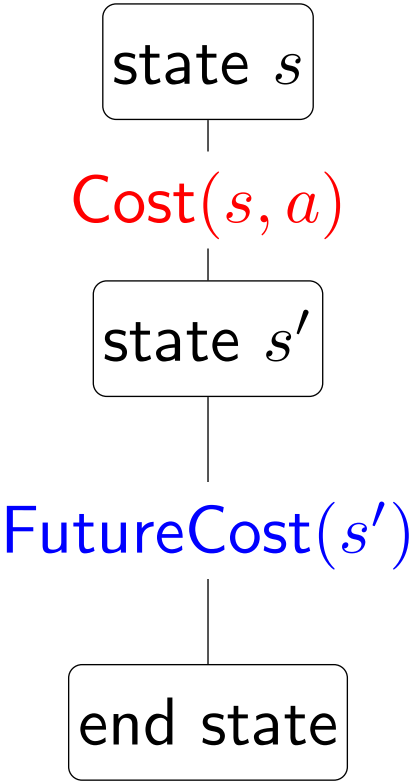
Search: dynamic programming

Dynamic programming



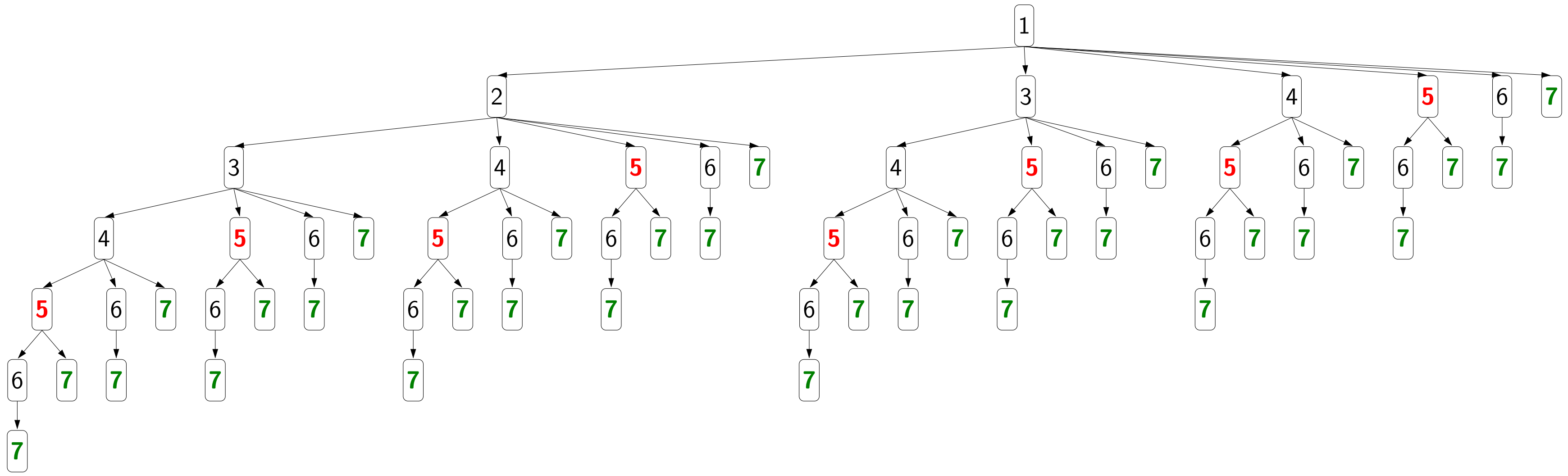
Minimum cost path from state s to a end state:

$$\text{FutureCost}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s, a) + \text{FutureCost}(\text{Succ}(s, a))] & \text{otherwise} \end{cases}$$

Motivating task

Example: route finding

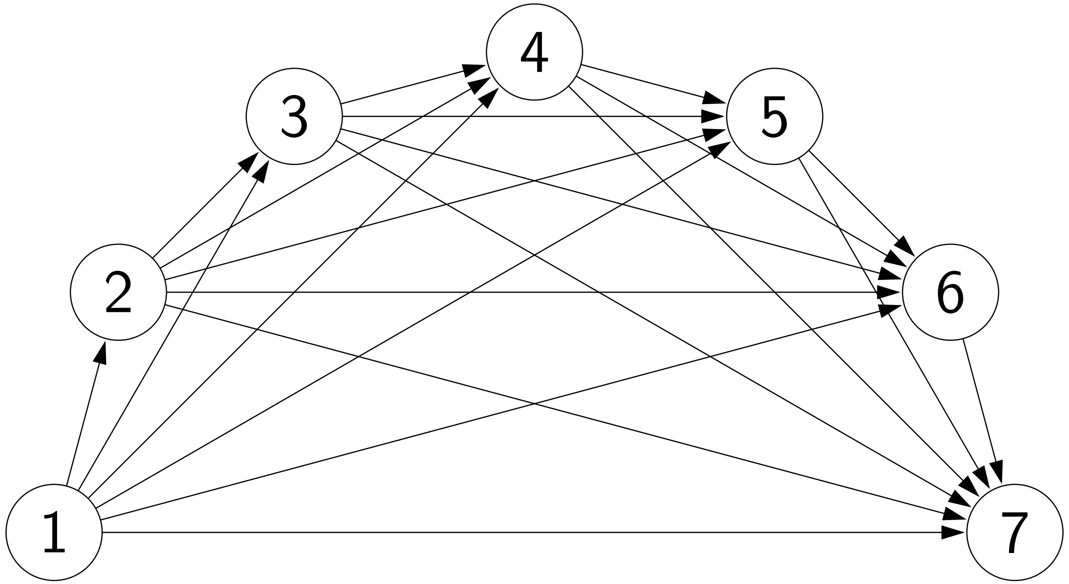
Find the minimum cost path from city 1 to city n, only moving forward. It costs c_{ij} to go from i to j.



Observation: future costs only depend on current city

Dynamic programming

State: ~~past sequence of actions~~ current city



Exponential saving in time and space!

Dynamic programming

Algorithm: dynamic programming

def DynamicProgramming(s):

  If already computed for s, return cached answer.

  If IsEnd(s): return solution

  For each action $a \in$ Actions(s): ...

[semi-live solution: `Dynamic Programming`]

Assumption: acyclicity

The state graph defined by Actions(s) and Succ(s, a) is acyclic.

Dynamic programming

Key idea: state

A **state** is a summary of all the past actions sufficient to choose future actions **optimally**.

past actions (all cities)	1 3 4 6
state (current city)	1 3 4 6

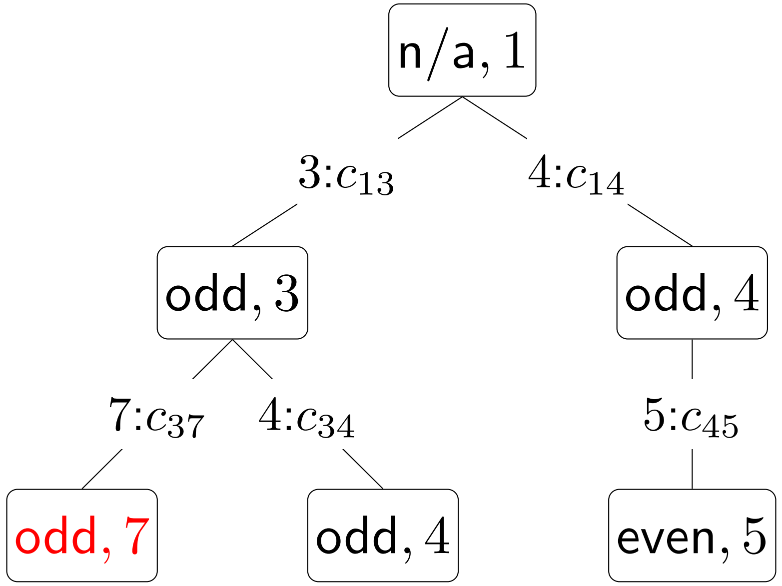
Handling additional constraints

Example: route finding

Find the minimum cost path from city 1 to city n, only moving forward. It costs c_{ij} to go from i to j.

Constraint: Can't visit three odd cities in a row.

State: (whether previous city was odd, current city)



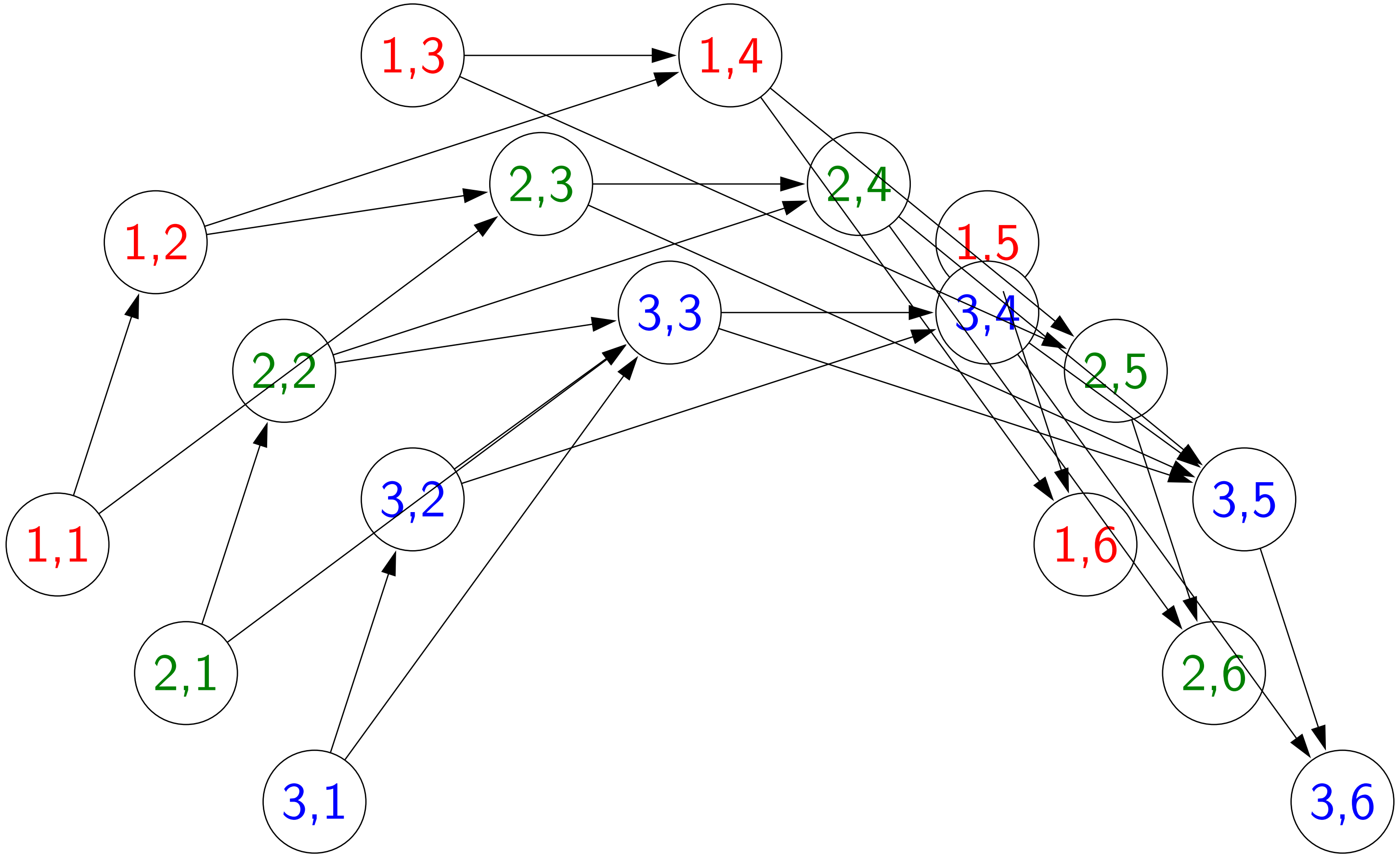
answer in chat

Question

Objective: travel from city 1 to city n, visiting at least 3 odd cities. What is the minimal state?

State graph

State: (min(number of odd cities visited, 3), current city)



answer in chat

Question

Objective: travel from city 1 to city n, visiting more odd than even cities. What is the minimal state?

Summary

- State: summary of past actions sufficient to choose future actions optimally

- Dynamic programming: backtracking search with **memoization** — potentially exponential savings

Dynamic programming only works for acyclic graphs...what if there are cycles?

Dynamic Programming Review

state s

$\text{Cost}(s, a)$

state s'

$\text{FutureCost}(s')$

end state

$$\text{FutureCost}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s, a) + \text{FutureCost}(\text{Succ}(s, a))] & \text{otherwise} \end{cases}$$

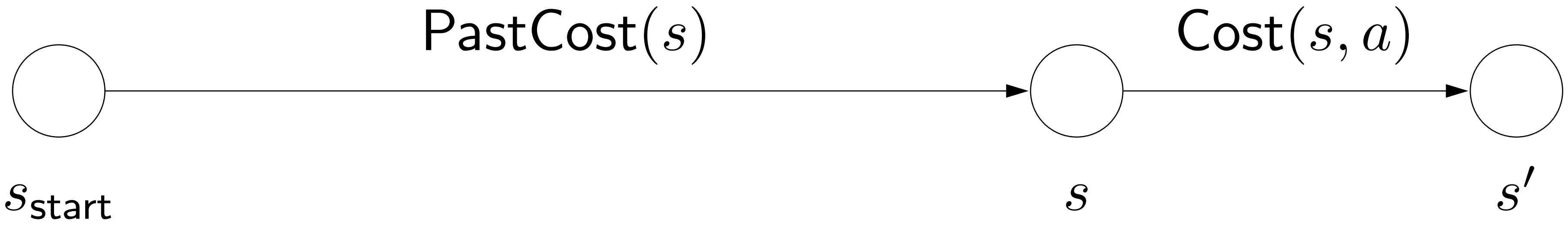
Key idea: state

A **state** is a summary of all the past actions sufficient to choose future actions **optimally**.

Search: uniform cost search

Ordering the states

Observation: prefixes of optimal path are optimal



Key: if graph is acyclic, dynamic programming makes sure we compute $\text{PastCost}(s)$ before $\text{PastCost}(s')$

If graph is cyclic, then we need another mechanism to order states...

Uniform cost search (UCS)

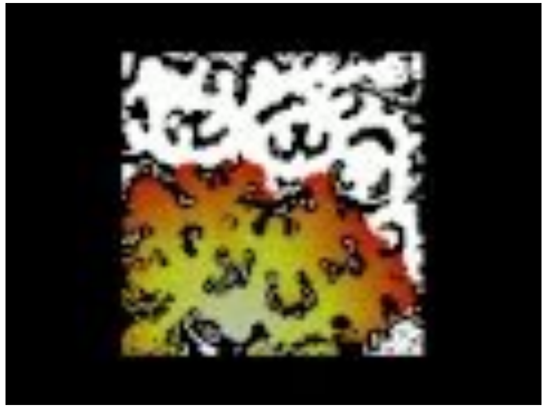
Key idea: state ordering

UCS enumerates states in order of increasing past cost.

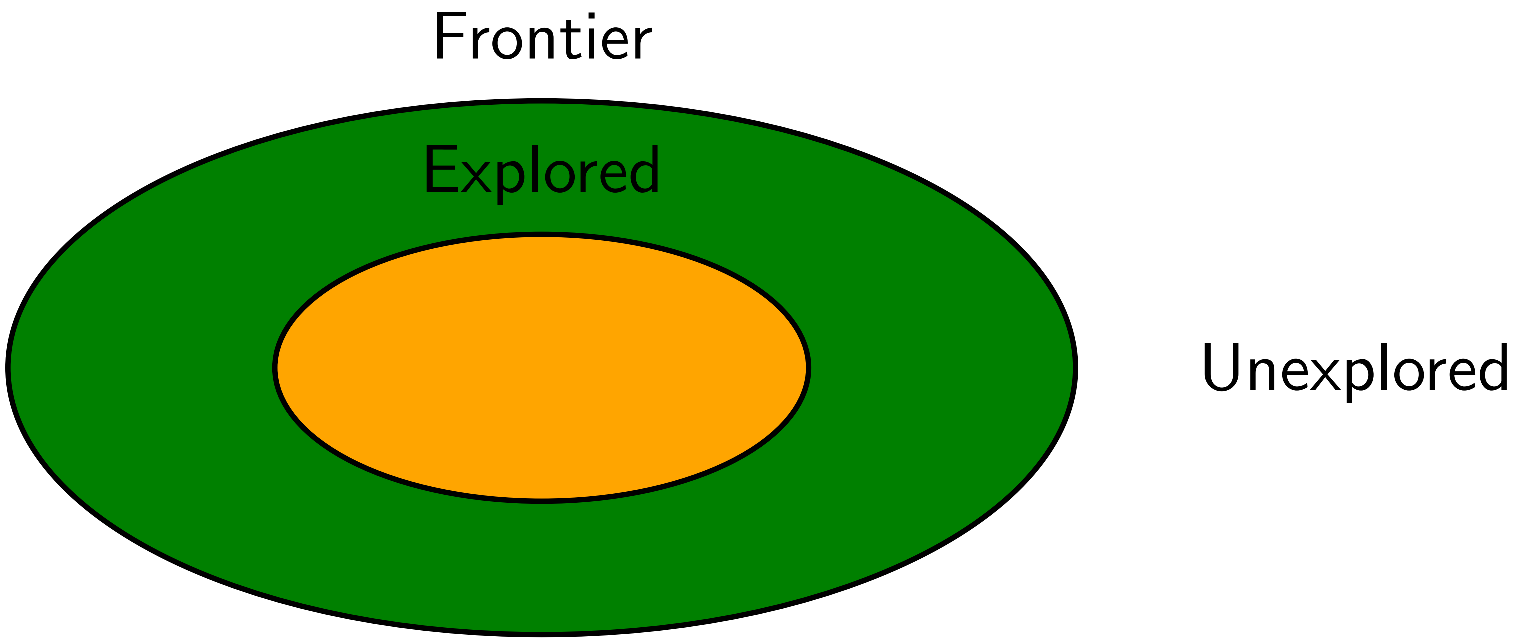
Assumption: non-negativity

All action costs are non-negative: $\text{Cost}(s, a) \geq 0$.

UCS in action:



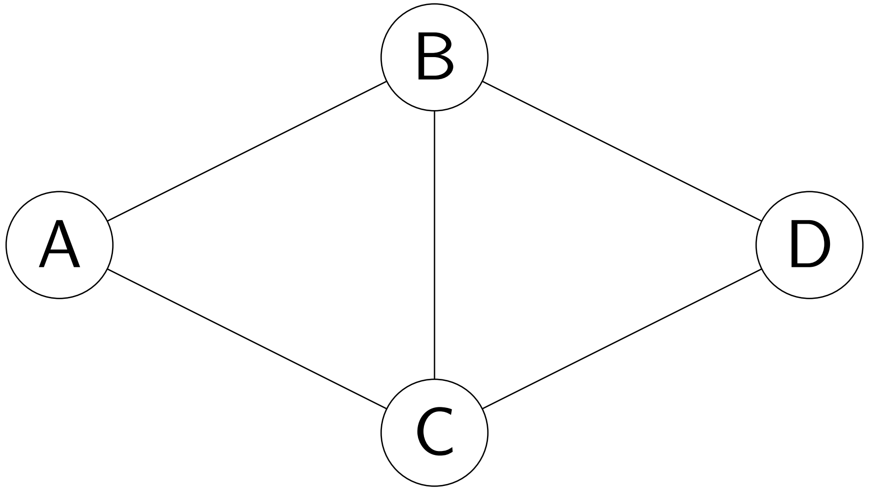
High-level strategy



- Explored: states we've found the optimal path to
- Frontier: states we've seen, still figuring out how to get there cheaply
- Unexplored: states we haven't seen

Uniform cost search example

Example: UCS example



Start state: A, end state: D

[whiteboard]

Minimum cost path:

A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D with cost 3

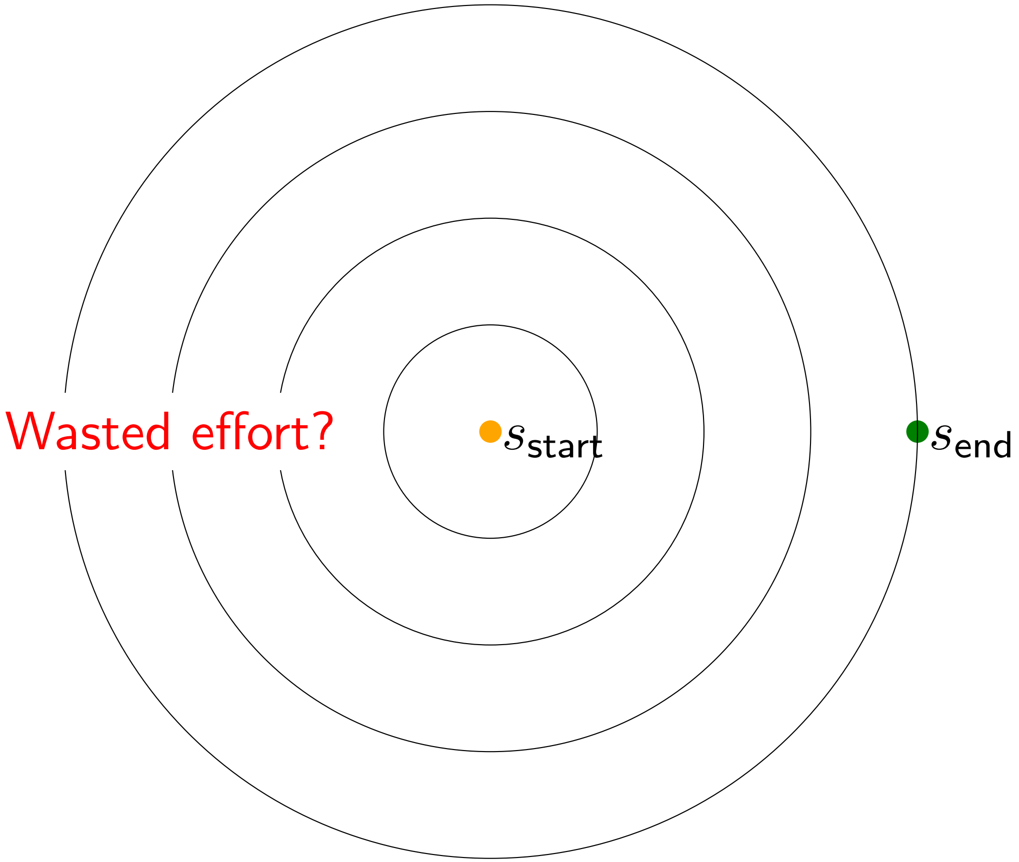
Search: A*

A* algorithm

UCS in action:

A* in action:

Can uniform cost search be improved?

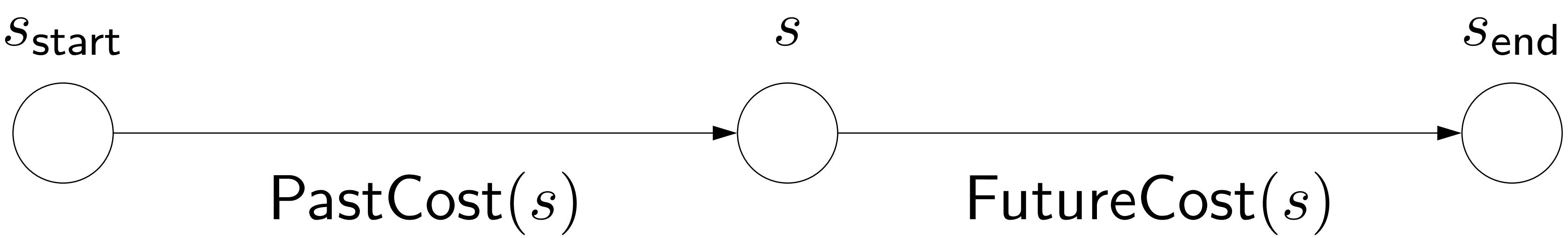


Problem: UCS orders states by cost from s_{start} to s

Goal: take into account cost from s to s_{end}

Exploring states

UCS: explore states in order of $\text{PastCost}(s)$



Ideal: explore in order of $\text{PastCost}(s) + \text{FutureCost}(s)$

A*: explore in order of $\text{PastCost}(s) + h(s)$

Definition: Heuristic function

A heuristic $h(s)$ is any estimate of $\text{FutureCost}(s)$.

A* search

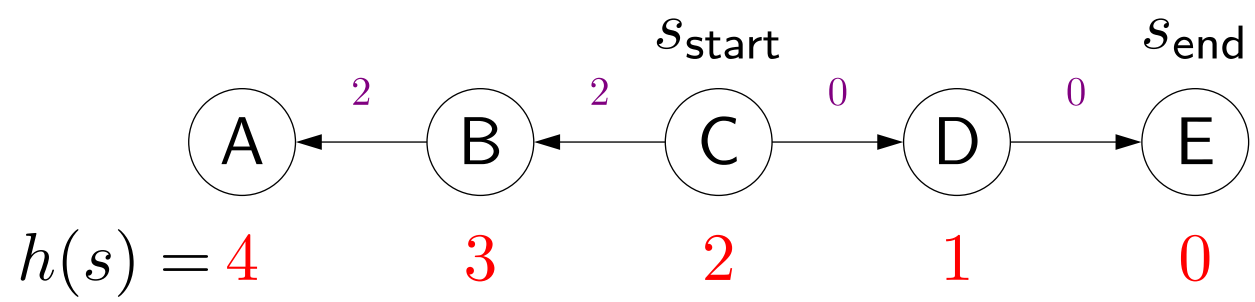
Algorithm: A* search [Hart/Nilsson/Raphael, 1968]

Run uniform cost search with **modified edge costs**:

$$\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)$$

Intuition: add a penalty for how much action a takes us away from the end state

Example:



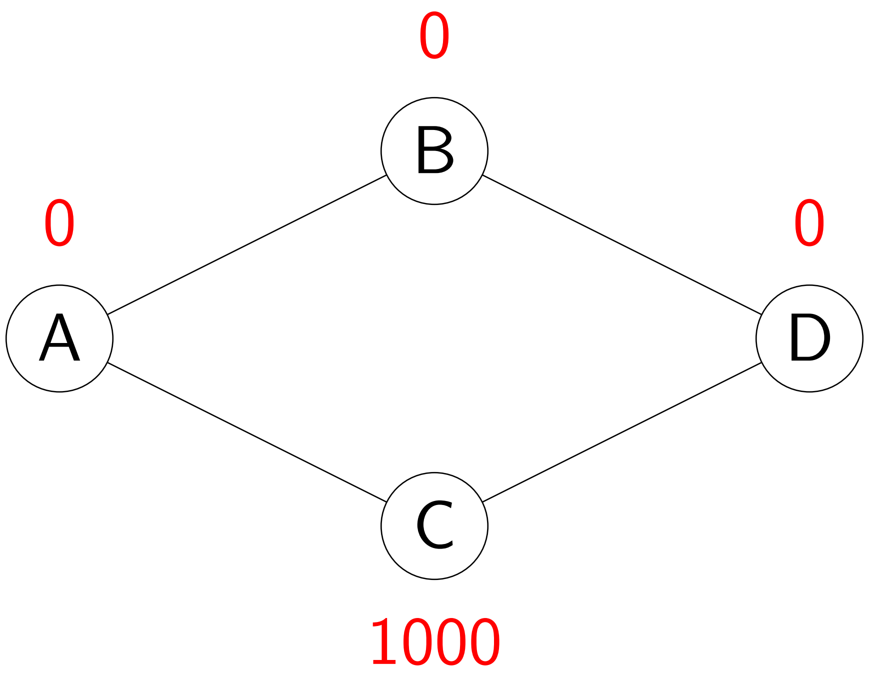
$$\text{Cost}'(C,B) = \text{Cost}(C,B) + h(B) - h(C) = 1 + (3-2) = 2$$

An example heuristic

Will any heuristic work?

No.

Counterexample:



Doesn't work because of **negative modified edge costs**!

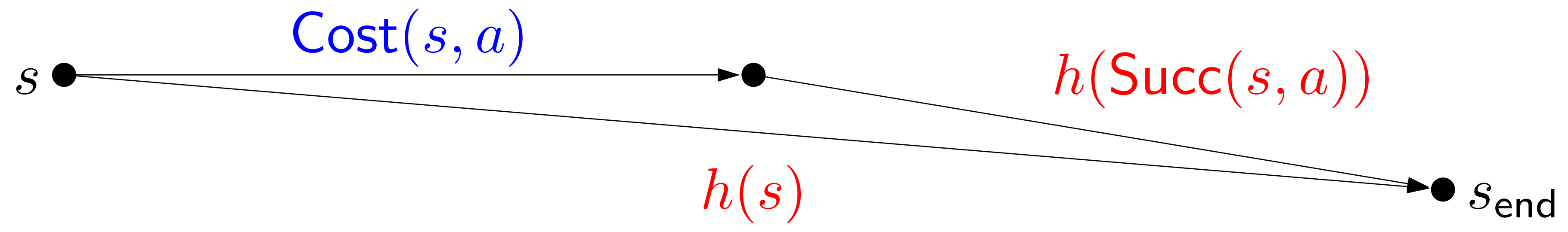
Consistent heuristics

Definition: consistency

A heuristic h is **consistent** if

- $\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

Condition 1: needed for UCS to work (triangle inequality).



Condition 2: $\text{FutureCost}(s_{\text{end}}) = 0$ so match it.

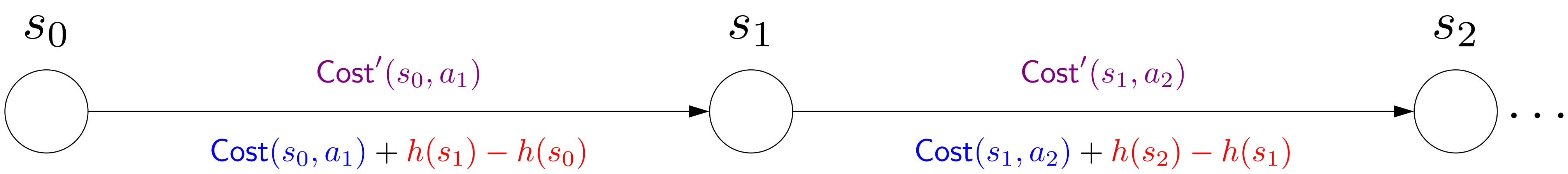
Correctness of A*

Proposition: correctness

If h is consistent, A* returns the minimum cost path.

Proof of A* correctness

- Consider any path $[s_0, a_1, s_1, \dots, a_L, s_L]$:



- Key identity:

$$\underbrace{\sum_{i=1}^{L} \text{Cost}'(s_{i-1}, a_i)}_{\text{modified path cost}} = \underbrace{\sum_{i=1}^{L} \text{Cost}(s_{i-1}, a_i)}_{\text{original path cost}} + \underbrace{h(s_L) - h(s_0)}_{\text{constant}}$$

- Therefore, A* (finding the minimum cost path using modified costs) solves the original problem (even though edge costs are all different!)

Efficiency of A*

Theorem: efficiency of A*

A* explores all states s satisfying
$\text{PastCost}(s) \leq \text{PastCost}(s_{\text{end}}) - h(s)$

Interpretation: the larger $h(s)$, the better

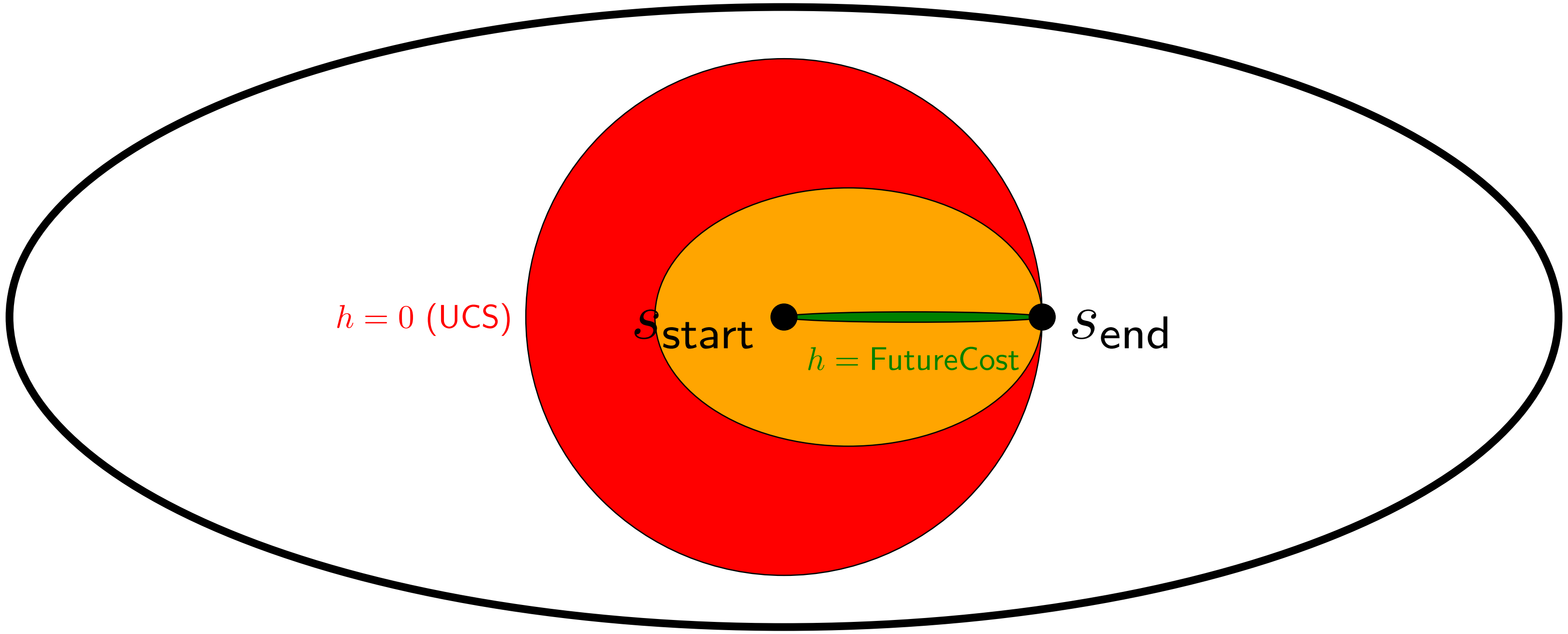
Proof: A* explores all s such that

$$\text{PastCost}(s) + h(s)$$

$$\leq$$

$$\text{PastCost}(s_{\text{end}})$$

Amount explored

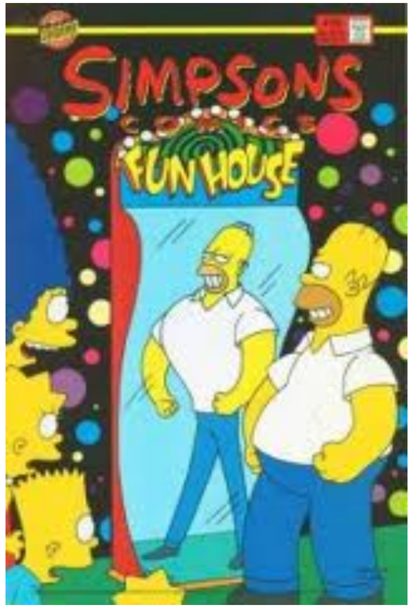


- If $h(s) = 0$, then A* is same as UCS.
- If $h(s) = \text{FutureCost}(s)$, then A* only explores nodes on a minimum cost path.
- Usually $h(s)$ is somewhere in between.

A* search

Key idea: distortion

A* distorts edge costs to favor end states.



Admissibility

Definition: admissibility

A heuristic $h(s)$ is admissible if

$$h(s) \leq \text{FutureCost}(s)$$

Intuition: admissible heuristics are optimistic

Theorem: consistency implies admissibility

If a heuristic $h(s)$ is **consistent**, then $h(s)$ is **admissible**.

Proof: use induction on $\text{FutureCost}(s)$

Search: A* relaxations

How do we get good heuristics? Just relax...

Relaxation

Intuition: ideally, use $h(s) = \text{FutureCost}(s)$, but that's as hard as solving the original problem.

Key idea: relaxation

Constraints make life hard. Get rid of them.

But this is just for the heuristic!

Relaxation overview

reduce costs

remove constraints

closed form solution

easier search

independent subproblems

combine heuristics using max