Agenda

Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task P.
Output: A plan for P, or 'unsolvable' if no plan for P exists.

Optimal Planning

Input: A planning task P.
Output: An optimal plan for P, or 'unsolvable' if no plan for P exists.

$\rightarrow$ The techniques successful for either one of these are almost disjoint!

$\rightarrow$ Satisficing planning is much more effective in practice

$\rightarrow$ Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

Decision Problems in Planning

Definition (PlanEx). By PlanEx, we denote the problem of deciding, given a planning task P, whether or not there exists a plan for P.

$\rightarrow$ Corresponds to satisficing planning.

Definition (PlanLen). By PlanLen, we denote the problem of deciding, given a planning task P and an integer B, whether or not there exists a plan for P of length at most B.

$\rightarrow$ Corresponds to optimal planning.

Reminder (?): **NP** and **PSPACE**

Def Turing machine: Works on a tape consisting of tape cells, across which its R/W head moves. The machine has internal states. There are transition rules specifying, given the current cell content and internal state, what the subsequent internal state will be, and whether the R/W head moves left or right or remains where it is. Some internal states are accepting ('yes'; else 'no').

Def NP: Decision problems for which there exists a non-deterministic Turing machine that runs in time polynomial in the size of its input. Accepts if at least one of the possible runs accepts.

Def PSPACE: Decision problems for which there exists a deterministic Turing machine that runs in space polynomial in the size of its input.

Relation: Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus **PSPACE = NPSPACE**, and hence (trivially) **NP** *subset* **PSPACE**.

→ For comprehensive details, please see a text book. My personal favorite is [Garey and Johnson (1979)].

Computational Complexity of PlanEx and PlanLen

Theorem. PlanEx and PlanLen is **PSPACE**-complete.

→ 'At least as hard as any other problem contained in **PSPACE**.'

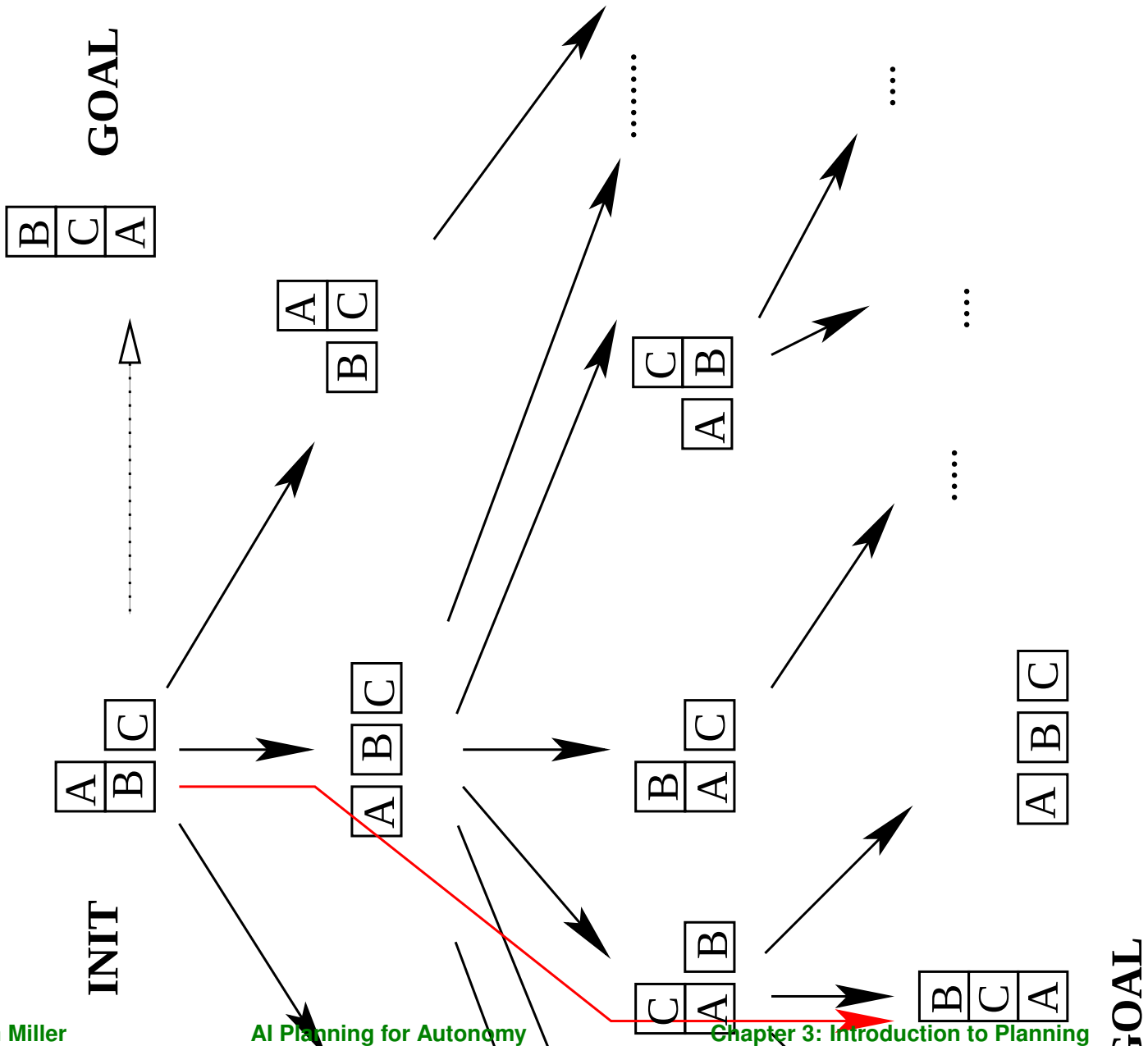
→ Details: [Bylander (1994)]

Domain-Specific PlanEx vs. PlanLen . . .

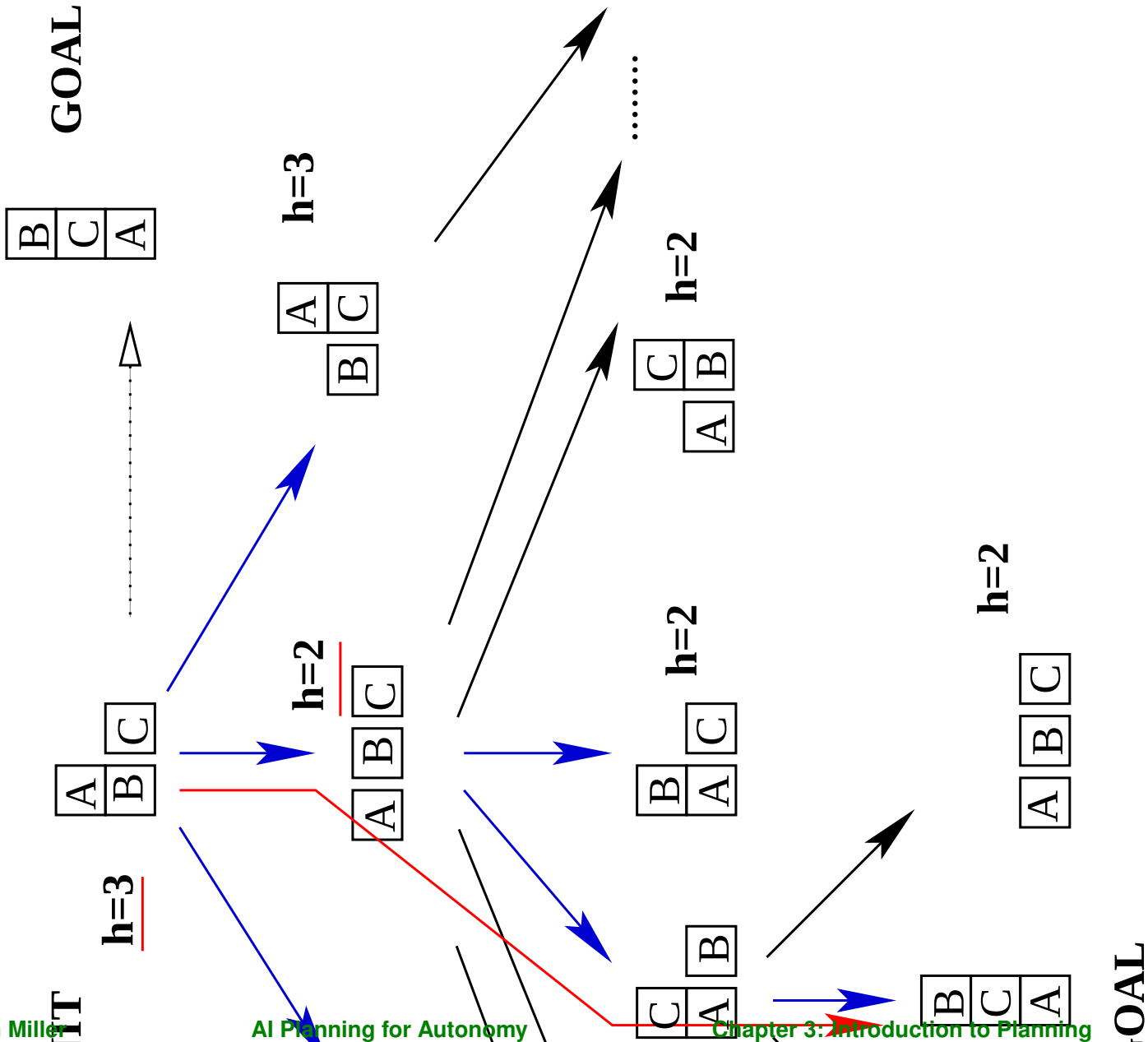
- In general, both have the same complexity.
- Within particular applications, bounded length plan existence is often harder than plan existence.
- This happens in many IPC benchmark domains: PlanLen is **NP**-complete while PlanEx is in **P**.
  - For example: Blocksworld and Logistics.

$\rightarrow$ In practice, optimal planning is (almost) never 'easy'

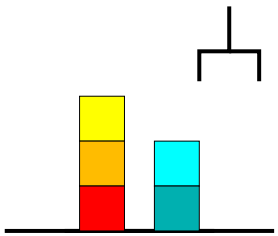
The Blocksworld is Hard?



The Blocksworld is Hard!

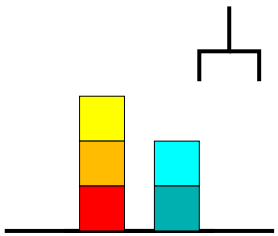


So, Why All the Fuss? Example Blocksworld



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

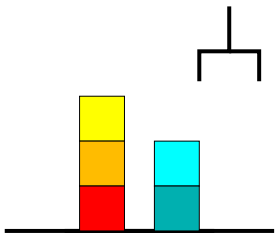
So, Why All the Fuss? Example Blocksworld



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

So, Why All the Fuss? Example Blocksworld

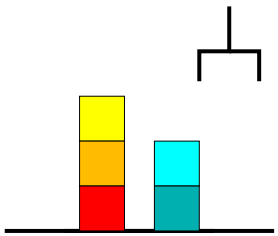


- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

$\rightarrow$ State spaces may be huge. In particular, the state space is typically exponentially large in the size of its specification via the problem Π (up next).

So, Why All the Fuss? Example Blocksworld



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

$\rightarrow$ State spaces may be huge. In particular, the state space is typically exponentially large in the size of its specification via the problem Π (up next).

$\rightarrow$ In other words: Search problems typically are computationallly hard (e.g., optimal Blocksworld solving is **NP**-complete).

Agenda

Computation: how to solve Strips planning problems?

Key issue: exploit two roles of **language**:

- specifcation: concise model description
- computation: reveal useful heuristic information (structure)

Two traditional approaches: search vs. decomposition

- explicit search of the state model $S(P)$ direct but not effective til recently
- near decomposition of the planning problem thought a better idea

Computational Approaches to Classical Planning

- **General Problem Solver (GPS) and Strips** (50's-70's): mean-ends analysis, decomposition, regression, ...
- **Partial Order (POCL) Planning** (80's): work on any open subgoal, resolve threats; UCPOP 1992
- **Graphplan** (1995 – 2000): build graph containing all possible **parallel** plans up to certain length; then extract plan by searching the graph backward from Goal
- **SATPlan** (1996 – ...): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- **Heuristic Search Planning** (1996 – ...): search state space $\mathcal{S}(P)$ with heuristic function h extracted from problem P
- **Model Checking Planning** (1998 – ...): search state space $\mathcal{S}(P)$ with 'symbolic' Breadth first search where sets of states represented by formulas implemented by BDDs ...

State of the Art in Classical Planning

- significant **progress** since Graphplan
- **empirical methodology**
  - standard PDDL language
  - planners and benchmarks available; competitions
  - focus on performance and scalability
- **large problems solved** (non-optimally)
- different **formulations** and **ideas**
  1. Planning as **Heuristic Search**
  2. Planning as **SAT**
  3. **Other:** Local Search (LPG), Monte-Carlo Search (Arvand), ...

I'll focus on **1** mainly, and partially on **2**

Agenda

The International Planning Competition (IPC)

Competition?

'Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners.'

- 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014
- PDDL [McDermott and others (1998); Fox and Long (2003); Hoffmann and Edelkamp (2005)]
- $\approx$ 40 domains, $\gg$ 1000 instances, 74 (!!) planners in 2011
- Optimal track vs. satisficing track
- Various others: uncertainty, learning, . . .

http://ipc.icaps-conference.org/

... Winners

- IPC 2000: Winner FF, heuristic search (HS), IPC 2002: Winner LPG, HS
- IPC 2004: Winner satisficing SGPlan, HS; optimal SATPLAN, compilation to SAT
- IPC 2006: Winner satisficing SGPlan, HS; optimal SATPLAN, compilation to SAT
- IPC 2008: Winner satisficing LAMA, HS; optimal Gamer, symbolic search
- IPC 2011: Winner satisficing LAMA, HS; optimal Fast-Downward, HS
- IPC 2014: Winner satisficing IBACOP, HS Portfolio; optimal SymbA*, symbolic search
- IPC 2018: Winner satisficing FD/BFWS-LAPKT, HS Portfolio/Width-Based planning; optimal Delfi, HS portfolio

→ For the rest of this chapter, we focus on planning as heuristic search
→ This is a VERY short summary of the history of the IPC! There are many different categories, and many different awards

Disclaimer on IPC

Question

If planners x, y both compete in IPC'YY, and x wins, is x 'better than' y?

(A): Yes. (B): No.

$\rightarrow$ Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a 'winner'! On other domains and/or according to other criteria, you may well be better off with the 'looser'.

$\rightarrow$ It's complicated, over-simplification is dangerous. (But, of course, nevertheless is being done all the time).

Agenda

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- Classical search problems require to find a path of actions leading from an initial state to a goal state.
- They assume a single-agent, fully-observable, deterministic, static environment. Despite this, they are ubiquitous in practice.
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- Plan existence (bounded or not) is **PSPACE**-complete to decide for STRIPS.
- PDDL is the de-facto standard language for describing planning problems.

Reading, ctd.

- *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* [*Joerg Hoffmann, 2011*]

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`

  Content: Joerg personal perspective on planning. Very modern indeed. Excerpt from the abstract:

  The area has long had an affinity towards playful illustrative examples, imprinting it on the mind of many a student as an area concerned with the rearrangement of blocks, and with the order in which to put on socks and shoes (not to mention the disposal of bombs in toilets). Working on the assumption that this "student" is you – the readers in earlier stages of their careers – I herein aim to answer three questions that you surely desired to ask back then already:

  What is it good for? Does it work? Is it interesting to do research in?

Extra material

Introduction to STRIPS, from simple games to StarCraft:
`http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/`

Online Editor to model in PDDL:
`http://editor.planning.domains`

AI Planning for Autonomy

4. Generating Heuristic Functions

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

Chris Ewin & Tim Miller

THE UNIVERSITY OF MELBOURNE

With slides by Nir Lipovetsky

Agenda

Agenda

Motivation: For You

Imagine . . .

You have completed your studies, and have been hired by some company in the software-making business.

Motivation: For You

Imagine . . .

You have completed your studies, and have been hired by some company in the software-making business.

Boss: *Hey you, here's that problem. Solve it.*

Motivation: For You

Imagine . . .

You have completed your studies, and have been hired by some company in the software-making business.

Boss: *Hey you, here's that problem. Solve it.*

You (thinking): *Hm, I think heuristic search might work.*

Motivation: For You

Imagine . . .

You have completed your studies, and have been hired by some company in the software-making business.

Boss: *Hey you, here's that problem. Solve it.*

You (thinking): *Hm, I think heuristic search might work.*

Motivation: For You

Imagine . . .

You have completed your studies, and have been hired by some company in the software-making business.

Boss: *Hey you, here's that problem. Solve it.*

You (thinking): *Hm, I think heuristic search might work.*

You (thinking): *Hm, I need a heuristic function. How?!?*

Motivation: For Planning Systems

Imagine . . .

You are a planning system.

Motivation: For Planning Systems

Imagine . . .

You are a planning system.

Somebody: *Hey you, here's that PDDL input. Solve it.*

Motivation: For Planning Systems

Imagine . . .

You are a planning system.

Somebody: *Hey you, here's that PDDL input. Solve it.*

You (thinking): *Hm, the only method I have is heuristic search.*

Motivation: For Planning Systems

Imagine . . .

You are a planning system.

Somebody: *Hey you, here's that PDDL input. Solve it.*

You (thinking): *Hm, the only method I have is heuristic search.*

Motivation: For Planning Systems

Imagine . . .

You are a planning system.

Somebody: *Hey you, here's that PDDL input. Solve it.*

You (thinking): *Hm, the only method I have is heuristic search.*

You (thinking): *Hm, I need a heuristic function.*
[Note: You programmer is presently in Falkland Islands.]

Motivation: For Planning Systems

Imagine . . .

You are a planning system.

Somebody: *Hey you, here's that PDDL input. Solve it.*

You (thinking): *Hm, the only method I have is heuristic search.*

You (thinking): *Hm, I need a heuristic function.*
[Note: You programmer is presently in Falkland Islands.]

You: *How?!?*

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).

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**).

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.

Agenda

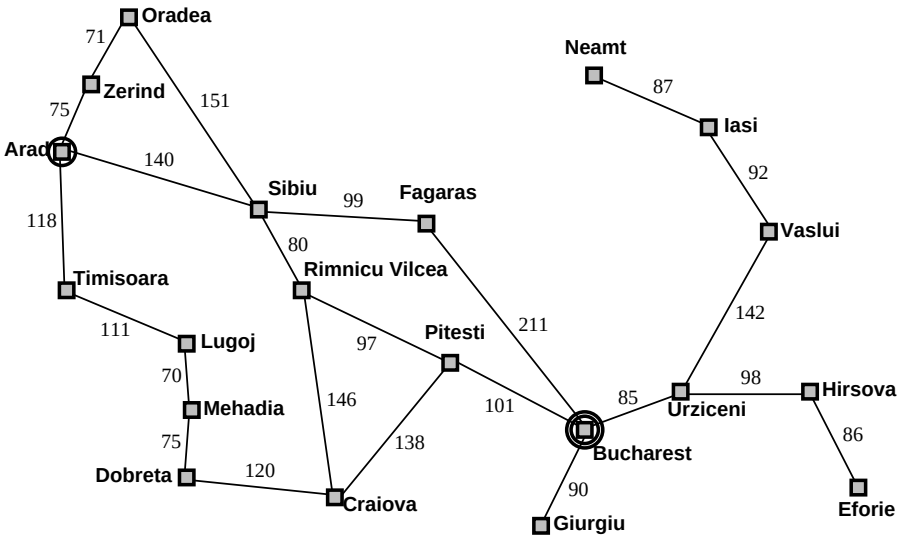
How to Relax Informally

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

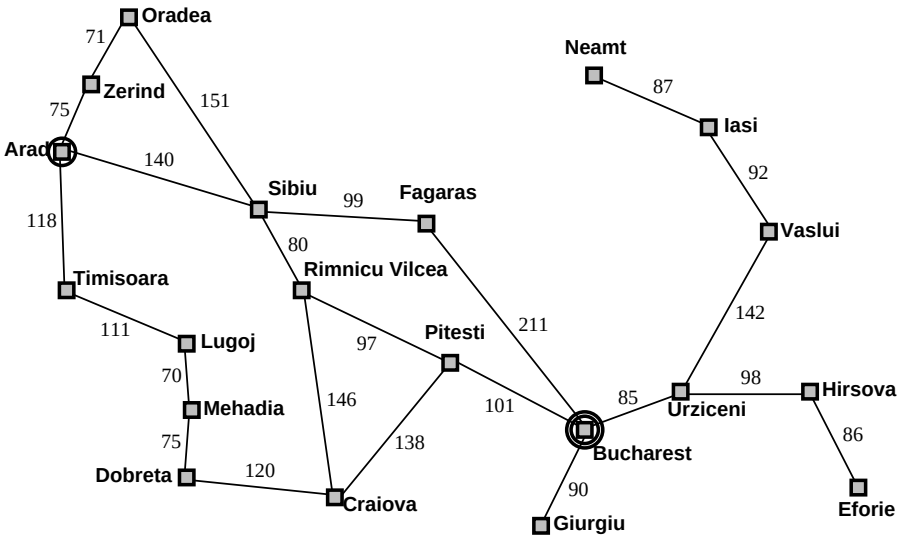


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

How to derive straight-line distance by relaxation?

- Problem $\mathcal{P}$: Route finding.
- Simpler problem $\mathcal{P}'$:

Relaxation in Route-Finding

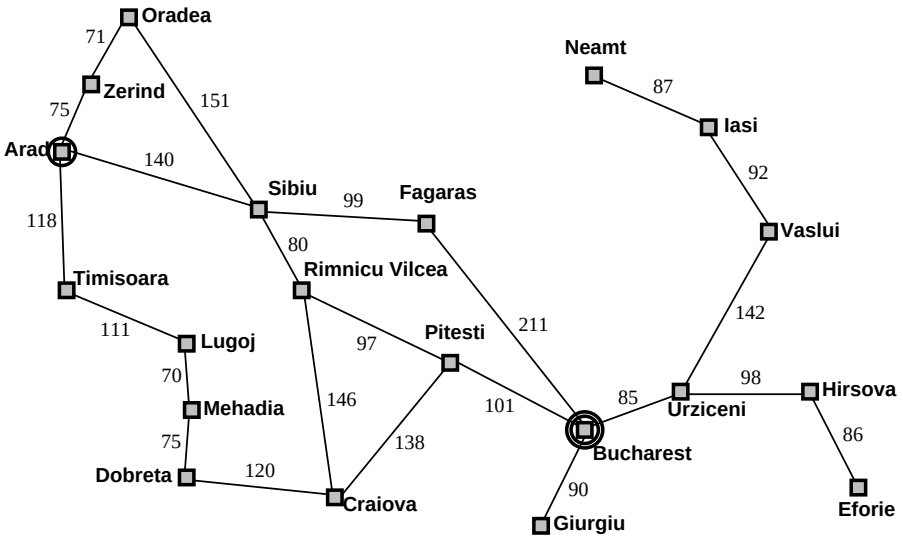


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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 $\mathcal{P}'$:

Relaxation in Route-Finding

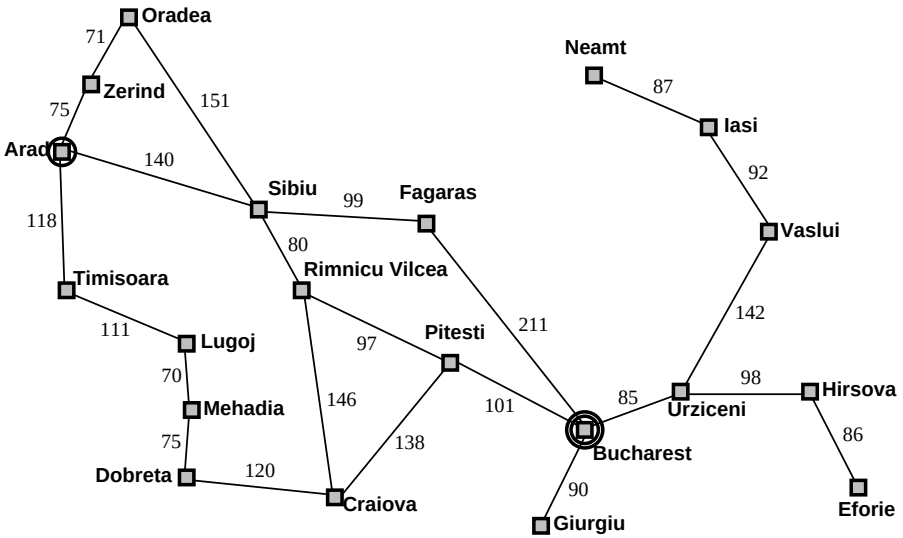


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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:

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.

"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:

"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?

"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.

"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.)

"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.

Agenda

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

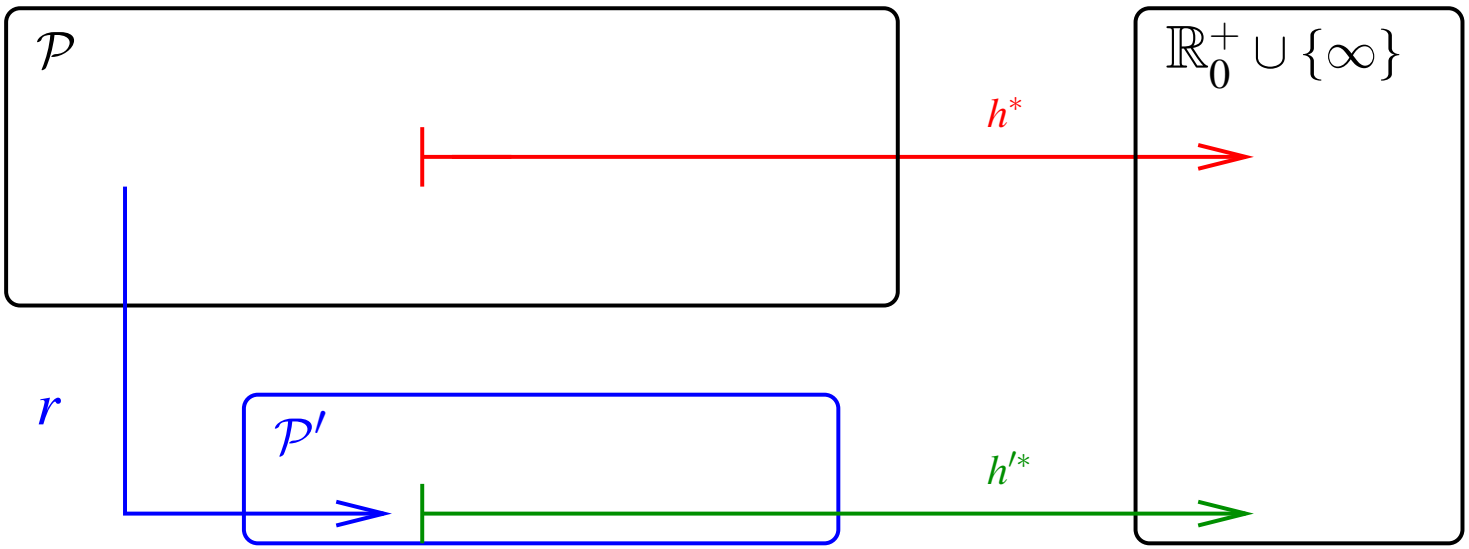
Definition (Relaxation). *Let $h^* : \mathcal{P} \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ be a function. A relaxation of h^* is a triple $\mathcal{R} = (\mathcal{P}', r, h'^*)$ where $\mathcal{P}'$ is an arbitrary set, and $r : \mathcal{P} \mapsto \mathcal{P}'$ and $h'^* : P' \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ 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 $h'^*(r(\Pi))$.

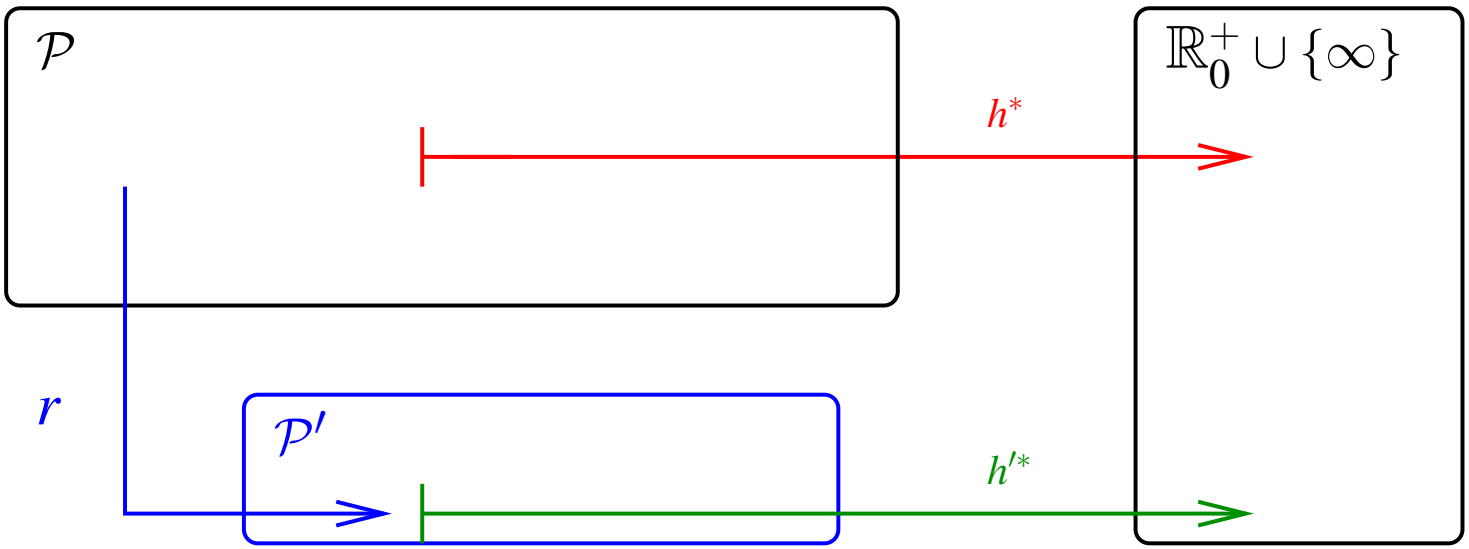
Relaxations: Illustration



Example route-finding:

- Problem $\mathcal{P}$: Route finding.
- Simpler problem $\mathcal{P}'$:

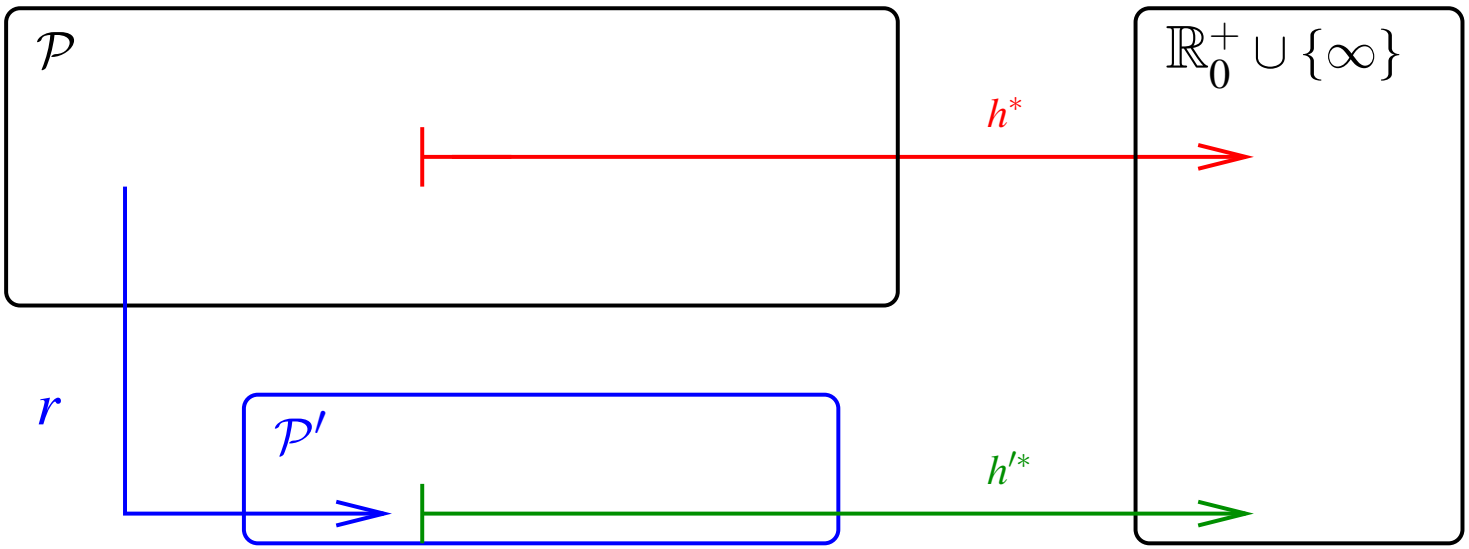
Relaxations: Illustration



Example route-finding:

- Problem $\mathcal{P}$: Route finding.
- Simpler problem $\mathcal{P}'$: Route finding for birds.
- Perfect heuristic h'^* for $\mathcal{P}'$:

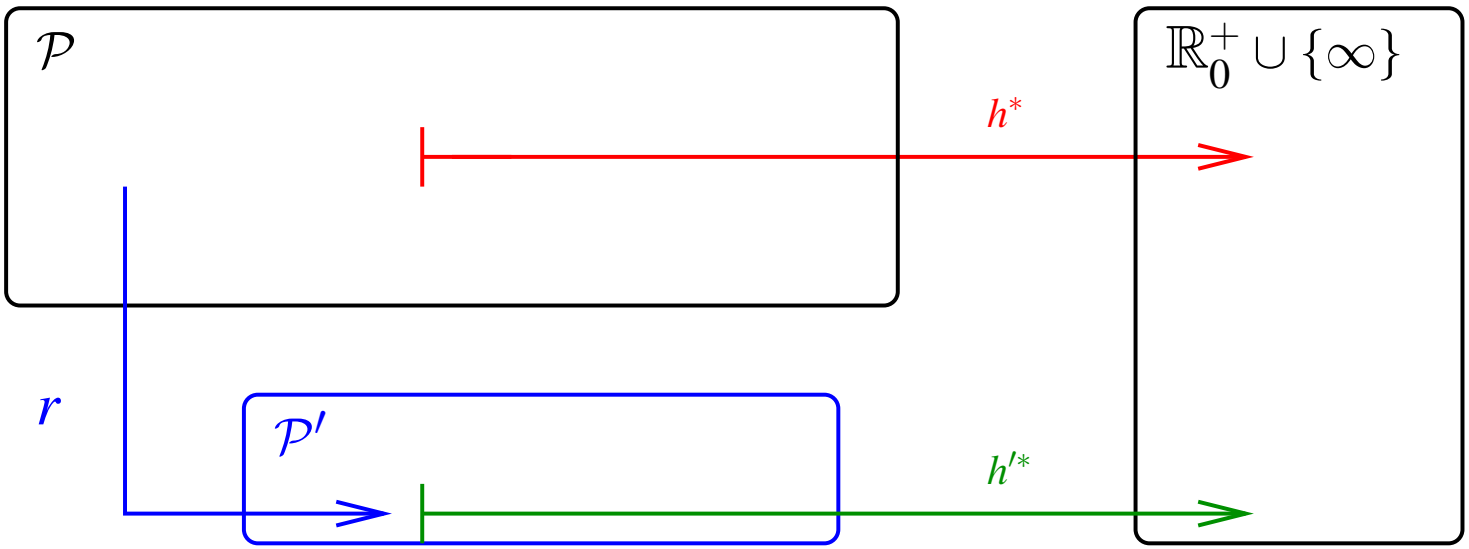
Relaxations: Illustration



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:

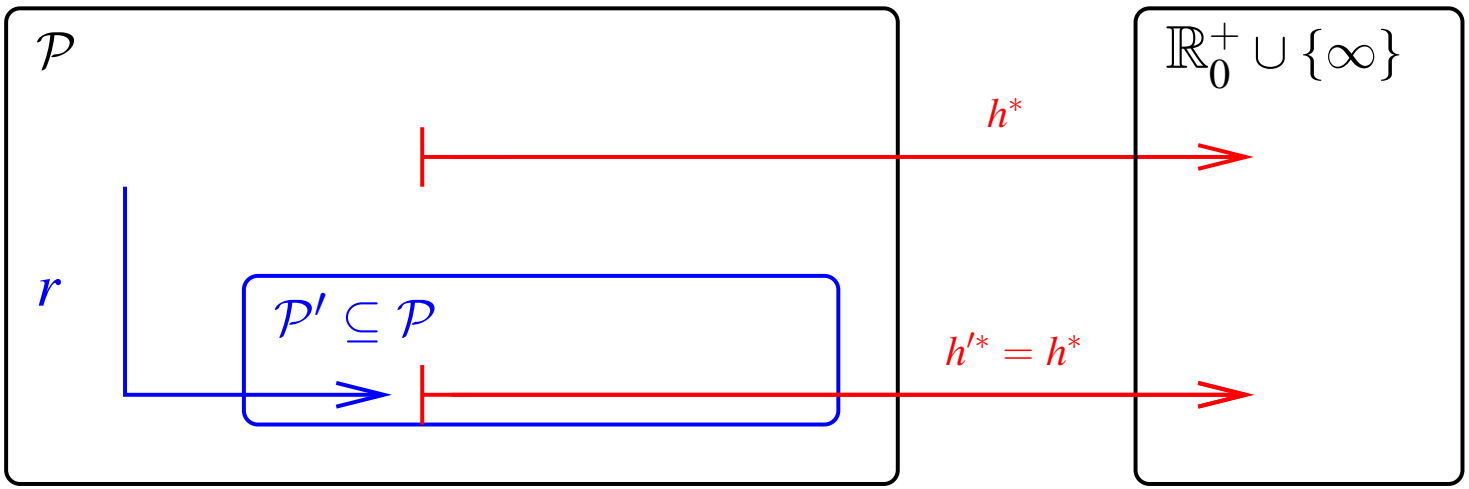
Relaxations: Illustration



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.

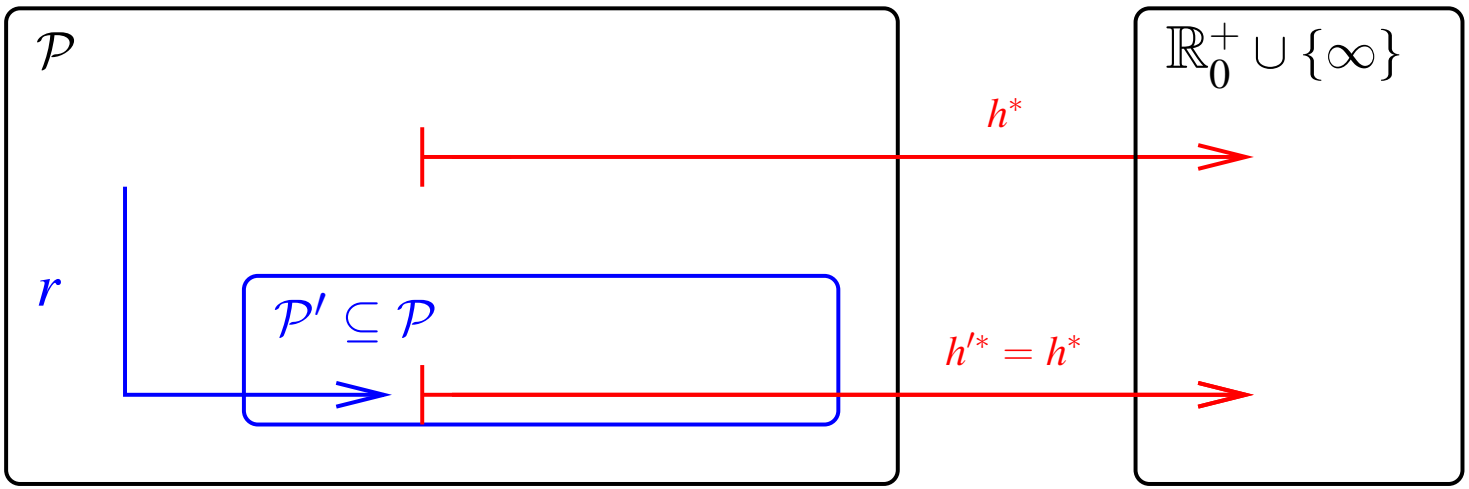
Native Relaxations: Illustration



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:

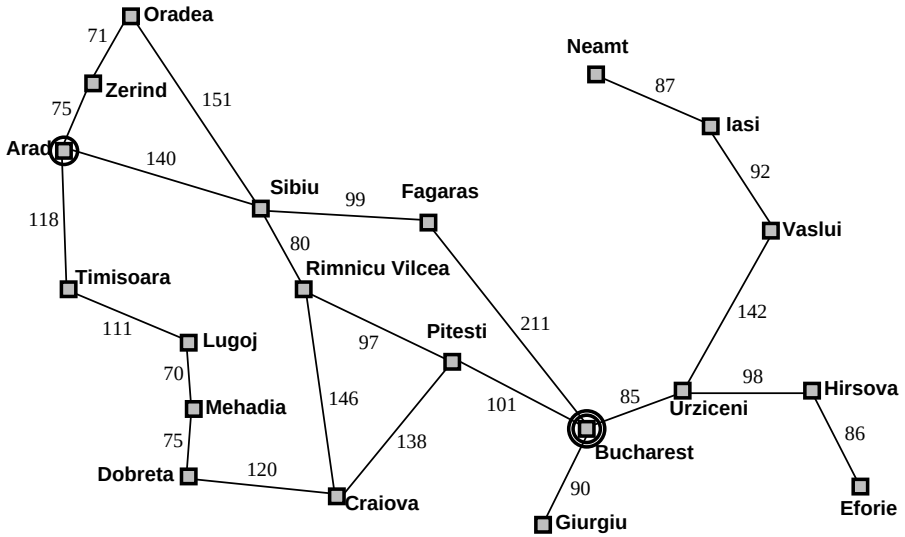
Native Relaxations: Illustration



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

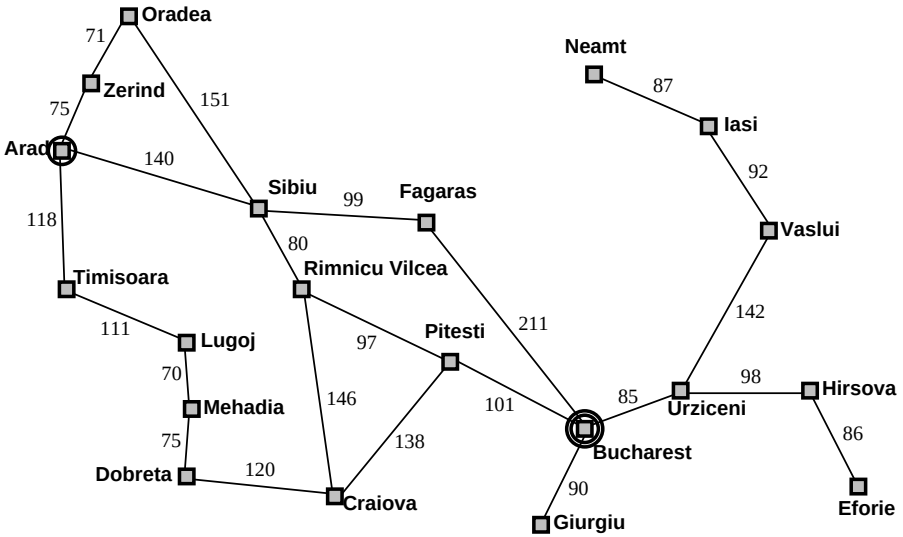


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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

- Native?

Relaxation in Route-Finding: Properties

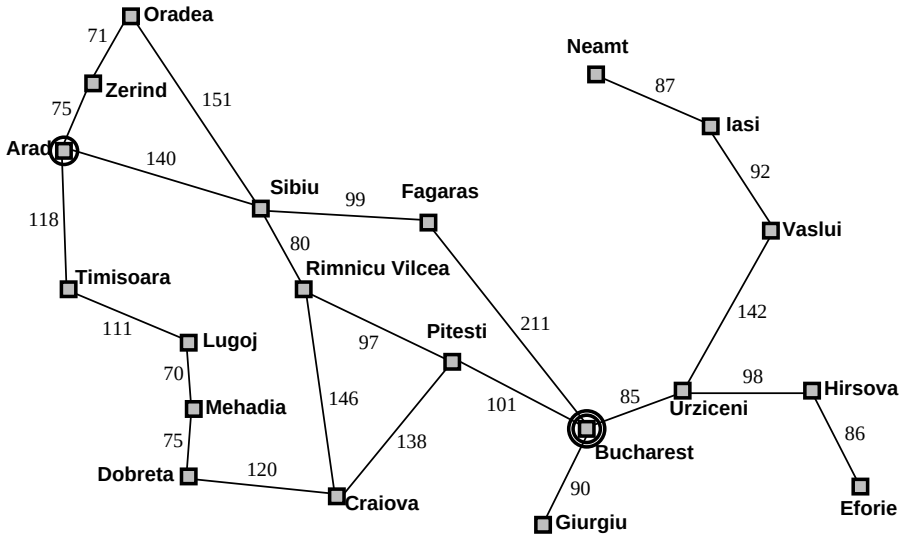


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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?

Relaxation in Route-Finding: Properties

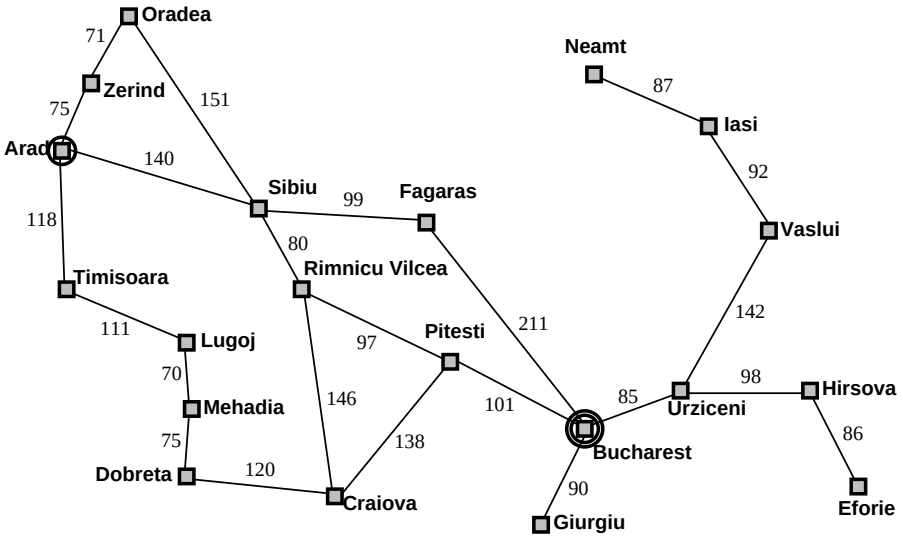


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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?

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).

What shall we do with the drunken 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 shall we do with the drunken 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?

"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?

"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?

"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).

"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 drunken relaxation?

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