15-780: Graduate AI
Lecture 13: Search

Aditi Raghunathan

Search problem

- s_{start}: start state
- Action(s): possible actions at state s
- Cost (s, a): cost of taking action a at state s
- Succ (s, a): state you end up in after you take action a at state s
- IsEnd(s): reached end state?

Recap: BFS vs DFS

- BFS: Root first, then all successors of root node, and then their successors and so on
    - Explore in waves of increasing depth

- DFS: Explore the deepest node first

Recap: BFS vs DFS

- Completeness
  - BFS is complete
  - DFS has issues with cycles

- Time:
  - BFS takes time $\boldsymbol{O(b^d)}$
  - DFS takes time $O(b^m)$

- Memory:
  - BFS requires $O(b^d)$
  - DFS requires $\boldsymbol{O(bm)}$

Can we get best of both?

- Prevent DFS from wandering down an infinite path
- **Depth limited:** fix depth to be l with DFS
  - Time complexity is $\boldsymbol{O(b^l)}$
  - Space complexity is $O(b\ l)$
- **Iterative deepening search:** keep increasing l from 1, 2, … d
  - Time complexity is $\boldsymbol{O(b^d)}$
  - Space complexity is $\boldsymbol{O(bd)}$

Summary

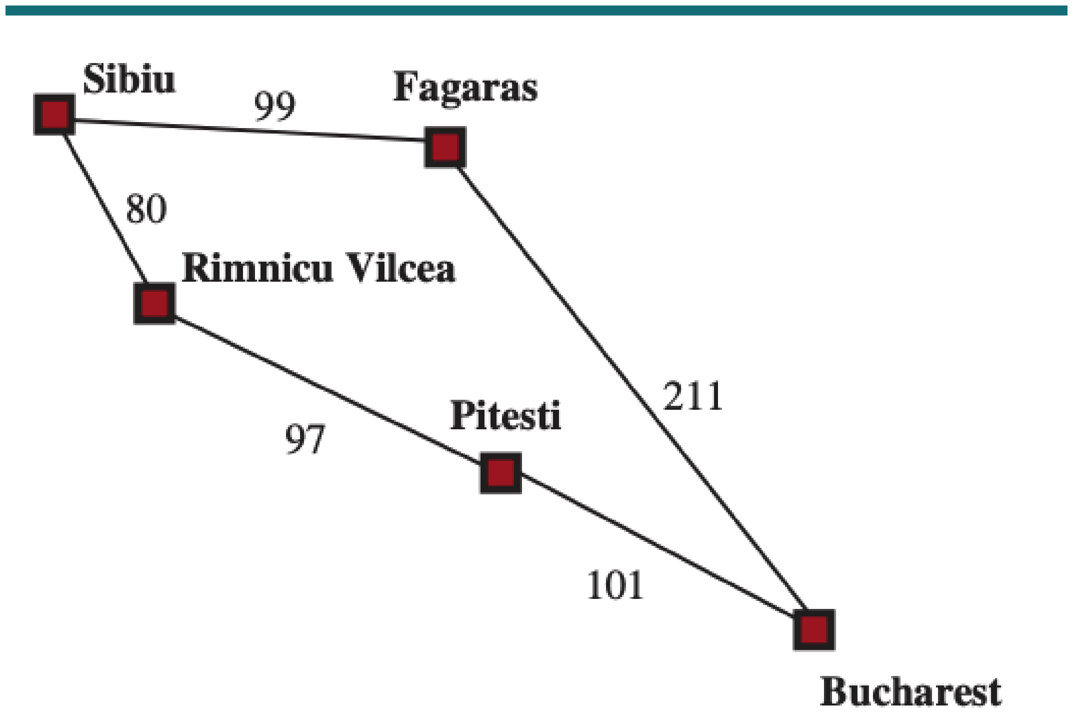
Criterion	Breadth-First	Uniform-Cost	Depth-First	Iterative Deepening
Complete?	Yes[1]	Yes[1,2]	No	Yes[1]
Optimal cost?	Yes[3]	Yes	No	Yes[3]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bd)$

Iterative deepening

- *Best of both*
  - Prevent DFS from wandering down an infinite path
- Depth limited: fix depth to be l with DFS
  - Time complexity is $O(b^l)$
  - Space complexity is $O(b\,l)$
- **Iterative deepening search:** keep increasing l from 1, 2, … d
  - Time complexity is $O(b^d)$
  - Space complexity is $O(bd)$

Djikstra or uniform-cost search

- Expand the node with a shortest path from root



Sibiu to Bucharest

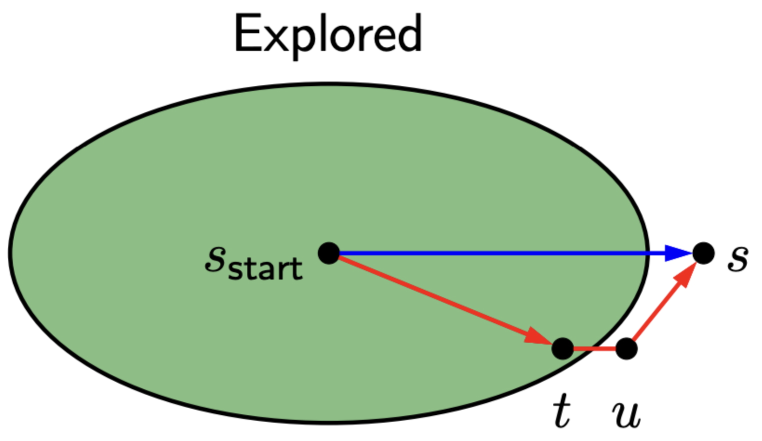
- Sibiu is added to the queue and expanded
- Candidates: R (80) and F(99)
    - Least cost is R
    - R is expanded and added to queue
- Candidates: P (177), F(99)
    - Least cost is F, added to queue and expanded
- Candidates: P(177), **B(310)**
    - Least cost is P, added to queue and expanded
- Candidates: B′(278), B(310)
    - Least cost is B′
    - Goal reached

High-level strategy

- Maintain three sets of nodes
  - Explored, Frontier, and Unexplored
- **Explored:** states we’ve found the optimal path to
- **Frontier:** states we’ve seen (i.e. found a path) but may not be optimal
- **Unexplored:** states we haven’t seen
- We keep moving states from unexplored to frontier, and then frontier to explored
  - Move node with smallest ***cost-from-start-node*** to the explored set
- Stop when **goal state** moves from frontier to explored

Correctness: key invariant

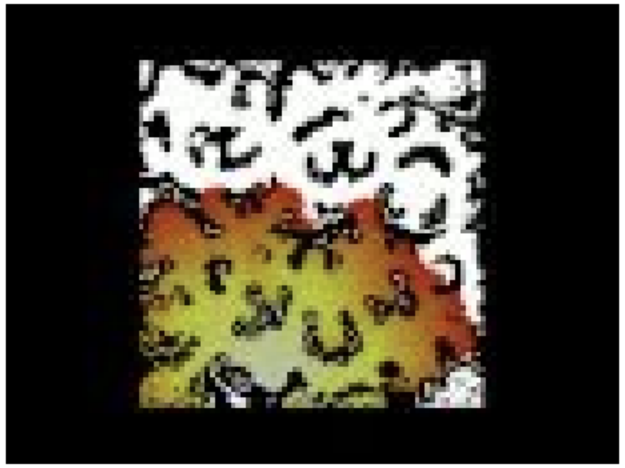
We have computed minimum cost paths from start to all the nodes in the explored set

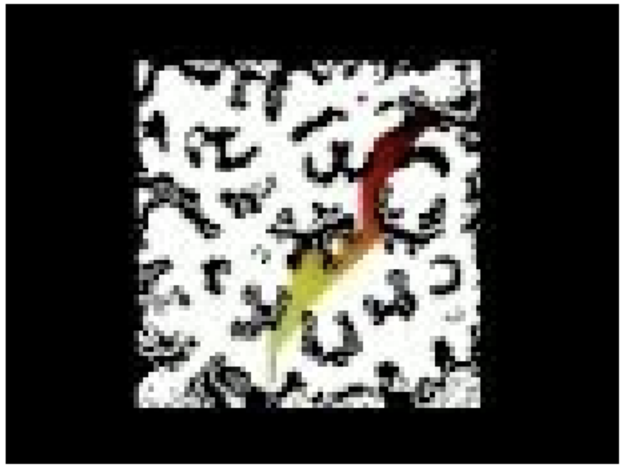


When s is moved from frontier to explored, there cannot exist a shorter path to s, if all costs are non-negative

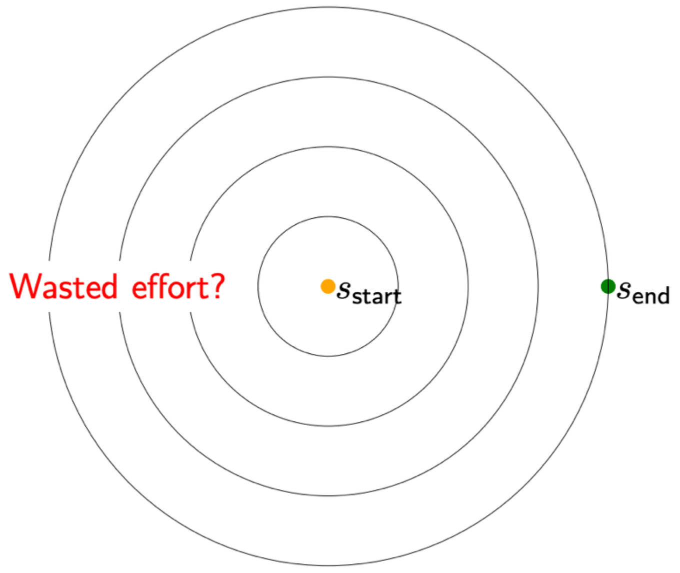
When goal state is explored, by the invariance property, we get correctness!

UCS can be inefficient


UCS in action


A* in action

Why is UCS wasteful?

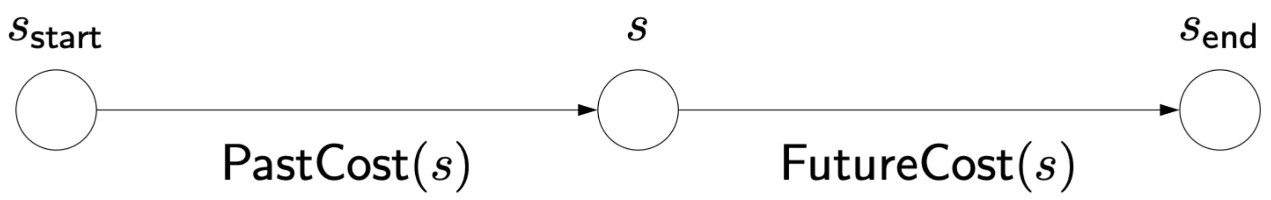


Only takes into account cost from start to state

Does not account for cost from state to end

Can we bias towards states that are closer to the end state?

A* algorithm



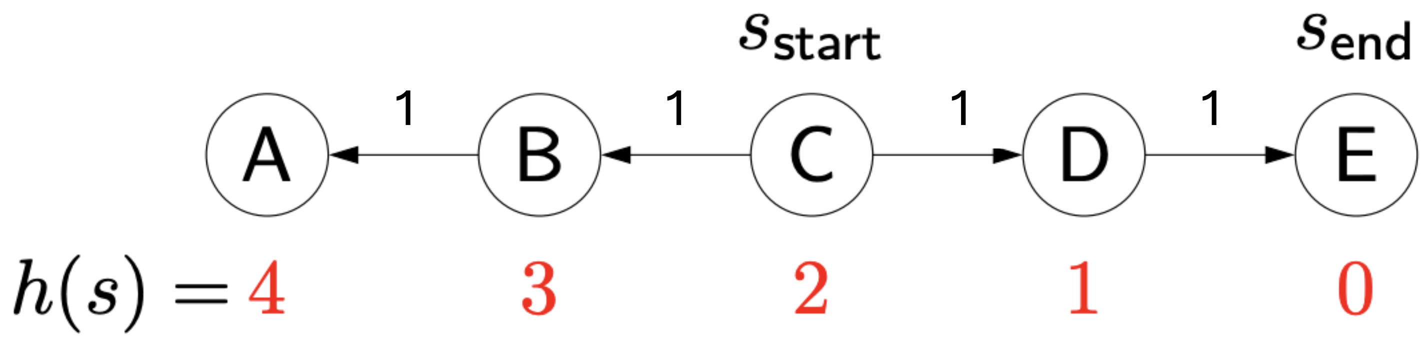
- UCS computes PastCost(s) and orders using it
- We want to use FutureCost(s) but don't know what it is
    - We haven't yet solved how to get from s to end

- **A* uses a heuristic *h*(*s*) as an estimate of FutureCost(s)**

A* algorithm

- Move nodes from frontier to explored based on **PastCost(s) + h(s)**

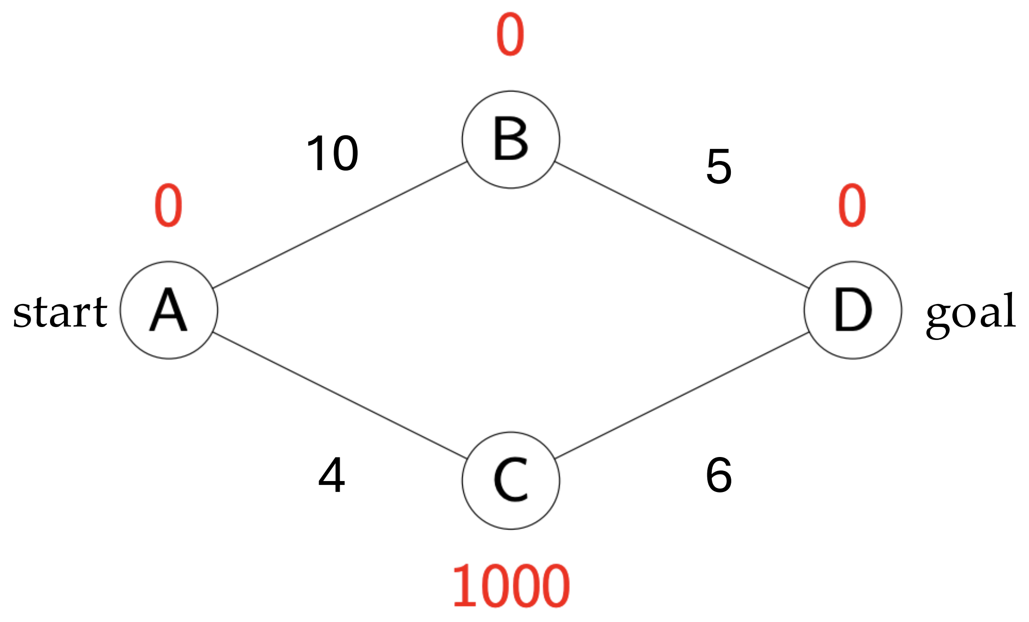
- Assuming h(s) is a good approximation of future costs, we explore states closer to the end-state first



- UCS explores all nodes but A* only explores the nodes on the path to end state

Can any heuristic work?

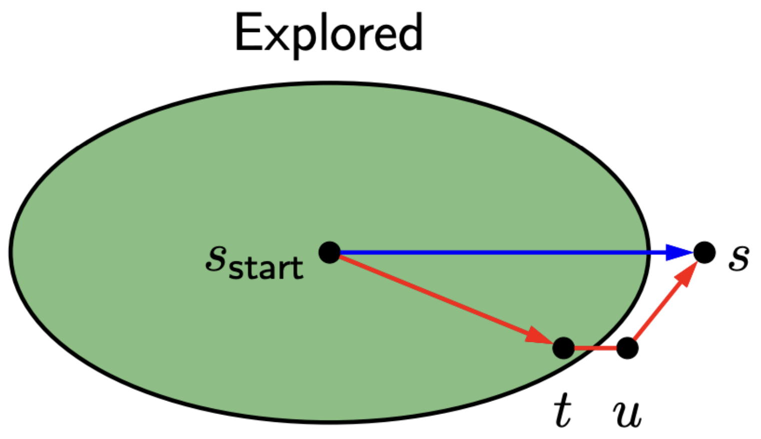
- Try to think of a counter example…



Can any heuristic work?

- We always get completeness but not cost-optimality
- How to get cost-optimality?
- **Intuition:** heuristic should not prevent exploration of a shorter path
- **Admissible heuristic: never overestimates the distance to the goal**
- A* with admissible heuristics is cost-optimal!

Admissibility => cost-optimality



$$h(t) \le \text{FutureCost}(t)$$

$$\text{PastCost(goal)} < h(t) + \text{PastCost}(t)$$
$$\Rightarrow \text{Cost of path 1} < \boldsymbol{h(t)} + \text{PastCost(t)}$$
$$\Rightarrow \text{Cost of path 1} < \textbf{FutureCost}(\boldsymbol{t}) + cost(u, s)$$
$$\Rightarrow \text{Cost of path 1} < \text{Cost of path 2}$$

Poll:

How to best combine two admissible heuristics?

(A) Min (h1, h2) is admissible but not max(h1, h2)
(B) Max (h1, h2) is admissible but not min(h1, h2)
(C) Neither max nor min are admissible
(D) Both admissible, better to use max
(E) Both admissible, better to use min

Consistent heuristics

$$h(s) \leq h\big(\text{Succ}(a, s)\big) + \text{Cost}(s, a)$$

Consistency implies admissibility

Extra credit homework: prove this!

Another perspective on A*

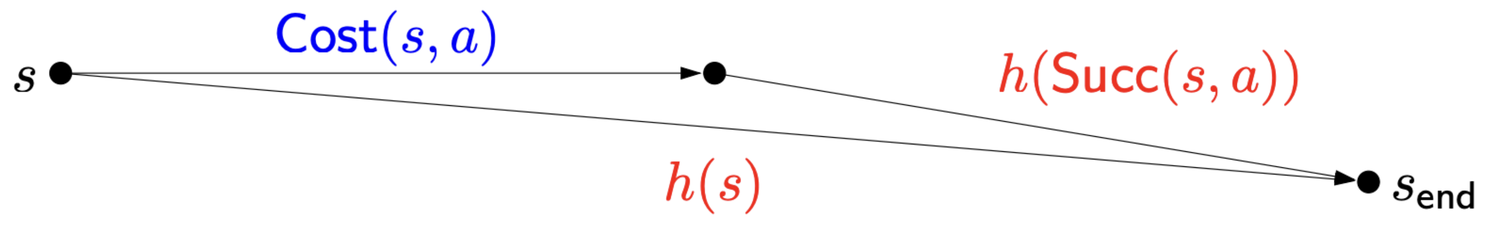
- UCS with modified edge costs

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

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

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

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



How to get admissible heuristics?

- Best heuristic is h(s) = FutureCost (s)
  - As expensive as the original problem itself
- Consistent heuristics are good too!
  - Make sure to never over-estimate the future cost

Key idea: relaxation

Constraints make life hard. Get rid of them.
But this is just for the heuristic!

Relaxations

Definition: relaxed search problem

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

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

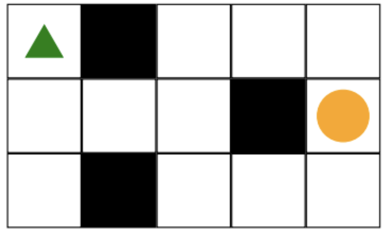
Definition: relaxed heuristic

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

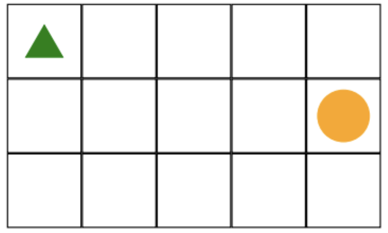
Relaxation: example

Example: knock down walls

Goal: move from triangle to circle



Hard



Easy

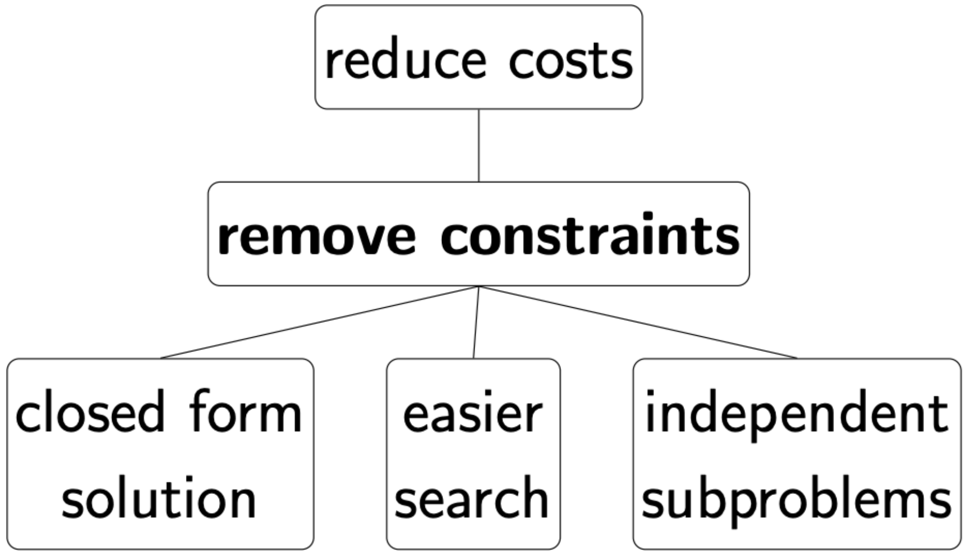
Heuristic:

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

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

Heuristics

Really a **modeling** question



Modeling relaxations

Efficiency:

$h(s) = \text{FutureCost}_{\text{rel}}(s)$ must be easy to compute

Closed form, easier search, independent subproblems

Tightness:

heuristic $h(s)$ should be close to $\text{FutureCost}(s)$

Don't remove too many constraints