Course "Automated Planning: Theory and Practice"
Chapter 12: The Relaxation Principle: a closer look

Teacher: Marco Roveri - marco.roveri@unitn.it
M.S. Course: Artificial Intelligence Systems (LM)
A.A.: 2025-2026
Where: DISI, University of Trento
URL: https://shorturl.at/A81hf

Last updated: Wednesday 29th October, 2025

Terms of Use and Copyright

The material (text, figures) in these slides is authored by Jonas Kvarnström and Marco Roveri.

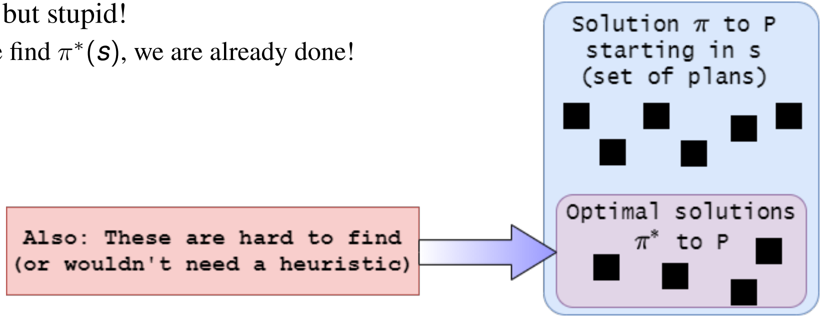
The Problem

- We have
  - An arbitrary planning problem $P = \langle \Sigma, s_0, S_g \rangle$
- Suppose we want:
  - A way to compute admissible heuristics $h(s)$
    - Given P and some states s in the search space

What do we do?
Where do we start?
How do we think?

Fundamental ideas

- One obvious method: Every time we need $h(s)$ for some state s ...
  1. Solve P optimally starting from s, resulting in an *actual* solution $\pi^*(s)$
  2. Let $h(s) = h^*(s) = cost(\pi^*(s))$
     - Admissible - why?
- Obvious, but stupid!
  - If we find $\pi^*(s)$, we are already done!



Fundamental ideas (cont.)

- Let's modify the obvious idea:
  - Change/Transform P to make it *easy* (quick) to solve
    - But make sure optimal solution cannot become more expensive!
    - Example: Add more goal states to S_g $\Longrightarrow$ more ways to reach them!

Relaxation will be one specific way of (1) finding a simplified transformation, and (2) proving "not more expensive"!

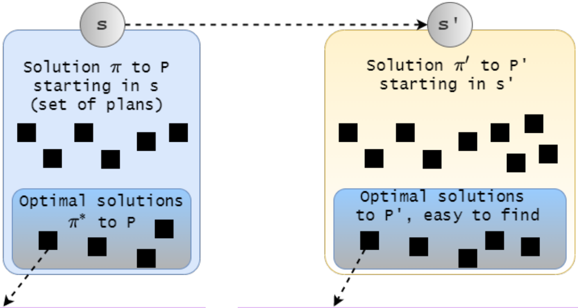
  - Compute an admissible heuristic:
    - Solve the modified planning problem optimally
    - $h(s) =$ cost of optimal solution for modified problem

      $\leq$

      $h^*(s) =$ cost of optimal solution for original problem
    - Definition of admissibility!
  - Preferably
    - Keep $h(s)$ as close as possible to $h^*(s)$ - we want *strong cost information*!

Fundamental ideas (cont.)

- More formally:
  - Before planning, find a simpler problem P', such that in every state s (of P):
    - We can quickly transform s into a state s' for P'
    - We can quickly find an optimal solution π' for P' starting in s'
    - The solution is never more expensive: $cost(\pi') \leq cost(\pi^*)$



π^*: an optimal *plan* for P

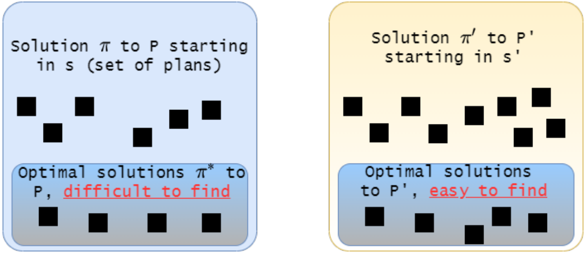
π': solution of another problem;
we only use it to compute a heuristic

Fundamental ideas (cont.)

- During planning:
  - Every time we need $h(s)$ for some state s:
    - Transform s into s'
    - Quickly solve problem P' optimally starting in s', resulting in solution π' - for the *transformed* problem
    - Let $h(s) = cost(\pi')$
    - Throw away π': It isn't interesting itself
- We then know:
  - $h(s) = cost(\pi'(s')) = cost(\text{optimal-solution}(P')) \leq cost(\text{optimal-solution}(P))$
  - $h(s)$ is admissible!

Fundamental ideas (cont.)

- Important:
  - What we need: $cost(\text{optimal-solution}(P')) \leq cost(\text{optimal-solution}(P))$
  - Could use any transformation, even with completely disjoint solution sets, if we just have a proof that optimal solution to P' are not more expensive!

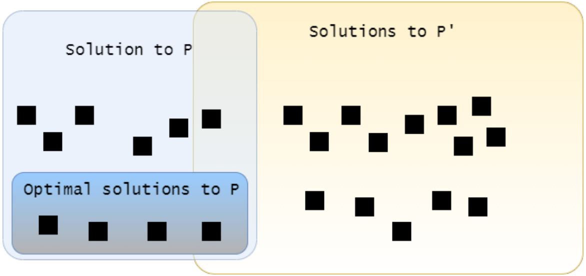


Difficult to find transformations, prove correctness - we need a *method*!

Fundamental ideas (cont.)

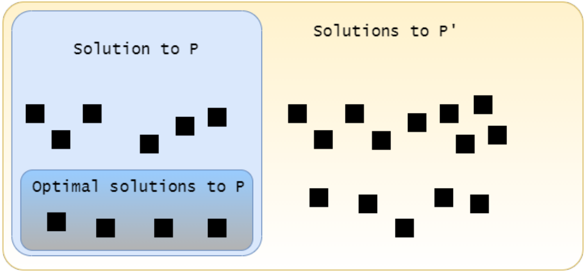
- How to prove $cost(\text{optimal-solution}(P')) \leq cost(\text{optimal-solution}(P))$?
  - Sufficient criterion: One optimal solution to P remains a solution for P'
    - $cost(\text{optimal-solution}(P')) = min\{cost(\pi) | \pi \text{ is any solution to } P'\} \leq cost(\text{optimal-solution}(P))$

Includes the optimal solution to P,
so $min\{...\}$ cannot be greater



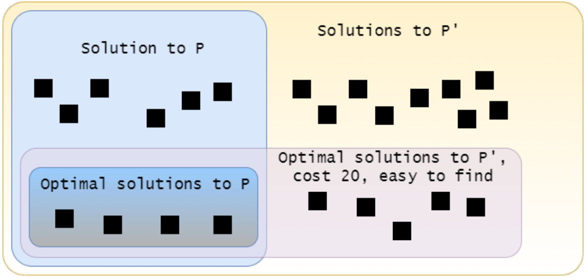
Fundamental ideas (cont.)

- Another sufficient criterion: All solutions to P remain solutions for P'
  - Stronger, but often easier to prove
  - This is called relaxation: P' is a relaxed version of P
  - Relaxes the constraints on what is accepted as a solution:
    The is-solution(plan) test is "expanded, relaxed" to cover additional plans



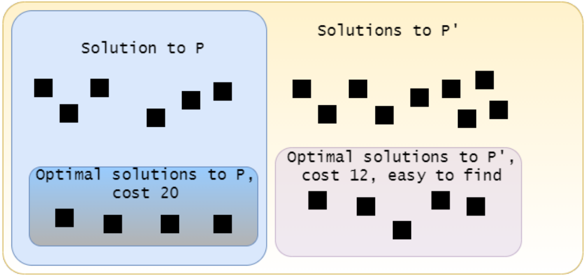
Fundamental ideas (cont.)

- Case I: P' has identical cost (for some starting state s)
  - Unlikely!



Fundamental ideas (cont.)

- Case II: P' has lower cost (for some starting state s)



Relaxation for Planning Problems

- A classical planning problem $P = \langle \Sigma, s_0, S_g \rangle$ has a set of solutions
  - $Solutions(P) = \{\pi | \pi$ is an executable action sequence leading from s_0 to some state in $S_g\}$
- Suppose that:
  - $P = \langle \Sigma, s_0, S_g \rangle$ is a classical planning problem
  - $P' = \langle \Sigma', s_0', S_g' \rangle$ is another classical planning problem
  - $Solutions(P) \subseteq Solutions(P')$
- Then (and only then): P' is a relaxation of P!

Solutions for P:

Sol1, cost 10 — Optimal in P

Sol2, cost 12

Sol3, cost 27

Solutions for P':

Sol1, cost 10

Sol2, cost 12

Sol3, cost 27

Sol4, cost 8

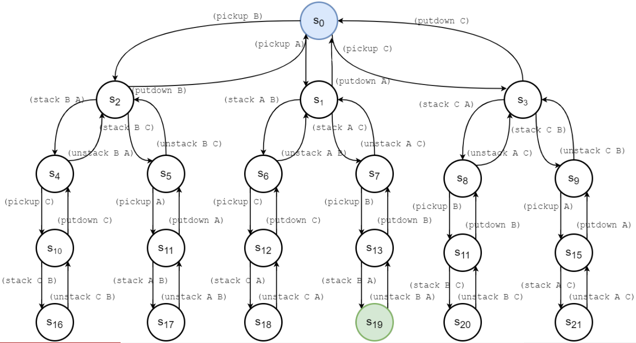
Sol5, cost 42

All old solutions remain solutions!

Now sol4 is optimal in P'

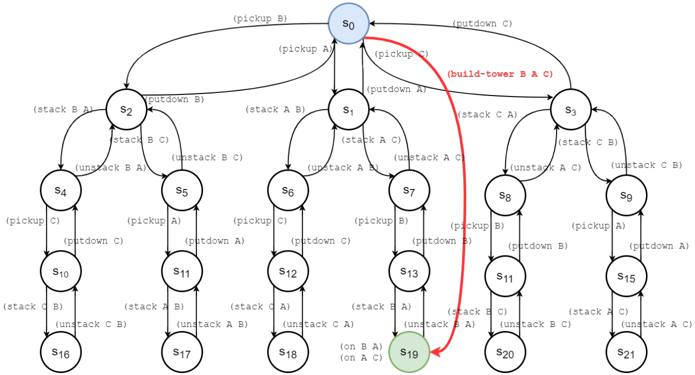
Relaxation Example: Basis

- A simple planning problem (domain + instance)
  - Blocks world 3 blocks
  - Initially all blocks on the table
  - Goal: `(and (on B A) (on A C))` (only satisfied in s_{19})
  - Solutions: All paths from init to goal (infinitely many - can have cycles)



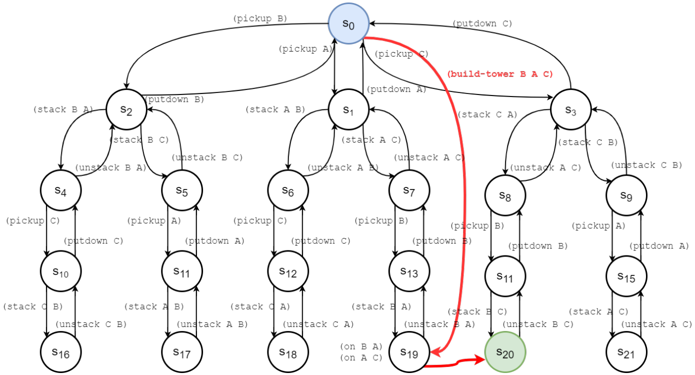
RELAXATION EXAMPLE (CONT.)

- Adding new actions
  - All old solutions still valid, but new solutions may exists
  - Modified the STS by adding new edges/transitions
  - This particular example: shorter solutions appear!



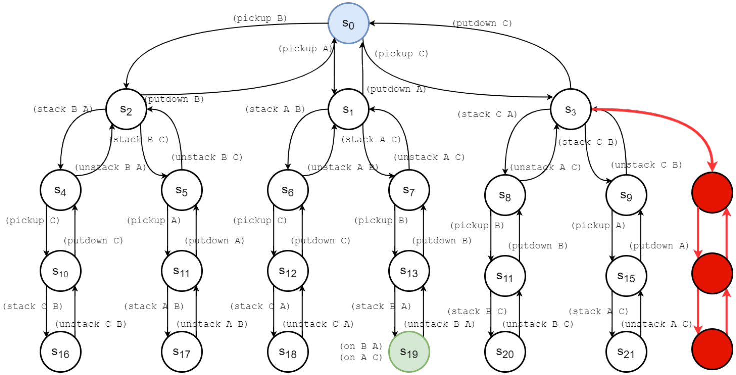
Relaxation Example (cont.)

- Adding new actions
  - In other cases, the new actions may not "help"!
  - New solutions ($s_0 \rightarrow s_{19} \rightarrow s_{20}$) are *longer* as well as *more expensive*
  - Still relaxation!



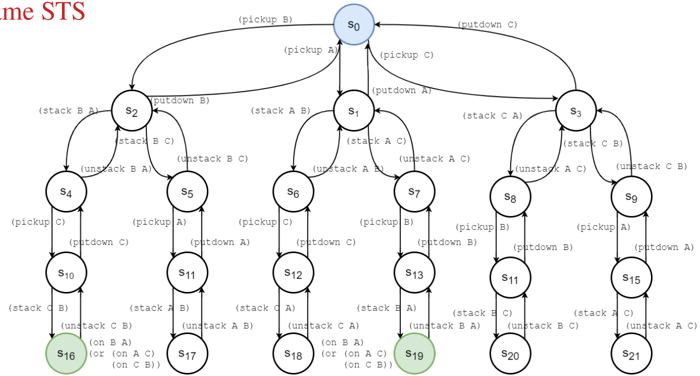
Relaxation Example (cont.)

- Adding new actions
  - May lead to previously unreachable states
  - May not result in new solutions at all
  - Still relaxation! Old solutions remain!



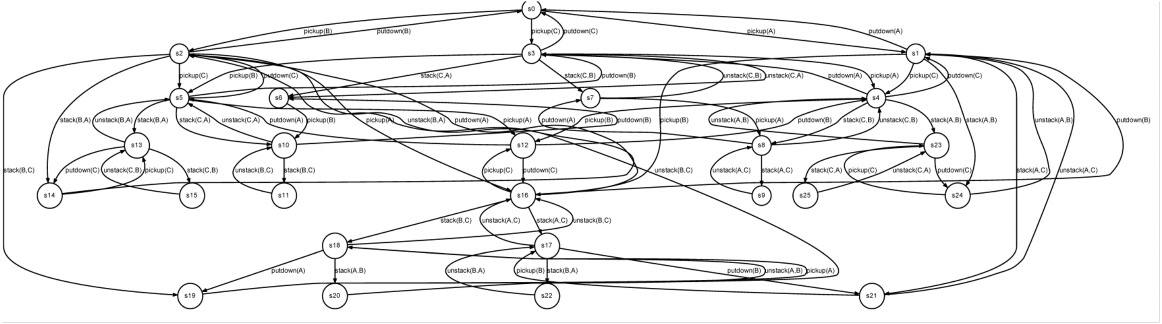
Relaxation Example (cont.)

- Adding goal states
  - New goal formula: `(and (on B A) (or (on A C) (on C B)))`
  - All old solutions still valid, but new solutions may exist
  - This particular example: Optimal solution from s_0 retains the same lenght
  - Retain the same STS



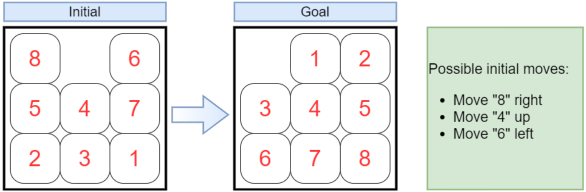
Relaxation Example (cont.)

- Ignoring state variables
  - Ignore `(handempty)` fact in preconditions and effects
  - Different state space, no simple addition or removal, but all the old solutions (action sequences) lead from s'_0 to new goal states in S'_g!
    - 22 reachable states $\Longrightarrow$ 26
    - 42 transitions $\Longrightarrow$ 72



Relaxation Example (cont.)

- Weakening preconditions of existing actions



- Precondition relaxation: Tiles can be moved across each other
  - Now we have 21 possible first moves: New transitions added to the STS
- All old solutions are still valid, but new ones are added
  - To move "8" into place:
  - Two steps to the right, two steps down, ends up in the same place as "1"

Essentially the same as adding actions!

Result in new transitions!

Can still be solved through search!
The optimal solution for the relaxed 8-puzzle
can never be more expensive than the optimal solution for the original 8-puzzle!

Relaxation Heuristics: Summary

- Relaxation: One general principle for designing admissible heuristics for optimal planning
  - Find a way of transforming planning problems, so that given a problem instance P:
    - Computing its transformation P' is easy (polynomial)
    - Finding an optimal solution to P' is easier than for P
    - All solutions to P are solutions to P', but the new problem can have additional solutions as well
  - Then the cost of an optimal solution to P' is an admissible heuristic for the original problem P

This is only *one* principle!
There are others, *not* based on relaxation!

Search or Direct Computation

- As stated:
  - Compute an actual solution π' for the relaxed problem P'
  - Compute $cost(\pi')$
- Example: The 8-puzzle...
  - Ignore `(blank ?x ?y)` in preconditions and effects
  - Run the problem through an optimal planner
  - Compute the cost of the resulting plan π'

```
(:action move-up
  :parameters (?t ?px ?py ?by)
  :precondition (and
                    (tile ?t) (position ?px) (position ?py) (position ?by)
                    (dec ?by ?py) (blank ?px ?by) (at ?t ?px ?py))
  :effect (and (not (blank ?px ?by)) (not (at ?t ?px ?py))
                (blank ?px ?py) (at ?t ?px ?by)))
```

Search or Direct Computation (cont.)

- But we only use π' to compute its cost!
  - Let's analyze the problem (8-tiles) ...
    - Each piece has to be moved to the intended row
    - Each piece has to be moved to the intended column
    - These are exactly the required actions given the relaxation!
  - $\Longrightarrow$ optimal cost for relaxed problem = sum of Manhattan distances
  - $\Longrightarrow$ admissible heuristic for original problem = sum of Manhattan distances
  - $\Longrightarrow$ Cost of any optimal solution π' can be computed efficiently *without* π':

$$\sum_{p \in pieces} xdistance(p) + ydistance(p)$$

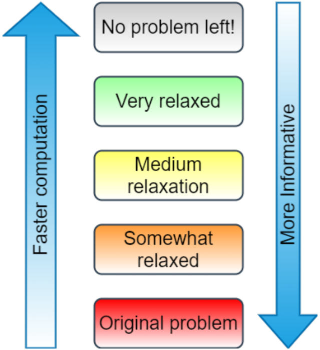
But now we had to analyze the problem:

1. Decide to ignore "blank"
2. Find "sum of manhattan distances"

How do we *automatically* find good relaxations + computation methods? – We will discuss soon!!

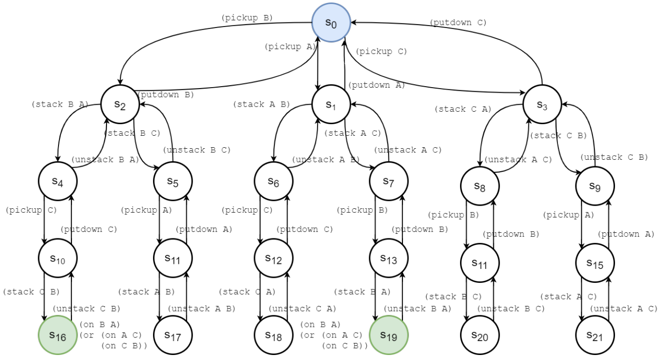
Relaxation Heuristics: Balance

- The reason for relaxation is rapid calculation
  - Shorter solutions are an *unfortunate side effect*:
    Leads to less informative heuristics
  - Relax too much $\implies$ not informative
    - Example: Any piece can teleport into the desired position
      $\implies h(n) =$ number of pieces left to move



Relaxation Heuristics: Important issues!

You cannot "use a relaxed problem as a heuristic".
What would that mean?
You use the cost of an optimal solution to the relaxed problem as a heuristic.



This is the problem!

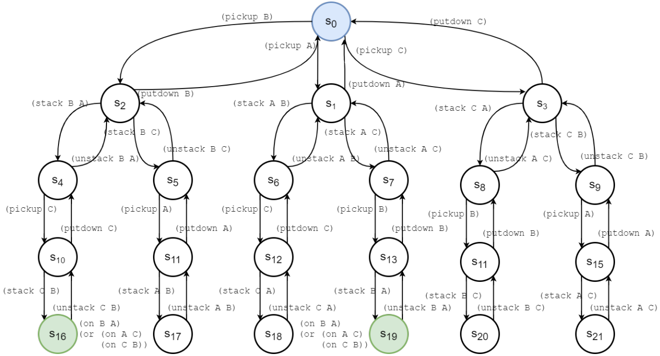
The *problem* is not a *heuristic*!

Relaxation Heuristics: Important issues! (cont.)

Solving the relaxed problem can result in a more expensive solution

$\Longrightarrow$ inadmissible!

You have to solve it <u>optimally</u> to get the admissibility guarantee.

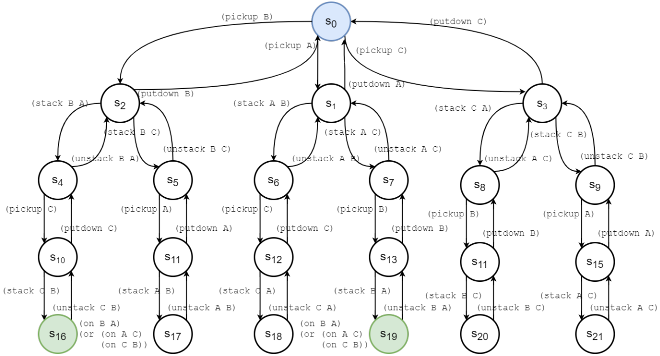


One solution to relaxed problem:

```
(pickup C)
(putdown C)
(pickup B)
(stack B A)
(pickup C)
(stack C B)
```

Relaxation Heuristics: Important issues! (cont.)

You don't just solve the relaxed problem once.
Every time you reach a new state and want to calculate a heuristic,
you have to solve the relaxed problem of getting from that state to the goal.



Calculate:

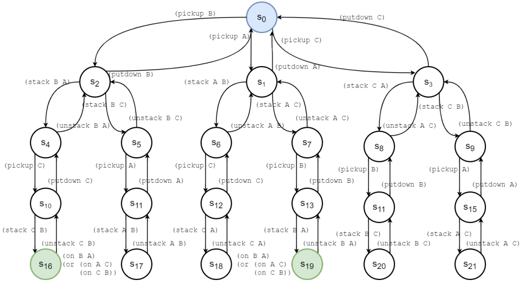
$h(s_0)$

$h(s_1)$, $h(s_2)$, $h(s_3)$

... then for every node you create, depending on the strategy

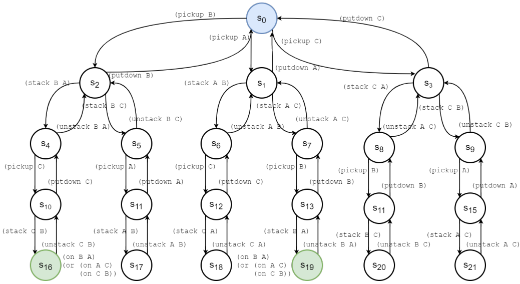
Relaxation Heuristics: Important issues! (cont.)

- Relaxation does not always mean "removing constraints" in the sense of *weakening preconditions* (moving across tiles, removing walls, ...)
- Sometimes we get new *goals*. Sometimes the entire *state space* is transformed.
- Sometimes action *effects* are modified, or some other change is made.
- What defines relaxation: All old solutions are valid, new solutions may exist.



Relaxation Heuristics: Important issues! (cont.)

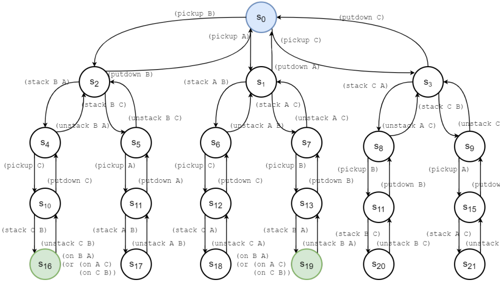
- Relaxation is useful for finding admissible heuristics.
- A heuristics cannot be admissible for some states.
  - Admissible = does not overestimates cost for any state!



Relaxation Heuristics: Important issues! (cont.)

- If you are asked "why is a relaxation heuristic admissible?"
  - Don't answer "because it cannot overestimate costs"!
  - This is the *definition* of admissibility!
- "Why is it admissible?" == "*Why* can't it overestimate costs?"

- Admissible heuristics *can* "lead you astray" and you *can* "visit" suboptimal solutions.
- But with the right search strategy, such as A*, the planner will eventually get around to finding an optimal solution.
  - This is not the case with A* with non-admissible heuristics.



REFERENCES I

[1] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. ISBN 9781608459698. doi: 10.2200/S00513ED1V01Y201306AIM022. URL `https://doi.org/10.2200/S00513ED1V01Y201306AIM022`.

[2] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.

[3] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. URL `http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB`.

[4] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. doi: 10.2200/S00900ED2V01Y201902AIM042. URL `https://doi.org/10.2200/S00900ED2V01Y201902AIM042`.