AI Planning for Autonomy

4. Generating Heuristic Functions

How to Relax: Formally, and Informally, and During Search

Tim Miller and Nir Lipovetzky

THE UNIVERSITY OF MELBOURNE

Winter Term 2019

Agenda

Motivation

→ "Relax"ing is a methodology to construct heuristic functions.

- You can use it when programming a solution to some problem you want/need to solve.
- Planning systems can use it to derive a heuristic function **automatically** from the planning task description (the PDDL input).
  - **Note 1:** If the user had to supply the heuristic function by hand, then we would lose our two main selling points (generality & autonomy & flexibility & rapid prototyping, cf. → **Lecture 1-2**).
  - **Note 2:** It can of course be of advantage to give the user the *possibility* to (conveniently) supply additional heuristics. Not covered in this course.

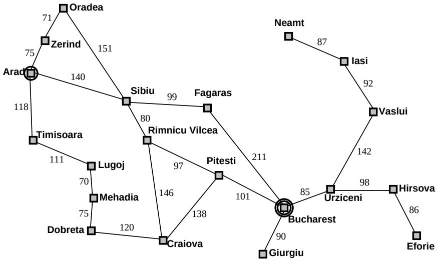
How to Relax Informally

How To Relax:

- You have a problem, $\mathcal{P}$, whose perfect heuristic h^* you wish to estimate.
- You define a simpler problem, $\mathcal{P}'$, whose perfect heuristic h'^* can be used to estimate h^*.
- You define a transformation, r, that simplifies instances from $\mathcal{P}$ into instances $\mathcal{P}'$.
- Given $\Pi \in \mathcal{P}$, you estimate $h^*(\Pi)$ by $h'^*(r(\Pi))$.

$\rightarrow$ Relaxation means to simplify the problem, and take the solution to the simpler problem as the heuristic estimate for the solution to the actual problem.

Relaxation in Route-Finding



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

How to derive straight-line distance by relaxation?

- Problem $\mathcal{P}$: Route finding.
- Simpler problem $\mathcal{P}'$: Route finding for birds.
- Perfect heuristic h'^* for $\mathcal{P}'$: Straight-line distance.
- Transformation r: Pretend you're a bird.

Relaxation in the 8-Puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Perfect heuristic h^* for $\mathcal{P}$: Actions = “A tile can move from square A to square B if A is adjacent to B and B is blank.”

- How to derive the Manhattan distance heuristic? $\mathcal{P}'$: Actions = “A tile can move from square A to square B if A is adjacent to B.”
- How to derive the misplaced tiles heuristic? $\mathcal{P}'$: Actions = “A tile can move from square A to square B.”
- h'^* (resp. r) in both: optimal cost in $\mathcal{P}'$ (resp. use different actions).
- Here: Manhattan distance = 18, misplaced tiles = 8.

"Goal-Counting" Relaxation in Australia

- Propositions P: $at(x)$ for $x \in \{Sy, Ad, Br, Pe, Da\}$; $v(x)$ for $x \in \{Sy, Ad, Br, Pe, Da\}$.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road; $pre_a = \{at(x)\}$, $add_a = \{at(y), v(y)\}$, $del_a = \{at(x)\}$.
- Initial state I: $at(Sy), v(Sy)$.
- Goal G: $at(Sy), v(x)$ for all x.

Let's "act as if we could achieve each goal directly":

- Problem $\mathcal{P}$: All STRIPS planning tasks.
- Simpler problem $\mathcal{P}'$: All STRIPS planning tasks with empty preconditions and deletes.
- Perfect heuristic h'^* for $\mathcal{P}'$: Optimal plan cost ($= h^*$).
- Transformation r: Drop the preconditions and deletes.
- Heuristic value here? 4.

$\rightarrow$ Optimal STRIPS planning with empty preconditions and deletes is still **NP**-hard! (Reduction from MINIMUM COVER, of goal set by add lists.)

$\rightarrow$ Need to approximate the perfect heuristic h'^* for $\mathcal{P}'$. Hence goal counting: just approximate h'^* by number-of-false-goals.

How to Relax Formally: Before We Begin

- The definition on the next slide is not to be found in any textbook, and not even in any paper.
- Methods generating heuristic functions differ widely, and it is quite difficult (impossible?) to make *one* definition capturing them *all* in a natural way.
- Nevertheless, a formal definition is useful to state precisely what are the relevant distinction lines in practice.
- The present definition does, I think, do a rather good job of this.

  $\rightarrow$ It nicely fits what is currently used in planning.

  $\rightarrow$ It is flexible in the distinction lines, and it captures the basic construction, as well as the essence of all relaxation ideas.

Relaxations

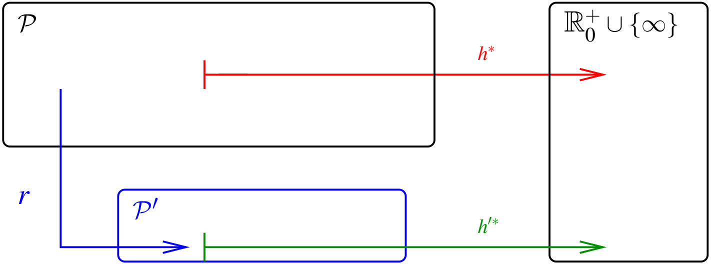
Definition (Relaxation). *Let* $h^* : \mathcal{P} \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ *be a function. A relaxation of* h^* *is a triple* $\mathcal{R} = (\mathcal{P}', r, h'^*)$ *where* $\mathcal{P}'$ *is an arbitrary set, and* $r : \mathcal{P} \mapsto \mathcal{P}'$ *and* $h'^* : P' \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ *are functions so that, for all* $\Pi \in \mathcal{P}$*, the relaxation heuristic* $h^{\mathcal{R}}(\Pi) := h'^*(r(\Pi))$ *satisfies* $h^{\mathcal{R}}(\Pi) \leq h^*(\Pi)$*. The relaxation is:*

- *native if* $\mathcal{P}' \subseteq P$ *and* $h'^* = h^*$*;*
- *efficiently constructible if there exists a polynomial-time algorithm that, given* $\Pi \in \mathcal{P}$*, computes* $r(\Pi)$*;*
- *efficiently computable if there exists a polynomial-time algorithm that, given* $\Pi' \in \mathcal{P}'$*, computes* $h'^*(\Pi')$*.*

Reminder:

- You have a problem, $\mathcal{P}$, whose perfect heuristic h^* you wish to estimate.
- You define a simpler problem, $\mathcal{P}'$, whose perfect heuristic h'^* can be used to (admissibly!) estimate h^*
- You define a transformation, r, from $\mathcal{P}$ into $\mathcal{P}'$.
- Given $\Pi \in \mathcal{P}$, you estimate $h^*(\Pi)$ by $h'^*(r(\Pi))$.

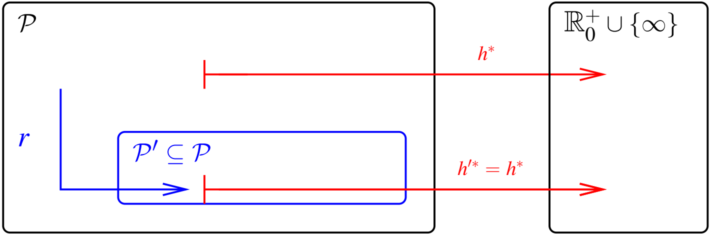
Relaxations: Illustration



Example route-finding:

- Problem $\mathcal{P}$: Route finding.
- Simpler problem $\mathcal{P}'$: Route finding for birds.
- Perfect heuristic h'^* for $\mathcal{P}'$: Straight-line distance.
- Transformation r: Pretend you're a bird.

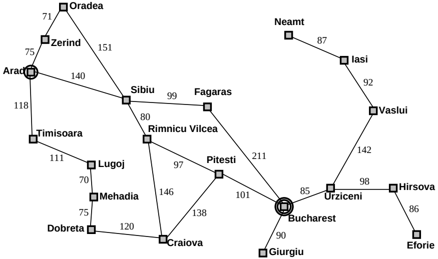
Native Relaxations: Illustration



Example "goal-counting":

- Problem $\mathcal{P}$: All STRIPS planning tasks.
- Simpler problem $\mathcal{P}'$: All STRIPS planning tasks with empty preconditions and deletes.
- Perfect heuristic h'^* for $\mathcal{P}'$: Optimal plan cost $= h^*$.
- Transformation r: Drop the preconditions and deletes.

Relaxation in Route-Finding: Properties



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Relaxation $\mathcal{R} = (\mathcal{P}', r, h'^*)$**:** Pretend you're a bird.

- Native? No: Birds don't do route-finding. (Well, it's equivalent to trivial maps with direct routes between everywhere.)
- Efficiently constructible? Yes (pretend you're a bird).
- Efficiently computable? Yes (measure straight-line distance).

Relaxation in the 8-Puzzle: Properties

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Relaxation $\mathcal{R} = (\mathcal{P}', r, h'^*)$**:** Use more generous actions rule to obtain Manhattan distance.

- Native? No: With the modified rules, it's not the "same puzzle" anymore. (Well, one could be generous in defining what the "same puzzle" is.)
- Efficiently constructible? Yes (exchange action set).
- Efficiently computable? Yes (count misplaced tiles/sum up Manhattan distances).

What shall we do with the relaxation?

What if $\mathcal{R}$ is not efficiently constructible?

- Either (a) approximate r, or (b) design r in a way so that it will typically be feasible, or (c) just live with it and hope for the best.
- Vast majority of known relaxations (in planning) are efficiently constructible.

What if $\mathcal{R}$ is not efficiently computable?

- Either (a) approximate h'^*, or (b) design h'^* in a way so that it will typically be feasible, or (c) just live with it and hope for the best.
- Many known relaxations (in planning) are efficiently computable, some aren't. The latter use (a); (b) and (c) are not used anywhere right now.

"Goal-Counting" Relaxation in Australia: Properties

- Propositions P: $at(x)$ for $x \in \{Sy, Ad, Br, Pe, Da\}$; $v(x)$ for $x \in \{Sy, Ad, Br, Pe, Da\}$.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road; $pre_a = \{at(x)\}$, $add_a = \{at(y), v(y)\}$, $del_a = \{at(x)\}$.
- Initial state I: $at(Sy), v(Sy)$.
- Goal G: $at(Sy), v(x)$ for all x.

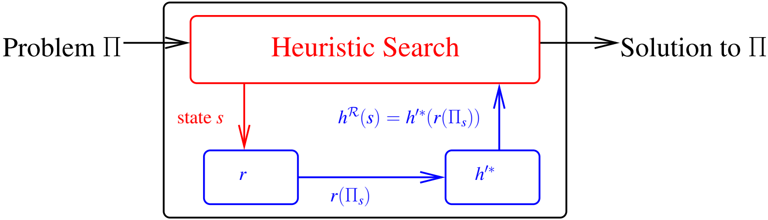
Relaxation $\mathcal{R} = (\mathcal{P}', r, h'^*)$: Remove preconditions and deletes, then use h^*.

- Native? Yes: Planning with empty preconditions and deletes is a special case of planning (i.e., a sub-class of $\mathcal{P}$).
- Efficiently constructible? Yes (drop preconditions and deletes).
- Efficiently computable? **No!** Optimal planning is still **NP**-hard in this case (MINIMUM COVER of goal set by add lists).

What shall we do with the relaxation? $\rightarrow$ Use method (a): Approximate h^* in $\mathcal{P}'$ by counting the number of goals not currently true.

How to Relax During Search: Diagram

Using a relaxation $\mathcal{R} = (\mathcal{P}', r, h'^*)$ during search:

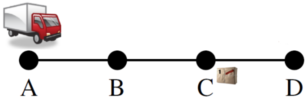


$\rightarrow$ Π_s: Π with initial state replaced by s, i.e., $\Pi = (F, A, c, I, G)$ changed to (F, A, c, s, G).

$\rightarrow$ The task of finding a plan for search state s.

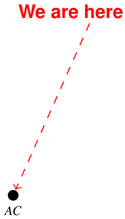
$\rightarrow$ We will be using this notation in the course!

How to Relax During Search: Goal-Counting

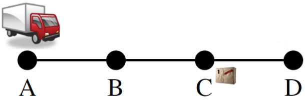


Real problem:
- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY$, loX, ulX.

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



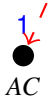
How to Relax During Search: Goal-Counting



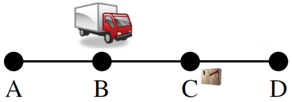
Relaxed problem:
- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 1$.

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

We are here



How to Relax During Search: Goal-Counting

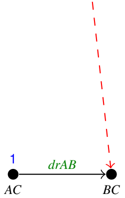


Real problem:

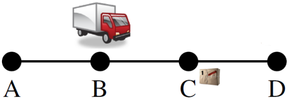
- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $AC \xrightarrow{drAB} BC$.

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

We are here



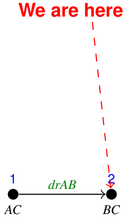
How to Relax During Search: Goal-Counting



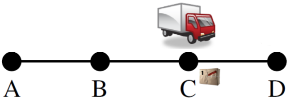
Relaxed problem:

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

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



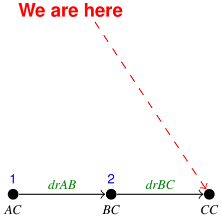
How to Relax During Search: Goal-Counting



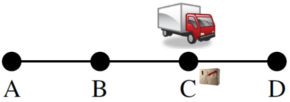
Real problem:

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

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



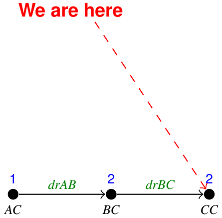
How to Relax During Search: Goal-Counting



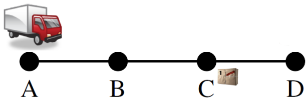
Relaxed problem:

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

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



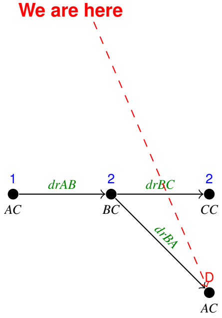
How to Relax During Search: Goal-Counting



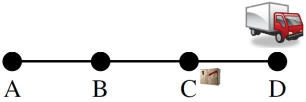
Real problem:

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

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



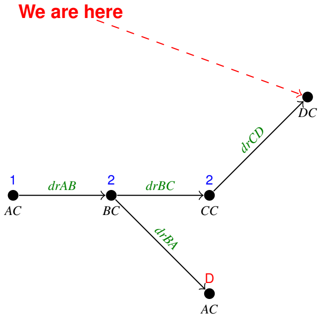
How to Relax During Search: Goal-Counting



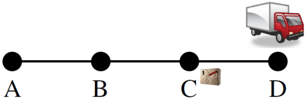
Real problem:

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

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



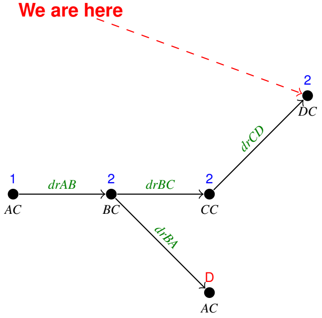
How to Relax During Search: Goal-Counting



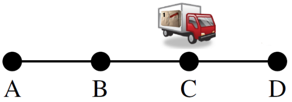
Relaxed problem:

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

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



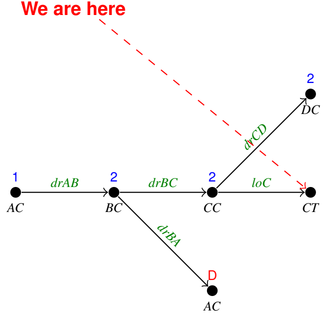
How to Relax During Search: Goal-Counting



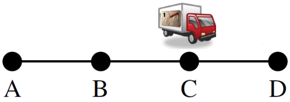
Real problem:

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

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



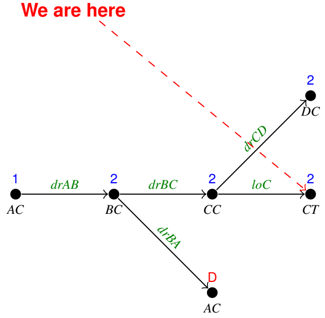
How to Relax During Search: Goal-Counting



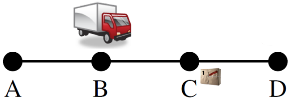
Relaxed problem:

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

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



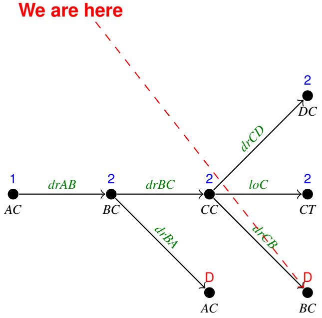
How to Relax During Search: Goal-Counting



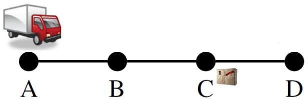
Real problem:

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

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



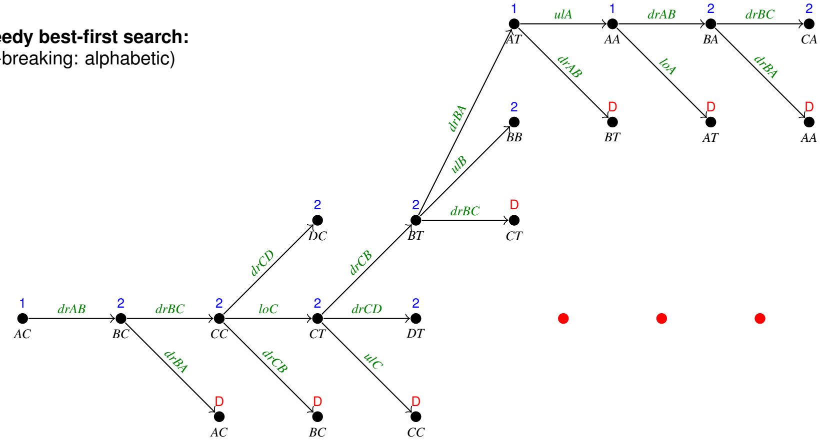
How to Relax During Search: Goal-Counting



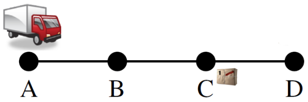
Real problem:

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

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



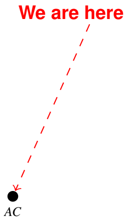
How to Relax During Search: Ignoring Deletes



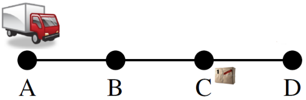
Real problem:

- 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



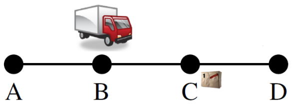
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: pre, add.
- $h^{\mathcal{R}}(s) = h^{+}(s) = 5$.

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

We are here

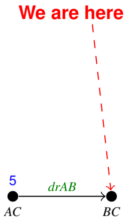
How to Relax During Search: Ignoring Deletes



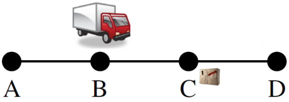
Real problem:

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

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



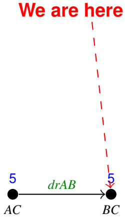
How to Relax During Search: Ignoring Deletes



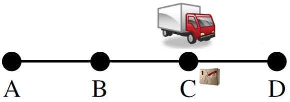
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: pre, add.
- $h^{\mathcal{R}}(s) = h^{+}(s) = 5$.

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



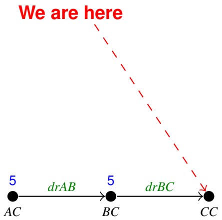
How to Relax During Search: Ignoring Deletes



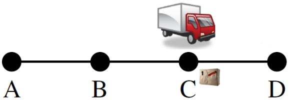
Real problem:

- 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

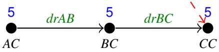


Relaxed problem:

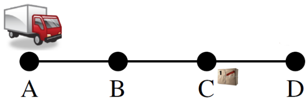
- State s: CC; goal G: AD.
- Actions A: pre, add.
- $h^{\mathcal{R}}(s) = h^{+}(s) = 5$.

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

We are here



How to Relax During Search: Ignoring Deletes

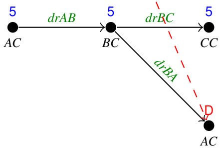


Real problem:

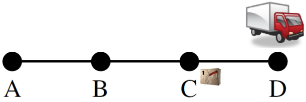
- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

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

We are here



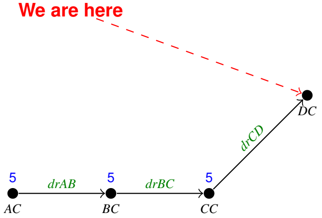
How to Relax During Search: Ignoring Deletes



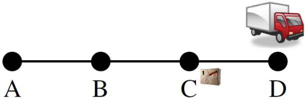
Real problem:

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

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

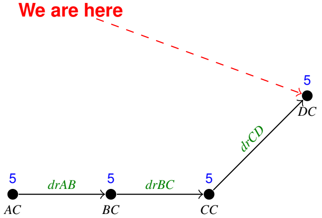


How to Relax During Search: Ignoring Deletes

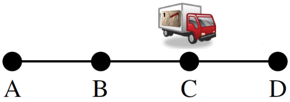


Relaxed problem:
- State s: DC; goal G: AD.
- Actions A: pre, add.
- $h^{\mathcal{R}}(s) = h^{+}(s) =$5.

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



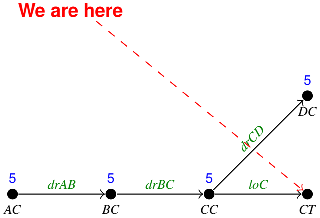
How to Relax During Search: Ignoring Deletes



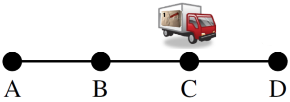
Real problem:

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

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

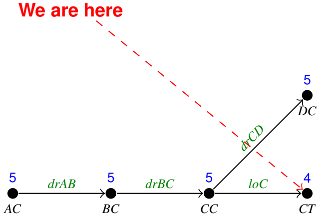


How to Relax During Search: Ignoring Deletes

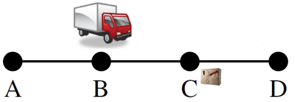


Relaxed problem:
- State s: CT; goal G: AD.
- Actions A: pre, add.
- $h^{\mathcal{R}}(s) = h^{+}(s) =$4.

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



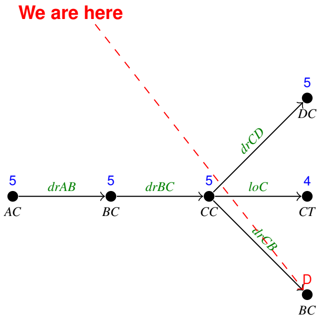
How to Relax During Search: Ignoring Deletes



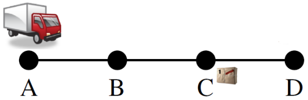
Real problem:

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

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



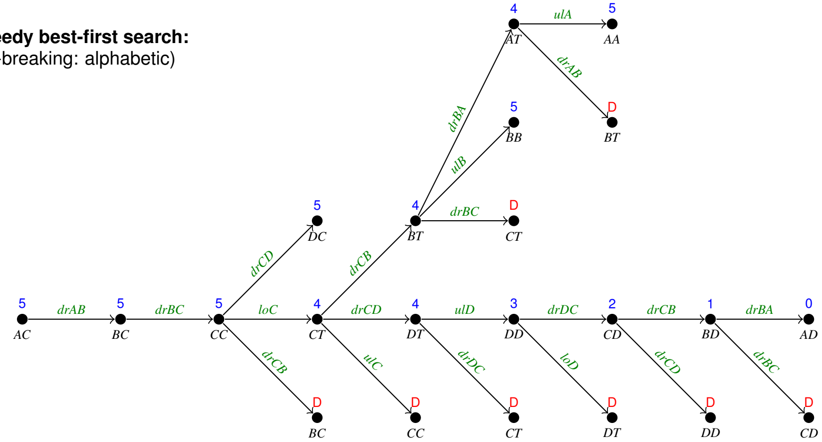
How to Relax During Search: Ignoring Deletes



Real problem:

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

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



Questionnaire

Question!

Say we have a robot with one gripper, two rooms A and B, and n balls we must transport. The actions available are $moveXY$, $pickB$ and $dropB$; say $h =$"number of balls not yet in room B". Can h be derived as $h^{\mathcal{R}}$ for a relaxation $\mathcal{R}$?

(A): No.

(B): Yes, just drop the deletes

(C): Sure, *every* admissible h can be derived via a relaxation.

(D): I'd rather relax at the beach.

$\rightarrow$ We can define $\mathcal{P}'$ as the problem of computing the cardinality of a finite set, and define r as the function that maps a state to the set of balls not yet in room B. So: (A) is incorrect, (B) is incorrect, should drop preconditions and deletes.

$\rightarrow$ (C): Yes. Admissibility of $h^{\mathcal{R}}$ is the only strict requirement made by the definition. Given admissible $h : \mathcal{P} \mapsto \mathbb{R}_0^+ \cup \{\infty\}$, we can simply define $\mathcal{P}' := \mathcal{P}$ and take r to be the identity function $id_{\mathcal{P}}$. In other words, $\mathcal{R} := (\mathcal{P}, id_{\mathcal{P}}, h)$ is a relaxation with $h^{\mathcal{R}} = h$. (And, yes, h here is admissible.)

Summary

- Relaxation is a method to compute heuristic functions.
- Given a problem $\mathcal{P}$ we want to solve, we define a relaxed problem $\mathcal{P}'$. We derive the heuristic by mapping into $\mathcal{P}'$ and taking the solution to this simpler problem as the heuristic estimate.
- Relaxations can be native, efficiently constructible, and/or efficiently computable. None of this is a strict requirement to be useful.

- During search, the relaxation is used *only inside* the computation of the heuristic function on each state; the relaxation does not affect anything else. (This can be a bit confusing especially for native relaxations like ignoring deletes.)

Remarks

The goal-counting approximation $h =$"count the number of goals currently not true" is **a very uninformative heuristic function:**

- Range of heuristic values is small ($0 \ldots |G|$).
- We can transform any planning task into an equivalent one where $h(s) = 1$ for all non-goal states s. How? Replace goal by new fact g and add a new action achieving g with precondition G.
- Ignores almost all structure: Heuristic value does not depend on the actions at all!

$\rightarrow$ By the way, is h safe/goal-aware/admissible/consistent? Only safe and goal-aware.

$\rightarrow$ We will see in $\rightarrow$ **the next lecture** how to compute **much** better heuristic functions.