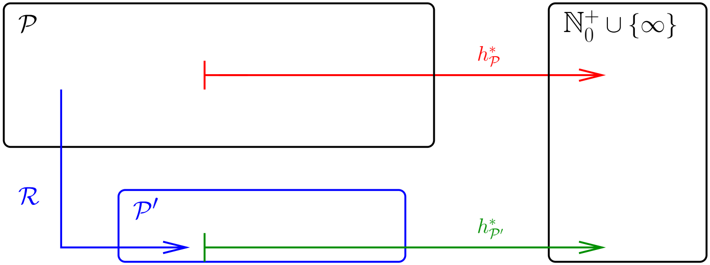
  → Turns out that, in practice, we must approximate $h^+$.

- **An Overview of Advanced Results:** And what else can we do?

  → This section gives a brief glimpse into the state of the art in the area as a whole.

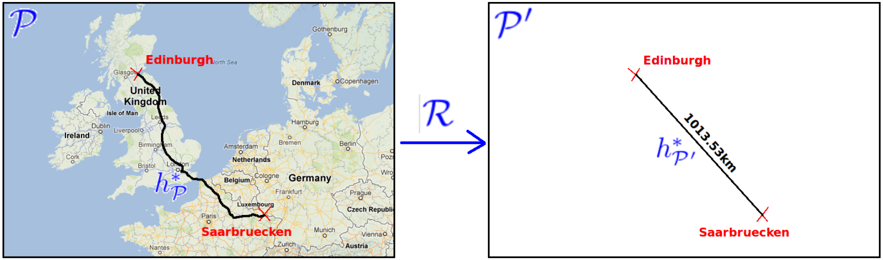
## Reminder: Heuristic Functions from Relaxed Problems

## How to Relax



- You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
- You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to *estimate* $h^*_{\mathcal{P}}$.
- You define a transformation – the relaxation mapping $\mathcal{R}$ – that maps instances $\Pi \in \mathcal{P}$ into instances $\Pi' \in \mathcal{P}'$.
- Given $\Pi \in \mathcal{P}$, you let $\Pi' := \mathcal{R}(\Pi)$, and estimate $h^*_{\mathcal{P}}(\Pi)$ by $h^*_{\mathcal{P}'}(\Pi')$.

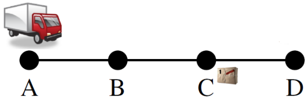
## Relaxation in Route-Finding



- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$:
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$:
- Transformation $\mathcal{R}$:

## How to Relax in Planning? (A Reminder!)

**Example:** "Logistics"



- Facts $P$: $\{truck(x) \mid x \in \{A, B, C, D\}\} \cup \{pack(x) \mid x \in \{A, B, C, D, T\}\}$.
- Initial state $I$: $\{truck(A), pack(C)\}$.
- Goal $G$: $\{truck(A), pack(D)\}$.
- Actions $A$: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $drive(x, y)$, where $x, y$ have a road: "$truck(x) \Rightarrow truck(y), \neg truck(x)$".
  - $load(x)$: "$truck(x), pack(x) \Rightarrow pack(T), \neg pack(x)$".
  - $unload(x)$: "$truck(x), pack(T) \Rightarrow pack(x), \neg pack(T)$".

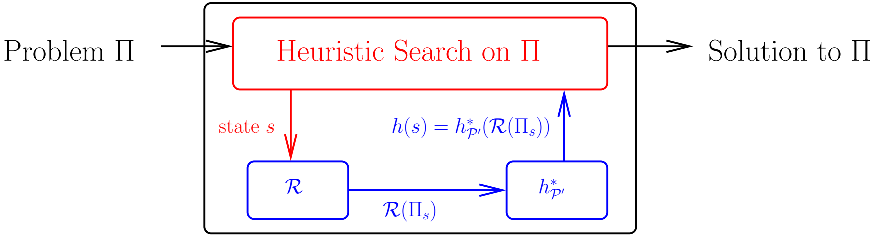
**Example "Only-Adds" Relaxation:** Drop the preconditions and deletes.

"$drive(x, y)$: $\Rightarrow truck(y)$"; "$load(x)$: $\Rightarrow pack(T)$"; "$unload(x)$: $\Rightarrow pack(x)$".

$\rightarrow$ Heuristic value for $I$ is?

## How to Relax During Search: Overview

**Attention!** Search uses the real (un-relaxed) $\Pi$. The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) **only within the call to $h(s)$!!!**

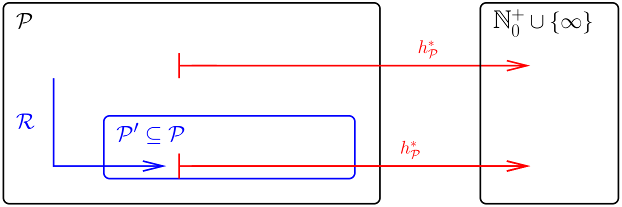


- Here, $\Pi_s$ is $\Pi$ with initial state replaced by $s$, i.e., $\Pi = (P, A, I, G)$ changed to $(P, A, s, G)$: The task of finding a plan for search state $s$.
- A common student mistake is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".
- The next slide illustrates the correct search process in detail.

## How to Relax During Search: Only-Adds

## Only-Adds is a "Native" Relaxation

**Native Relaxations:** Confusing special case where $\mathcal{P}' \subseteq \mathcal{P}$.


$\mathcal{P}$
$\mathbb{N}_0^+ \cup \{\infty\}$
$h^*_{\mathcal{P}}$
$\mathcal{R}$
$\mathcal{P}' \subseteq \mathcal{P}$
$h^*_{\mathcal{P}}$

- Problem class $\mathcal{P}$: STRIPS planning tasks.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length $h^*$ of a shortest plan.
- Transformation $\mathcal{R}$: Drop the preconditions and delete lists.
- Simpler problem class $\mathcal{P}'$ is a special case of $\mathcal{P}$, $\mathcal{P}' \subseteq P$: STRIPS planning tasks with empty preconditions and delete lists.
- Perfect heuristic for $\mathcal{P}'$: Shortest plan for only-adds STRIPS task.

## Questionnaire

Question!

**Does Only-Adds yield a "good heuristic" (accurate goal distance estimates) in . . .**

(A): Freecell?

(B): SAT? (#unsatisfied clauses)

(C): Blocksworld?

(D): Path Planning?

## How the Delete Relaxation Changes the World

## The Delete Relaxation

**Definition (Delete Relaxation).** *Let $\Pi = (P, A, I, G)$ be a planning task. The delete-relaxation of $\Pi$ is the task $\Pi^+ = (P, A^+, I, G)$ where $A^+ = \{a^+ \mid a \in A\}$ with $pre_{a^+} = pre_a$, $add_{a^+} = add_a$, and $del_{a^+} =$*

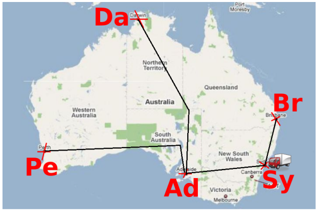
→ In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS planning tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.

**Definition (Relaxed Plan).** *Let $\Pi = (P, A, I, G)$ be a planning task, and let $s$ be a state. A relaxed plan for $s$ is a plan for . A relaxed plan for $I$ is called a relaxed plan for $\Pi$.*

→ A relaxed plan for $s$ is an action sequence that solves $s$ when pretending that all delete lists are empty.

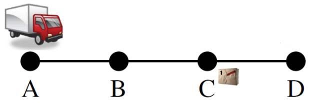
→ Also called delete-relaxed plan; "relaxation" is often used to mean "delete-relaxation" by default.

## A Relaxed Plan for "TSP" in Australia



1. **Initial state:** $\{at(Sydney), visited(Sydney)\}$.
2. **Apply** $drive(Sydney, Brisbane)^+$**:** $\{at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
3. **Apply** $drive(Sydney, Adelaide)^+$**:** $\{at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
4. **Apply** $drive(Adelaide, Perth)^+$**:** $\{at(Perth), visited(Perth), at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
5. **Apply** $drive(Adelaide, Darwin)^+$**:** $\{at(Darwin), visited(Darwin), at(Perth), visited(Perth), at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.

## A Relaxed Plan for "Logistics"



- Facts $P$: $\{truck(x) \mid x \in \{A,B,C,D\}\} \cup pack(x) \mid x \in \{A,B,C,D,T\}\}$.
- Initial state $I$: $\{truck(A), pack(C)\}$.
- Goal $G$: $\{truck(A), pack(D)\}$.
- Relaxed actions $A^+$: (Notated as "precondition $\Rightarrow$ adds")
  - $drive(x,y)^+$: "$truck(x) \Rightarrow truck(y)$".
  - $load(x)^+$: "$truck(x), pack(x) \Rightarrow pack(T)$".
  - $unload(x)^+$: "$truck(x), pack(T) \Rightarrow pack(x)$".

**Relaxed plan:**

## Questionnaire

Question!

**How does ignoring delete lists simplify Sokoban?**

(A): Free positions remain free.

(B): You can walk through walls.

(C): A single action can push 2 stones at once.

(D): You will never “lock yourself in”.

## PlanEx$^+$

**Definition (Relaxed Plan Existence Problem).** *By PlanEx$^+$, we denote the problem of deciding, given a planning task $\Pi = (P, A, I, G)$, whether or not there exists a relaxed plan for $\Pi$.*

$\rightarrow$ This is easier than PlanEx for general STRIPS!

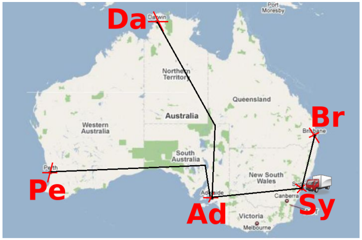
**Proposition (PlanEx$^+$ is Easy).** *PlanEx$^+$ is a member of* **P**.

**Proof.** The following algorithm decides PlanEx$^+$:

```
$F := I$
while $G \not\subseteq F$ do
        $F' := F \cup \bigcup_{a \in A: pre_a \subseteq F} add_a$
        (*) if $F' = F$ then return "unsolvable" endif
        $F := F'$
endwhile
return "solvable"
```

The algorithm terminates after at most

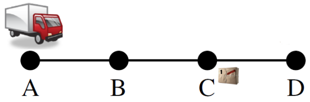
## Deciding PlanEx$^+$ in "TSP" in Australia



**Iterations on $F$:**

1. $\{at(Sydney), visited(Sydney)\}$
2. $\cup \; \{at(Adelaide), visited(Adelaide), \; at(Brisbane), visited(Brisbane)\}$
3. $\cup \; \{at(Darwin), visited(Darwin), \; at(Perth), visited(Perth)\}$

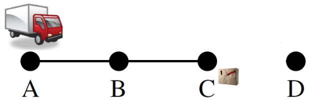
## Deciding PlanEx$^+$ in “Logistics”



**Iterations on $F$:**

1. $\{truck(A), pack(C)\}$
2. $\cup$
3. $\cup$
4. $\cup$
5. $\cup$

## Deciding PlanEx$^+$ in Unsolvable "Logistics"

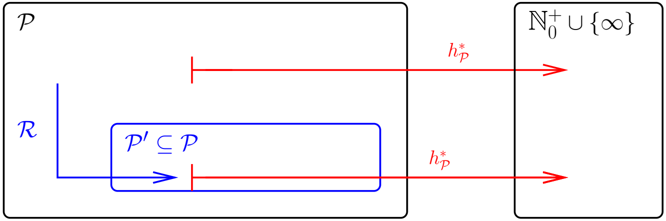


**Iterations on $F$:**

1. $\{truck(A), pack(C)\}$
2. $\cup$
3. $\cup$
4. $\cup$
5. $\cup$
6. $\cup$

## PlanEx$^+$ Algorithm: Proof

## Hold on a Sec – Where are we?


$\mathcal{P}$
$\mathbb{N}_0^+ \cup \{\infty\}$
$h^*_{\mathcal{P}}$
$\mathcal{R}$
$\mathcal{P}' \subseteq \mathcal{P}$
$h^*_{\mathcal{P}}$

- $\mathcal{P}$: STRIPS planning tasks; $h^*_{\mathcal{P}}$: Length $h^*$ of a shortest plan.
- $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS planning tasks with empty delete lists.
- $\mathcal{R}$: Drop the delete lists.
- Heuristic function: Length of a shortest *relaxed* plan $(h^* \circ \mathcal{R})$.

$\rightarrow$ PlanEx$^+$ is not actually what we're looking for. PlanEx$^+ =$ relaxed plan *existence*; we want relaxed plan *length* $h^* \circ \mathcal{R}$.

## $h^+$: The Ideal Delete Relaxation Heuristic

**Definition (Optimal Relaxed Plan).** *Let $\Pi = (P, A, I, G)$ be a planning task, and let $s$ be a state. An optimal relaxed plan for $s$ is an optimal plan for $(P, A, s, G)^+$.*

$\rightarrow$ Same as slide 22, just adding the word "optimal".

**Here's what we're looking for:**

**Definition ($h^+$).** *Let $\Pi = (P, A, I, G)$ be a planning task with states $S$. The ideal delete-relaxation heuristic $h^+$ for $\Pi$ is the function $h^+ : S \mapsto \mathbb{N}_0 \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for $s$ if a relaxed plan for $s$ exists, and $h^+(s) = \infty$ otherwise.*

$\rightarrow$ In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

## $h^+$ is Admissible

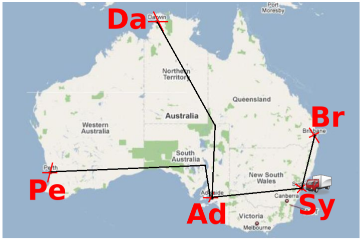
**Lemma.** *Let $\Pi = (P, A, I, G)$ be a planning task, and let $s$ be a state. If $\langle a_1, \dots, a_n \rangle$ is a plan for $(P, A, s, G)$, then $\langle a_1^+, \dots, a_n^+ \rangle$ is a plan for $(P, A, s, G)^+$.*

**Proof Sketch.** Show by induction over $0 \leq i \leq n$ that $appl(s, \langle a_1, \dots, a_i \rangle) \subseteq appl(s, \langle a_1^+, \dots, a_i^+ \rangle)$.
$\rightarrow$ "If we ignore deletes, the states along the plan can only get bigger."

**Theorem.** *$h^+$ is Admissible.*

**Proof.** Let $\Pi = (P, A, I, G)$ be a planning task with states $S$, and let $s \in S$. $h^+(s)$ is defined as optimal plan length in $(P, A, s, G)^+$. With the above lemma, any plan for $(P, A, s, G)$ also constitutes a plan for $(P, A, s, G)^+$. Thus optimal plan length in $(P, A, s, G)^+$ can only be shorter than that in $(P, A, s, G)$, and the claim follows.

## $h^+$ in "TSP" in Australia



**Planning vs. Relaxed Planning:**

- Optimal plan: $\langle drive(Sydney, Brisbane), drive(Brisbane, Sydney), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Perth, Adelaide), drive(Adelaide, Darwin), drive(Darwin, Adelaide), drive(Adelaide, Sydney)\rangle$.
- Optimal relaxed plan: $\langle drive(Sydney, Brisbane), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Adelaide, Darwin)\rangle$.
- $h^*(I) = 8$; $h^+(I) = 4$.

## How to Relax During Search: Ignoring Deletes

## On the "Accuracy" of $h^+$

**Reminder:** Heuristics based on ignoring deletes are the key ingredient to almost all IPC winners of the last decade.

$\rightarrow$ **Why?**

$\rightarrow$ A heuristic function is useful if its estimates are "accurate".

**How to measure this?**

- **Known method 1:** Error relative to $h^*$, i.e., bounds on $|h^*(s) - h(s)|$.
- **Known method 2:** Properties of the search space surface: Local minima etc.

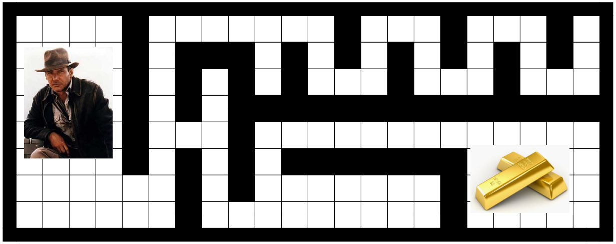
$\rightarrow$ For $h^+$, method 2 is the road to success:

$\rightarrow$ In many benchmarks, under $h^+$, local minima *provably* do not exist! [Hoffmann (2005)]

## A Brief Glimpse of $h^+$ Search Space Surfaces

## $h^+$ in Graphs

## Questionnaire



**Question!**

**In this domain, $h^+$ is equal to?**

(A): Manhattan Distance.

(B): Horizontal distance.

(C): Vertical distance.

(D): $h^*$.

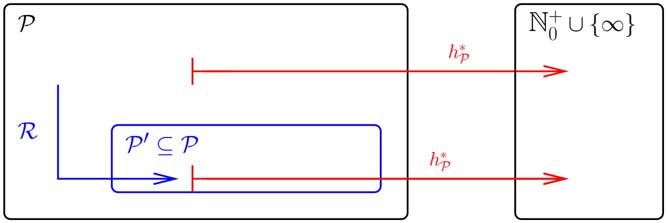
## How to Compute $h^+$?

**Definition (PlanLen$^+$).** *By PlanLen$^+$, we denote the problem of deciding, given a planning task $\Pi$ and an integer $B$, whether there exists a relaxed plan for $\Pi$ of length at most $B$.*

$\rightarrow$ By computing $h^+$, we solve PlanLen$^+$.

**Theorem.** *PlanLen$^+$ is* **NP***-complete.*

## Hold on a Sec – Where are we?


$\mathcal{P}$
$\mathbb{N}_0^+ \cup \{\infty\}$
$h^*_{\mathcal{P}}$
$\mathcal{R}$
$\mathcal{P}' \subseteq \mathcal{P}$
$h^*_{\mathcal{P}}$

- $\mathcal{P}$: STRIPS planning tasks; $h^*_{\mathcal{P}}$: Length $h^*$ of a shortest plan.
- $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS planning tasks with empty delete lists.
- $\mathcal{R}$: Drop the delete lists.
- Heuristic function: $h^+ = h^* \circ \mathcal{R}$, which is hard to compute.

$\rightarrow$ We can't compute our heuristic $h^+$ efficiently. So we approximate it instead.

## Approximating $h^+$: $h^{\text{FF}}$

**Definition ($h^{\text{FF}}$).** *Let $\Pi = (P, A, I, G)$ be a planning task with states $S$. A relaxed plan heuristic $h^{\text{FF}}$ for $\Pi$ is a function $h^{\text{FF}} : S \mapsto \mathbb{N}_0 \cup \{\infty\}$ returning the length of some, not necessarily optimal, relaxed plan for $s$ if a relaxed plan for $s$ exists, and returning $h^{\text{FF}}(s) = \infty$ otherwise.*

**Notes:**

- $h^{\text{FF}} \geq h^+$, i.e., $h^{\text{FF}}$ never under-estimates $h^+$.
- We may have $h^{\text{FF}} > h^*$, i.e., $h^{\text{FF}}$ is not admissible! Thus $h^{\text{FF}}$ can be used for satisficing planning only, not for optimal planning.

**Observe:** $h^{\text{FF}}$ as per this definition is not unique. How do we find *"some, not necessarily optimal, relaxed plan for $(P, A, s, G)$"*?

$\rightarrow$ In what follows, we consider the following algorithm computing relaxed plans, and therewith (one variant of) $h^{\text{FF}}$:

1. Chain forward to build a relaxed planning graph (RPG).
2. Chain backward to extract a relaxed plan from the RPG.

## Computing $h^{\text{FF}}$: Relaxed Planning Graphs (RPG)

```
$F_0 := s$, $t := 0$
**while** $G \not\subseteq F_t$ **do**
        $A_t := \{a \in A \mid pre_a \subseteq F_t\}$
        $F_{t+1} := F_t \cup \bigcup_{a \in A_t} add_a$
        **if** $F_{t+1} = F_t$ **then** stop **endif**
        $t := t + 1$
**endwhile**
```

$\rightarrow$ Does this look familiar to you?

## Computing $h^{\mathsf{FF}}$: Extracting a Relaxed Plan

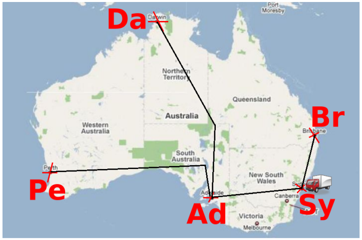
**Information from the RPG:** ($\min$ over an empty set is $\infty$)

- For $p \in P$: $level(p) := \min\{t \mid p \in F_t\}$.
- For $a \in A$: $level(a) := \min\{t \mid a \in A_t\}$.

```
$M := \max\{level(p) \mid p \in G\}$
**If** $M = \infty$ **then** $h^{\mathsf{FF}}(s) := \infty$; stop **endif**
**for** $t := 0, \dots, M$ **do**
    $G_t := \{g \in G \mid level(g) = t\}$
**endfor**
**for** $t := M, \dots, 1$ **do**
    **for** all $g \in G_t$ **do**
        select $a$, $level(a) = t - 1$, $g \in add_a$
        **for** all $p \in pre_a$ **do** $G_{level(p)} := G_{level(p)} \cup \{p\}$ **endfor**
    **endfor**
**endfor**
$h^{\mathsf{FF}}(s) :=$ number of selected actions
```

**"Logistics" example:** Blackboard.

## Computing $h^{FF}$ in “TSP” in Australia



**RPG:**

- $F_0 =$
- $A_0 =$
- $F_1 = F_0 \cup$
- $A_1 = A_0 \cup \quad drive(Adelaide, Sydney), \; drive(Brisbane, Sydney)\}.$
- $F_2 = F_1 \cup$

## Other Approximations of $h^+$: $h^{\max}$

$h^{\max}$: Approximate the cost of fact set $g$ by the most costly single fact $p \in g$

$$h^{\max}(s,g) := \begin{cases} 0 & g \subseteq s \\ min_{a \in A, p \in add_a} 1 + h^{\max}(s, pre_a) & g = \{p\} \\ \max_{p \in g} h^{\max}(s, \{p\}) & |g| > 1 \end{cases}$$

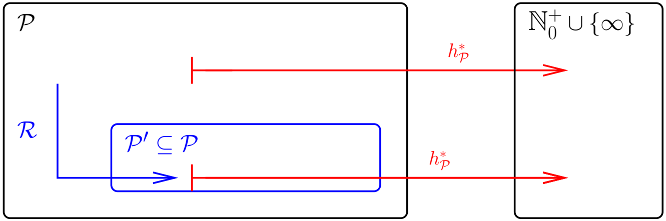
Computing $h^{\max}$(s) with the relaxed planning graph algorithm:

- The $h^{\max}$ value of a fact in a state $h^{\max}(s,\{p\})$ corresponds to the layer of the relaxed planning graph in which it first appeared, $h^{\max}(s,\{p\}) = level(p)$.
- The $h^{\max}$ value of a state is $h^{\max}(s) = \max_{p \in G} h^{\max}(s,\{p\})$. So, if $F_i$ is the first layer such that $G \subseteq F_i$, $h^{\max}(s) = i$.

For example, in TSP Australia: $h^{\max}(I) = \max(h^{\max}(I, \{v(Sy)\}),$ $h^{\max}(I,\{v(Ad)\}), h^{\max}(I,\{v(Da)\}), h^{\max}(I,\{v(Pe)\}), h^{\max}(I,\{v(Br)\}),$ $h^{\max}(I,\{at(Sy)\})) = max(0,1,2,2,1,0) = 2.$

$\rightarrow$ Admissible, but very uninformative (under-estimates vastly).

## How Does it All Fit Together?


$\mathcal{P}$
$\mathbb{N}_0^+ \cup \{\infty\}$
$h^*_{\mathcal{P}}$
$\mathcal{R}$
$\mathcal{P}' \subseteq \mathcal{P}$
$h^*_{\mathcal{P}}$

$\mathcal{P}$: STRIPS planning tasks. $h^*_{\mathcal{P}}$: Length $h^*$ of a shortest plan. $\mathcal{P}'$: STRIPS planning tasks with empty delete lists. $\mathcal{R}$: Drop the delete lists. $h^* \circ \mathcal{R}$: Length $h^+$ of a shortest relaxed plan.

$\rightarrow$ Use $h^{\text{FF}}$ to approximate $h^+$ which itself is hard to compute.

$\rightarrow$ $h^+$ and $h^{\text{max}}$ are admissible; $h^{\text{FF}}$ is not.

## Before We Begin

**Challenge:** Given a planning task $\Pi$, simplify $\Pi$ to obtain a relaxed planning task $\Pi'$, then solve $\Pi'$ to obtain the heuristic estimate $h$. All of this must be fully automatic.

**Method 1:** $\Pi' :=$ delete relaxation, $h := h^{\mathsf{FF}}$.

**This is a HUGE playground!** Abstract/relax the world WHICHEVER way!

**Methods 2, 3, 4, . . . :** Up next!

$\rightarrow$ In what follows, I'm going to give you an overview over the state of the art in this area.

**ATTENTION!** You're not going to understand all of this, and it's not *intended* for you to understand all of this.

$\rightarrow$ It's intended as a (hopefully interesting) glimpse into this area, and as an appetizer for my specialized lecture **Automatic Planning** this winter. (And, no, it's not relevant to the exam.)

## The 4 Families (That We Know Up To Now)

**Ignoring Deletes**

**Abstractions**

**Critical Paths**

...

**Landmarks**

## Abstractions: Idea

## Abstractions: Example

## Abstractions: Details

- What is an abstraction, formally?

  $\rightarrow$ An abstraction is a function $\alpha$ mapping the state space (all world states) to a (smaller) set of abstract world states.
- How is the corresponding heuristic function $h^\alpha$ defined?
- What is a pattern database heuristic?

  $\rightarrow$ A pattern database heuristic (PDB) is an abstraction heuristic $h^\alpha$ where $\alpha$ is a projection, i.e., $\alpha(s) = \alpha(t)$ iff $s$ and $t$ agree on a subset of the state variables (e.g., those encoding the positions of $1, \dots, 7$ and the blank).
- What is a merge-and-shrink heuristic?

  $\rightarrow$ A merge-and-shrink heuristic (M&S) is an abstraction heuristic $h^\alpha$ constructed by starting with projections on single variables, then iteratively merging two abstractions (replacing them with their synchronized product) and shrinking an abstraction (aggregating pairs of abstract states).

## Abstractions: (Some) Recent Results

## Landmarks: Example

## Landmarks: Details

- What is a fact landmark?
- What is a disjunctive action landmark?
- How can we turn a fact landmark into a disjunctive action landmark?
- Can *all* disjunctive action landmarks be derived that way?

## Landmarks: (Some) Recent Results

## Critical Paths: Example

## Critical Paths: Details

- How is $h^1$ defined?

  $h^1(s) := h^1(s, G)$ where $h^1(s, g)$ is the point-wise greatest function that satisfies $h^1(s, g) =$

  $$\begin{cases} 0 & g \subseteq s \\ min_{a \in A, regr(g,a) \text{ is defined}} \, c(a) + h^1(s, regr(g, a)) & |g| = 1 \\ \max_{g' \in g} h^1(s, \{g'\}) & |g| > 1 \end{cases}$$

  $\rightarrow$ This is the same as $h^{\text{max}}$ (cf. slide 48).

- How is $h^m$ defined?

## Critical Paths: (Some) Recent Results

## Ignoring Deletes: (Some) Recent Results

## And, BTW: “Compilability” Between the Families

## Questionnaire

Question!

**What do you think the compilability results are good for?**

(A): Nothing.

(B): Theory.

(C): Configuring solvers.

(D): Inventing new solvers.

## Summary

- Heuristic search on classical search problems relies on a function $h$ mapping states $s$ to an estimate $h(s)$ of their goal distance. Such functions $h$ are derived by solving relaxed problems.
- In planning, the relaxed problems are generated and solved automatically. There are four known families of suitable relaxation methods: abstractions, landmarks, critical paths, and ignoring deletes (aka delete relaxation).
- The delete relaxation consists in dropping the deletes from STRIPS planning tasks. A relaxed plan is a plan for such a relaxed task. $h^+(s)$ is the length of an optimal relaxed plan for state $s$. $h^+$ is **NP**-hard to compute.
- $h^{\mathsf{FF}}$ approximates $h^+$ by computing some, not necessarily optimal, relaxed plan. That is done by a forward pass (building a relaxed planning graph), followed by a backward pass (extracting a relaxed plan).

## Topics We Didn't Cover Here

- Abstractions, Landmarks, Critical-Path Heuristics, Cost Partitionings, Compilability between Heuristic Functions, Planning Competitions: $\rightarrow$ Upcoming course **Automatic Planning**.
- Tractable fragments: Planning sub-classes that can be solved in polynomial time. Often identifed by properties of the "causal graph" and "domain transition graphs". $\rightarrow$ **Automatic Planning**.
- Planning as SAT: Compile length-$k$ bounded plan existence into satisfiability of a CNF formula $\varphi$. Extensive literature on how to obtain small $\varphi$, how to schedule different values of $k$, how to modify the underlying SAT solver.
- Compilations: Formal framework for determining whether planning formalism $X$ is (or is not) at least as expressive as planning formalism $Y$.
- Admissible pruning/decomposition methods: Partial-order reduction, symmetry reduction, simulation-based dominance pruning, factored planning, decoupled search. $\rightarrow$ **Automatic Planning**.
- Numeric planning, temporal planning, planning under uncertainty: ...

## Reading (RN: Same As Previous Chapter)

- *Chapters 10: Classical Planning and 11: Planning and Acting in the Real World* [Russell and Norvig (2010)].

  Content: Although the book is named "A Modern Approach", the planning section was written long before the IPC was even dreamt of, before PDDL was conceived, and several years before heuristic search hit the scene. As such, what we have right now is the attempt of two outsiders trying in vain to catch up with the dramatic changes in planning since 1995.

  Chapter 10 is Ok as a background read. Some issues are, imho, misrepresented, and it's far from being an up-to-date account. But it's Ok to get some additional intuitions in words different from my own.

  Chapter 11 is annoyingly named (I've seen lots of classical planning in the "real world"), but is useful in our context here because I don't cover any of it. If you're interested in extended/alternative planning paradigms, do read it.

## Reading, ctd.

- *The FF Planning System: Fast Plan Generation Through Heuristic Search* [Hoffmann and Nebel (2001)].

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/jair01.pdf`

  Content: The main reference for delete relaxation heuristics.

- *Semi-Relaxed Plan Heuristics* [Keyder *et al.* (2012)].

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/icaps12a.pdf`

  Content: Computes relaxed plan heuristics within a compiled planning task $\Pi^C_{ce}$, in which a subset $C$ of all fact conjunctions in the task is represented explicitly. $C$ can in principle always be chosen so that $h^+_{\Pi^C_{ce}}$ is perfect, so the technique allows to interpolate between $h^+$ and $h^*$. In practice, small sets $C$ sometimes suffice to obtain dramatically more informed relaxed plan heuristics.

## References I

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Jörg Hoffmann. Where ‘ignoring delete lists’ works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

Emil Keyder, Jörg Hoffmann, and Patrik Haslum. Semi-relaxed plan heuristics. 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)*, pages 128–136. AAAI Press, 2012.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2010.