CS 440
Introduction to Artificial Intelligence

Lecture 3:

Search & Genetic Algorithms

28 January 2020

SPN numbers

SPN numbers have been issued

Lecture Outline

Review Hill climbing

Genetic Algorithms

Heuristic Search

Hill Climbing

Hill climbing

Always select action that leads to state with best heuristic

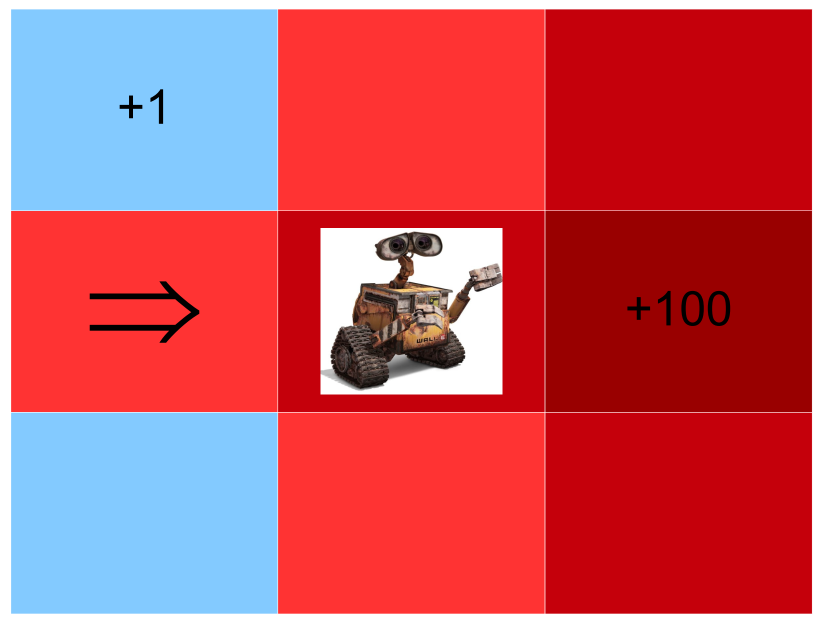
Let s = current state

  For all $a \epsilon A$

    $s' = \tau(s,a)$

    $h_a = h(s')$

  Select a with largest reward h_a



Gradient Ascent / Descent

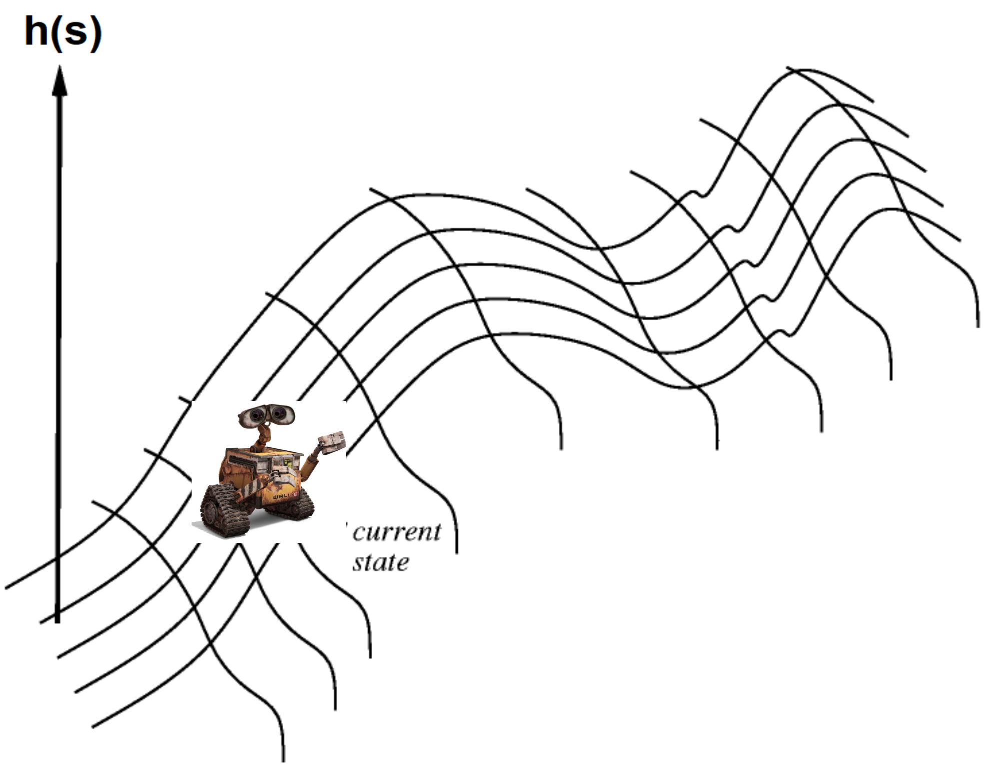
Can we apply hill climbing to continuous state spaces?

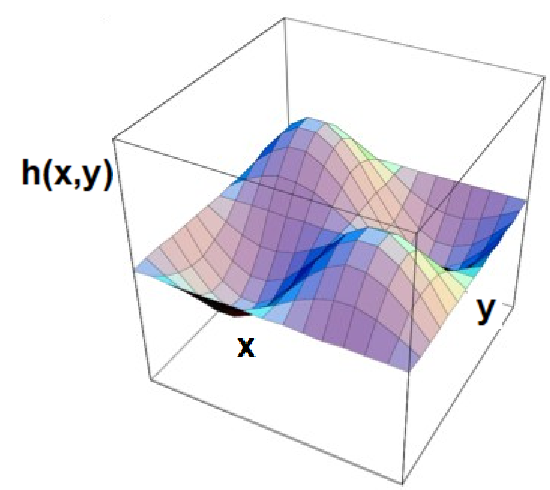
Example 2D Euclidean space

Gradient ascent / descent

Move in direction of gradient

$\nabla h(s)$

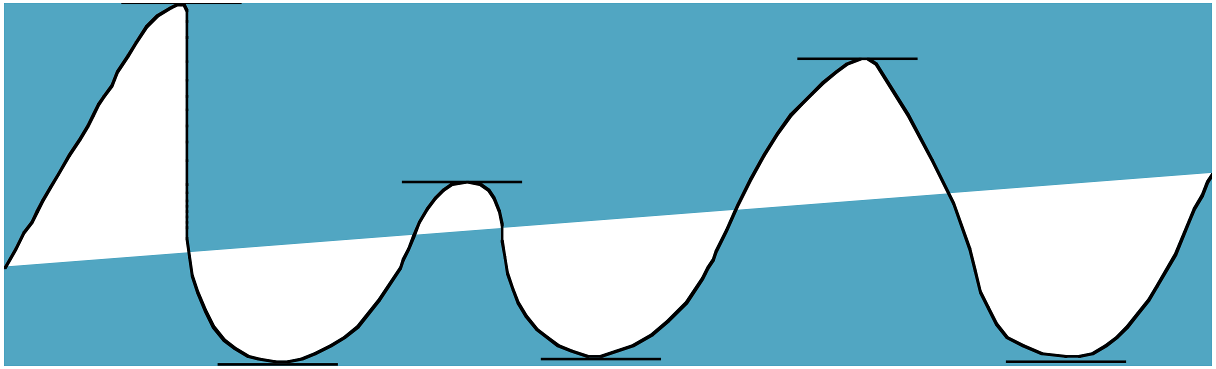




Gradient Descent

Discussion

- How does GD algorithm know it has reached min/max?
  - Hint GD computes direction by computing $\nabla h(s)$
- What are some methods for escaping local minimums?
- What are some methods for approximating gradient descent if gradient can't be computed?



What *is* a genetic algorithm?

- Genetic algorithms are a randomized local search <u>strategy</u>.

- Basic idea: Simulate natural selection, where the population is composed of
  - *an evolving population of candidate solutions*.

- Focus is on evolving a population from which strong and diverse candidates can emerge via:
  - survival of the fittest,
  - crossover (mating),
  - and mutation.

Basic genetic algorithm

Create an initial population, either random or “blank”.

While the best candidate so far is not a solution:

> Create new population using successor functions.
>
> Evaluate the fitness of each candidate in the population.

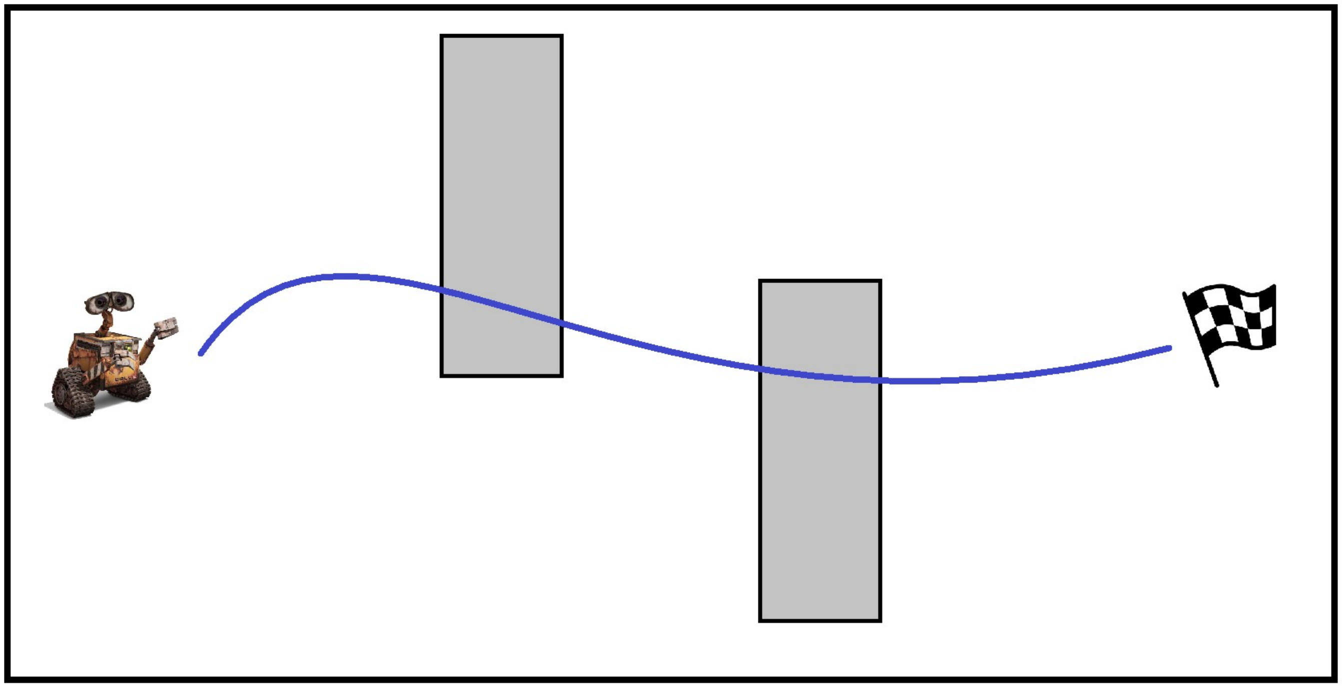
Return the best candidate found.

Simple example – alternating string

- Let's try to evolve a length 4 alternating string
  - Fitness function - number of consecutive occurrences
  - Evolve by flipping 1 bit
- Initial population: C1=0011
- We roll the dice and end up creating C1' = 1011 and C2' = 0001.
  - C1' = 1011 $\Rightarrow$ score of -1
  - C2' = 0001 $\Rightarrow$ score of -2
  - Keep C1'
- Roll the dice again and end up creating C1'' = 1001 and C2'' = 1010.
  - We run our solution test on each. C2'' is a solution, so we return it and are done.

Example - Robot path planning

Genetic algorithm example: Robot path planning



Simple example – alternating string

- Let's try to evolve a length 4 alternating string
  - Fitness function - number of consecutive occurrences
- Initial population: C1=1000, C2=0011
- We roll the dice and end up creating C1' = cross (C1, C2) = 1011 and C2' = cross (C1, C1) = 1000.
  - C1' $\Rightarrow$ score of -1
  - C2' $\Rightarrow$ score of -3
  - Keep C1'
- We mutate C1' and the fourth bit flips, giving 1010. We mutate C2' and get 1000.
  - We run our solution test on each. C1' is a solution, so we return it and are done.

Generic Algorithms - Discussion

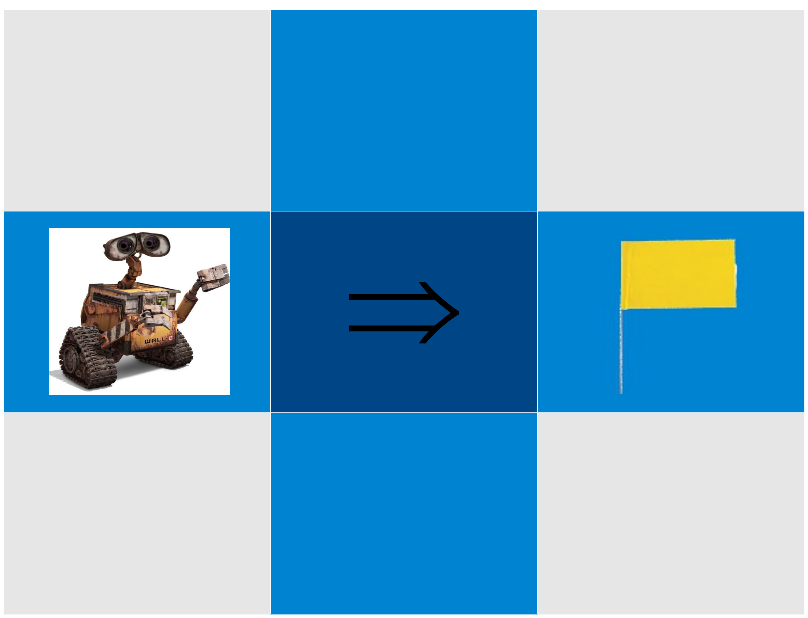
- What are some possible variations of genetic algorithms?
  - When would these variations be advantageous
- What are some limitations of genetic algorithms?

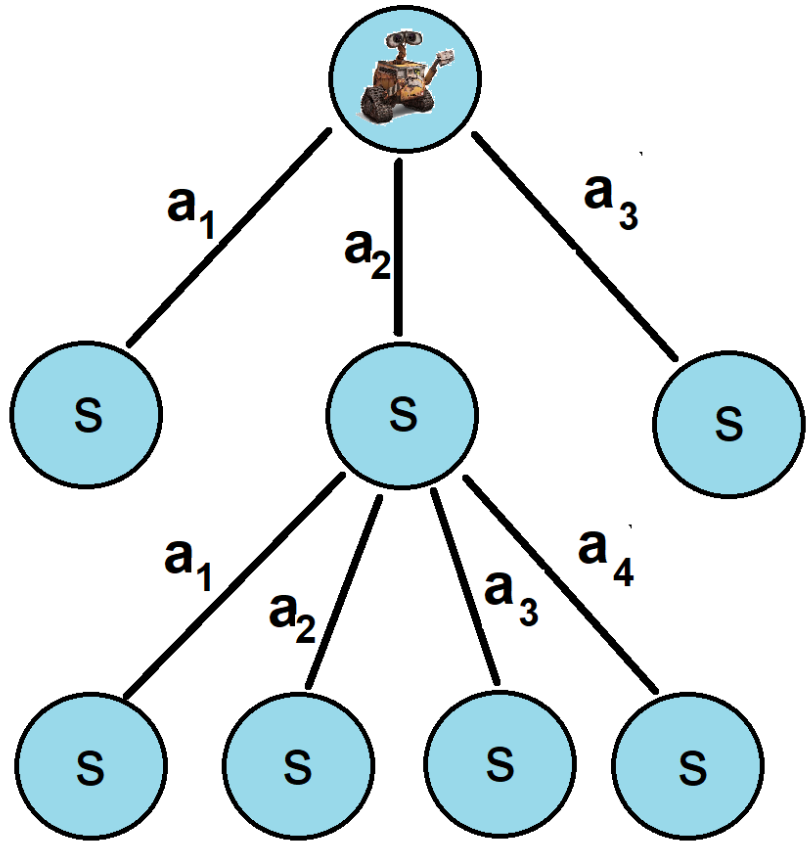
Alternating string with corssover

- Initial population: C1=0011, C2=1000
- We roll the dice and end up creating C1’ = cross (C1, C2) = 1011, C2’ = cross (C2, C2) = 1000, C3’ = cross (C1, C1) = 0011 and C2’ = cross (C1, C2) = 0110
  - C1' = 1011 $\Rightarrow$ score of -1
  - C2' = 1000 $\Rightarrow$ score of -2
  - C3' = 0011 $\Rightarrow$ score of -2
  - C4' = 0110 $\Rightarrow$ score of -1
  - Keep C1' and C4'
- Roll the dice again and end up creating C1’’ = cross (C1, C1) = 1011 and C2’’ = cross (C1, C4) = 1010.
  - We run our solution test on each. C2’’ is a solution, so we return it and are done.

Represent as a tree

- Nodes represent states
- Edges represent actions
- Root is current state
- Children - states you get by taking each action

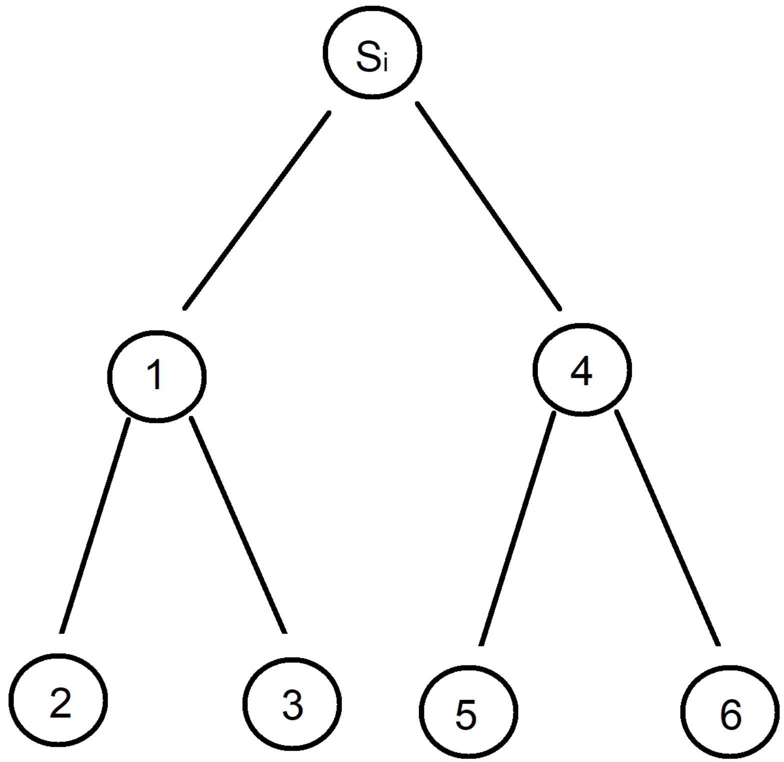




Depth First (Beam) Search

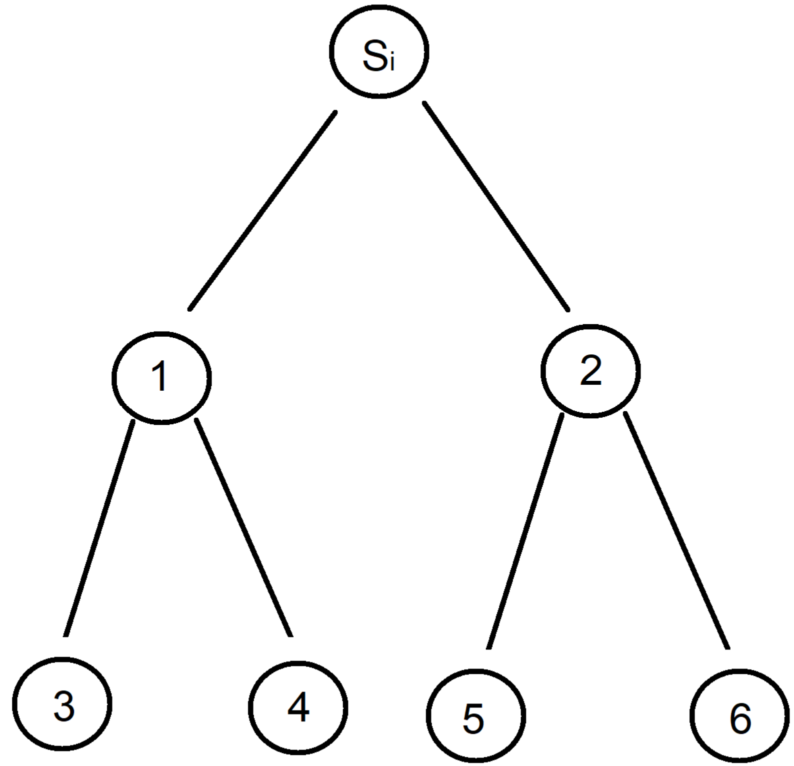
Depth first search

```
stack.push(initial state)
While(!stack.empty())
    s = stack.pop
    if(s == goal)
        return
    if (!s.visited)
        s.visited=true
        for all actions aϵA
            s'=τ(s,a)
            stack.push(s')
```



Breath first search

```
queue.push(initial state)
While(!queue.empty())
    s = queue.pop
    if(s == goal)
        return
    if (!s.visited)
        s.visited=true
        for all actions aϵA
            s'=τ(s,a)
            queue.push_back(s')
```



Weighted Actions

Weighted actions

- Each action has a cost associated with it
- c(a) = Cost of performing action a in state s

Variations

- Cost of going through state
- Cost of performing action while in state
  - c(a,s)

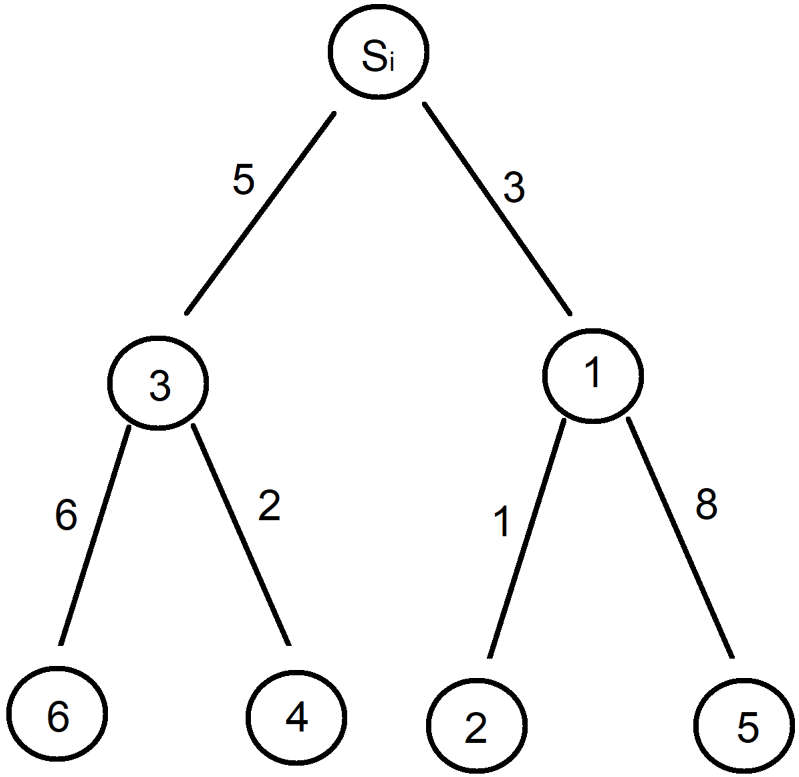
How would we adapt BFS to accommodated weighted actions

- Find shortest path to goal

Shortest Path First (SPF) Algorithm

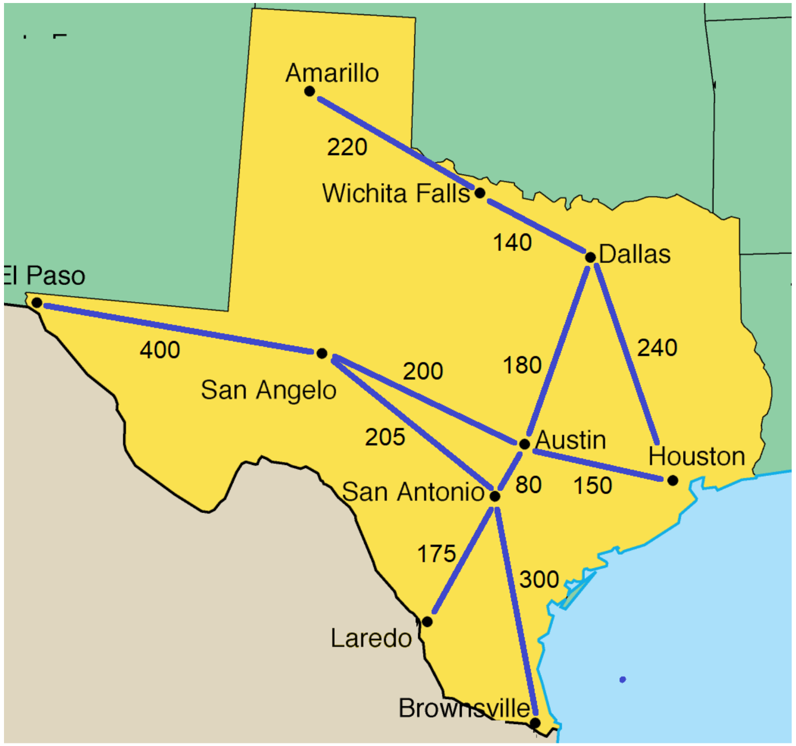
- Priority queue ordered by path length

```
priority_queue.push(initial state, 0)
While(!priority_queue.empty())
    s = priority_queue.pop()
    if(s == goal)
        return
    if (!s.visited)
      • s.visited=true
    for all actions aϵA
        s'=τ(s,a)
        s'_path_length = s_path_length + length(a)
        priority_queue.push(s', s'_path_length)
```



SPF Example

- Map of driving routs
  - Find shortest rout
- Find shortest path first algorithm also called Dijkstra's Algorithm



Heuristic

- Estimate of utility of state
- Real value indicating utility of state
- Must be defined for all states in **S**

Convention

- $h(s) \Rightarrow \mathcal{R}$

Examples

- Chess: Value of pieces captured/lost
- Robotics: Euclidean distance from goal
- Sudoku: Numbers placed, rows/columns/blocks filled in.

Heuristic Searches

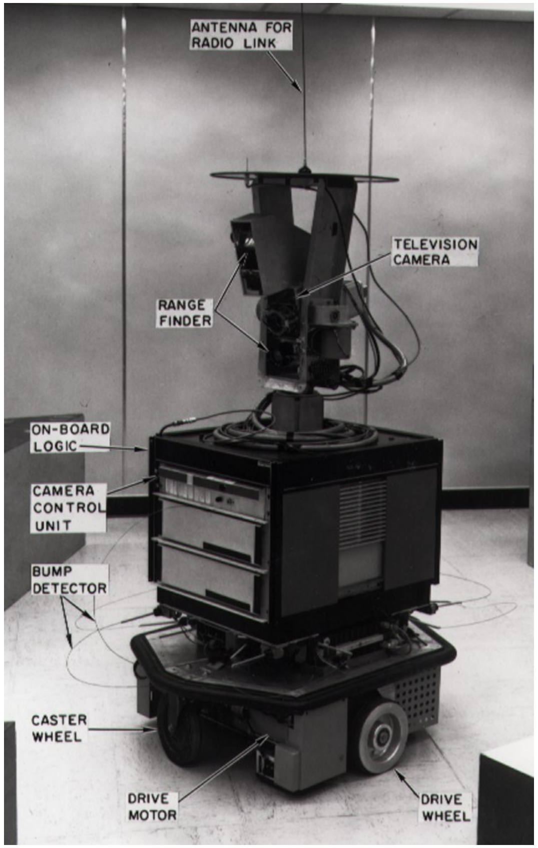
Always select open state with best heuristic value

```
priority_queue.push(initial state, 0)
While(!priority_queue.empty())
    s = priority_queue.pop()
    if(s == goal)
        return
    if (!s.visited)
      • s.visited=true
    for all actions aϵA
        s'=τ(s,a)
        priority_queue.push(s', h(s'))
```

Not guaranteed to find optimal path

A* Search

- Order nodes by sum of path length and heuristic
- Priority queue order by path(s) + h(s)
- Will find optimal path if using admissible heuristic



priority_queue.push(initial state, 0)
While(!priority_queue.empty())
  s = priority_queue.pop()
  if(s == goal)
    return
  if (!s.visited)
  - s.visited=true
  - for all actions $a \in A$

    $s' = \tau(s,a)$

    $s'_{path\_length} = s_{path\_length} + length(a)$

    priority_queue.push(s', $s'_{path\_length} + h(s')$)

Admissible Heuristics

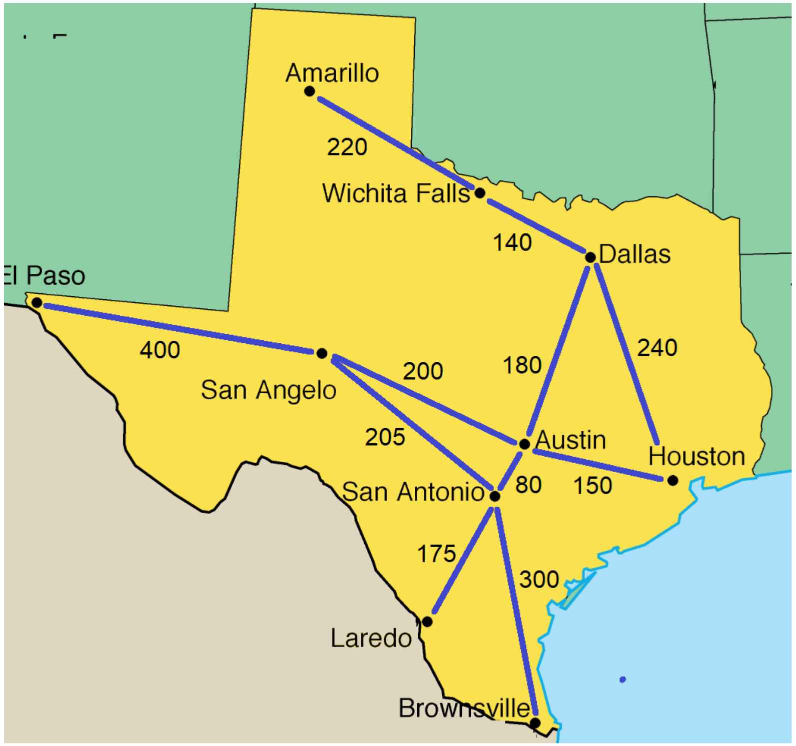
- Never overestimates cost to goal
  - h(s) < optimal_path(s,goal)

- What can we say about A* search with admissible heuristic

- Examples of admissible heuristics for
  - Robot
  - Sudoku
  - Chess
    - Not trivial

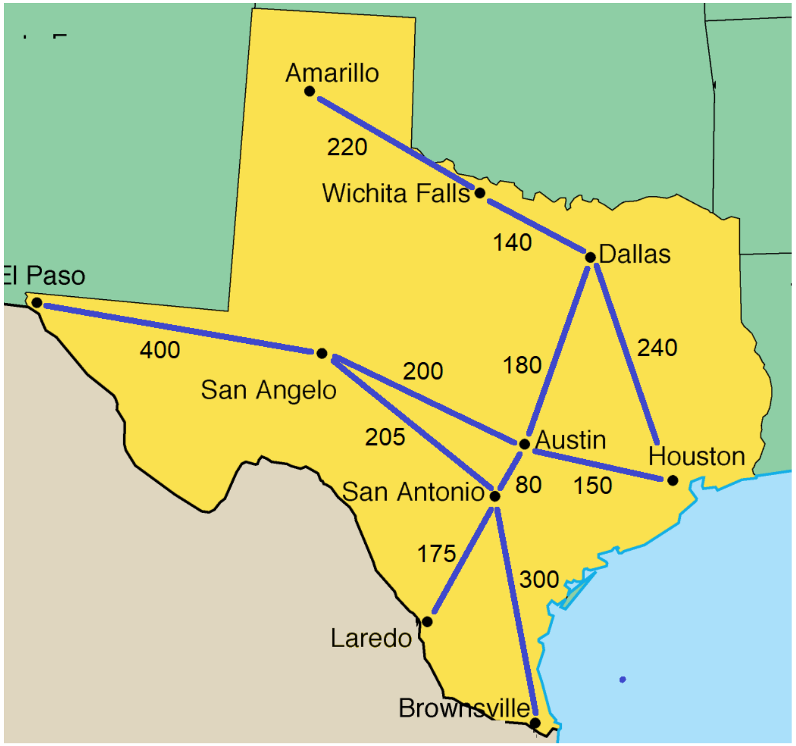
A* Search Example

- Map of driving routs
  - Find shortest rout
  - What are some good heuristics?



A* Search Example

- Map of driving routs
  - Find shortest rout
  - Heuristic = Euclidean distance to goal



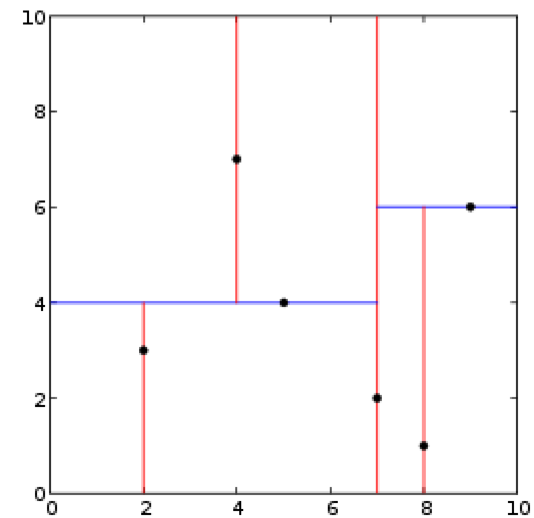
A* for continuous state space

How can we apply A* to problems with continuous state spaces?

- Robot in XY-plane

A* for continuous state space

- Generate graph in space of problem
- Grid up state space
  - What resolution to use
    - More cells $\Rightarrow$ higher computation cost
  - Complexity exponential with respect to dimension of state space
    - $O(x^d)$
    - Curse of dimensional
- Generate graph in space of problem
  - Example robotics
- Adaptive grid
  - Recursively subdivide cells that are interesting
  - Example: Robotics - use higher resolution cells in difficult regions



Can we apply A* to chess?

RUTGERS

Min max search

- Apply to adversarial/strategic problems
- Select best move during your turn and worst move (opponent's best move) during opponent's turn
  - Move you would make if you were opponent