Relaxation

Heuristics

General procedure for obtaining a heuristic

Solve an easier version of the problem. Two common methods:

- **relaxation:** consider less constrained version of the problem
- **abstraction:** consider smaller version of real problem

Both have been very successfully applied in planning (separately and together).

Today relaxation:

General idea = (Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

From the last time:

A heuristic h is called

- safe if $h^*(\sigma) = \infty$ for all $\sigma \in \Sigma$ with $h(\sigma) = \infty$
- goal-aware if $h(\sigma) = 0$ for all goal nodes $\sigma \in \Sigma$
- admissible if $h(\sigma) \leq h^*(\sigma)$ for all nodes $\sigma \in \Sigma$
- consistent if $h(\sigma) \leq h(\sigma') + 1$ for all nodes $\sigma, \sigma' \in \Sigma$ such that σ' is a successor of σ

Precision matters

Given two admissible heuristics h_1, h_2, if $h_2(\sigma) \geq h_1(\sigma)$ for all search nodes σ, then h_2 dominates h_1 and is better for optimizing search

Combining admissible heuristics

For any admissible heuristics $h_1, \ldots, h_k$,

$$h(\sigma) = \max_{i=1}^{k}\{h_i(\sigma)\}$$

is also admissible and dominates all individual h_i

Towards domain-independent agents: How to get heuristics automatically?

Domain specific heuristic examples:

- Straight-line heuristic (route planning): Ignore the fact that one must stay on roads.
- Manhattan heuristic (15-puzzle): Ignore the fact that one cannot move through occupied tiles.

We want to apply the idea of relaxations to planning.

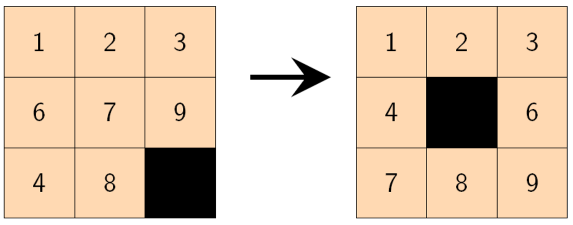
Informally, we want to ignore bad side effects of applying actions.

In STRIPS, good and bad effects are easy to distinguish:

Effects that make atoms true are good (add effects).

Effects that make atoms false are bad (delete effects).

Idea for the heuristic: Ignore all delete effects.



How can we use relaxations for heuristic planning in practice?

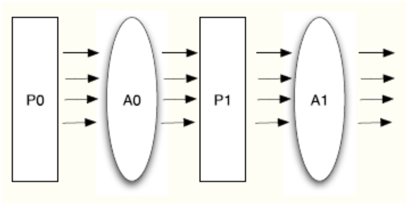
Different possibilities:

- Implement an optimal planner for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.
  $\rightsquigarrow h^+$ heuristic *(not that realistic. why?)*
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.
  $\rightsquigarrow h_{\text{max}}$ heuristic, h_{add} heuristic
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but "reasonable".
  $\rightsquigarrow h_{\text{FF}}$ heuristic

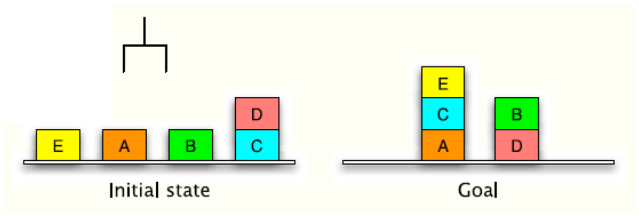
Relaxed planning graph & generic template for heuristics

- Build a layered reachability graph $P_0, A_0, P_1, A_1, \ldots$

$$\begin{aligned} P_0 &= \{p \in I\} \\ A_i &= \{a \in A \mid \text{pre}(a) \subseteq P_i\} \\ P_{i+1} &= P_i \cup \{p \in \text{add}(a) \mid a \in A_i\} \end{aligned}$$



- Terminate when $G \subseteq P_i$



Blackboard: Relaxed planning graph for this example (*)

s = {a,b}, *g*={b,e,f}

$a_1 = \langle\{a\},\varnothing,\{b,c\}\rangle$

$a_2 = \langle\{a,c\},\varnothing,\{d\}\rangle$

$a_3 = \langle\{b,c\},\varnothing,\{e\}\rangle$

$a_4 = \langle\{b,d\},\varnothing,\{e,f,g\}\rangle$

1. $\{on(E,Table), clear(E), on(A,Table), clear(A), on(B,Table), clear(B), on(C,Table), on(D,C), clear(D), holding(NIL)\}$
2. $\{\ldots, holding(E), holding(A), holding(B), holding(D), clear(C)\}$
3. $\{\ldots, holding(C), on(E,A), on(A,E), \ldots\}$
4. $\{\ldots, on(C,A), \ldots\}$

Computing heuristics from relaxed planning graphs

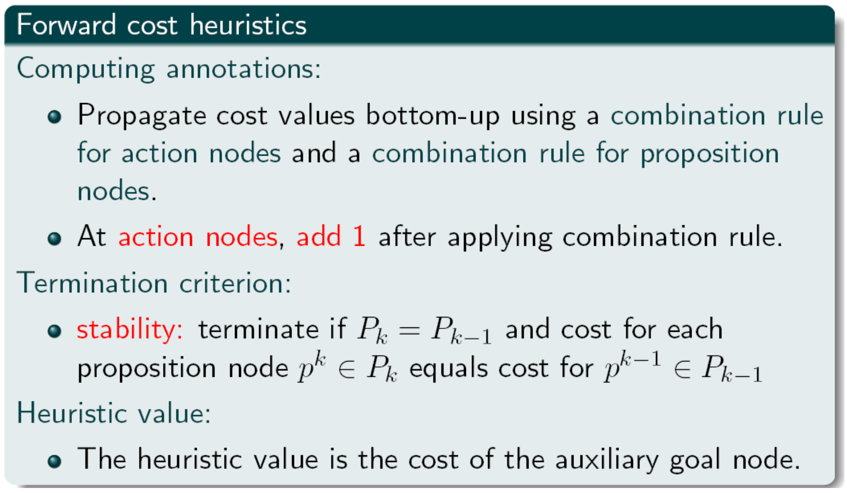
```
def generic-rpg-heuristic($\langle P, I, O, G\rangle, s$):
    $\Pi^+ := \langle P, s, O^+, G\rangle$
    for $k \in \{0, 1, 2, \ldots\}$:
        rpg := $RPG_k(\Pi^+)$
        if $G \subseteq P_k$:
            Annotate nodes of rpg.
            if termination criterion is true:
                return heuristic value from annotations
        else if $k = |P|$:
            return $\infty$
```

Many planning heuristics fit the generic template: **hmax, hadd, hFF**

Forward cost heuristics

The simplest relaxed planning graph heuristics are forward cost heuristics. Examples: **hmax, hadd**

- Here, node annotations are cost values (natural numbers).
- The cost of a node estimates how expensive (in terms of required operators) it is to make this node true.



Forward cost heuristics

Computing annotations:

- Propagate cost values bottom-up using a combination rule for action nodes and a combination rule for proposition nodes.
- At action nodes, add 1 after applying combination rule.

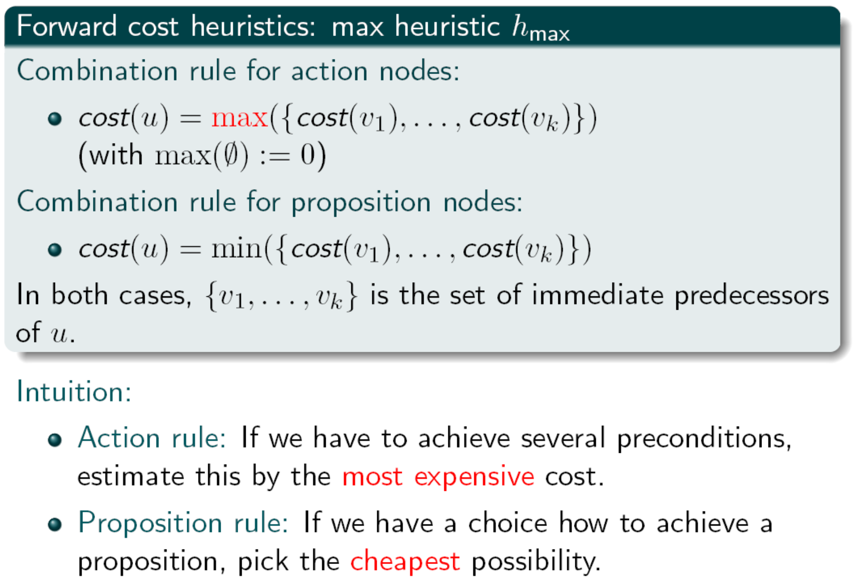
Termination criterion:

- stability: terminate if $P_k = P_{k-1}$ and cost for each proposition node $p^k \in P_k$ equals cost for $p^{k-1} \in P_{k-1}$

Heuristic value:

- The heuristic value is the cost of the auxiliary goal node.

The max heuristic



Forward cost heuristics: max heuristic h_{max}

Combination rule for action nodes:

- $\mathit{cost}(u) = \max(\{\mathit{cost}(v_1), \dots, \mathit{cost}(v_k)\})$ (with $\max(\emptyset) := 0$)

Combination rule for proposition nodes:

- $\mathit{cost}(u) = \min(\{\mathit{cost}(v_1), \dots, \mathit{cost}(v_k)\})$

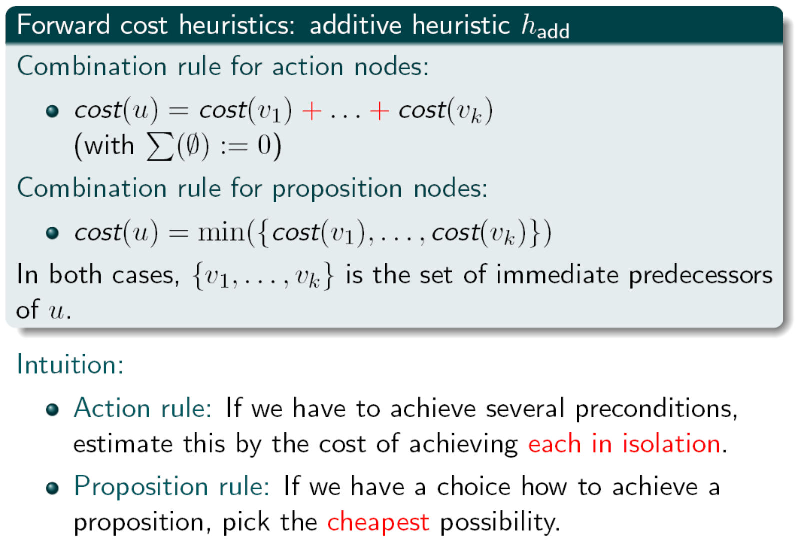
In both cases, $\{v_1, \dots, v_k\}$ is the set of immediate predecessors of u.

Intuition:

- Action rule: If we have to achieve several preconditions, estimate this by the most expensive cost.
- Proposition rule: If we have a choice how to achieve a proposition, pick the cheapest possibility.

EXAMPLE (*) – calculate hmax heuristic

The additive heuristic



Forward cost heuristics: additive heuristic h_{add}

Combination rule for action nodes:

- $\mathit{cost}(u) = \mathit{cost}(v_1) + \dots + \mathit{cost}(v_k)$ (with $\sum(\emptyset) := 0$)

Combination rule for proposition nodes:

- $\mathit{cost}(u) = \min(\{\mathit{cost}(v_1), \dots, \mathit{cost}(v_k)\})$

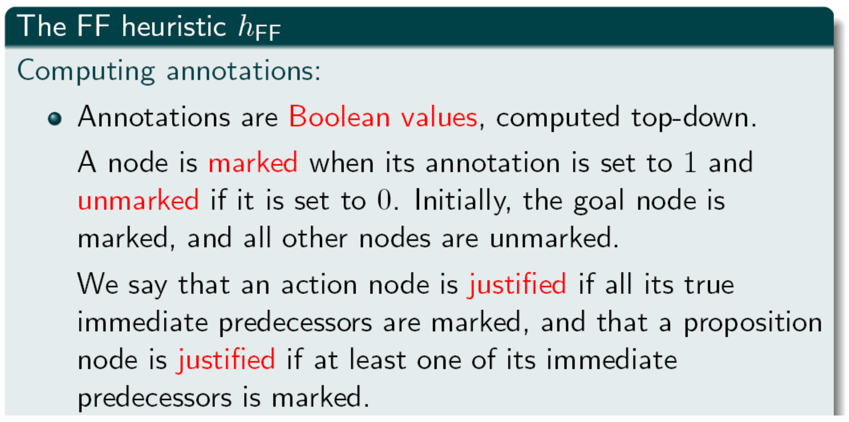
In both cases, $\{v_1, \dots, v_k\}$ is the set of immediate predecessors of u.

Intuition:

- Action rule: If we have to achieve several preconditions, estimate this by the cost of achieving each in isolation.
- Proposition rule: If we have a choice how to achieve a proposition, pick the cheapest possibility.

EXAMPLE (*) – calculate hadd heuristic

FF heuristic



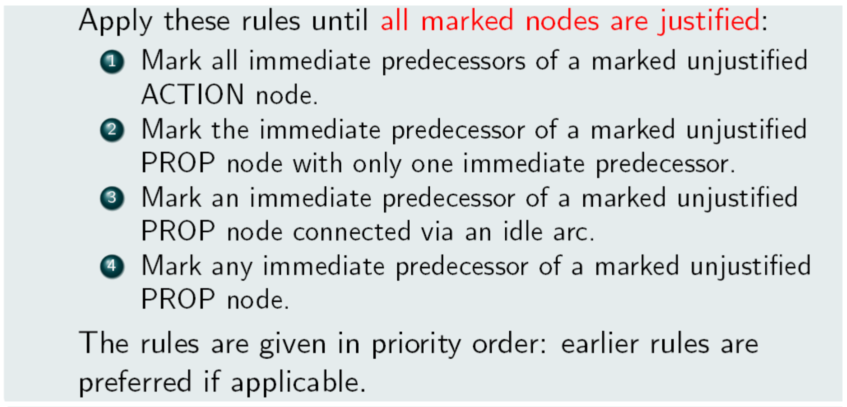
The FF heuristic h_{FF}

Computing annotations:

- Annotations are Boolean values, computed top-down.

  A node is marked when its annotation is set to 1 and unmarked if it is set to 0. Initially, the goal node is marked, and all other nodes are unmarked.

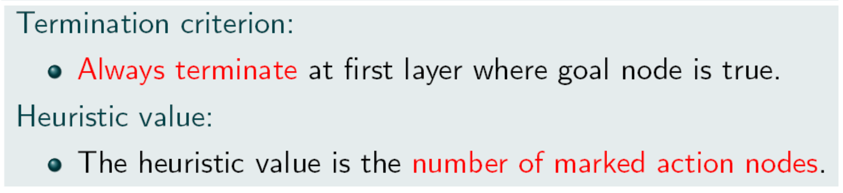
  We say that an action node is justified if all its true immediate predecessors are marked, and that a proposition node is justified if at least one of its immediate predecessors is marked.



Apply these rules until all marked nodes are justified:

1. Mark all immediate predecessors of a marked unjustified ACTION node.
2. Mark the immediate predecessor of a marked unjustified PROP node with only one immediate predecessor.
3. Mark an immediate predecessor of a marked unjustified PROP node connected via an idle arc.
4. Mark any immediate predecessor of a marked unjustified PROP node.

The rules are given in priority order: earlier rules are preferred if applicable.



Termination criterion:

- Always terminate at first layer where goal node is true.

Heuristic value:

- The heuristic value is the number of marked action nodes.

EXAMPLE (*) – calculate hadd heuristic