Artificial Intelligence

13. Planning, Part II: Algorithms

How to *Solve* Arbitrary Search Problems

Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Online (Summer) Term 2020

Agenda

Reminder: Our Agenda for This Topic

→ Our treatment of the topic "Planning" consists of Chapters 12 and 13.

- **Chapter 12:** Background, planning languages, complexity.

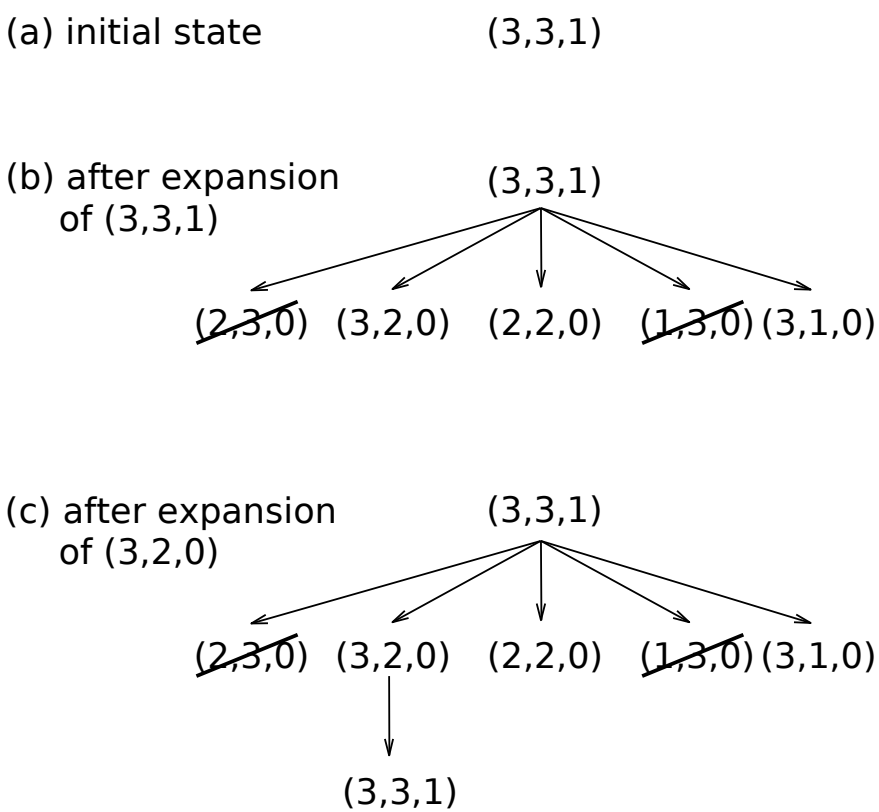
  → Sets up the framework. Computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions (see next).
- **This Chapter:** How to automatically generate a heuristic function, given planning language input?

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

→ We focus on model-based techniques. The use of neural networks is an active research topic (in my research group among others). It's difficult due to the extremely general nature of planning languages.

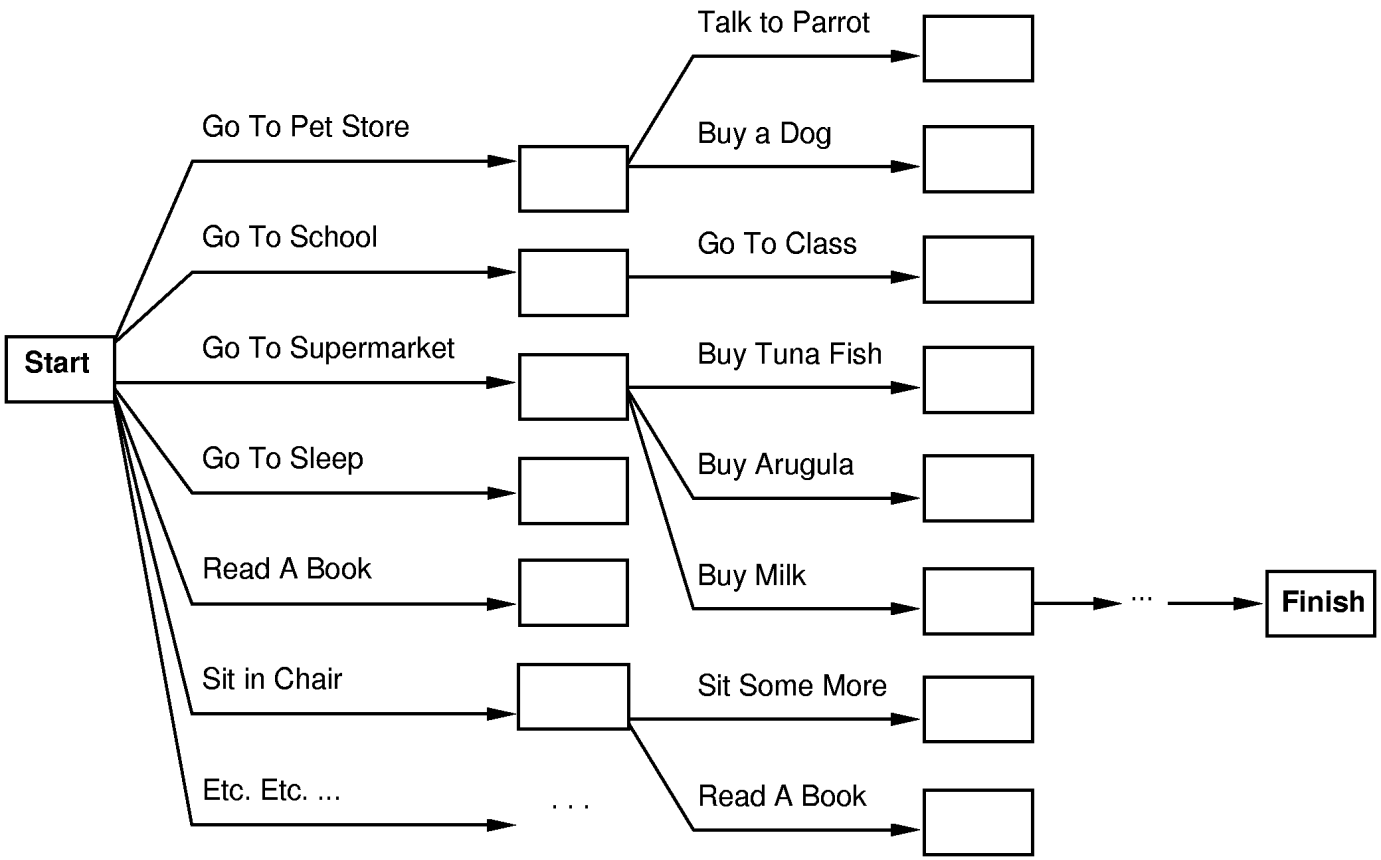
Reminder: Search

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



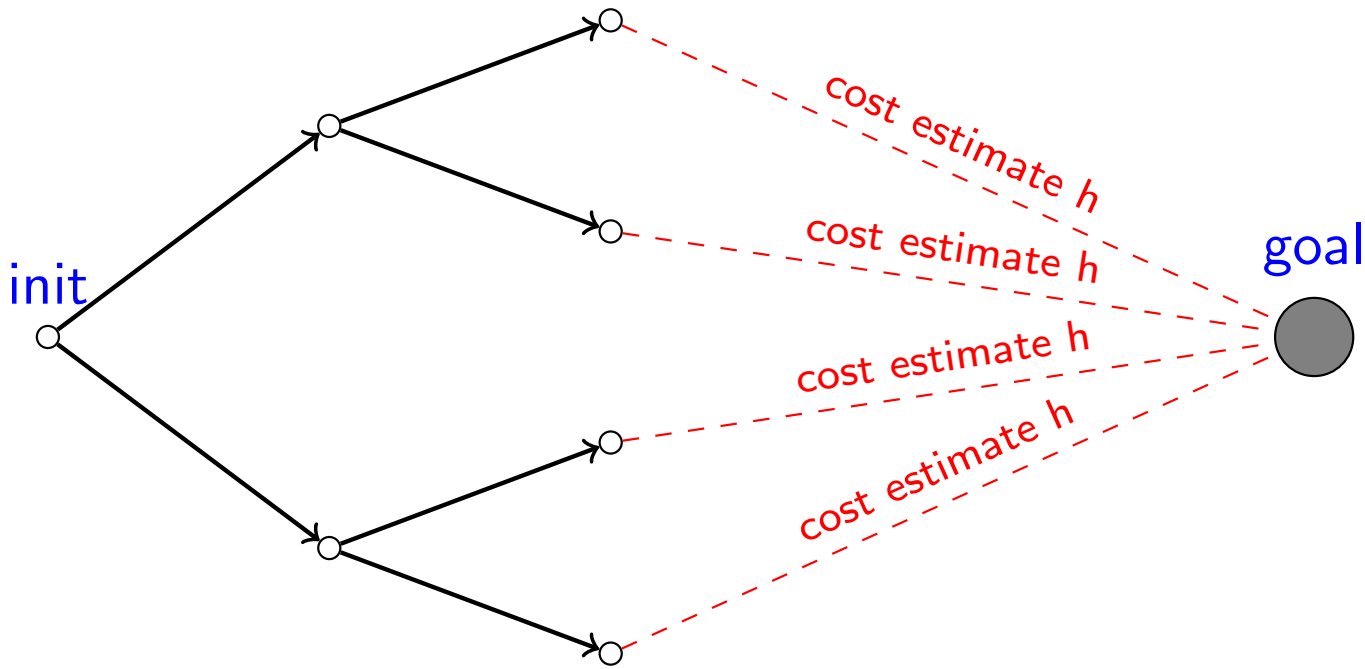
→ In planning, this is referred to as forward search, or forward state-space search.

Search in the State Space?



→ Use heuristic function to guide the search towards the goal!

Reminder: Heuristic Search



$\rightarrow$ 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 Π be a planning task with states S. A heuristic function, short heuristic, for Π 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 2**. Except, because we assume unit costs here, we use $\mathbb{N}_0^+$ instead of $\mathbb{R}_0^+$.

Definition (h^*, Admissibility). *Let Π 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 ∞ if no such path exists. A heuristic function h for Π is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$.*

$\rightarrow$ Exactly like our definition from **Chapter 2**, 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 $\leftarrow$ a node n with n.*state*=*problem.InitialState*
  frontier $\leftarrow$ a priority queue ordered by ascending h [$g + h$], only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n.*State*) **do**
      $n' \leftarrow$ *ChildNode(problem,*n*,*a*)*
      Insert(n', $h(n')$ [$g(n') + h(n')$], *frontier*)

$\rightarrow$ Is Greedy Best-First Search optimal? No $\Longrightarrow$ *satisficing* planning.

$\rightarrow$ Is A^* optimal? Yes, but only if h is admissible $\Longrightarrow$
optimal planning, **with such** h.

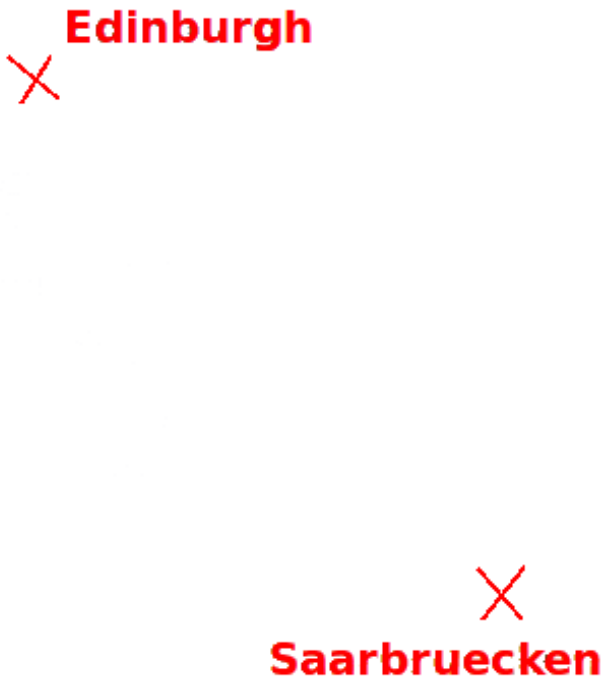
Our Agenda for This Chapter

- **How to Relax:** How to relax a problem?
  $\rightarrow$ Basic principle for generating heuristic functions.
- **The Delete Relaxation:** How to relax a planning problem?
  $\rightarrow$ The delete relaxation is the most successful method for the *automatic* generation of heuristic functions. It is a key ingredient of many IPC winners during the last two decades. It relaxes STRIPS planning tasks by ignoring the delete lists.
- **The h^+ Heuristic:** What is the resulting heuristic function?
  $\rightarrow$ h^+ is the "ideal" delete relaxation heuristic.
- **Approximating h^+:** How to actually compute a heuristic?
  $\rightarrow$ Turns out that, in practice, we must approximate h^+.
- **An Overview of Advanced Results:** Is that all?
  $\rightarrow$ No! This section gives a brief glimpse into the research area of heuristic search planning.

Heuristic Functions from Relaxed Problems

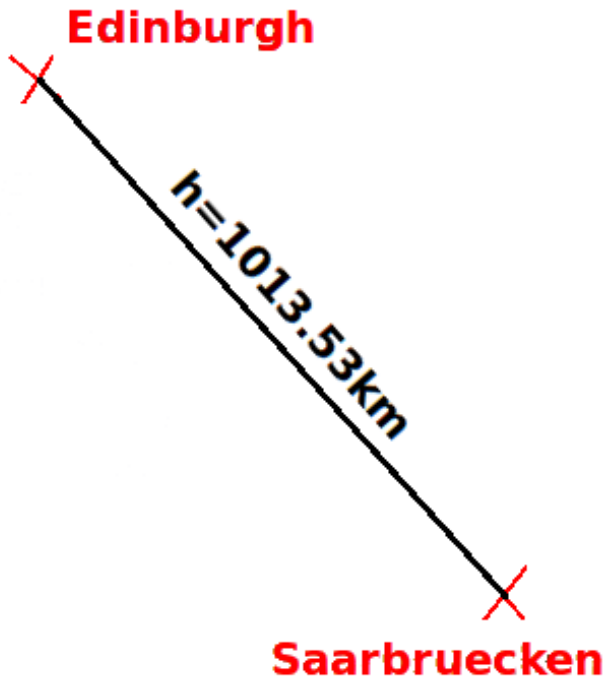
Problem II: Find a route from Saarbruecken To Edinburgh.

Heuristic Functions from Relaxed Problems



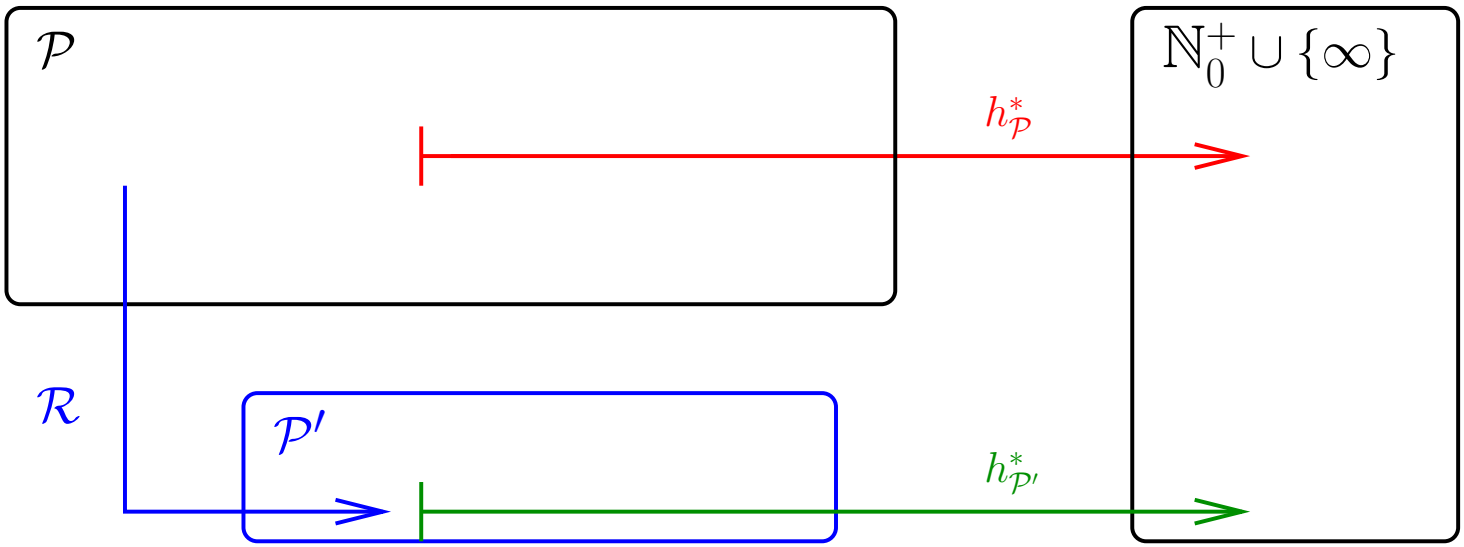
Relaxed Problem Π': Throw away the map.

Heuristic Functions from Relaxed Problems



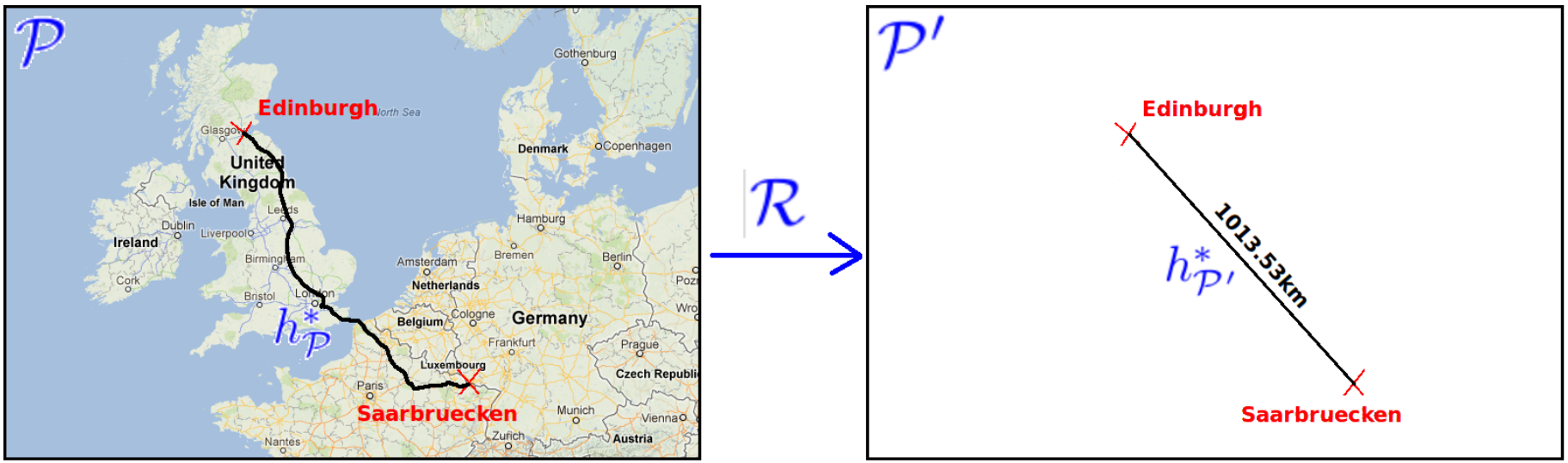
Heuristic function h: Straight line distance.

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 $h^*_{\mathcal{P}'}(\Pi')$.

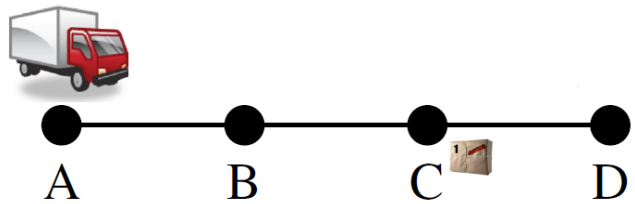
Relaxation in Route-Finding



- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$: Route finding on an empty map.
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$: Straight-line distance.
- Transformation $\mathcal{R}$: Throw away the map.

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

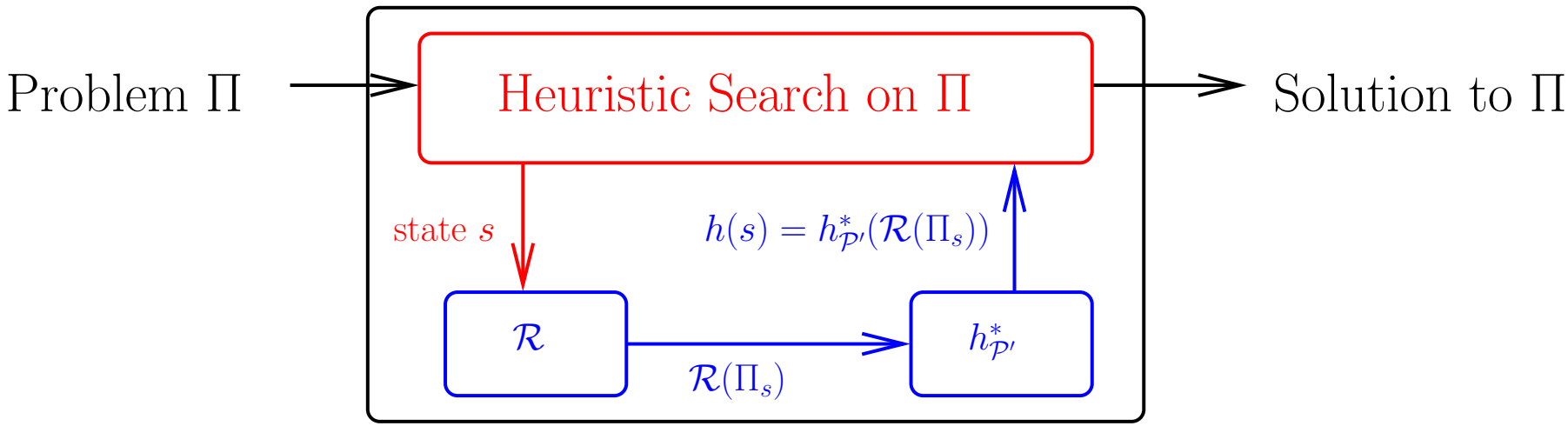
Example "Only-Adds" Relaxation: Drop the preconditions and deletes.

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

$\rightarrow$ Heuristic value for I is? 1: A plan for the relaxed task is $\langle unload(D) \rangle$.

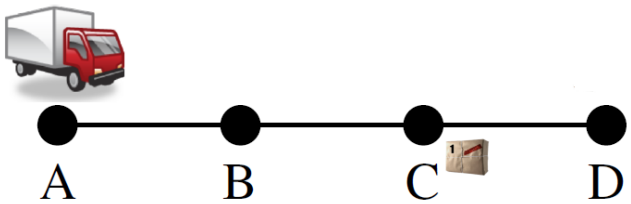
How to Relax During Search: Overview

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



- Here, Π_s is Π 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.

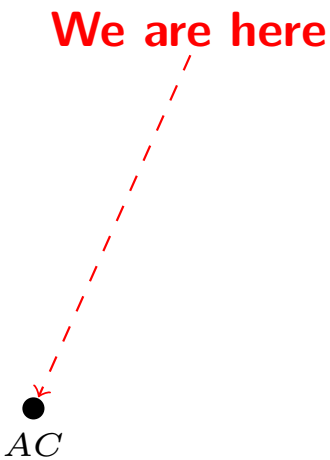
How to Relax During Search: Only-Adds



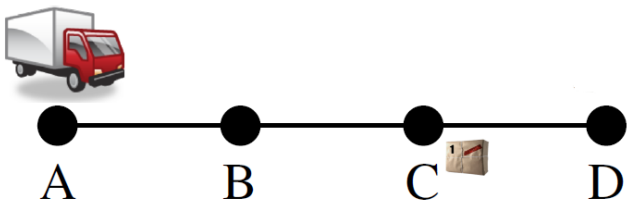
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



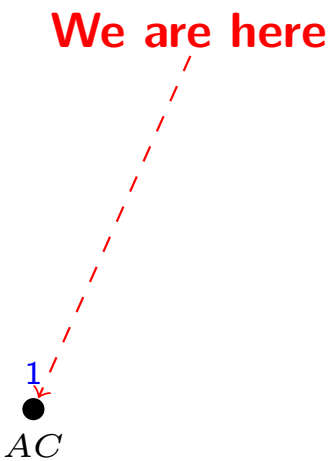
How to Relax During Search: Only-Adds



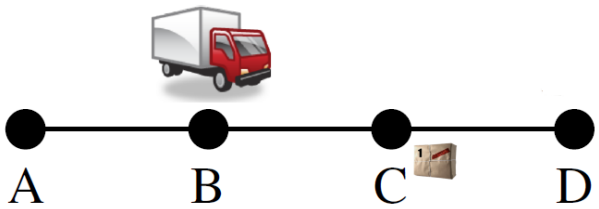
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 1$: $\langle ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



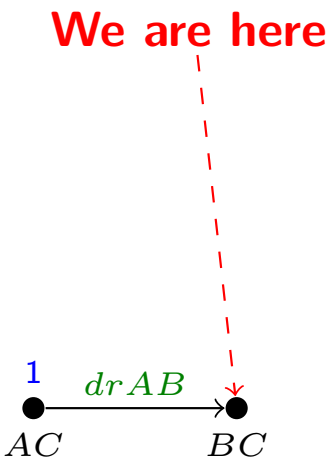
How to Relax During Search: Only-Adds



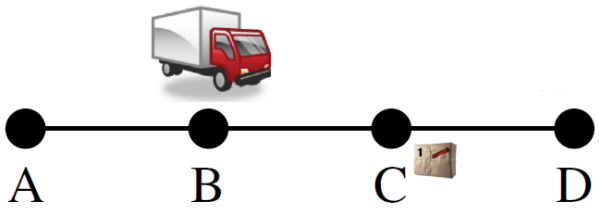
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search:
(tie-breaking: alphabetic)



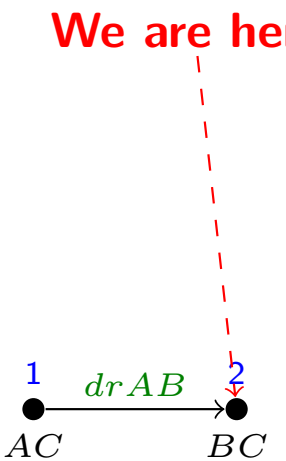
How to Relax During Search: Only-Adds



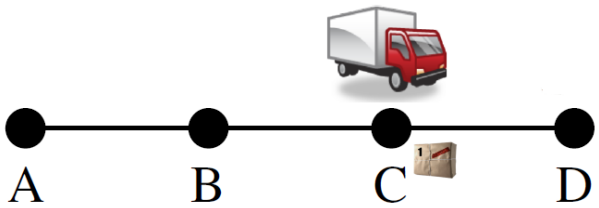
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



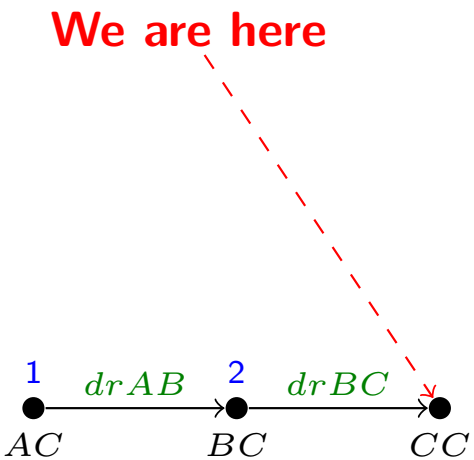
How to Relax During Search: Only-Adds



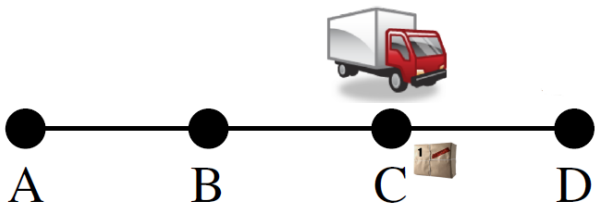
Real problem:

- State s: CC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search:
(tie-breaking: alphabetic)



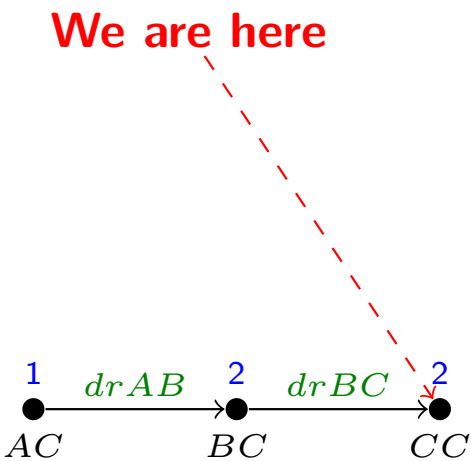
How to Relax During Search: Only-Adds



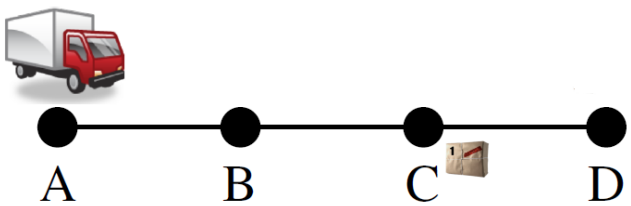
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



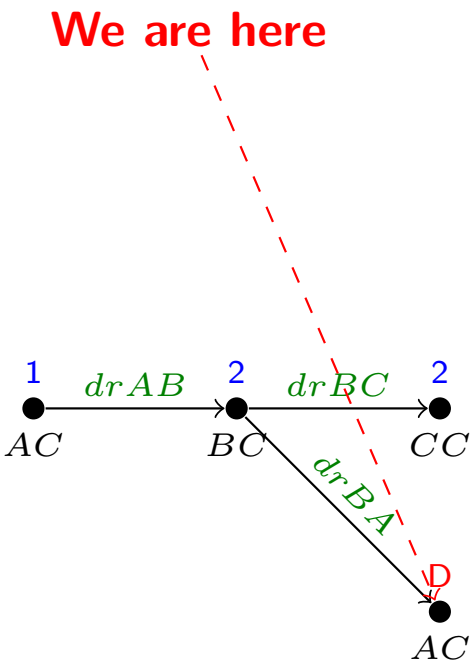
How to Relax During Search: Only-Adds



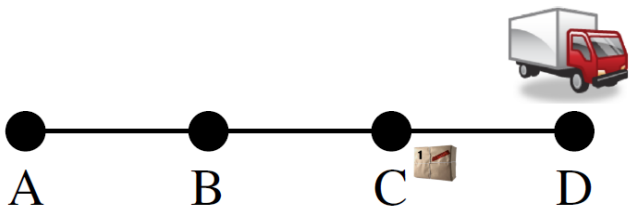
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)



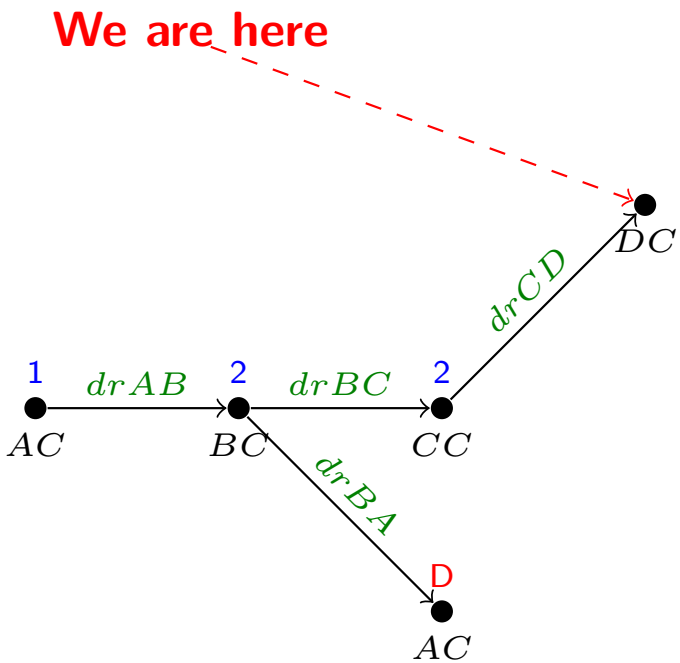
How to Relax During Search: Only-Adds



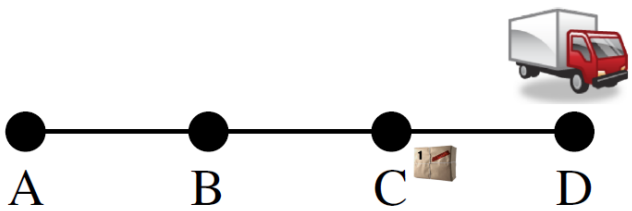
Real problem:

- State s: DC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search:
(tie-breaking: alphabetic)



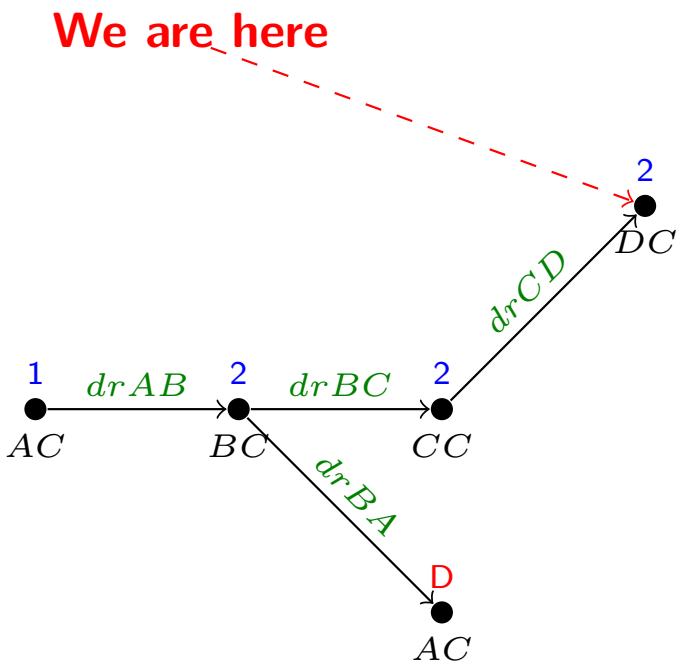
How to Relax During Search: Only-Adds



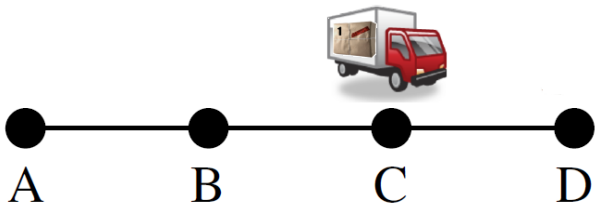
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



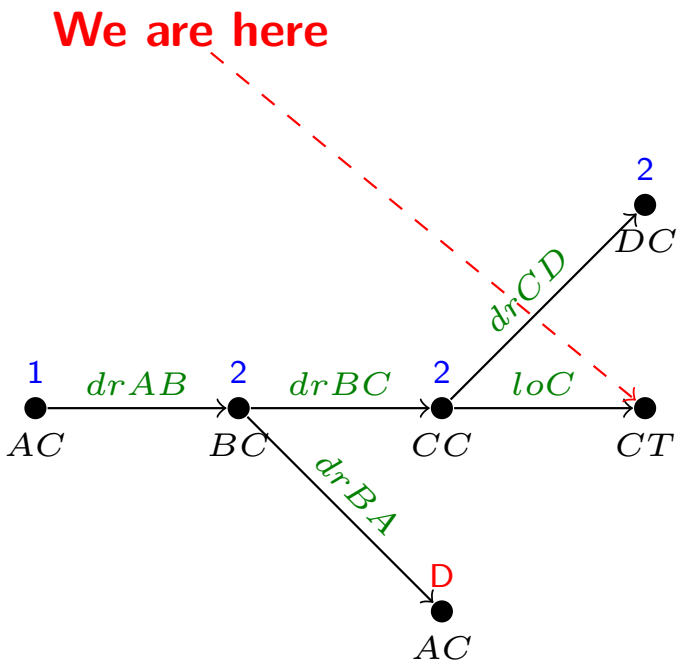
How to Relax During Search: Only-Adds



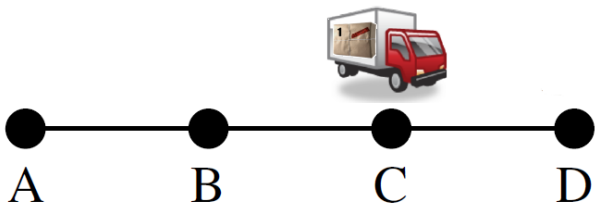
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search:
(tie-breaking: alphabetic)



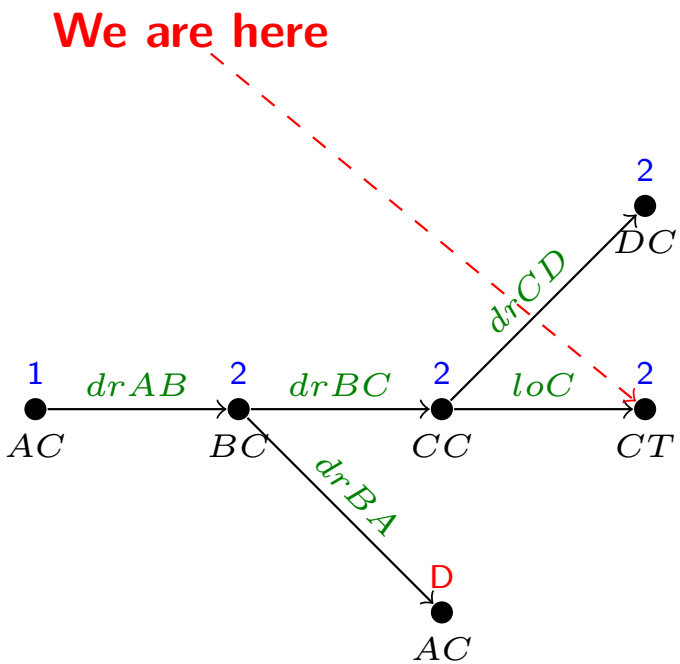
How to Relax During Search: Only-Adds



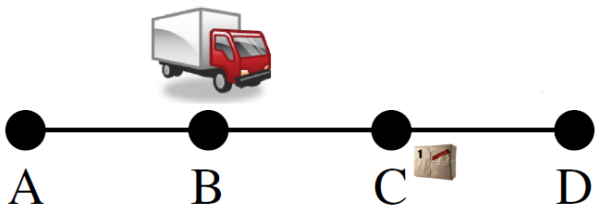
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s)$ =2: $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



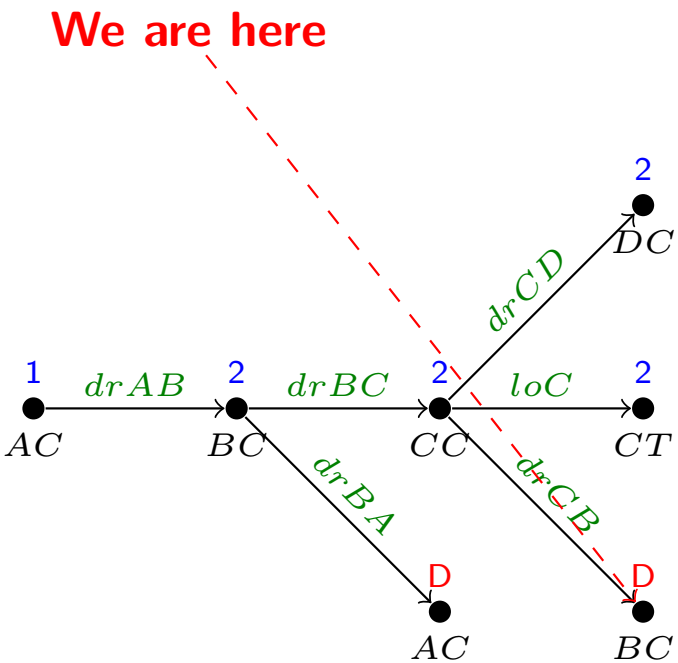
How to Relax During Search: Only-Adds



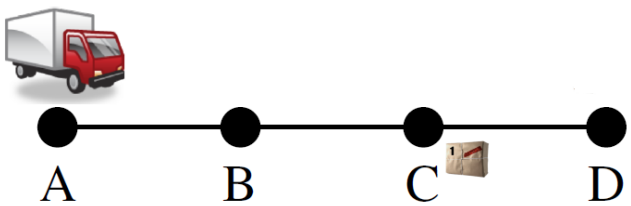
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)



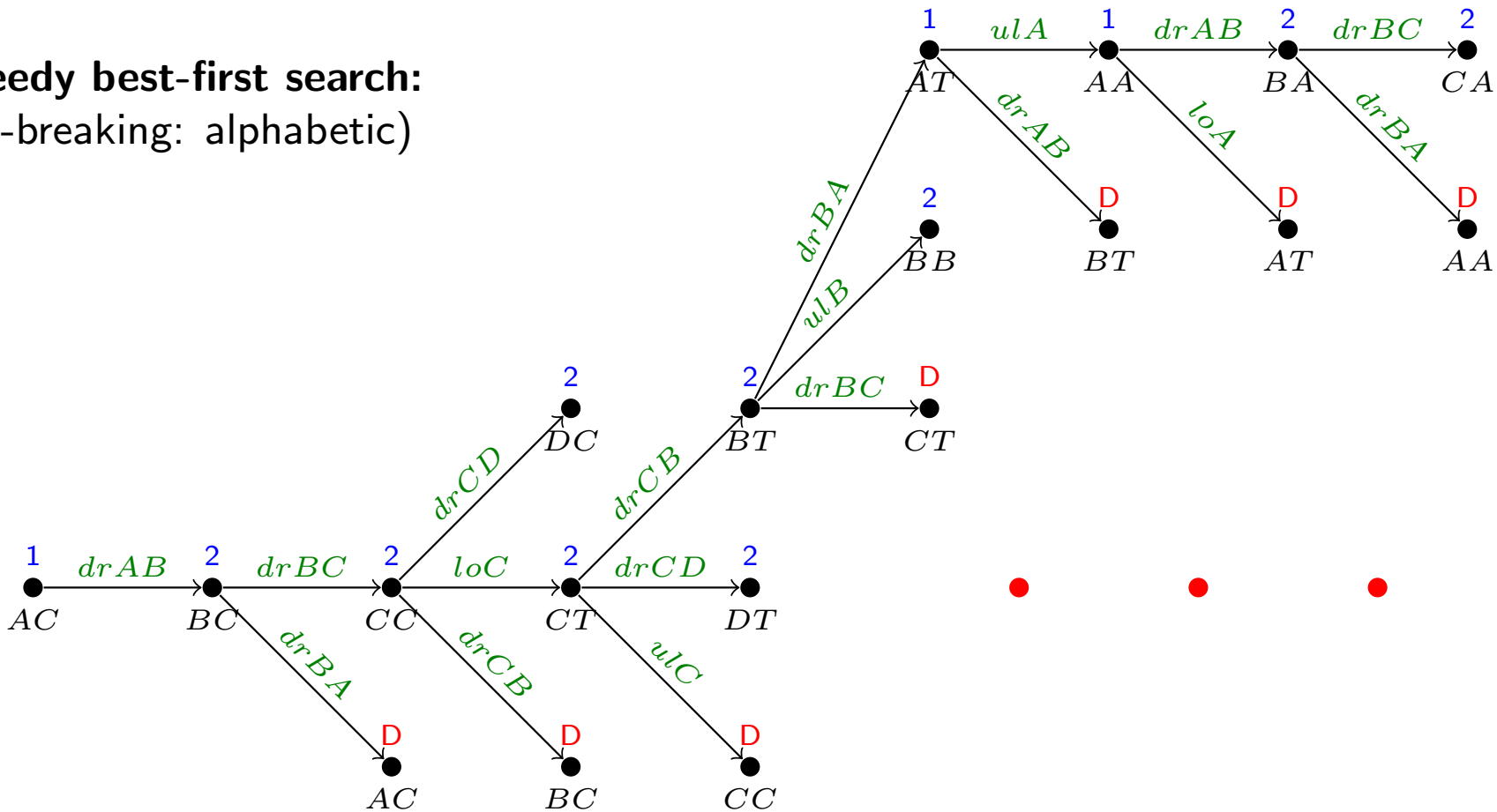
How to Relax During Search: Only-Adds



Real problem:

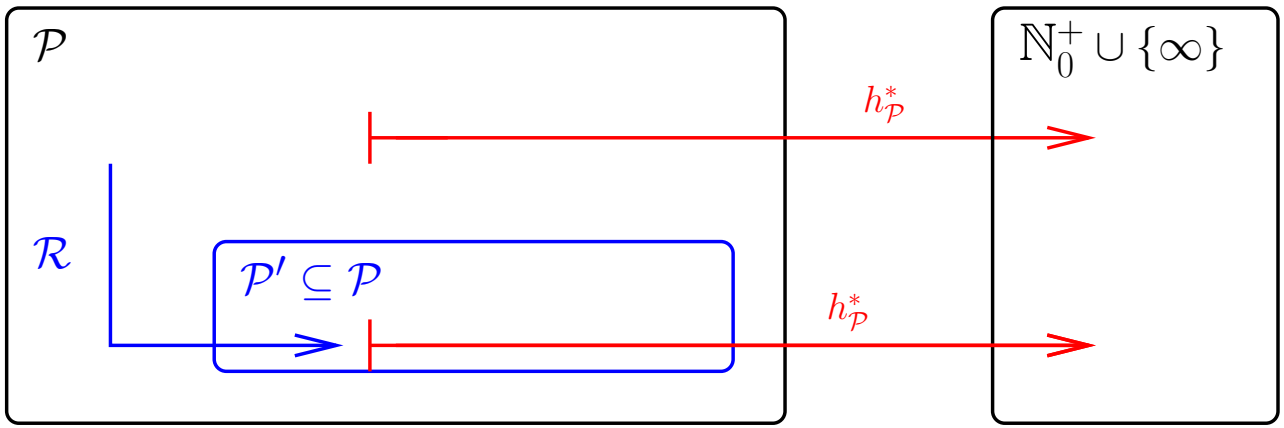
- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



Only-Adds is a "Native" Relaxation

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



- 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?

$\rightarrow$ (A): No: The heuristic value does take into account how many cards are already "home", but it is completely independent of the placement of all the other cards. In particular, dead-end avoidance is essential in Freecell, but the heuristic is unable to detect any dead ends.

$\rightarrow$ (B): No: Typically, it is easy to satisfy many clauses, but then satisfying the remaining ones involves re-doing the entire assignment. (Nevertheless, this heuristic is being used in local search for SAT!)

$\rightarrow$ (C): No: e.g., if a single block A still needs to move elsewhere, but there are 100 blocks on top of A, then the heuristic value is 1.

$\rightarrow$ (D): No! The heuristic remains constantly 1 until we reach the actual goal state.

How the Delete Relaxation Changes the World

Relaxation mapping $\mathcal{R}$ saying that:

"When the world changes, its previous state remains true as well."

Relaxed world: (before)

How the Delete Relaxation Changes the World

Relaxation mapping $\mathcal{R}$ saying that:

“When the world changes, its previous state remains true as well.”

Relaxed world: (after)

The Delete Relaxation

Definition (Delete Relaxation). *Let $\Pi = (P, A, I, G)$ be a planning task. The delete-relaxation of Π 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 $del_{a^+} = \emptyset$.*

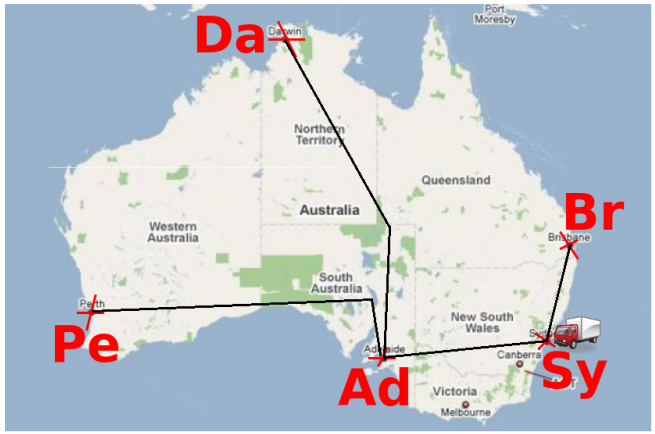
$\rightarrow$ In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS planning tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.

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 $(P, A, s, G)^+$. A relaxed plan for I is called a relaxed plan for Π.*

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

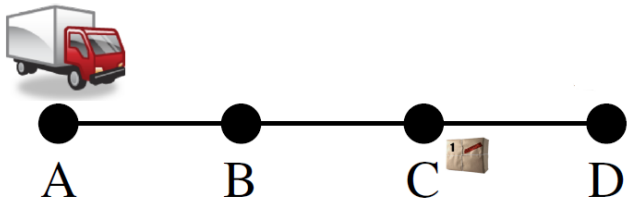
$\rightarrow$ Also called delete-relaxed plan; "relaxation" is often used to mean "delete-relaxation" by default.

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:

$$\langle drive(A, B)^+, drive(B, C)^+, load(C)^+, drive(C, D)^+, unload(D)^+ \rangle$$

$\rightarrow$ We don't need to drive the truck back, because "it is still at A".

Questionnaire

Question!

How does ignoring delete lists simplify Sokoban?

(A): You will never "lock yourself in".

(B): Free positions remain free.

(C): You can walk through walls.

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

→ (A): Yes, because of (B).

→ (B): Yes, when we move a stone into a free space, that space is still free afterwards.

→ (C): No, we dont get any new moves in the relaxation.

→ (D): Only if we give names to the stones. Within the relaxed problem, it may happen that two stones are in the same position, so in principle we can push them both. However, without distinguishing stone names, it is impossible to separate them again, so the two stones in fact become (behave in all relevant ways exactly like) a single stone.

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 Π.*

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

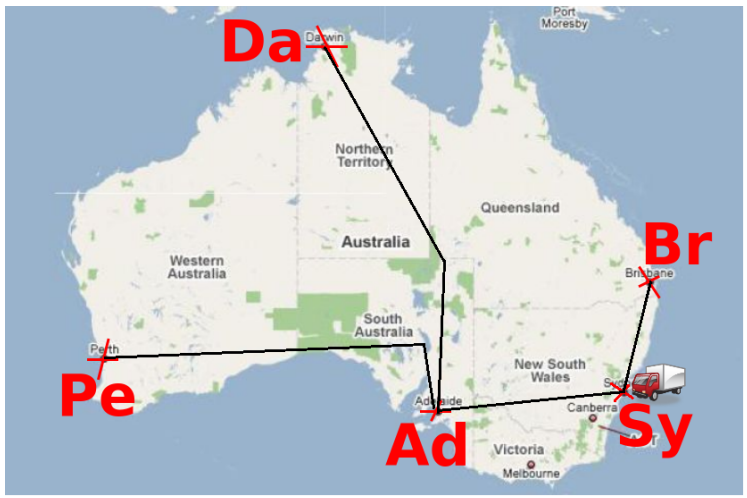
Proposition (PlanEx$^+$ is Easy). *PlanEx$^+$ is a member of* **P**.

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 $|P|$ iterations, and thus runs in polynomial time. Correctness: See slide 30.

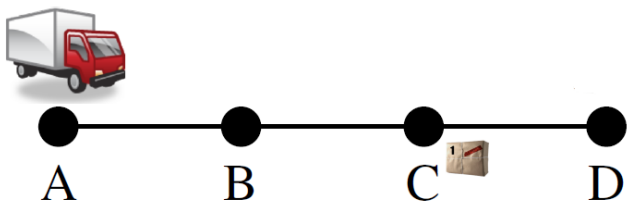
Deciding PlanEx$^+$ in "TSP" in Australia



Iterations on F:

- $\{at(Sydney), visited(Sydney)\}$
- $\cup \{at(Adelaide), visited(Adelaide),\ at(Brisbane), visited(Brisbane)\}$
- $\cup \{at(Darwin), visited(Darwin),\ at(Perth), visited(Perth)\}$

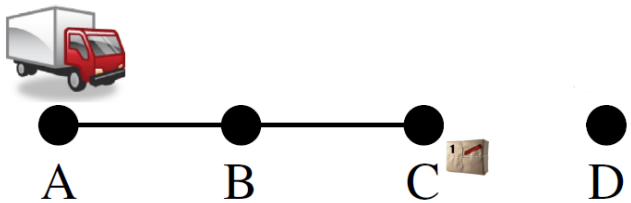
Deciding PlanEx$^+$ in "Logistics"



Iterations on F:

- $\{truck(A), pack(C)\}$
- $\cup \{truck(B)\}$
- $\cup \{truck(C)\}$
- $\cup \{truck(D), pack(T)\}$
- $\cup \{pack(A), pack(B), pack(D)\}$

Deciding PlanEx$^+$ in Unsolvable "Logistics"



Iterations on F:

- $\{truck(A), pack(C)\}$
- $\cup\ \{truck(B)\}$
- $\cup\ \{truck(C)\}$
- $\cup\ \{pack(T)\}$
- $\cup\ \{pack(A), pack(B)\}$
- $\cup\ \emptyset$

PlanEx$^+$ Algorithm: Proof

$\rightarrow$ Show: The algorithm returns "solvable" iff there exists a relaxed plan for Π.

Denote by F_i the content of F after the ith iteration of the while-loop, and denote by A_i the set of actions a where $pre_a \subseteq F_i$.

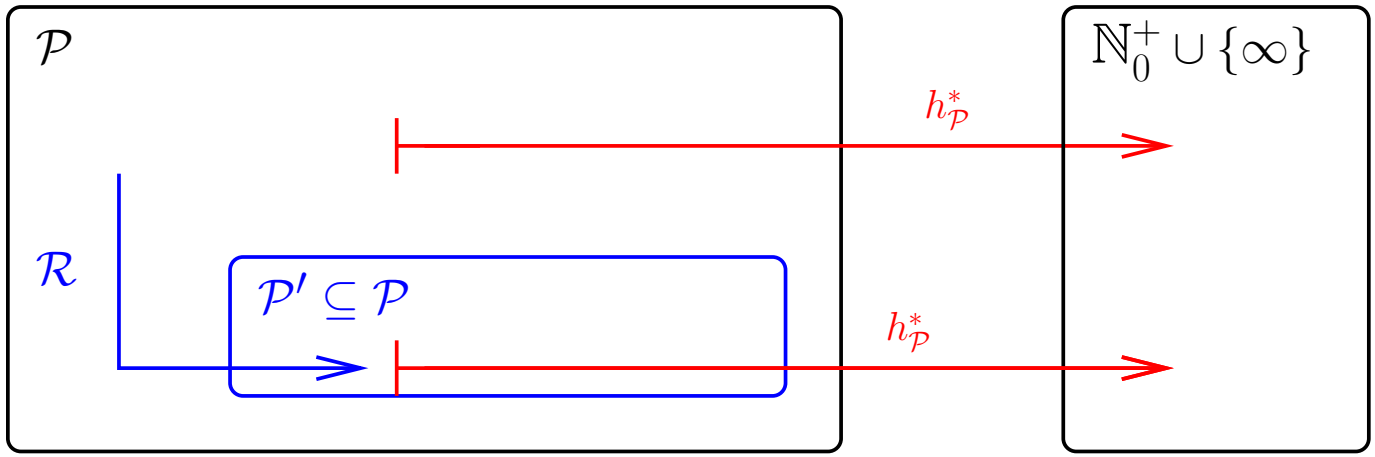
All $a \in A_0$ are applicable in I, all $a \in A_1$ are applicable in $appl(I, A_0^+)$, and so forth. Thus $F_i = appl(I, \langle A_0^+, \ldots, A_{i-1}^+ \rangle)$. (Within each A_j^+, we can sequence the actions in any order.)

Direction "$\Rightarrow$": If "solvable" is returned after iteration n then $G \subseteq F_n = appl(I, \langle A_0^+, \ldots, A_{n-1}^+ \rangle)$ so $\langle A_0^+, \ldots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan which shows the claim.

Direction "$\Leftarrow$": Let $\langle a_0^+, \ldots, a_{n-1}^+ \rangle$ be a relaxed plan, hence $G \subseteq appl(I, \langle a_0^+, \ldots, a_{n-1}^+ \rangle)$. Assume, for the moment, that we drop line (*) from the algorithm. It is then easy to see that $a_i \in A_i$ and $appl(I, \langle a_0^+, \ldots, a_{i-1}^+ \rangle) \subseteq F_i$, for all i. We get $G \subseteq appl(I, \langle a_0^+, \ldots, a_{n-1}^+ \rangle) \subseteq F_n$, and the algorithm returns "solvable" as desired.

Assume to the contrary of the claim that, in an iteration $i < n$, (*) fires. Then $G \not\subseteq F_i$ and $F_i = F_{i+1}$. But, then, $F_i = F_j$ for all $j > i$, and we get $G \not\subseteq F_n$ in contradiction.

Hold on a Sec – Where are we?



- $\mathcal{P}$: 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 Π 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 $\langle a_1, \ldots, a_n \rangle$ is a plan for (P, A, s, G), then $\langle a_1^+, \ldots, a_n^+ \rangle$ is a plan for $(P, A, s, G)^+$.*

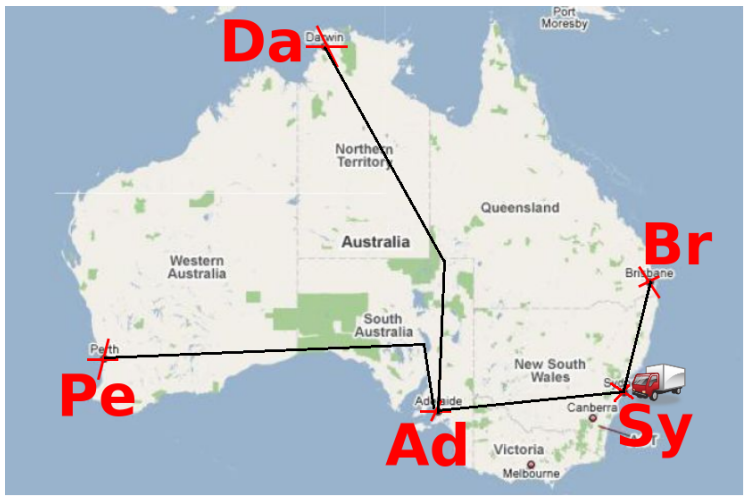
Proof Sketch. Show by induction over $0 \leq i \leq n$ that $appl(s, \langle a_1, \ldots, a_i \rangle) \subseteq appl(s, \langle a_1^+, \ldots, a_i^+ \rangle)$.

$\rightarrow$ "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 $(P, A, s, G)^+$ cannot be longer than that in (P, A, s, G), and the claim follows.

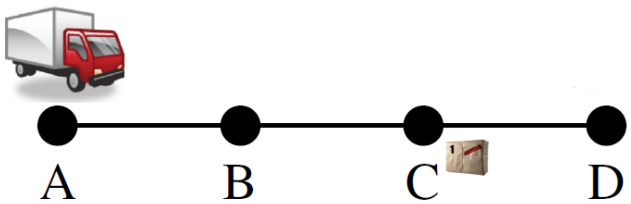
h^+ in "TSP" in Australia



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

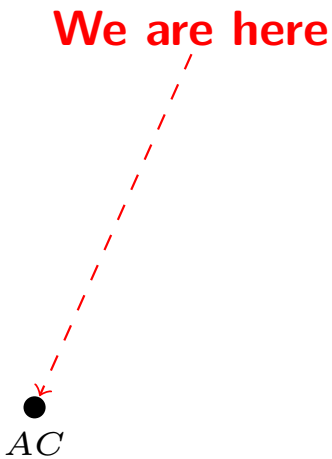
How to Relax During Search: Ignoring Deletes



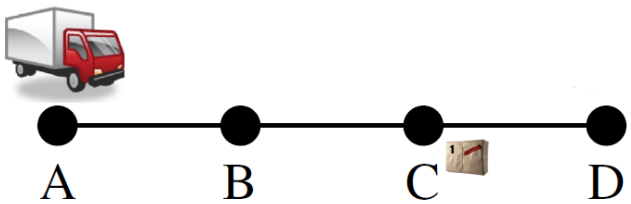
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



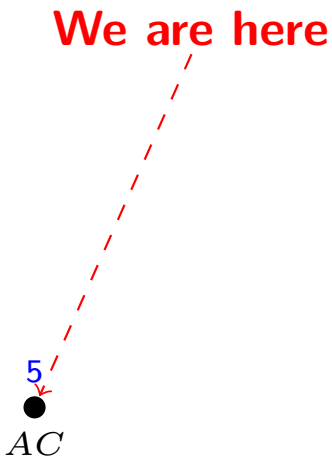
How to Relax During Search: Ignoring Deletes



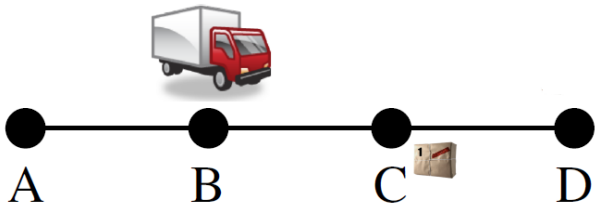
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drAB, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



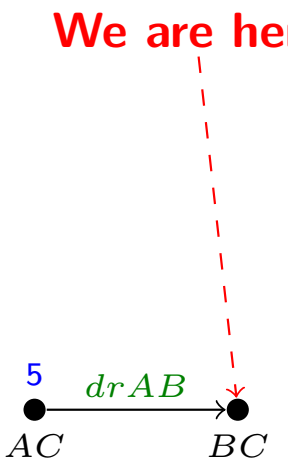
How to Relax During Search: Ignoring Deletes



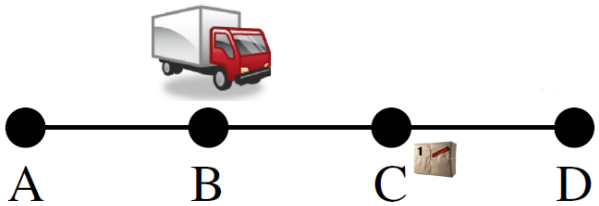
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search:
(tie-breaking: alphabetic)



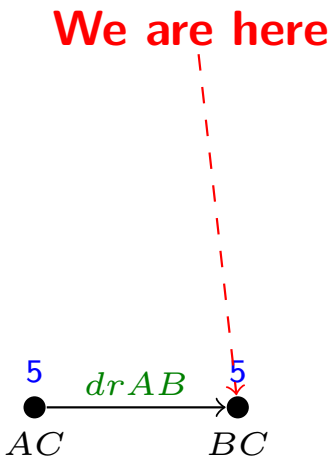
How to Relax During Search: Ignoring Deletes



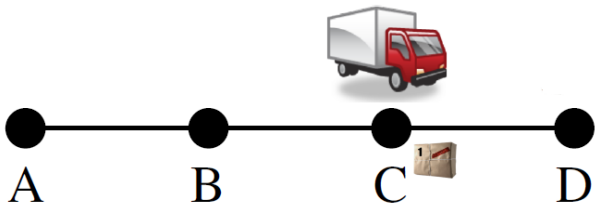
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



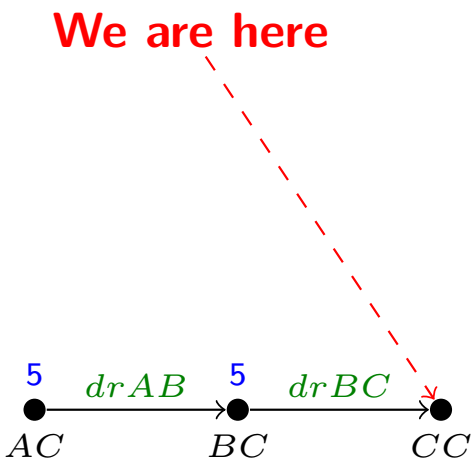
How to Relax During Search: Ignoring Deletes



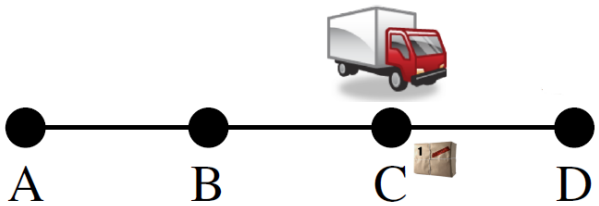
Real problem:

- State s: CC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search:
(tie-breaking: alphabetic)



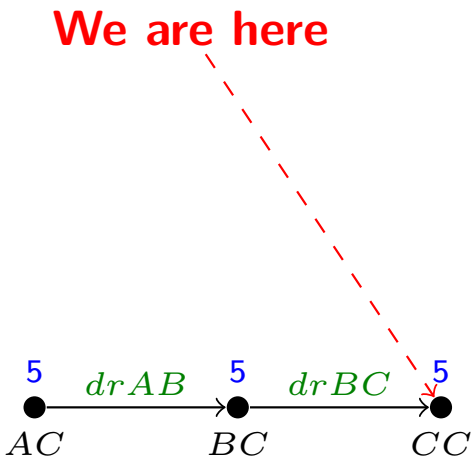
How to Relax During Search: Ignoring Deletes



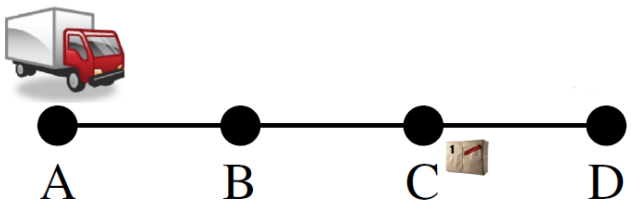
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drCB, drBA, drCD, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



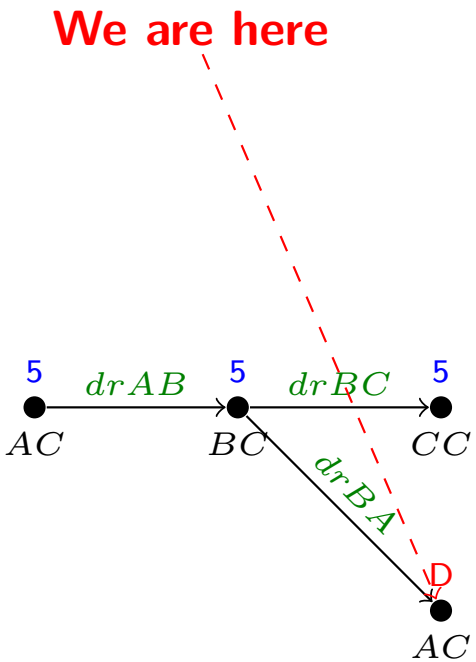
How to Relax During Search: Ignoring Deletes



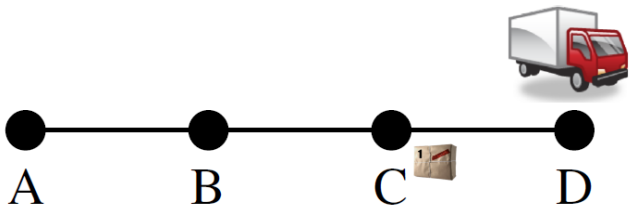
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)



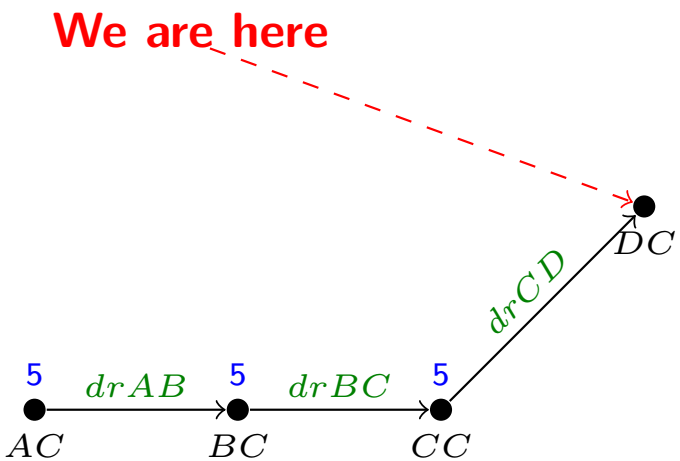
How to Relax During Search: Ignoring Deletes



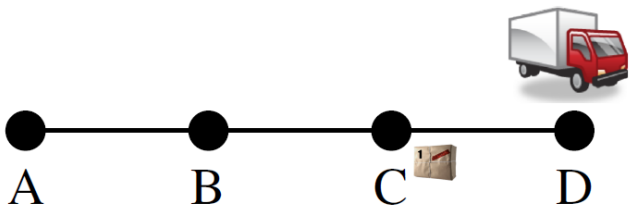
Real problem:

- State s: DC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search:
(tie-breaking: alphabetic)



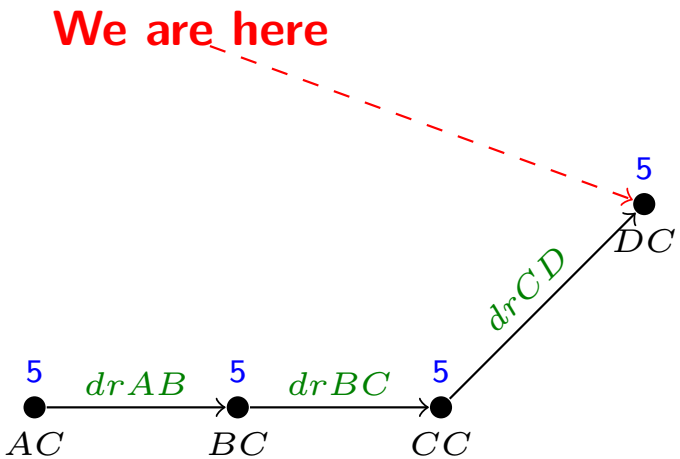
How to Relax During Search: Ignoring Deletes



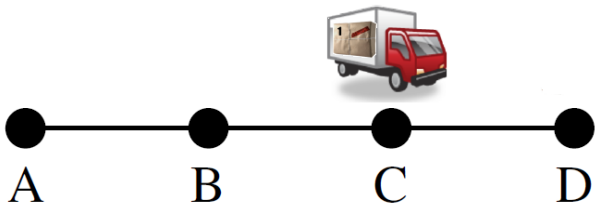
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drDC, drCB, drBA, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



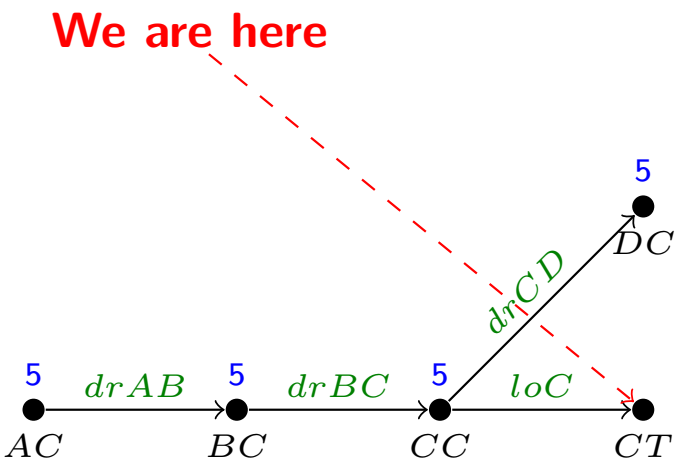
How to Relax During Search: Ignoring Deletes



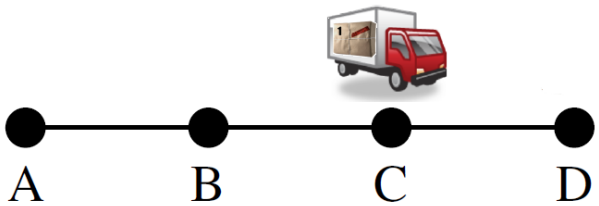
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search:
(tie-breaking: alphabetic)



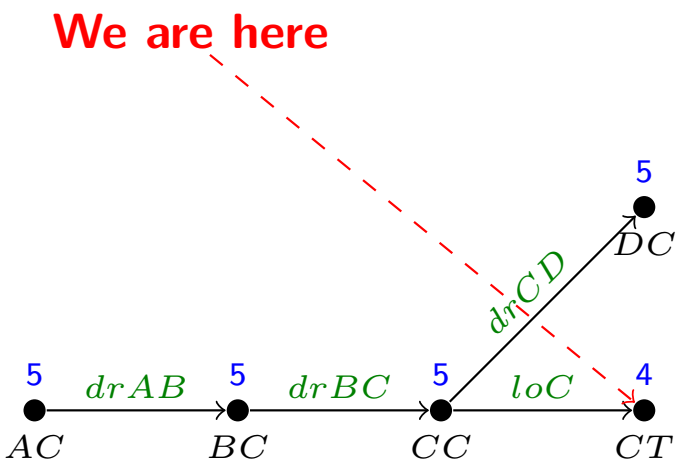
How to Relax During Search: Ignoring Deletes



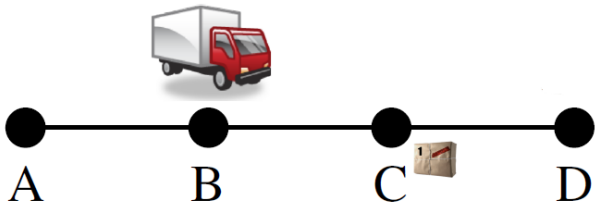
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 4$: e.g. $\langle drCB, drBA, drCD, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



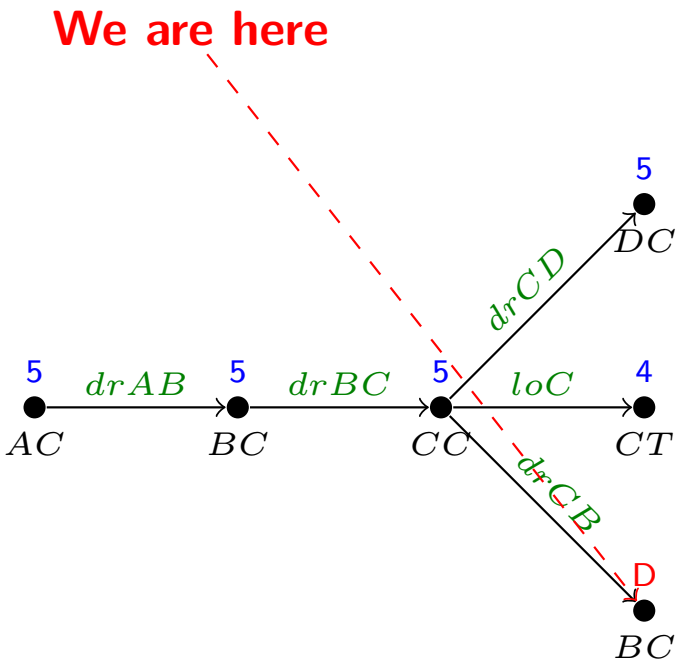
How to Relax During Search: Ignoring Deletes



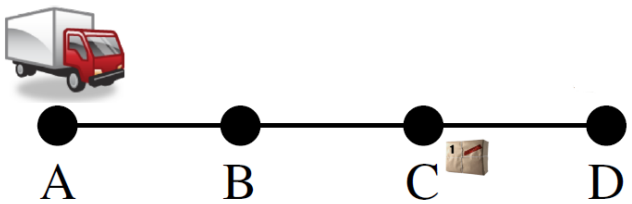
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)



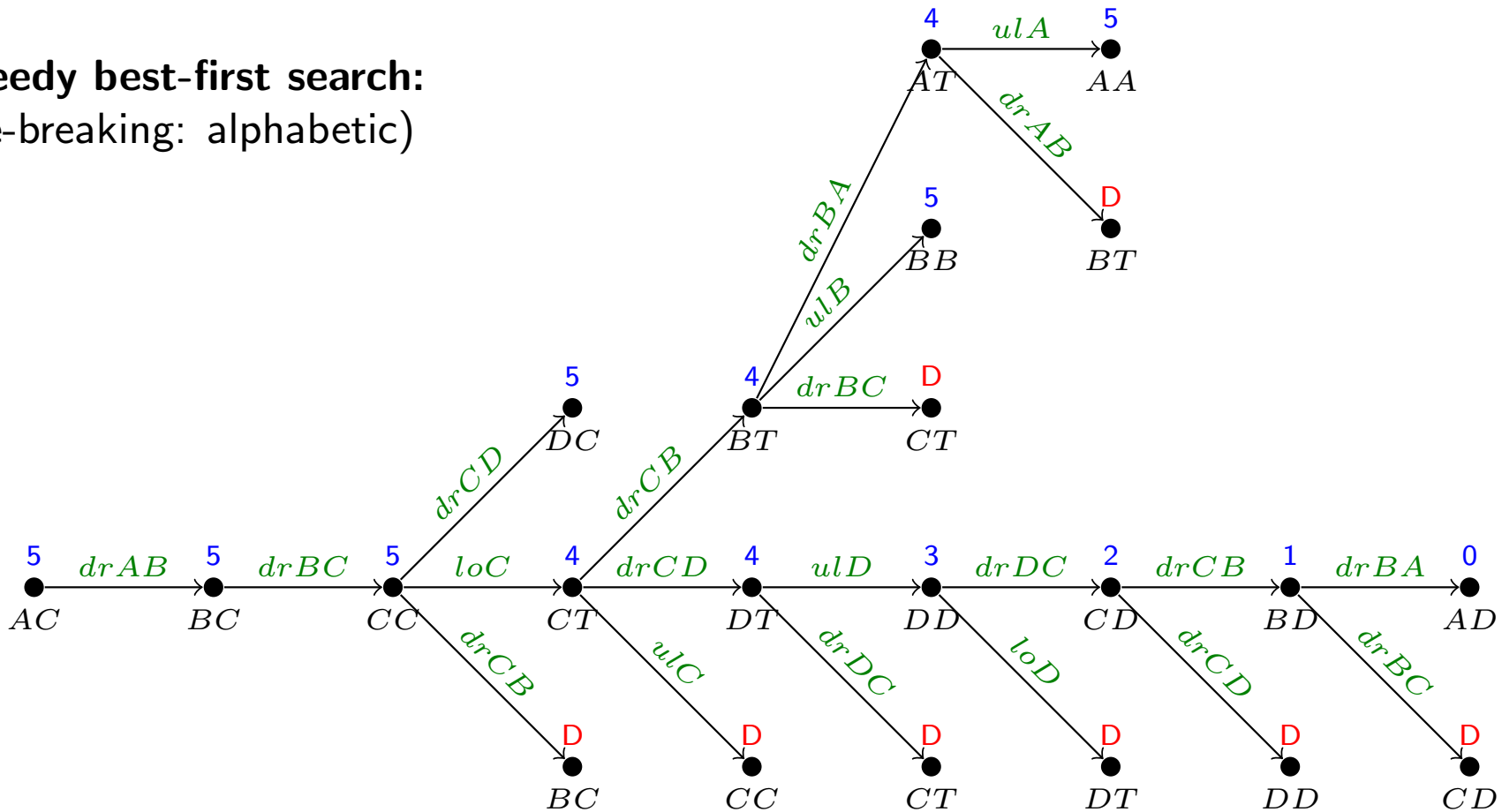
How to Relax During Search: Ignoring Deletes



Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



On the "Accuracy" of h^+

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

→ **Why?**

→ A heuristic function is useful if its estimates are "accurate".

How to measure this?

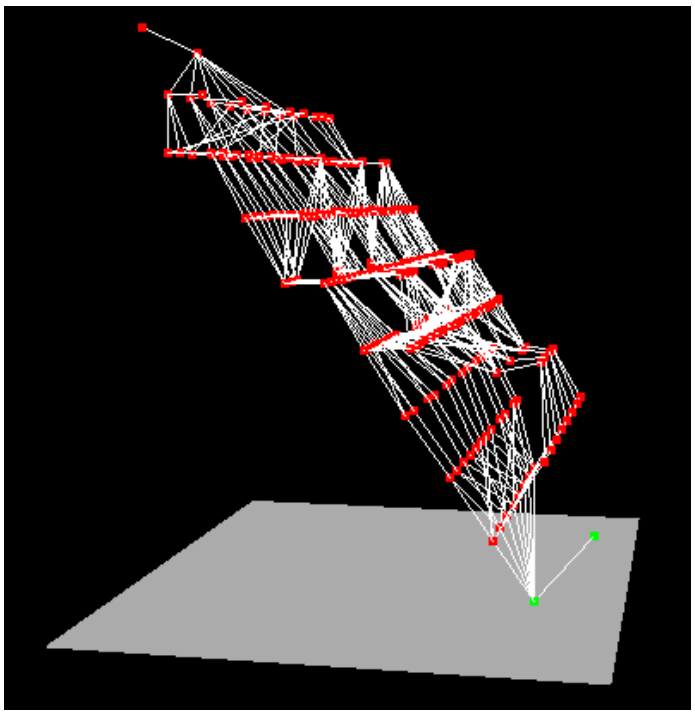
- **Known method 1:** Error relative to h^*, i.e., bounds on $|h^*(s) - h(s)|$.
- **Known method 2:** Properties of the search space surface: Local minima etc.

→ For h^+, method 2 is the road to success:

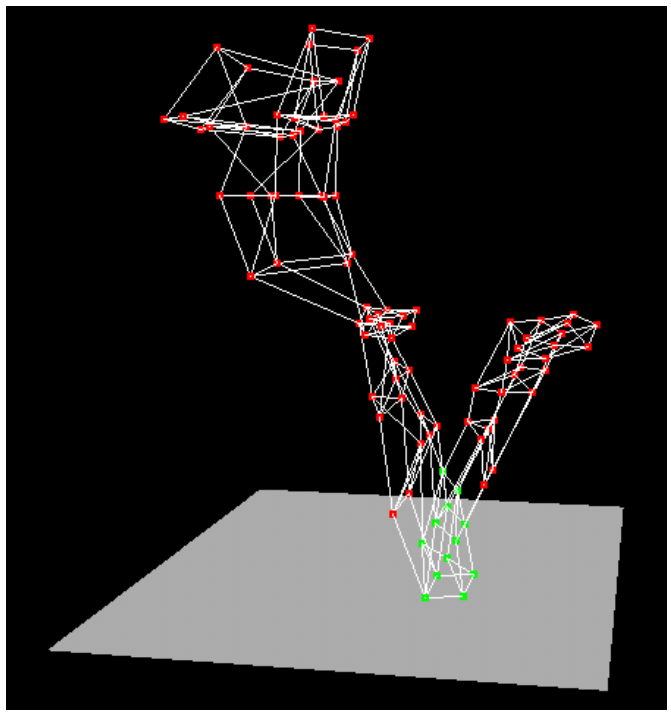
→ In many benchmarks, under h^+, local minima *provably* do not exist! [Hoffmann (2005)]

A Brief Glimpse of h^+ Search Space Surfaces

$\rightarrow$ Graphs = state spaces, vertical height = h^+:

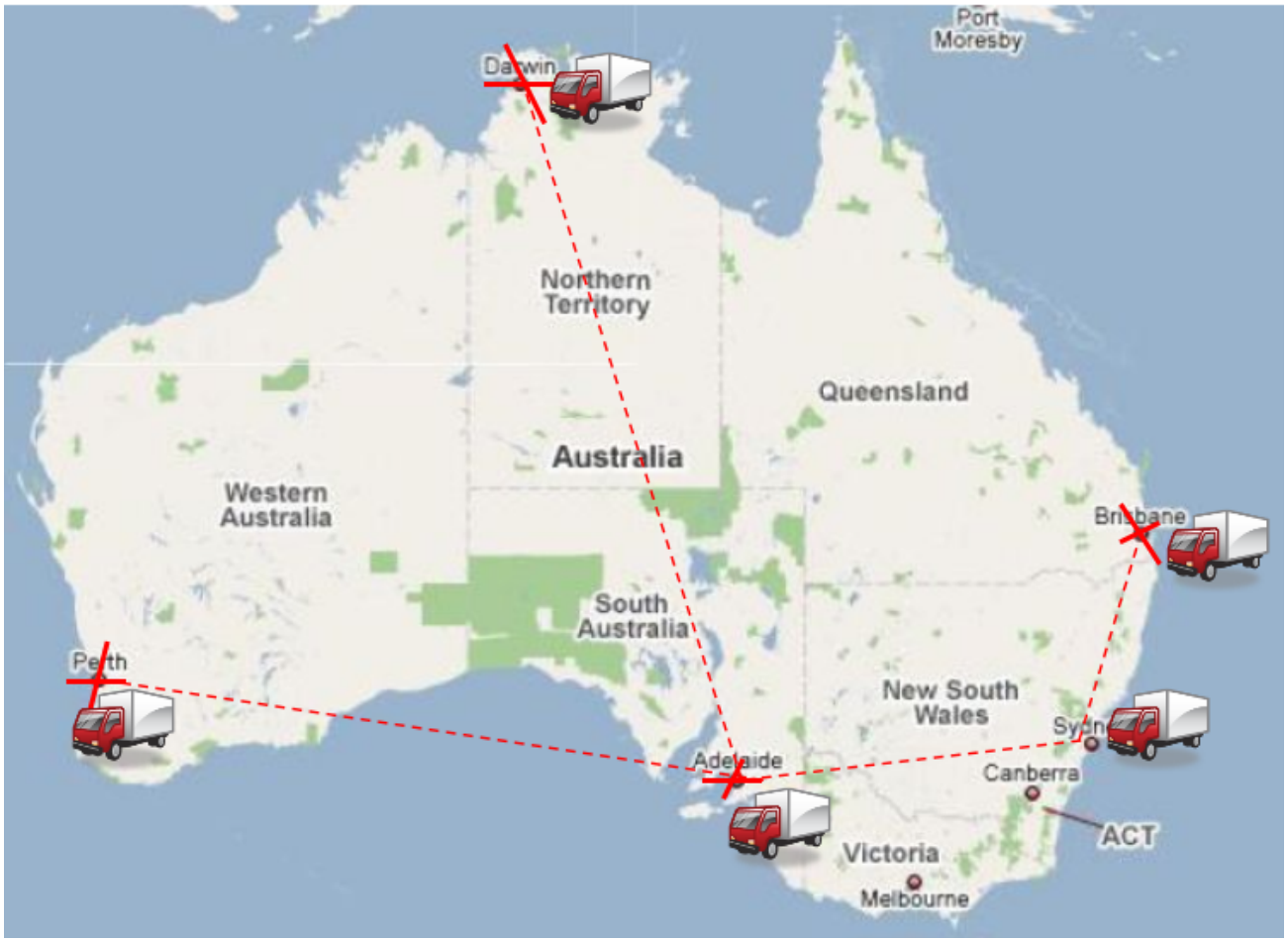


"Gripper"



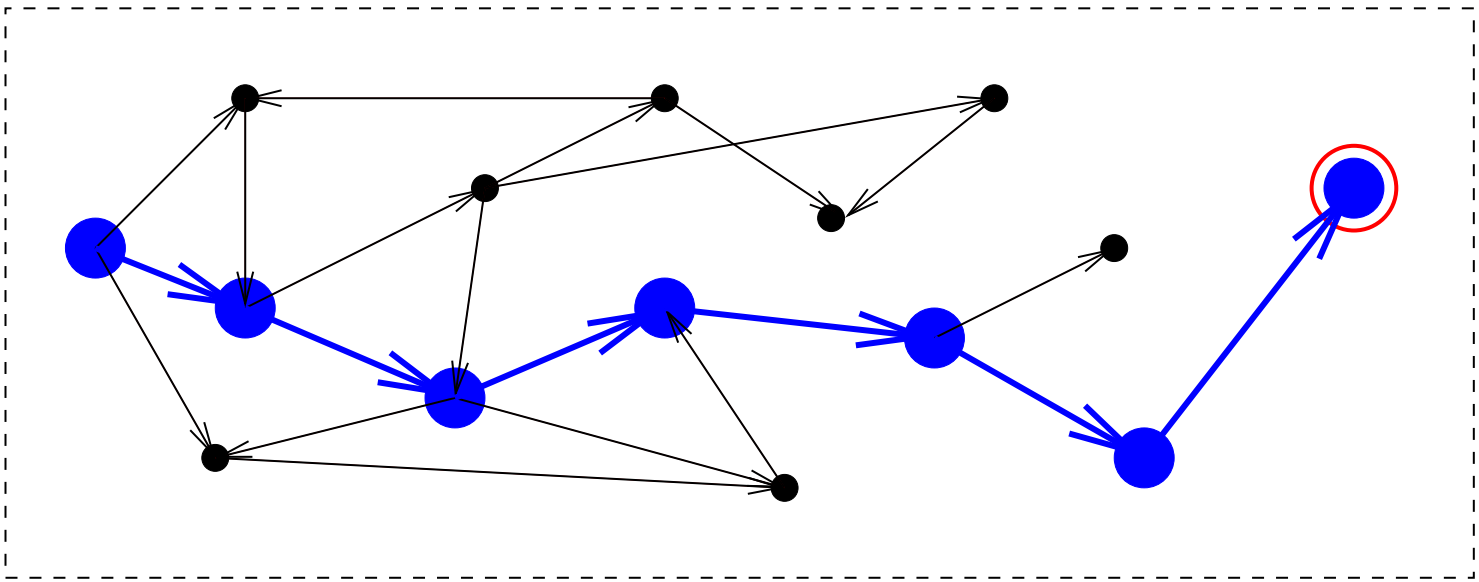
"Logistics"

h^+ in (the Real) TSP



$\rightarrow h^+ =$ **Minimum Spanning Tree**

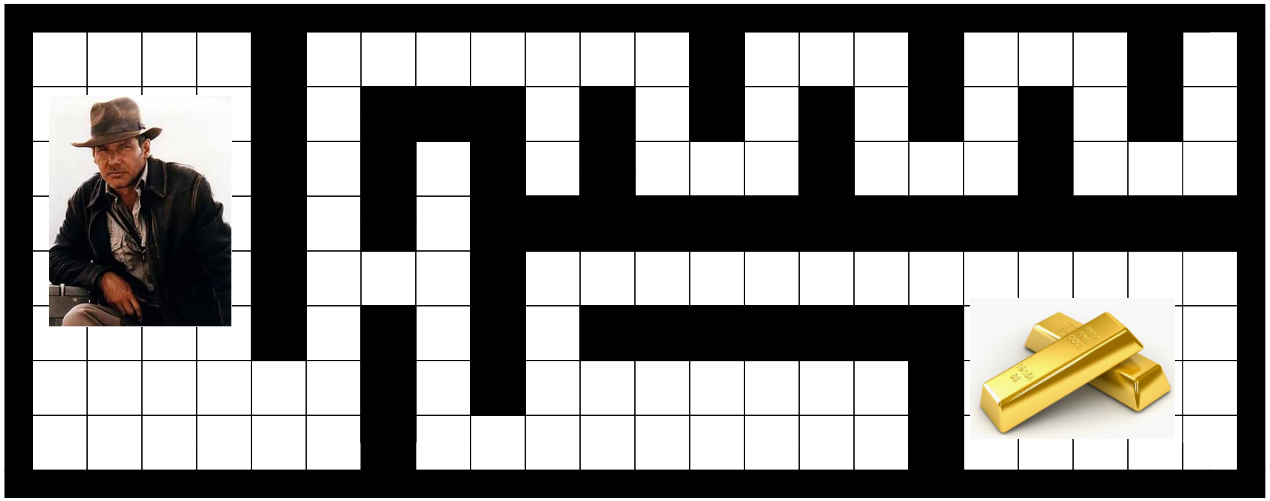
h^+ in Graphs



$h^+(\text{Graph-Distance}) = \text{real distance}$

(shortest paths never "walk back")

Questionnaire



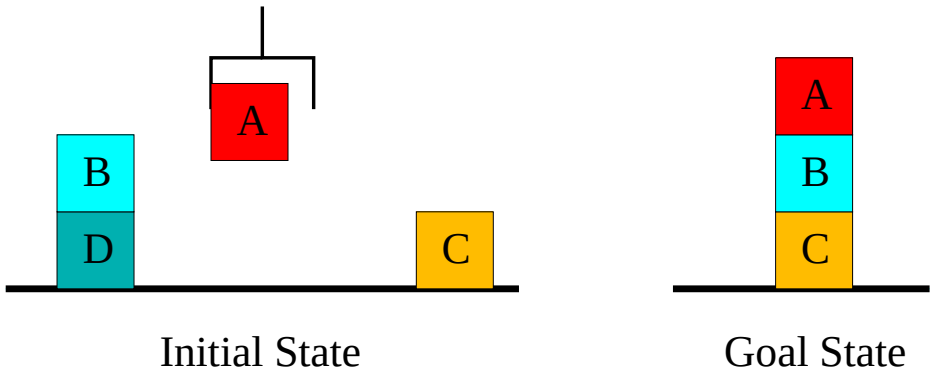
Question!

In this domain, h^+ is equal to?

(A): Manhattan Distance.
(B): Horizontal distance.
(C): Vertical distance.
(D): h^*.

$\rightarrow$ (A): No, relaxed plans can't walk through walls. (B), (C): No, relaxed plans must move both horizontally and vertically. (D): Yes, optimal plan = shortest path = optimal relaxed plan (cf. previous slide).

h^+ in the Blocksworld



- Optimal plan: $\langle putdown(A),\ unstack(B,D),\ stack(B,C),\ pickup(A),\ stack(A,B)\rangle$.
- Optimal relaxed plan: $\langle stack(A,B),\ unstack(B,D),\ stack(B,C)\rangle$.

Observe: What can we say about the "search space surface" at the initial state here? The initial state lies on a local minimum under h^+, together with the successor state s where we stacked A onto B. All direct other neighbors of these two states have a strictly higher h^+ value.

How to Compute h^+?

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

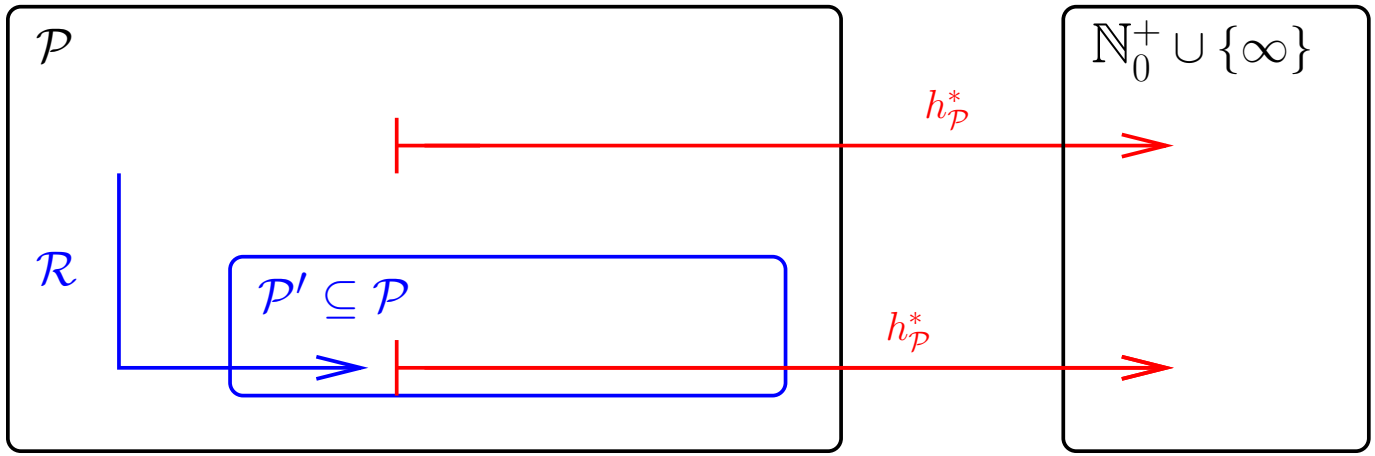
$\rightarrow$ By computing h^+, we solve PlanLen$^+$.

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

Proof. Membership: Easy. Hardness: 1. Trivial from our prior results, because this generalizes optimal planning under the Only-Adds relaxation, cf. **Chapter 12**. 2. However, hardness of optimal Only-Adds comes from hardness of Minimum Cover, which is easy for sets (add lists) of size ≤ 2. One can prove by reduction from SAT that PlanLen$^+$ remains hard even with small add lists. Construction outline, example $\{C_1 = \{A\}, C_2 = \{\neg A\}\}$:

- Action "$SetX_i true$" for every variable X_i: pre empty, add $\{Atrue, Aset\}$.
- Action "$SetX_i false$" for every variable X_i: pre empty, add $\{Afalse, Aset\}$.
- Action "$SatisfyC_j$" for every clause C_j: pre $Atrue$, add $C_1 sat$; pre $Afalse$, add $C_2 sat$.
- Goal "$X_i set$" for all variables X_i, "$C_j sat$" for all clauses C_j: $Aset, C_1 sat, C_2 sat$.
- Length bound $B :=$ number of variables $+$ number of clauses ($= 3$ here).

Hold on a Sec – Where are we?



- $\mathcal{P}$: 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^{FF}

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

Notes:

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

Observe: h^{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^{FF}:

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

Computing h^{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

→ Does this look familiar to you? Could. It's the same algorithm we used to decide PlanEx$^+$ (slide 26).

"Logistics" example: Blackboard (similar to slide 28).

Are we done? cf. slide 30: "$\langle A_0^+, \ldots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan". Could use this as the basis of h^{FF}.

→ But this would overestimate vastly!

In "Logistics", $\sum_{i=0}^{n-1} |A_i| = 11 > 8 = h^*$. And now imagine there are 100 packages only one of which needs to be transported ...

Computing h^{FF}: Extracting a Relaxed Plan

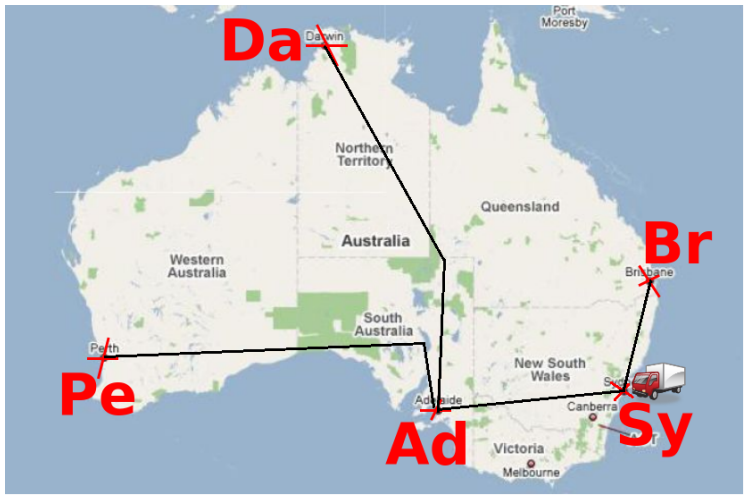
Information from the RPG: ($\min$ over an empty set is ∞)

- 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 $t := 0, \ldots, M$ **do**
    $G_t := \{g \in G \mid level(g) = t\}$
endfor
for $t := M, \ldots, 1$ **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.

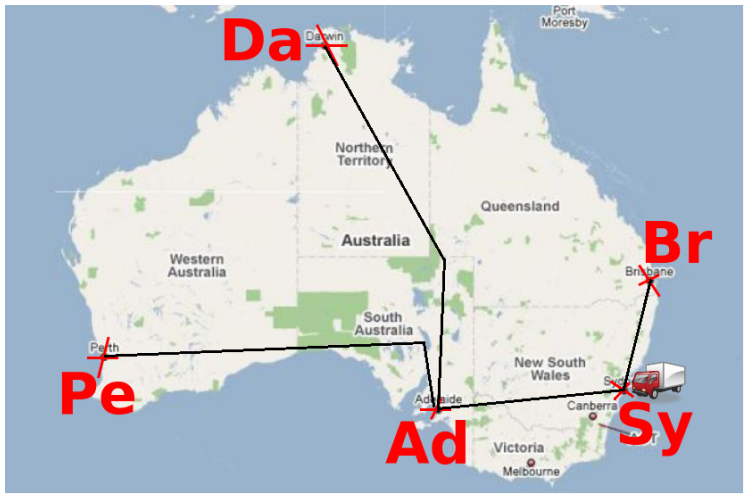
Computing h^{FF} in "TSP" in Australia



RPG:

- $F_0 = \{at(Sydney),\ visited(Sydney)\}$.
- $A_0 = \{drive(Sydney, Adelaide),\ drive(Sydney, Brisbane)\}$.
- $F_1 = F_0 \cup \{at(Adelaide),\ visited(Adelaide),\ at(Brisbane),\ visited(Brisbane)\}$.
- $A_1 = A_0 \cup \{drive(Adelaide, Darwin),\ drive(Adelaide, Perth),$
  $drive(Adelaide, Sydney),\ drive(Brisbane, Sydney)\}$.
- $F_2 = F_1 \cup \{at(Darwin),\ visited(Darwin),\ at(Perth),\ visited(Perth)\}$.

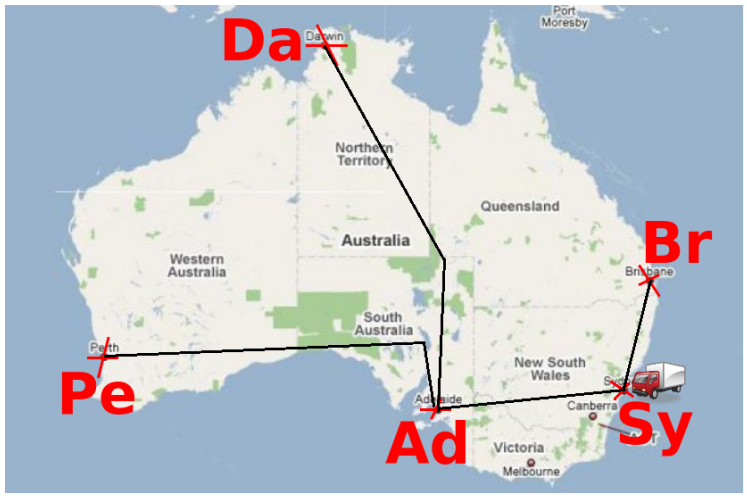
Computing h^{FF} in "TSP" in Australia



Inserting the goals:

- F_0: $at(Sydney)$, $visited(Sydney)$.
- A_0: $drive(Sydney, Adelaide)$, $drive(Sydney, Brisbane)$.
- F_1: $at(Adelaide)$, $visited(Adelaide)$, $at(Brisbane)$, $visited(Brisbane)$.
- A_1: $drive(Adelaide, Darwin)$, $drive(Adelaide, Perth)$, $drive(Adelaide, Sydney)$, $drive(Brisbane, Sydney)$.
- F_2: $at(Darwin)$, $visited(Darwin)$, $at(Perth)$, $visited(Perth)$.

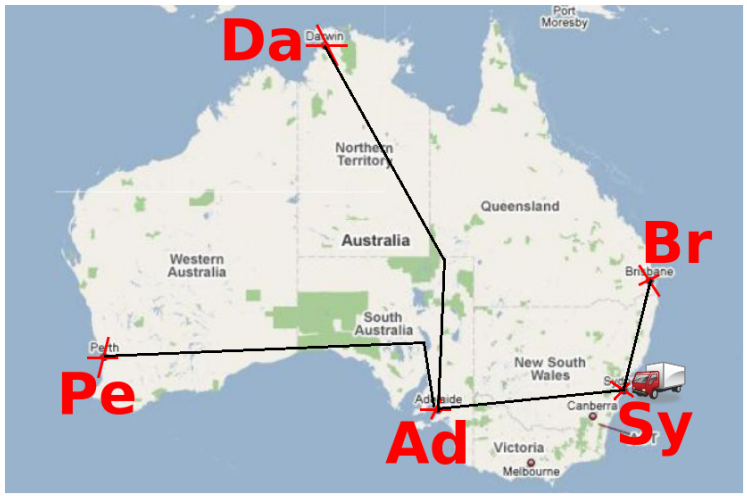
Computing h^{FF} in "TSP" in Australia



Supporting the goals at $t = 2$:

- F_0: $at(Sydney)$, $visited(Sydney)$.
- A_0: $drive(Sydney, Adelaide)$, $drive(Sydney, Brisbane)$.
- F_1: $at(Adelaide)$, $visited(Adelaide)$, $at(Brisbane)$, $visited(Brisbane)$.
- A_1: $drive(Adelaide, Darwin)$, $drive(Adelaide, Perth)$, $drive(Adelaide, Sydney)$, $drive(Brisbane, Sydney)$.
- F_2: $at(Darwin)$, $visited(Darwin)$, $at(Perth)$, $visited(Perth)$.

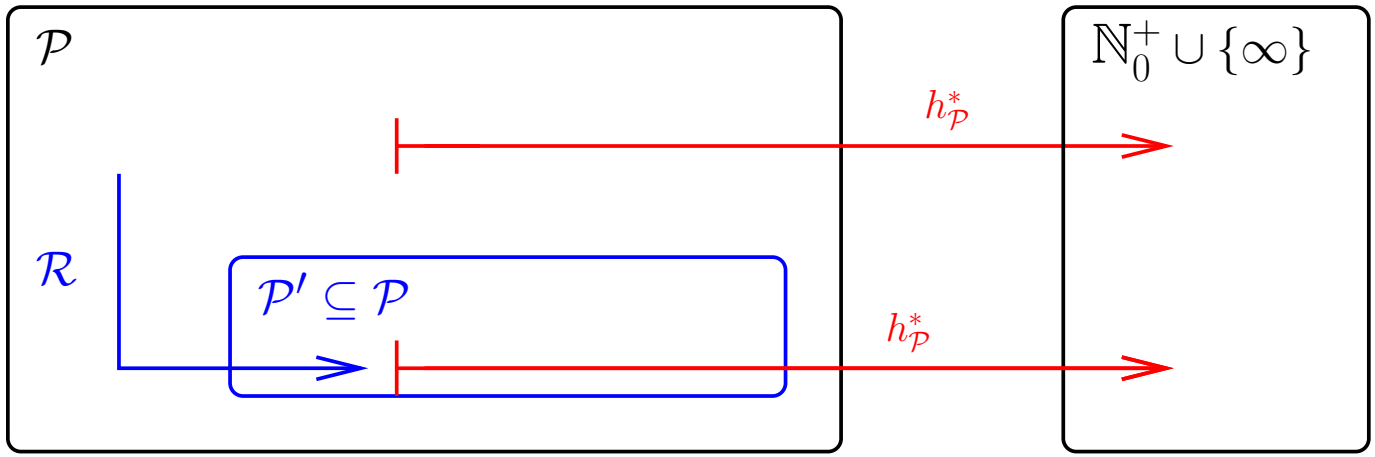
Computing h^{FF} in "TSP" in Australia



Supporting the goals at $t = 1$:

- F_0: $at(Sydney)$, $visited(Sydney)$.
- A_0: $drive(Sydney, Adelaide)$, $drive(Sydney, Brisbane)$.
- F_1: $at(Adelaide)$, $visited(Adelaide)$, $at(Brisbane)$, $visited(Brisbane)$.
- A_1: $drive(Adelaide, Darwin)$, $drive(Adelaide, Perth)$, $drive(Adelaide, Sydney)$, $drive(Brisbane, Sydney)$.
- F_2: $at(Darwin)$, $visited(Darwin)$, $at(Perth)$, $visited(Perth)$.

How Does it All Fit Together?



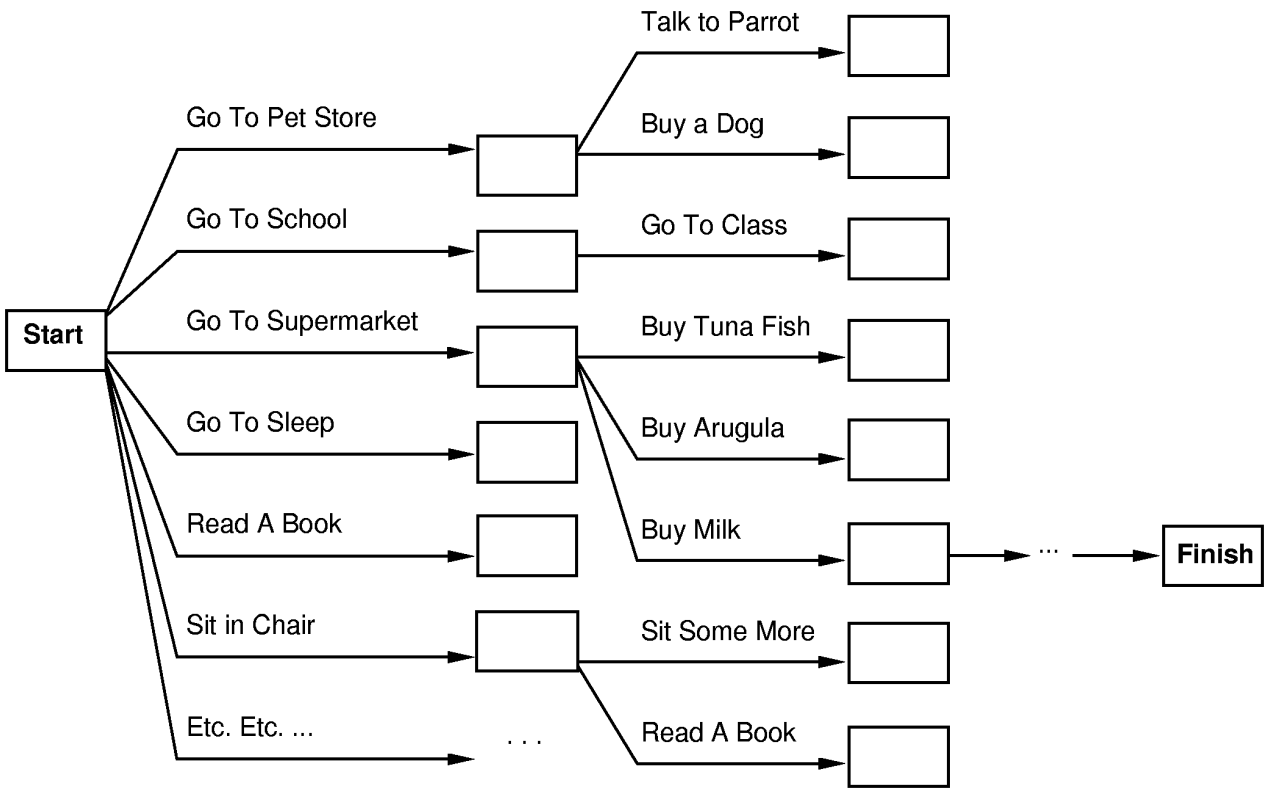
$\mathcal{P}$: STRIPS planning tasks. $h^*_{\mathcal{P}}$: Length h^* of a shortest plan. $\mathcal{P}'$: STRIPS planning tasks with empty delete lists. $\mathcal{R}$: Drop the delete lists. $h^* \circ \mathcal{R}$: Length h^+ of a shortest relaxed plan.

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

$\rightarrow$ h^+ is admissible; h^{FF} is not.

Helpful Actions Pruning

Idea: In search, expand only those actions contained in the relaxed plan.

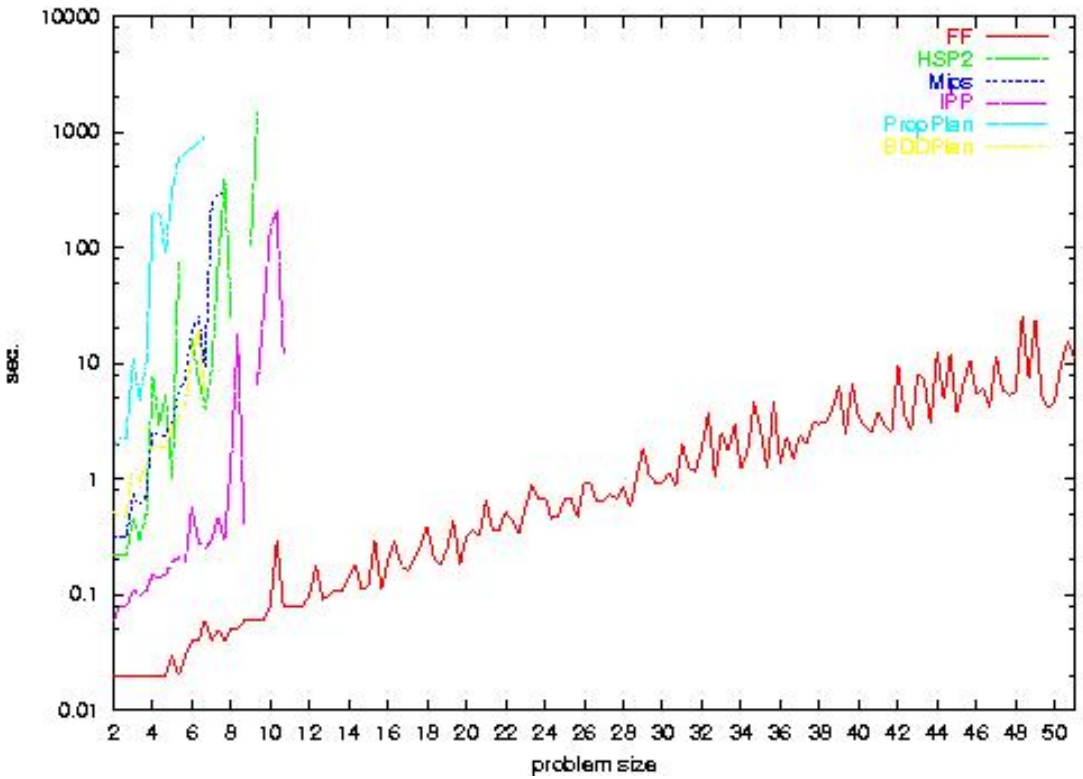


$\rightarrow$ Relaxed plan = "Go To Supermarket, Buy Milk, . . . "

$\rightarrow$ Absolutely essential, used in all state-of-the-art satisficing planners.

Helpful Actions Pruning

Idea: In search, expand only those actions contained in the relaxed plan.



(Schedule domain:
many tools,
many objects.)

→ Relaxed plan does *not* drill holes into objects that need to be painted.

→ Absolutely essential, used in all state-of-the-art satisficing planners.

For Reference: Other Approximations of h^+

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

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

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

h^{add}: Use instead the *sum* of the costs of the single facts $p \in g$

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

$\rightarrow$ Not admissible, and prone to over-estimation; h^{FF} works better.

Good lower bounds on h^+: (cf. next section)

- Admissible landmarks [Karpas and Domshlak (2009)]
- LM-cut [Helmert and Domshlak (2009)].

Questionnaire

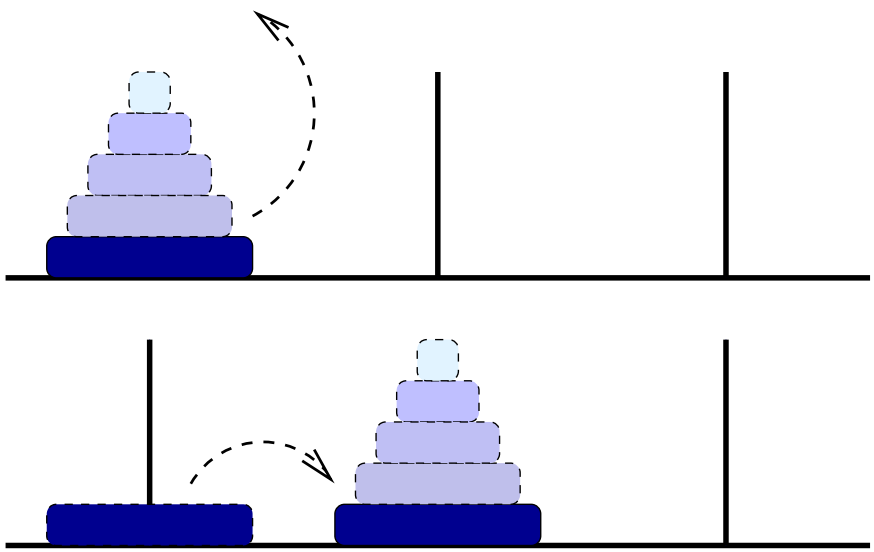
Question!

In the initial state of the Towers of Hanoi task with 5 discs, what is the value of h^+?

(A): 1 (B): 2

(C): 5 (D): 32

$\rightarrow$ (C): The discs always "remain stacked", so we can just clear the bottom disc and move it over. For n discs, this takes $h^+(I) = n$ steps.



Before We Begin

Challenge: Given a planning task Π, simplify Π to obtain a relaxed planning task Π', then solve Π' 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 a snapshot of this research 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.)

4 of the 5 Families We Know Up to Now

Ignoring Deletes

h^+

Abstractions

PDB

M&S

Critical Paths

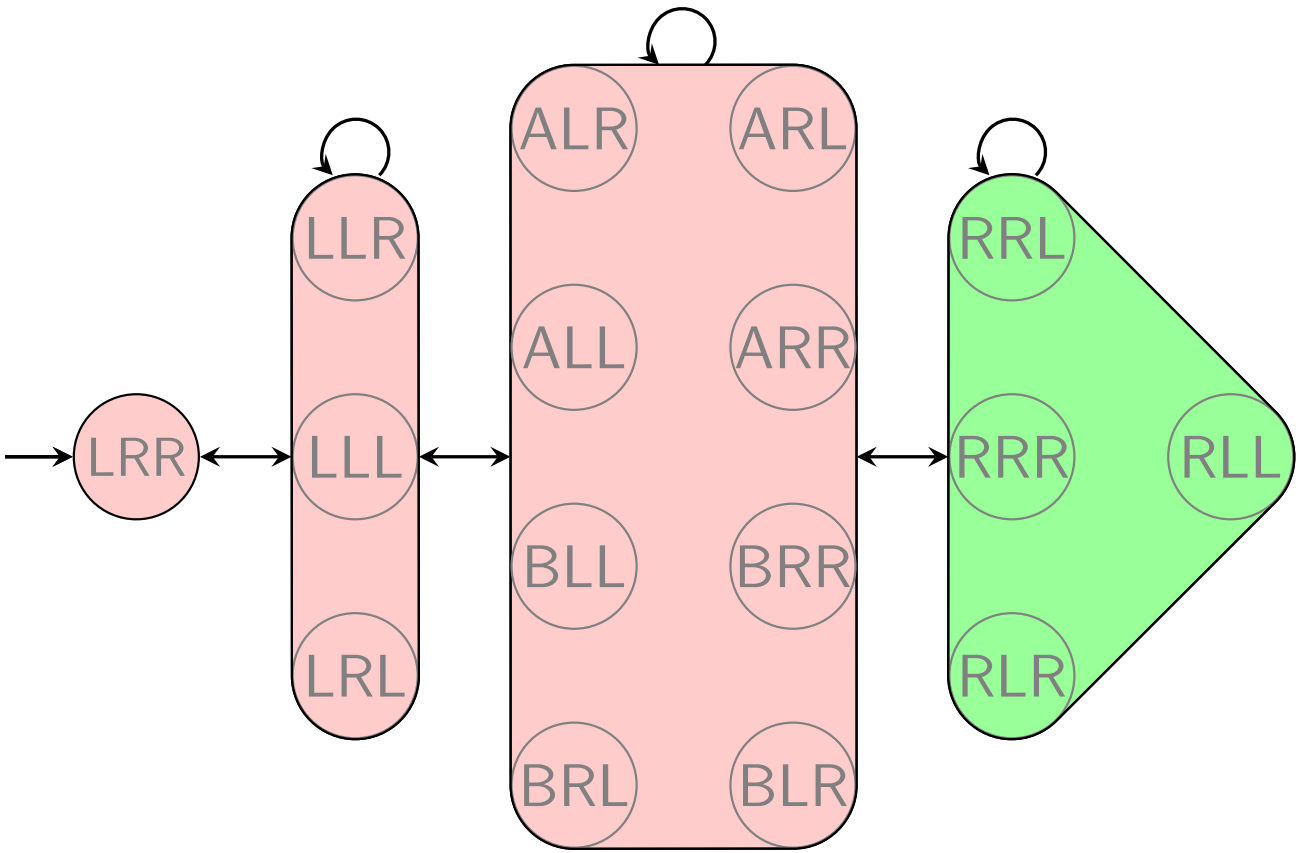
h^1
h^2
h^3
...

Landmarks

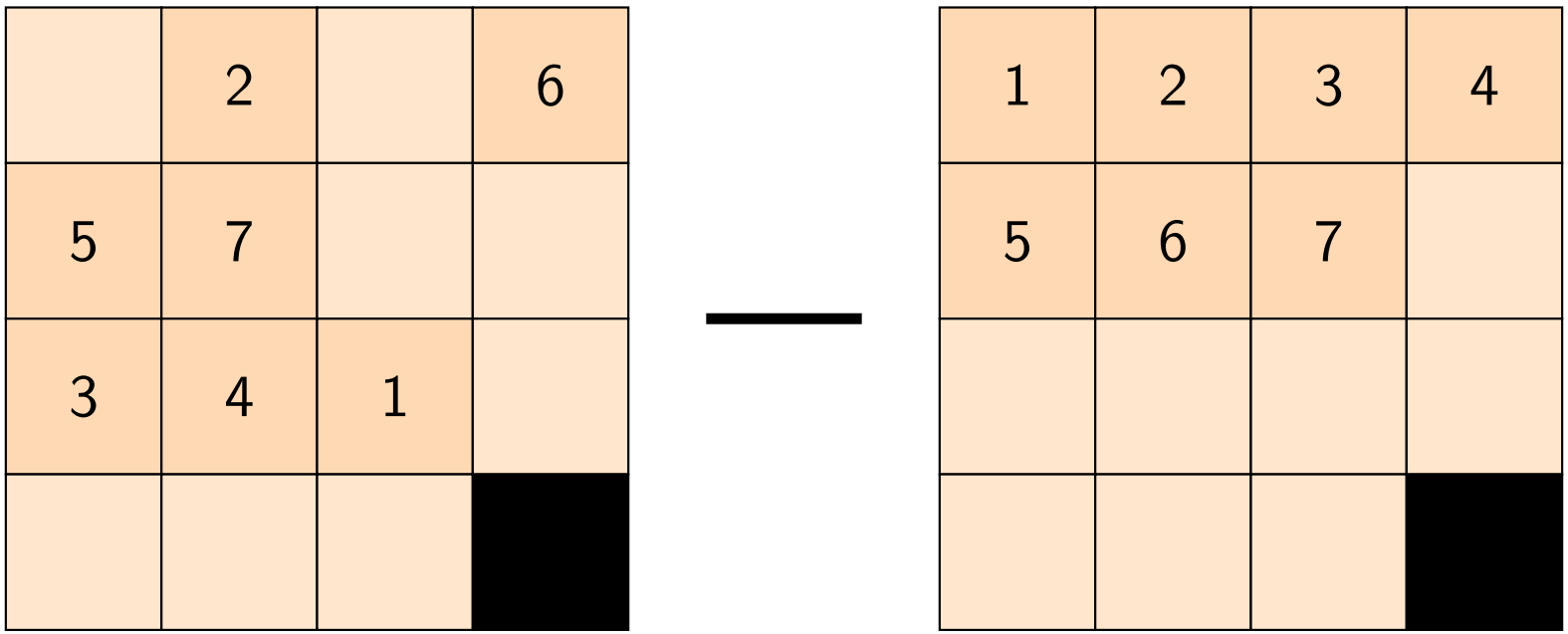
h_L^{LM}

Abstractions: Idea

Abstract state space:



Abstractions: Example



$\rightarrow h =$ **Solution in abstract state space of projected puzzle**

Abstractions: Details

- What is an abstraction, formally?

  $\rightarrow$ An abstraction is a function α mapping the state space (all world states) to a (smaller) set of abstract world states.

- How is the corresponding heuristic function h^α defined?

  $\rightarrow$ Given a world state s, $h^\alpha(s) = h^*_{\theta^\alpha}(\alpha(s))$ where $h^*_{\theta^\alpha}$ is goal distance in the abstract state space θ^α: The quotient of the state space over the equivalence relation $\sim$ where $s \sim t$ iff $\alpha(s) = \alpha(t)$.

- What is a pattern database heuristic?

  $\rightarrow$ A pattern database heuristic (PDB) is an abstraction heuristic h^α where α 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 $1, \ldots, 7$ and the blank).

- What is a merge-and-shrink heuristic?

  $\rightarrow$ A merge-and-shrink heuristic (M&S) is an abstraction heuristic h^α 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

"Recent": The last 10 or so years.

"Some": A sample of results that I personally find interesting.

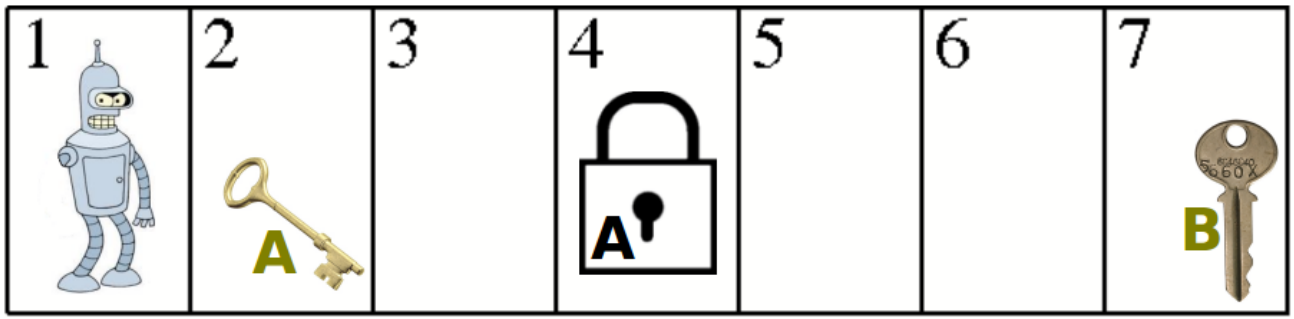
a. **Automatic generation of pattern database heuristics [Haslum et al. (2007)]:** Find a collection of patterns by hill-climbing in the space of pattern collections, pruning useless choices based on the causal graph.

b. **Shrinking by bisimulation [Nissim et al. (2011)]:** Bisimulation is a well-known concept from Verification. If we use it in merge-and-shrink [Helmert *et al.* (2007, 2014)], to decide which abstracts to aggregate when shrinking an abstraction, we get the perfect heuristic, $h^\alpha = h^*$; but this is prohibitively expensive. Conservative label reduction can yield exponential savings at no information loss.

c. **Shrinking by approximate bisimulation [Katz et al. (2012)]:** To trade accuracy for speed, need coarser notion of state similarity. K-catching bisimulation is bisimulation relative to an action subset K; choosing K enables the trade-off (and is loss-free in certain cases).

Landmarks: Example

Problem: Bring key B to position 1.



Landmarks:

- robot-at-2, robot-at-3, robot-at-4, robot-at-5, robot-at-6, robot-at-7.
- Lock-open.
- Have-key-A.
- Have-key-B.
- ...

$\rightarrow h =$ **"Number of open items on the to-do list"**

Landmarks: Details

- What is a fact landmark?

  $\rightarrow$ A fact landmark for a state s is a fact that must be true at some point along any plan for s.

- What is a disjunctive action landmark?

  $\rightarrow$ A disjunctive action landmark for a state s is a set of actions L at least one of which must be used by any plan for s.

- How can we turn a fact landmark into a disjunctive action landmark?

  $\rightarrow$ If p is a fact landmark for s, and p is not true in s, then the set L of all actions whose effect includes p is a disjunctive action landmark for s.

- Can *all* disjunctive action landmarks be derived that way?

  $\rightarrow$ No! (Simple counting argument; alternative solution paths.)

Landmarks: (Some) Recent Results

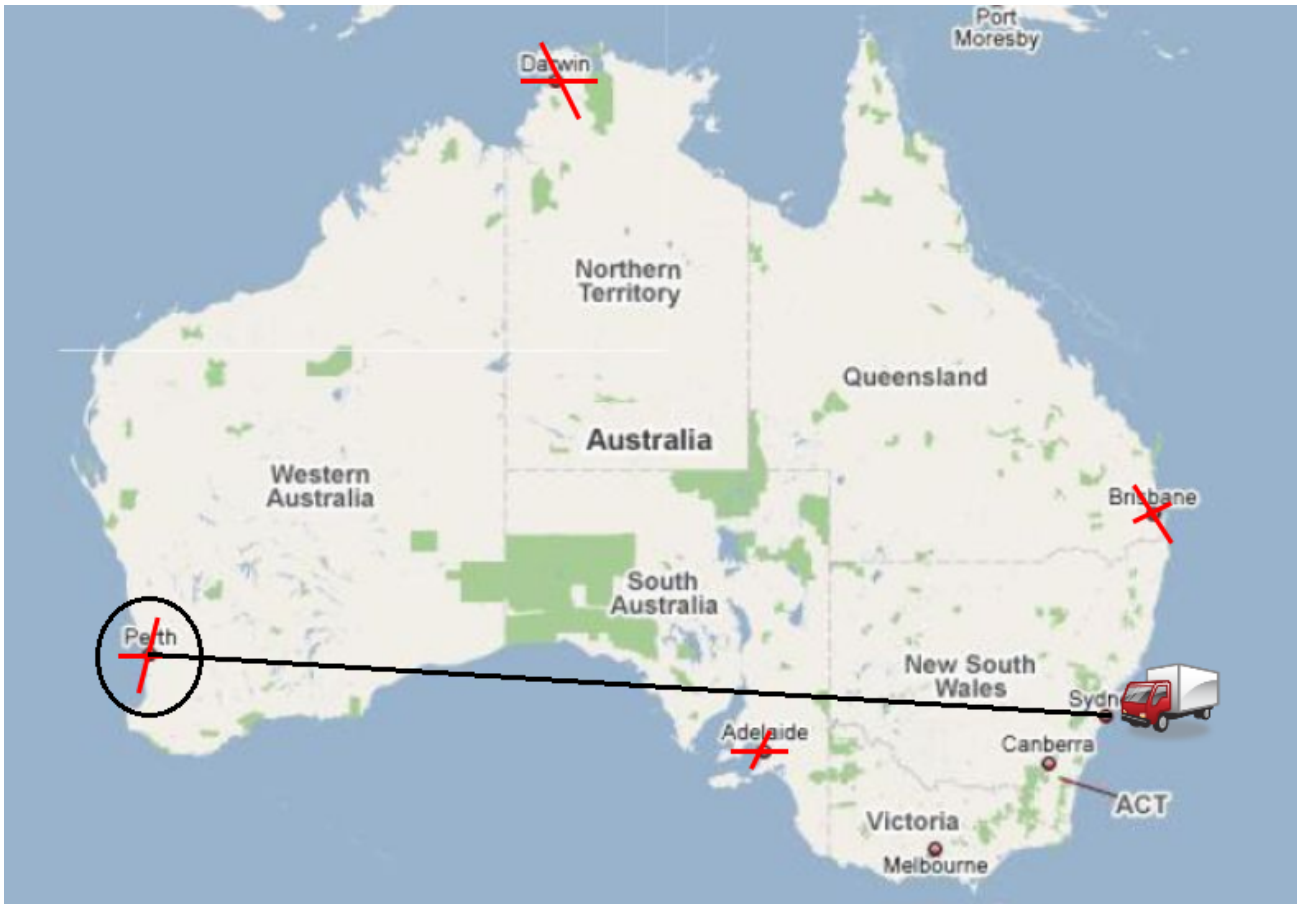
a. **Inadmissble LM h for satisficing planning in LAMA [Richter and Westphal (2010)]:** Find fact LMs for initial state, incremental maintenance of open LMs for search states. h =count of open LMs.

b. **Admissible LM h for optimal planning [Karpas and Domshlak (2009)]:** Find fact LMs for initial state, incremental maintenance of open LMs for search states; consider the induced disjunctive action LMs. h =admissible combination, using cost partitioning.

c. **LM-cut [Helmert and Domshlak (2009)]:** Find disjunctive action LMs anew for every search state, using iterated cuts in a graph over facts, where edges (p, q) correspond to actions with p in precondition and q in effect. The best admissible heuristic we have at this point!

d. **From landmarks via hitting sets to h^+ [Bonet and Helmert (2010)]:** Any plan must be a hitting set for all disjunctive action LMs. Thus the minimum cost hitting set yields an admissible heuristic. Given sufficiently many LMs, this is equal to h^+ . . . !

Critical Paths: Example



$\rightarrow h^1 =$ **Most Expensive** 1**-Sub-Tour**

Critical Paths: Example

$\rightarrow h^2 =$ **Most Expensive 2-Sub-Tour**

Critical Paths: Example

$\rightarrow h^m =$ **Most Expensive m-Sub-Tour**

Critical Paths: Details

- How is h^1 defined?

  $h^1(s) := h^1(s, G)$ where $h^1(s, g) =$

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

  $\rightarrow$ This is the same as $h^{\max}$ (cf. slide 52).

- How is h^m defined?

  $h^m(s) := h^m(s, G)$ where $h^m(s, g) =$

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

Critical Paths: (Some) Recent Results

a. **Compiling h^m into h^1 [Haslum (2009)]:** Given planning task Π, construct a compiled task Π^m such that $h^1(\Pi^m) = h^m(\Pi)$. Π^m represents all fact conjunctions c of size $\leq m$ explicitly, i.e., via a new fact π_c whose truth represents truth of c.

b. **Marriage of h^m with h^+ [Keyder et al. (2014)]:** Given planning task Π, choose a set C of fact conjunctions and construct a compiled task Π^C representing C explicitly, such that:

   1. $h^+(\Pi^C) \geq h^C$ and $h^+(\Pi^C) \geq h^+$. (h^C: Version of h^m taking into account sub-goals C.)
   2. $h^+(\Pi^C) \leq h^*(\Pi)$.
   3. For suitable C, $h^+(\Pi^C) = h^*(\Pi)$.

$\rightarrow$ Interpolates between h^+ and h^*. Systematic method for "taking some deletes into account".

$\rightarrow$ Unfortunately, $\|\Pi^C\|$ grows exponentially in $|C|$. But: See next.

Ignoring Deletes: (Some) Recent Results

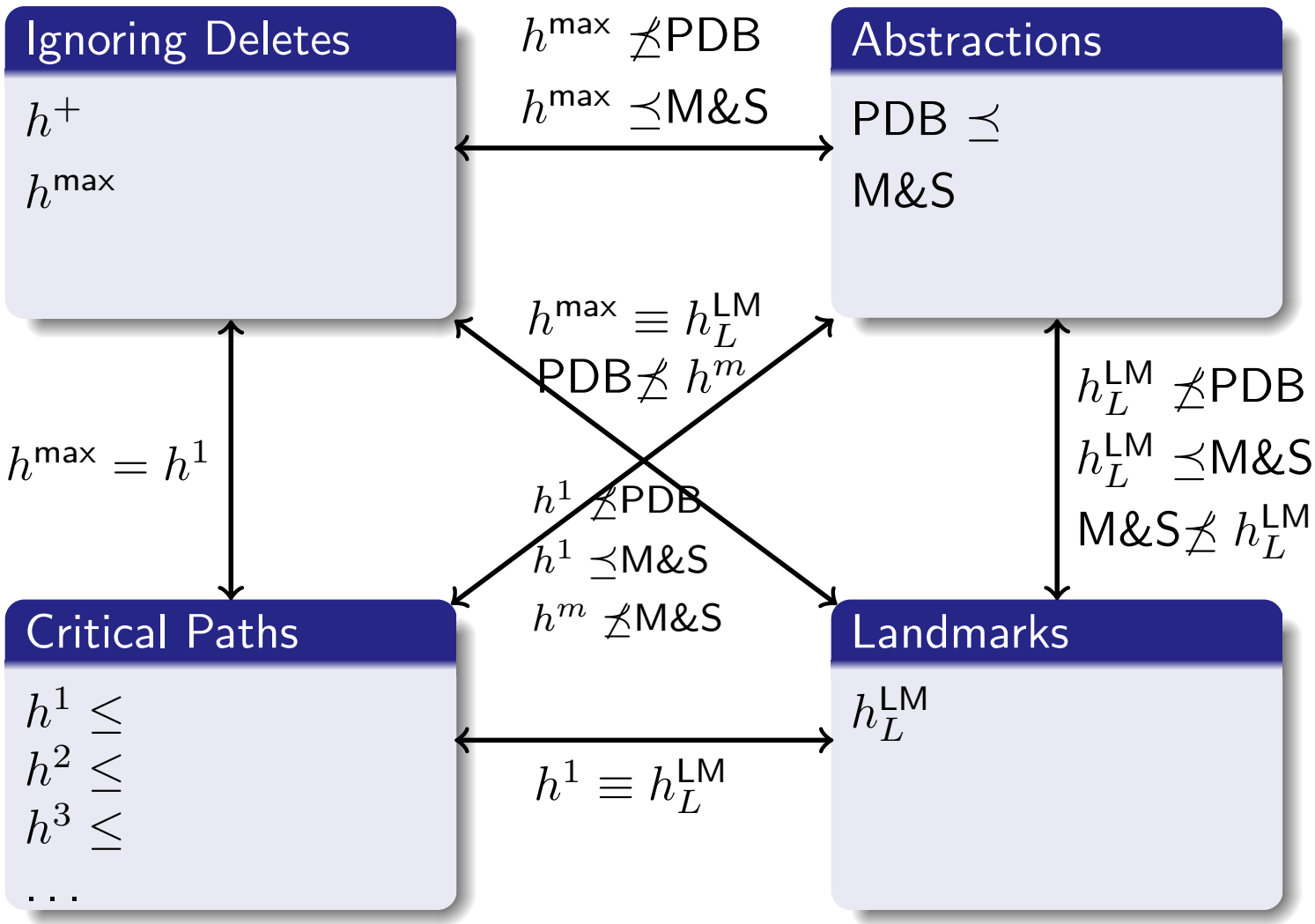
a. **Efficient marriage of h^m with h^+ [Fickert et al. (2016)]:** Actually one can characterize $h^+(\Pi^C)$ in terms of the original planning task Π, avoiding the compilation altogether while preserving (1–3); and computing $h^{\mathsf{FF}}(\Pi^C)$ in polynomial time.

b. **Automatic h^+ search space surface analysis [Hoffmann (2011)]:** One can identify classes of planning tasks whose surface has particular properties (absence of local minima), based on properties of the "causal graph" and "domain transition graphs". This connection can be exploited for automatic analysis predicting "how difficult" a task is for delete relaxation heuristics.

c. **Relaxing only some of the state variables [Domshlak et al. (2015)]:** Red variables accumulate their values (= delete-relaxed semantics), black variables switch between them (= regular semantics). Allows to systematically "take some deletes into account", interpolating between h^+ and h^*.

And, BTW: "Compilability" Between the Families

→ Framework and results by [Helmert and Domshlak (2009)]:



Questionnaire

Question!

What do you think the compilability results are good for?

(A): Nothing.
(B): Theory.
(C): Configuring solvers.
(D): Inventing new solvers.

→ (A): Nope, definitely good for something.

→ (B): Certainly good for that :-)

→ (C): To some extent, yes: If it emerges that heuristic h^A is generally dominated by heuristic h^B, then there is good reason to use h^B.

→ (D): Big big YES! Remember I mentioned LM-cut as the strongest admissible heuristic we have at this point? LM-cut was discovered as a side-product of proving that h^1 can be compiled into landmarks heuristics!

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^{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).

Interested? BSc/MSc/HiWi@FAI

Winter term course Automatic Planning:

- Heuristic functions: Partial delete relaxation, abstractions, landmarks, critical-path heuristics, cost partitionings.
- Analyzing heuristic functions: Compilability between heuristic functions, search space surface analysis.
- Optimality-preserving state-space reduction methods: Partial-order reduction, symmetry reduction, simulation-based dominance pruning.
- International Planning Competition: Overview of languages, systems, results.
- Practical experience: You'll implement your own planning system and participate in a planning competition at the end of the course.

Active research in my research group FAI:

- Star-Topology Decoupling: A new state-space decomposition technique that can yield vast savings on problems with a client-server nature. Currently: Model checking, liveness properties, weak memory models.

Interested? BSc/MSc/HiWi@FAI

Active research in my research group FAI: (continued)

- NoGood learning: The first technique able to learn sound and generalizing knowledge from dead-end states during state space search. Currently: Probabilistic planning.
- Planning and ML: Using neural networks (NN) to learn heuristic functions and/or action policies. Model checking and visualizing NN action policies.
- Explainable AI Planning (XAIP): Analyzing plan-property dependencies to explain the space of plans. User studies, NASA collaboration, deeper "Why?" questions, identifying relevant plan properties automatically, ...
- Minecraft instruction generation (collab CoII department): Hierarchical planning methods combined with natural language generation to plan instruction texts for constructing complex objects in Minecraft.
- Automated security testing (collab CISPA): Modeling attackers on networks, email infrastructure, etc. to analyze possible attacks; modeling defenses to assess cost-security trade-offs.

Reading (RN: Same As Previous Chapter)

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

  Content: Ok as a background read, but not a good introduction to modern planning techniques.

  Chapter 10 gives some background. 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 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 Π^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

Blai Bonet and Malte Helmert. Strengthening landmark heuristics via hitting sets. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pages 329–334, Lisbon, Portugal, August 2010. IOS Press.

Carmel Domshlak, Jörg Hoffmann, and Michael Katz. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, 221:73–114, 2015.

Maximilian Fickert, Jörg Hoffmann, and Marcel Steinmetz. Combining the delete relaxation with critical-path heuristics: A direct characterization. *Journal of Artificial Intelligence Research*, 56(1):269–327, 2016.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In Adele Howe and Robert C. Holte, editors, *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, pages 1007–1012, Vancouver, BC, Canada, July 2007. AAAI Press.

References II

Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 354–357. AAAI Press, 2009.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 176–183, Providence, Rhode Island, USA, 2007. Morgan Kaufmann.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, 61(3):16:1–16:63, 2014.

References III

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.

Jörg Hoffmann. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research*, 41:155–229, 2011.

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1728–1733, Pasadena, California, USA, July 2009. Morgan Kaufmann.

Michael Katz, Jörg Hoffmann, and Malte Helmert. How to relax a bisimulation? 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 101–109. AAAI Press, 2012.

References IV

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.

Emil Keyder, Jörg Hoffmann, and Patrik Haslum. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research*, 50:487–533, 2014.

Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 1983–1990. AAAI Press/IJCAI, 2011.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

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