Linköping University

Automated Planning

Heuristics: An Overview

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

Heuristic Search

jonkv@ida

- General **Search-Based Planning Algorithm** (repetition)

```
search(problem) {
    initial-node ← make-initial-node(problem) //
    open ← { initial-node }
    while (open ≠ ∅) {
        node ← search-strategy-remove-from(o
        if is-solution(node) then  // [4]
            return extract-plan-from(node) // [5

        foreach newnode ∈ successors(node) { // [
            add newnode to open
        }
    }
    // Expanded the entire search space without finding a solution
    return failure;
}
```

A **heuristic strategy** bases decisions on:

- **Heuristic value h(*n*)**
- Often other factors, such as **g(n)** **= cost of reaching *n***

Best first search: Greedy, A*, …

Modifications: IDA*, D*, …

Simulated annealing, hill-climbing, …

Requires a **heuristic function**

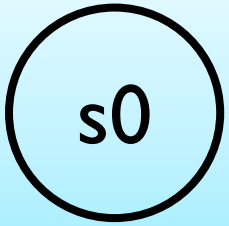
How do we calculate $h(n)$?
Landmarks,
pattern databases,
…

Heuristics in Forward State Space Search: Introduction

Example (1)

jonkv@ida

Example: 3 blocks, all on the table in s0



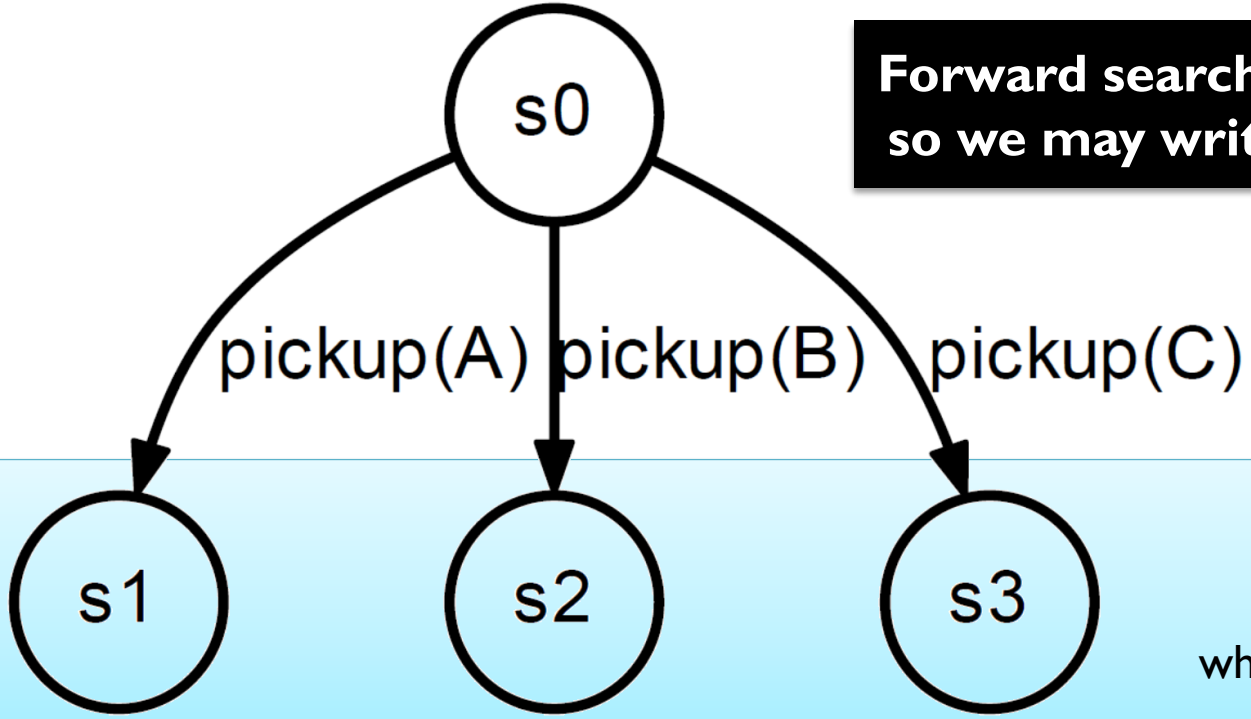
We now have
1 *open node*,
which is *unexpanded*

Example (2)

jonkv@ida

We visit s_0 and expand it

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



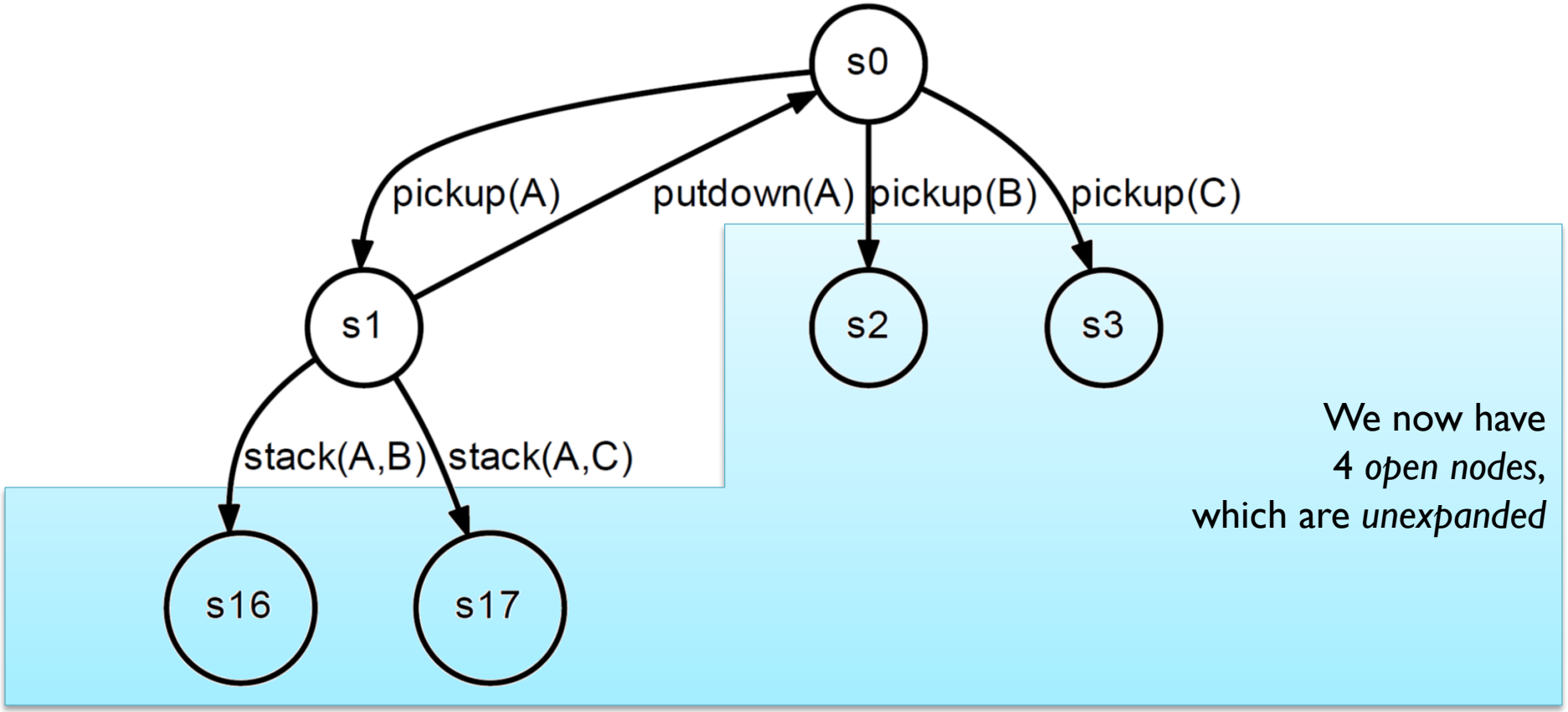
We now have
3 *open nodes*,
which are *unexpanded*

A ***heuristic function*** estimates the distance from each open node to the goal:
We calculate $h(s_1), h(s_2), h(s_3)$
A ***heuristic strategy*** uses this value (***and*** other info) to *prioritize*

Example (3)

jonkv@ida

Suppose the strategy chooses to visit s_1:



2 new heuristic values are calculated: $\boldsymbol{h(s_{16}), h(s_{17})}$

The ***search strategy*** now has 4 nodes to prioritize

Heuristic Functions: What to Measure?

What to Measure?

jonkv@ida

Question 1A: **What** should a heuristic function **measure**?

- A **heuristic strategy** bases its decisions on:
  - **Heuristic value h(*s*)**
  - Often other factors, such as **g(s) = cost of reaching *s***

Very general definition
➔ **could** measure **anything** that **some** strategy might find useful!

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**

➔ **Question 1B: What is ”cost”?**

Plan Quality and Action Costs

jonkv@ida

- Maybe: **Long** plan = **expensive** plan
  - $c(\pi) = |\pi|$ (number of actions in plan π)
    - Reasonable in Towers of Hanoi
    - But: How to make sure your car is clean?

Heuristic $h(s)$ estimates:

"How many actions will I need to reach the goal from s?"

go to car wash	get supplies	wash car
go to car dealer	buy new car	shortest plan is best?

- Would prefer to support different **action costs**
  - Supported by most current planners
    - Each action $a \in A$ associated with a cost $c(a)$
  - **Total cost**:

$$c(\pi) = \sum_{a \in \pi} c(a)$$

Heuristic $h(s)$ estimates:

"How expensive actions will I need to reach the goal from s?"

Action Costs in PDDL

jonkv@ida

- PDDL: Specify requirements
  - (:**requirements** :**action-costs**)
- **Numeric state variable** for the total cost, called **(total-cost)**
  - And possibly numeric state variables to *calculate* action costs
  - (:**functions** (total-cost)
    (travel-slow-cost ?f1 - count ?f2 - count)
    (travel-fast-cost ?f1 - count ?f2 - count)
    - number
    - number
    - number)

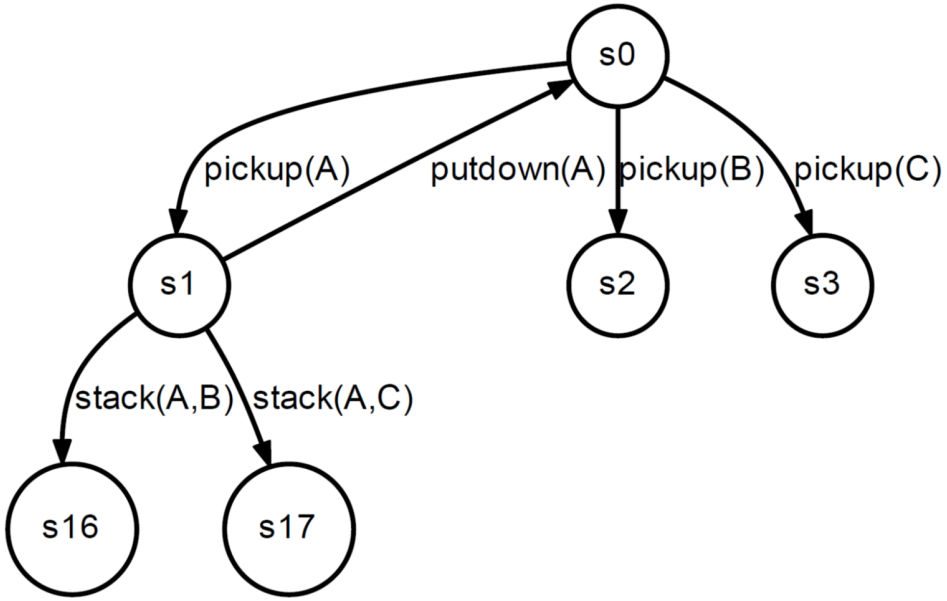
    Built-in type supported by cost-based planners
- **Initial state**
  - (:**init** (= (total-cost) 0)
    (= (travel-slow-cost n0 n1) 6) (= (travel-slow-cost n0 n2) 7)
    (= (travel-slow-cost n0 n3) 8) (= (travel-slow-cost n0 n4) 9)
    ...)
- Special **increase effects** to increase total cost
  - (:**action** move-up-slow
    :**parameters** (?lift - slow-elevator ?f1 - count ?f2 - count)
    :**precondition** (and (lift-at ?lift ?f1) (above ?f1 ?f2) (reachable-floor ?lift ?f2))
    :**effect** (and (lift-at ?lift ?f2) (not (lift-at ?lift ?f1))
    (increase (total-cost) (travel-slow-cost ?f1 ?f2))))

Remaining Costs

jonkv@ida

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

- The cost of an **optimal solution** to (Σ, s_0, S_g):
  - $h^*(s_0)$

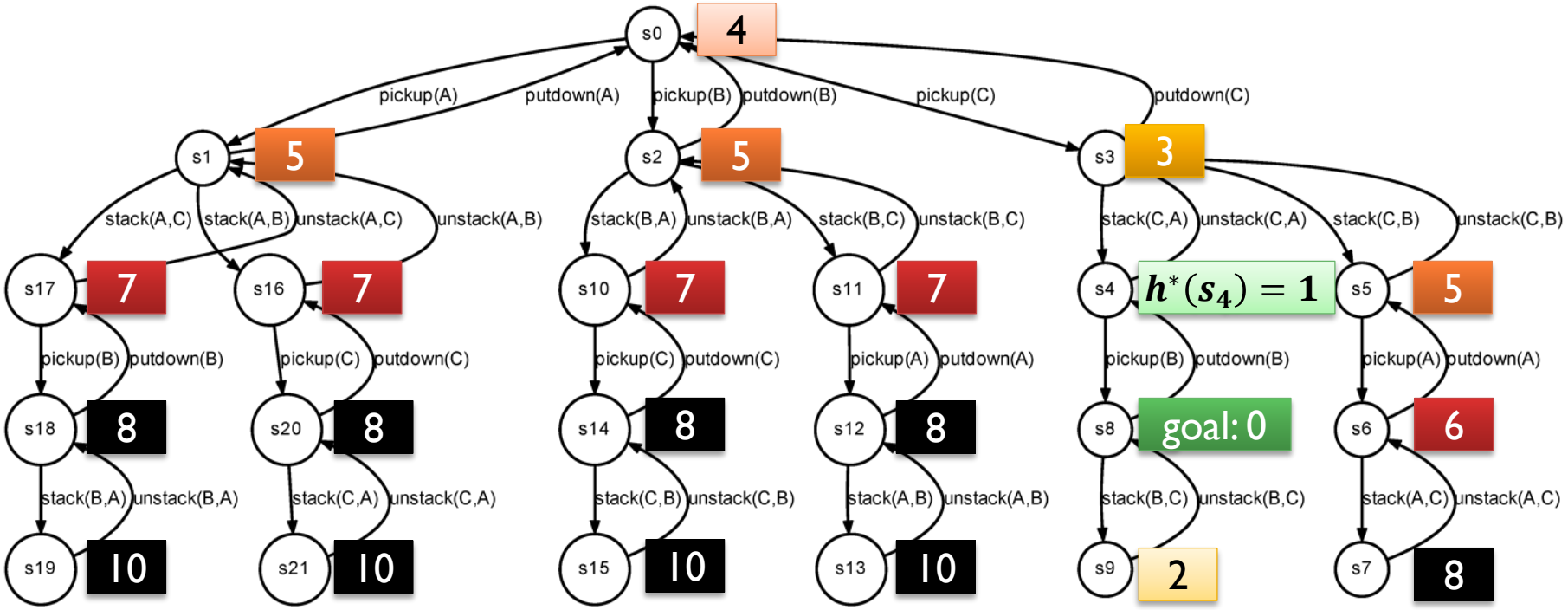


True Remaining Costs (1)

jonkv@ida

True Cost of Reaching a Goal from n: $h^*(n)$

Initially: A,B,C on the table
pickup, putdown cost 1
stack, unstack cost 2 (must be more careful)

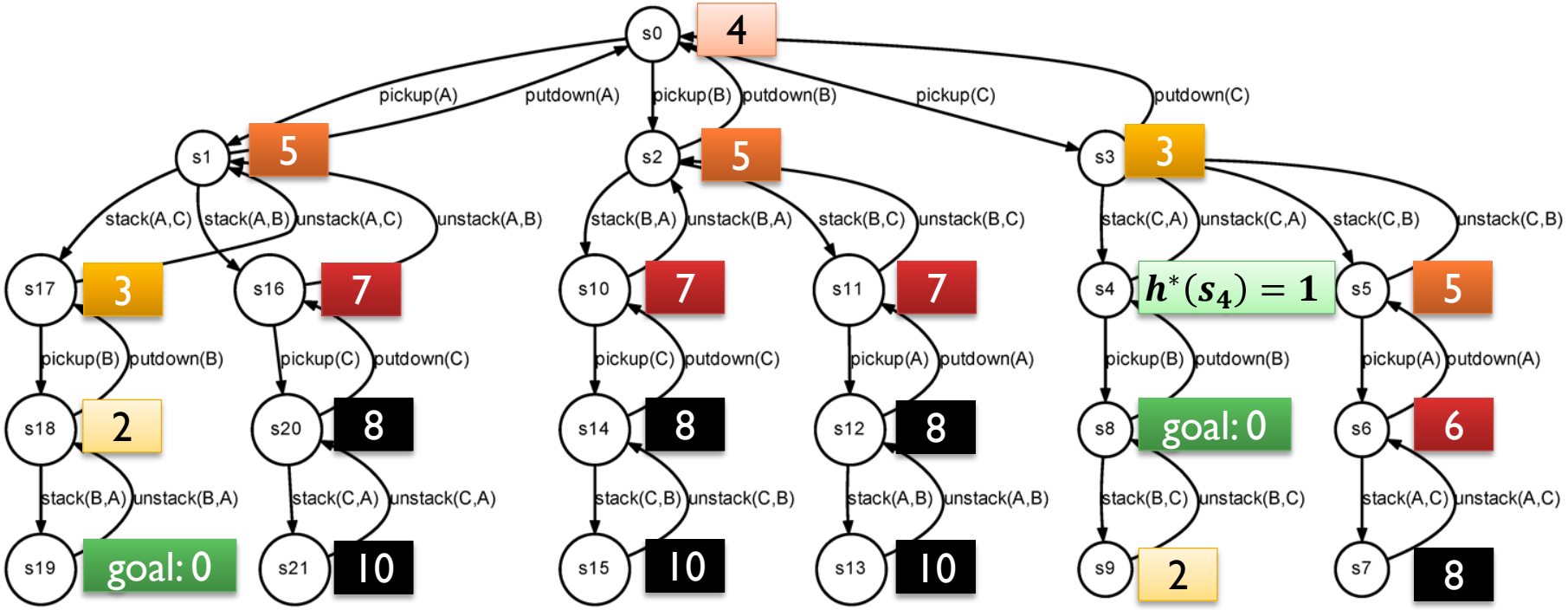


True Remaining Costs (2)

jonkv@ida

True Cost of Reaching a Goal: $h^*(n)$

Two reachable goal nodes

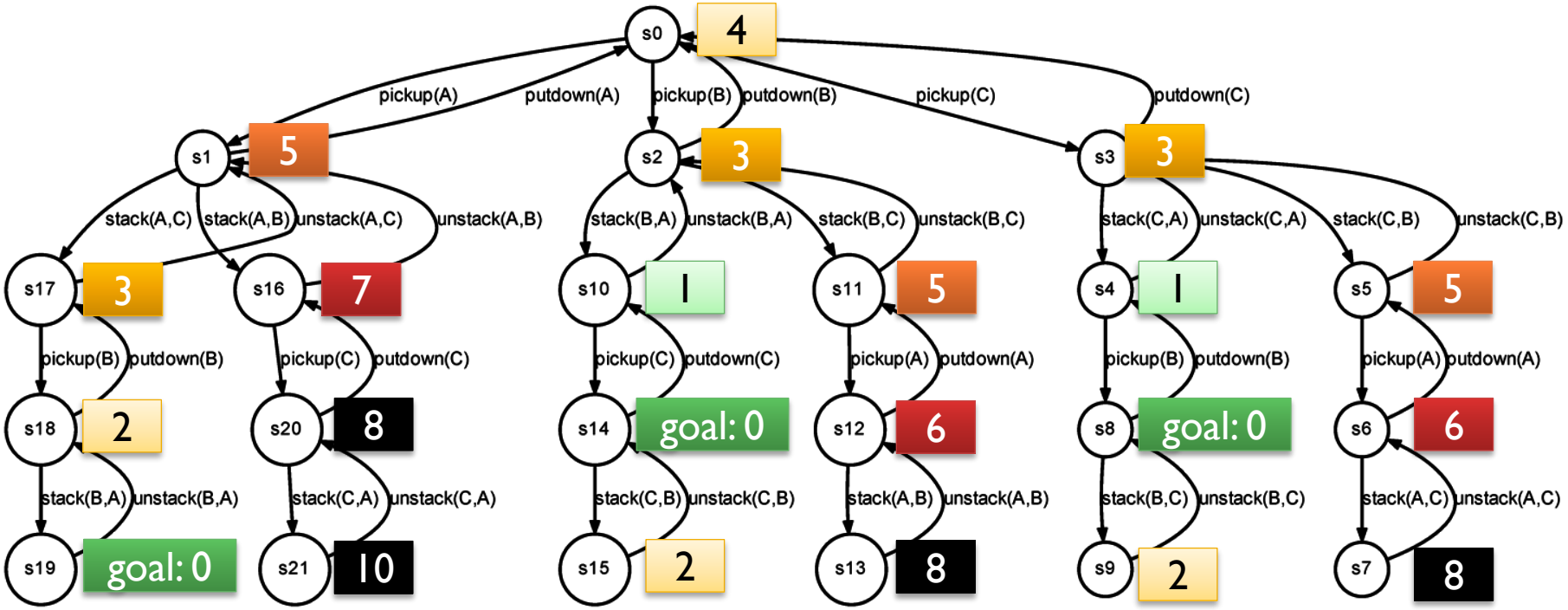


True Remaining Costs (3)

jonkv@ida

True Cost of Reaching a Goal: $h^*(n)$

Three reachable goal nodes
(there can be many)



True Remaining Costs (4)

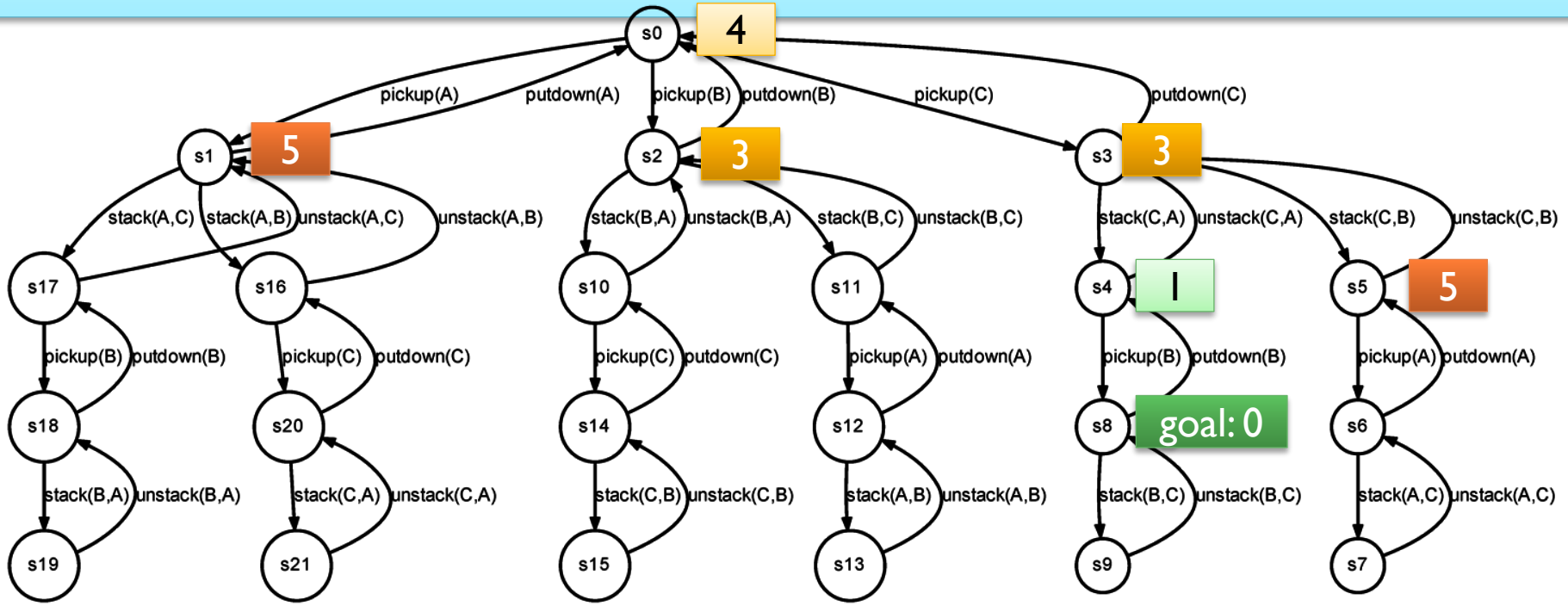
jonkv@ida

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

Algorithm simplePlan:

node ← *initial-node*
while (not reached goal) {
  node ← a successor of *node* with minimal h*(n)
}

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

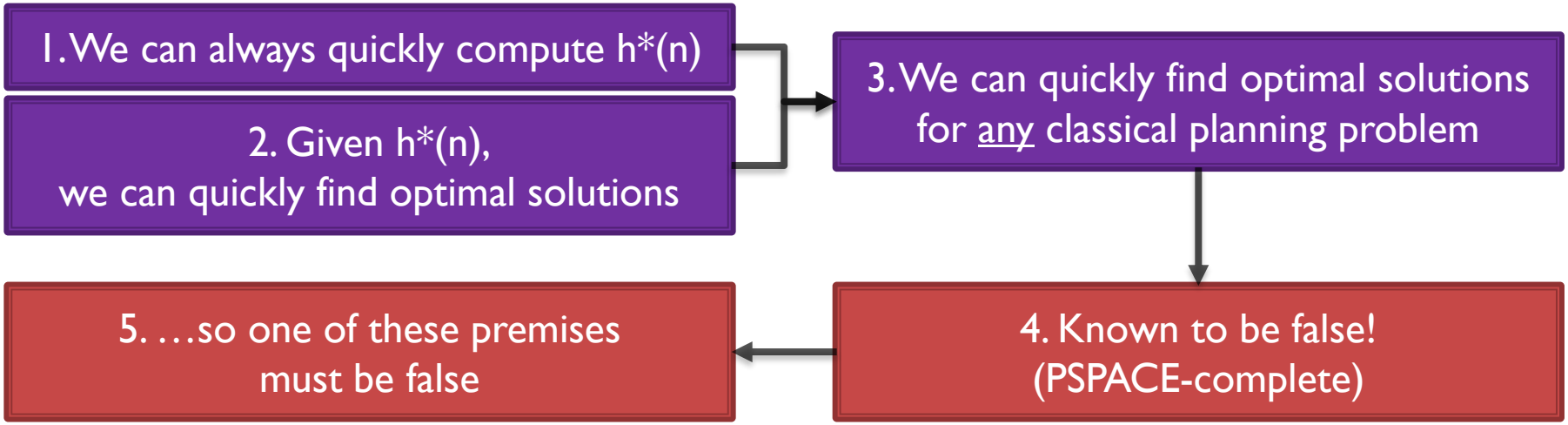


Reflections

jonkv@ida

- 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 $h^*(n)$…



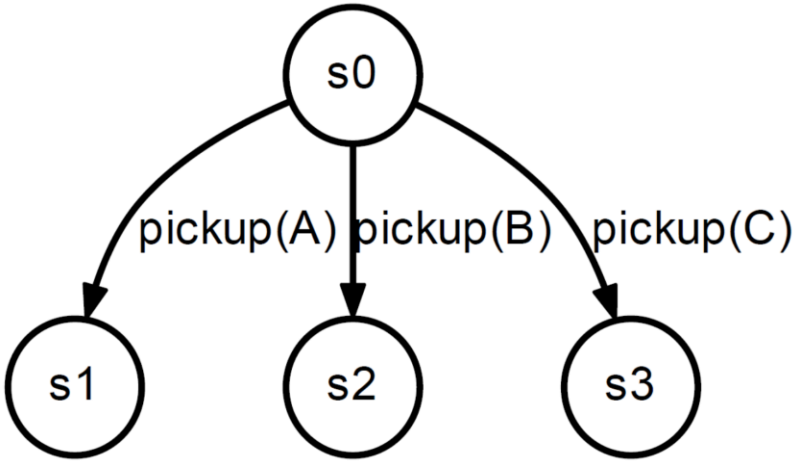
Must settle for an **estimate** that helps us **search less** than otherwise

Heuristic Functions:
What properties should an estimate have?

Minimization: Intro

jonkv@ida

Example Strategy: **Depth first search**; select a child with **minimal** $h(s)$



h*(s1)=55 h*(s2)=57 h*(s3)=62

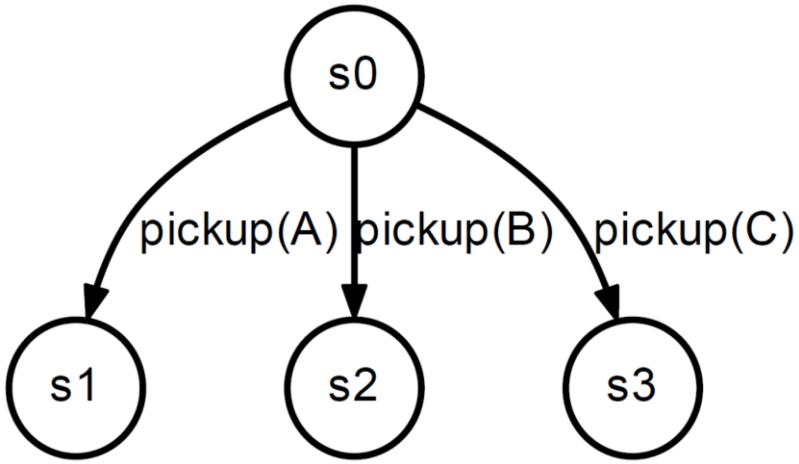
If I start with pickup(A),
then make **optimal** choices:
Plan cost = 55

If I start with pickup(C),
then make **optimal** choices:
Plan cost = 62

Minimization, case 1

jonkv@ida

Strategy: Depth first search; select a child with **minimal** $h(s)$



h*=55	h*=57	h*=62
hA=50	hA=53	hA=55
hB=4	hB=20	hB=21

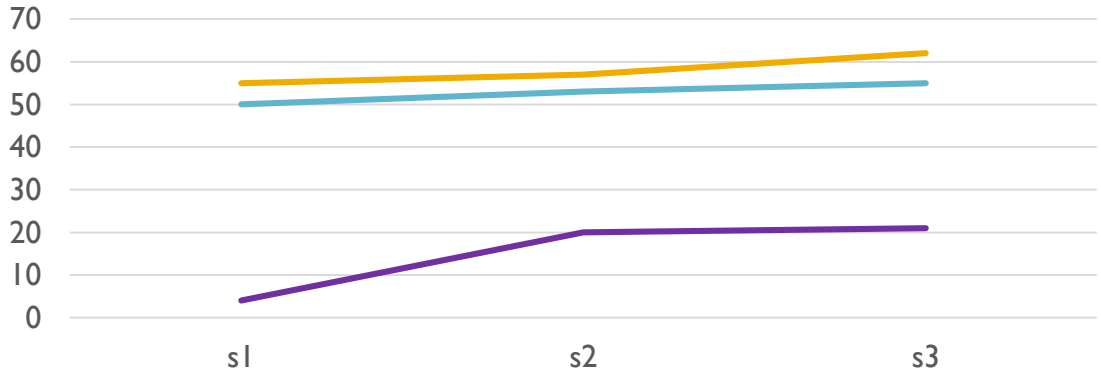
Which is best?

The strategy only cares about relative values

h*, hA, hB result in identical choices: s_1 first!

Close!

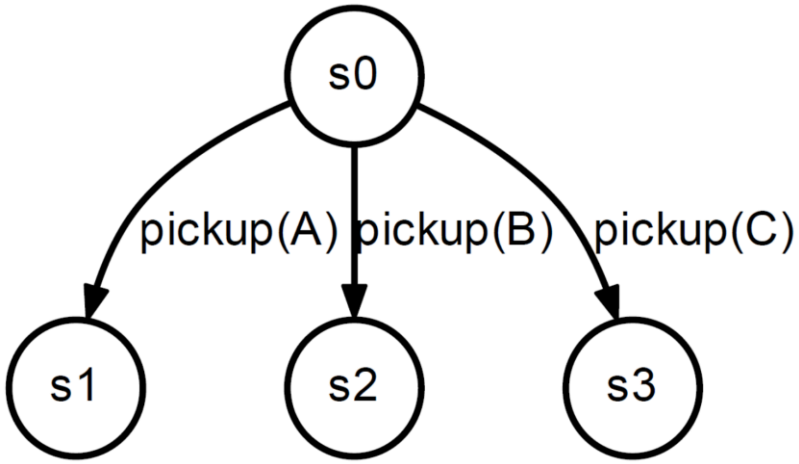
Far from the truth…



Minimization, case 2

jonkv@ida

Strategy: Depth first search; select a child with **minimal** $h(s)$



h*=55	h*=57	h*=62
hA=50	hA=53	hA=55
hB=107	hB=258	hB=522

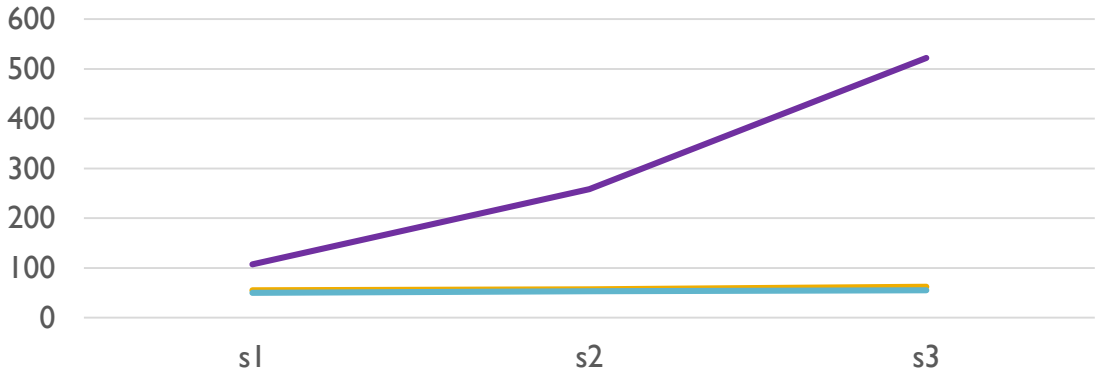
Close!

Large overestimate!

Which is best?

The strategy only cares about relative values

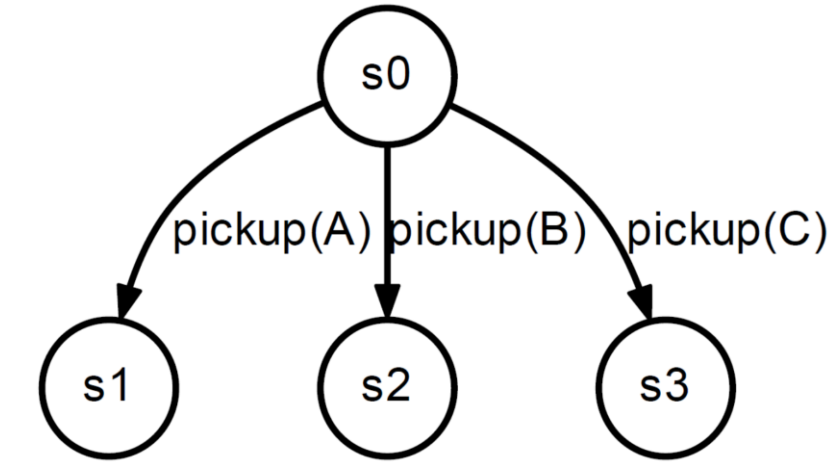
h*, hA, hB result in identical choices: s_1 first!



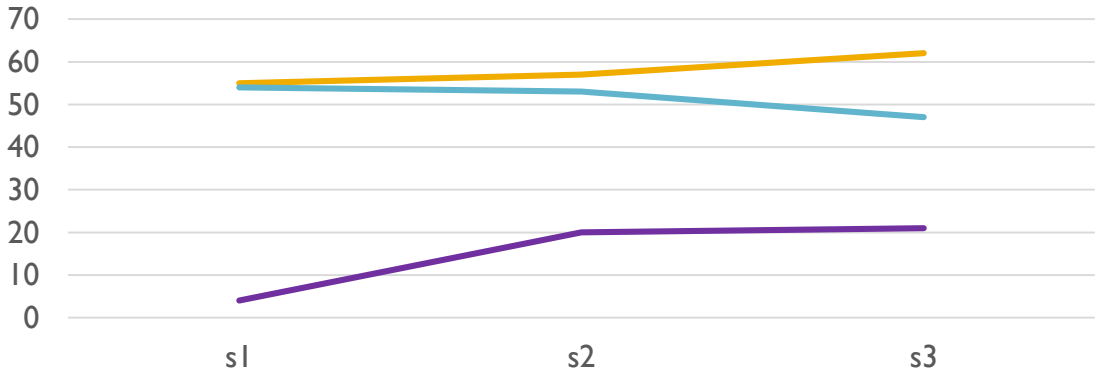
Minimization, case 3

jonkv@ida

Strategy: Depth first search; select a child with **minimal** $h(s)$



h*=55	h*=57	h*=62
hA=54	hA=53	hA=47
hB=4	hB=20	hB=21



Which is best?

h* and hB result in identical choices

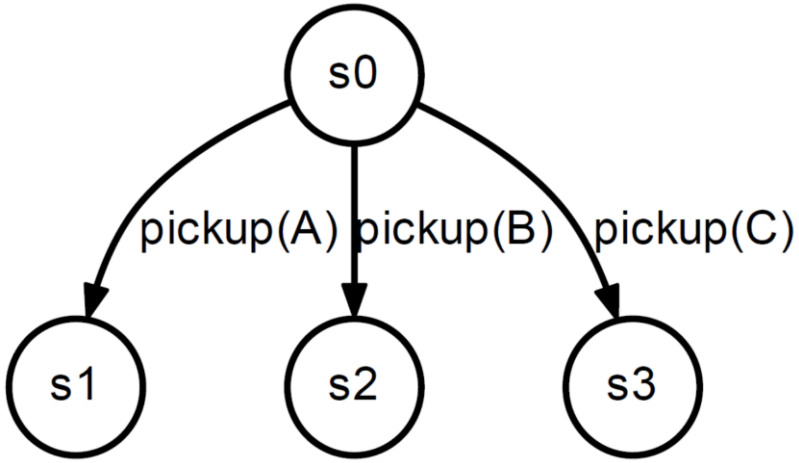
hA is worse for this strategy, despite being closer to h*: Goes to s_3 first

Even if we continue optimally, cost $\geq$ 62!

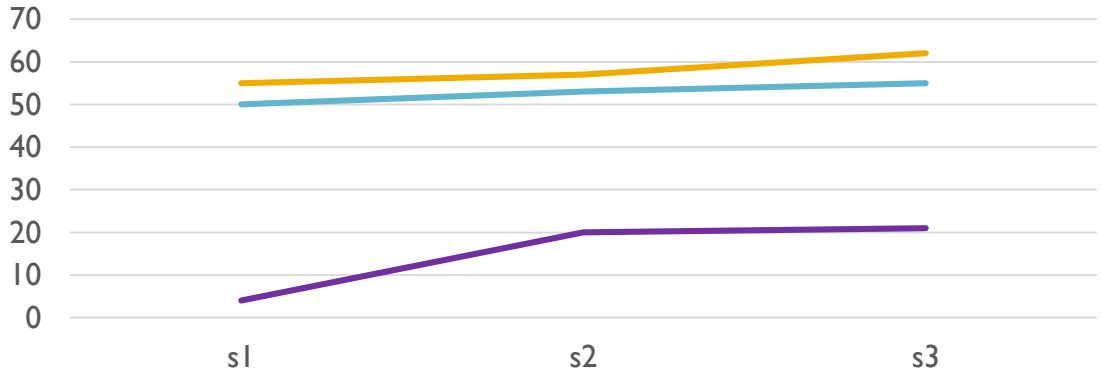
A*, case 1

jonkv@ida

Back to case 1 – but suppose the **strategy** is A*



h*=55	h*=57	h*=62
hA=50	hA=53	hA=55
hB=4	hB=20	hB=21



Which is best?

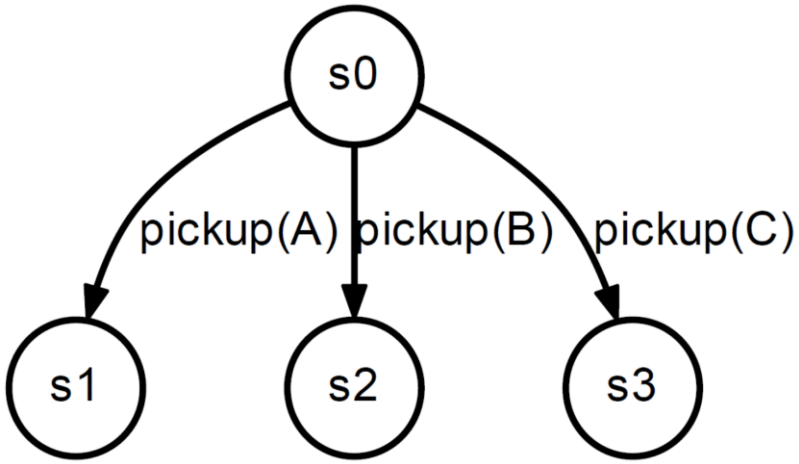
A* expands all nodes where $g(s) + h(s) \leq$ optcost

As long as h is admissible [$\forall s: h(s) \leq h^*(s)$], increasing it is always better

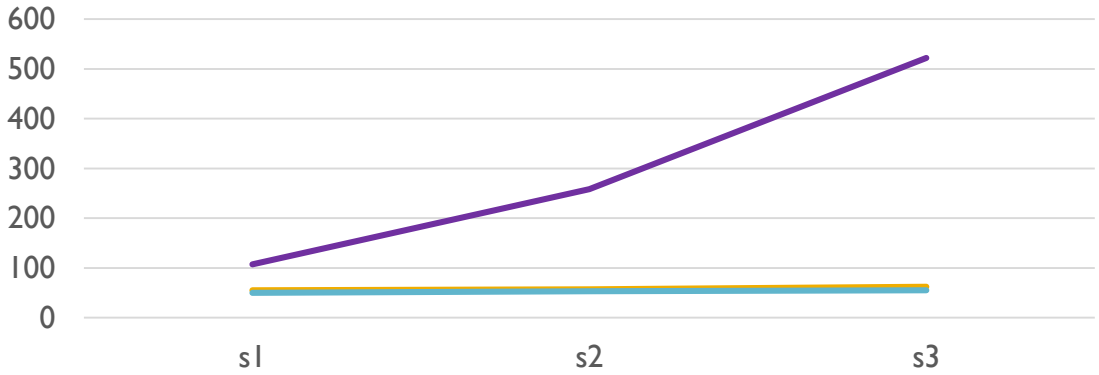
A*, case 2

jonkv@ida

Case 2: Suppose the **strategy** is A*



h*=55	h*=57	h*=62
hA=50	hA=53	hA=55
hB=107	hB=258	hB=522



Which is best?

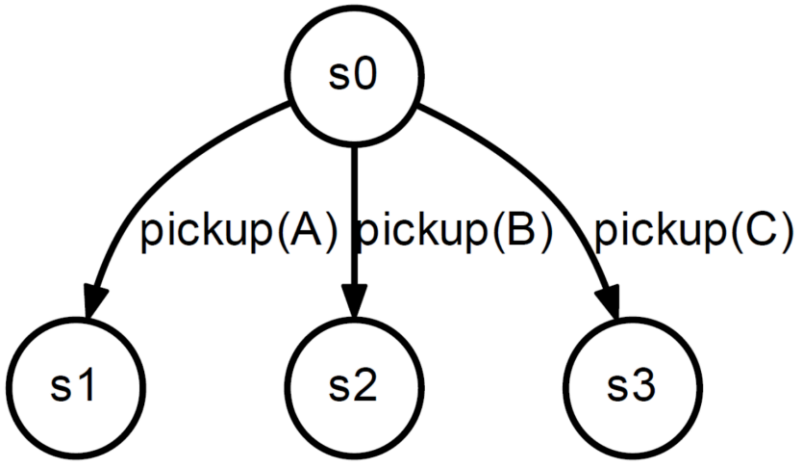
A* expands all nodes where $g(s) + h(s) \leq$ optcost

Because hB is not admissible, optimal solutions may be missed!

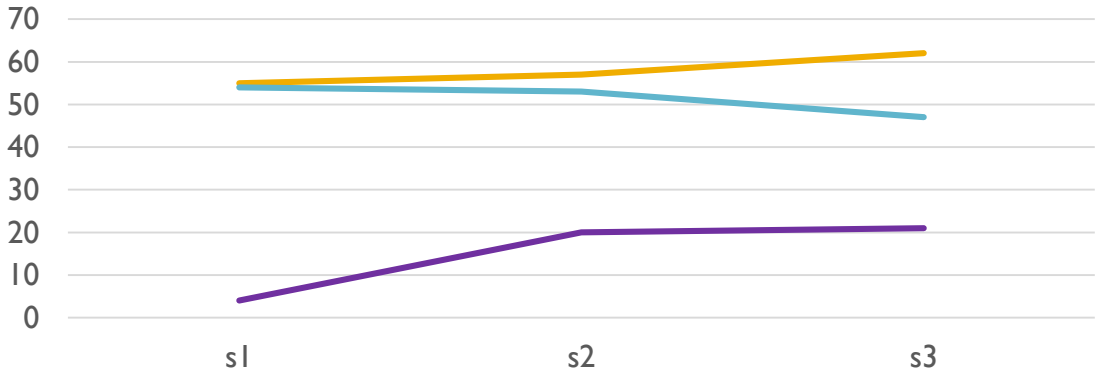
A*, case 3

jonkv@ida

Case 3: Suppose the **strategy** is A*



h*=55	h*=57	h*=62
hA=54	hA=53	hA=47
hB=4	hB=20	hB=21



Which is best?

A* expands all nodes where $g(s) + h(s) \leq$ optcost

As long as $h(s)$ is admissible [$h(s) \leq h^*(s)$], increasing it is always better
hA better than hB

Two Requirements for Heuristic Guidance

jonkv@ida

- Heuristic planners must consider **two** requirements

Define a **search strategy** able to take guidance into account	Find a **heuristic function** suitable for **the selected strategy**
Examples: A* uses a heuristic function Hill-climbing uses a heuristic… differently!	**Example**: Find a heuristic function suitable specifically for A* or hill-climbing
	Can be **domain-specific**, given as input in 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 (1)

jonkv@ida

- What properties do **good heuristic functions** have?
  - **Informative**, of course:
    Provide good guidance to the specific search strategy we use
    - Admissible?
    - Close to $h^*(n)$?
    - Correct ”ordering”?
    - …

Some Desired Properties (2)

jonkv@ida

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

Heuristic quality	Nodes expanded	Expanding one node	Calculating h for one node	Total time
Worst	100000	100 µs	1 µs	10100 ms
Better	20000	100 µs	10 µs	2200 ms

Some Desired Properties (3)

jonkv@ida

- [Table copy for the online lecture notes!]

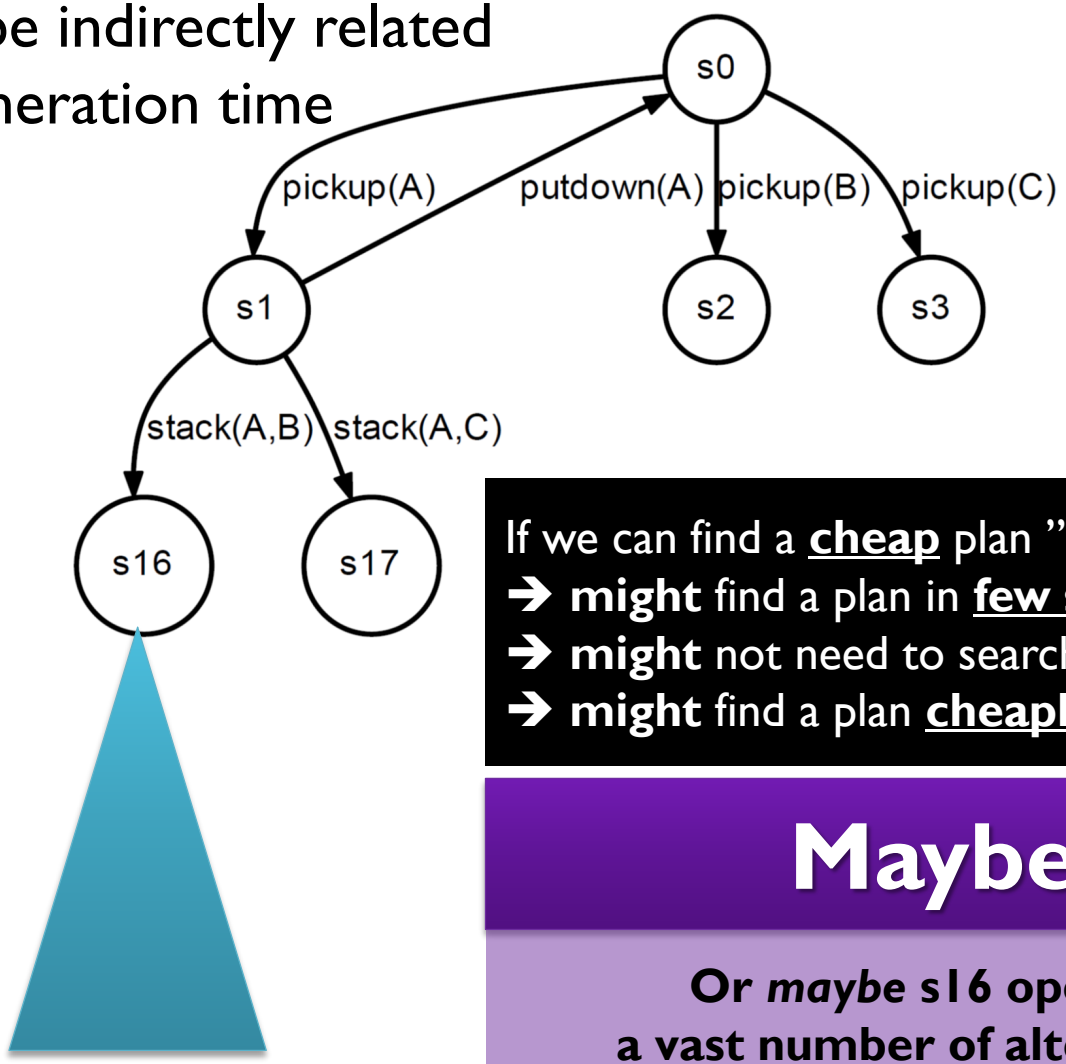
Heuristic quality	Nodes expanded	Expanding one node	Calculating h for one node	Total time
Worst	100000	100 µs	1 µs	10100 ms
Better	20000	100 µs	10 µs	2200 ms
…	5000	100 µs	100 µs	1000 ms
…	2000	100 µs	1000 µs	2200 ms
…	500	100 µs	10000 µs	5050 ms
Best	200	100 µs	100000 µs	20020 ms

Speed vs. Cost

Cheap Plans, Cheap Planning?

jonkv@ida

- Cost can be indirectly related to plan generation time



If we can find a **cheap** plan "under" s_{16}
➔ **might** find a plan in **few steps**
➔ **might** not need to search so many nodes
➔ **might** find a plan **cheaply**

Maybe!

Or *maybe* s16 opens up a vast number of alternatives, so finding a solution takes more time…

Prioritizing Speed or Plan Cost

jonkv@ida

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

Should prefer

Open nodes

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

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

Accumulated plan 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!