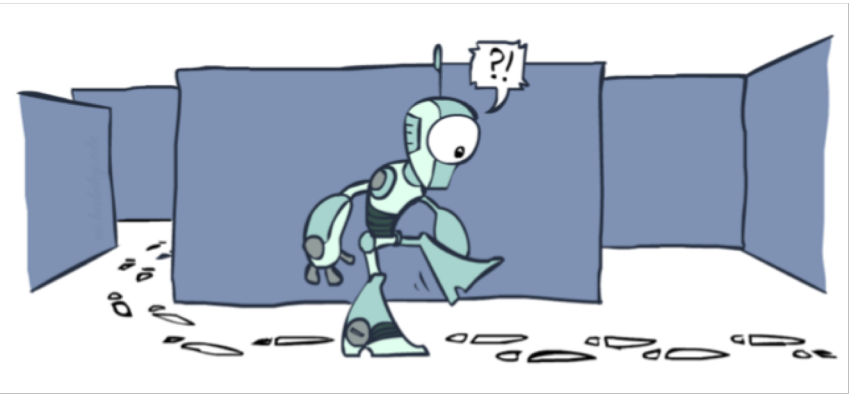
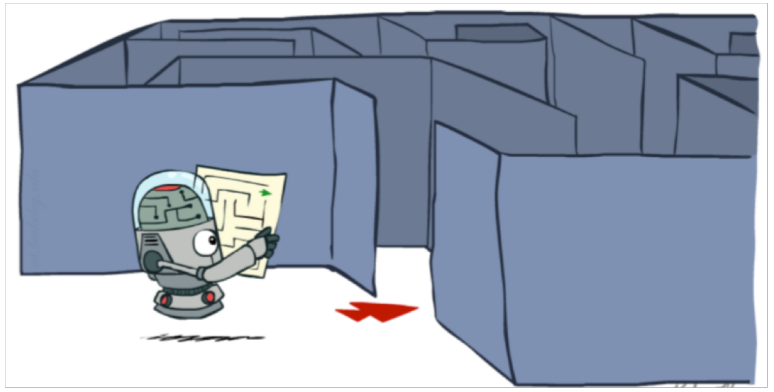
CS 1501: Intro to Robotics

Autonomy, AI, and Applications

Motion Planning III

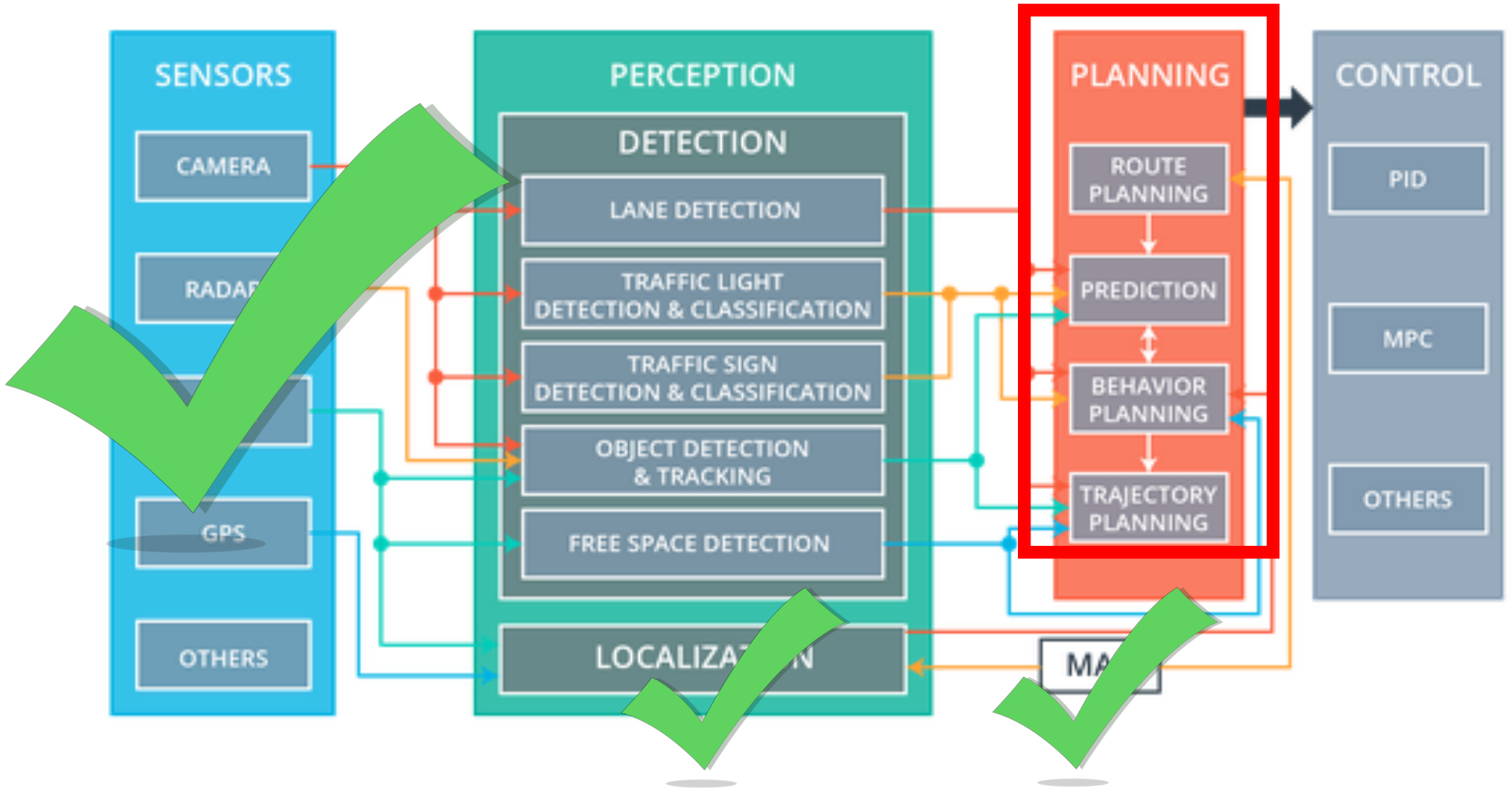




Rohan Raval

Monday 1-1:50pm, MEC 213

Roadmap: See-Think-Act



Roadmap: Today's Lecture...

Goal: *Real-world* approaches to solve Motion Planning

Topics:

- What's the problem with Dijkstra's?
- Heuristics
- Greedy Heuristic Search
- **A* Search**
  - What makes a good heuristic?
- Comparison of Graph Search Algorithms
- Real-world Applications
- **Artificial Potential Fields**
  - Gradient Descent and Local Minima

Recap: Dijkstra's Algorithm

An implementation of Breadth-first Search (BFS)

- **If all weights are equal, then Dijkstra's == Grassfire == BFS**

Pseudocode:

- Initialize each vertex's distance as infinity
- Start at a given vertex *s*
  - Update *s*'s distance to be 0
- Repeat
  - Pick the next unknown vertex with the **shortest distance** to be the next *v*
    - If no more vertices are unknown, terminate loop
- Mark *v* as known
- For each edge from *v* to adjacent unknown vertices *w*
  - If the total distance to *w* is less than the current distance to *w*
    - **Update** *w*'s distance and the path to *w*

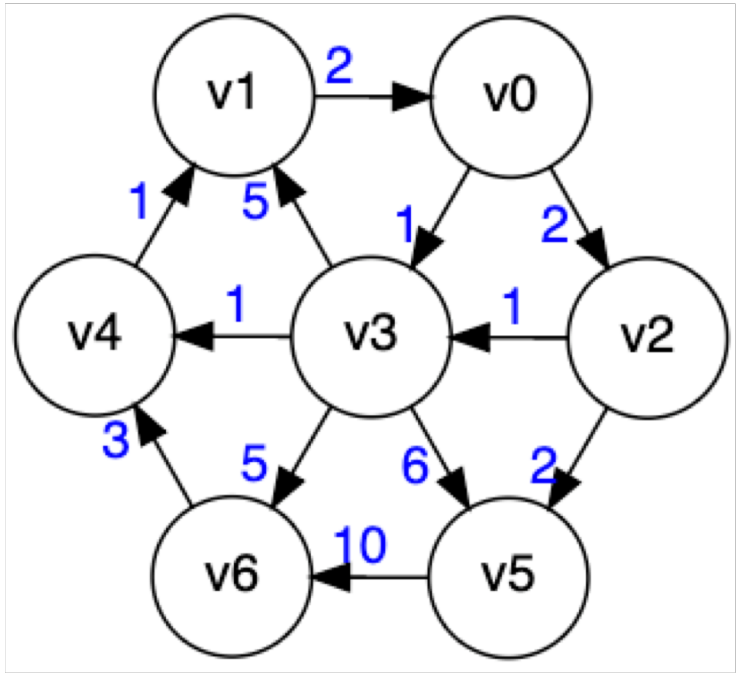
Code adopted from UVA CS 2150 F16

Recap: Dijkstra's Algorithm

Try it yourself!

Example 1: want to get from v0 to v6…

Best path: v0 → v3 → v6



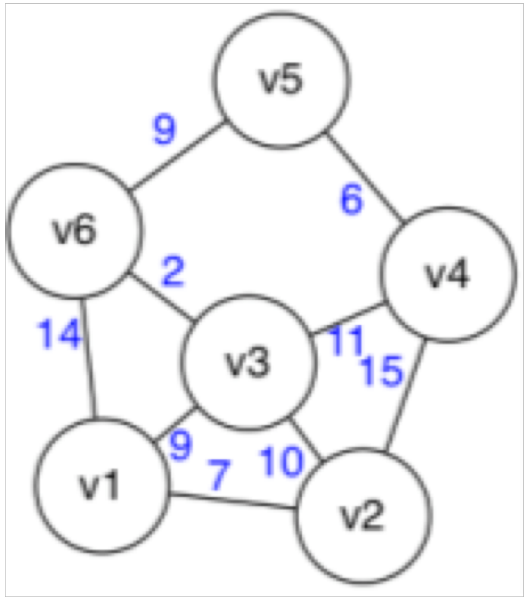
V	Known?	Distance	Path
v0	✓	0	
v1	✓	~~6~~ 3	~~v3~~ v4
v2	✓	2	v0
v3	✓	1	v0
v4	✓	2	v3
v5	✓	~~7~~ 4	~~v3~~ v2
v6	✓	6	v3

Recap: Dijkstra's Algorithm

Try it yourself!

Example 2: want to get from v1 to v5…

Best path: v1 → v3 → v6 → v5

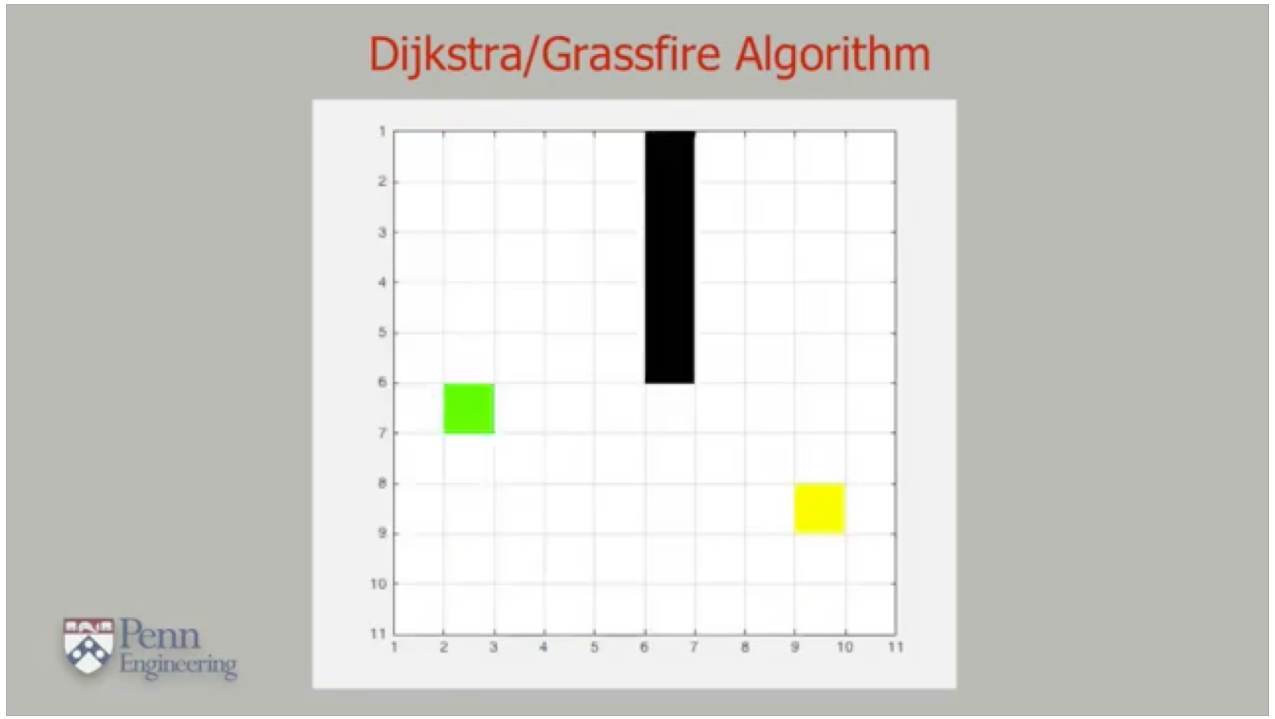


V	Known?	Distance	Path
v1	✓	0	
v2	✓	7	v1
v3	✓	9	v1
v4		~~21~~ 20	~~v2~~ v3
v5	✓	20	v6
v6	✓	~~14~~ 11	~~v1~~ v3

Dijkstra's Algorithm

It **guarantees** to find us the **shortest path!!** (if one exists)

So... what's the problem??

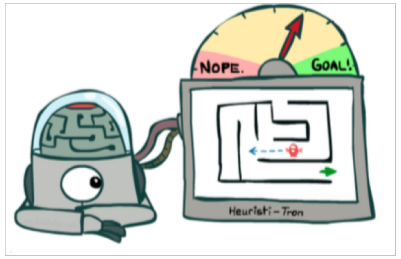


Video adopted from UPenn "Robotics: Computational Motion Planning"

Search Heuristics

Problem with Dijkstra's:

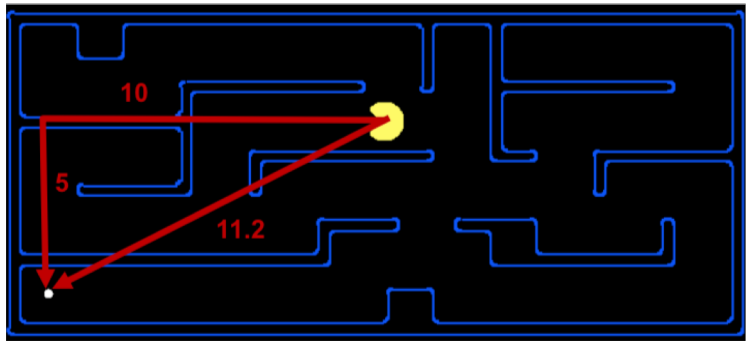
- Even though optimal path is guaranteed in Dijkstra's...
- We could waste lots of time in areas that take us further from the goal



What if we had a "guide" that nudged us in the right direction?

Heuristic:

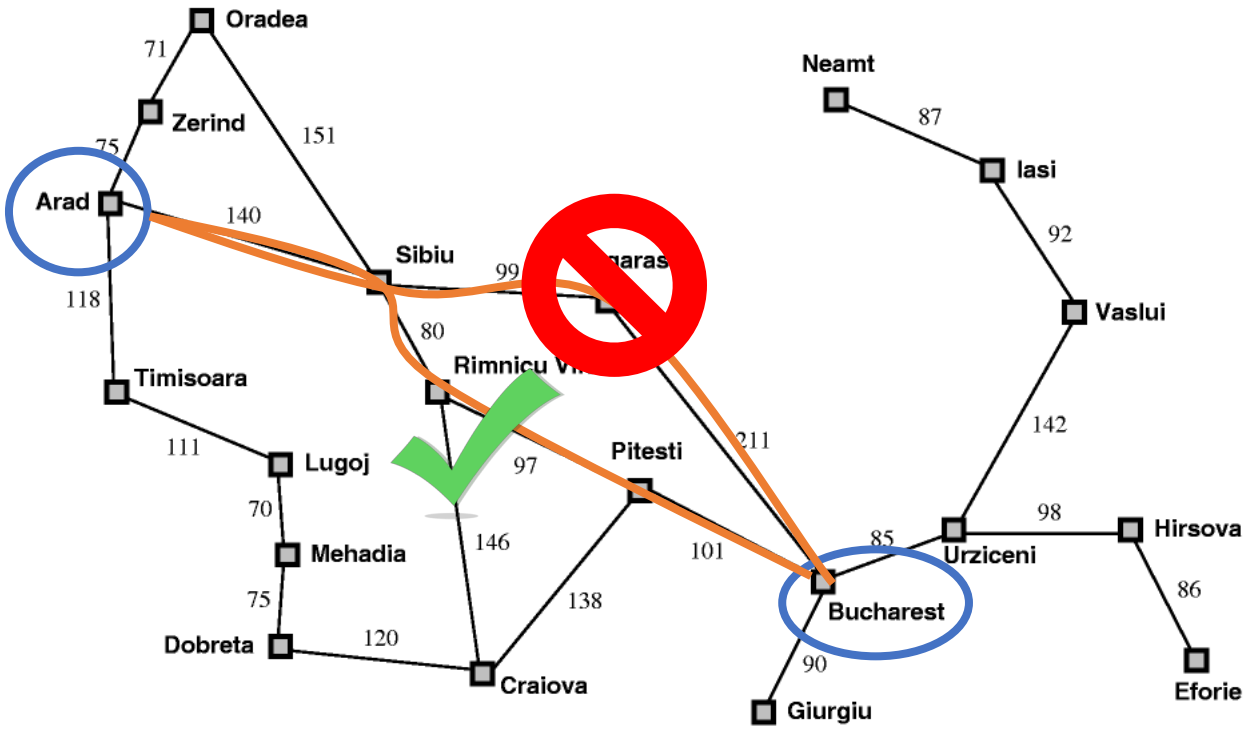
- Function that *estimates* how close a node is to goal
- Designed for particular search problem
- Examples: Euclidean Distance, Manhattan Distance



Slide adopted from UC Berkeley CS188 F13

Greedy Heuristic Search

Only use the heuristic $h(n)$ to help guide us in the "right" direction



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

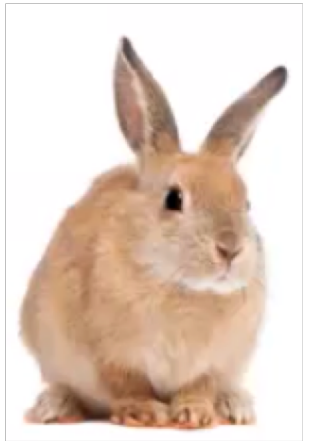
Heuristic $h(n)$

← OPTIMAL

Problem: It could give us the **sub-optimal** path!

A* Search

Dijkstra (Uniform Cost Search)



Greedy Heuristic Search

A* search

Slide adopted from UC Berkeley CS188 F13

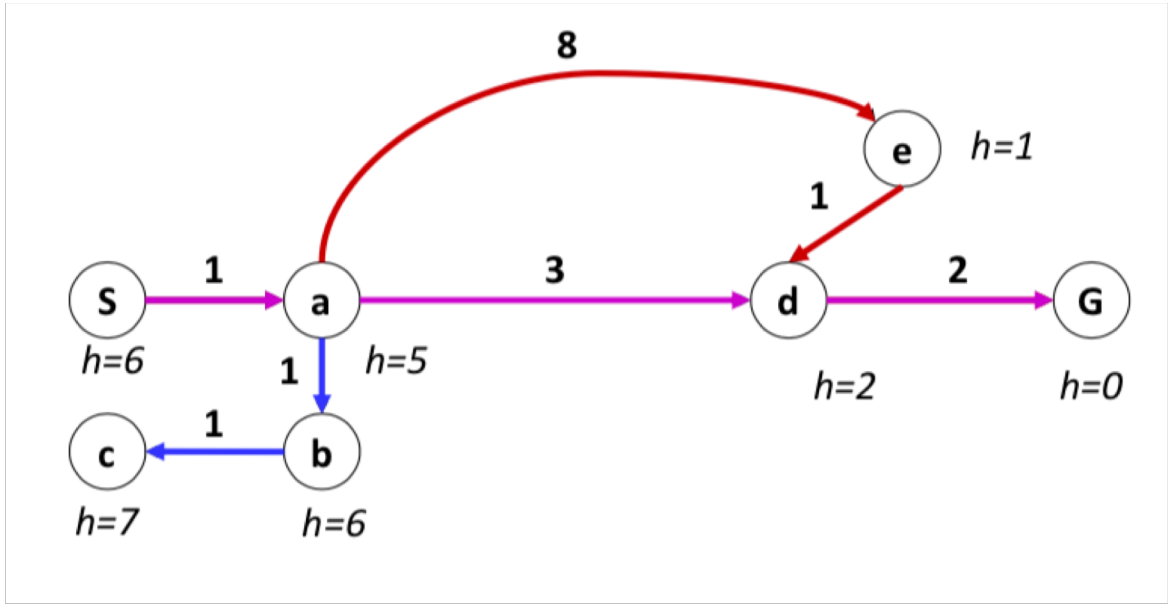
A* Search

Dijkstra's orders by path cost: $g(n)$
Greedy Heuristic orders by goal proximity: $h(n)$
A* orders by the sum $\boldsymbol{f(n) = g(n) + h(n)}$

Slide adopted from UC Berkeley CS188 F13

A* Search

Dijkstra's orders by path cost: *g(n)*
Greedy Heuristic orders by goal proximity: *h(n)*
A* orders by the sum ***f(n) = g(n) + h(n)***



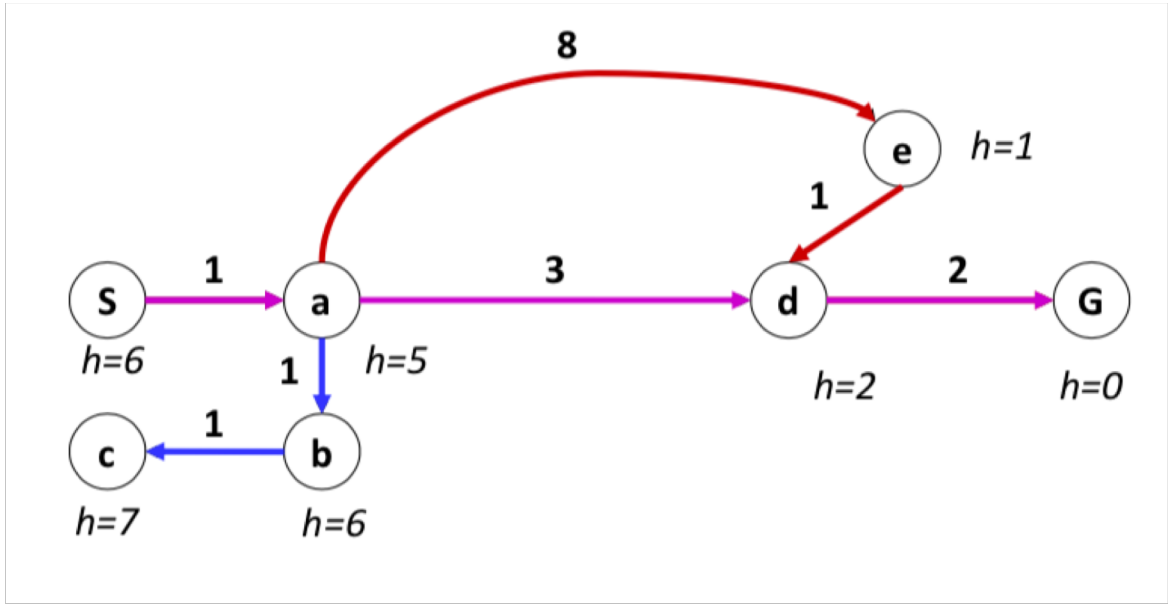
Dijkstra's

- Procedure
  - g(S → a) = 1
  - g(S → a → e) = 1 + 8 = 9
  - g(S → a → d) = 1 + 3 = 4
  - g(S → a → b) = 1 + 1 = 2
  - g(S → a → b → c) = 2 + 1 = 3
  - g(S → a → d → G) = 4 + 2 = 6
- Path
  - S → a → (b → c) → d → G = 1 + 1 + 1 + 3 + 2 = **8**
  - **Eventually** find optimal path

Slide adopted from UC Berkeley CS188 F13

A* Search

Dijkstra's orders by path cost: *g(n)*
Greedy Heuristic orders by goal proximity: *h(n)*
A* orders by the sum ***f(n) = g(n) + h(n)***



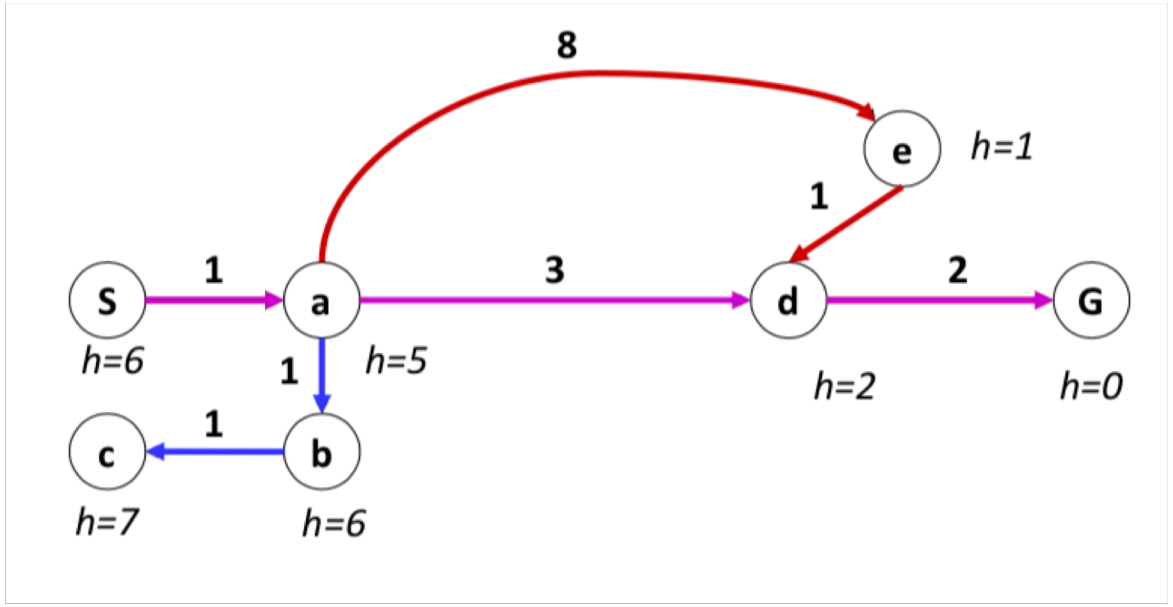
Greedy Heuristic

- Procedure
  - h(S → a) = 5
  - h(S → a → e) = h(e) 1
  - h(S → a → d) = h(d) = 2
  - h(S → a → b) = h(b) = 6
  - h(S → a → e → d) = h(d) = 2
  - h(S → a → e → d → G) = h(G) = 0
- Path
  - S → a → e → d → G = 1 + 8 + 1 + 2 = **11**
  - **Sub**-optimal path!

Slide adopted from UC Berkeley CS188 F13

A* Search

Dijkstra's orders by path cost: *g(n)*
Greedy Heuristic orders by goal proximity: *h(n)*
A* orders by the sum ***f(n) = g(n) + h(n)***



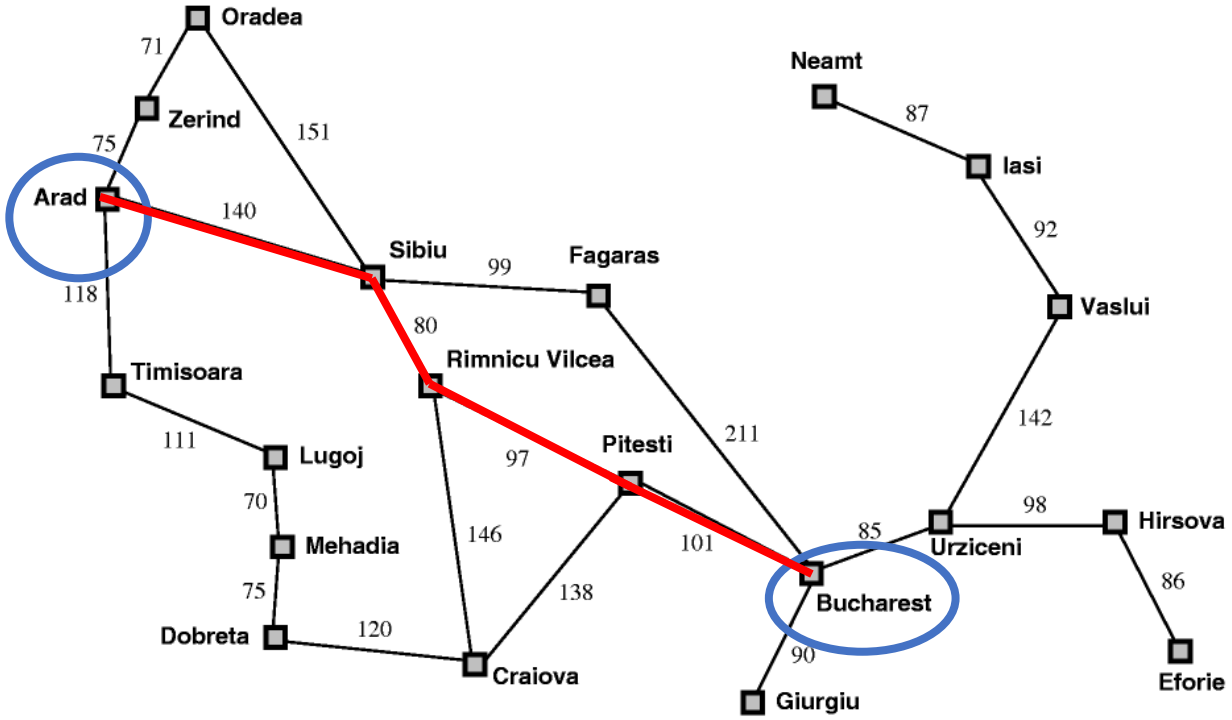
A*

- Procedure
  - f(S → a) = g + h = 1 + 5 = 6
  - f(S → a → e) = g + h = 9 + 1 = 10
  - f(S → a → d) = g + h = 4 + 2 = 6
  - f(S → a → b) = g + h = 2 + 6 = 8
  - f(S → a → d → G) = g + h = 6 + 0 = 6
- Path
  - S → a →d → G = 1 + 3 + 2 = **6**
  - **Optimal** path!

Slide adopted from UC Berkeley CS188 F13

A* Search

Try it yourself!



Heuristic *h(n)*

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

A → B = g(n) + h(n) = f(n)

Arad → Sibiu = 140 + 253 = 393

Arad → Zerind = 75 + 374 = 449

Arad → Timisoara = 118 + 329 = 447

Sibiu → Fagaras = 99 + 178 = 277

Sibiu → Rimnicu = 80 + 193 = 273

Rimnicu → Pitesti = 97 + 98 = 195

Rimnicu → Craiova = 146 + 160 = 306

Pitesti → Bucharest = 101 + 0 = 101

*What if this were **500**??*

A* Search

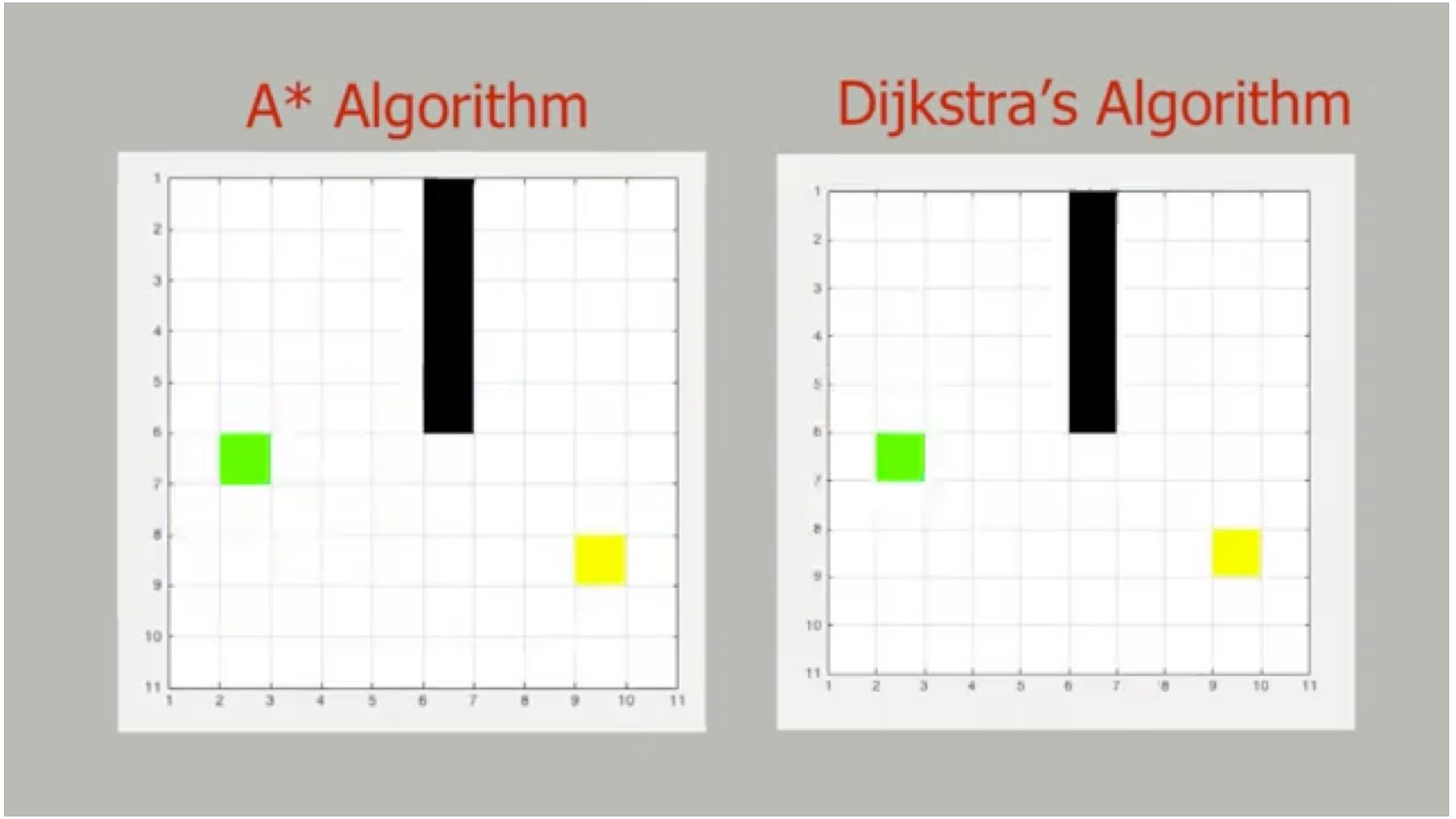
Does it guarantee the shortest path?

- Yes (conditions apply)
- All depends on your heuristic

Heuristic has to be **admissible** and **consistent**

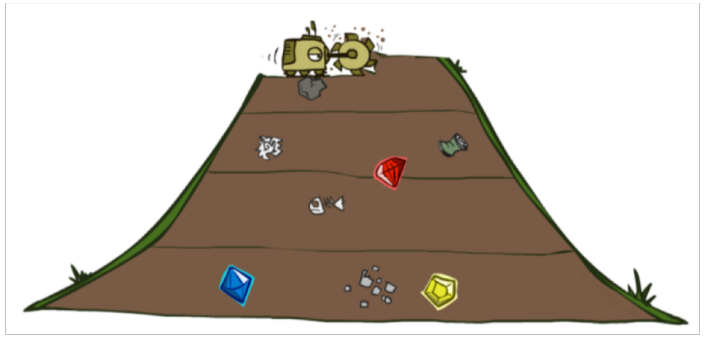
Active area of research: how to make a good *application-specific* heuristic?

Comparing Graph Search Algorithms

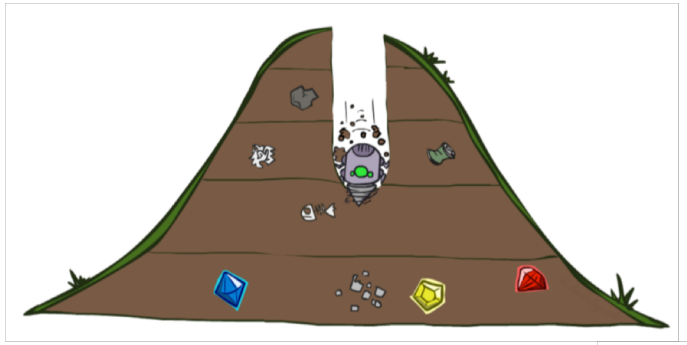


Video adopted from UPenn "Robotics: Computational Motion Planning"

Comparing Graph Search Algorithms



Breadth First Search (or Grassfire)



Depth First Search

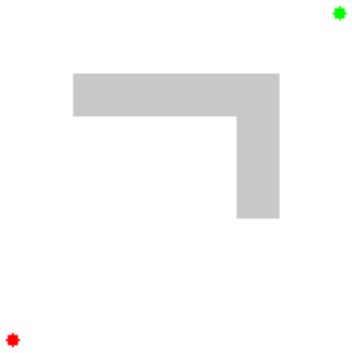
Dijkstra's (Uniform Cost Search)

A*

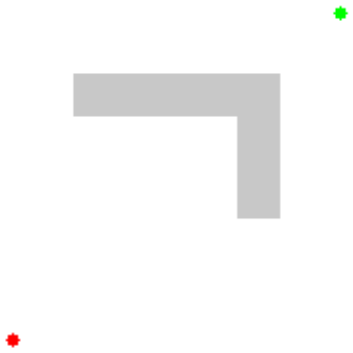
Greedy Heuristic

Artwork adopted from UC Berkeley CS188 F13

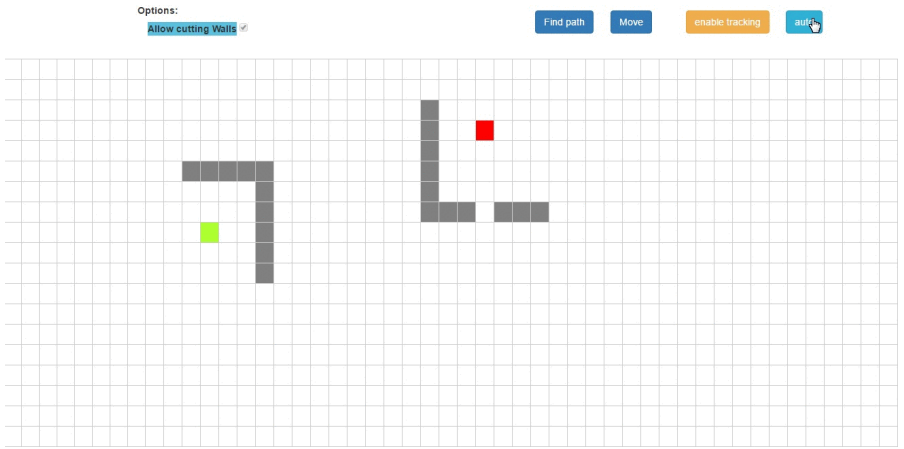
Comparing Graph Search Algorithms



Dijkstra's (Uniform Cost Search)



A*



A*

Gifs from Wikipedia

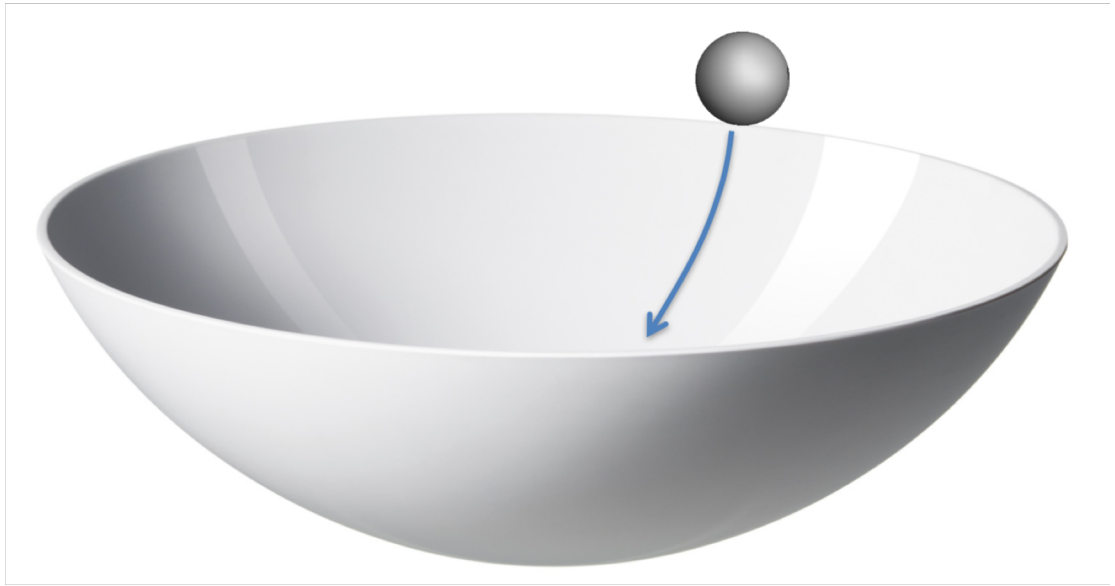
Artificial Potential Fields

What if we want to search *incrementally* (on-the-fly)?
What if we want to figure out where to go from just *local* data?
What if we operate in a *continuous* space? (not discrete, like graph/grid)

A* is a high-level, discrete planner...

One idea... **bowls**

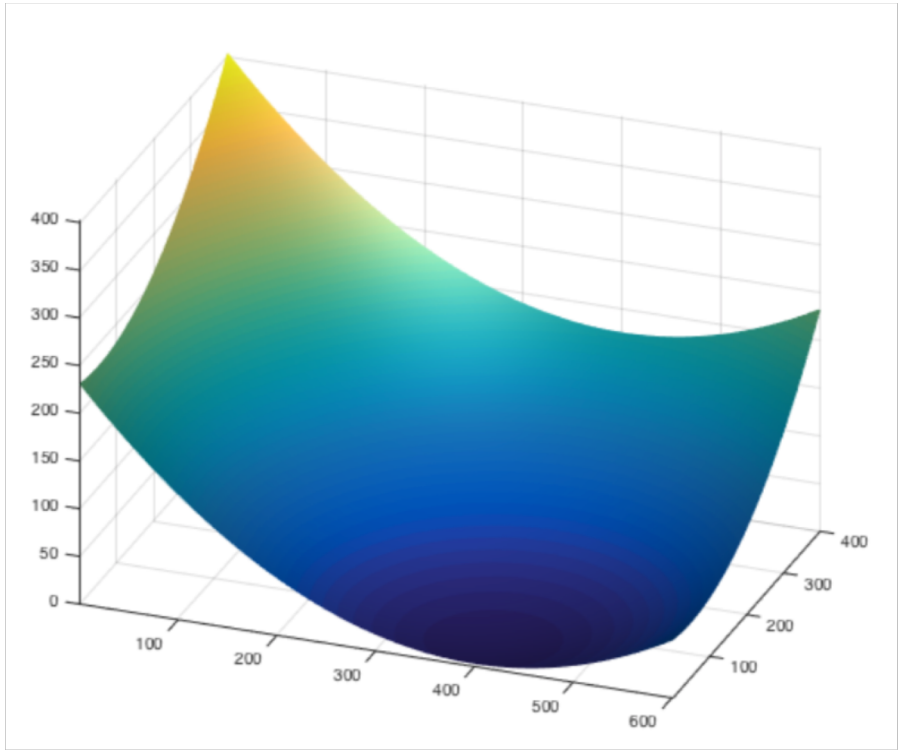
- Goal at bottom
- Robot at top



Artificial Potential Fields

Attractive Potential Field

- Moves us closer to the goal
- Example: 0 at goal, d^2 at any distance d away from goal (quadratic/parabolic)

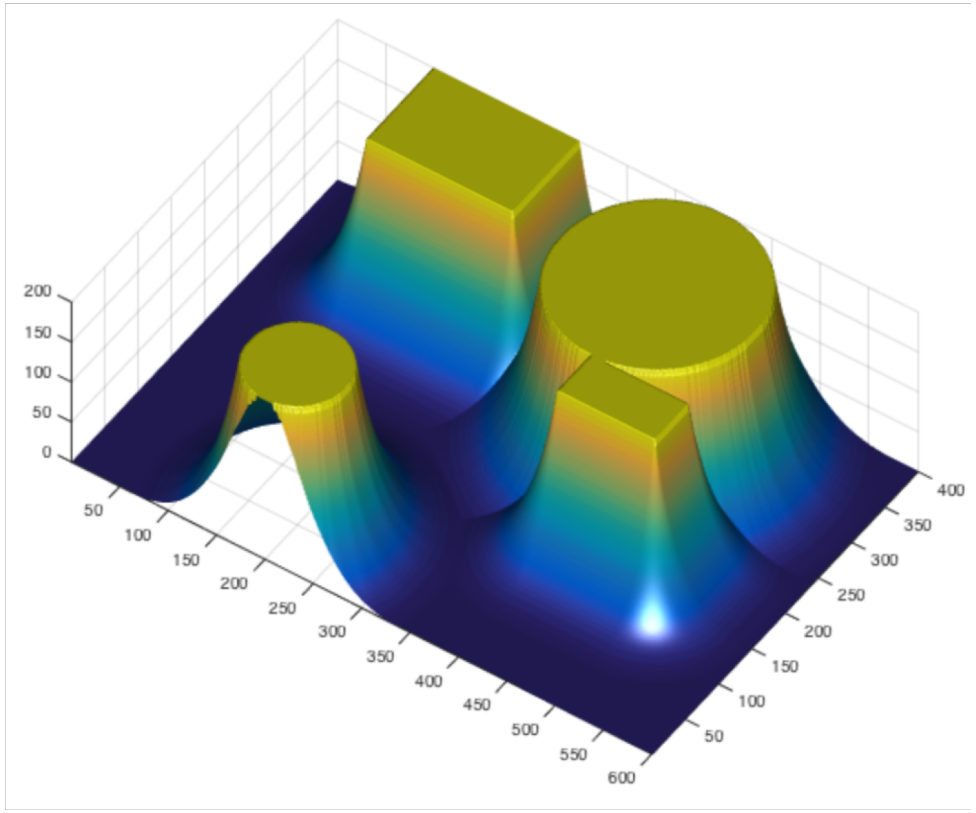


Artificial Potential Fields

What about obstacles?

Repulsive Potential Field

- Moves us away from obstacles
- Example: ∞ near goal, 0 at any other point

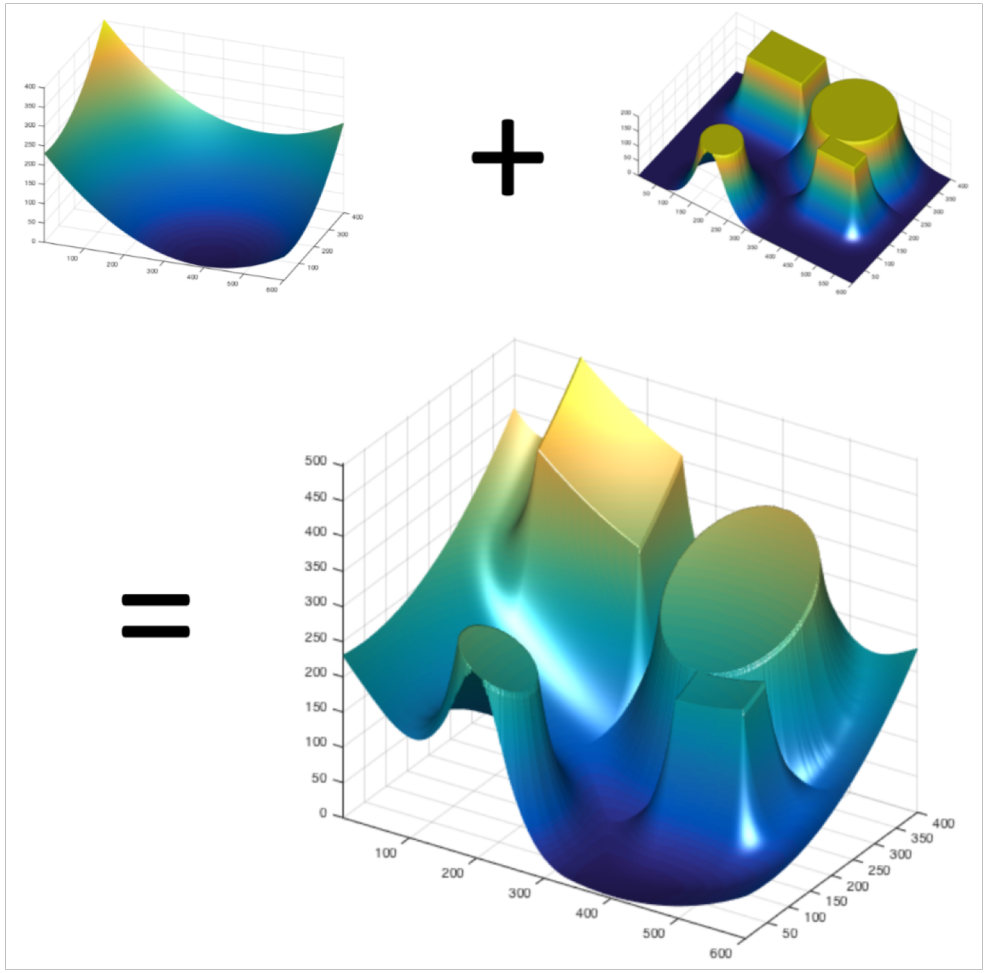


Artificial Potential Fields

As with most things in this class...

Artificial Potential Fields

Artificial Potential Field = Attractive Field + Repulsive Field



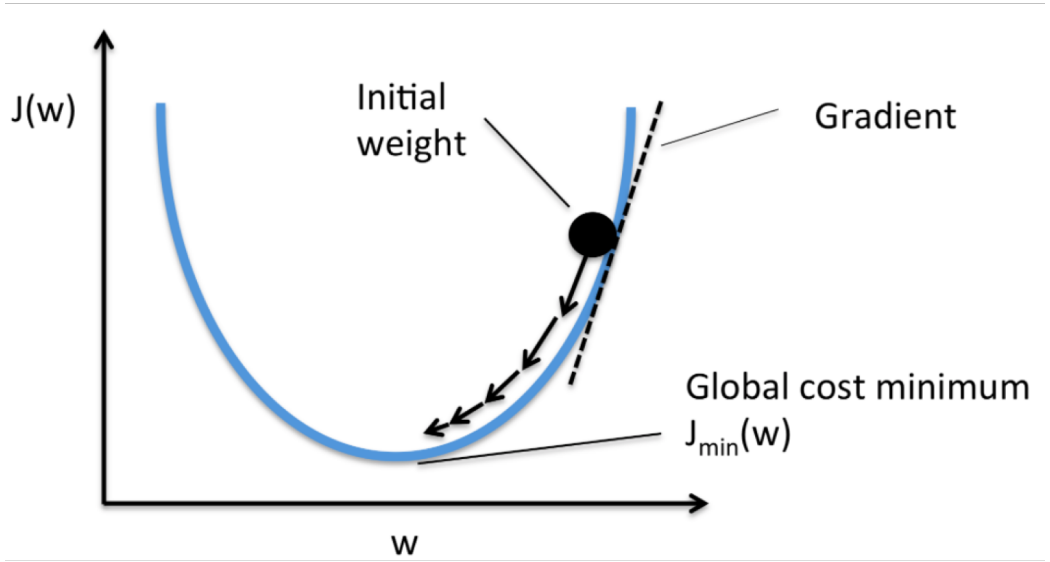
Artificial Potential Fields

How do you know which *direction* to go?

- **Gradient!**
- This is called Gradient Descent (common ML technique)

How do you know what *speed* to go at?

- You can set it!
- Some learning rate?



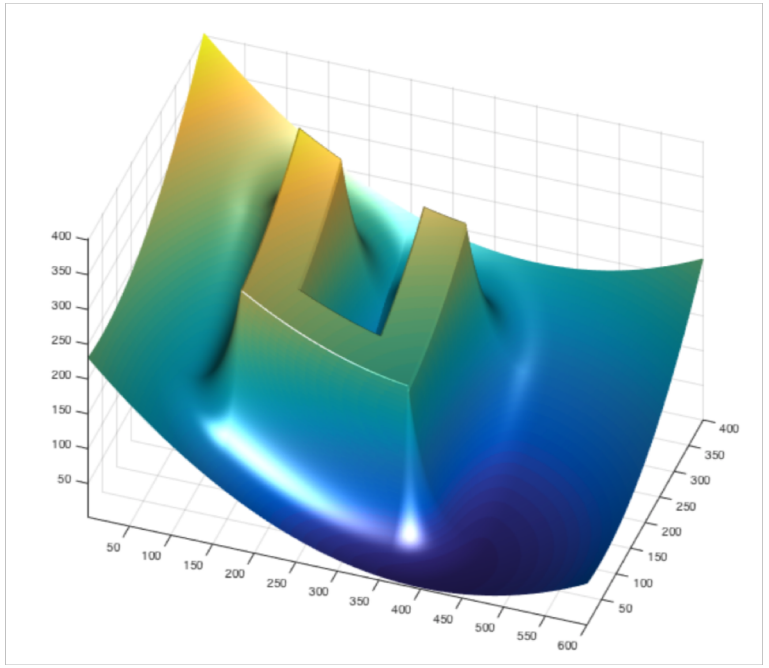
Artificial Potential Fields

Pros

- Can get away with using just position data (for attractive field) and local sensors (for repulsive field)... use this on the fly!
- Usually inexpensive computational/storage

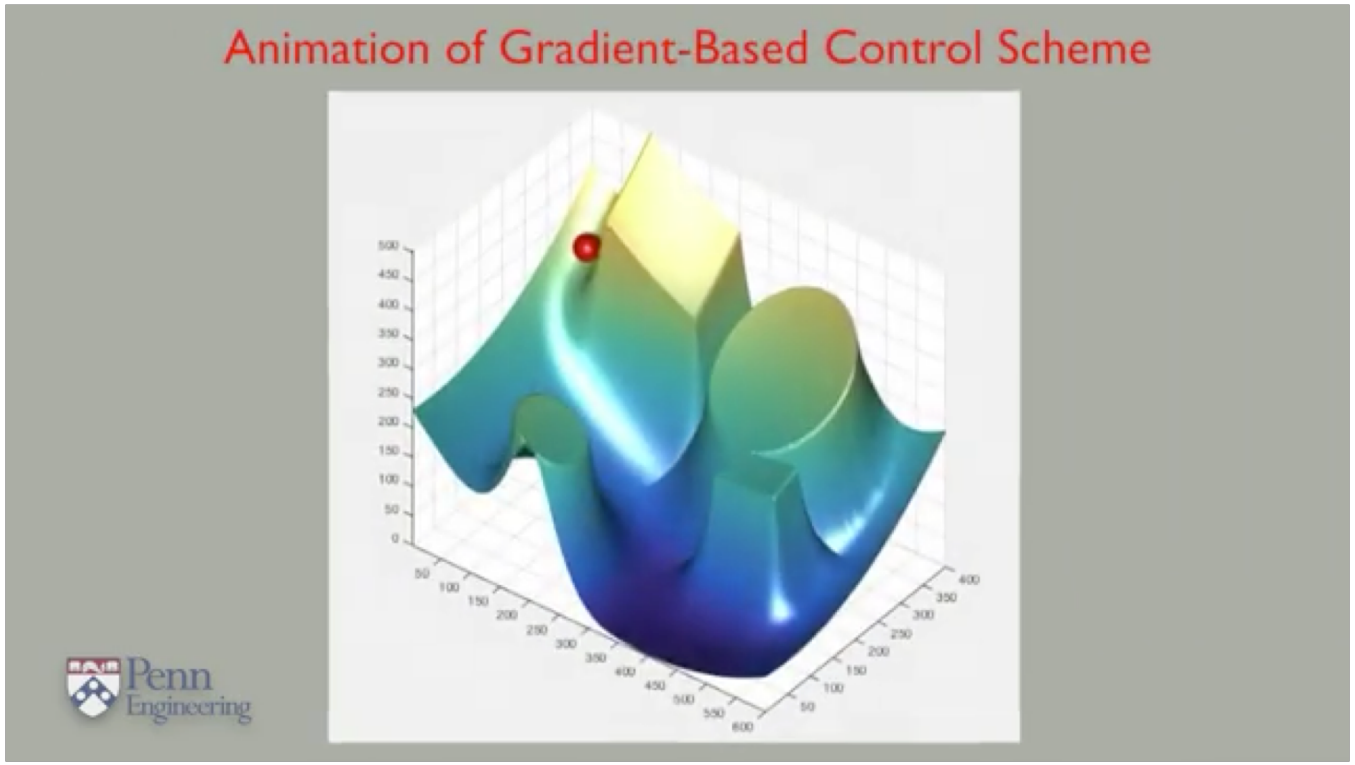
Cons

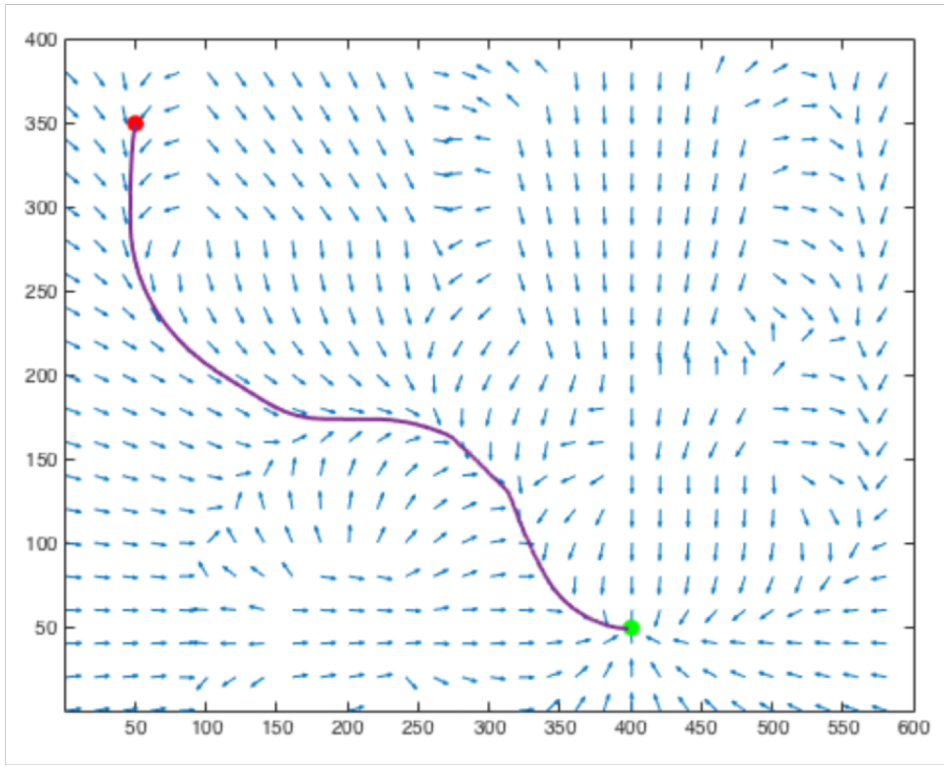
- Get stuck in Local Minima!
  - Classic Gradient Descent Problem...
- Too close to obstacles?
  - **Not optimal**!



Artificial Potential Fields

Examples





Adopted from UPenn “Robotics: Computational Motion Planning”