Agents that Plan Ahead: A* Search

Russell and Norvig: Chapter 3.1-3.4, 3.5-3.6

CSE 240: Winter 2023

Lecture 4

Guest Lecture: Razvan Marinescu

Announcements

- Assignment 1 is up
- Quizzes will be all remote on Canvas.

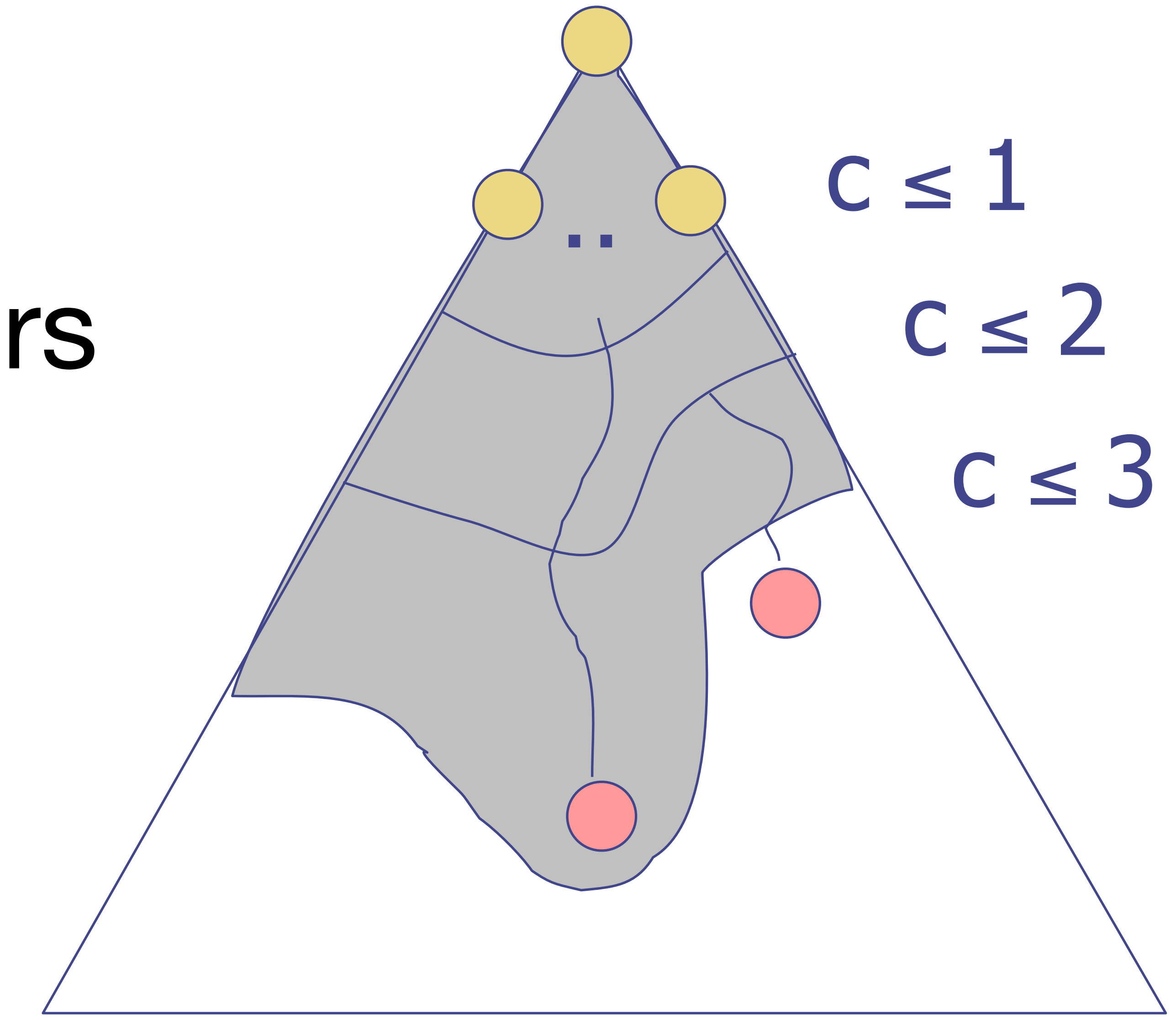
Agenda

Today

- Informed search strategies
    - A* search algorithm
    - Heuristics

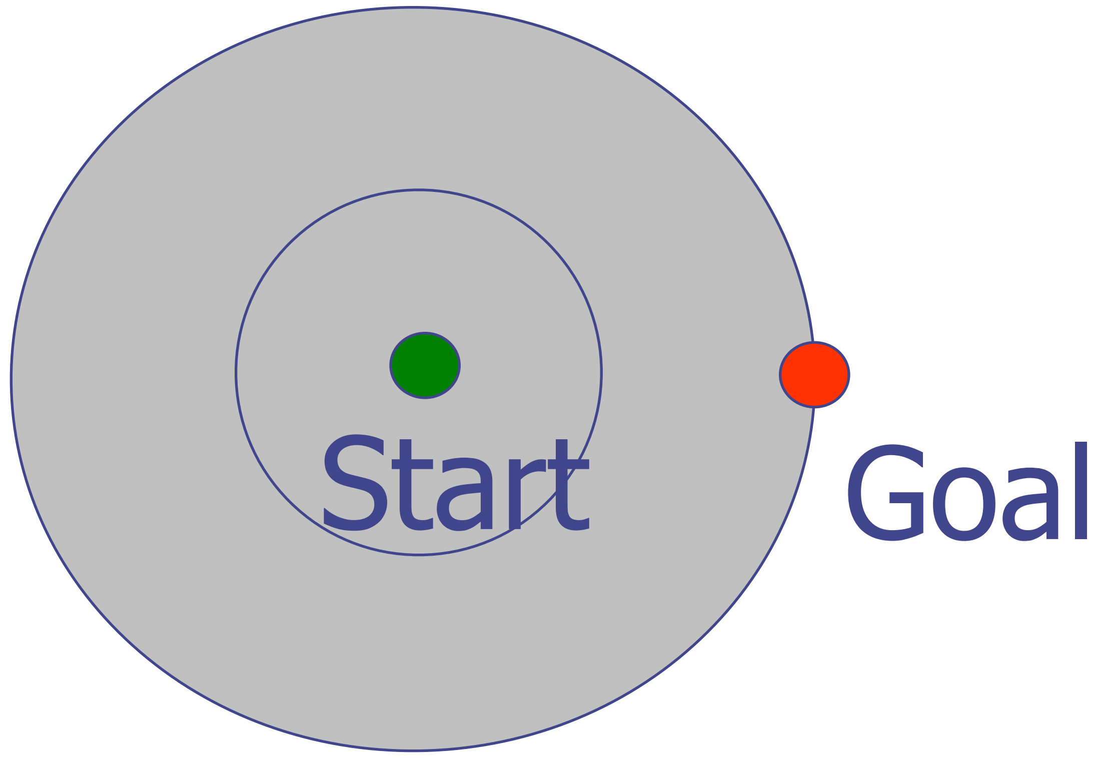
Recap: Uniform Cost Issues

- Remember: explores increasing cost contours



- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

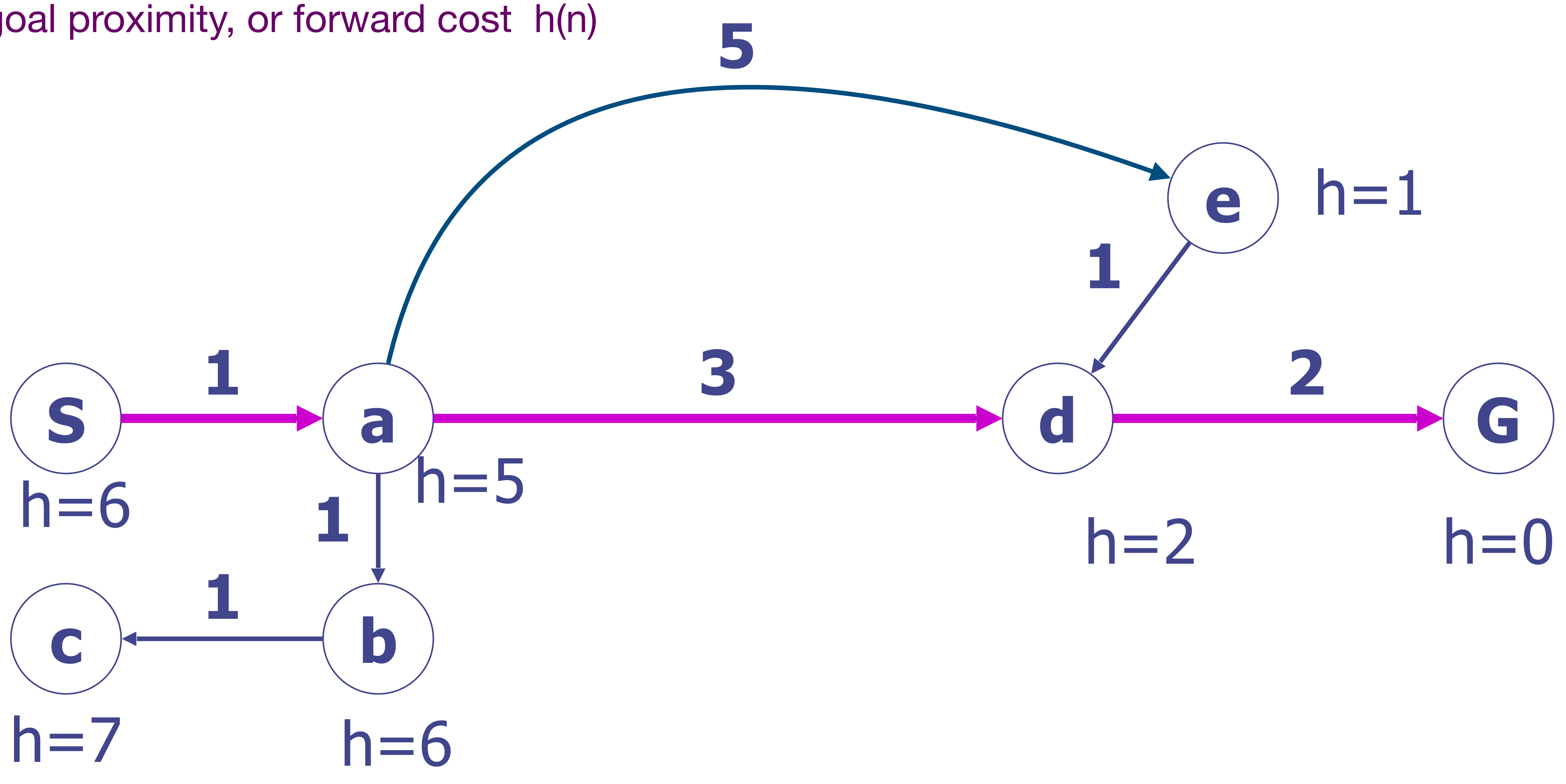


A* Search

Combining UCS and Greedy

$$f(n) = g(n) + h(n)$$

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



- A* Search orders by the sum: f(n) = g(n) + h(n)

Node	Fringe	f(n)
s	s->a	6
s->a	s->a->b	8
s->a	s->a->d	6
s->a	s->a->e	7

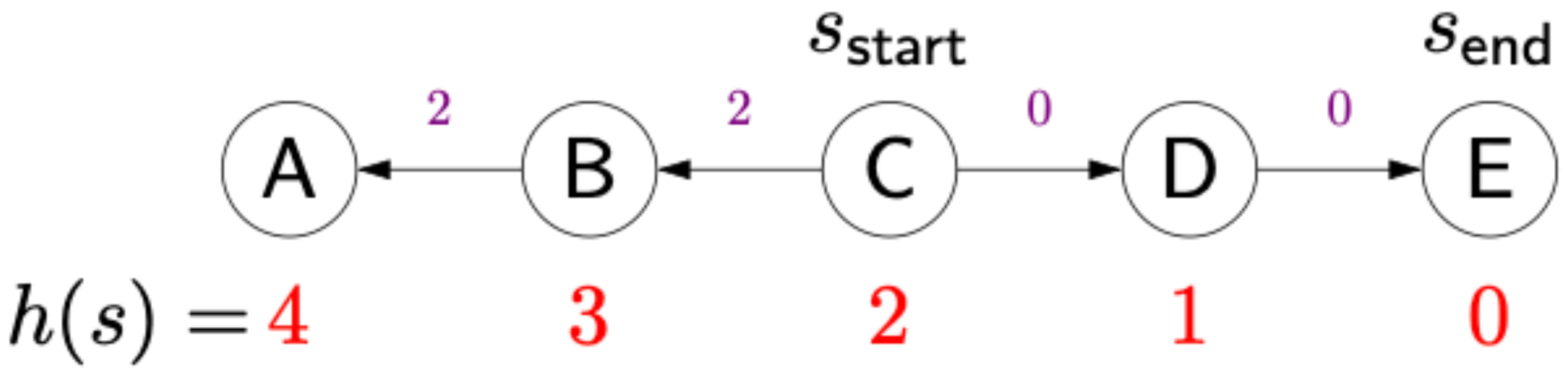
Example: Teg Grenager

Another Way to Implement A*

Run UCS with modified edge costs in order to account for closeness to the goal state

$$Cost'(s,a) = Cost(s,a) + h\big(succ(s,a)\big) - h(s)$$

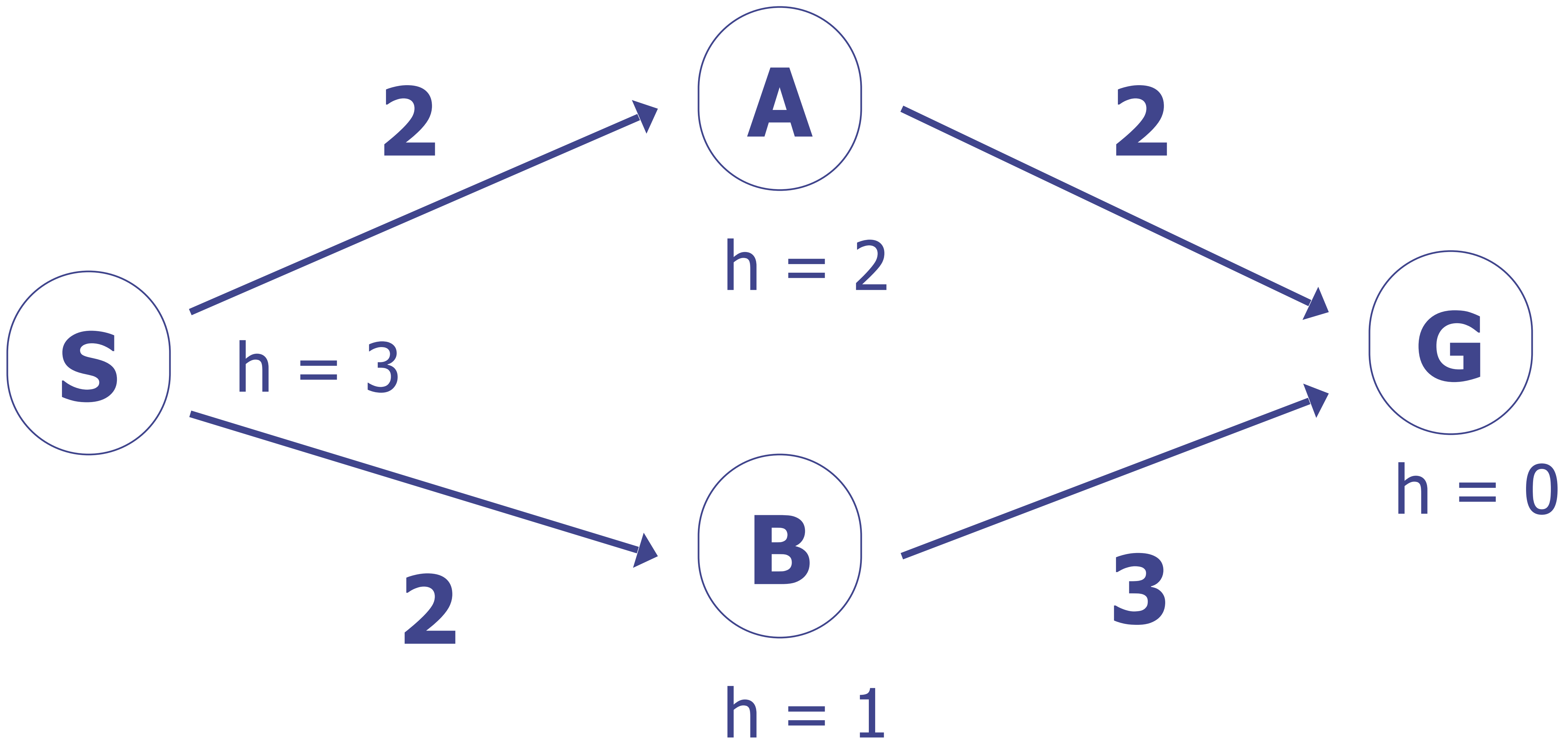
- Intuition: add a penalty for how much action 'a' takes us away from the end state



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

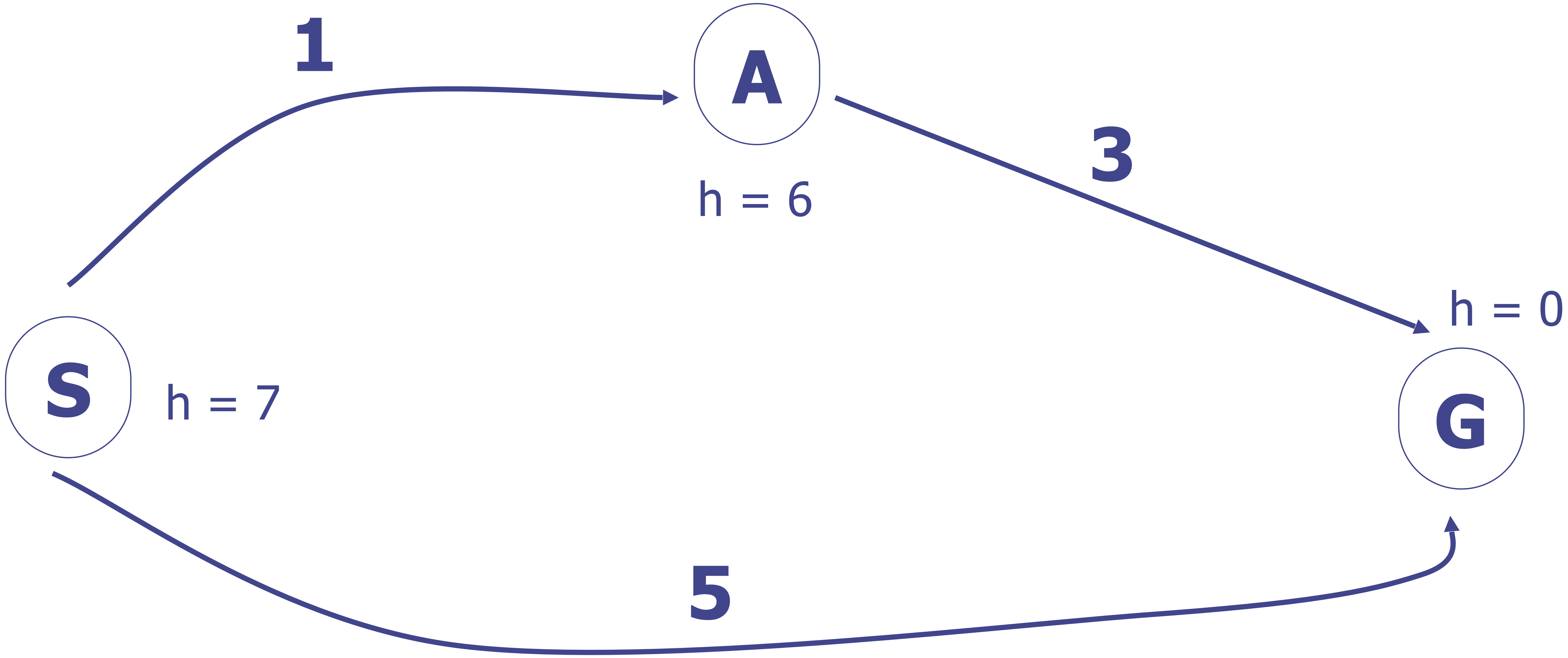
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

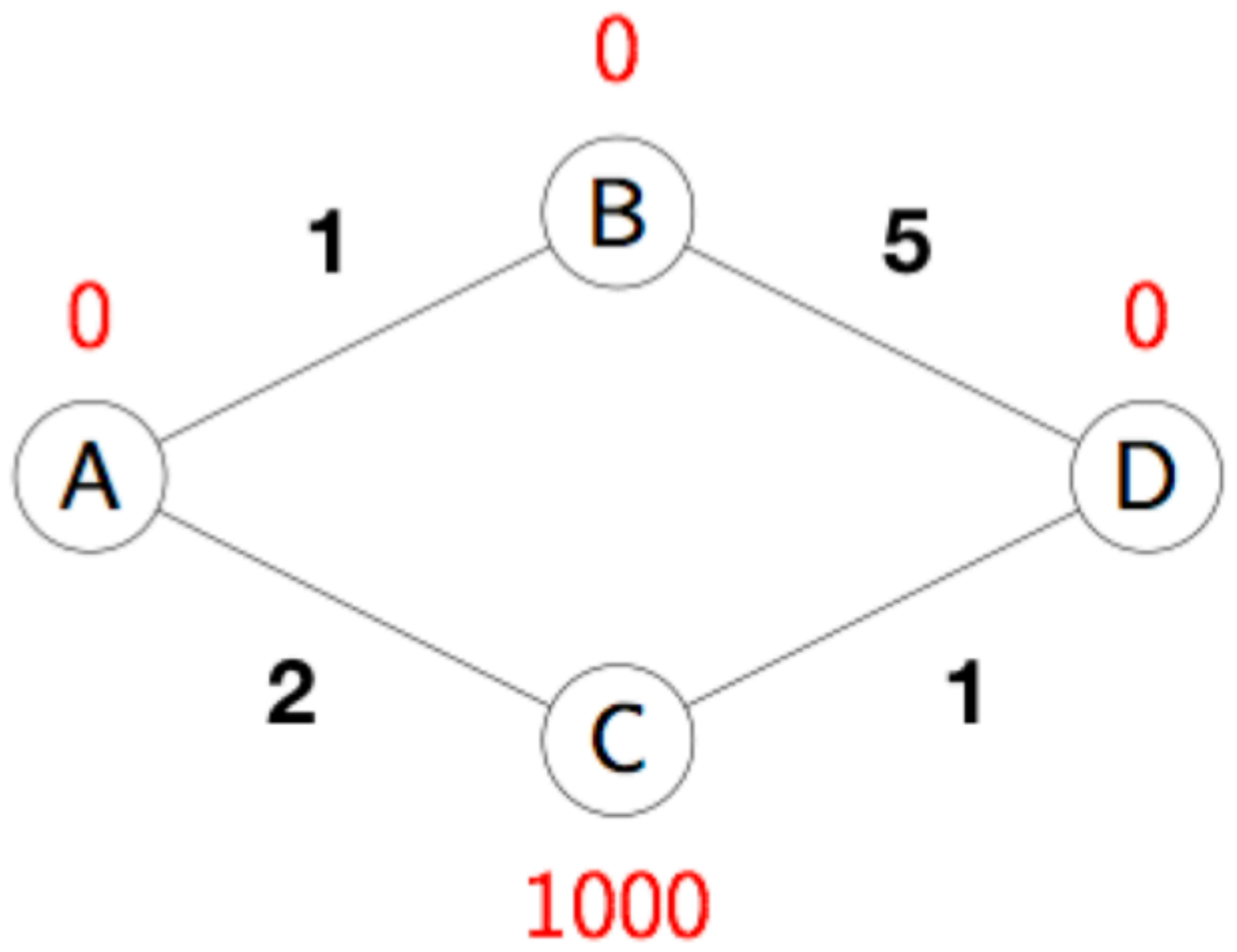
Is A* Optimal?



- What went wrong?
- Heuristic estimate at node A higher than true cost
- We need estimates to be less than actual costs!

An Example Heuristic

- Would any heuristic work?
- Doesn’t work because of the negative modified edge costs (or being pessimistic about the correct path)



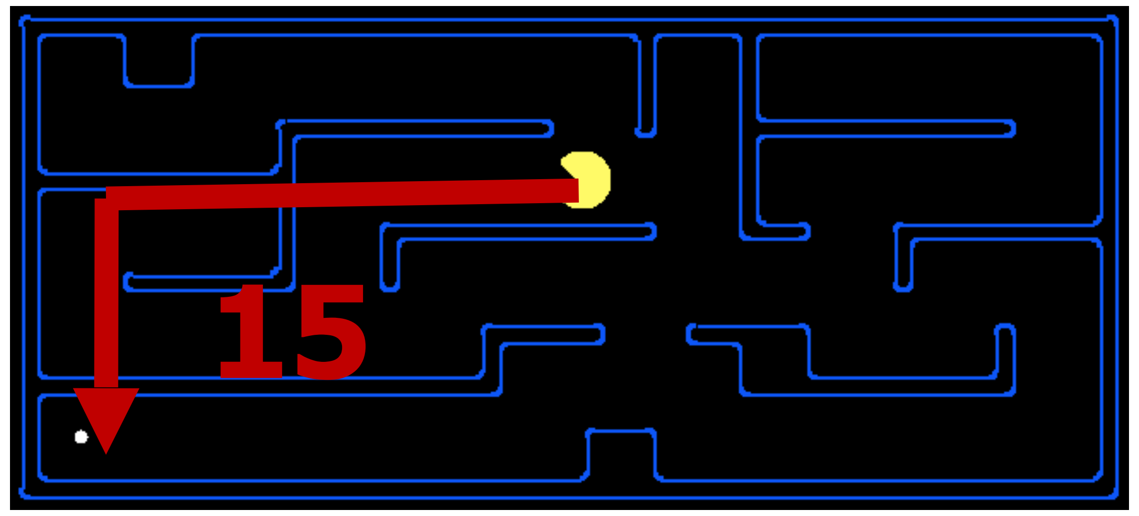
Admissible Heuristics

- A heuristic h is admissible (optimistic) if:

$$h(n) \leq h^*(n)$$

- where $h^*(n)$ is the true cost to a nearest goal

- Example:

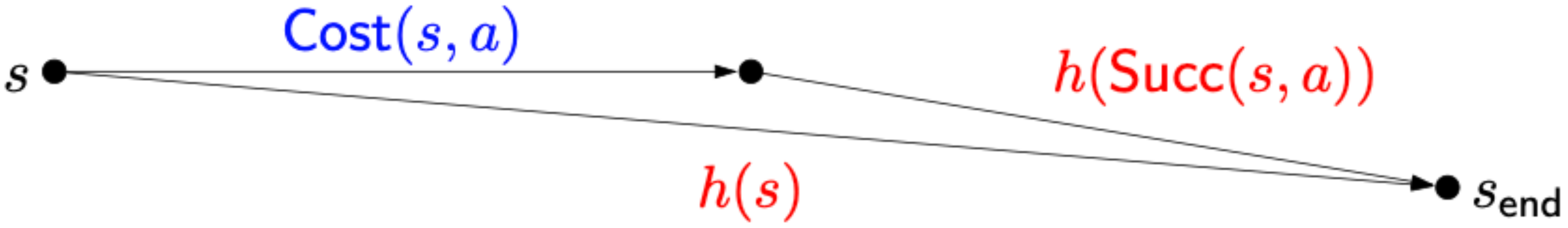


- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Consistent Heuristic

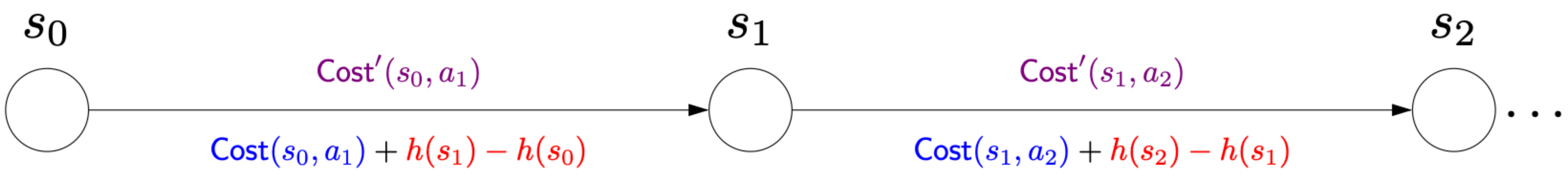
A heuristic h is "**consistent**" if

- $\boldsymbol{Cost'(s,a) = Cost(s,a) + h(succ(s,a)) - h(s) \geq 0}$
- $\boldsymbol{h(s_{end}) = 0}$



Correctness of A*

- If h is consistent, A* returns the minimum cost path.
- Consider any path
- Key identity:

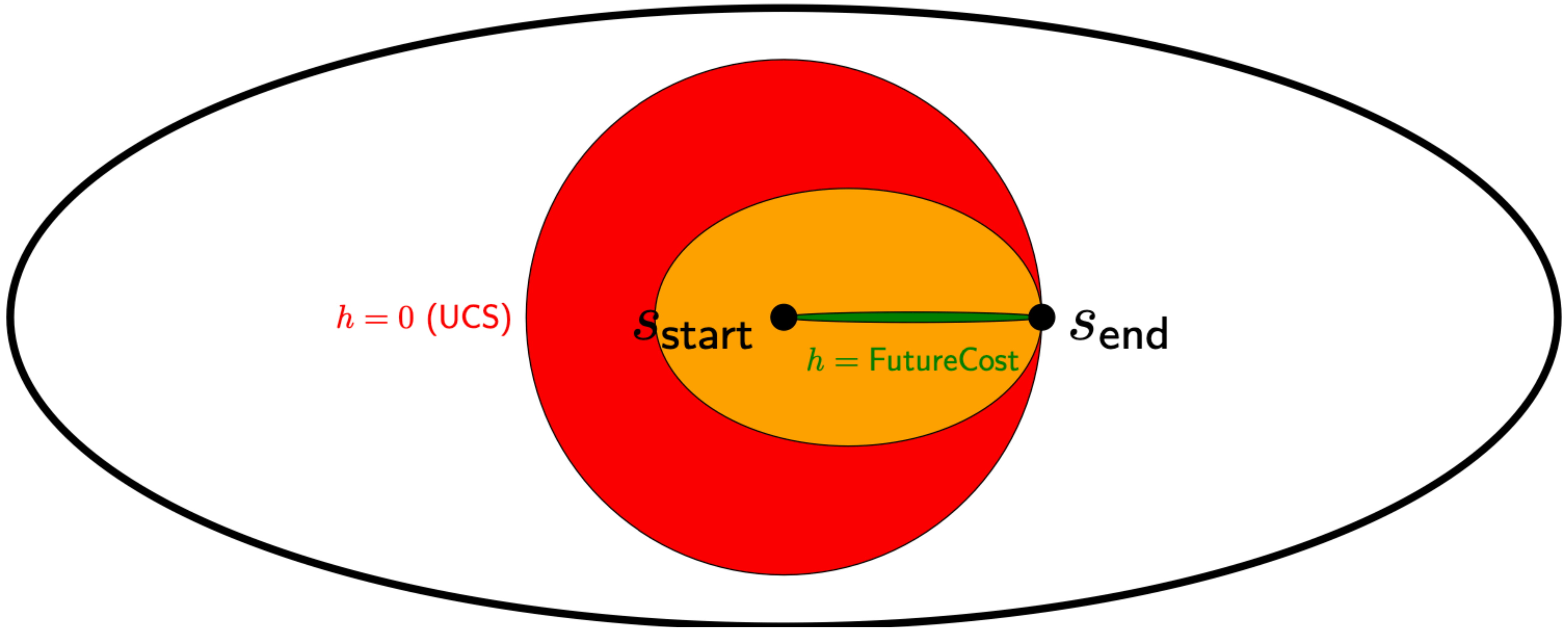


$$\underbrace{\sum_{i=1}^{L} \text{Cost}'(s_{i-1}, a_i)}_{\text{modified path cost}} = \underbrace{\sum_{i=1}^{L} \text{Cost}(s_{i-1}, a_i)}_{\text{original path cost}} + \underbrace{h(s_L) - h(s_0)}_{\text{constant}}$$

- Therefore, A* solves the original problem using UCS, and therefore the algorithm is complete.

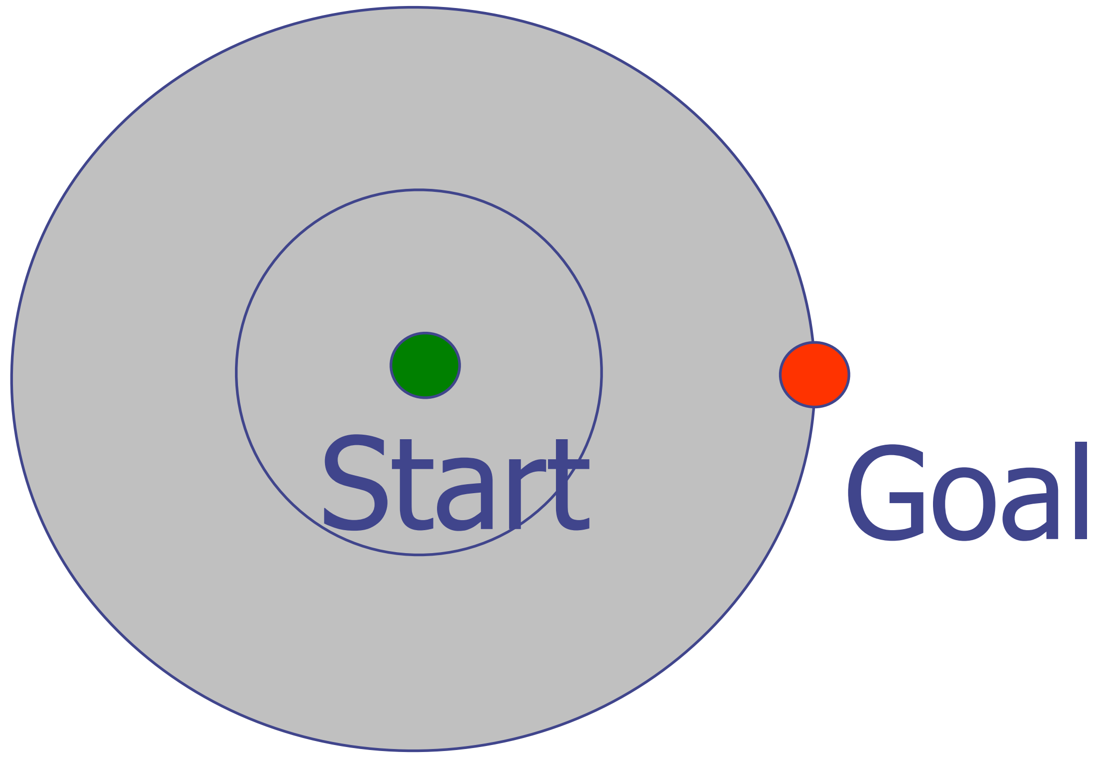
Amount Explored

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

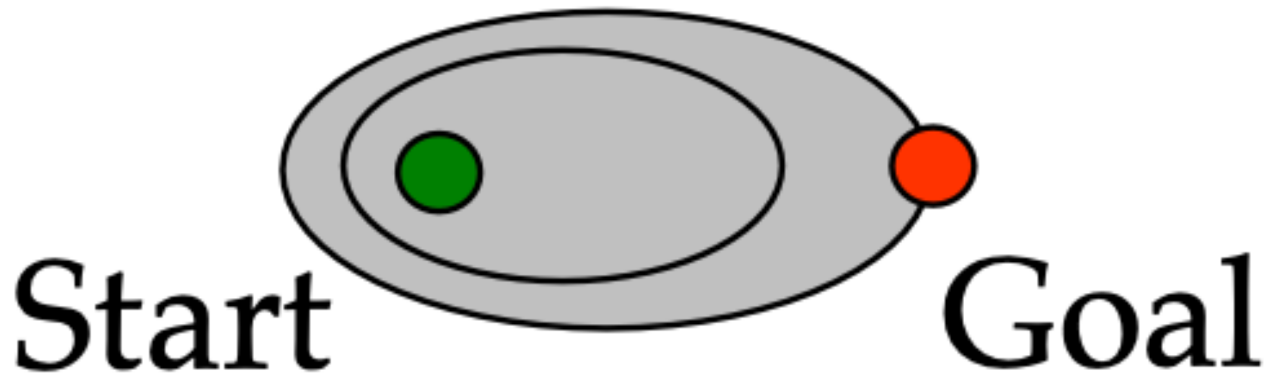


UCS versus A* Contours

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



How Do we Get Good Heuristics?

Just Relax!

Relaxation

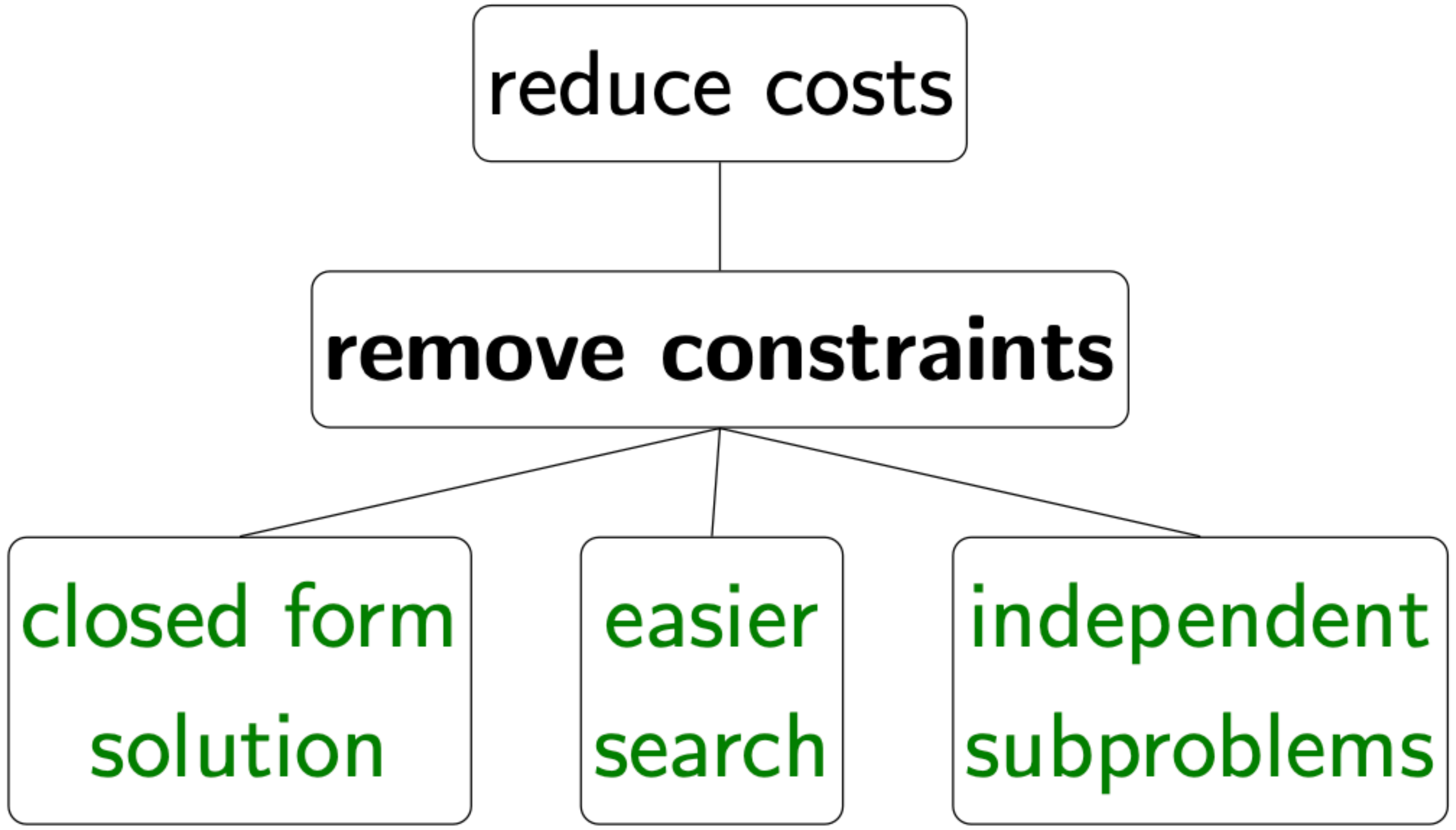
- Ideally, we use h(s) = FutureCost(s), but that's as hard as solving the original problem.

Key idea: relaxation

Constraints make life hard. Get rid of them.

But this is just for the heuristic!

Relaxation Overview

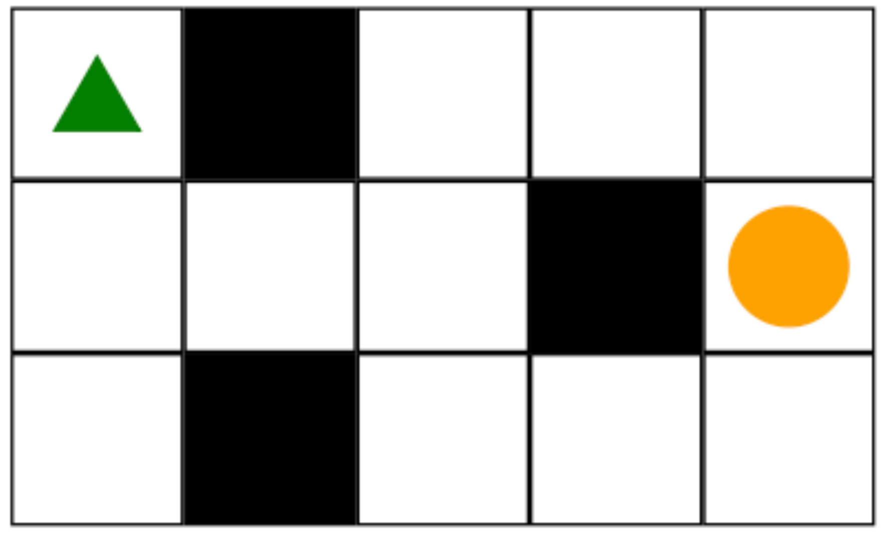


combine heuristics using max

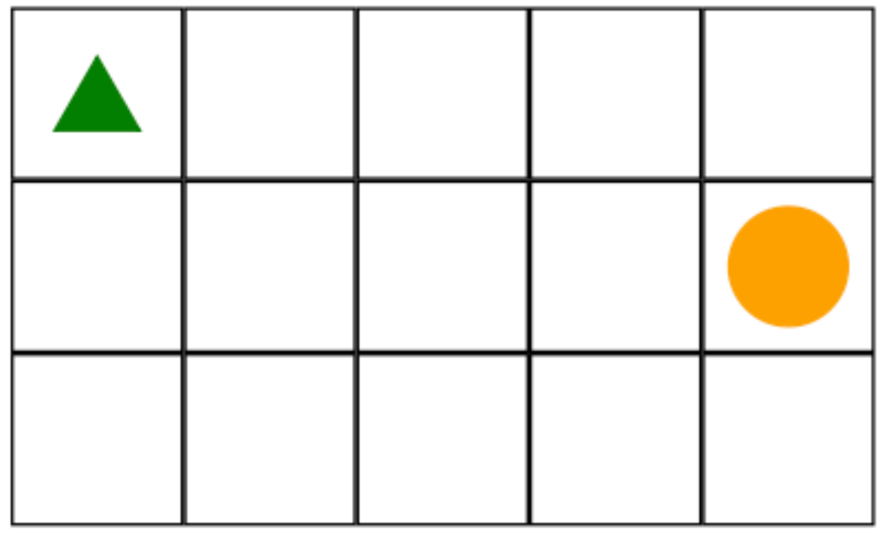
Closed Form Solution

Example: knock down walls

Goal: move from triangle to circle



Hard



Easy

Heuristic:

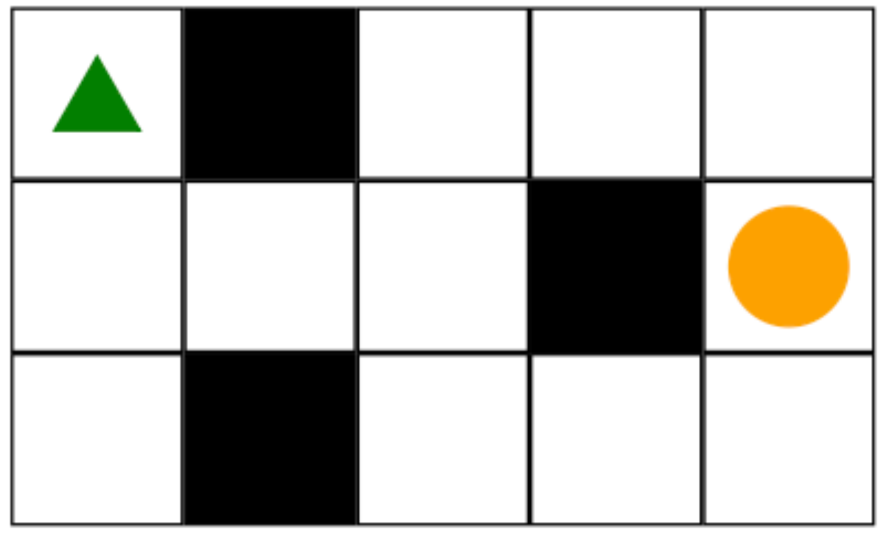
$$h(s) = \text{ManhattanDistance}(s, (2, 5))$$

e.g., $h((1, 1)) = 5$

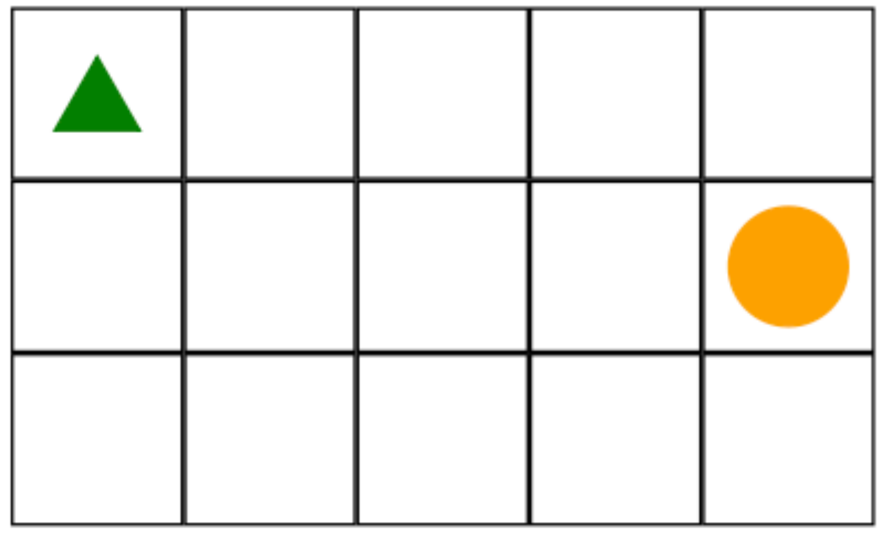
CE 4: What are other Relaxations of this Problem?

Example: knock down walls

Goal: move from triangle to circle



Hard



Easy

Heuristic:

$$h(s) = \text{ManhattanDistance}(s, (2, 5))$$

e.g., $h((1, 1)) = 5$

Easier Search

Example: original problem

Start state: 1

Walk action: from s to $s+1$ (cost: 1)

Tram action: from s to $2s$ (cost: 2)

End state: n

Constraint: can't have more tram actions than walk actions.

State: (location, #walk - #tram)

Number of states goes from $O(n)$ to $O(n^2)$!

Easier Search

Example: relaxed problem

Start state: 1

Walk action: from s to $s+1$ (cost: 1)

Tram action: from s to $2s$ (cost: 2)

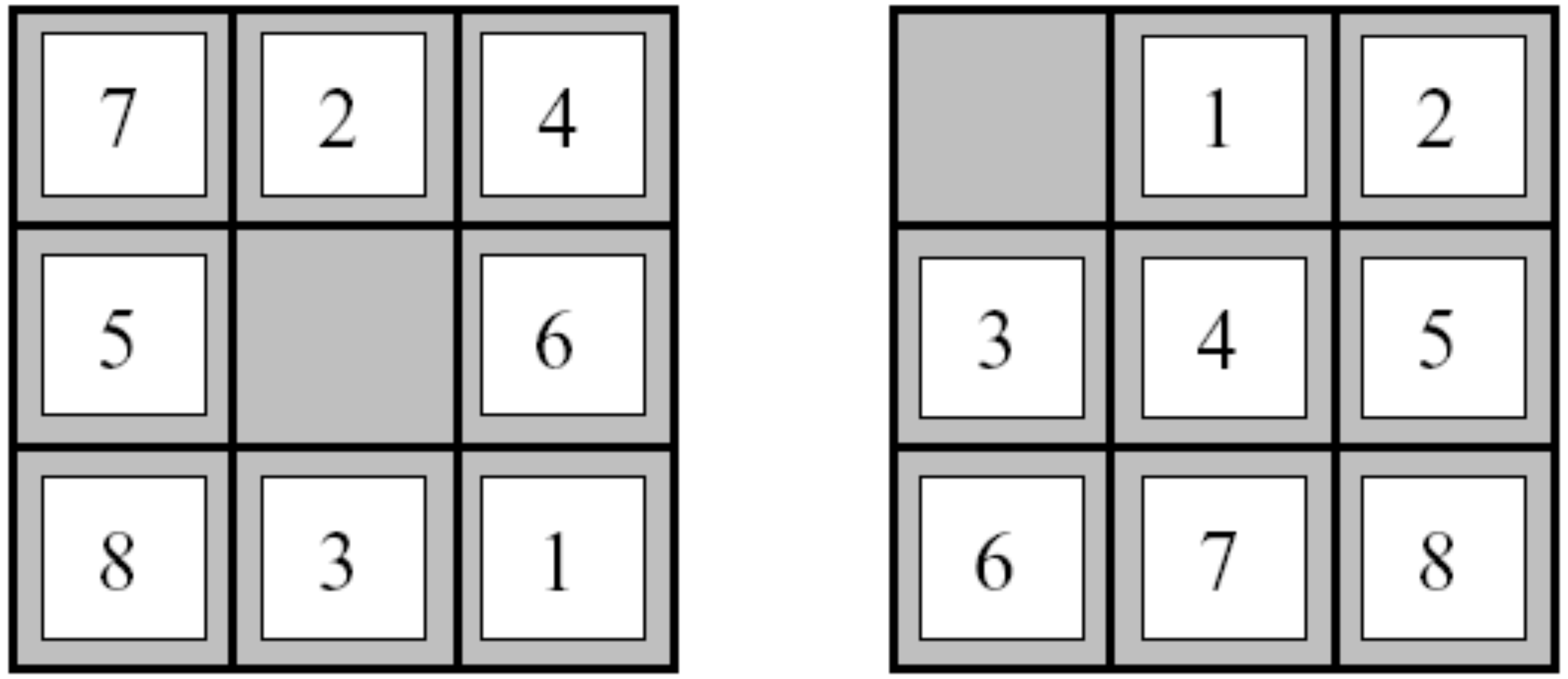
End state: n

~~Constraint: can't have more tram actions than walk actions.~~

Original state: (location, #walk - #tram)

Relaxed state: location

Independent Subproblems

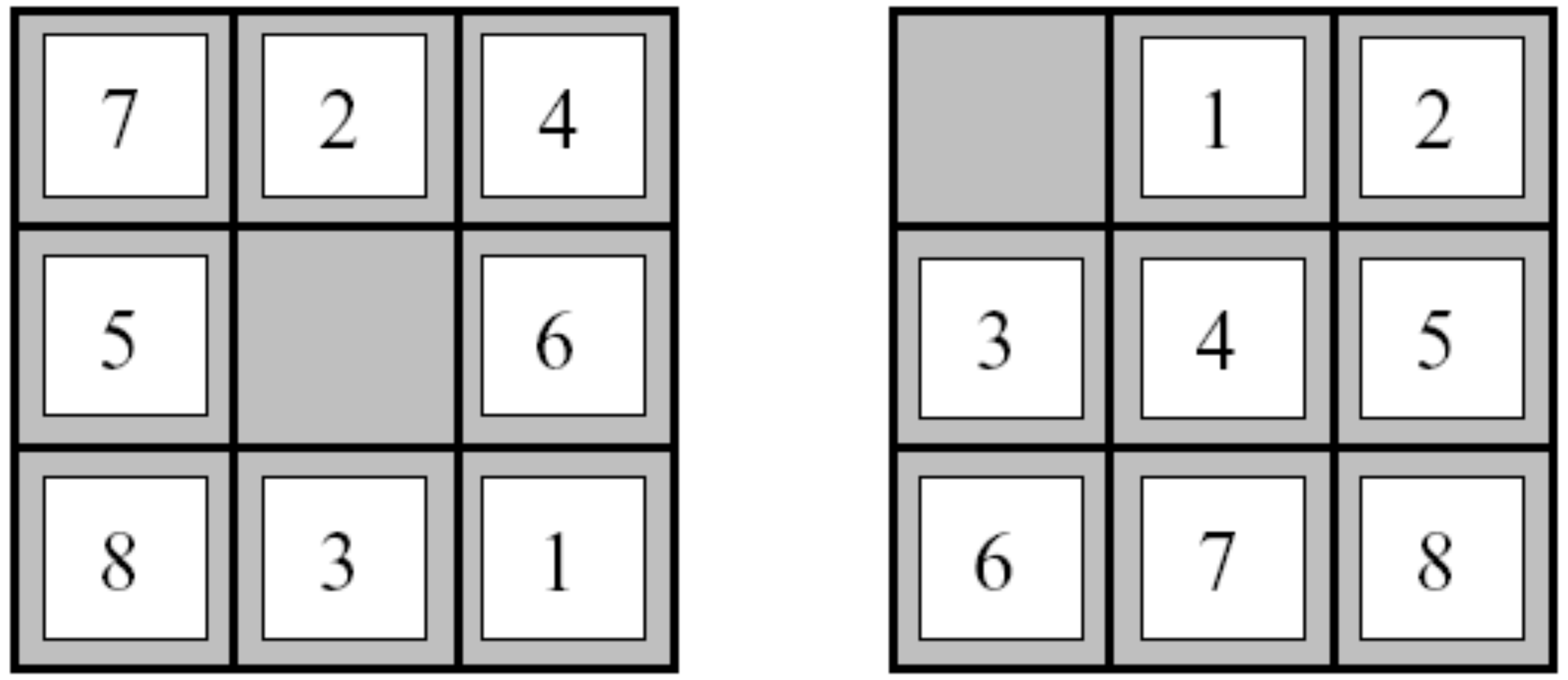


Start State Goal State

- Original problem: tiles cannot overlap (constraint)
- Relaxed problem: tiles can overlap (no constraint)
- Relaxed solution: 8 indep. problems, each in closed form

8 Puzzle I

- Heuristic: Number of tiles misplaced
- What is the relaxed problem?



Start State Goal State

- h(start) = 8

- With a heuristic given by the relaxed problem, the number of nodes expanded decreases significantly

	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

8 Puzzle II

- Heuristic: Manhattan distance
- What is the relaxed problem?

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- h(start) =

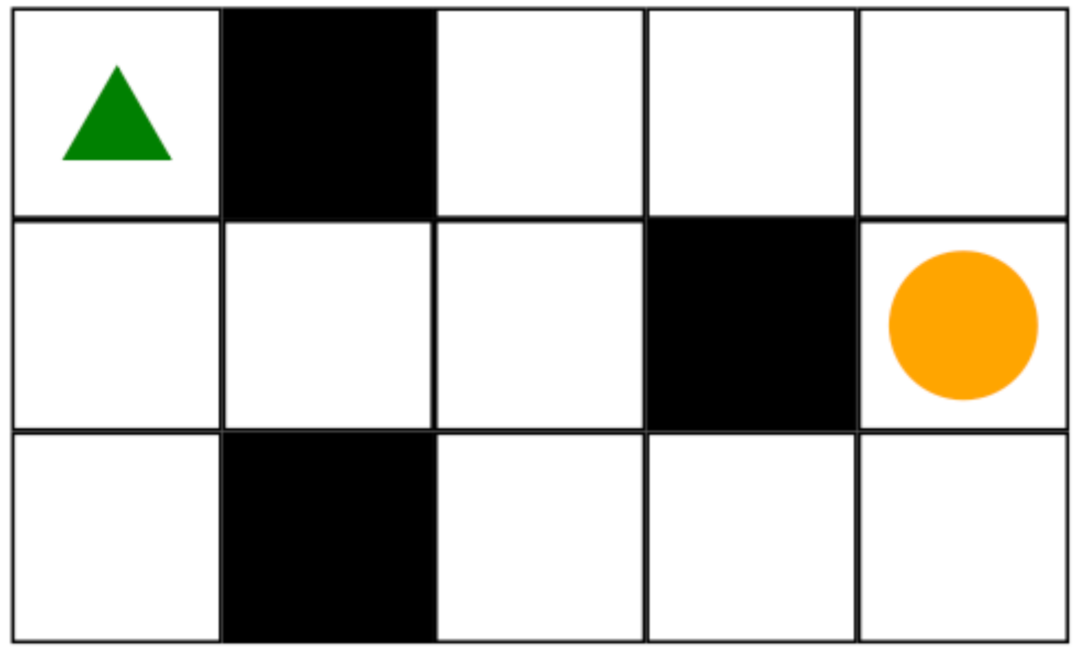
3 + 1 + 2 + ...

= 18

	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

General Framework

- Removing constraints (knock down walls, walk/tram freely, overlap pieces)
- Reducing edge costs (from ∞ to some finite cost)
- Example:



- Original: Cost((1, 1), East) = ∞
- Relaxed: Cost_rel((1,1), East) = 1

General Framework

Definition: relaxed search problem

A **relaxation** P_{rel} of a search problem P has costs that satisfy:

$$\text{Cost}_{\text{rel}}(s, a) \leq \text{Cost}(s, a).$$

Definition: relaxed heuristic

Given a relaxed search problem P_{rel}, define the **relaxed heuristic** $h(s) = \text{FutureCost}_{\text{rel}}(s)$, the minimum cost from s to an end state using $\text{Cost}_{\text{rel}}(s, a)$.

Consistency

Theorem: consistency of relaxed heuristics

Suppose $h(s) = \text{FutureCost}_{\text{rel}}(s)$ for some relaxed problem P_{rel}.

Then $h(s)$ is a consistent heuristic.

- Proof:

$$h(s) \leq \text{Cost}_{\text{rel}}(s, a) + h(\text{Succ}(s, a)) \text{ [triangle inequality]}$$

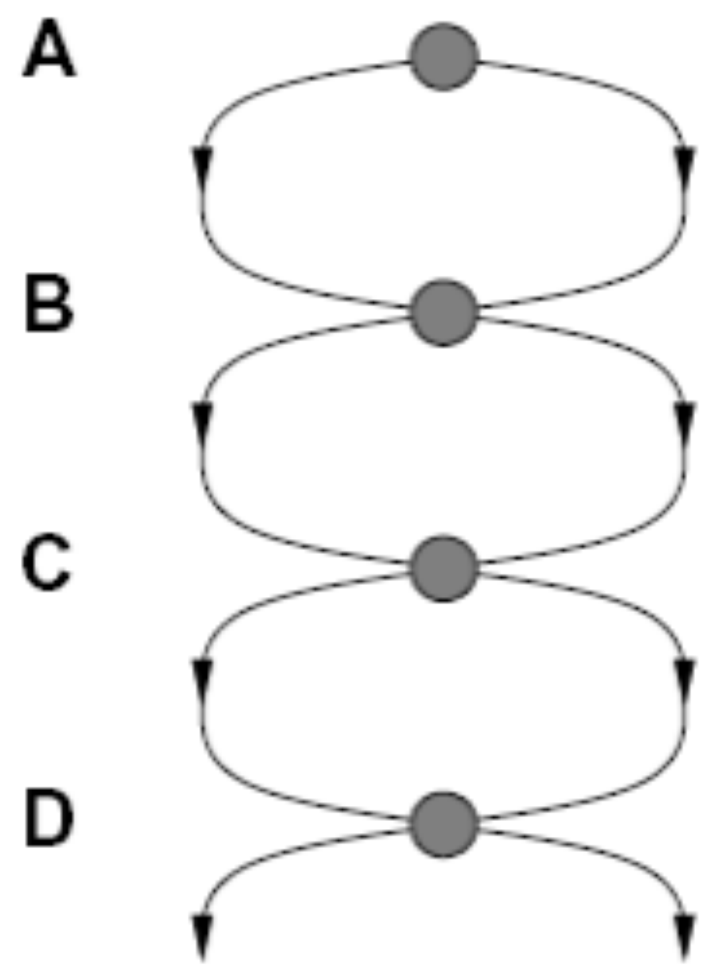
$$\leq \text{Cost}(s, a) + h(\text{Succ}(s, a)) \text{ [relaxation]}$$

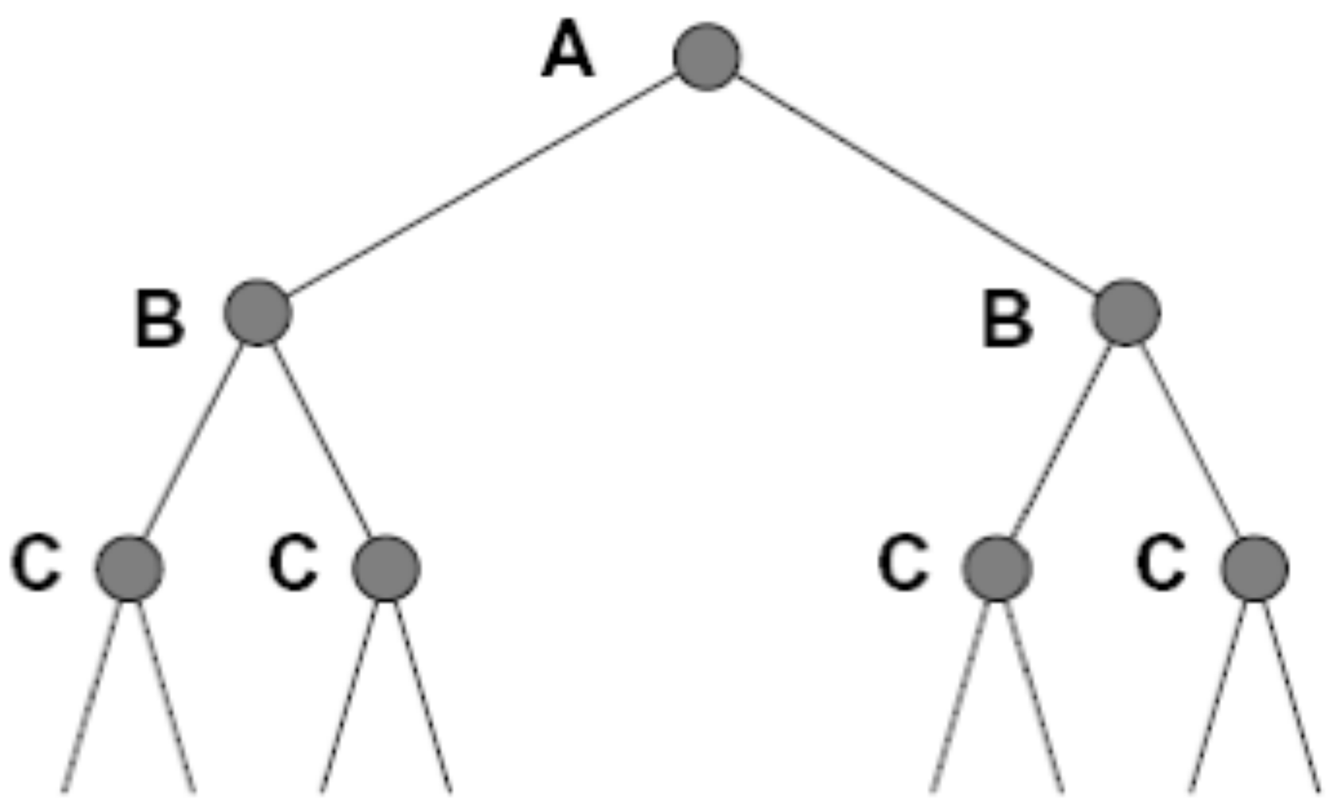
Other A* Applications

- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Tree Search: Extra Work!

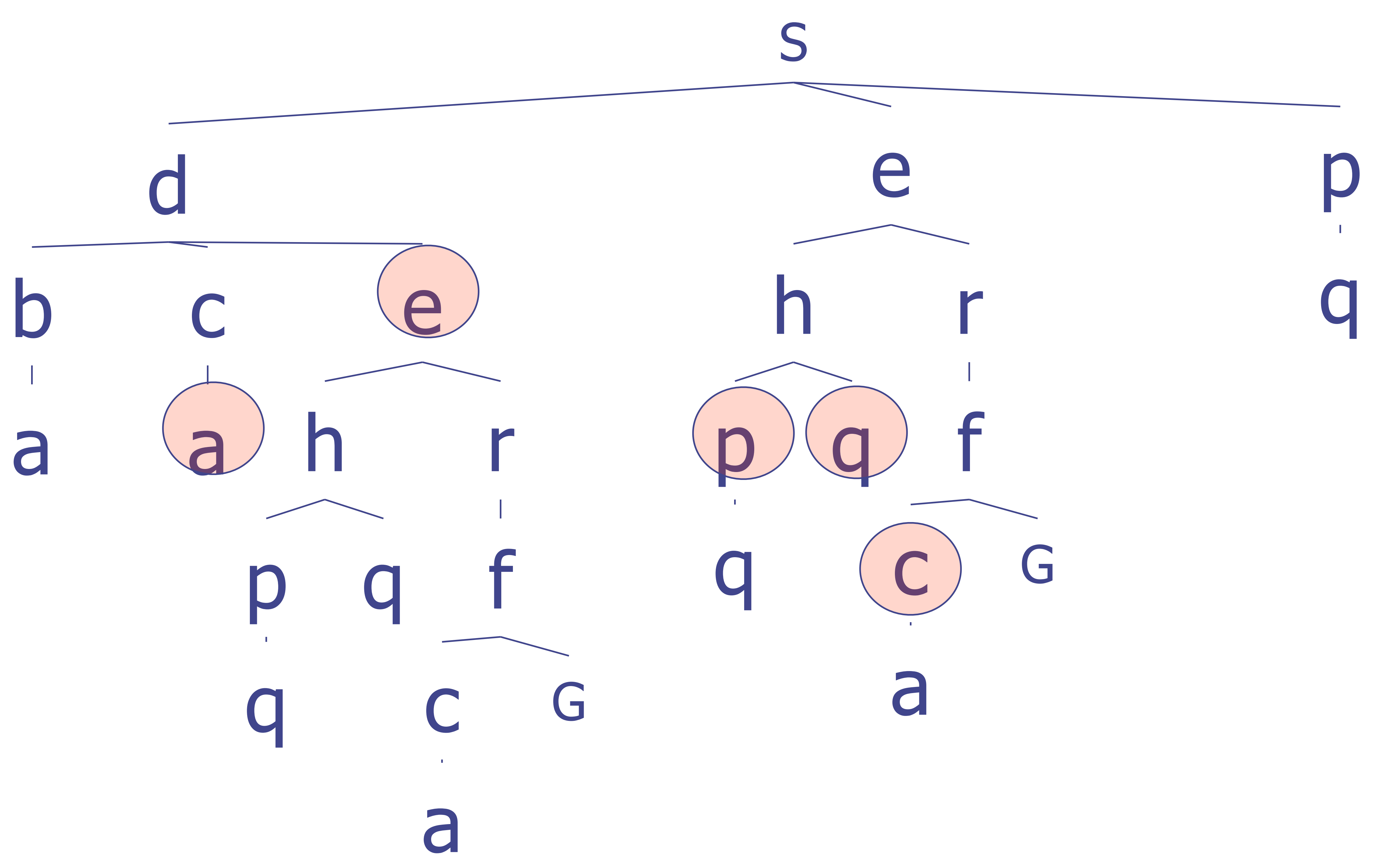
- Failure to detect repeated states can cause exponentially more work. Why?





Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state is new
- Store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

Graph Search

- Very simple fix: never expand a state twice

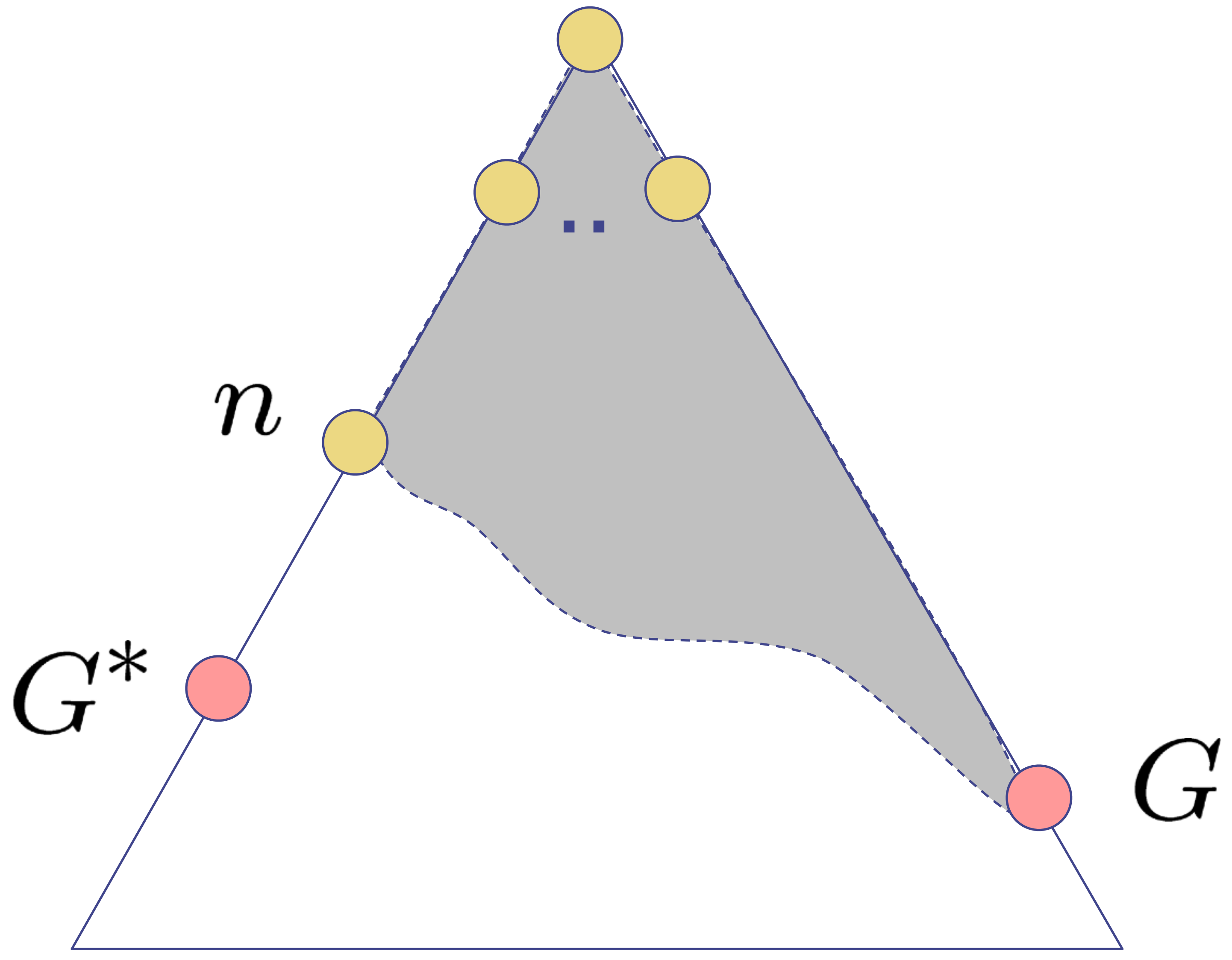
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST(problem, STATE[node]) then return node
      if STATE[node] is not in closed then
          add STATE[node] to closed
          fringe ← INSERTALL(EXPAND(node, problem), fringe)
  end
```

Other things to take into account

- Efficiency
  - h(s) = FutureCost_rel (s) must be easy to compute
  - Closed form, easier search, independent subproblems
- Tightness
  - heuristic h(s) should be close to FutureCost(s)
  - Don’t remove too many constraints

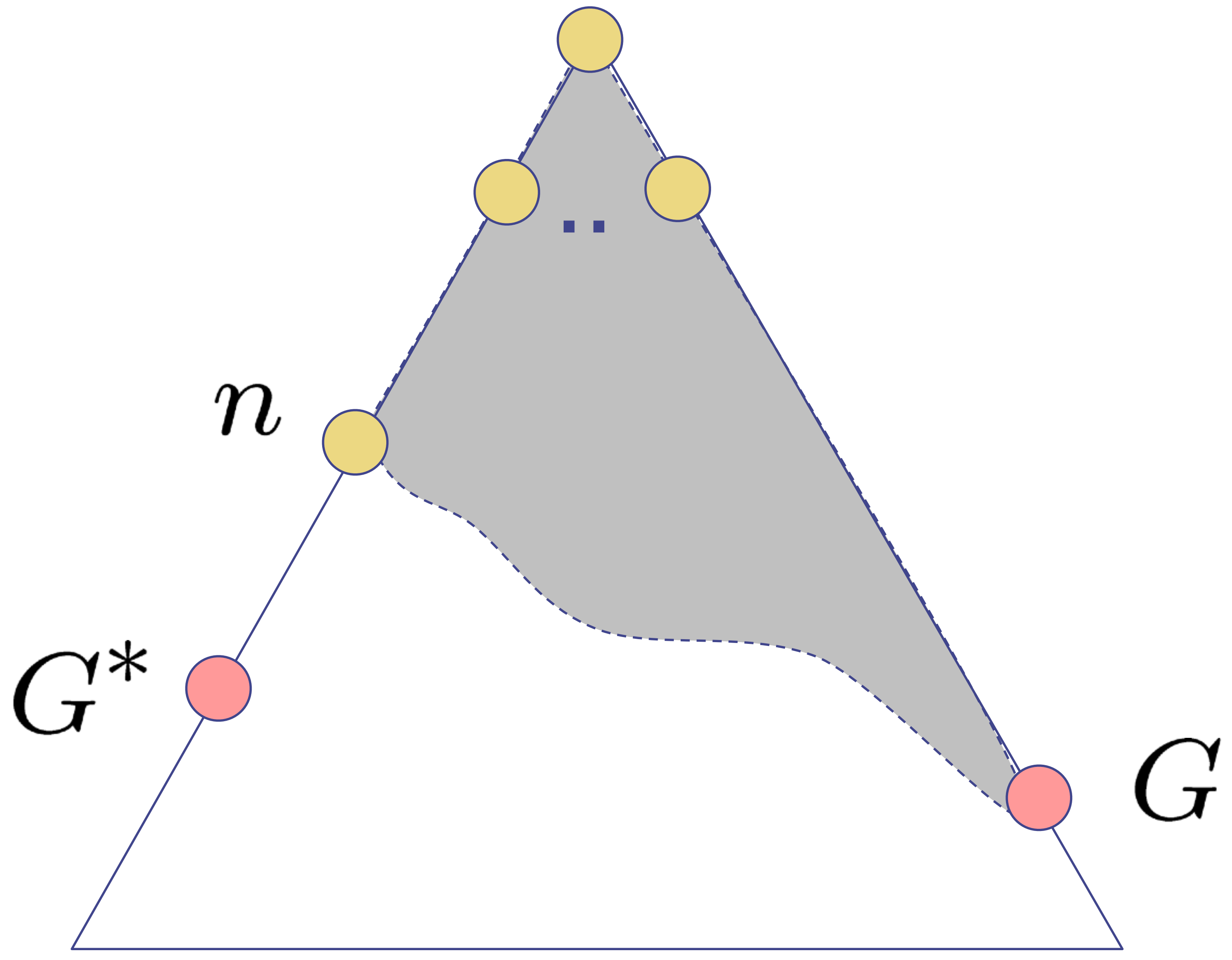
Optimality of A*: Blocking

- Notation:
- g(n) = cost to node n, g*(n) optimal cost to node n
- h(n) = estimated cost from n to the nearest goal (heuristic), h*(n) optimal cost to nearest goal
- f(n) = g(n) + h(n) = estimated total cost via n
- C*: cost lowest cost goal G*
- C: cost of another goal node G, that is not as good, that was returned



Optimality of A*: Blocking

- Proof:
- What could go wrong?
- We'd have to have to pop a suboptimal goal G off the fringe before G*
- This can't happen:
  - Imagine a suboptimal goal G is on the queue
  - Some node n which is a subpath of G* must also be on the fringe
  - n will be popped before G



f(n) > C*, otherwise n would have been expanded
f(n) = g(n) + h(n), by definition
f(n) = g*(n) + h(n), because n is on an optimal path
f(n) <= g*(n) + h*(n), by admissibility, h(n) <= h*(n)
f(n) <= C*, by definition C* = g*(n) + h*(n)

Recap

Week 2 Summary

- Solving problems by searching
  - Informed search strategies
  - Heuristics functions

Next Week

- Search in complex environments
  - Hill climbing, simulated annealing, local beam search, evolutionary algorithm.