Artificial Intelligence

CSC 665

tyler dae devlin

Search III

2.13.2024

Recap

- **Search:** make decisions by looking ahead
- **Logic:** deduce new facts from existing facts
- **Constraints:** find a way to satisfy a given specification
- **Probability:** reason quantitatively about uncertainty
- **Learning:** make future predictions from past observations

Search

Modeling: start state, actions, cost, transition model, goal test

Inference: backtracking, DFS, BFS, UCS — *all uninformed search algorithms*

Backtracking search (last time)

Global state: minimum cost path, set of explored ~~nodes~~

function search(s, path) :

- **if** IsEnd(s) :
  - update the minimum cost path
- **for each** action $a \in$ Actions(s) :
  - **if** Succ(s, a) hasn't been explored yet:
    - add ~~it~~ to the explored set
    - extend path with Succ(s, a) and Cost(s, a)
    - recurse: search(Succ(s, a), path)

Backtracking search (last time)

Global state: minimum cost path, set of explored **(node, cost)** pairs

function search(s, path) :

- **if** IsEnd(s) :
  - update the minimum cost path
- **for each** action $a \in$ Actions(s) :
  - **if** Succ(s, a) hasn't been explored at Cost(s, a) yet:
    - add (Succ(s, a), Cost(s, a)) to the explored set
    - extend path with Succ(s, a) and Cost(s, a)
    - recurse: search(Succ(s, a), path)

Uniform Cost Search (UCS, Dijkstra's Algorithm)

- Start with a frontier that contains s_0, and an empty set of explored nodes
- **While** the frontier is nonempty:
  - Pop the node s with smallest priority p from the frontier
  - **If** $\text{IsEnd}(s)$: **return** solution
  - Add s to the explored set
  - **For** each $a \in \text{Actions}(s)$,
    - Get $s' = \text{Succ}(s, a)$
    - **If** s' is already explored: **continue**
    - Add s' to frontier with priority $p + \text{Cost}(s, a)$

[UCS example on board]

Correctness of UCS

Theorem: Assume action costs are non-negative. If a node s is popped from the frontier with priority p, then p is the cost of the min-cost path from s_0 to s.

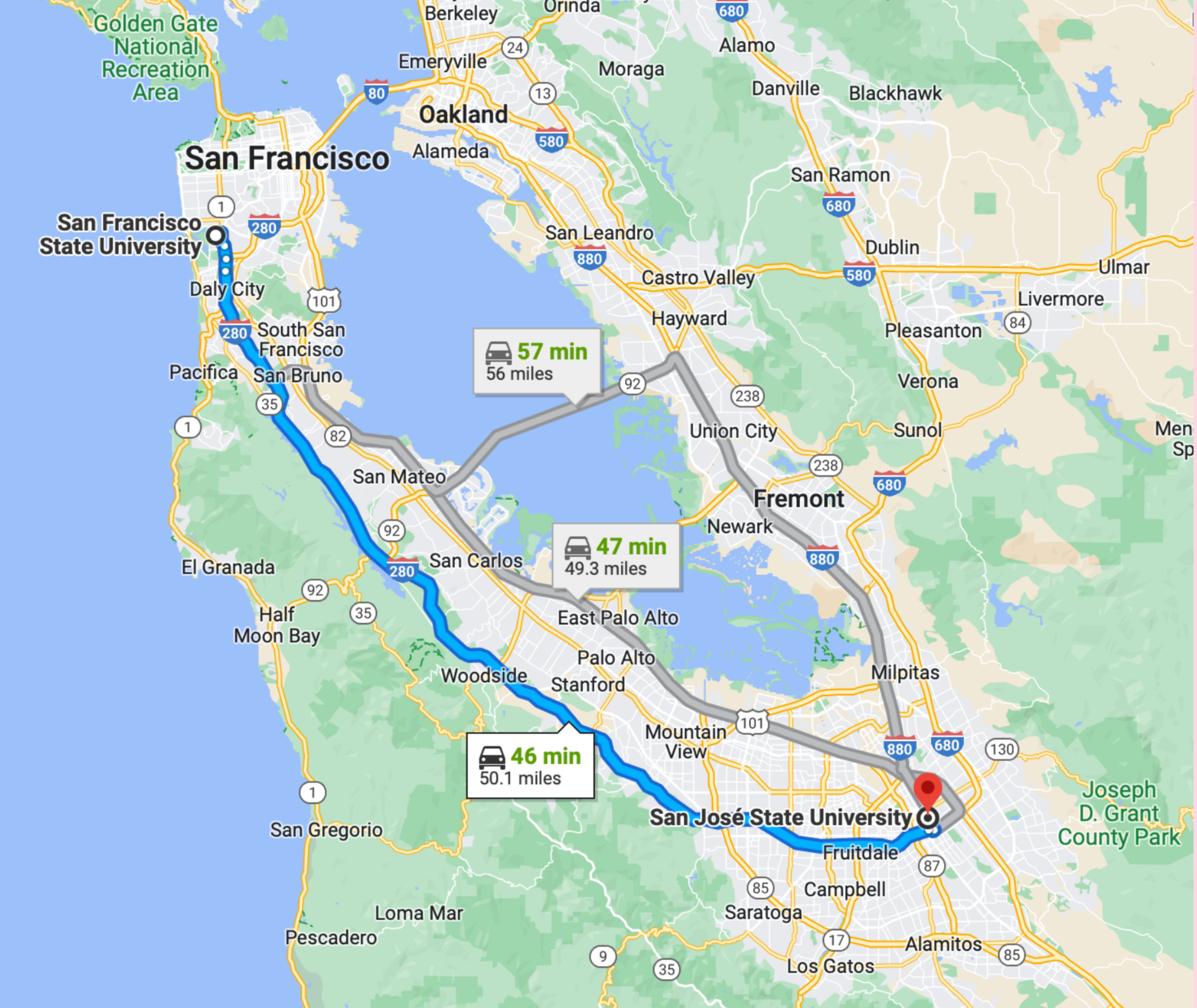
Proof: Take CSC 510 (or come to office hours).

Corollary: UCS computes the min-cost path to the goal node.

Using domain knowledge

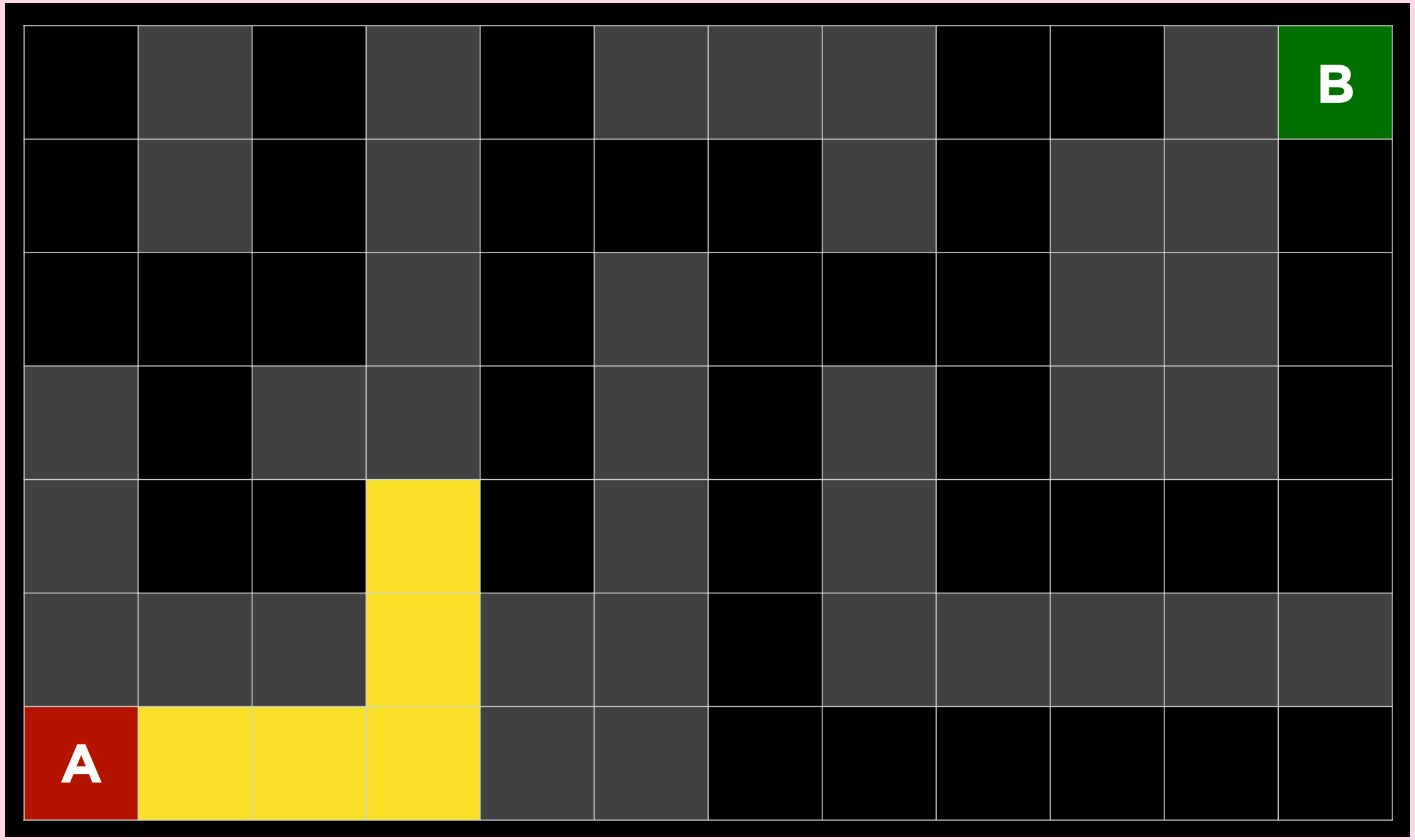
- So far: **uninformed search**
  - Algorithms that don't use problem-specific information
  - **Pro:** completely generic — same algorithm works for all search problems
  - **Con:** can't useful domain knowledge
- Next: **informed search**
  - Use a heuristic function $h : S \to \mathbb{R}$ to estimate progress toward goal

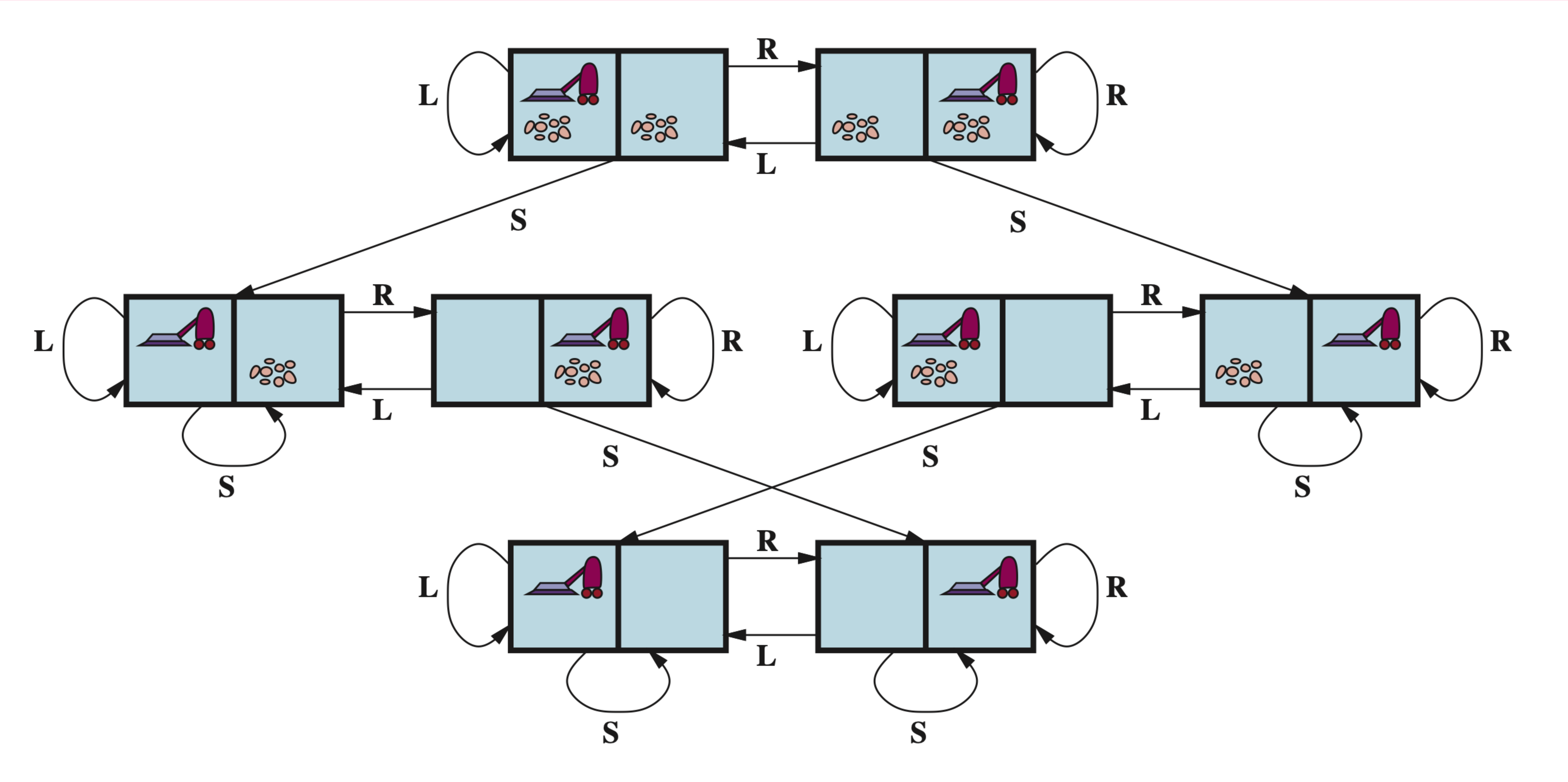
Informed search



Probably not good to start driving toward Marin

Probably not good to turn left



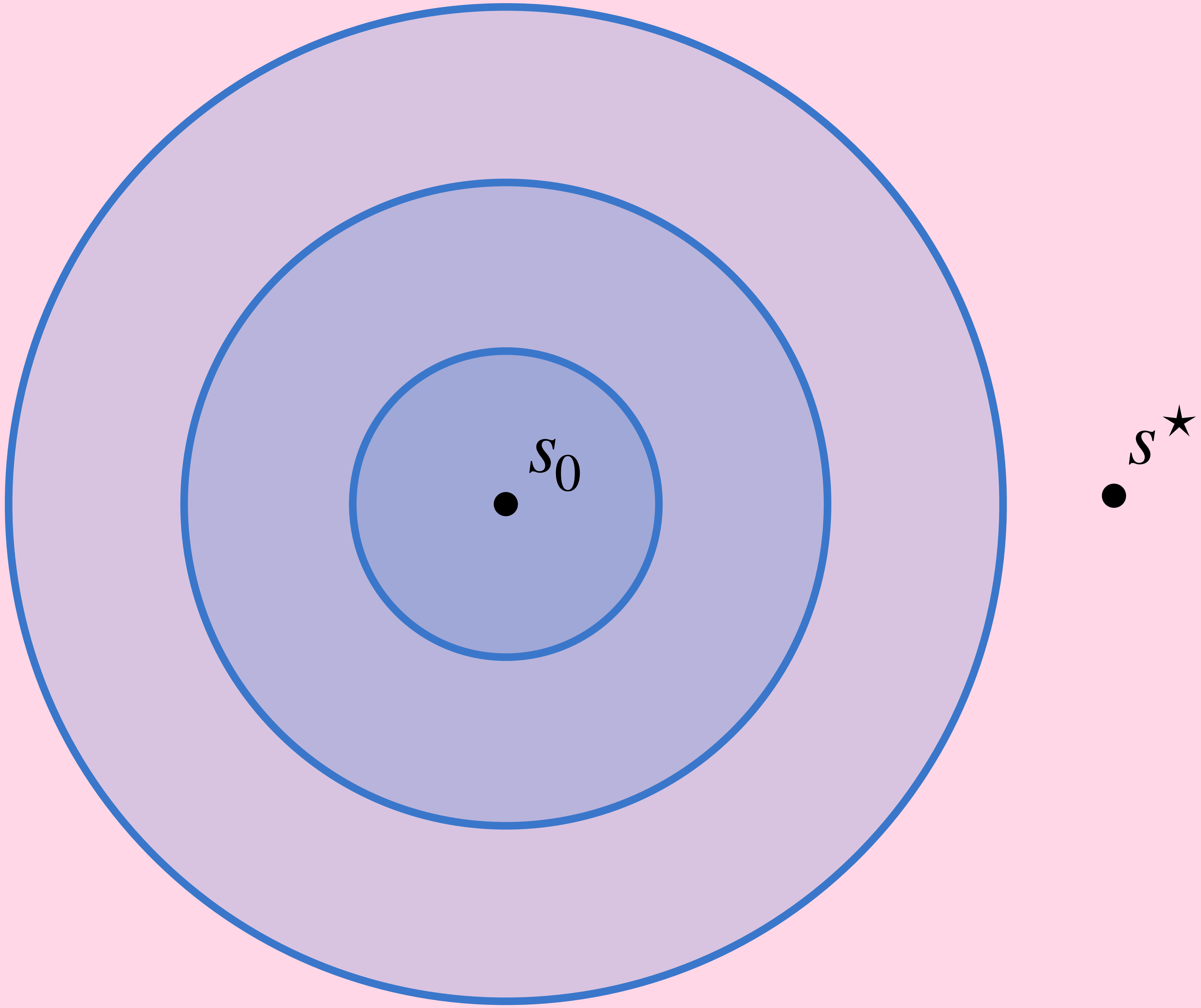


Starting in the upper left state, probably not good to move right before sucking

How do we know?

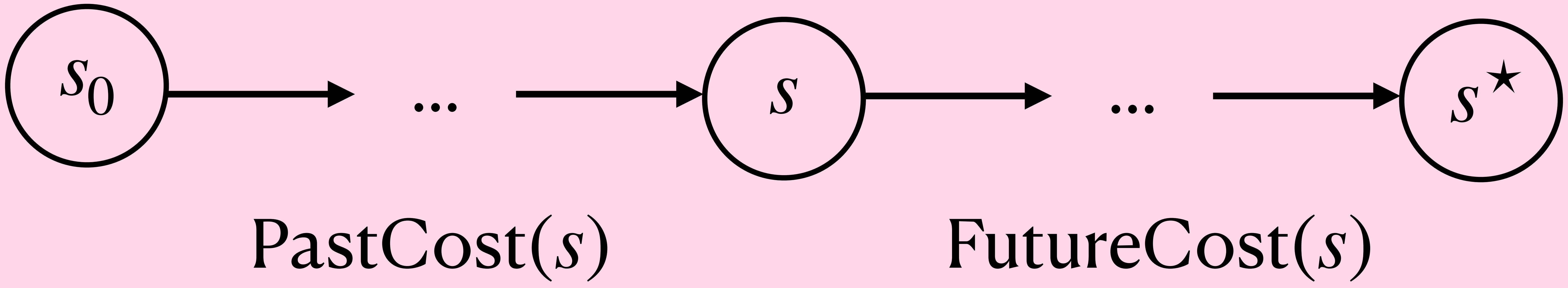
Uniform cost frontier is a good idea.

But why bother searching in this direction?



Heuristic functions

Consider getting from s_0 to $s^\star$ on a path through s.



UCS and BFS work by maintaining a frontier of **uniform PastCost**.

FutureCost is unknown, otherwise we could immediately find an optimal solution.

But we can **estimate** FutureCost(s) with a simple **heuristic** $h(s)$.

Naïve Idea

- If we had access to FutureCost, then an optimal algorithm is to always expand the node that minimizes FutureCost.
- If all we have is an estimate h, then why not pick the node that minimizes h?
- This is called **greedy search**.

[greedy search examples]

A* Search

UCS

- Maintains a frontier of uniform PastCost
- **Correct** but **slow**.

Greedy search

- Chooses the node that minimizes h
- **Incorrect** but **potentially fast**.

A*

- Maintains a frontier of uniform PastCost $+ h$
- **Sometimes correct** and **potentially fast**.

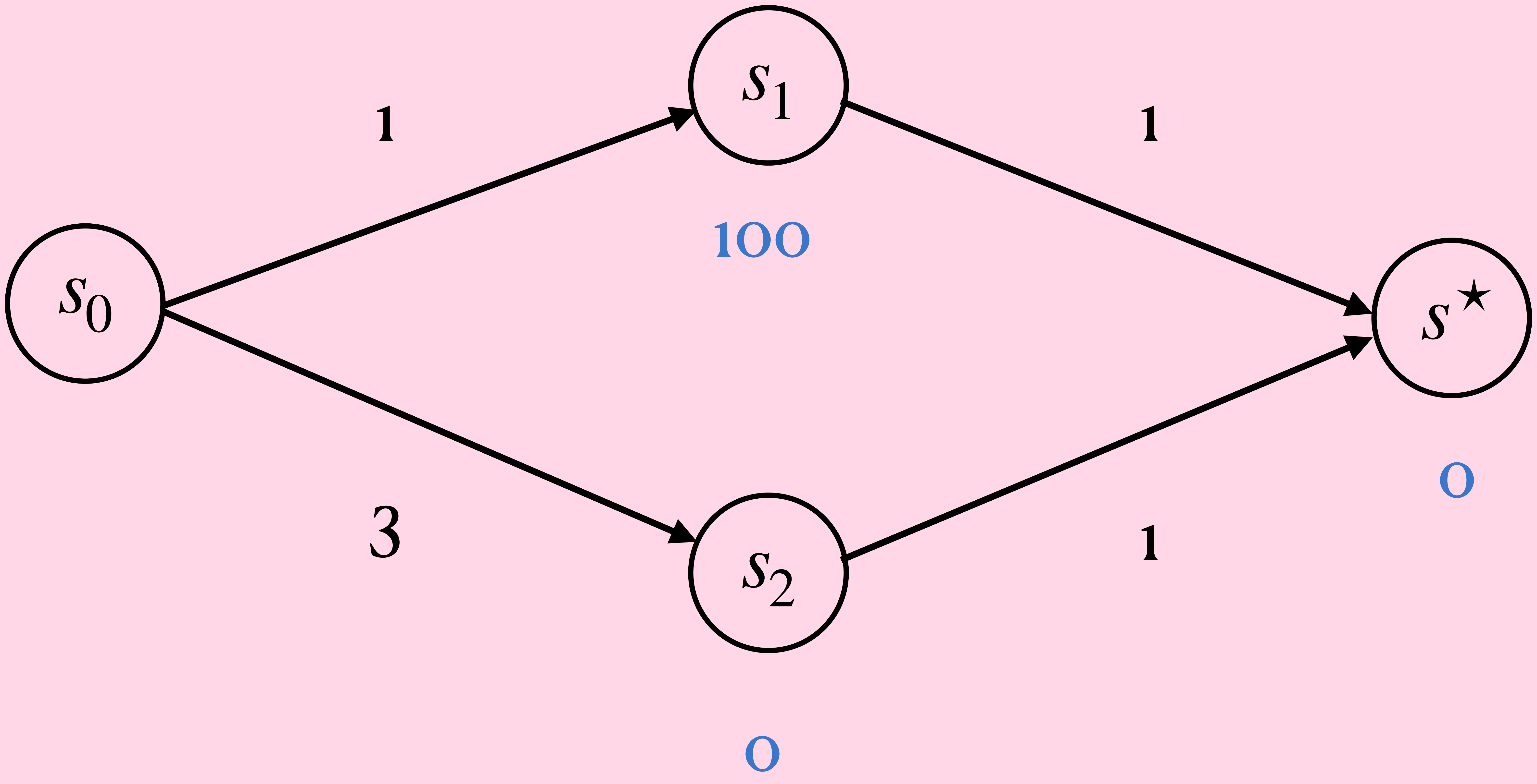
A* vs. Greedy

Problem: short-term greediness can get you into long-term trouble (true for all greedy algorithms in computer science and in life).

Key insight: computing PastCost is easy (just accumulate edge costs), and helps us realize when a prior greedy decision has led us astray.

[A maze example]*

A* can be wrong



Action costs

h