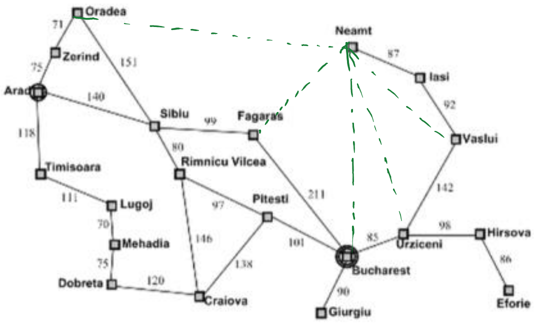
Relaxation

Method for automatically constructing heuristics in planning models

→ 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



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.
  Change connections to fully connected graph

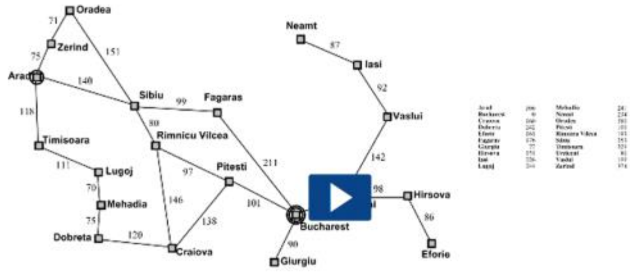
Definition (Relaxation). *Let $h^* : \mathcal{P} \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ be a function. A relaxation of h^* is a triple $\mathcal{R} = (\mathcal{P}', r, h'^*)$ where $\mathcal{P}'$ is an arbitrary set, and $r : \mathcal{P} \mapsto \mathcal{P}'$ and $h'^* : P' \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ 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 $\mathcal{P}' \subseteq \mathcal{P}$ and $h'^* = h^*$;* Simplified Problem can be constructed in the original Problem
- *efficiently constructible if there exists a polynomial-time algorithm that, given $\Pi \in \mathcal{P}$, computes $r(\Pi)$;* Simplification is not expensive
- *efficiently computable if there exists a polynomial-time algorithm that, given $\Pi' \in \mathcal{P}'$, computes $h'^*(\Pi')$.* Calculating heuristic is not expensive

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 $h'^*(r(\Pi))$.

Relaxation in Route-Finding: Properties



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).

"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. Empty pre and del lists still STRIPS problem
- Efficiently constructible? Yes. Remove pre and del
- Efficiently computable? Generally no, if multiple goals can be reached with single actions then NP-HARD

What shall we do with the relaxation?

Approximate h^* using goal counting

Delete Relaxation

For STRIPS problems: drop delete lists

Π^+: Delete relaxed STRIPS planning task

h^+: Optimal Delete relaxation Heuristic

s^+: State in Π^+

- Anything that becomes true in Π^+ stays true
- h^+ always admissible, but hard to compute

State Dominance

S_1 dominates S_2 if $S_2 \subset S_1$

• All True facts in S_2 are present in S_1

Greedy Relaxed Planning for Π_s^+

$s^+ := s; \vec{a}^+ := \langle\rangle$
while $G \not\subseteq s^+$ **do**: While not in goal
  if $\exists a \in A$ s.t. $pre_a \subseteq s^+$ and $appl(s^+, a^+) \neq s^+$ **then**
    select one such a Apply an action satisfied by Preconditions
    $s^+ := appl(s^+, a^+); \vec{a}^+ := \vec{a}^+ \circ \langle a^+ \rangle$ that adds new facts
  else return "Π_s^+ is unsolvable" **endif** If no new facts can be added then return unsolvable
endwhile
return $\vec{a}^+$

- Safe
- goal aware
- non admissible

Approximations to h^+ for multiple objectives

h^{add} : calculate cost of achieving each individual objective in relaxed problem, and sum the total cost

h^{max} : return cost of achieving most expensive objective in relaxed problem.

$h^{max} \leq h^+ \leq h^*$ admissible

$h^{add} \geq h^+$ may not be admissible

Bellman-Ford

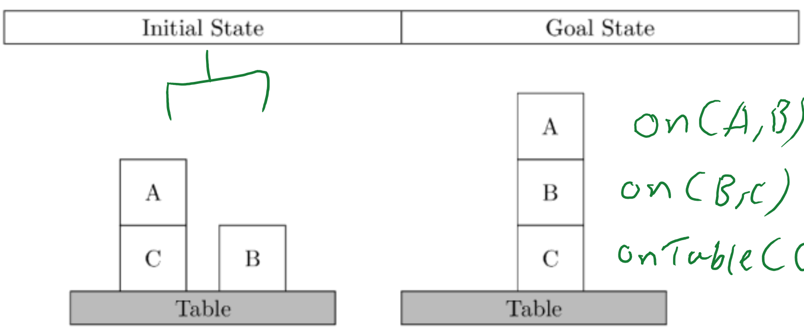
Bellman-Ford variant computing h^{add} for state s

new table $T_0^{add}(g)$, for $g \in F$
For all $g \in F$: $T_0^{add}(g) := \begin{cases} 0 & g \in s \\ \infty & \text{otherwise} \end{cases}$
fn $c_i(g) := \begin{cases} T_i^{add}(g) & |g| = 1 \\ \sum_{g' \in g} T_i^{add}(g') & |g| > 1 \end{cases}$
fn $f_i(g) := \min[c_i(g), \min_{a \in A, g \in add_a} c(a) + c_i(pre_a)]$
do forever:
  new table $T_{i+1}^{add}(g)$, for $g \in F$
  For all $g \in F$: $T_{i+1}^{add}(g) := f_i(g)$
  if $T_{i+1}^{add} = T_i^{add}$ **then** stop **endif**
  $i := i + 1$
enddo

Add column for every predicate
Set True facts to 0 and ∞ otherwise
Until convergence:
Apply every possible action that adds new facts / decreases value of a fact
Value of fact = Action cost + Sum of values for preconditions on prev line
(for h^{max}: value = Action cost + max Precondition value)

→ Basically the same algorithm works for h^{max}, just change $\sum$ for max

HAVE TO WAIT UNTIL CONVERGENCE !! because values can update in later iterations



on(A,B)
on(B,C)
onTable(C)

h^add table

i	clear(a)	clear(b)	clear(c)	handempty()	holding(a)	holding(b)	holding(c)	on(a, a)	**on(a, b)**	on(a, c)	on(b, a)	on(b, b)	**on(b, c)**	on(c, a)	on(c, b)	on(c, c)	ontable(a)	ontable(b)	**ontable(c)**
0	0	0	inf	0	inf	inf	inf	inf	inf	0	inf	inf	inf	inf	inf	inf	inf	0	0
1	0	0	1	0	1	1	inf	inf	inf	0	inf	inf	inf	inf	inf	inf	inf	0	0
2	0	0	1	0	1	1	2	2	2	0	2	2	3	inf	inf	inf	2	0	0
3	0	0	1	0	1	1	2	2	2	0	2	2	3	3	3	4	2	0	0
4	0	0	1	0	1	1	2	2	**2**	0	2	2	**3**	3	3	4	2	0	**0**
h^add	5																		

h^max table

i	clear(a)	clear(b)	clear(c)	handempty()	holding(a)	holding(b)	holding(c)	on(a, a)	**on(a, b)**	on(a, c)	on(b, a)	on(b, b)	**on(b, c)**	on(c, a)	on(c, b)	on(c, c)	ontable(a)	ontable(b)	**ontable(c)**
0	0	0	inf	0	inf	inf	inf	inf	inf	0	inf	inf	inf	inf	inf	inf	inf	0	0
1	0	0	1	0	1	1	inf	inf	inf	0	inf	inf	inf	inf	inf	inf	inf	0	0
2	0	0	1	0	1	1	2	2	2	0	2	2	2	inf	inf	inf	2	0	0
3	0	0	1	0	1	1	2	2	2	0	2	2	2	3	3	3	2	0	0
4	0	0	1	0	1	1	2	2	**2**	0	2	2	**2**	3	3	3	2	0	**0**
h^max	2																		