Course "Automated Planning: Theory and Practice"
Chapter 08: Heuristics: An Overview

Teacher: Marco Roveri - marco.roveri@unitn.it
M.S. Course: Artificial Intelligence Systems (LM)
A.A.: 2023-2024
Where: DISI, University of Trento
URL: https://bit.ly/3zOkGk8

Last updated: Wednesday 18th October, 2023

Terms of Use and Copyright

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

Heuristic Search (repetition)

```
function SEARCH(problem)
    initial-node ← MAKE-INITIAL-NODE(problem)
    open ← {initial-node}
    while (open ≠ ∅) do
        node ← SEARCH-STRATEGY-REMOVE-FROM(open)
        if IS-SOLUTION(node) then
            return EXTRACT-PLAN-FROM(node)
        end if
        for each newnode ∈ SUCCESSORS(node) do
            open ← open ∪ {newnode}
        end for
    end while
    return Failure
end function
```

An *heuristic strategy* bases decisions on:
$\Longrightarrow$ Heuristic value $h(n)$
$\Longrightarrow$ Often other factors e.g. $g(n)$ i.e. the cost of reaching n

Best first search: Greedy, A*, ...
Modifications: IDA*, D*, ...
Simulated annealing, hill climbing, ...

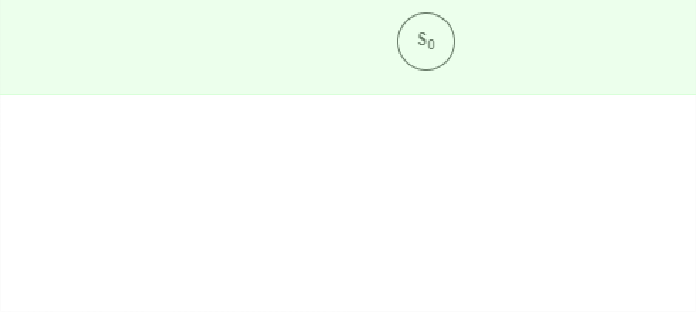
Requires an heuristic function!

How do we *calculate* $h(n)$?

Landmarks,
Pattern databases,
Relaxed plan graph,
...

Example

3 blocks, all on the table in S_0

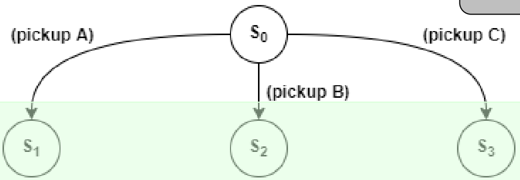


We now have
1 open node,
which is *unexpanded*

Example (cont.)

We visit S_0 and we expand it!

Forward search: node $\approx$ state
so we may write $h(n)$ or $h(s)$

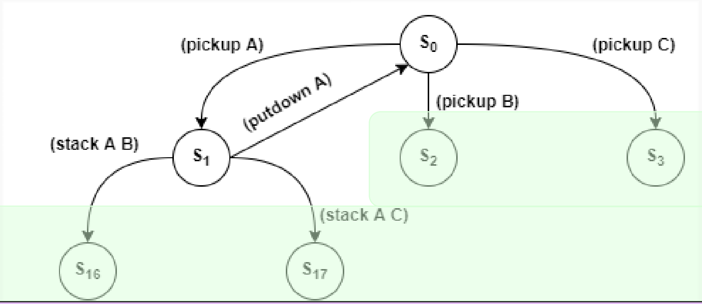


We now have
3 open nodes,
which are *unexpanded*

An heuristic function estimates the distance from each open node to the goal:
We calculate $h(S_1)$, $h(S_2)$, $h(S_3)$
An heuristic strategy uses this value (and other info) to prioritize the search

Example (cont.)

Suppose the strategy chooses to visit S_1!



We now have
4 open nodes,
which are *unexpanded*

2 new heuristic values are calculated $h(S_{16})$, $h(S_{17})$
The search strategy now has 4 nodes to prioritize

What to measure?

Question 1A: What should an heuristic function measure?

- An heuristic strategy bases its decisions on:
  - Heuristic value $h(s)$
  - Other factors: e.g. $g(s)$ i.e. cost of reaching s

- A very general definition!
  - $\Longrightarrow$ could measure anything that some strategy might find useful!

Question 1B: What is "cost"?

- Often: $h(s)$ *tries* to approximate the cost of achieving the goal from s!
  - Useful for finding cheap plans, and often as a side effect, for finding plans cheaply!
  - But... What is "cost"?

Plan Quality and Action Costs

- Maybe: long plan = expensive plan
  - $c(\pi) = |\pi|$, i.e. number of actions in plan π
    - Reasonable in some domains: e.g. Tower of Hanoi
    - But: How to make sure your car is clean?

go to car wash | get supplies | wash car

go to car dealer | buy new car

shortest plan is best?

Heuristic $h(s)$ estimates:
"How many actions are needed
to reach the goal from s"

- Would prefer to support different action costs
  - Supported by most current planners
    - Each action $a \in A$ is associated with a cost $c(a)$
  - Total cost: $c(\pi) = \sum_{a \in \pi} c(a)$

Heuristic $h(s)$ estimates:
"How *expensive* actions are needed
to reach the goal from s"

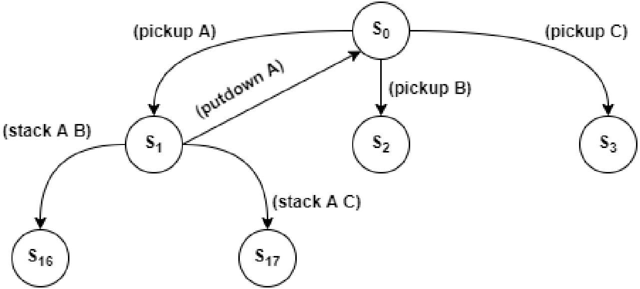
Action Costs in PDDL

- PDDL: Specify requirements:
  - `(:requirements :action-costs)`
- Numeric state variables for the total cost, called `(total-cost)`
  - And possibly numeric variables to *calculate* action costs
  -
```
(:functions (total-cost)
            (travel-slow-cost ?f1 - count ?f2 - count)
            (travel-fast-cost ?f1 - count ?f2 - count))
```
- Initial state
  -
```
(:init (= (total-cost) 0)
       (= (travel-slow-cost n0 n1) 6) (= (travel-slow-cost n0 n2) 7) ...
       (= (travel-fast-cost n0 n1) 8) (= (travel-fast-cost n0 n2) 9) ...
       ...)
```
- Special increase effects to increase total cost
  -
```
(:action move-up-slow
  :parameters (?l - slow-elevator ?f1 - count ?f2 - count)
  :precondition (and (lift-at ?l ?f1) (above ?f1 ?f2) (reachable-floor ?l ?f2))
  :effect (and (lift-at ?l ?f2) (not (lift-at ?l ?f1))
               (increase (total-cost) (travel-slow-cost ?f1 ?f2))) )
```

Remaining Costs

- The remaining cost in any search state s
  - The cost of a cheapest (optimal) solution starting in s
  - Denoted by $h^*(s)$
  - Star $* \Longrightarrow$ the best, optimal, estimate: *exact* cost

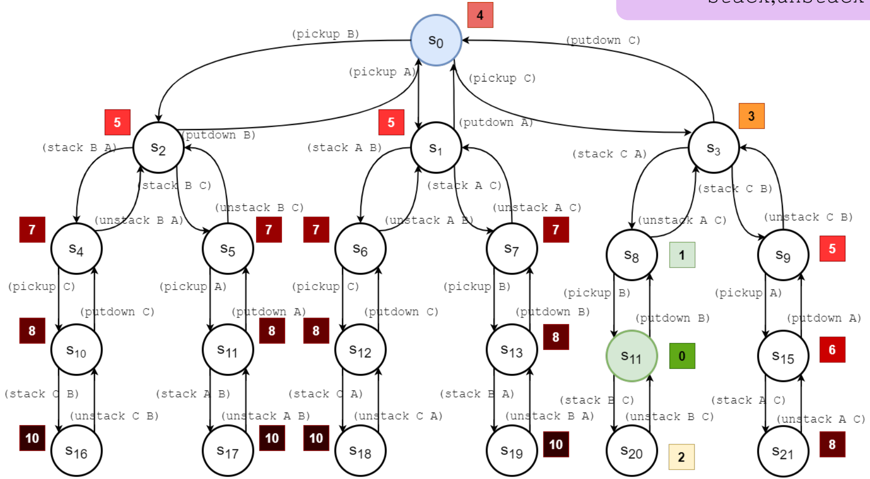
- The cost of an optimal solution to (Σ, S_0, S_g)
  - $h^*(S_0)$



True Remaining Costs

- True cost of reaching a goal node from n: $h^*(n)$

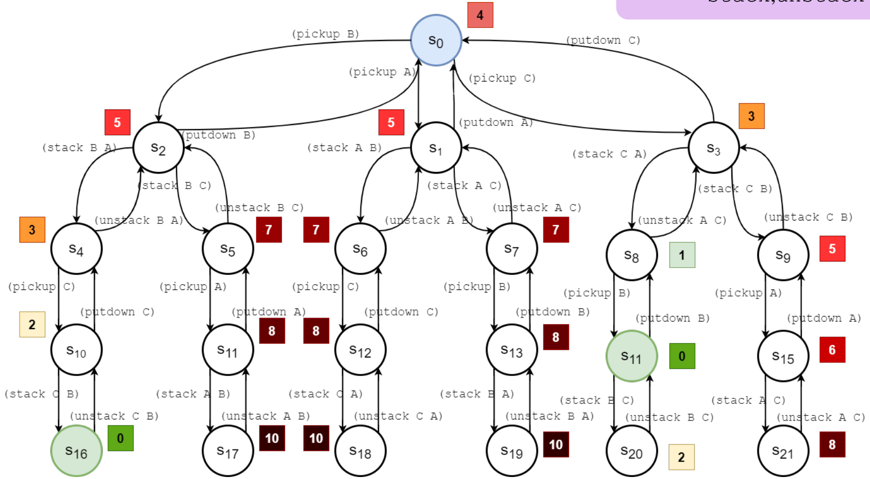
Initially A,B,C on the table
`pickup`,`putdown` cost 1
`stack`,`unstack` cost 2



True Remaining Costs (cont.)

- True cost of reaching a goal node from n: $h^*(n)$

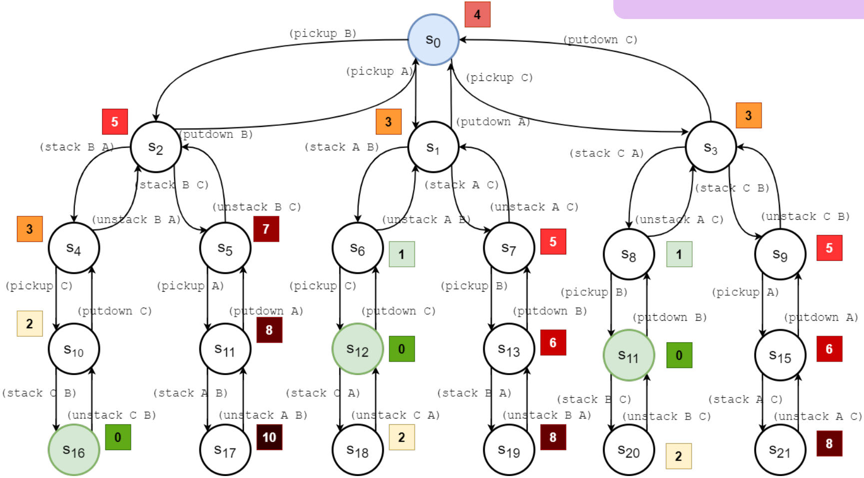
Two reachable goal nodes
pickup,putdown cost 1
stack,unstack cost 2



True Remaining Costs (cont.)

- True cost of reaching a goal node from n: $h^*(n)$

Three reachable goal nodes
(there can be many)

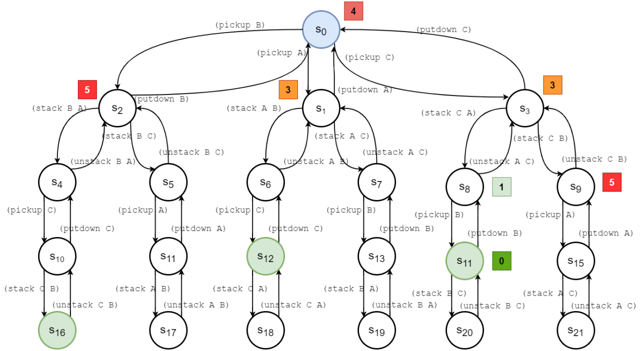


True Remaining Costs (cont.)

- If we *knew* the true remaining cost $h^*(n)$ for every node:

function ALGORITHM SIMPLEPLAN(problem)
  initial-node ← MAKE-INITIAL-NODE(problem)
  while (not reached goal) **do**
    node ← A-SUCCESSOR-NODE-MINIMAL-H*(node)
  end while
end function

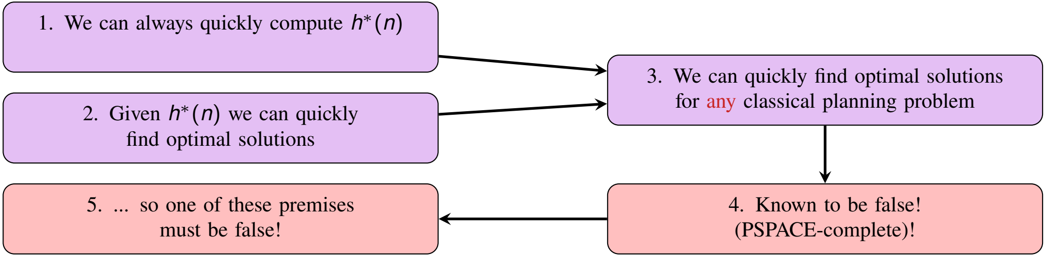
Trivial straight-line path minimizing h^* values gives an *optimal* solution!



REFLECTIONS

- What does this mean?
  - Calculating $h^*(n)$ is a good idea, because then we can easily find optimal plans?

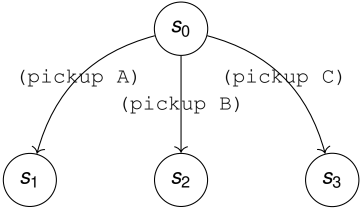
- No!!! - because we can prove that finding optimal plans is hard!
  - So the hard part must be calculating $h^*(n)$ ...



- Must settle for an estimate that helps us search less than otherwise!

MINIMIZATION: INTRODUCTION

- Example strategy: *depth-first search*: select a child with minimal $h(s)$

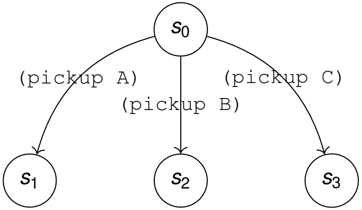


$h^*(s_1) = 55$ $h^*(s_2) = 57$ $h^*(s_3) = 62$

- If I start with `(pickup A)`, then make optimal choices:
  - Plan cost = 55
- If I start with `(pickup C)`, then make optimal choices:
  - Plan cost = 62

MINIMIZATION: CASE 1

- Example strategy: *depth-first search*: select a child with minimal $h(s)$

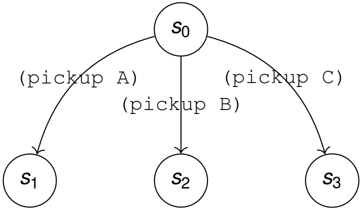


$h^*(s_1) = 55$	$h^*(s_2) = 57$	$h^*(s_3) = 62$	
$h^A(s_1) = 50$	$h^A(s_2) = 53$	$h^A(s_3) = 55$	Close!
$h^B(s_1) = 4$	$h^B(s_2) = 20$	$h^B(s_3) = 21$	Far from truth...

- Which is best?
  - The strategy only cares about relative values!
    - h^*, h^A, and h^B all results in identical choices: s_1 first!

Minimization: Case 2

- Example strategy: *depth-first search*: select a child with minimal $h(s)$

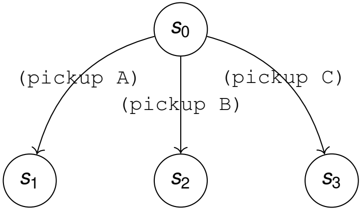


$h^*(s_1) = 55$	$h^*(s_2) = 57$	$h^*(s_3) = 62$	
$h^A(s_1) = 50$	$h^A(s_2) = 53$	$h^A(s_3) = 55$	Close!
$h^B(s_1) = 107$	$h^B(s_2) = 258$	$h^B(s_3) = 522$	Large overestimate!

- Which is best?
  - The strategy only cares about relative values!
    - h^*, h^A, and h^B all results in identical choices: s_1 first!

MINIMIZATION: CASE 3

- Example strategy: *depth-first search*: select a child with minimal $h(s)$

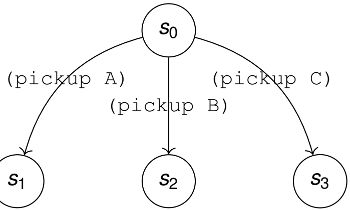


$h^*(s_1) = 55$	$h^*(s_2) = 57$	$h^*(s_3) = 62$	
$h^A(s_1) = 54$	$h^A(s_2) = 53$	$h^A(s_3) = 47$	Close!
$h^B(s_1) = 4$	$h^B(s_2) = 20$	$h^B(s_3) = 21$	Far from truth...

- Which is best?
  - The strategy only cares about relative values!
    - h^A is worse for this strategy, despite being closer to h^*: goes to s_3 first!
    - Even if we continue optimally, cost ≥ 62!

A*: Case 1

- Example strategy: $A*$

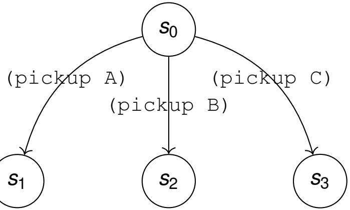


$h^*(s_1) = 55$	$h^*(s_2) = 57$	$h^*(s_3) = 62$
$h^A(s_1) = 50$	$h^A(s_2) = 53$	$h^A(s_3) = 55$
$h^B(s_1) = 4$	$h^B(s_2) = 20$	$h^B(s_3) = 21$

- Which is best?
  - A* expands all nodes where $h(s) + g(s) \leq h^*(s)$
    - As long as h is admissible $[\forall s.h(s) \leq h^*(s)]$, increasing it is always better

A*: CASE 2

- Example strategy: $A*$

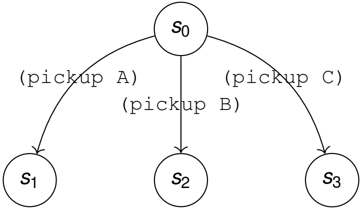


$h^*(s_1) = 55$	$h^*(s_2) = 57$	$h^*(s_3) = 62$
$h^A(s_1) = 50$	$h^A(s_2) = 53$	$h^A(s_3) = 55$
$h^B(s_1) = 107$	$h^B(s_2) = 258$	$h^B(s_3) = 522$

- Which is best?
  - A* expands all nodes where $h(s) + g(s) \leq h^*(s)$
    - Because h^B is not admissible, optimal solutions may be missed!

A*: Case 3

- Example strategy: $A*$



$h^*(s_1) = 55$	$h^*(s_2) = 57$	$h^*(s_3) = 62$
$h^A(s_1) = 54$	$h^A(s_2) = 53$	$h^A(s_3) = 47$
$h^B(s_1) = 4$	$h^B(s_2) = 20$	$h^B(s_3) = 21$

- Which is best?
  - A* expands all nodes where $h(s) + g(s) \leq h^*(s)$
    - As long as $h(s)$ is admissible $[\forall s.h(s) \leq h^*(s)]$, increasing is always better: $\Longrightarrow h^A$ better then h^B

Two requirements for Heuristic Guidance

Define search strategy able to take guidance into account

- Example:
  - A* uses a heuristic function
  - Hill-climbing uses a heuristic ... differently!
  - ...

Find a heuristic function suitable for the selected strategy

- Example:
  - Find an heuristic function suitable specifically for A*
  - Find an heuristic function suitable specifically for hill-climbing
  - ...

- Can be domain specific, given as input to the planning problem!
- Can be domain independent, generated automatically by the planner given the problem domain!

We will consider both – heuristics more than strategies!

SOME DESIRED PROPERTIES

- What properties do good heuristic functions have?
  - Shall be Informative: provide good guidance to the specific search strategy we use
    - Admissible?
    - Close to $h^*(n)$?
    - Correct "ordering"?
    - ...

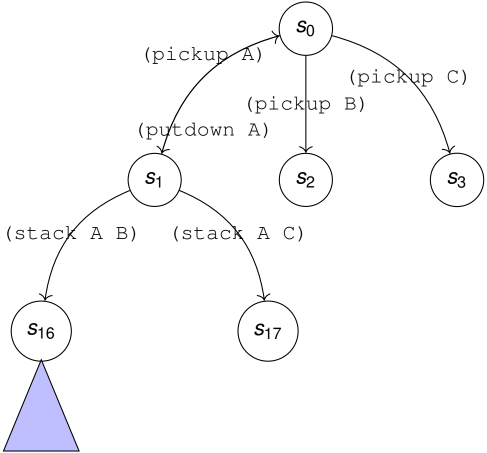
Some Desired Properties (cont.)

- What properties do good heuristic functions have?
  - Shall be efficiently computable
    - Spend as little time as possible deciding which nodes to expand
  - Shall be balanced...
    - Many planners spend almost all their time calculating heuristics
    - But: Don't spend more time computing h than the time you gain by expanding few nodes!
    - Illustrative (made-up) example:

Heuristic quality	Nodes expanded	Exp. 1 node	Calc. h one node	Total time
Worst	100000	100μs	1 μs	10100ms
Better	20000	100μs	10 μs	2200ms
...	5000	100μs	100 μs	1000ms
...	2000	100μs	1000 μs	2200ms
...	500	100μs	10000 μs	5050ms
Best	200	100μs	100000μs	20020ms

Cheap Plans, Cheap Planning

- Cost can be indirectly related to plan generation time!



- If we can find a cheap plan "under" s_{16}
  - $\Longrightarrow$ might find a plan in few steps
  - $\Longrightarrow$ might not need to search so many nodes
  - $\Longrightarrow$ might find a plan cheaply
- Maybe!
  - Or maybe s_{16} opens up a vast number of alternatives, so finding a solution may take more time...

Prioritizing Speed or Plan Cost

- Can design strategies to prioritize speed or plan cost

Find a solution quickly

Expand nodes where you think you can easily find a way to a goal node

Find a good solution

Expand nodes where you think you can find a way to a good (high-quality) solution, even if finding it will be difficult!

Should prefer

Should prefer

Open nodes

Accumulated plan cost $g(n) = 50$, estimated "cost distance" $h(n) = 10$

Accumulated plan cost $g(n) = 5$, estimated "cost distance" $h(n) = 30$

Often one strategy+heuristic can achieve *both* reasonably well, but for optimum performance, the distinction can be important!

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`.