Search & AI

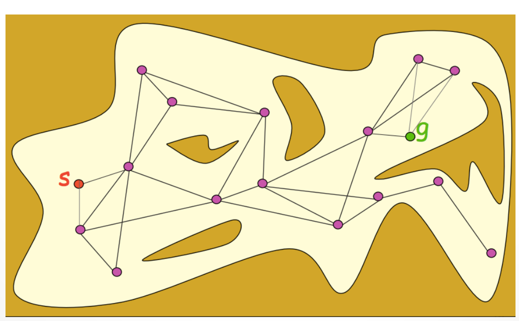
Stephen J. Guy

Many images from *Lavalle, Planning Algorithms* and *Peter Abbeel*

1

Search

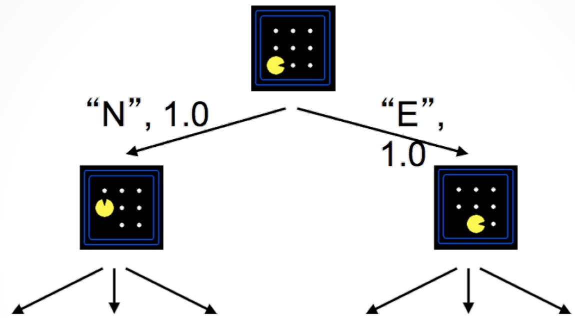
- Very common in AI
  - "AI is search"
    - Game trees, decision trees, puzzle solving
- We'll see the same things in path planning



2

2

Search Trees



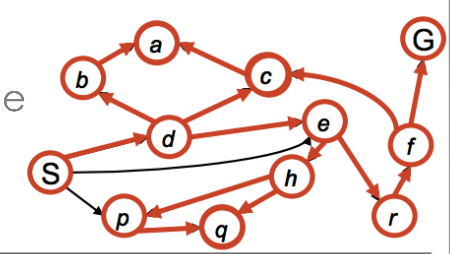
- “What if” analysis of plans and outcomes
- **Root:** Initial state
- **Children:** Successors following an action
- **Node:** Contain states; node’s parents correspond to plans reaching that state
- **Fringe:** Nodes with unexplored action

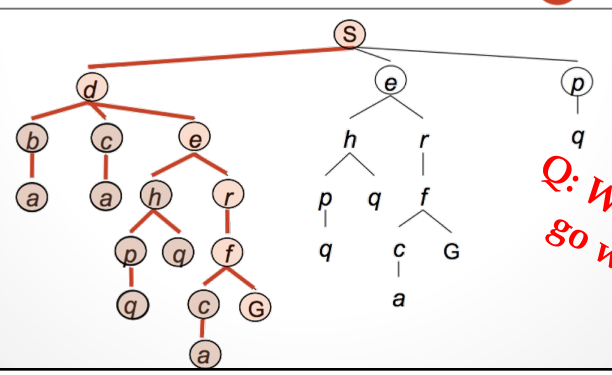
3

3

Depth First Search

- Strategy:
  - o Expand deepest node
- Implementation:
  - o Fringe is LIFO stack





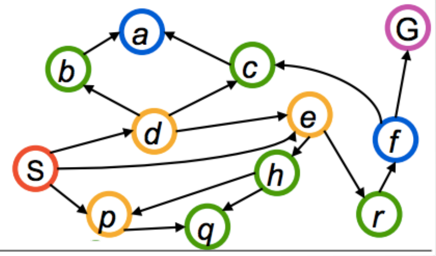
Q: What can go wrong?

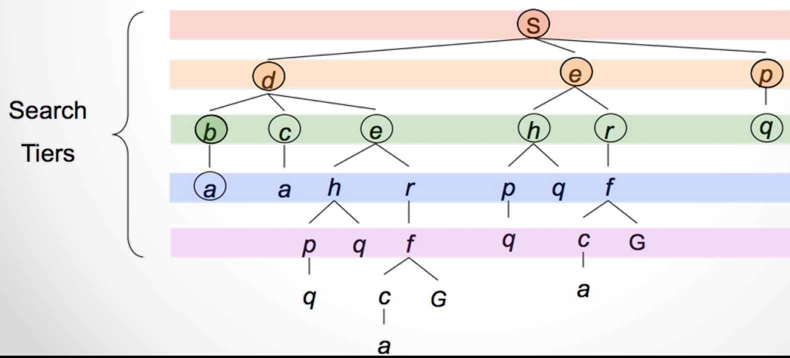
4

4

Breadth First Search

- Strategy:
  - Expand shallowest node
- Implementation:
  - Fringe is FIFO queue





5

5

Breadth-first vs Depth-first

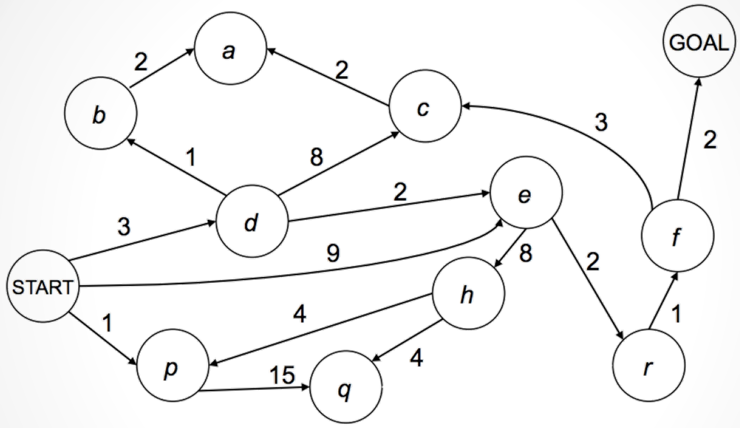
- DFS:
  - Less Memory
  - Will not find optimal solution
  - Gets suck in loops and infinite paths
- Breadth First:
  - Lots of memory required
  - Finds solution with least nodes
  - Handles loops & infinite paths

- Hybrid Options…

6

6

Costs on Actions



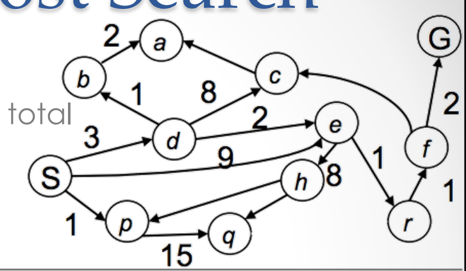
- What if different actions have differing costs
- BFS finds least num. of actions, not least cost path

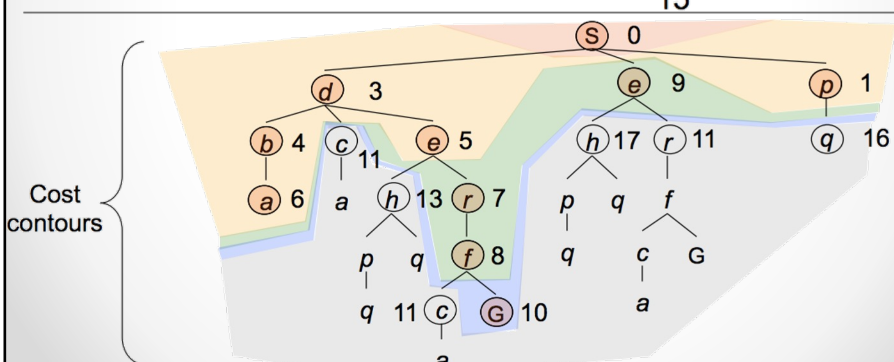
7

7

Uniform Cost Search

- Strategy:
  - Expand node w/ cheapest total
- Implementation:
  - Fringe is a priority queue





8

8

Aside: Priority Queue

- Priority queue allows insert of (key, value) pairs

pq.push(key, value)	inserts *(key, value)* into the queue.
pq.pop()	returns the key with the lowest value, and removes it from the queue.

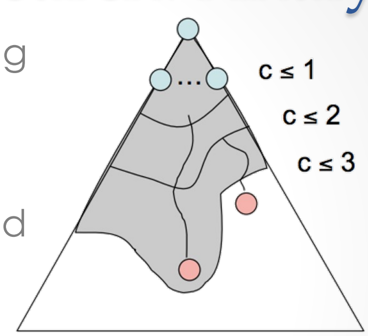
- Insertions normally O(log n)
- Decrease value: pop(), push(key, val-1)
- Priority Queues used in cost-sensitive search methods

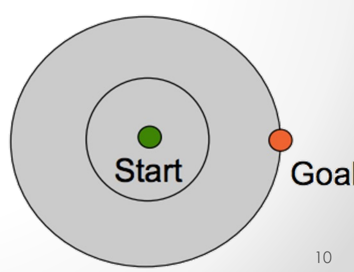
9

9

Uniform Cost Search: Analysis

- UCS: Explore increasing cost contours
- Advantages:
  - UCS is complete and optimal
- Issue:
  - Explores in every "direction" equally
  - Uses no info. about goal



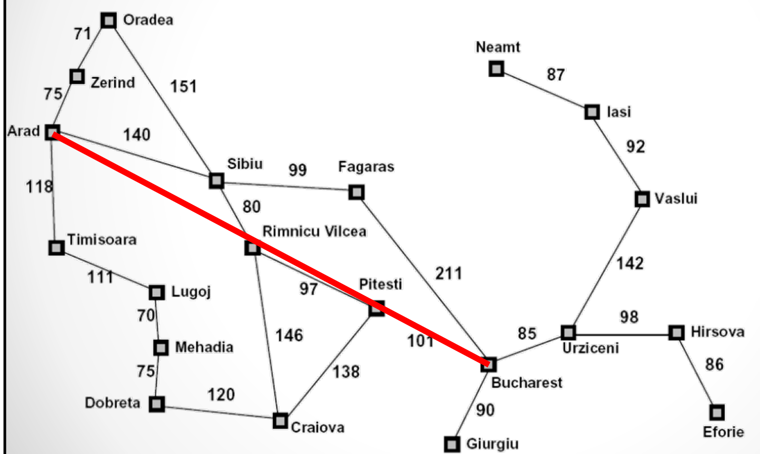


10

10

Heuristics

- Straight-line heuristic



Straight−line distance to Bucharest

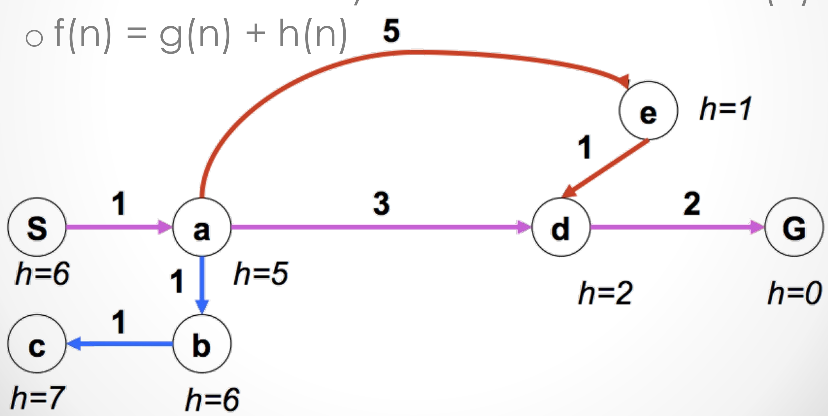
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

11

11

UCS + Heuristic

- Uniform-cost orders by path-cost so far
  - Cost-to-node: $g(n)$
- A* Search orders by total estimated cost $f(n)$
  - $f(n) = g(n) + h(n)$

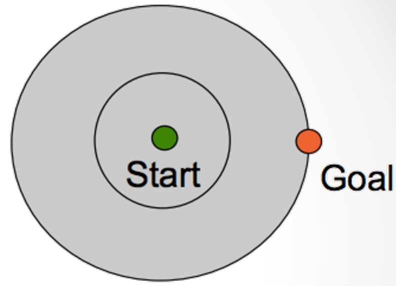


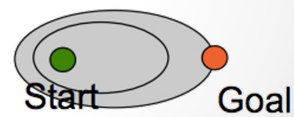
12

12

UCS vs A* Contours

- Uniform cost expands in all directions
- A* expands mainly towards the goal
  - Hedges by expanding outward as needed



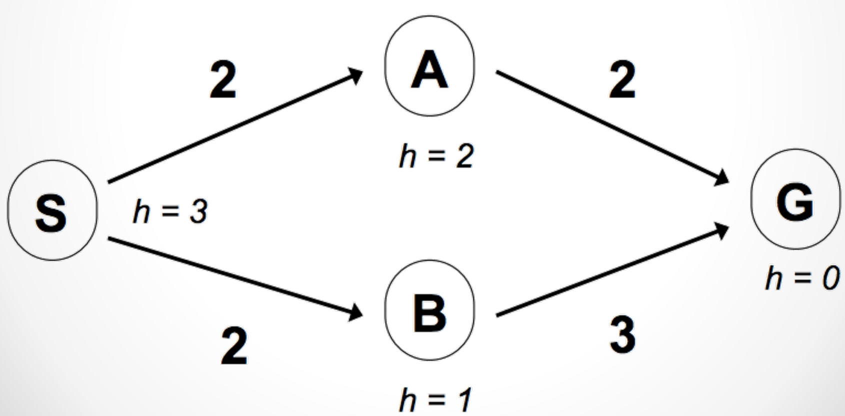


13

13

Terminating A*

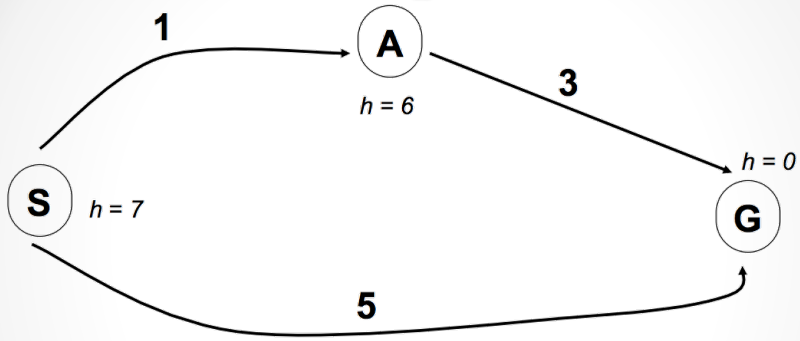
- Can we stop when we push the goal onto the PQ?
- No – Terminate on dequeueing a goal



14

14

Is A* Optimal?



- What went wrong?
  - Actual cost to goal < estimate coast to goal
- Heuristic must be less than the actual cost!
  - A* assumes it will take at least h() to get to goal

15

15

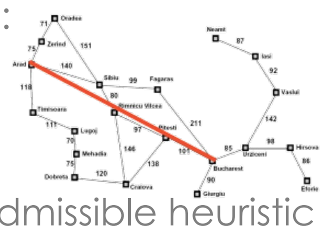
Admissible Heuristic

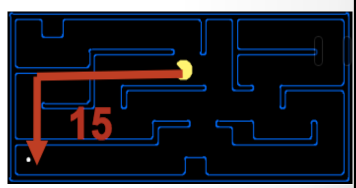
- A heuristic h is admissible if

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

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

- Examples:





- Finding admissible heuristic is main part of using A* in practice

16

16

Optimality

- Terminating A*
  - A* is done when the goal is the lowest thing on the priority queue.

- A* finds optimal path if heuristic is admissible!

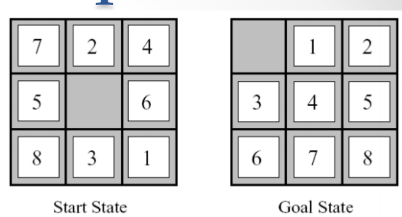
17

17

Classic Example

- 8 Puzzle
- Empty space can slide up, down, left, or right

- Ideas for heuristics?
  - Num displace tiles
  - Total Manhattan dist.

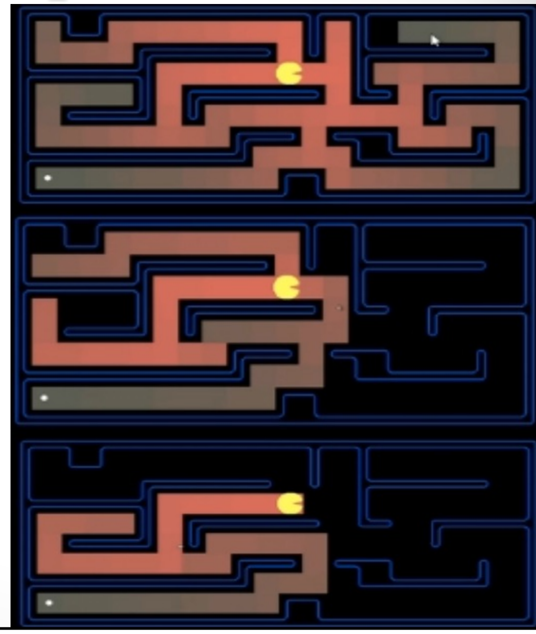


	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227
MANHATTAN	12	25	73

18

Comparison

- UCS
- A*
- Greedy
  - o *Inadmissible*



19

Weighed A* f = g+εh

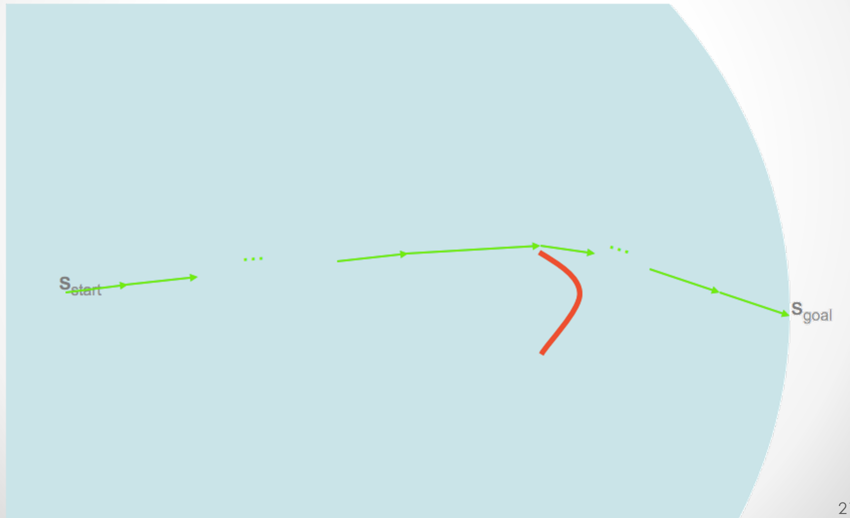
- Weighted A*
  - o Expands nodes in order of $f(n) = g(n) + \varepsilon h(n)$
  - o $\varepsilon > 1$
    - Bias towards goal nodes

20

20

Weighed A*: ε = 0

- ε = 0 → Uniform cost search

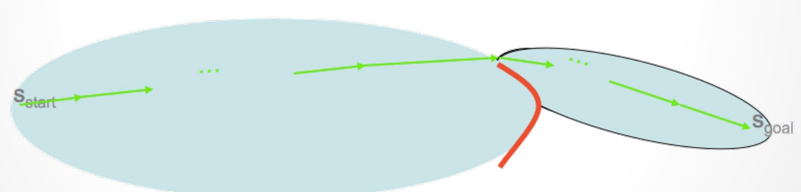


21

21

Weighed A*: ε = 1

- ε = 1 → A*

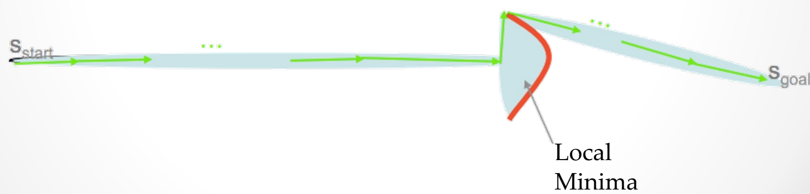


22

22

Weighed A*: ε > 1

- ε > 1 → Targeted path towards goal



23

Anytime Algorithms

24

Weighted A* $f = g+\varepsilon h : \varepsilon > 1$

- Trades off optimality for speed
- ε-suboptimal:
  - cost(solution) ≤ ε* cost(optimal solution)
  - Test your understanding by trying to prove this!
- In many domains, it has been shown to be orders of magnitude faster than A* !
- Research challenge is to find heuristics with shallow local minima

25

25

Anytime Algorithm

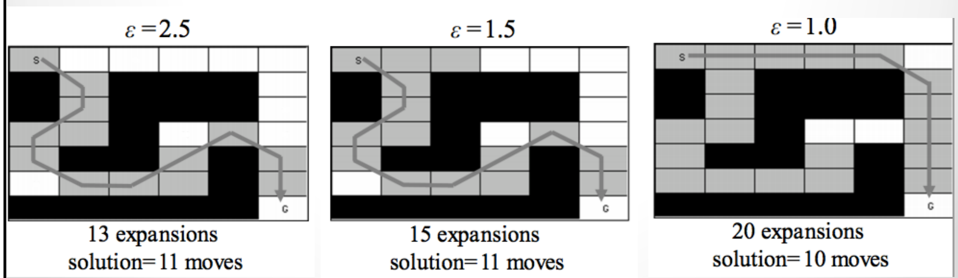
- Algorithm that returns valid answer when interrupted mid run
- Use in robotics
  - Robot can make best current decision as it moves given current information
- Use in games/VR
  - NPCs can make the best decision given time budget for AI

26

26

Anytime A*

- Weighted A*
  - Trades off speed for optimality
  - ε-suboptimal
- Anytime A*
  - Run weighted A* large ε
  - Rerun with decreasing ε to refine path



27

A* Variants

- **Anytime Repairing A* (ARA*)** [Likhachev, Gordon, Thrun 2004]
  - Efficient version of Anytime A* that reuses state values within each iteration
- **Anytime Nonparametric A* (ANA*)** [van den Berg, Shah, Huang, Goldberg, 2011]
  - Provides the "right" next ε to ARA*
- **Lifelong Planning A* (LPA*)**
  - Modified A* to allow updates in weights (e.g., passage becomes blocked)

28

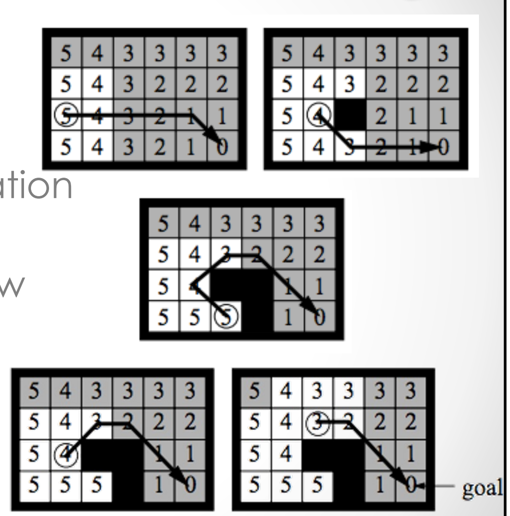
28

D*: A*, but Goal to Start

- A* with consistent heuristic finds shortest path from goal to start
- Flip start and goal
  - Shortest path from any point to start
- Policy for any point in space
  - Can account for noise/errors in motion

D* Lite – Online Planning

- Combine
  - Lifelong Planning A*
  - Flip Start and Goal
  - Some other optimization
- Result
  - Can account for new observations as updated weights on edges
    - e.g. blocked paths



D* in practice

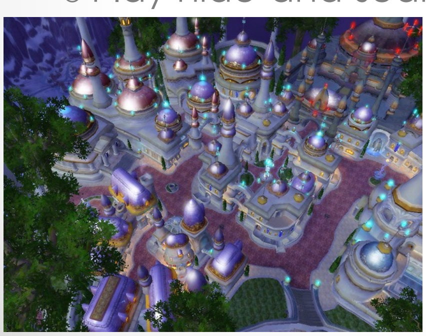
- **Field D*** implemented on Mars rovers Spirit and Opportunity

- **Anytime D*** used in DARA urban challenge winner

31

Research/Project Ideas

- Can you use LPA*/D* ideas to create NPCs which explore games worlds intelligently
  - Allow dynamic environments
  - NPCs in “unknown” environments
  - Play hide-and-seak



32

32

Terms

- Search Tree (Root, Node, Child, Fringe)
- Depth First Search (DFS)
- Breadth First Search (BFS)
- Heuristics
- A*
- Admissibility
- Anytime Algorithms
- Policies

33

33