COSC 411: Artificial Intelligence

Informed and Local Search

Dr. Shuangquan (Peter) Wang
(spwang@salisbury.edu)

Department of Computer Science

Salisbury University

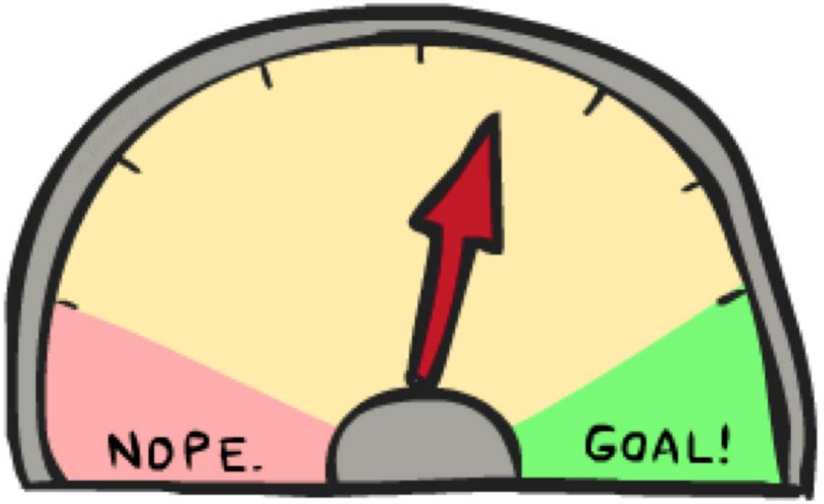
About this note

- Most slides of this note are from:
  - "CS 188 Introduction to Artificial Intelligence" teaching material, UC Berkeley, Summer 2020 (instructor: Nikita Kitaev)
    - https://inst.eecs.berkeley.edu/~cs188/su20/
  - "CS 188 Introduction to Artificial Intelligence" teaching material, UC Berkeley, Spring 2019 (instructor: Sergey Levine and Stuart Russell)
    - https://inst.eecs.berkeley.edu/~cs188/sp19/
  - "CSE 480/580 Introduction to Artificial Intelligence" teaching material, Old Dominion University, Spring 2023 (instructor: Vikas Ashok)
    - https://www.cs.odu.edu/~vashok/cs480.html

Contents

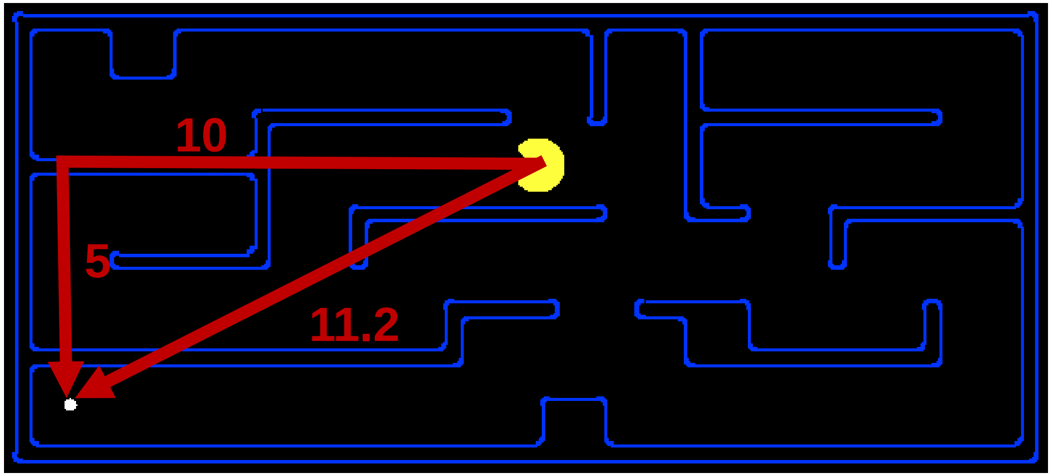
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
- Local Search

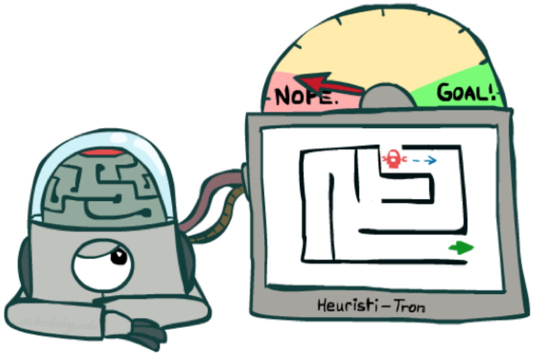
1. Informed Search

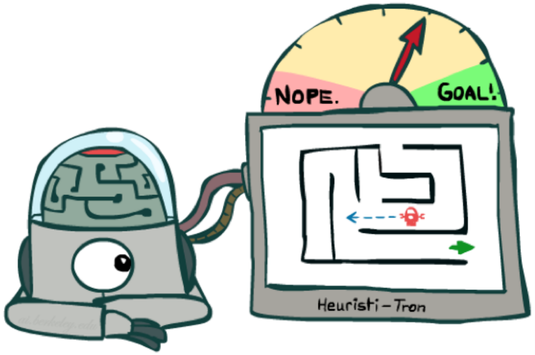


Search Heuristics

- 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

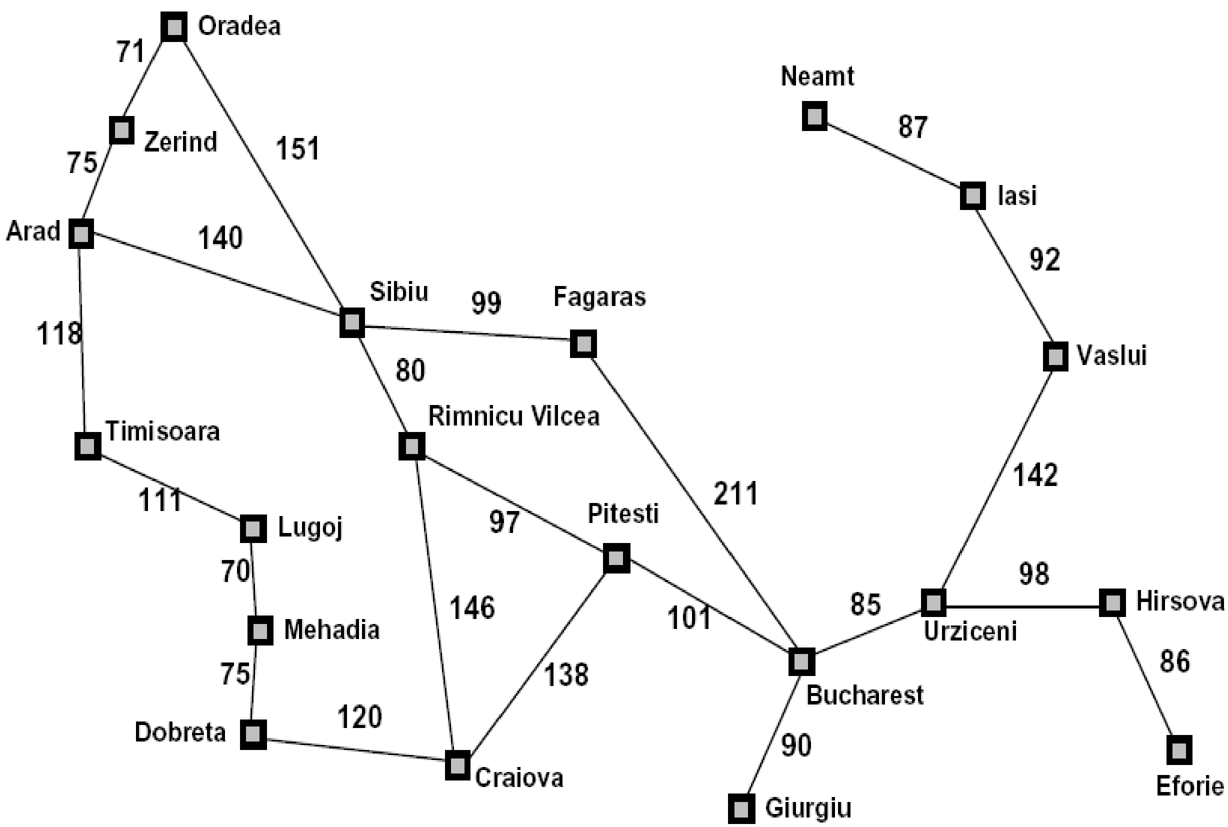






Greedy Search

Example: Heuristic Function



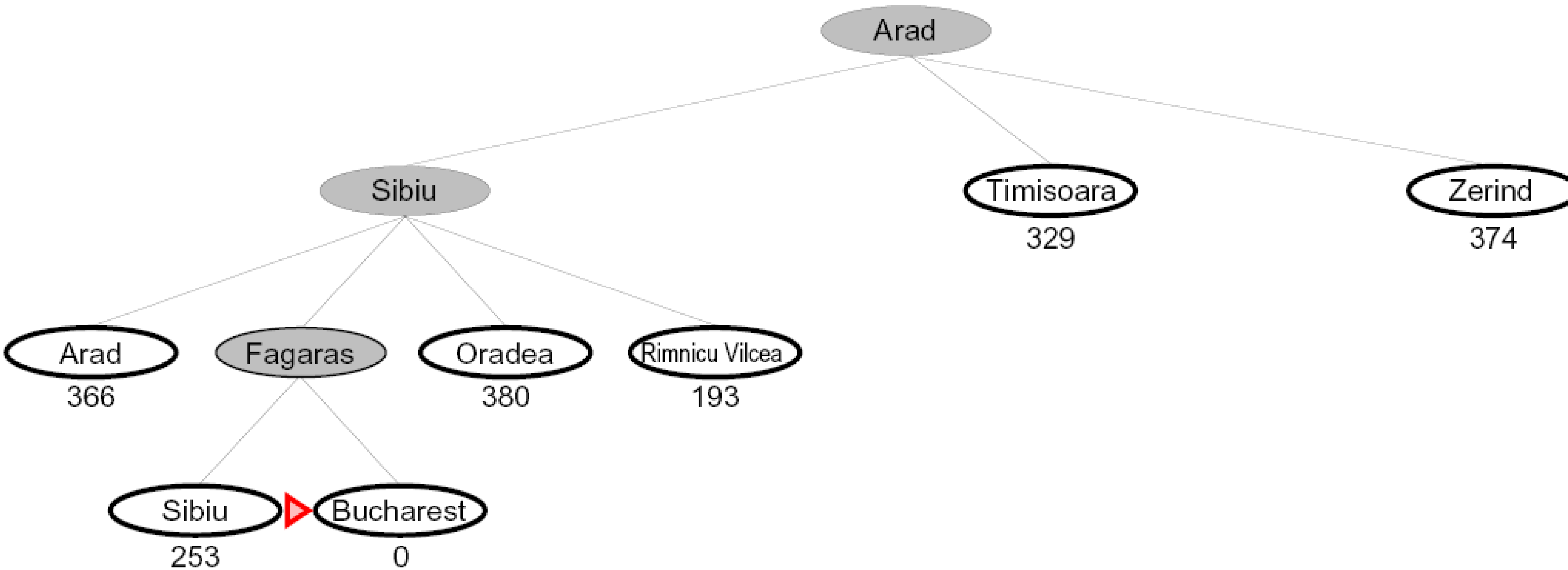
Straight−line distance to Bucharest

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

h(x)

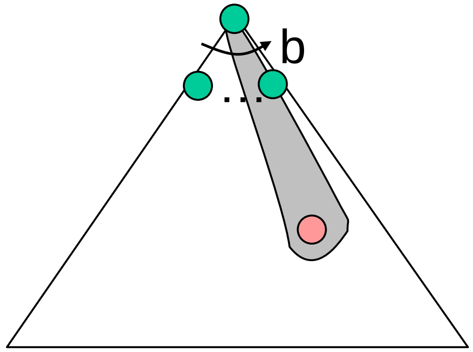
Greedy Search

- Expand the node that **seems closest**…

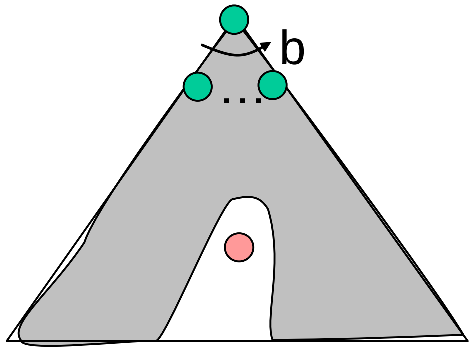


Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state

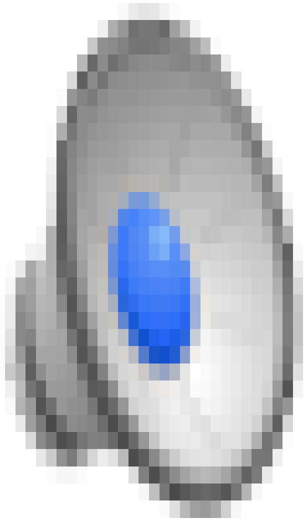


- A common case:
  - Best-first takes you straight to the (wrong) goal

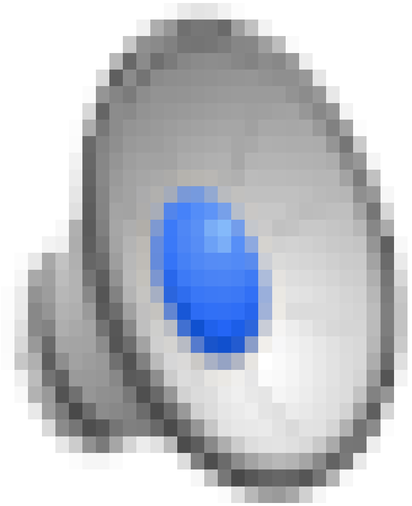


- Worst-case: like a badly-guided DFS

Video of Demo Contours Greedy (Empty)

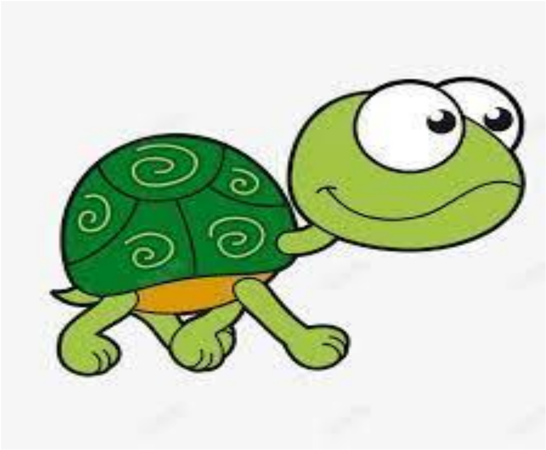


Video of Demo Contours Greedy (Pacman Small Maze)

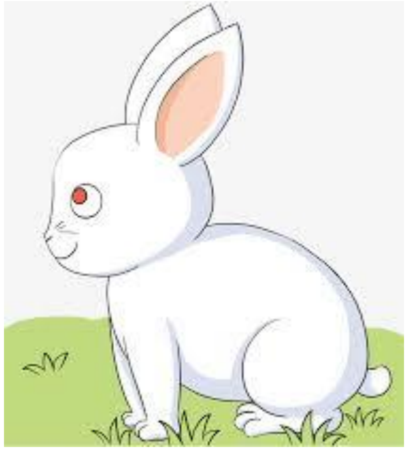


A* Search

A* Search



UCS



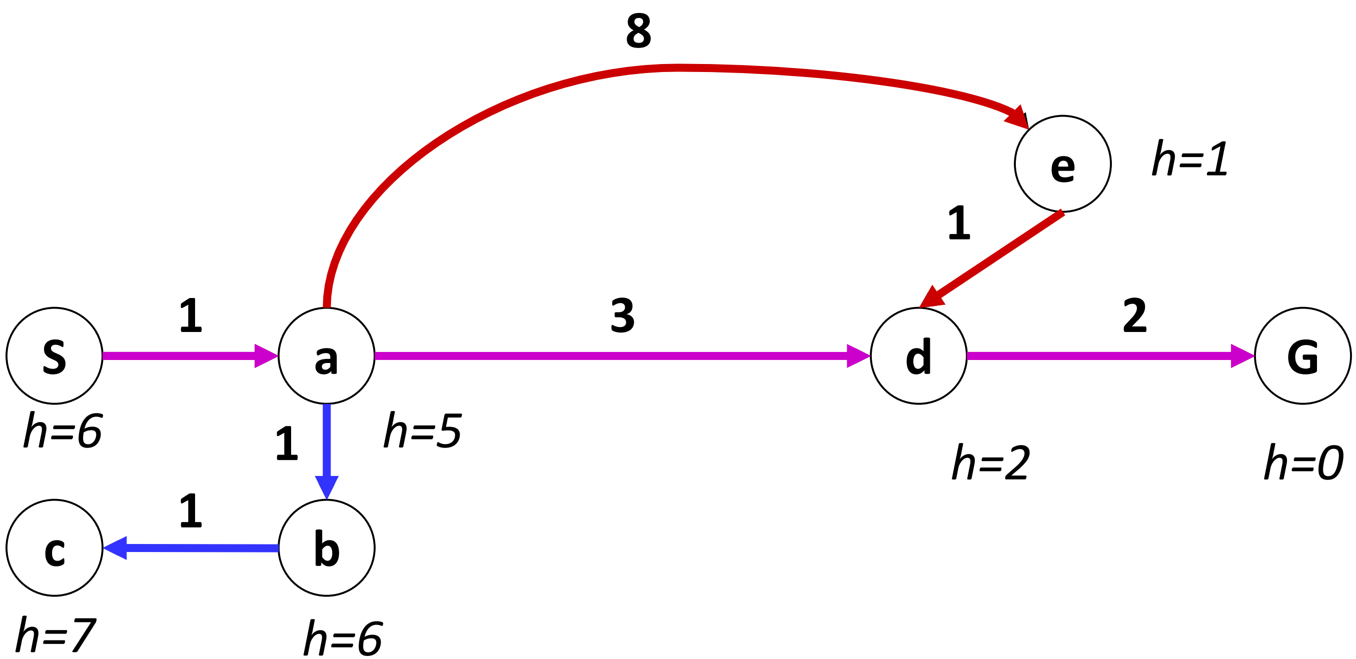
Greedy

A*

Note: these images are from google image search

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)

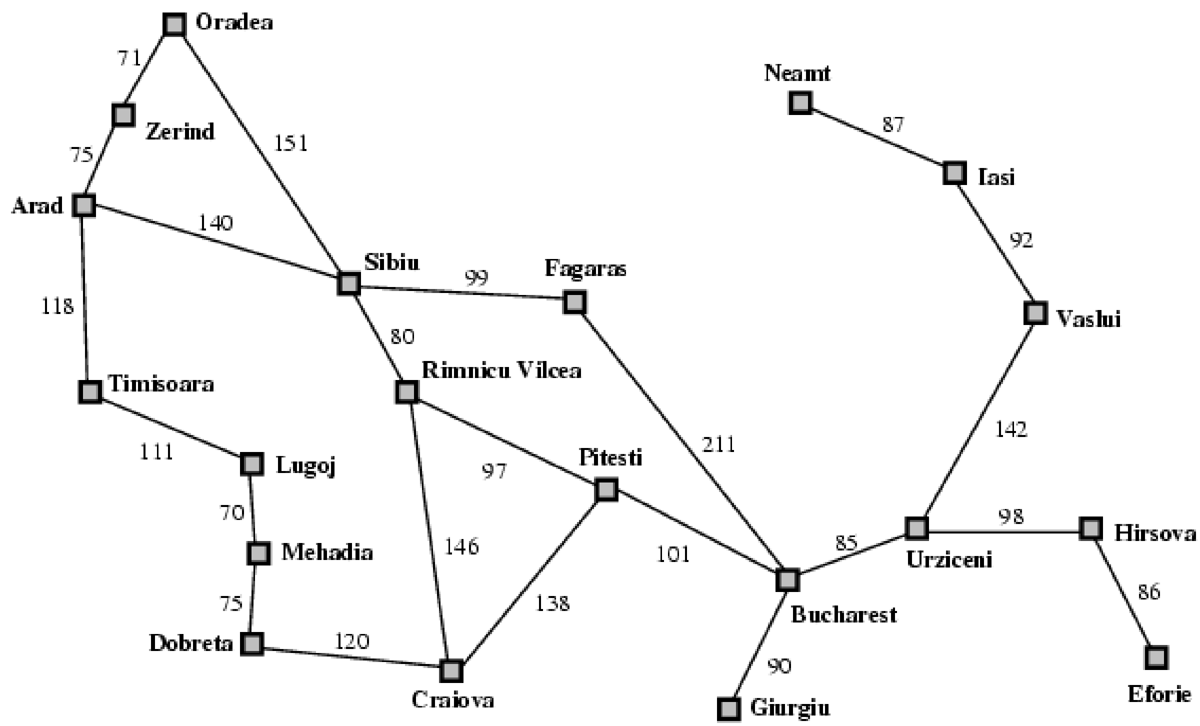


- A* Search orders by the sum: f(n) = g(n) + h(n)
  - *g(n): cost paid so far from the start node to this node*
  - *h(n): estimated min cost from this node to the destination node*

Example: Route Finding A* Search

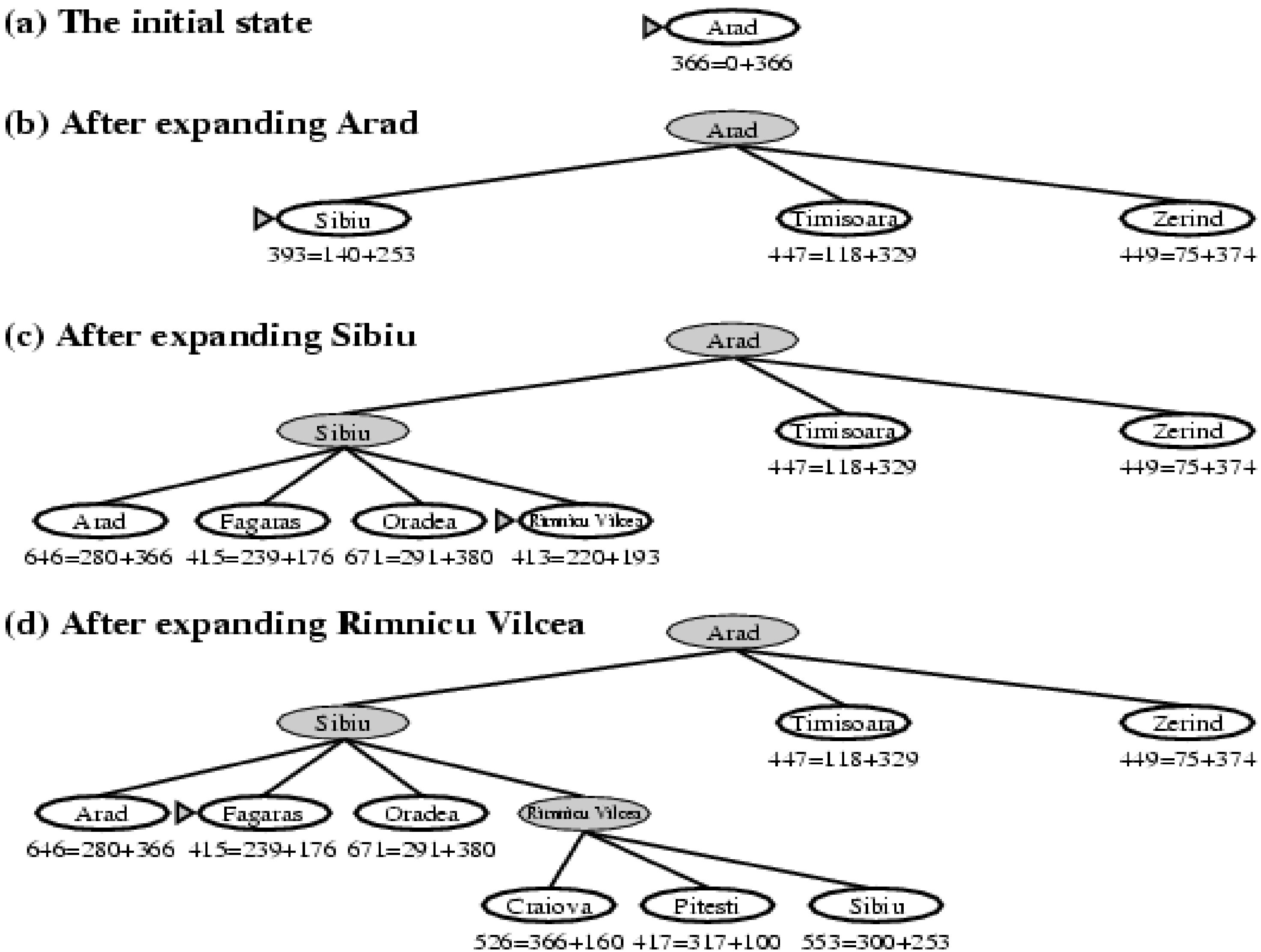
- Heuristic Function
  - Using the straight-line distance
- Path Cost Function
  - Using the path cost from the graph

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	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

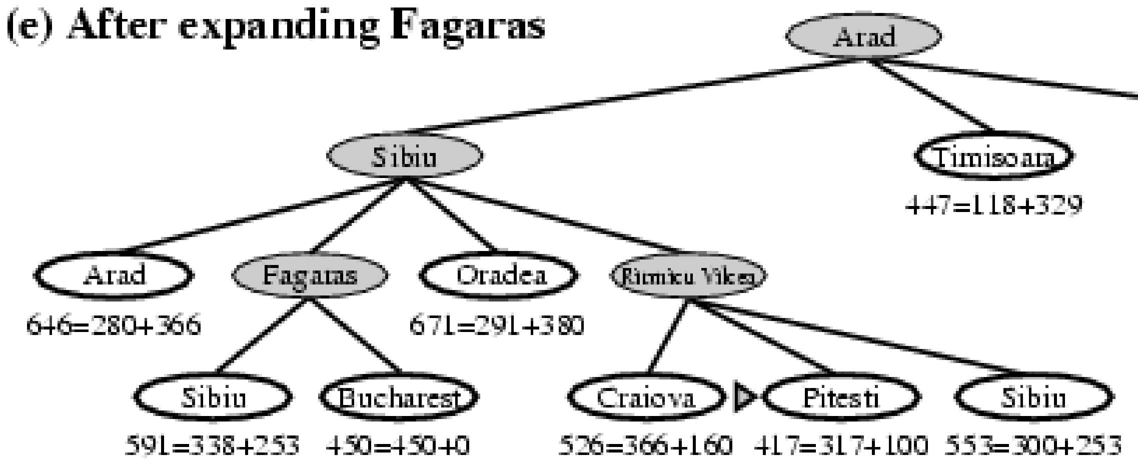


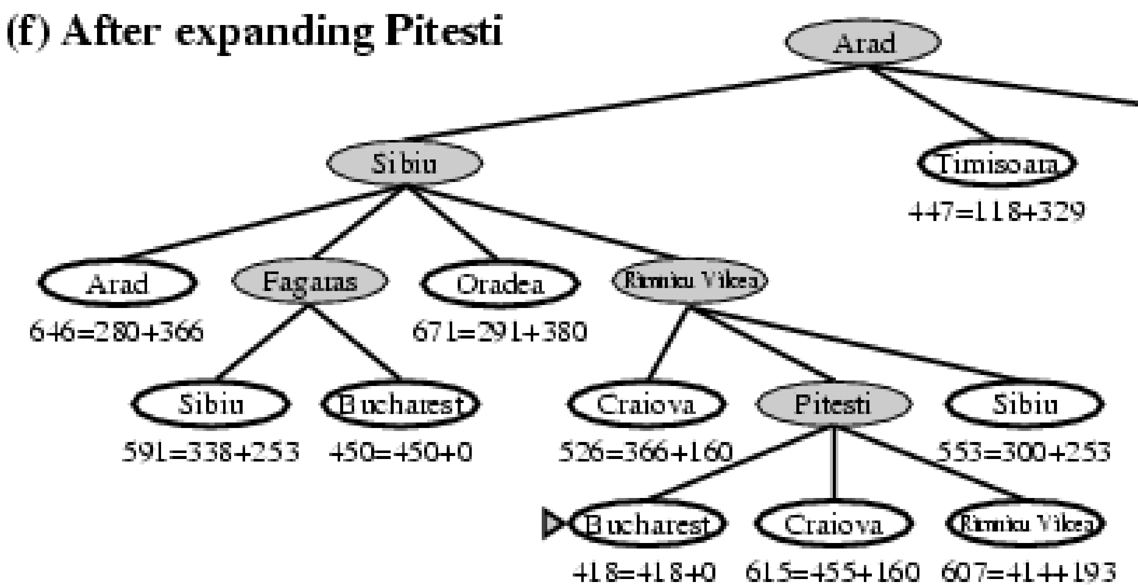
This slide is from https://www.cs.odu.edu/~vashok/cs480.html

Progress in A* Search

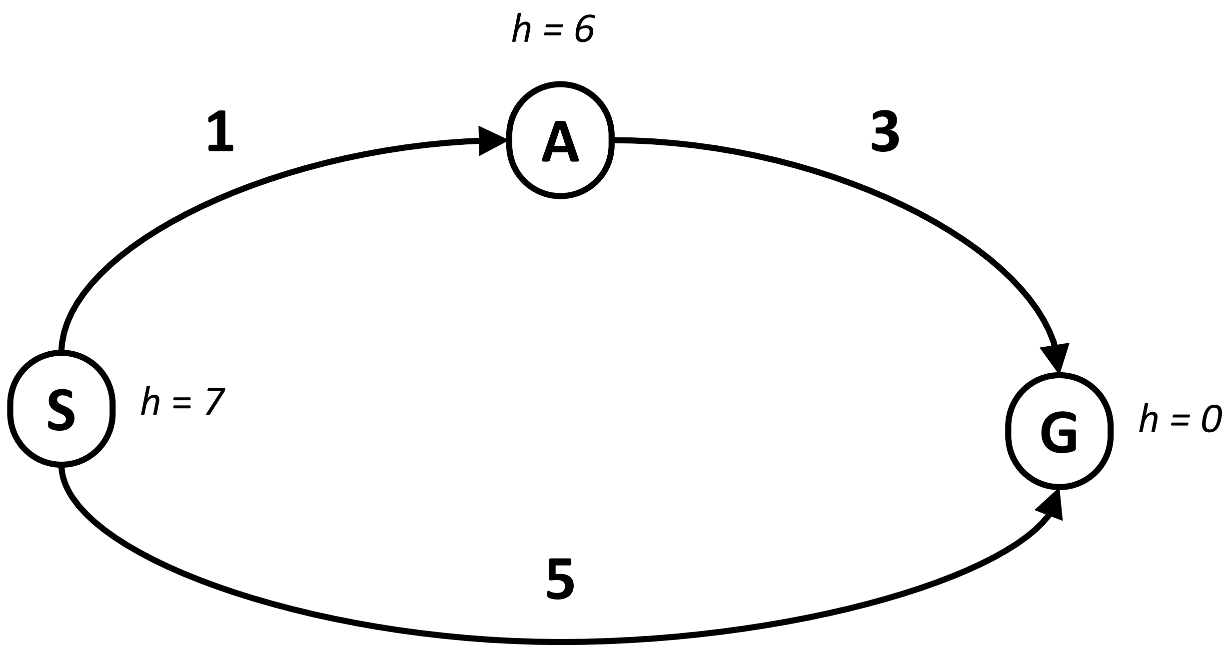


Progress in A* Search (Continue)



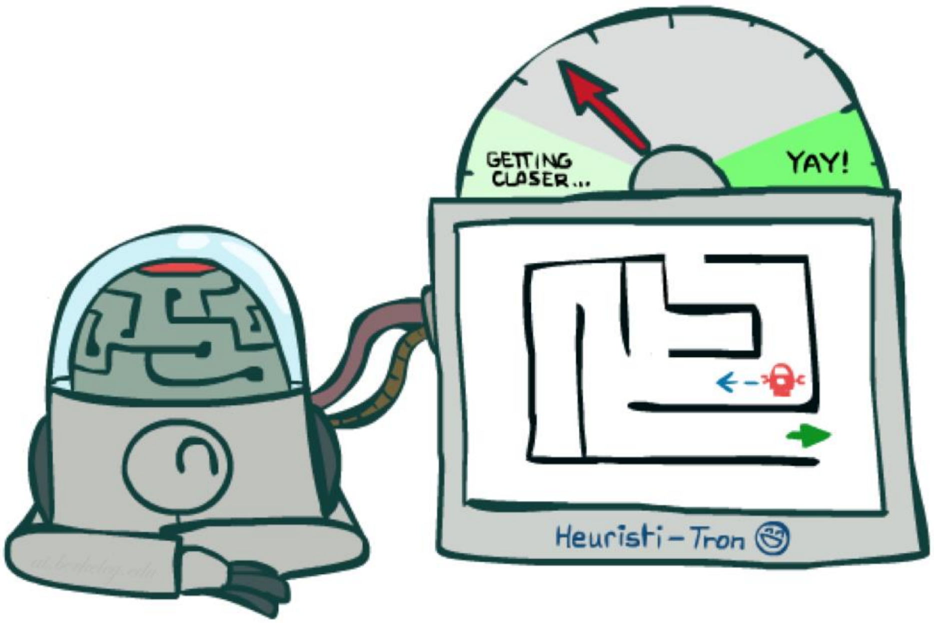


Is A* Optimal?

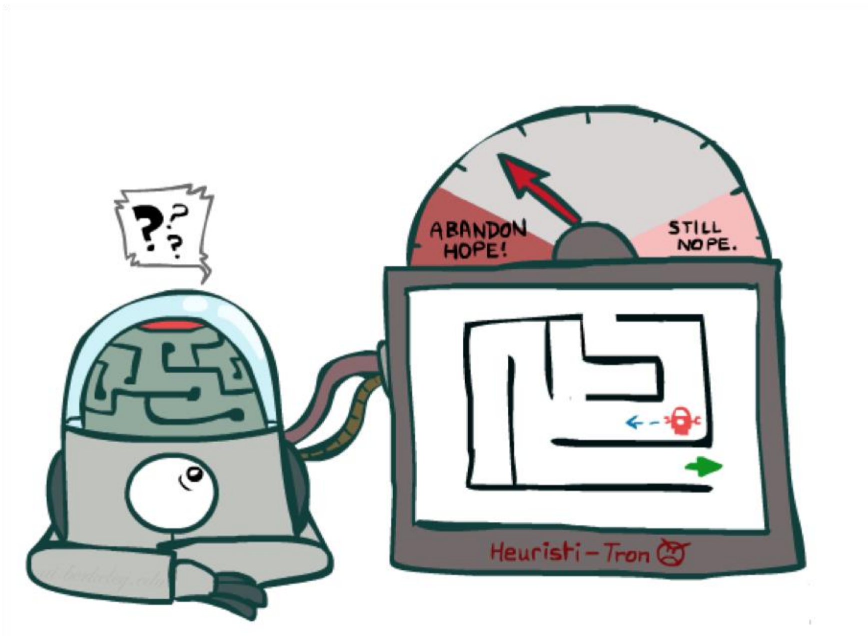


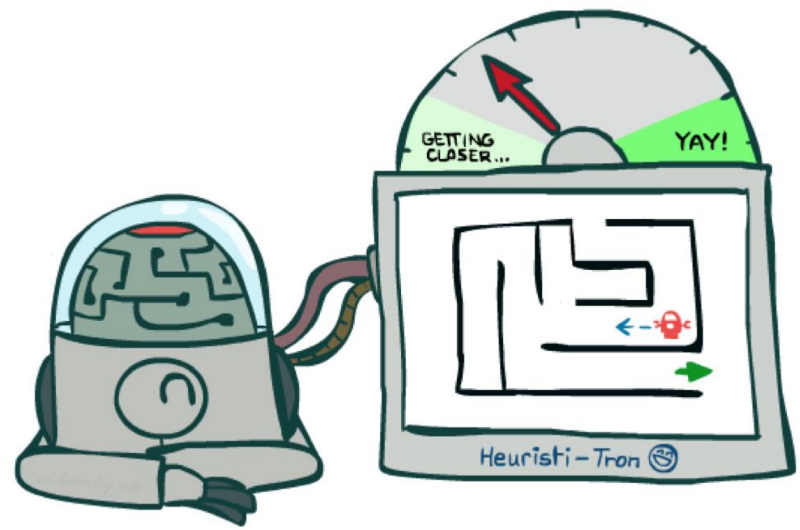
- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Admissible Heuristics



Idea: Admissibility





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

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

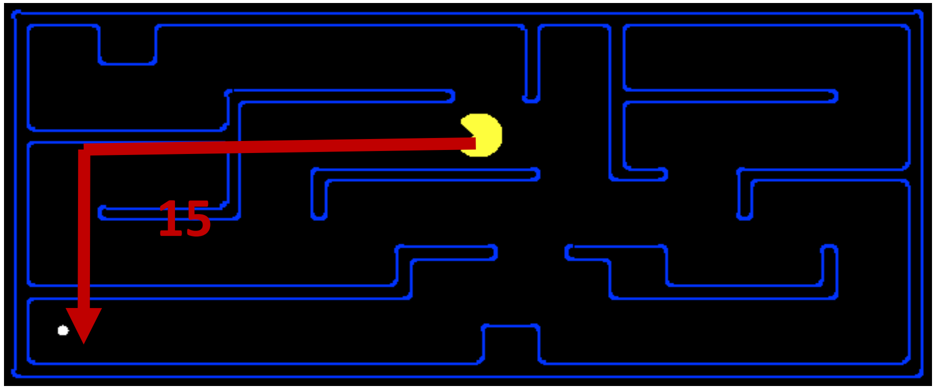
Admissible Heuristics

- 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

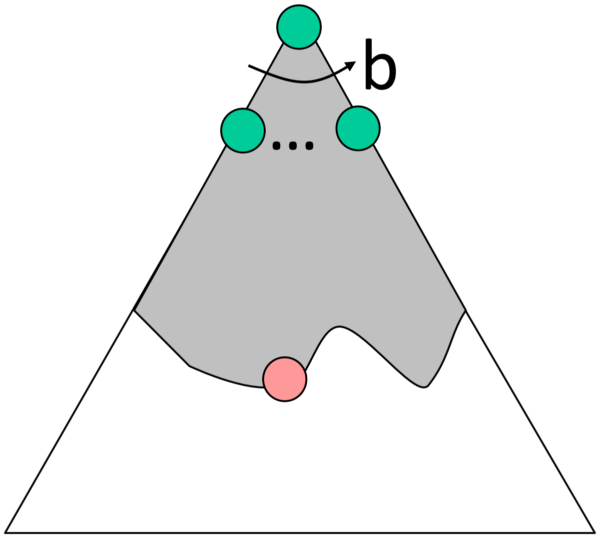
- Examples:



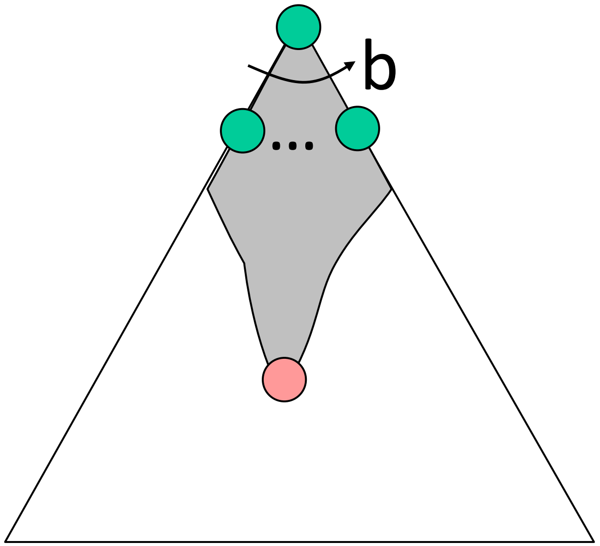
- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Properties of A*

Uniform-Cost

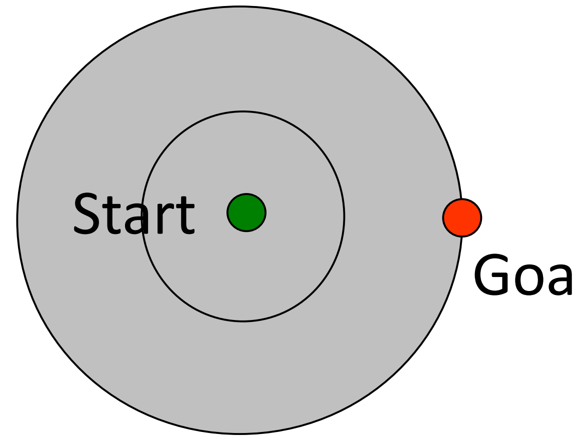


A*

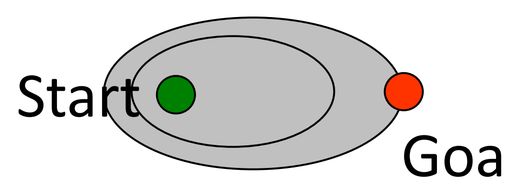


UCS vs A* Contours

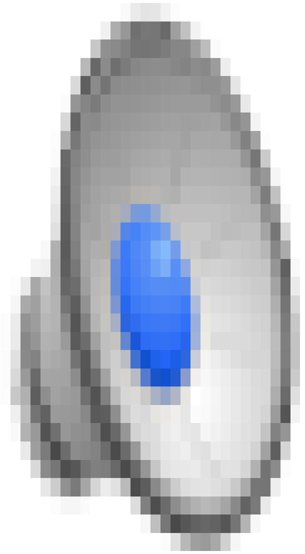
- Uniform-cost expands equally in all “directions”



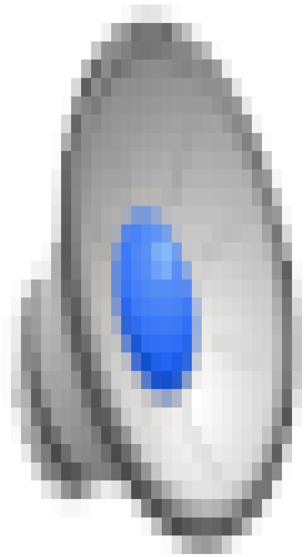
- A* expands mainly towards the goal, but does hedge its bets to ensure optimality



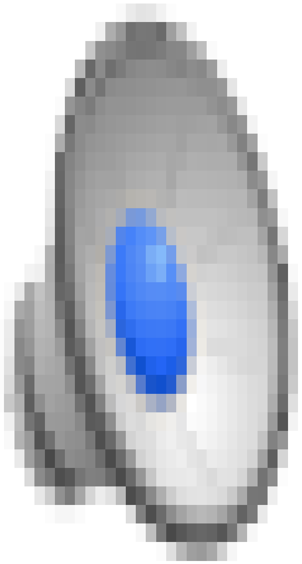
Video of Demo Contours (Empty) -- UCS



Video of Demo Contours (Empty) -- Greedy

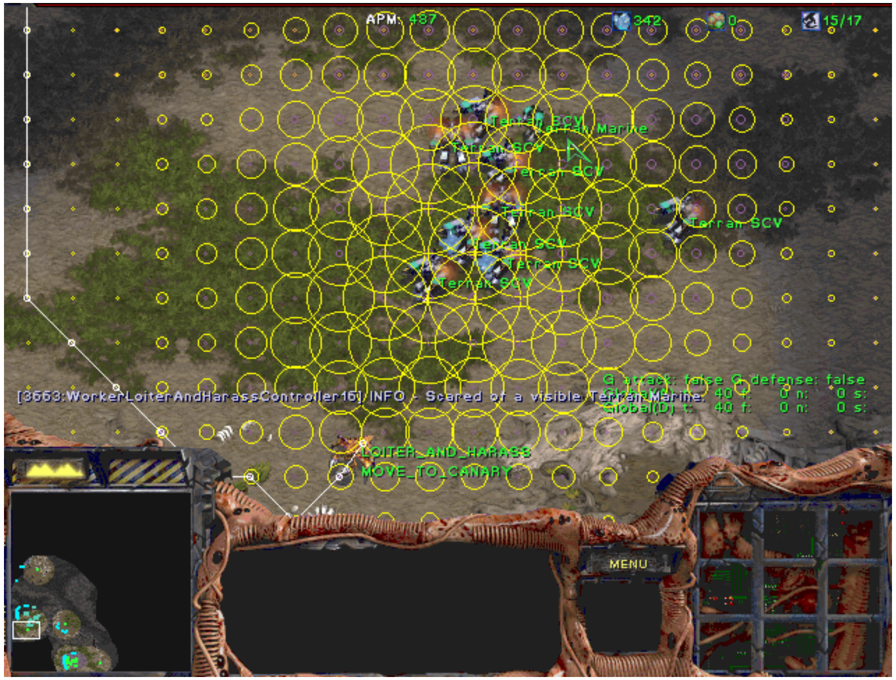


Video of Demo Contours (Empty) – A*

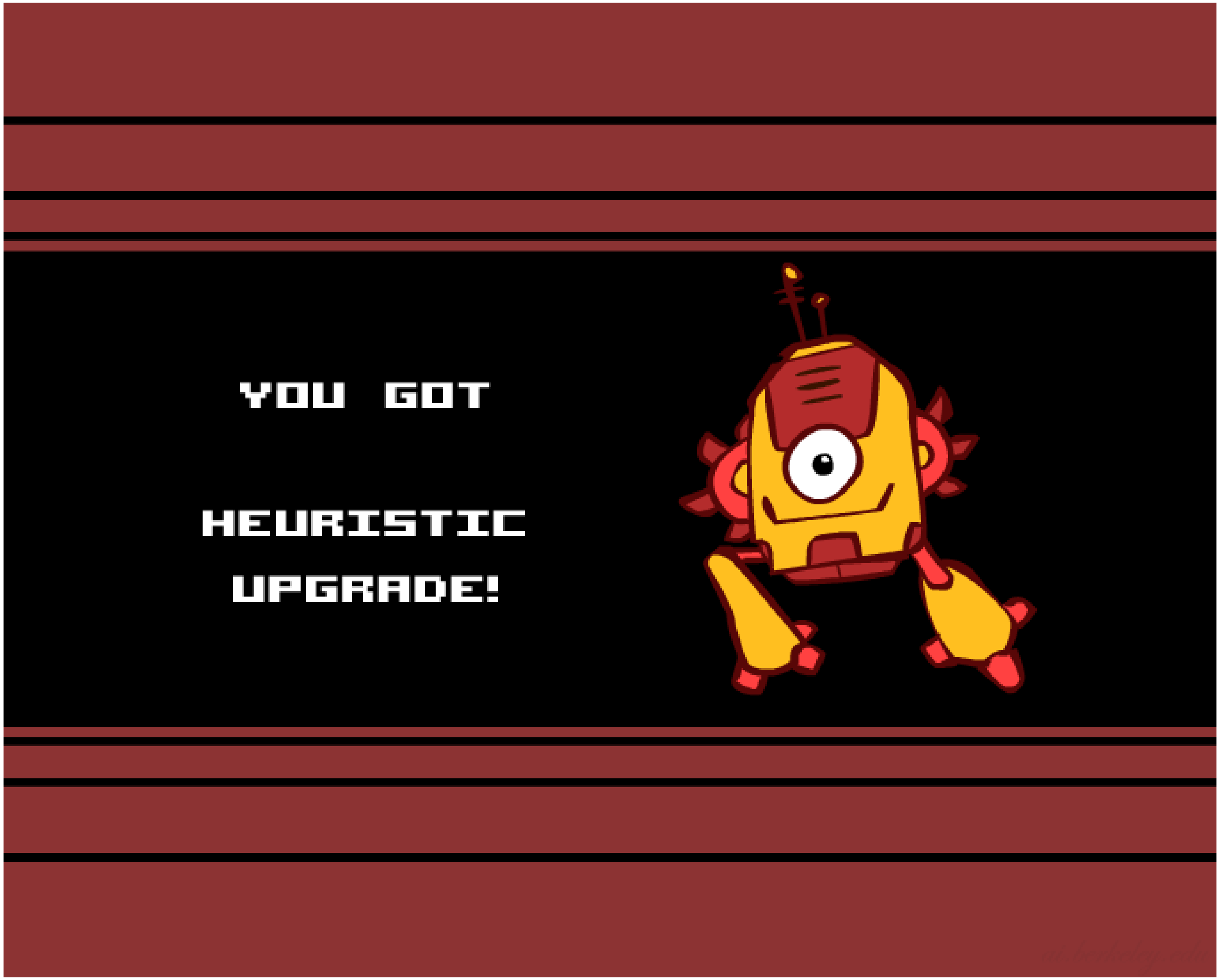


A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …



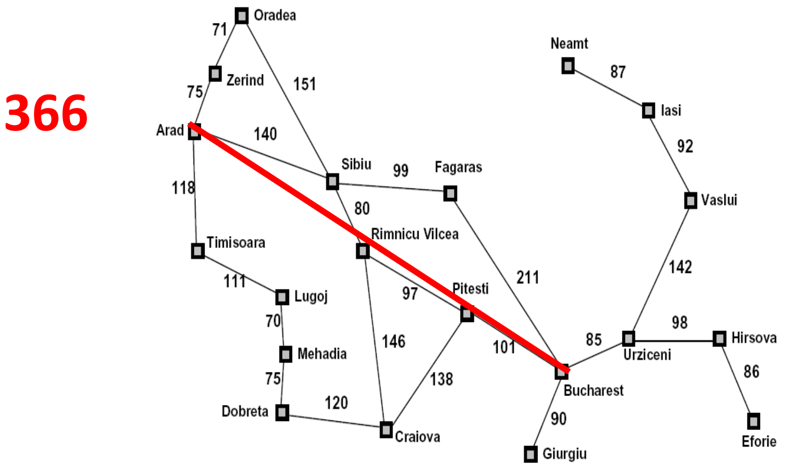
Creating Heuristics

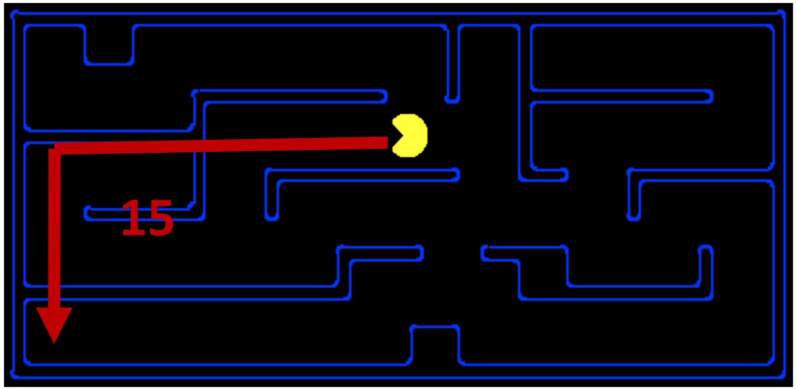


Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

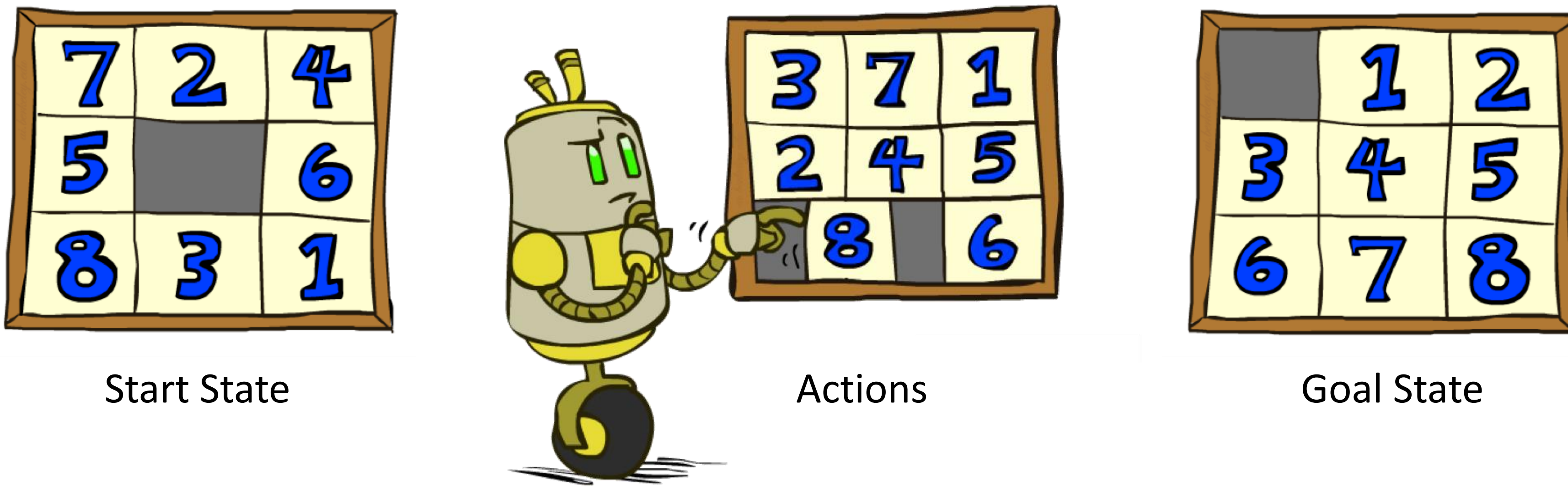
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available





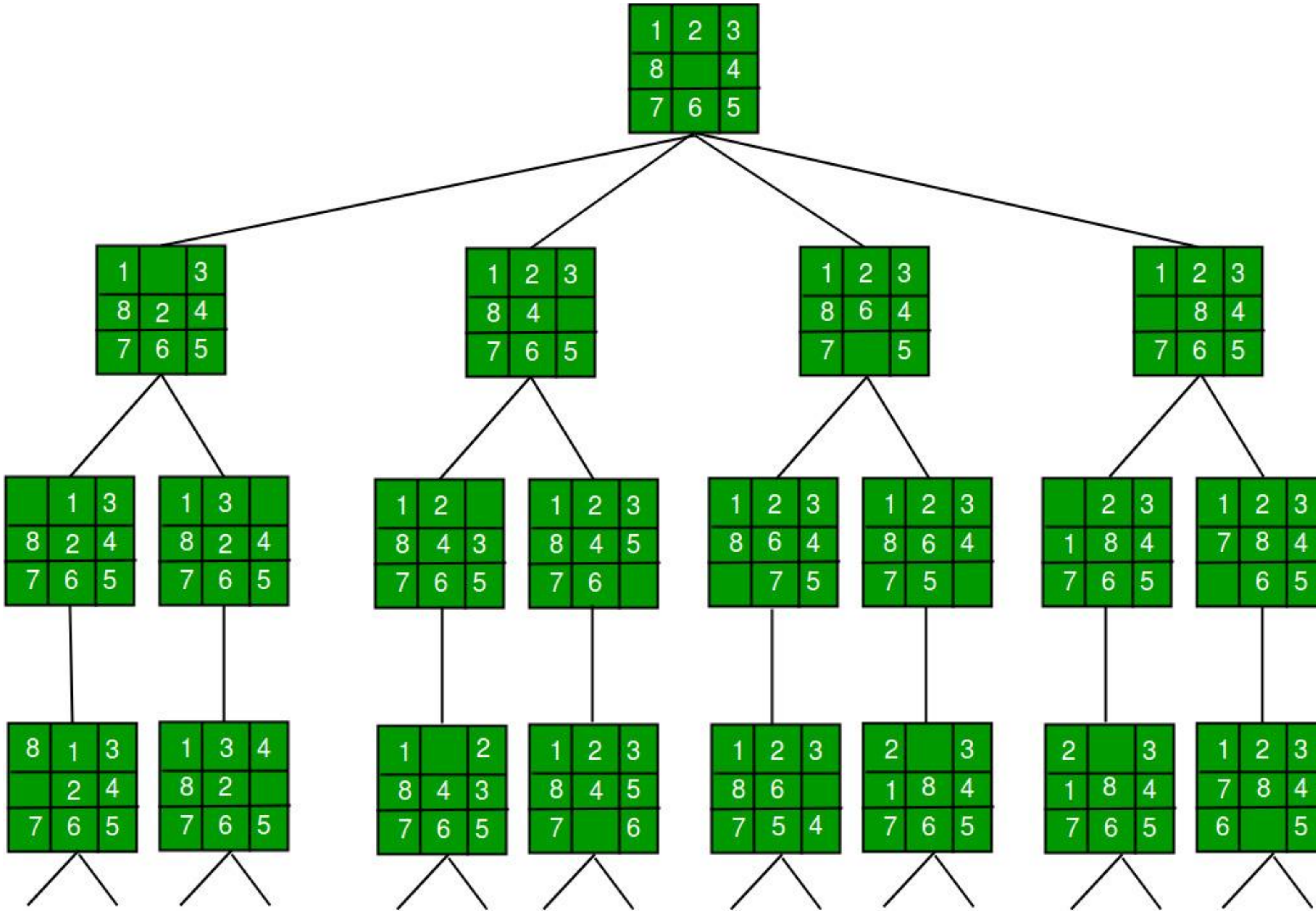
- Inadmissible heuristics are often useful too

Example: 8 Puzzle



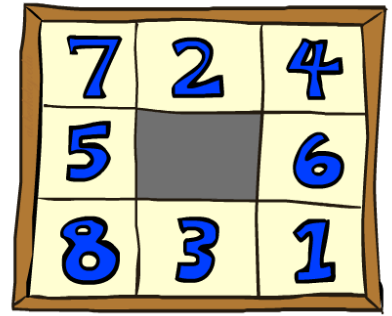
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

State-space tree

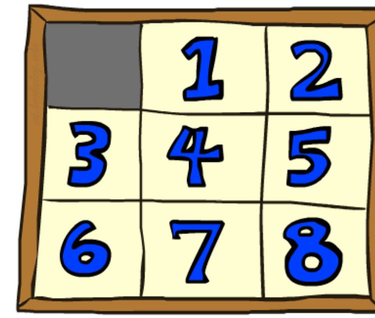


8 Puzzle I

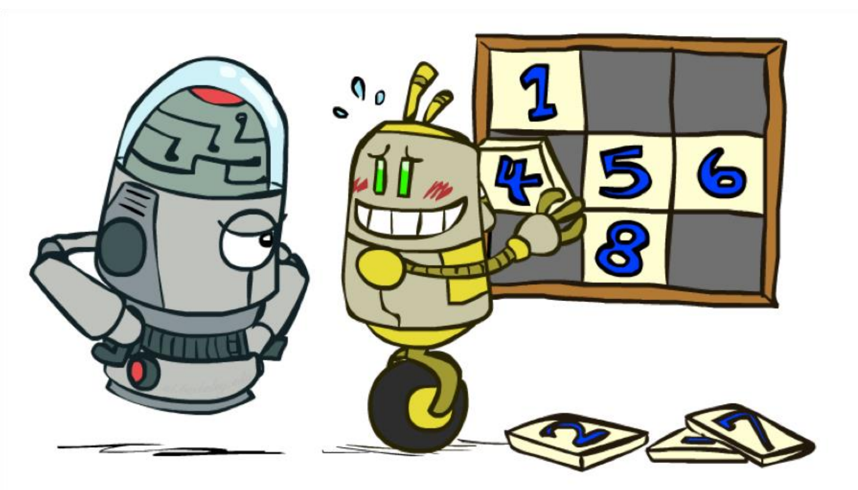
- Heuristic: Number of **tiles** misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



Goal State



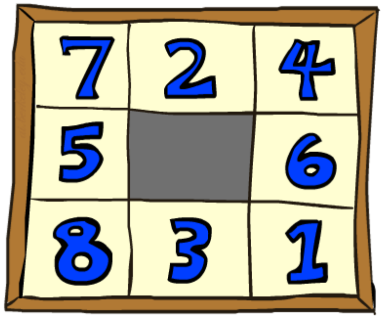
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

Statistics from Andrew Moore

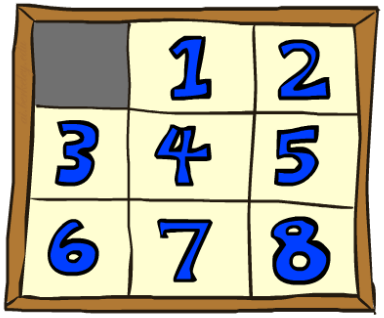
8 Puzzle II

- 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 (tile 1) + 1 (tile 2) + 2 (tile 3) + …

  = 18



Start State

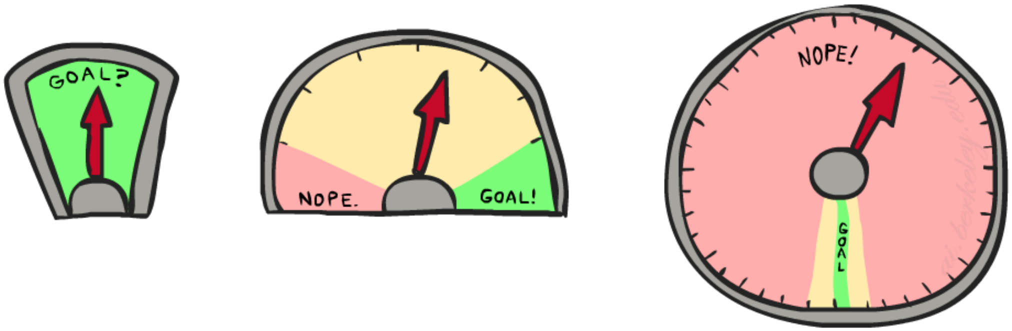


Goal State

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

8 Puzzle III

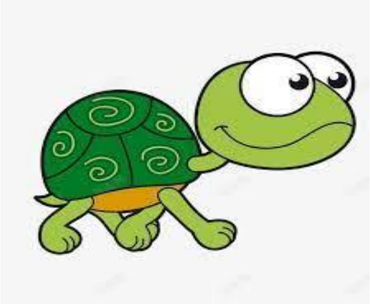
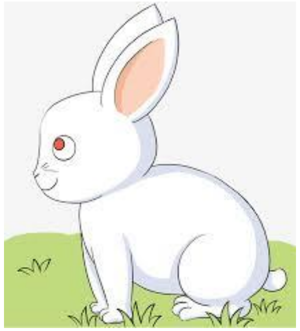
- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- 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

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

 +  = 

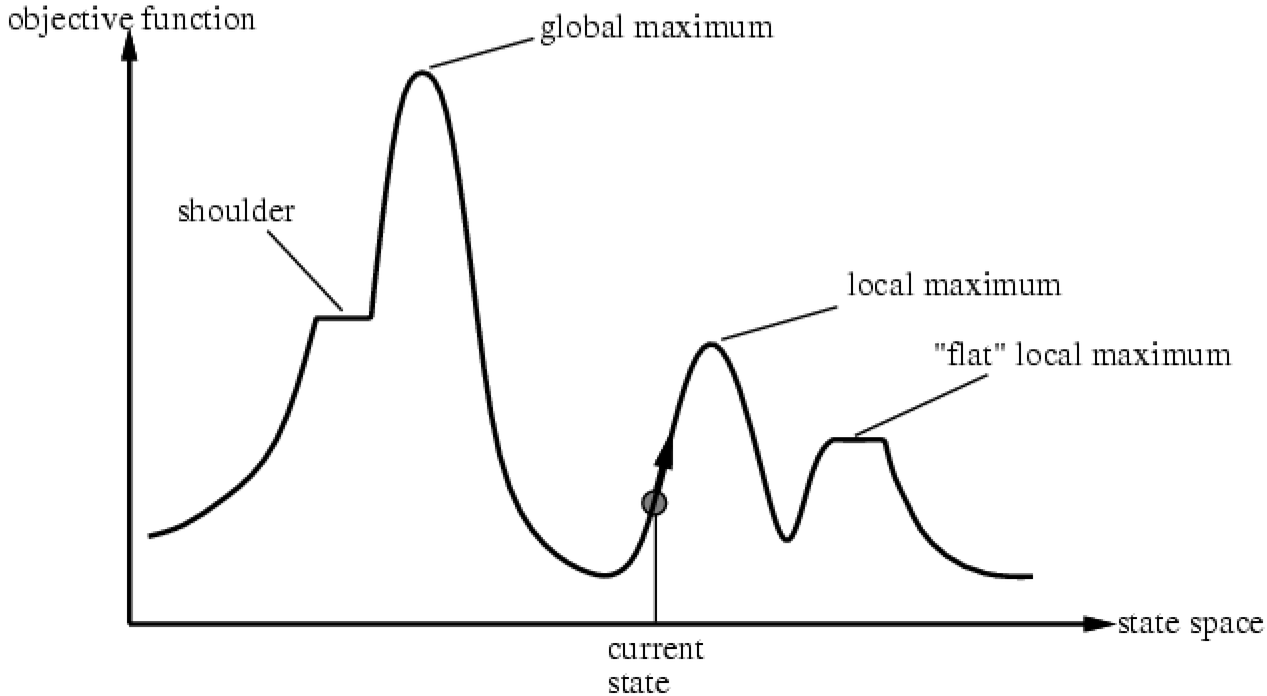
2. Local Search

- Objective (Fitness) Function *f(x)*
  - Local search problems have an objective function to specify how “good” a state is
- Strategy
  - Keep only a single (complete) state in memory
  - Generate only the neighbours of that state
  - Keep one of the neighbours and discard others
- Two key advantages
  - Very little memory required
  - Often find reasonable solutions in large or infinite state spaces
- Usage
  - Pure optimization problem
  - Find the best state according to an objective function

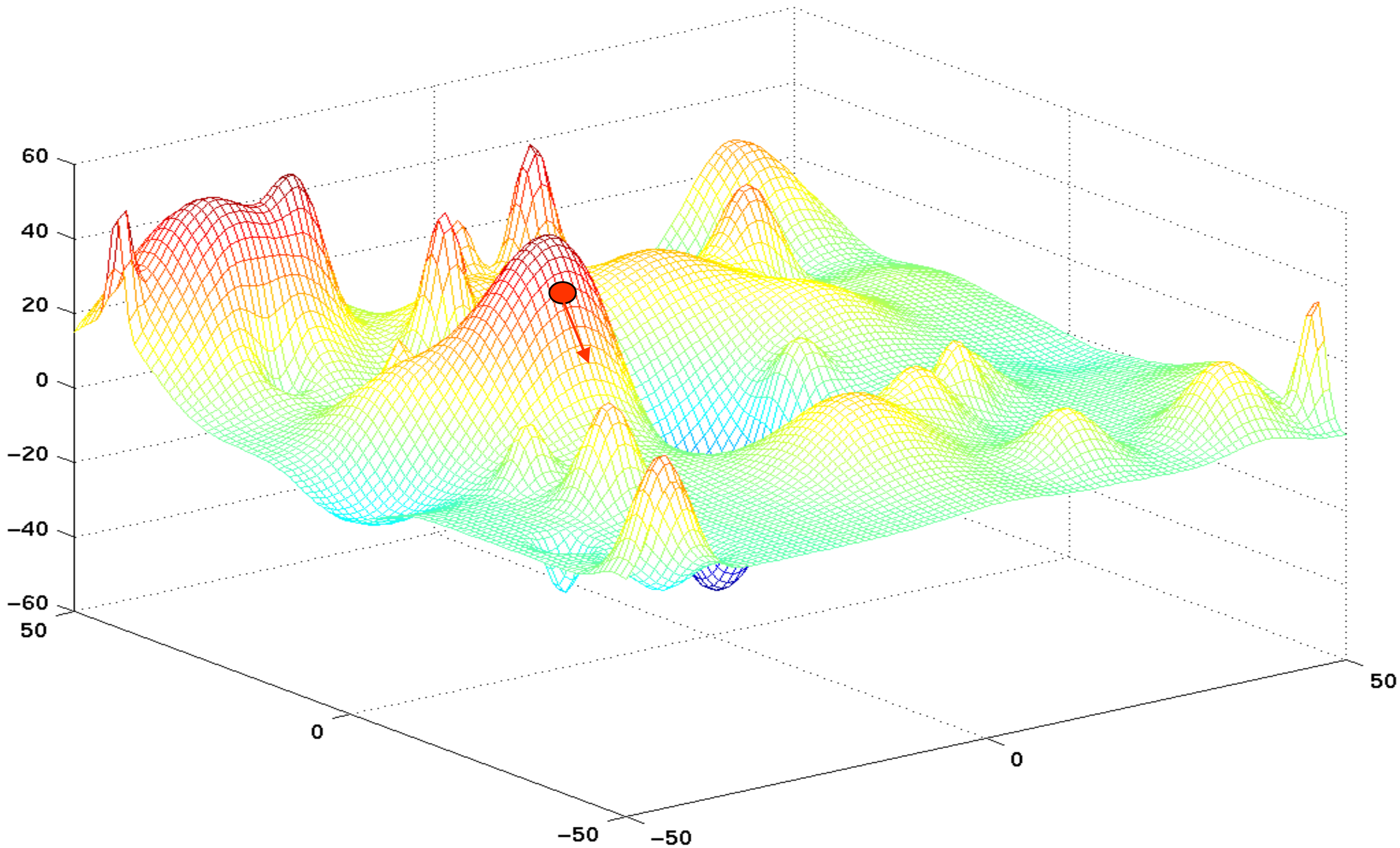
The following slides are from https://www.cs.odu.edu/~vashok/cs480.html

State Space Landscape

- Global minimum
  - The lowest valley
- Global maximum
  - The highest peak



2-D State Space



Hill Climbing Algorithm

- Moves
  - Only permit to move to neighbors that improve $f(x_{now})$
- Choice
  - Choose $x_{next} \in \text{Neighbor}(x_{now})$ that $f(x_{now}) < f(x_{next})$
- Termination
  - If no x_{next} can be found

Hill-Climbing

- continually moves uphill
  - increasing value of the evaluation function
  - gradient descent search is a variation that moves downhill
- very simple strategy with low space requirements
  - stores only the state and its evaluation, no search tree
- problems
  - local maxima
    - the peak is higher than all its neighbors, but not the global maximum
    - algorithm can’t go higher, but is not at a satisfactory solution
  - plateau
    - area where the evaluation function is flat
  - ridges
    - search may oscillate slowly
    - almost like a plateau

Further Variants of Hill Climbing

- A problem
  - Success rate may be a little low

- Solutions
  - Stochastic hill-climbing:
    - Choose at random among uphill moves
  - First-choice hill-climbing:
    - Generate neighbourhood in random order
    - Move to first generated that represents an uphill move

Thanks