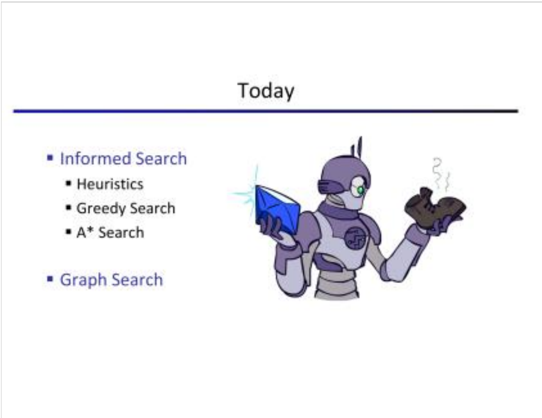
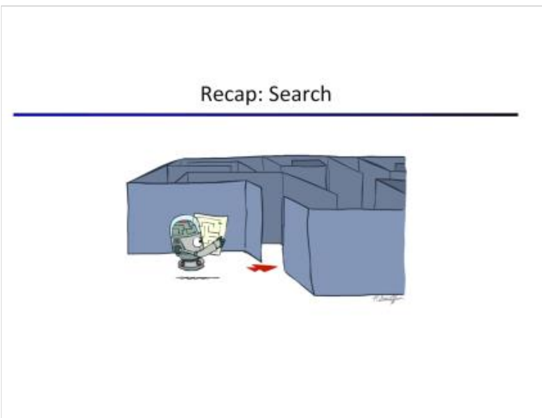
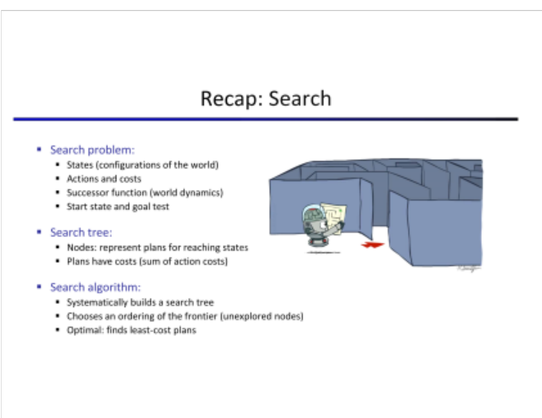
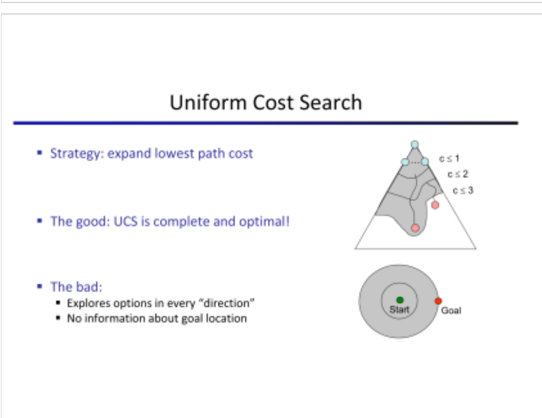
Lecture 3 -- Informed Search

Wednesday, January 26, 2022 3:32 PM









Video of Demo Contours UCS Empty

Video of Demo Contours UCS Pacman Small Maze

Informed Search

What we would like to have happen

Guide search ***towards the goal*** instead of ***all over the place***

Informed

Uninformed

Search Heuristics

- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing

Heuristic:
- a function that estimates how close the current state is to a goal.
- designed for a particular search problem

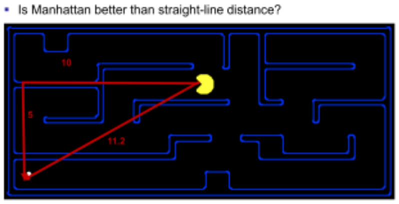
Romanian Heuristic

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

$h(n)$ = straight-line distance to Bucharest

Example: pathing in Pacman

- $h(n)$ = Manhattan distance = $|\Delta x| + |\Delta y|$
- Is Manhattan better than straight-line distance?



Greedy Search

Greedy Search

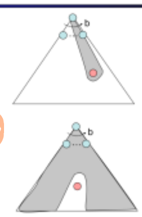
→ everything is based on how good the heuristic is

- Expand the node that seems closest…

- What can go wrong?

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS



→ Sometimes, an idiot, othertimes a servant

– Essentially, DFS (or BFS) with PQ

Video of Demo Contours Greedy (Empty)

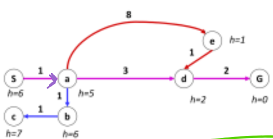
Video of Demo Contours Greedy (Pacman Small Maze)

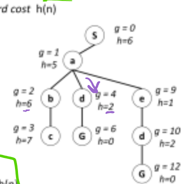
A* Search

* → optimal

Combining UCS and Greedy

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



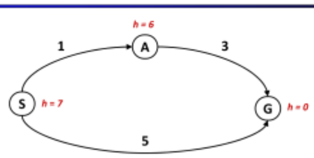


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

backward cost + forward cost

Is A* Optimal?



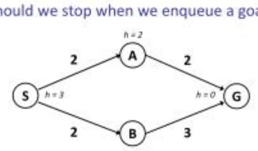
What went wrong?
- ***Actual*** bad solution cost < ***estimated*** good solution cost
- We need estimates to be less than actual costs!

Optimal:
Lowest cost among all solutions

→ Our heuristic sucks!
our estimates need to be less than or equal to actual costs

When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

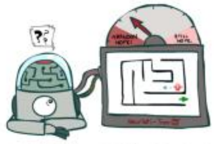
→ expand B

enque, pull next thing off

Stop when you dequeue the goal, not enqueue it

Admissible Heuristics

Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the frontier

Snitty

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Less Shitty

A "bad" heuristic will break optimality by trapping good plans on the frontier.

UNDERESTIMATE

Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

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

- Examples:

- Coming up with admissible heuristics is most of what's involved in using A* in practice.
  - As we will talk about later, these should also be fast to compute.

if my heuristic guess is positive and less than the true cost to the nearest goal. In other words, a heuristic is optimistic if its positive and less than the actual true cost. NOT OKAY W OVERESTIMATING

15 is closer to true cost

Optimality of A* Tree Search

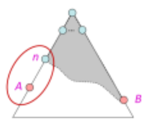
Optimality of A* Tree Search

Assume:
- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:
- A will be chosen for expansion before B
- In other words, A will exit the frontier before B

Optimality of A* Tree Search: Blocking

Proof:
- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (maybe A itself!)
- Claim: n will be expanded before B
  1. $f(n)$ is less than or equal to $f(A)$



$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \le g(A)$	Admissibility of h
$g(A) = f(A)$	$h = 0$ at a goal

backward

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

① $f(n) \le f(A)$

② $f(A) \le f(B)$

③ $\therefore$ n exits before B

Proof:

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

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

$f(A) = g(A)$ ← b.c A is an optimal goal

$g(n) \le g(A) \implies h(n) \le g(A) - g(n)$

actual cost to n — actual cost to n

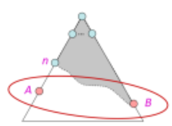
↳ this is the true cost of n to A. $h(n)$ has to UNDER estimate this distance!

$f(n) \le g(n) + g(A) - g(n)$

$f(n) \le g(A)$

Optimality of A* Tree Search: Blocking

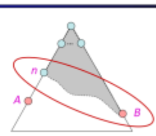
Proof:
- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (maybe A itself!)
- Claim: n will be expanded before B
  1. $f(n)$ is less than or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$



$g(A) < g(B)$	Suboptimality of B
$f(A) < f(B)$	$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:
- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (maybe A itself!)
- Claim: n will be expanded before B
  1. $f(n)$ is less than or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$
  3. n is expanded before B
- All ancestors of A are expanded before B
- A is expanded before B
- **A* tree search is optimal**



$f(n) \le f(A) < f(B)$

backward cost — forward

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

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

$f(A) = g(A) + 0$ ←

$\implies f(A) = g(A)$

$g(A) < g(B)$

$f(A) < f(B)$

b/c h is admissible

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

$\le g(A)$

$\le f(A)$

UCS vs A* Contours

- Uniform-cost expands equally in all "directions"

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

Comparison

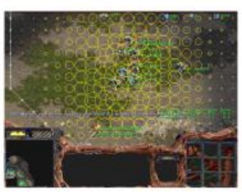
Greedy (h)

Uniform Cost (g)

A* (g+h)

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- Protein design
- Chemical synthesis
- ...



Creating Heuristics

Proove A* optimal

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

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

$f(A) \leq g(A) + 0$

$\Rightarrow f(A) = g(A)$

$g(n) \leq g(A) \Rightarrow h(n) \leq g(A) - g(n)$

↳ under estimate!

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

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

$f(A) = g(A)$

$\therefore f(n) \leq f(A) < f(B)$

$f(A) \leq g(A)$

$f(A) = g(A)$

$g(n) \leq g(A)$

Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in finding admissible heuristics
- Often, admissible heuristics are solutions to ***relaxed problems***, where new actions are available

→ Take some of the "restrictions" and throw them away

→ example: removing walls & manhattan

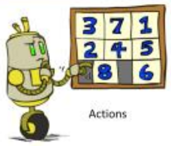
→ relaxed problem = under estimate

- Problem P_2 is a relaxed version of P_1 if $\mathcal{A}_2(s) \supseteq \mathcal{A}_1(s)$ for every s
- Theorem: $h_2^*(s) \leq h_1^*(s)$ for every s, so $h_2^*(s)$ is admissible for P_1
- Inadmissible heuristics are often useful too (will discuss later why!)

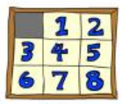
Example: 8 Puzzle

Heuristic → how far am I from the thing

Start State



Actions



Goal State

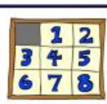
→ relaxed problem: manipulate however you want and make heuristic

- What are the states?
- How many states?
- What are the actions?
- What are the step costs?

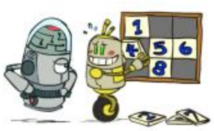
8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8

Start State



Goal State



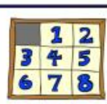
	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
A*TILES	13	39	227

Statistics from Andrew Moore

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18

Start State

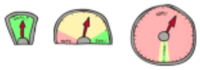


Goal State

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
A*TILES	13	39	227
A*MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?

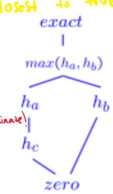


- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Semi-Lattice of Heuristics

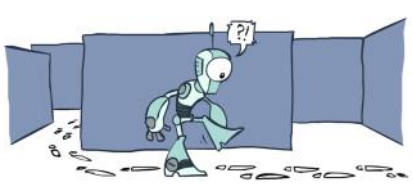
Trivial Heuristics, Dominance

- Dominance: $h_a \geq h_c$ if → b/c its closest to true cost
  $$\forall n : h_a(n) \geq h_c(n)$$
  - Basically, this is a fancy way of saying that if you are bigger, you can assert dominance. Just like in real life.
- What if no one can assert dominance?
  - Create a super strain of dominance. . . is it admissible?
  $$h(n) = max(h_a(n), h_b(n))$$
- Heuristics form a semi-lattice: max so far = dominate!
  - See the picture!
- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



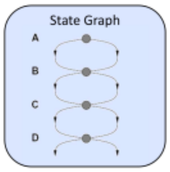
find the silver lining

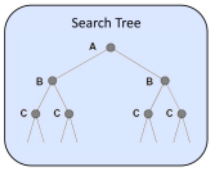
Graph Search



Tree Search: Extra Work!

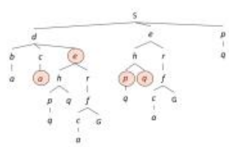
- Failure to detect repeated states can cause exponentially more work.





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 has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list

issue = we have to store every single state

- Can graph search wreck completeness? Why/why not?

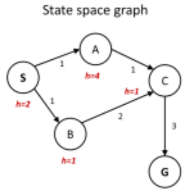
Yes

- How about optimality?

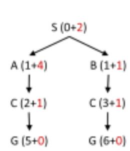
Yes,

A* Graph Search Gone Wrong?

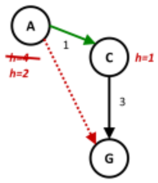
State space graph



Search tree



Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

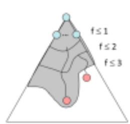
    h(A) – h(C) ≤ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality of A* Graph Search

Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
  - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
  - Result: A* graph search is optimal



Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- Most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary

A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems