Artificial Intelligence

13. Planning, Part II: Algorithms

How to *Solve* Arbitrary Search Problems

Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Online (Summer) Term 2020

Agenda

1. Introduction
2. How to Relax
3. The Delete Relaxation
4. The h^+ Heuristic
5. Approximating h^+
6. An Overview of Advanced Results (for Reference Only!)
7. Conclusion

Reminder: Our Agenda for This Topic

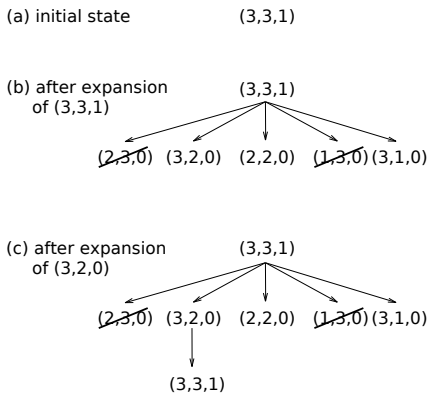
→ Our treatment of the topic "Planning" consists of Chapters 12 and 13.

- **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.

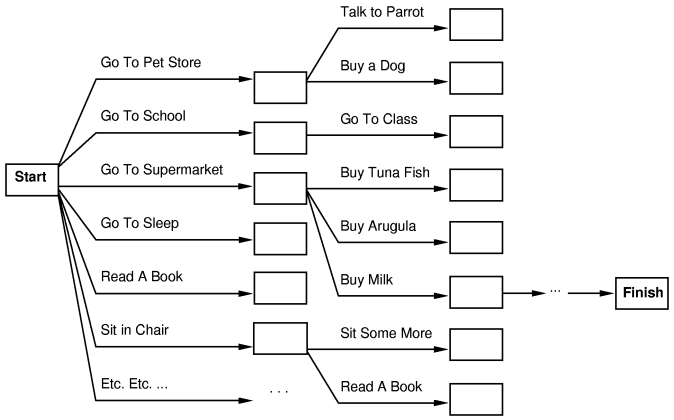
Reminder: Search

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



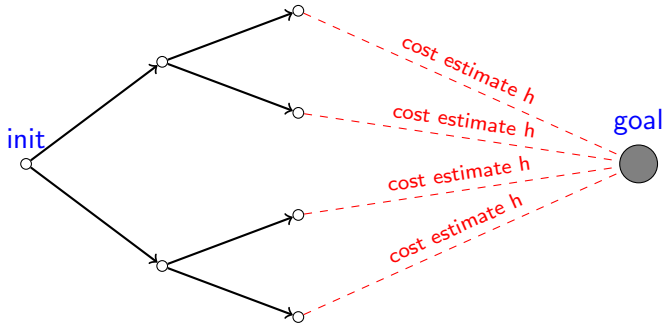
→ In planning, this is referred to as forward search, or forward state-space search.

Search in the State Space?



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

Reminder: Heuristic Search



→ 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 $[\mathrm{A}^*]$*(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.

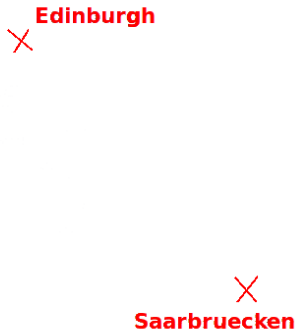
Our Agenda for This Chapter

- **How to Relax:** How to relax a problem?
  → 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 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?
  → 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:** Is that all?
  → No! This section gives a brief glimpse into the research area of heuristic search planning.

Heuristic Functions from Relaxed Problems

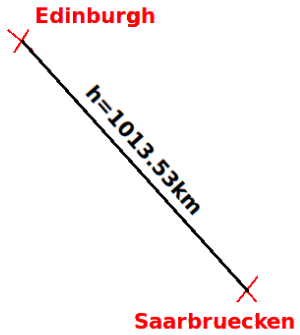
Problem II: Find a route from Saarbruecken To Edinburgh.

Heuristic Functions from Relaxed Problems



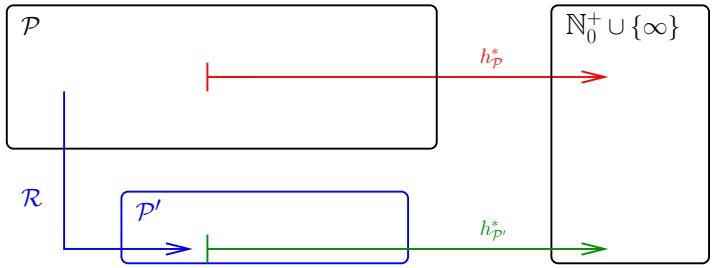
Relaxed Problem Π': Throw away the map.

Heuristic Functions from Relaxed Problems



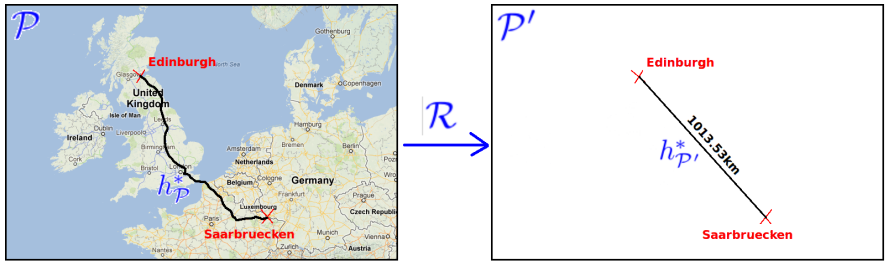
Heuristic function h: Straight line distance.

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

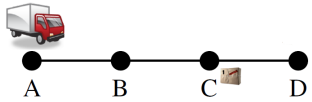
Relaxation in Route-Finding



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

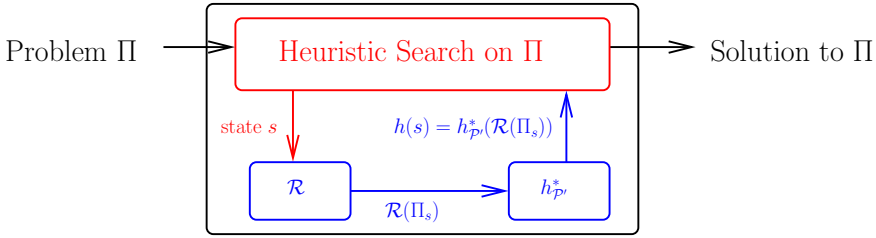
Example "Only-Adds" Relaxation: Drop the preconditions and deletes.

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

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

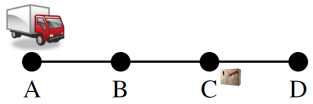
How to Relax During Search: Overview

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.

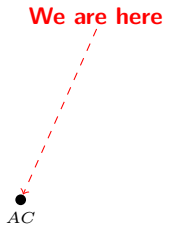
How to Relax During Search: Only-Adds



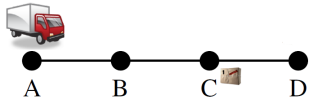
Real problem:

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

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



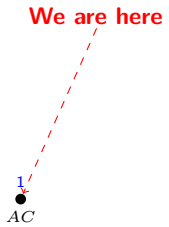
How to Relax During Search: Only-Adds



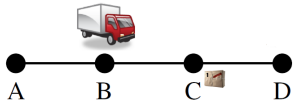
Relaxed problem:

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

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



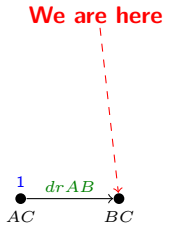
How to Relax During Search: Only-Adds



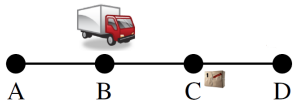
Real problem:

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

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

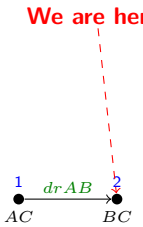


How to Relax During Search: Only-Adds

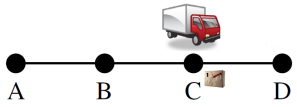


Relaxed problem:
- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s)$ =2: $\langle drBA, ulD \rangle$.

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



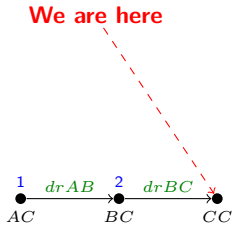
How to Relax During Search: Only-Adds



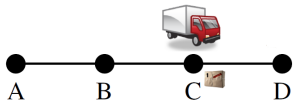
Real problem:

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

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



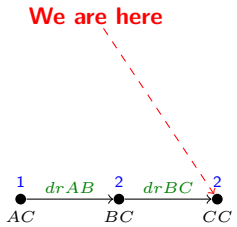
How to Relax During Search: Only-Adds



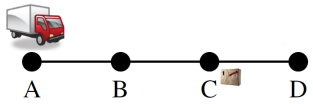
Relaxed problem:

- 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)



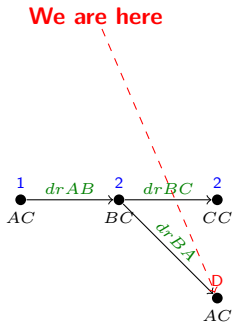
How to Relax During Search: Only-Adds



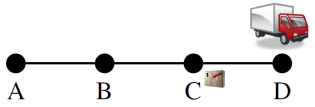
Real problem:

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

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



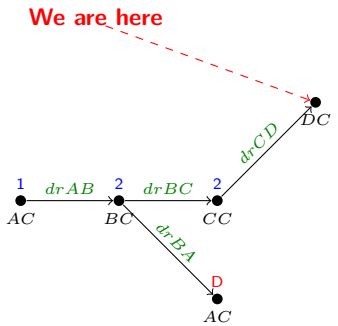
How to Relax During Search: Only-Adds



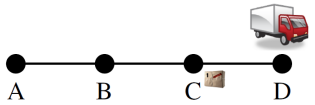
Real problem:

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

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



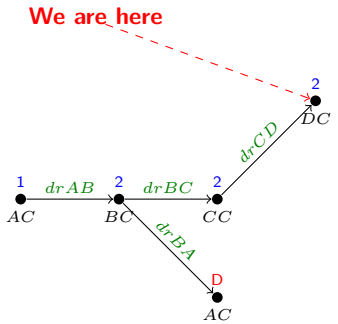
How to Relax During Search: Only-Adds



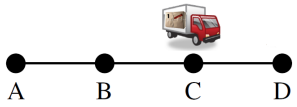
Relaxed problem:

- 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)



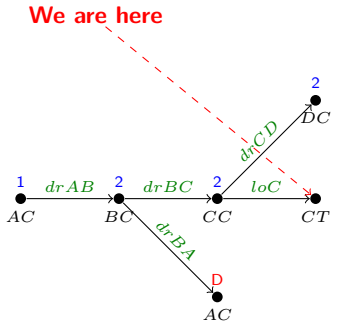
How to Relax During Search: Only-Adds



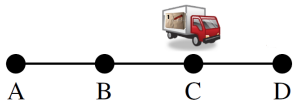
Real problem:

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

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



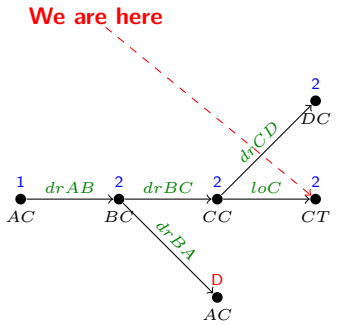
How to Relax During Search: Only-Adds



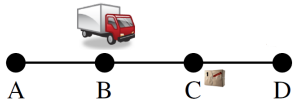
Relaxed problem:

- 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)



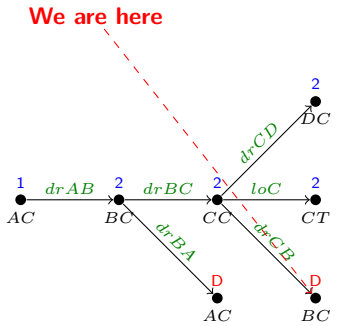
How to Relax During Search: Only-Adds



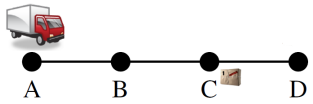
Real problem:

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

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



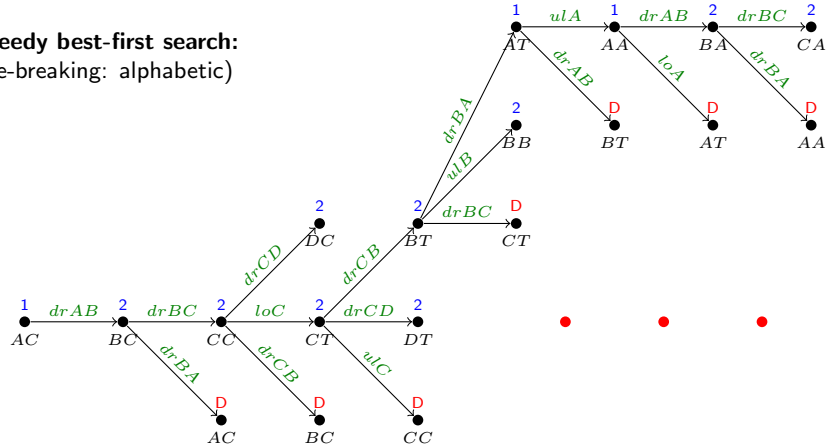
How to Relax During Search: Only-Adds



Real problem:

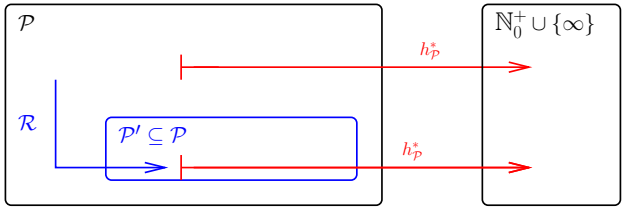
- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

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?

→ (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.

→ (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!)

→ (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.

→ (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 $del_{a^+} = \emptyset$.*

→ 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.*

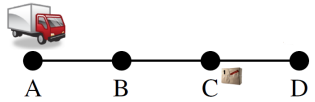
→ A relaxed plan for s is an action sequence that solves s when pretending that all delete lists are empty.

→ 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:

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

$\rightarrow$ 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.*

→ 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"

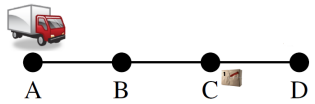
The algorithm terminates after at most $|P|$ iterations, and thus runs in polynomial time. Correctness: See slide 30.

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

Iterations on F:

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

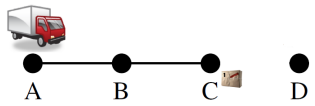
Deciding PlanEx$^+$ in "Logistics"



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

→ 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 $pre_a \subseteq F_i$.

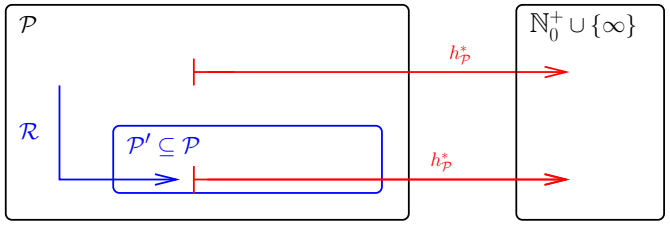
All $a \in A_0$ are applicable in I, all $a \in A_1$ are applicable in $appl(I, A_0^+)$, and so forth. Thus $F_i = appl(I, \langle A_0^+, \ldots, A_{i-1}^+ \rangle)$. (Within each A_j^+, we can sequence the actions in any order.)

Direction "⇒": 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 "⇐": 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 $(P, A, s, G)^+$.*

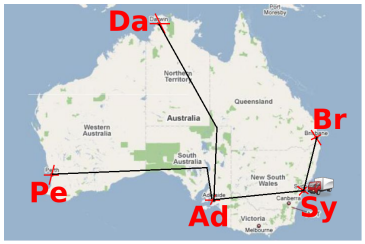
Proof Sketch. Show by induction over $0 \leq i \leq n$ that $appl(s, \langle a_1, \ldots, a_i \rangle) \subseteq appl(s, \langle a_1^+, \ldots, a_i^+ \rangle)$.

$\rightarrow$ "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.

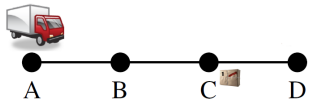
h^+ in "TSP" in Australia



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

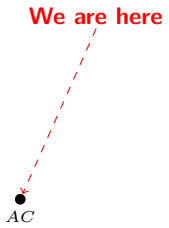
How to Relax During Search: Ignoring Deletes



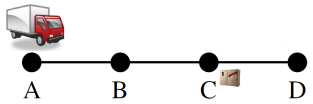
Real problem:

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

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



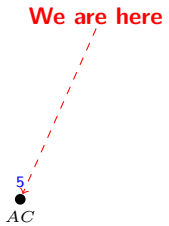
How to Relax During Search: Ignoring Deletes



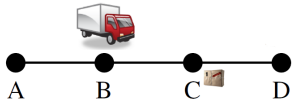
Relaxed problem:

- 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)



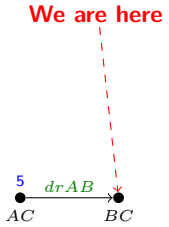
How to Relax During Search: Ignoring Deletes



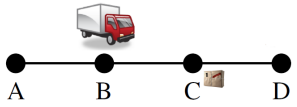
Real problem:

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

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



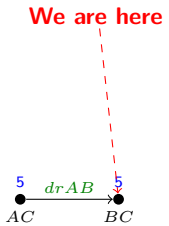
How to Relax During Search: Ignoring Deletes



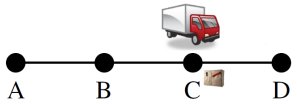
Relaxed problem:

- 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)



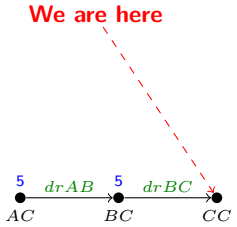
How to Relax During Search: Ignoring Deletes



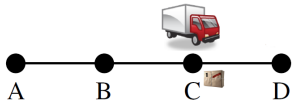
Real problem:

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

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



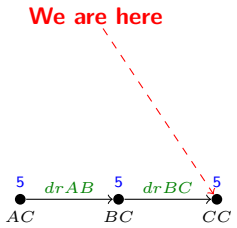
How to Relax During Search: Ignoring Deletes



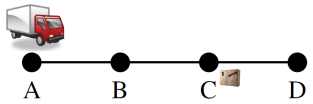
Relaxed problem:

- 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)



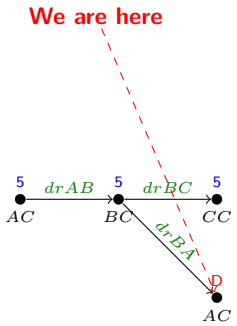
How to Relax During Search: Ignoring Deletes



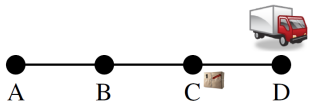
Real problem:

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

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



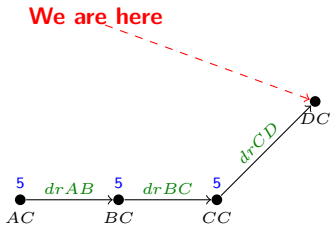
How to Relax During Search: Ignoring Deletes



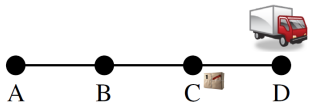
Real problem:

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

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



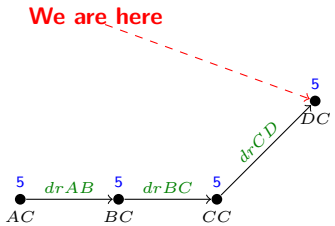
How to Relax During Search: Ignoring Deletes



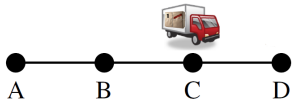
Relaxed problem:

- 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)



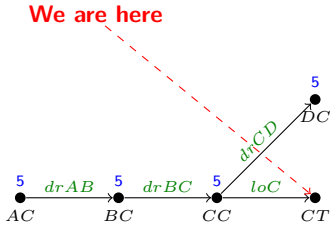
How to Relax During Search: Ignoring Deletes



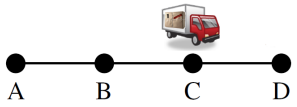
Real problem:

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

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



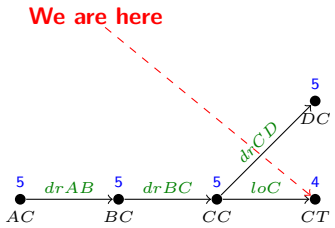
How to Relax During Search: Ignoring Deletes



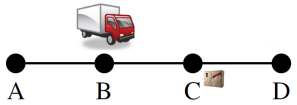
Relaxed problem:

- 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)



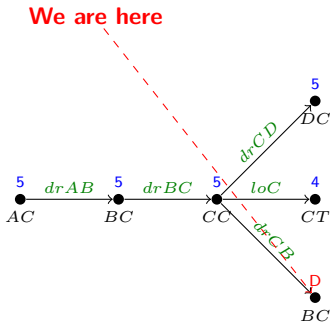
How to Relax During Search: Ignoring Deletes



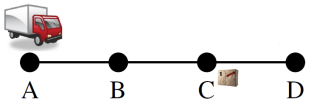
Real problem:

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

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



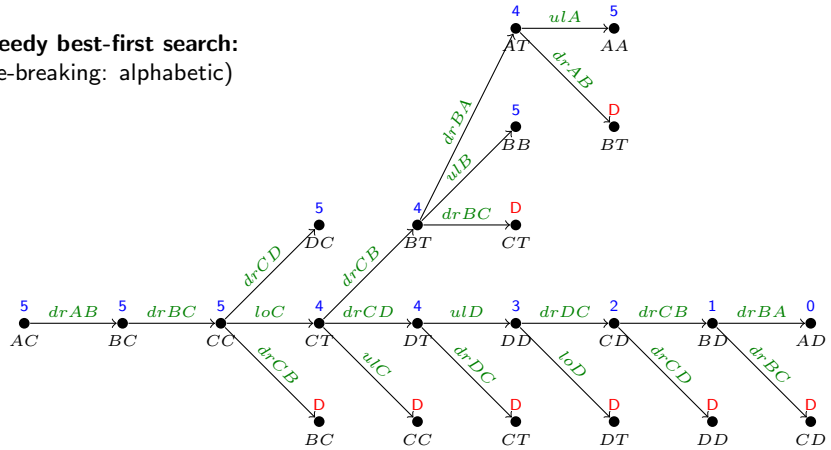
How to Relax During Search: Ignoring Deletes



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?

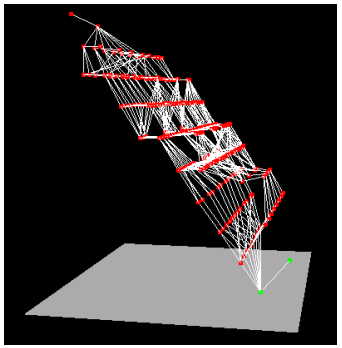
- **Known method 1:** Error relative to h^*, i.e., bounds on $|h^*(s) - h(s)|$.
- **Known method 2:** Properties of the search space surface: Local minima etc.

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

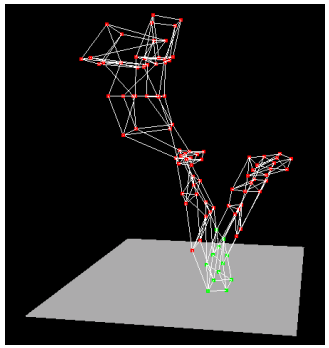
→ In many benchmarks, under h^+, local minima *provably* do not exist! [Hoffmann (2005)]

A Brief Glimpse of h^+ Search Space Surfaces

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

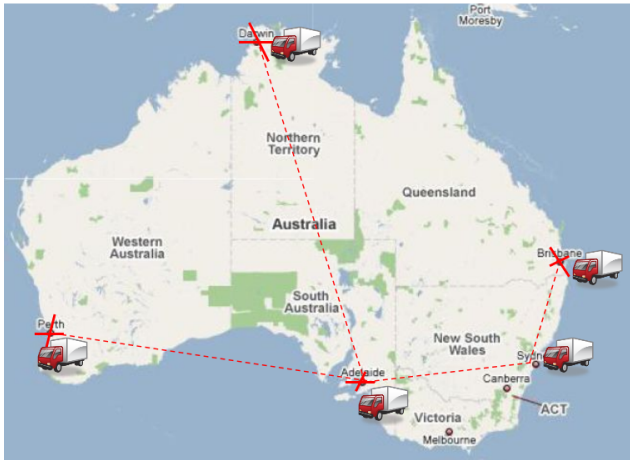


"Gripper"



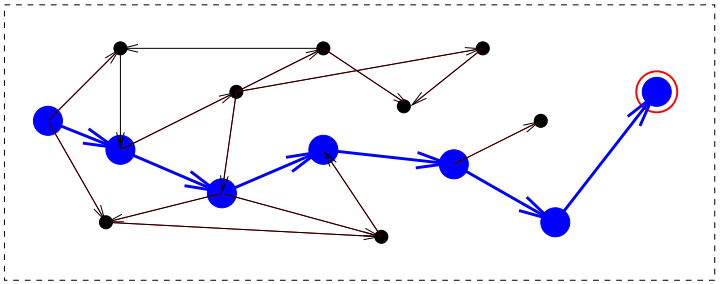
"Logistics"

h^+ in (the Real) TSP



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

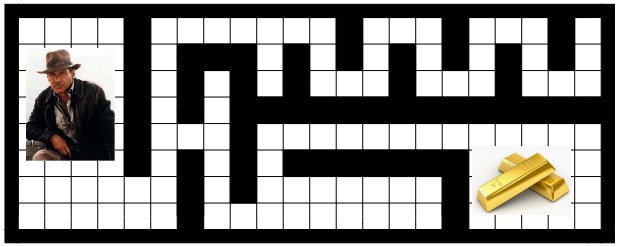
h^+ in Graphs



h^+(Graph-Distance) = 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^*.

→ (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).