Artificial Intelligence: Search Controls

CPSC 433: Artificial Intelligence
Fall 2022

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Thursday, October 13, 2022

UNIVERSITY OF CALGARY

Search Controls

General tasks:

- Determining all possible transitions, i.e.
  $\{(s_1,s_2) \in T \mid s_1 \text{ is actual state}\}$
- Selecting the next state

Important observation:

Transitions are usually based on applying general rules to parts of the actual state
  Examples:
  - extension rules in set-based search
  - processing a leaf in tree- or graph-based search

☞ use observation to make tasks easier and faster

Determining all possible transitions

Many general rules that were applicable in the last state usually are applicable in the current one

Therefore

- Have list of potential transitions from last state
- Delete from list potential transitions not possible any more (parts of state used for them do not exist now)
- Update remaining transitions if necessary (remember: we are in a new state now)
- Add newly possible transitions (that are not already in the list)

☞List of all candidates for next transition

Selecting the next state

Have to find best transition

☞ evaluation necessary

- Store evaluation with transition so that evaluation can be reused (but not always reusable, remember min-max search)
- Organize list of transitions as heap, since always the transition with best evaluation is looked for
  - Finding best transition takes constant time
  - Inserting new transitions much faster than in ordered list

Evaluating transitions

Candidates for measuring

- Result state
- Parts of actual state enabling general rule for transition
- Parts new in the result state vs actual state

What to use?

Depends on how difficult it is to compute needed data

(i.e. resulting state resp. parts)

General Ideas for What to Measure

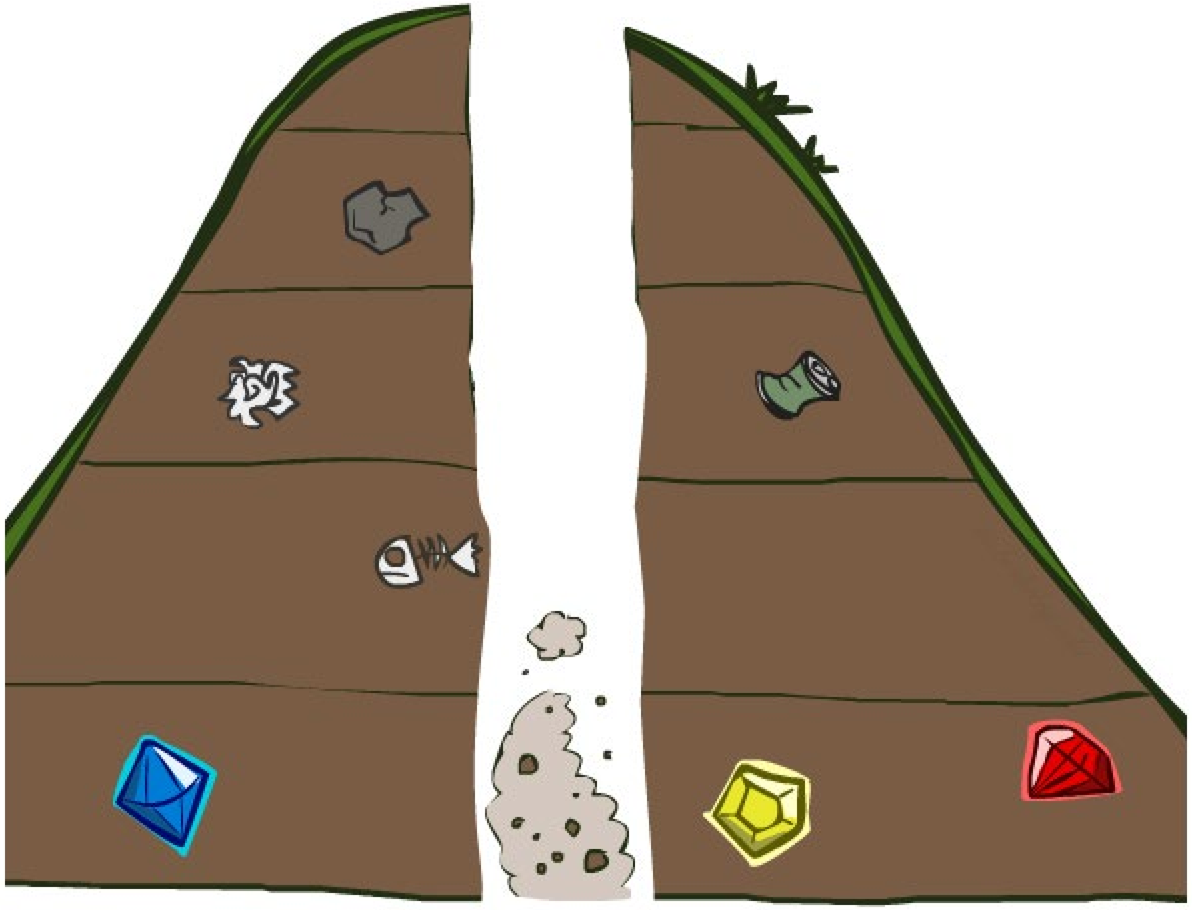
- Distance to a goal state or parts of it
- Best that can be achieved from a state (using an approximation, used for optimization problems)
- Difficulty of new problems in state (needs knowledge about problems)
- Number of transitions that become possible
- Size of state
- History of search
- Use of similar search experiences

General Problems (and solution approaches)

- States get too big
  (☞ local search, backtracking, forget history)
- Measuring states too time consuming
  (☞ abstract to significant parts, use less complex measures)
- Combining pieces of knowledge
  (☞ normalizing weights + weighted sums)
- Contradicting control knowledge
  (☞ distributed search approaches, competition)

Simple Tree Search Controls

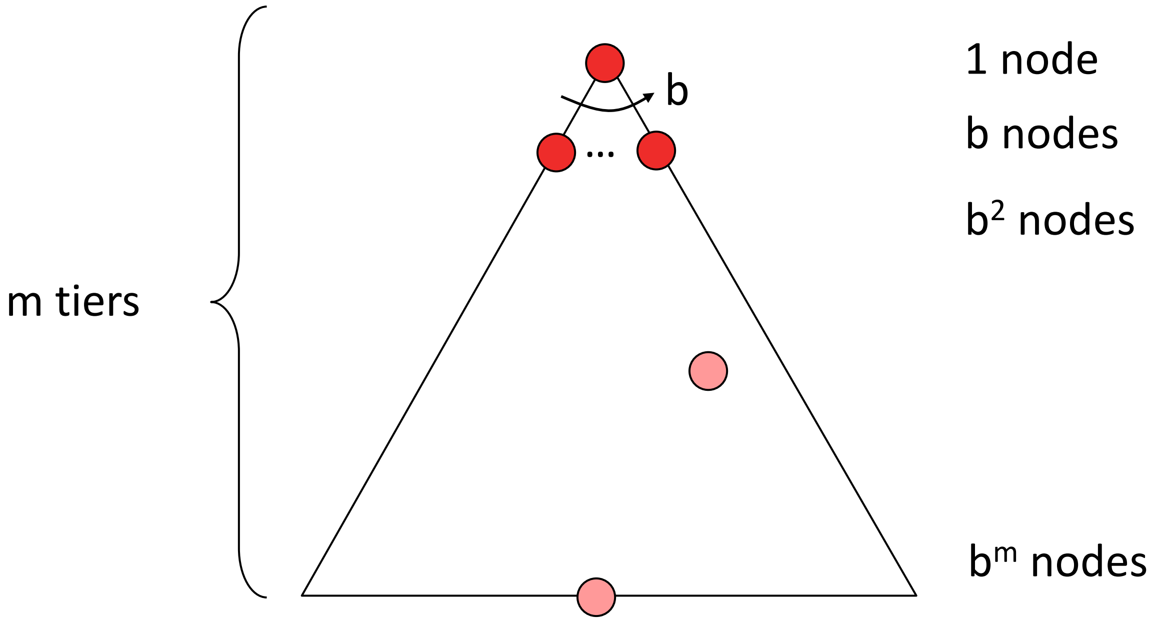
Search Algorithm Properties



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

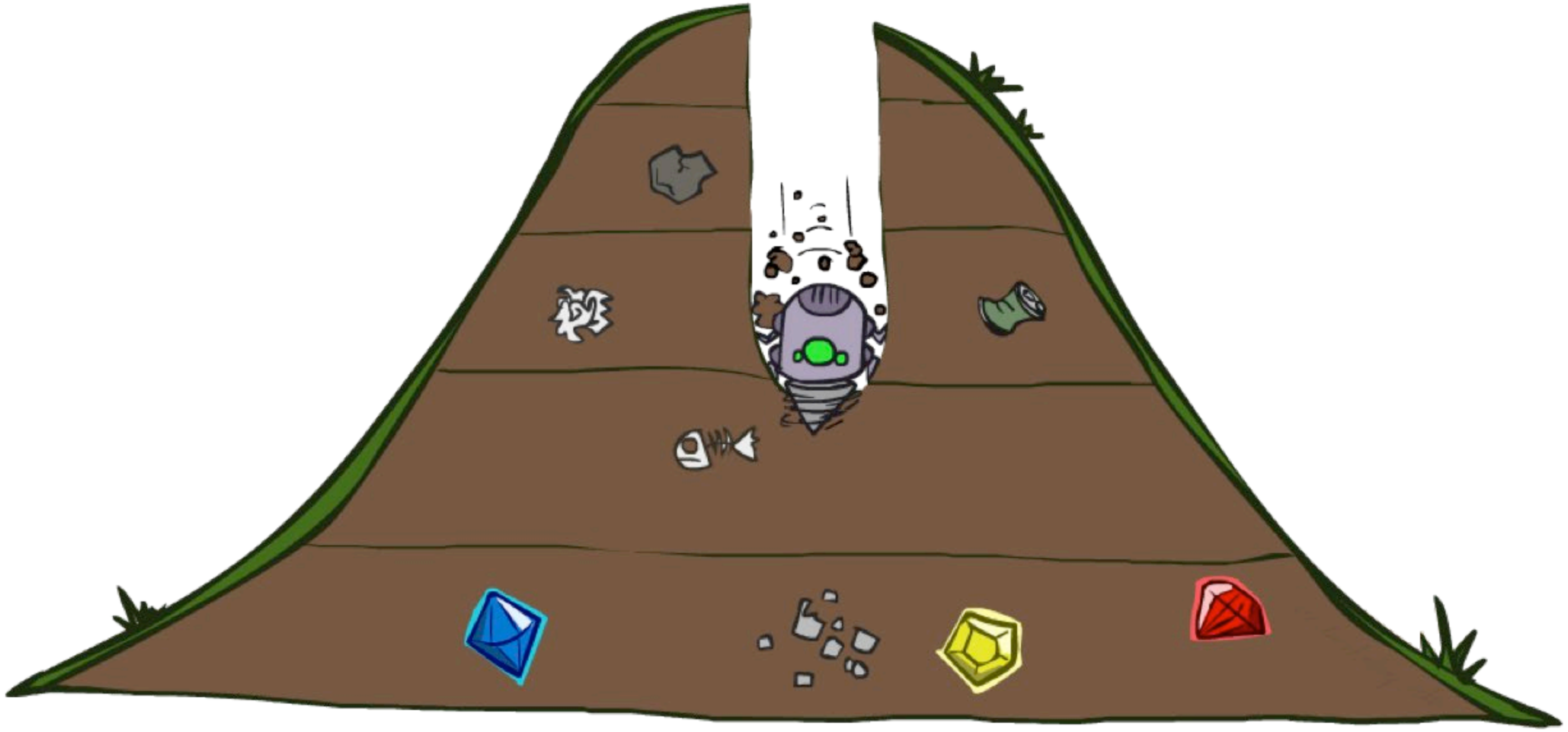
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots.\ b^m = O(b^m)$

DFS

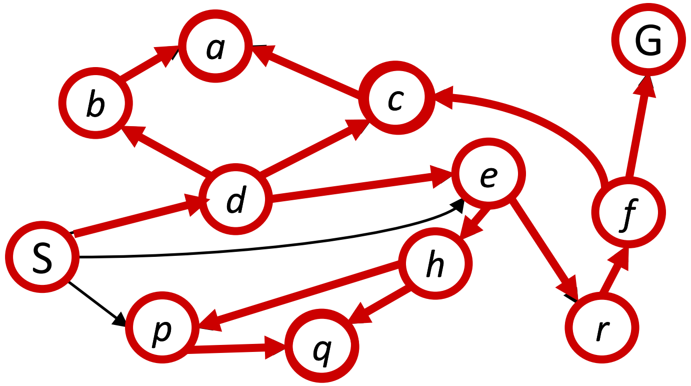
Depth-First Search

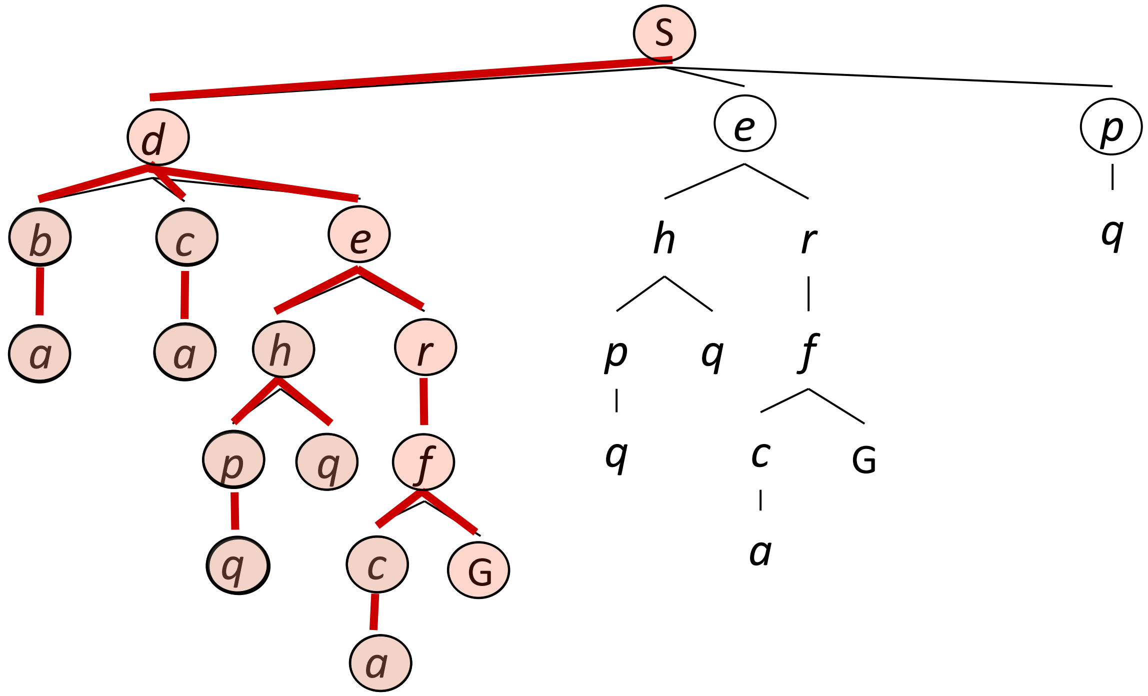


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





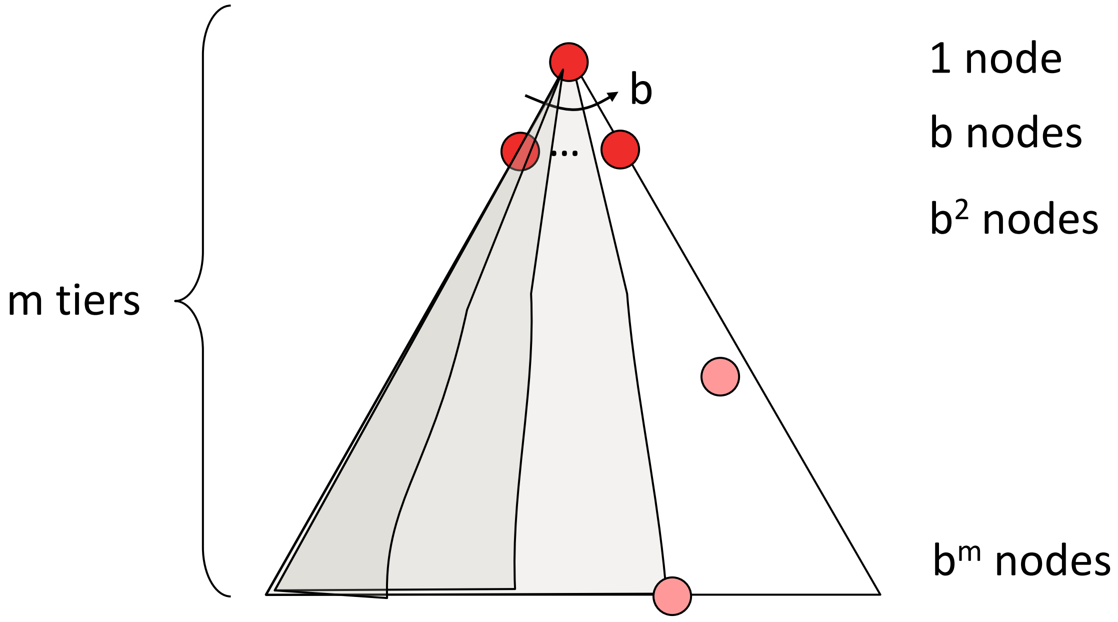
Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$

- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$

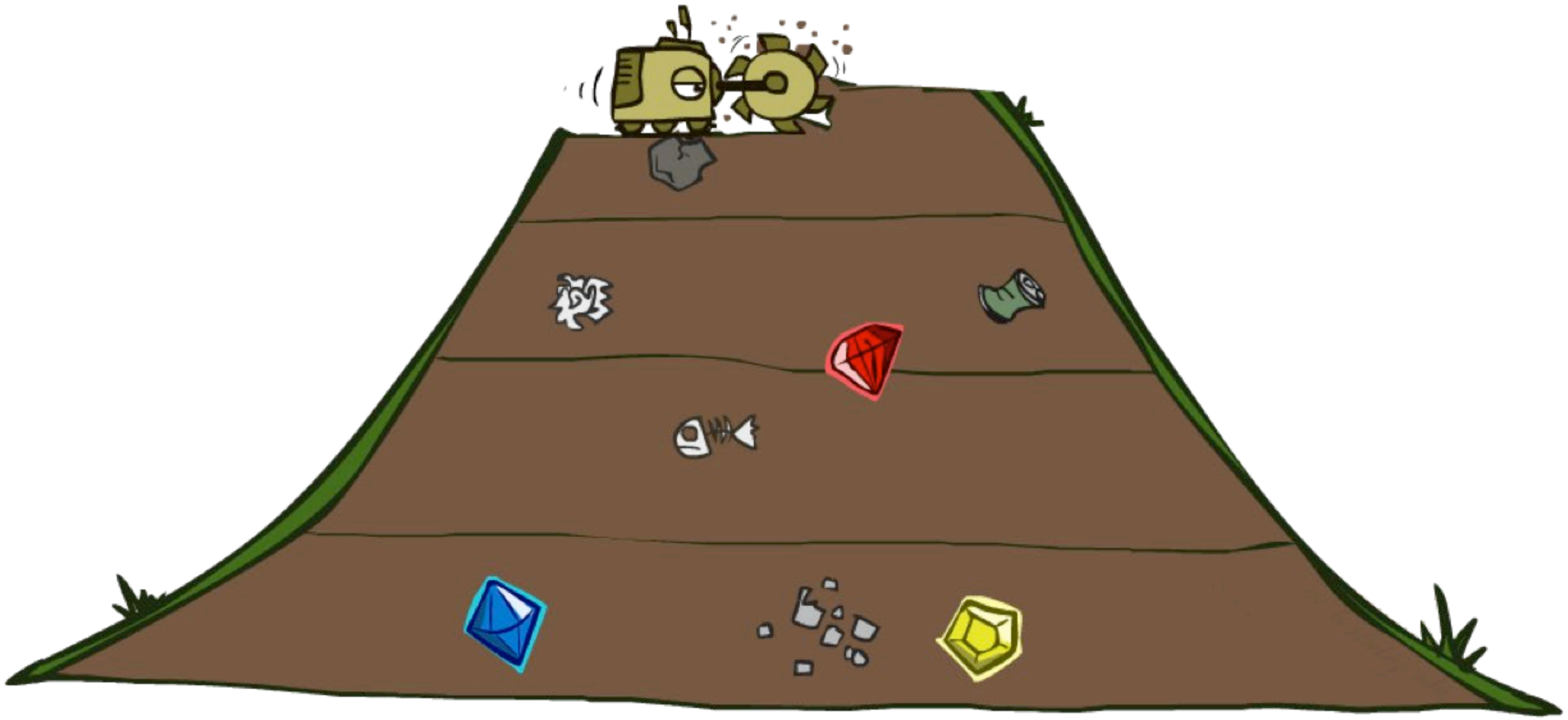
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)

- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



BFS

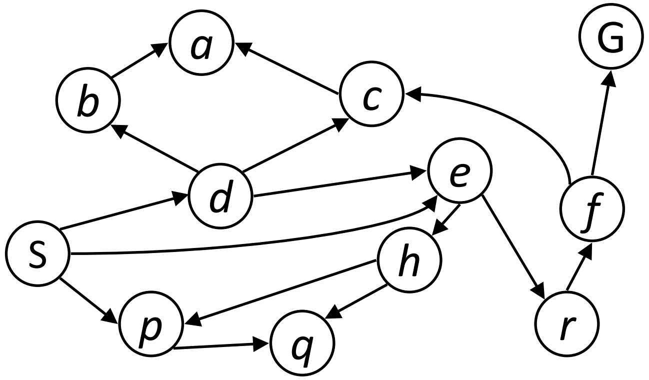
Breadth-First Search



Breadth-First Search

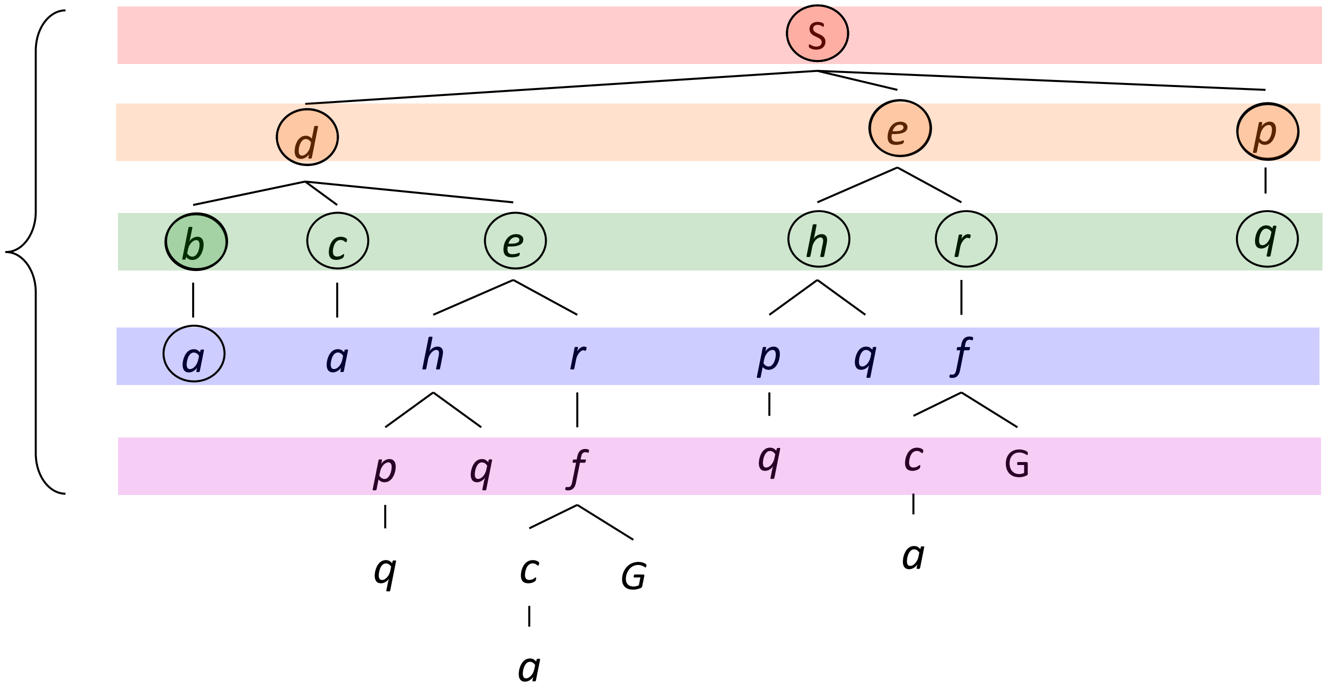
Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



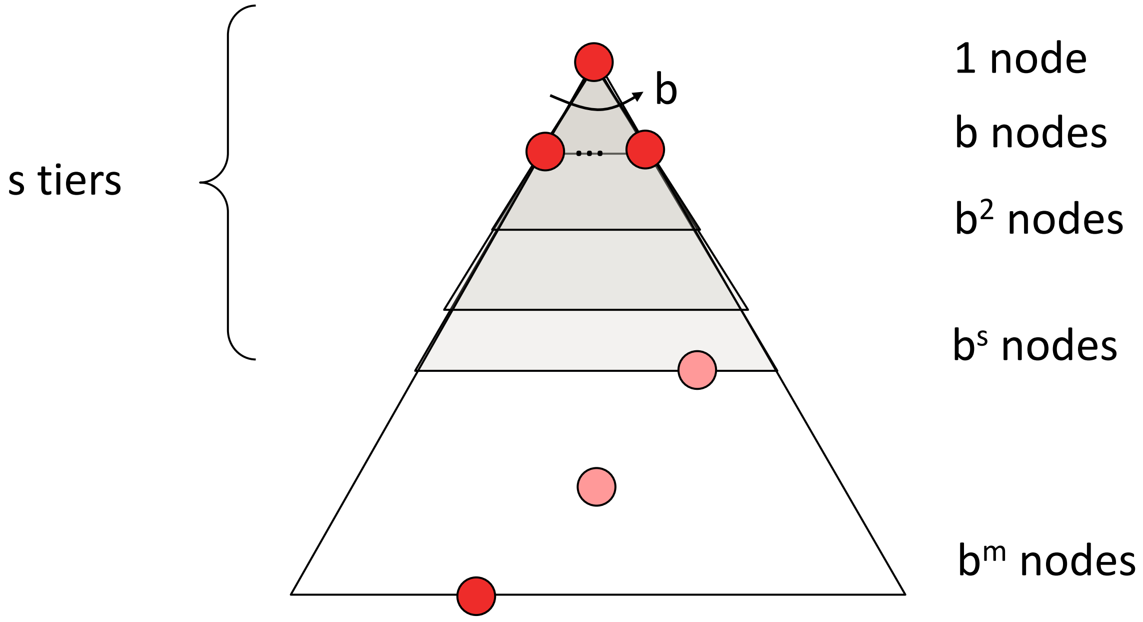
Search

Tiers

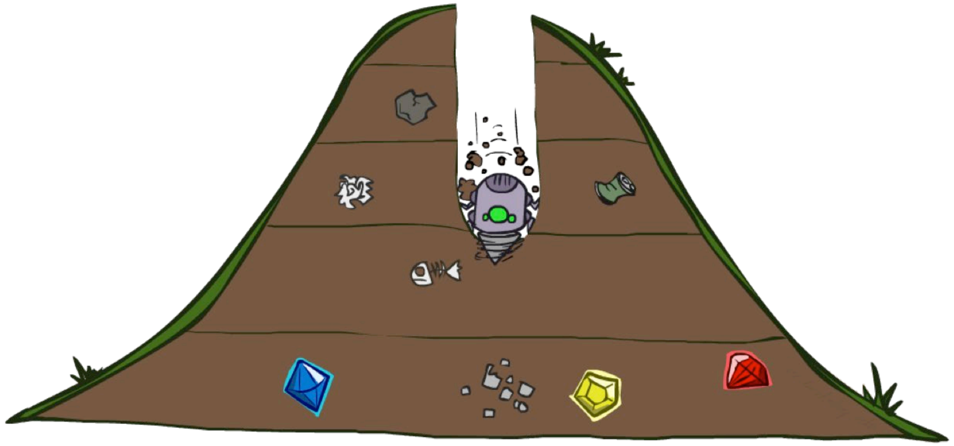


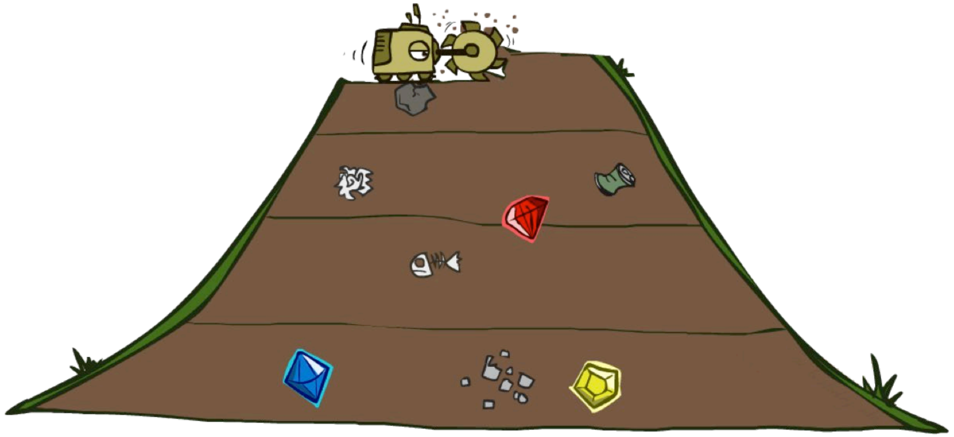
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



DFS vs BFS





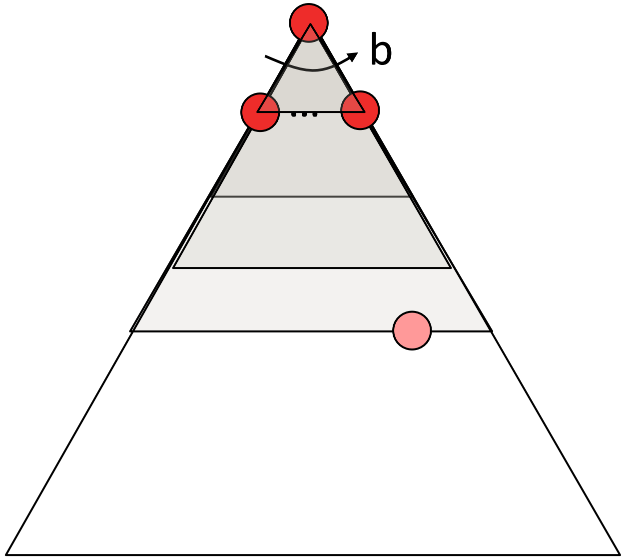
UNIVERSITY OF CALGARY

Iterative Deepening

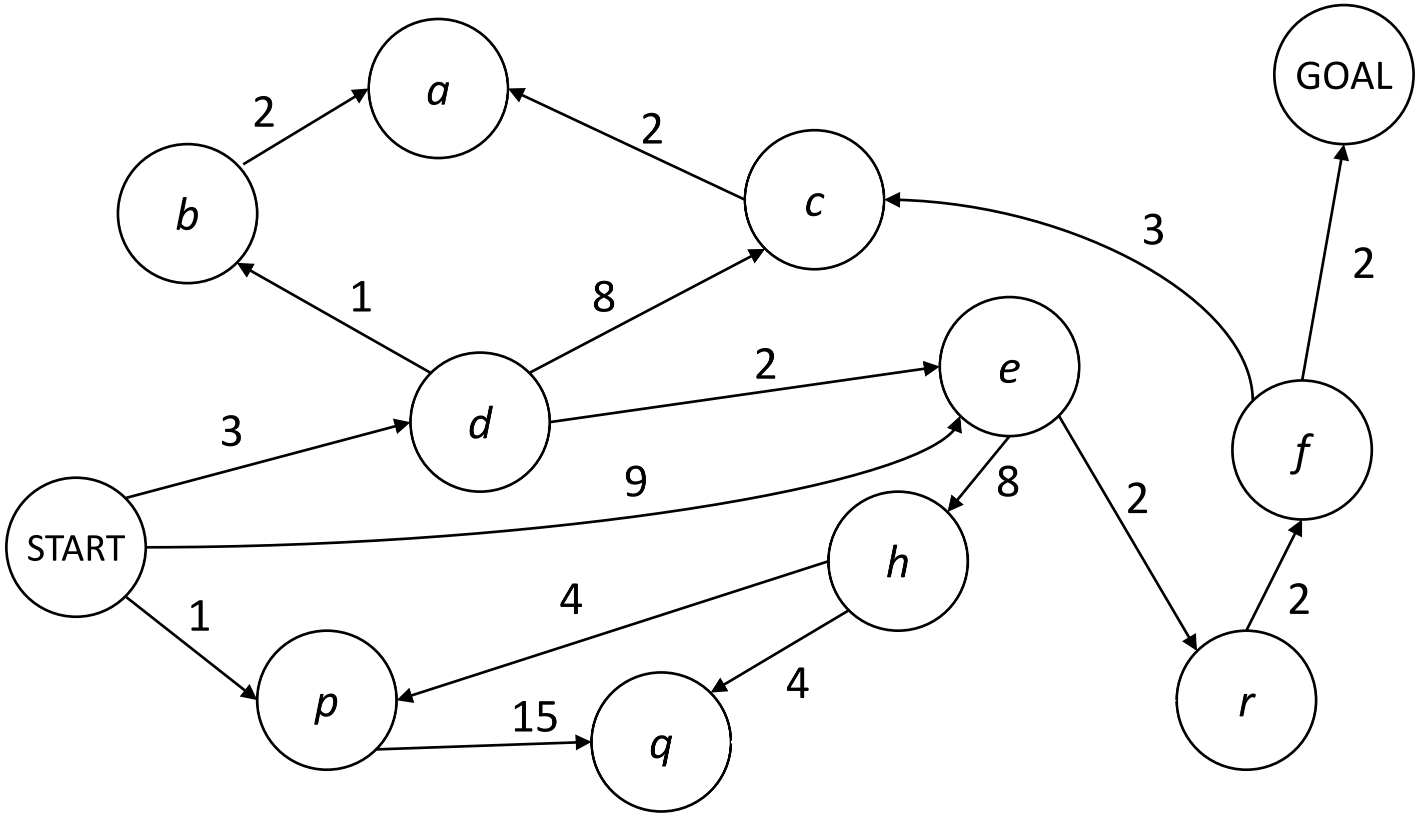
Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!



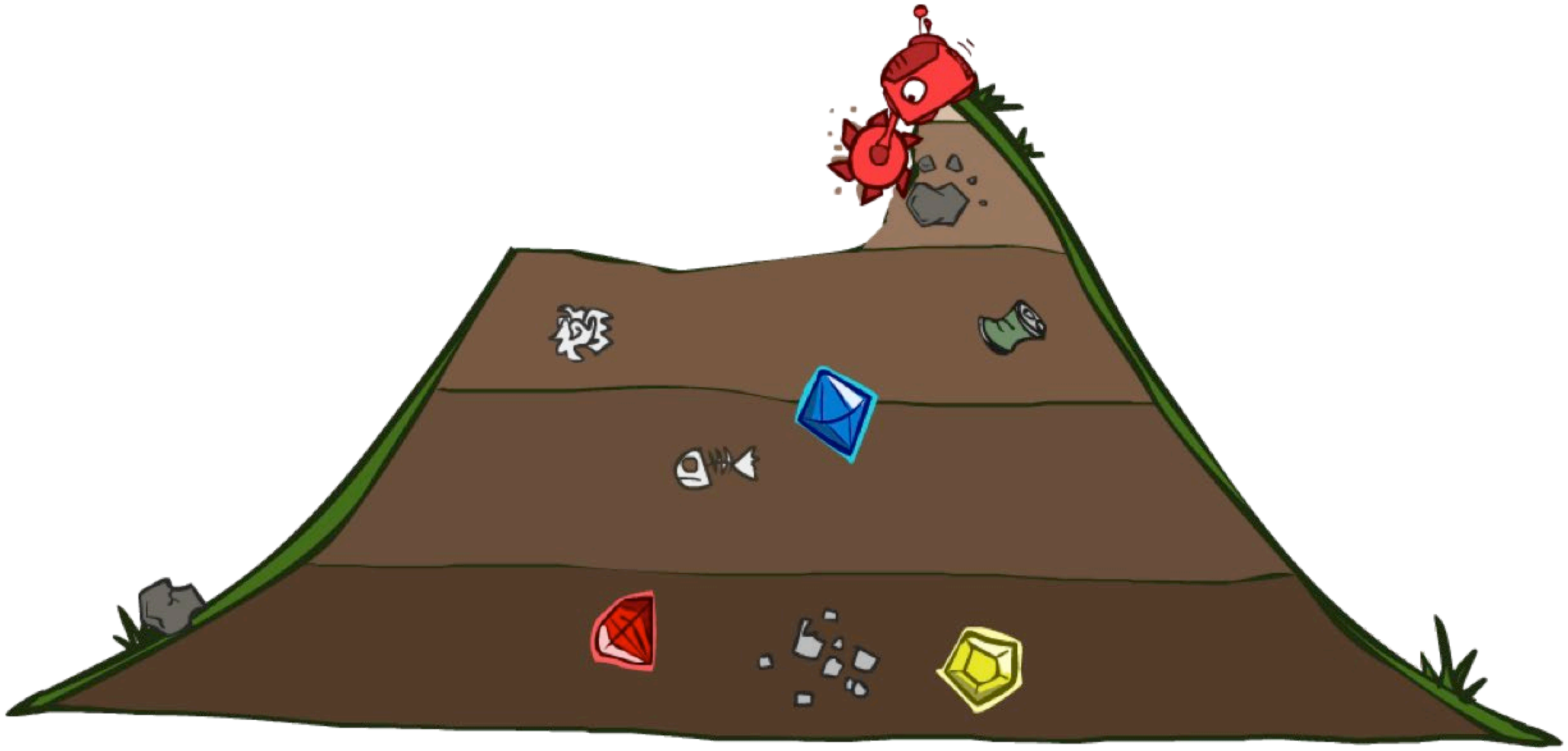
Cost-Sensitive Search



How?

Uniform Cost

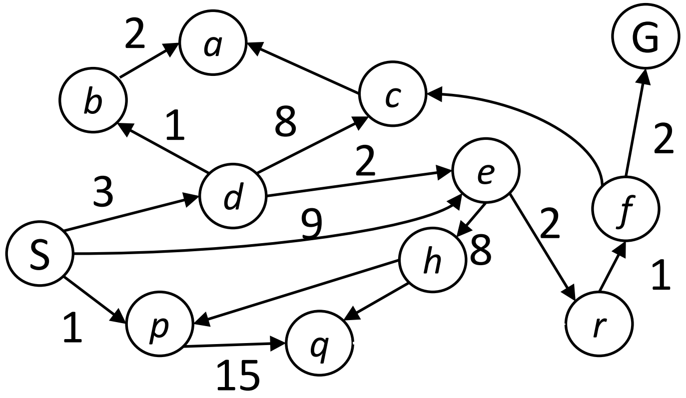
Uniform Cost Search



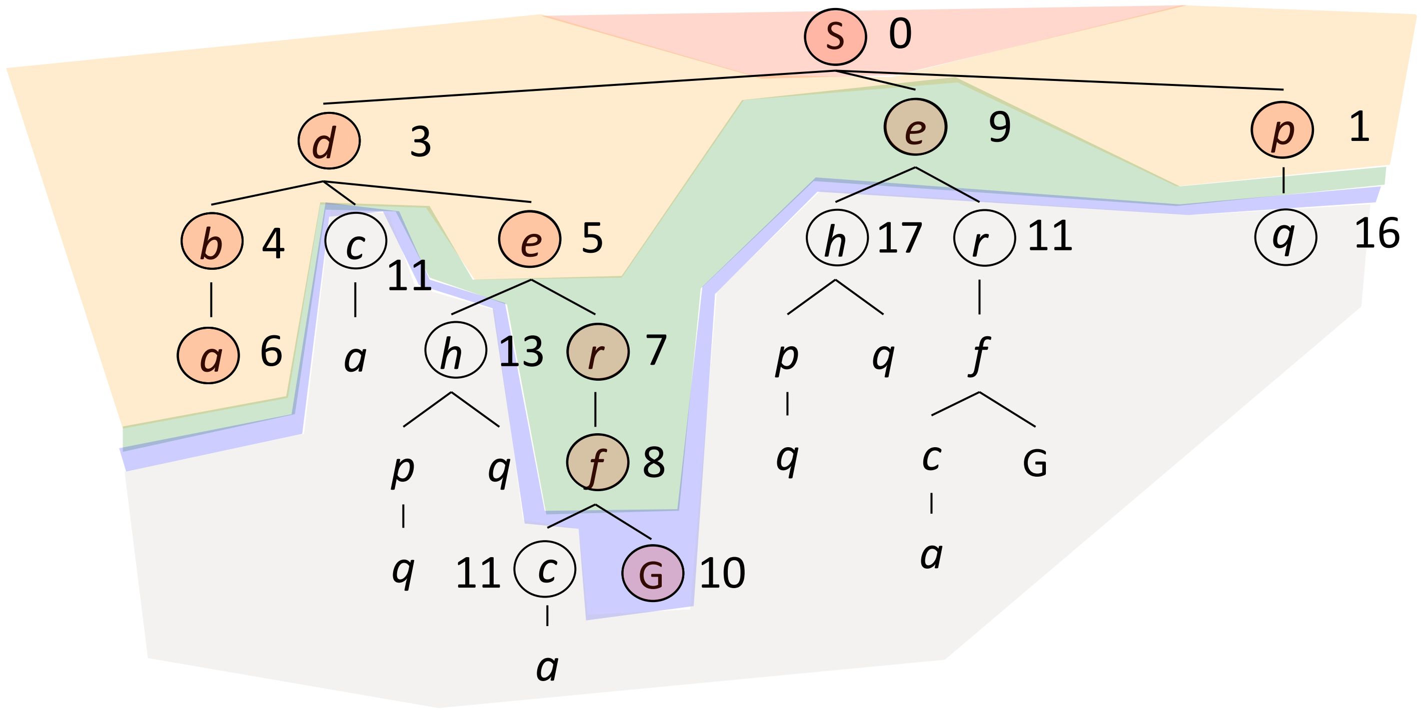
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



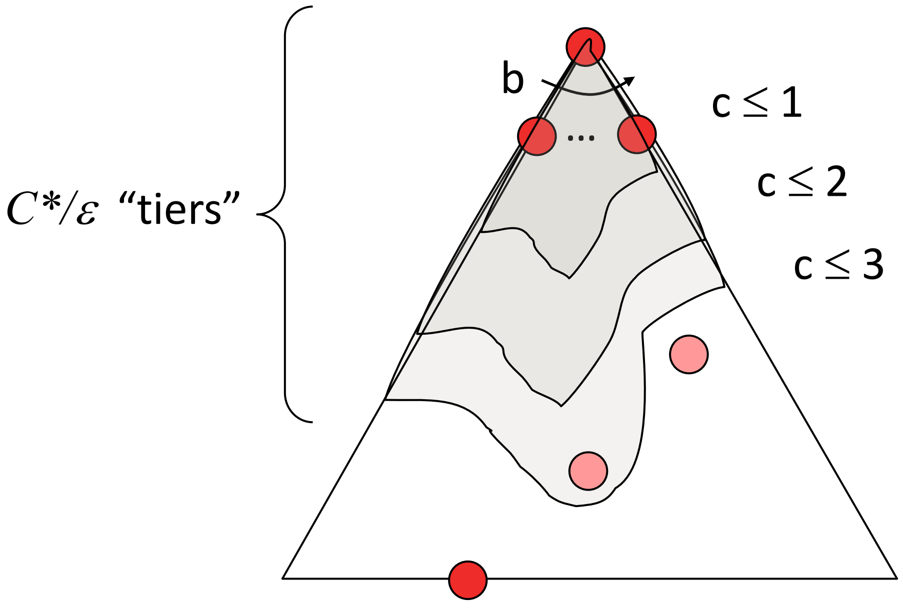
Cost contours



UNIVERSITY OF CALGARY

Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time O($b^{C^*/\varepsilon}$) (exponential in effective depth)



- How much space does the fringe take?
  - Has roughly the last tier, so O($b^{C^*/\varepsilon}$)

- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!

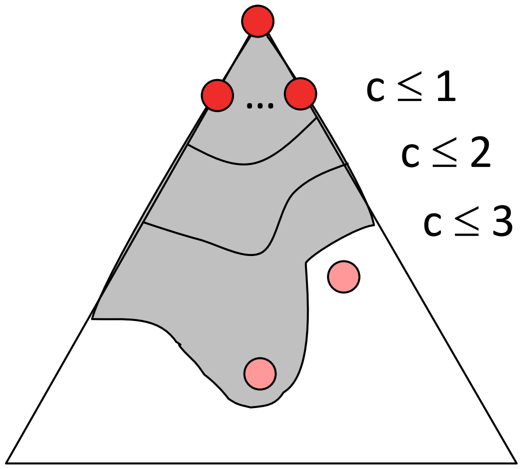
- Is it optimal?
  - Yes! (A*)

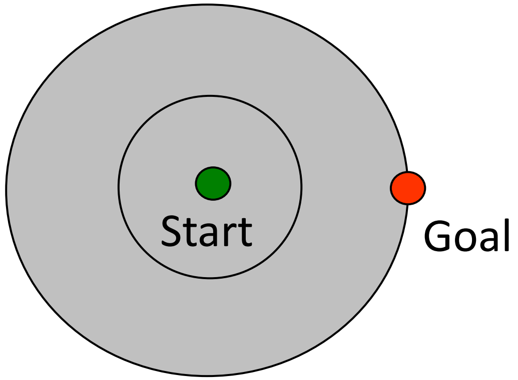
Uniform Cost Issues

- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location





UNIVERSITY OF CALGARY

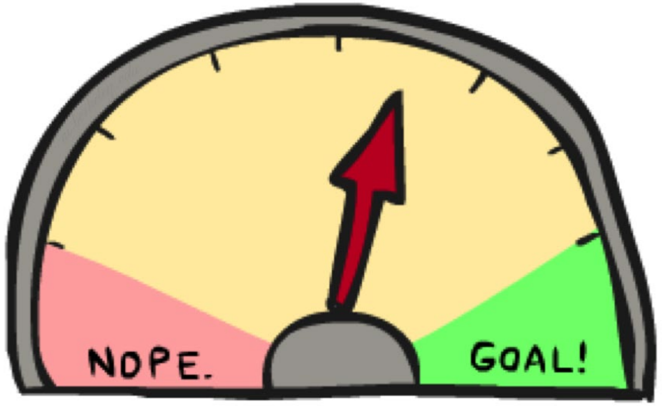
Informed Search

Informed Search

- Uninformed Search
  - DFS
  - BFS
  - UCS

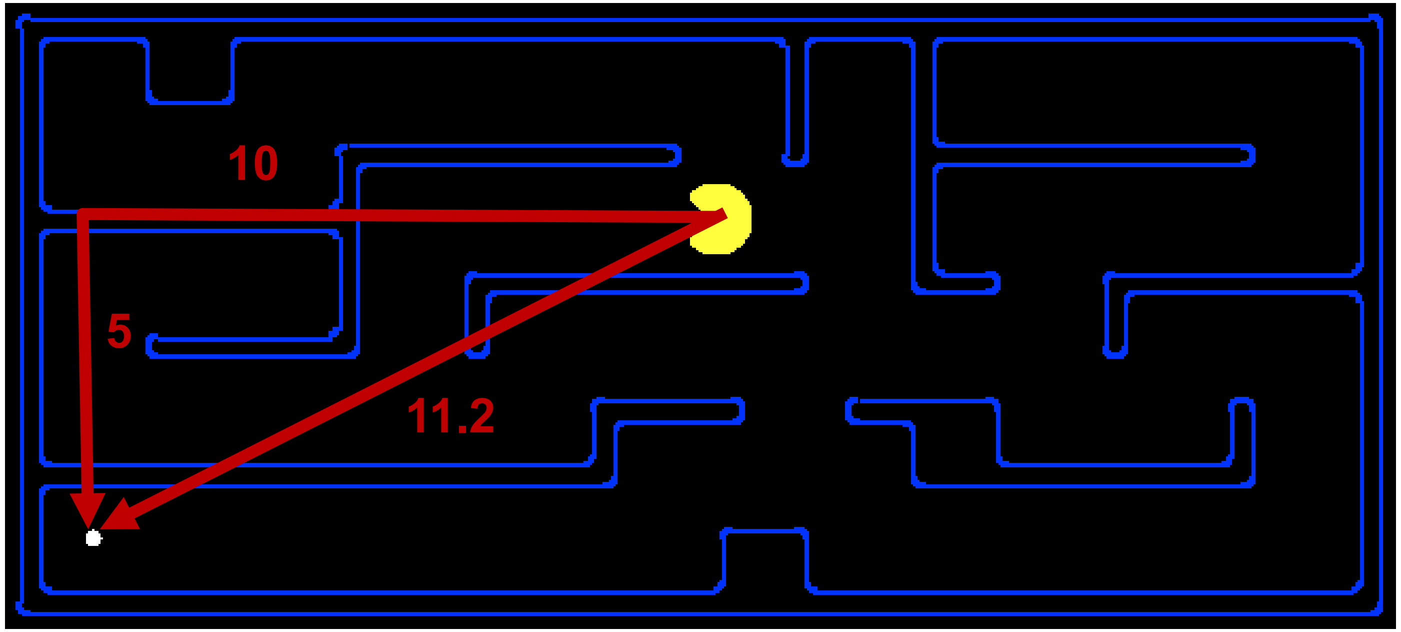
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search

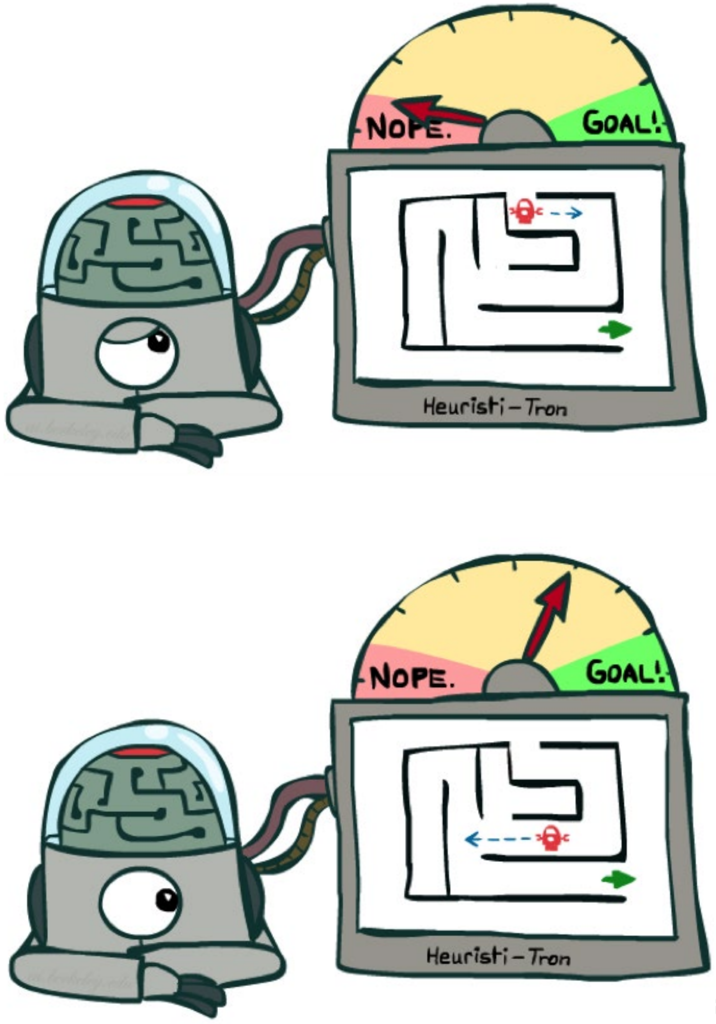
www.shutterstock.com · 149559782



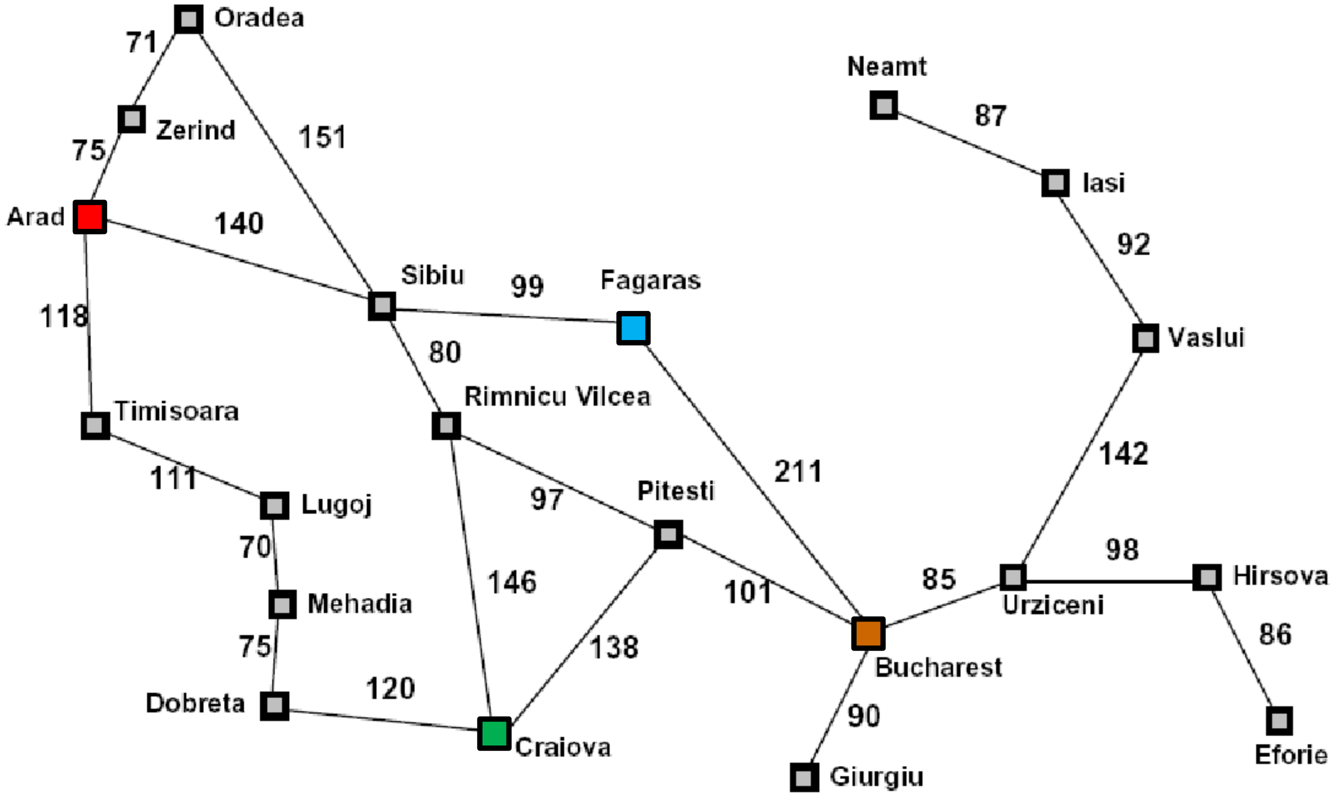
Search Heuristics

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





Example: Heuristic Function



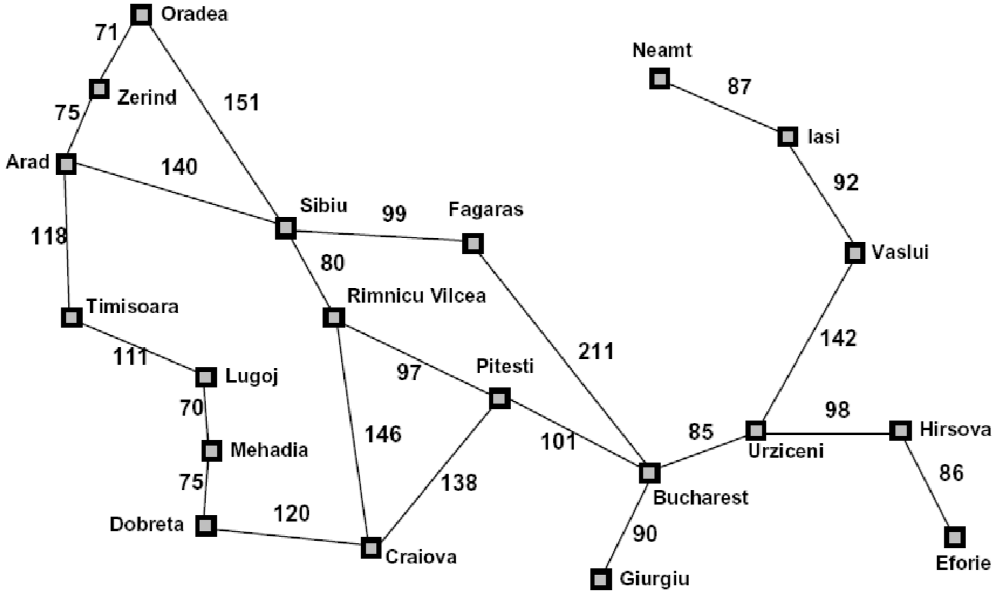
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

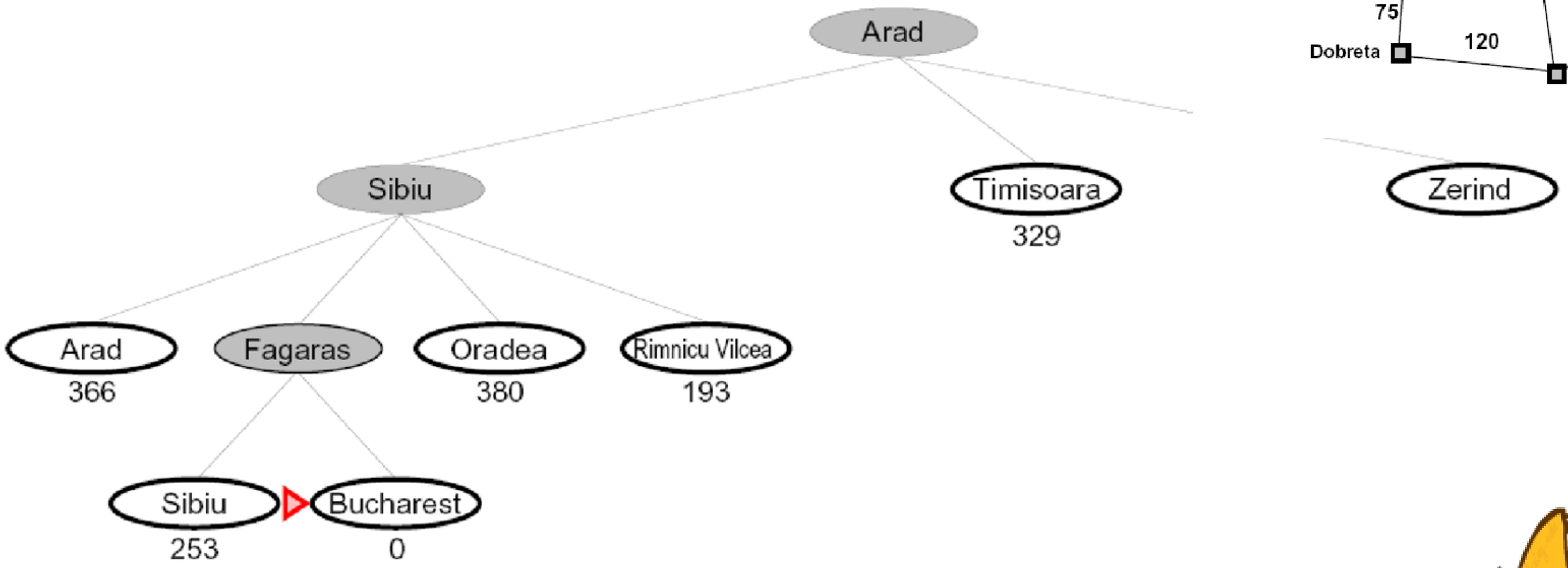
Greedy Search

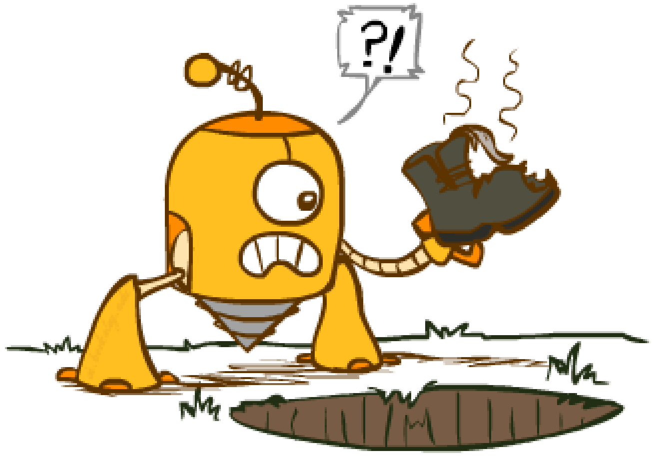
Greedy Search

Greedy Search



- o Expand the node that seems closest…



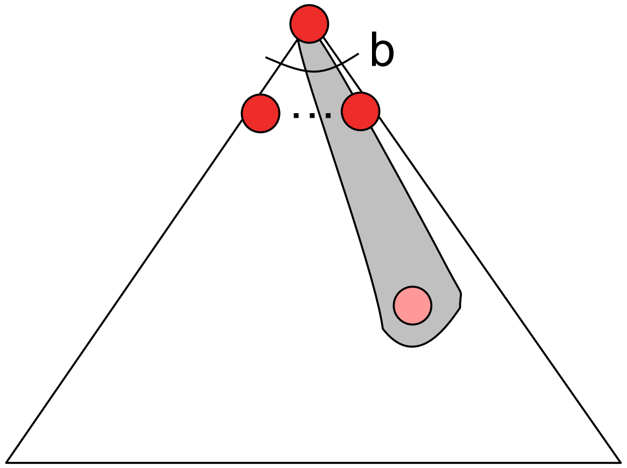


- o Is it optimal?
  - o No. Resulting path to Bucharest is not the shortest!

UNIVERSITY OF CALGARY

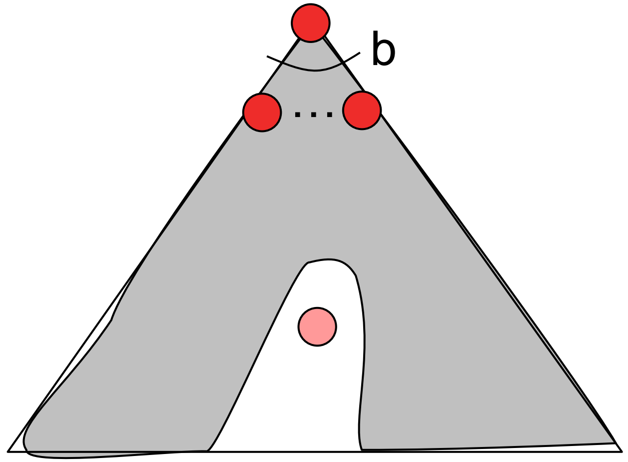
Greedy Search

- 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



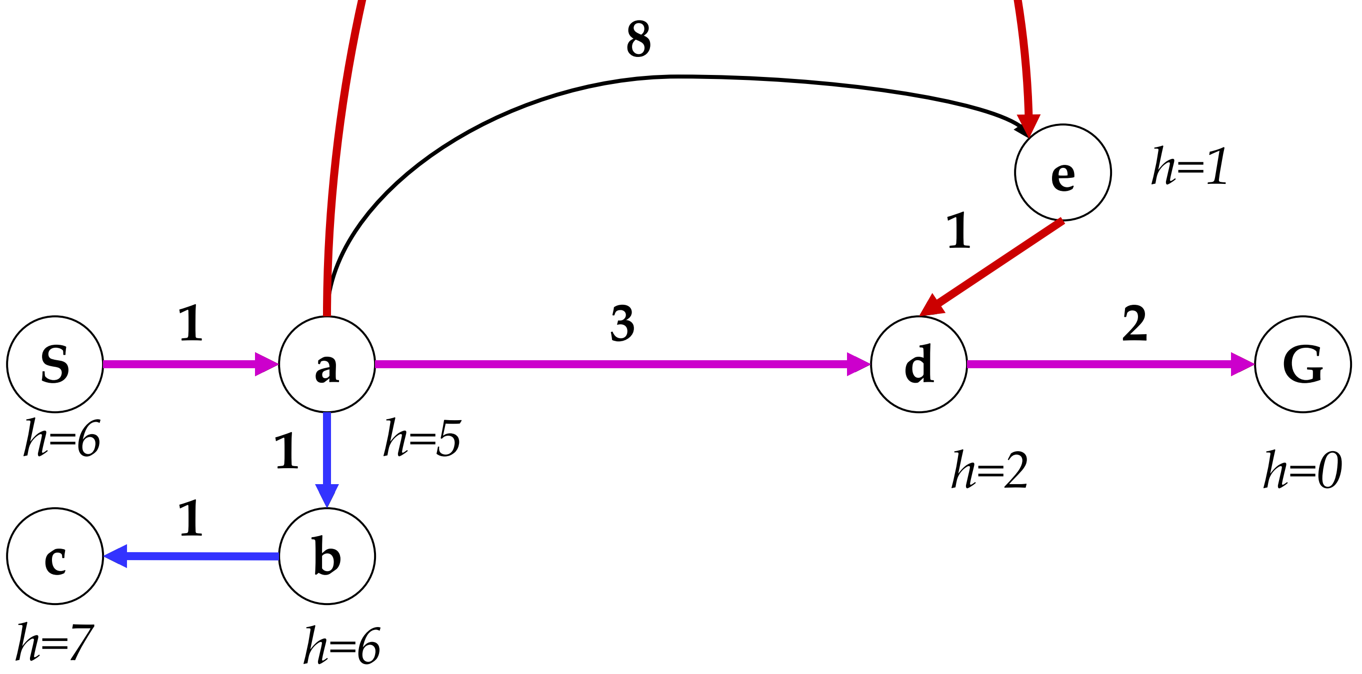
A* Search

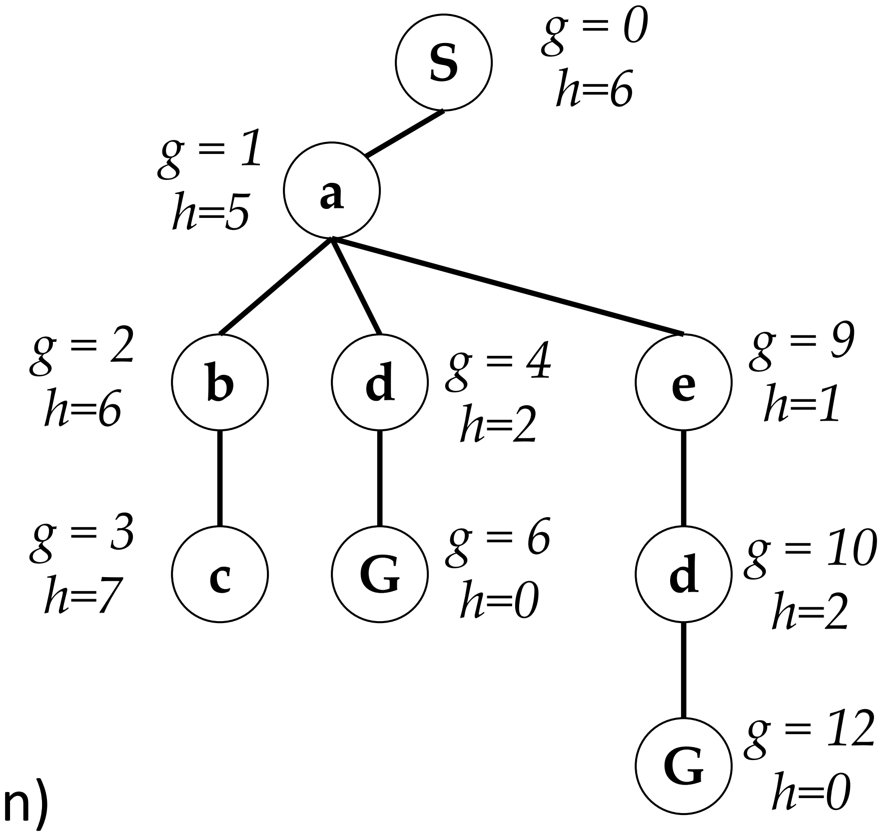
A* Search

A* 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)





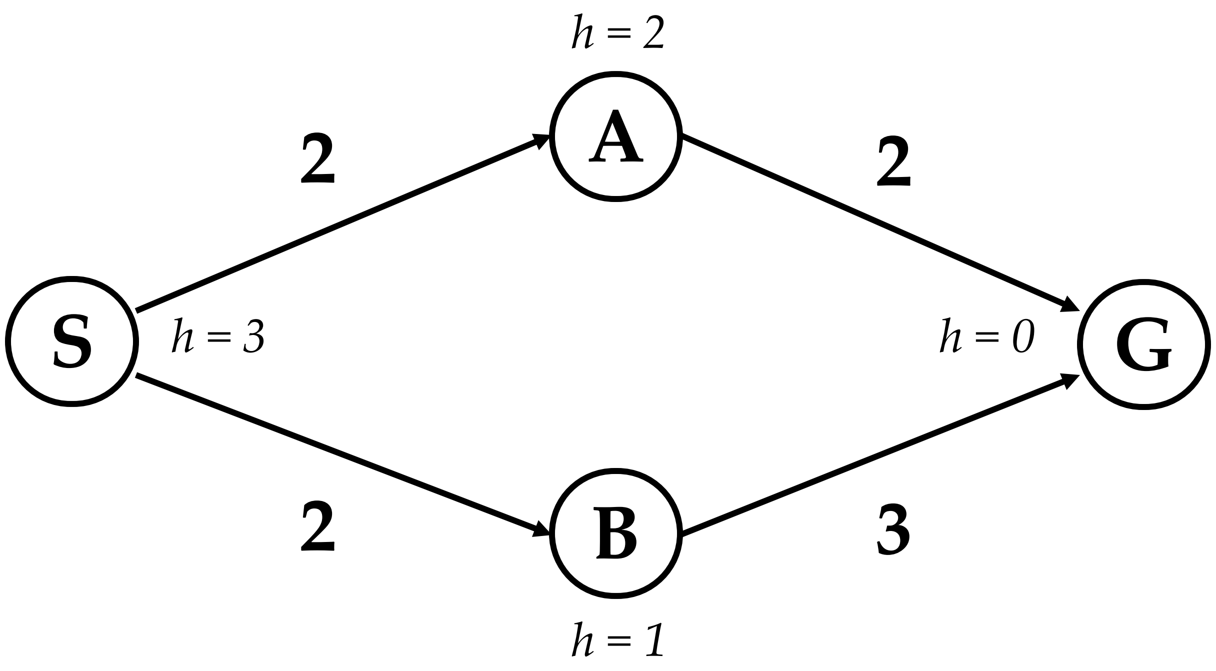
- A* Search orders by the sum: f(n) = g(n) + h(n)

UNIVERSITY OF CALGARY

Example: Teg Grenager

When should A* terminate?

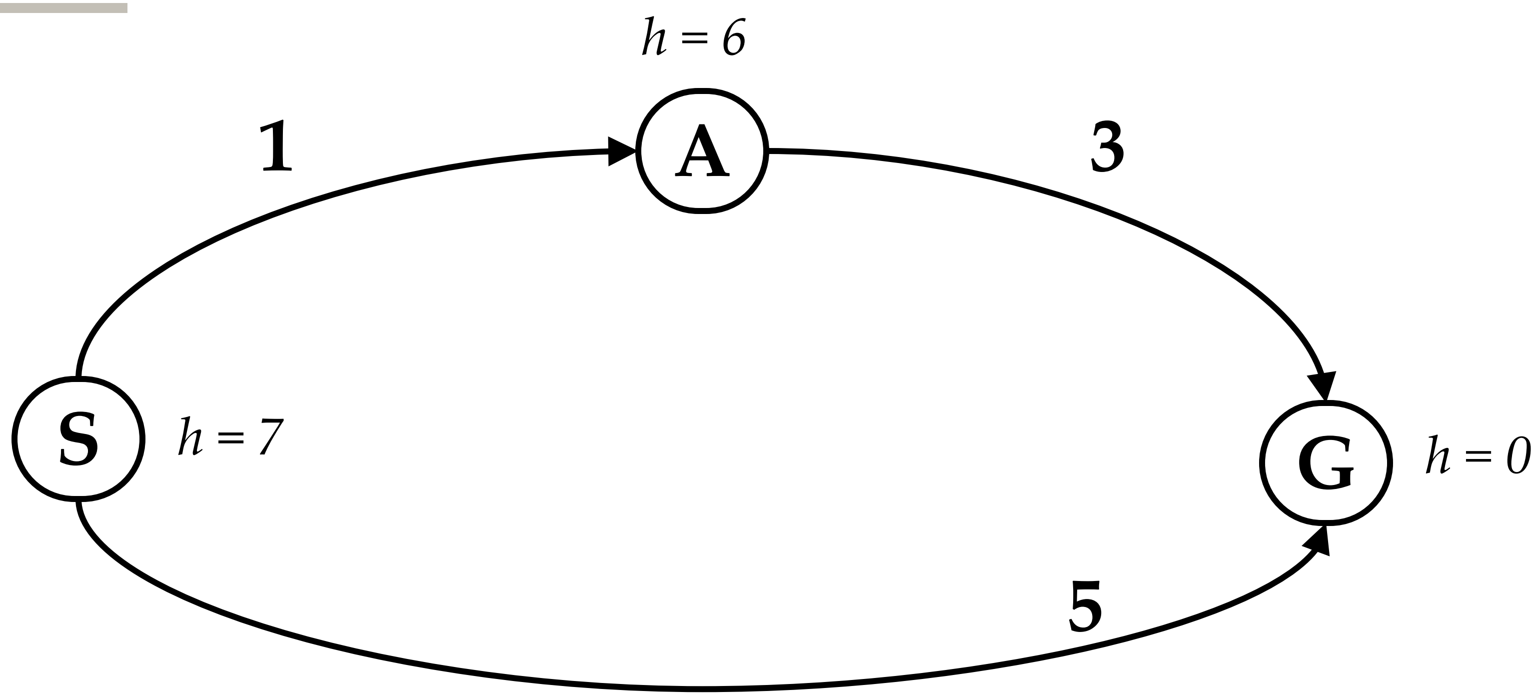
- Should we stop when we enqueue a goal?



	g	h	+
~~S~~	~~0~~	~~3~~	~~3~~
~~S->A~~	~~2~~	~~2~~	~~4~~
~~S->B~~	~~2~~	~~1~~	~~3~~
S->B->G	5	0	5
$\boxed{\text{S->A->G}}$	4	0	4

- No: only stop when we dequeue a goal

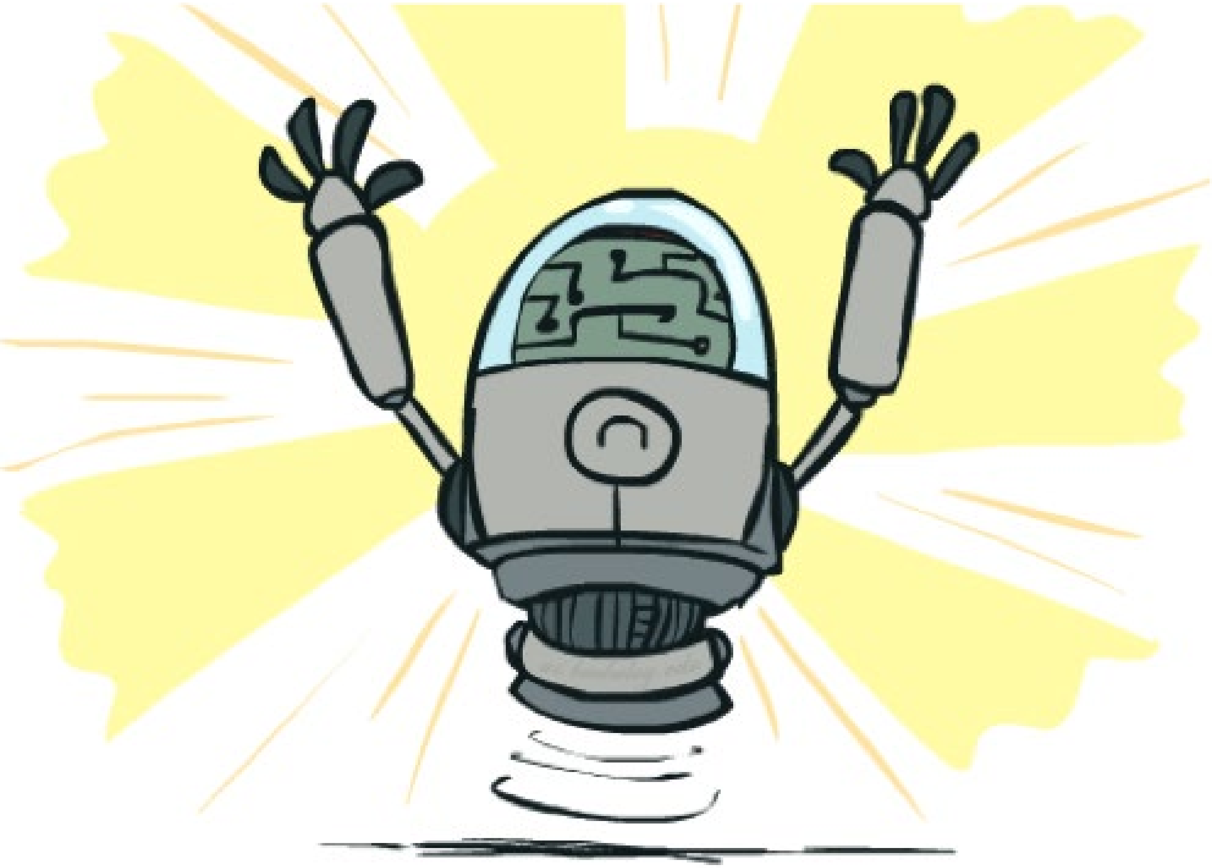
Is A* Optimal?



	g	h	+
~~S~~	~~0~~	~~7~~	~~7~~
S->A	1	6	7
S->G	5	0	5

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

Optimality of A* Tree Search



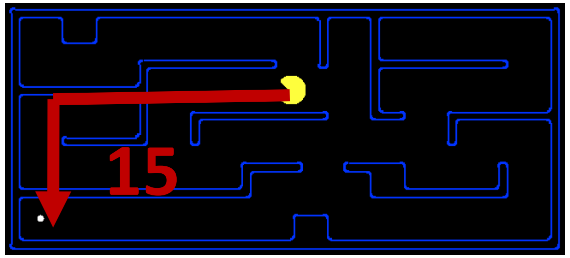
Admissible Heuristics

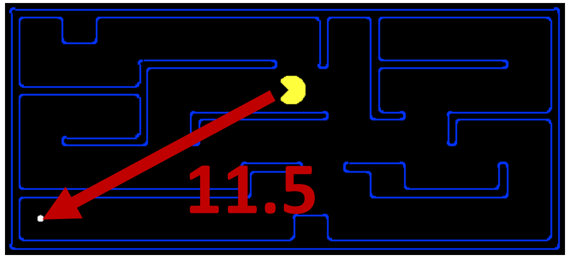
- A heuristic h is *admissible* (optimistic) if:

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

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.

UNIVERSITY OF CALGARY

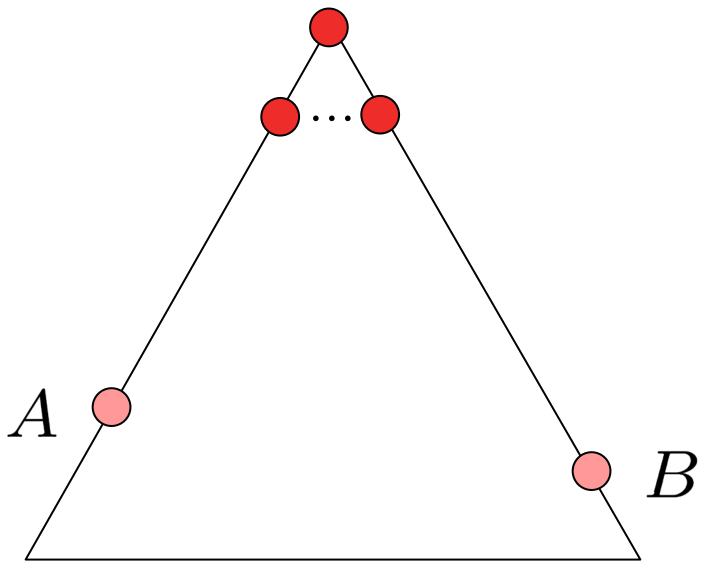
Optimality of A* Tree Search

Assume:

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

Claim:

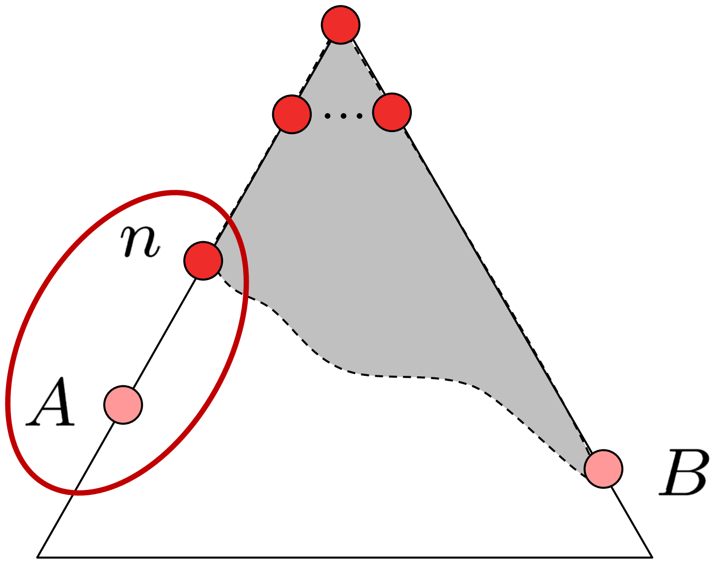
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

Proof:

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

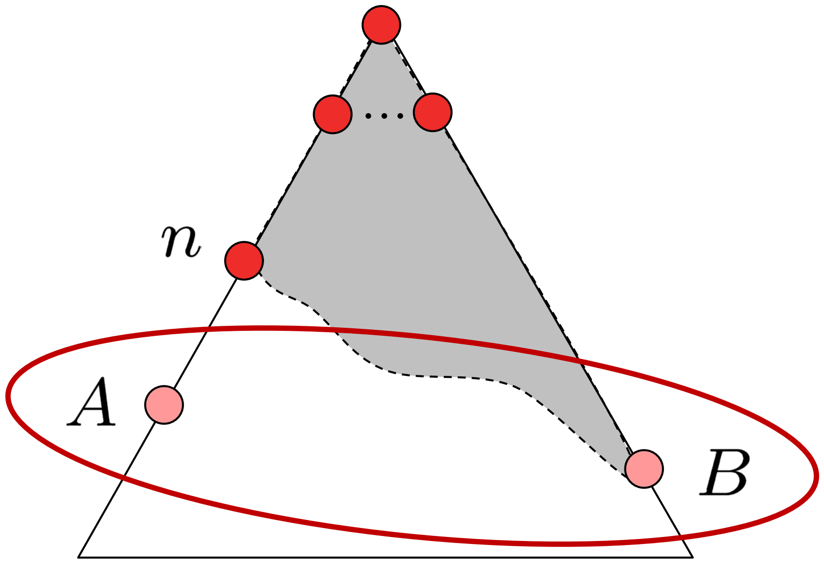


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

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



$$g(A) < g(B)$$ B is suboptimal

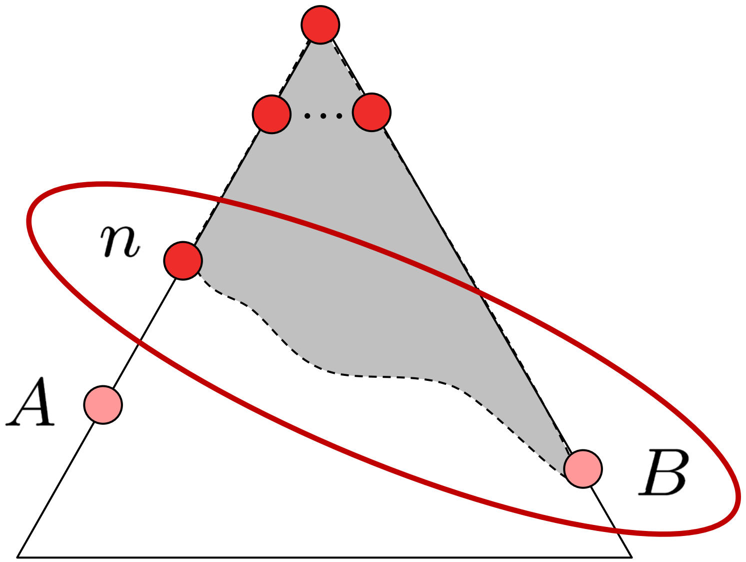
$$f(A) < f(B)$$ h = 0 at a goal

UNIVERSITY OF CALGARY

Optimality of A* Tree Search: Blocking

Proof:

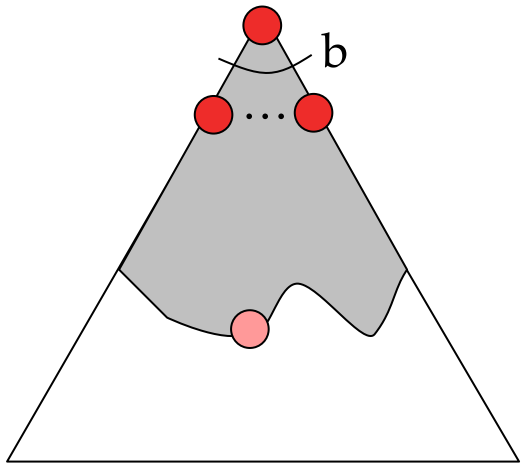
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



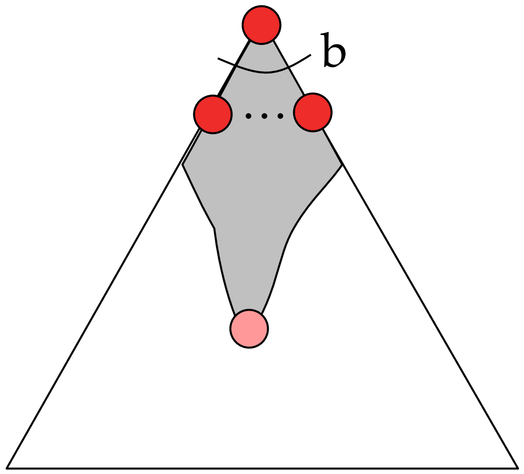
$$f(n) \leq f(A) < f(B)$$

Properties of A*

Uniform-Cost

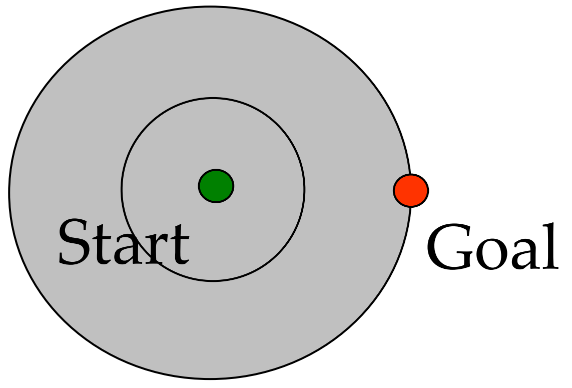


A*

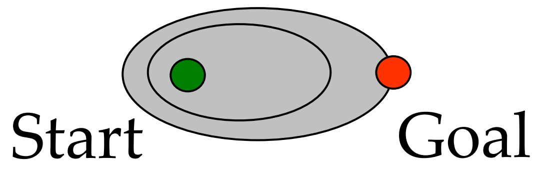


UCS vs A* Contours

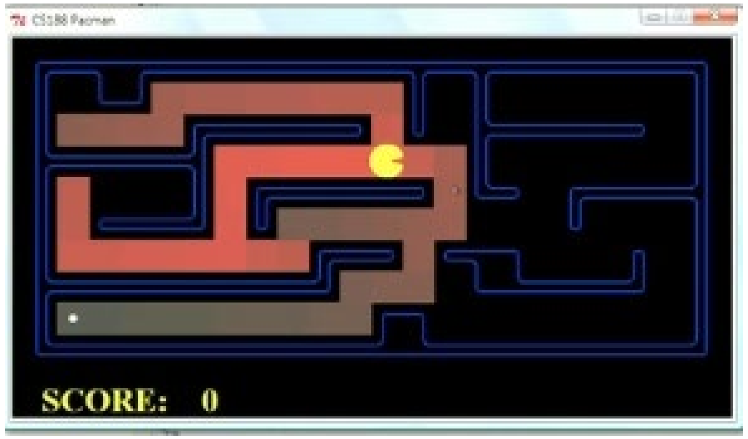
- Uniform-cost expands equally in all “directions”



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



Comparison



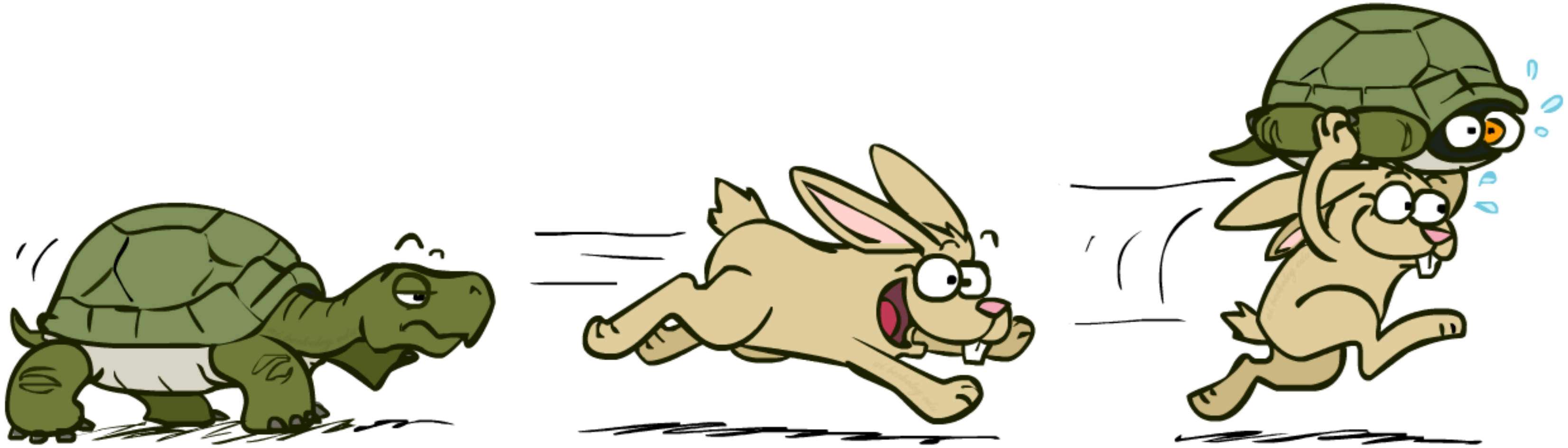
Greedy

Uniform Cost

A*

A*: Summary

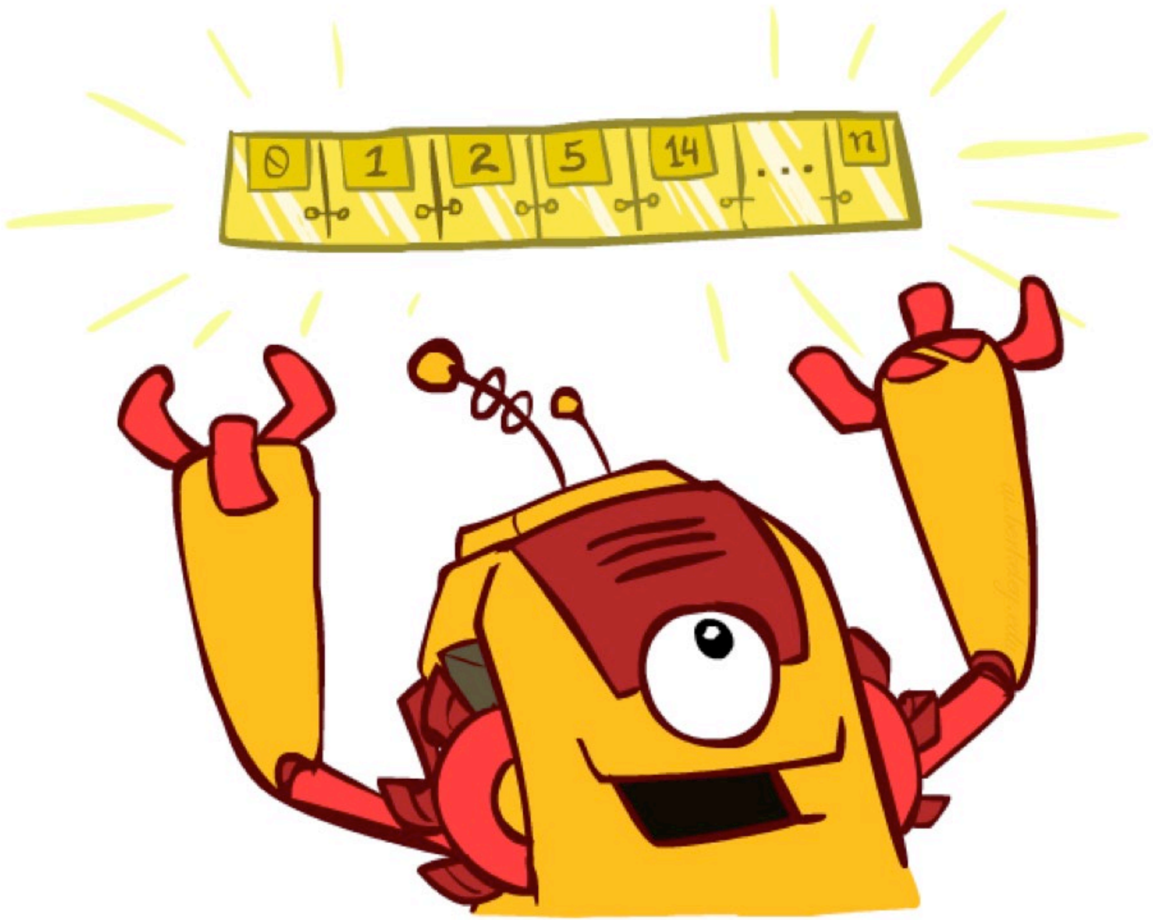
- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Queueing

The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object



UNIVERSITY OF CALGARY

Local Search

Local Search

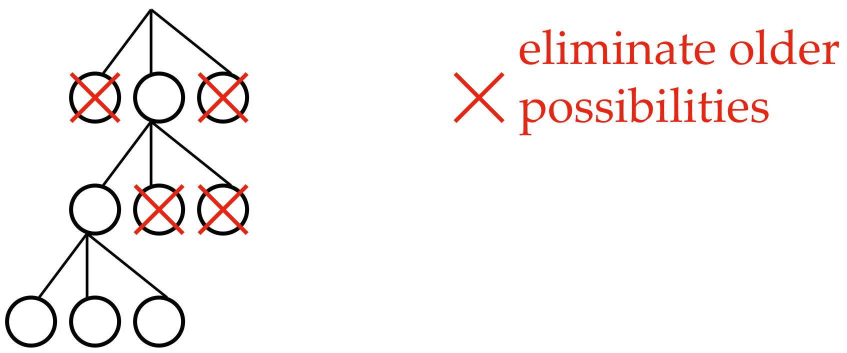
Local Search (I)

General Idea:

After selecting a transition, do not consider any transitions that were possible in previous states

☞ "Never-look-back-Heuristic"

Example: trees (works for sets also ☞ one-element sets)



Local Search (II)

Advantages:

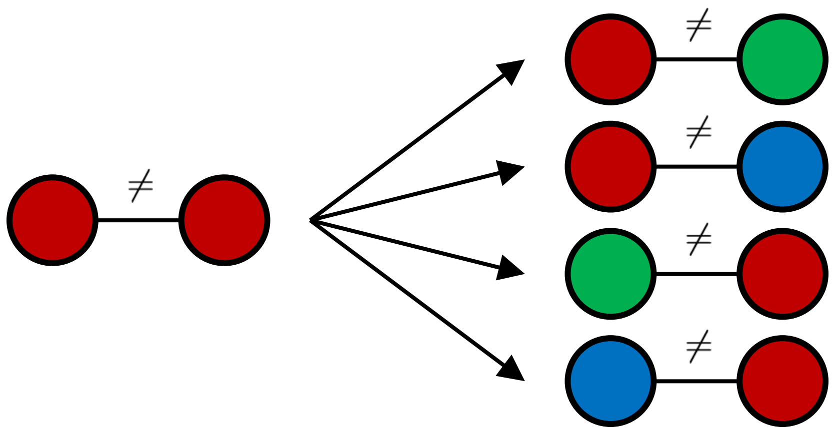
- Less decisions
- Complexity can be bound by depth of tree (number of solution steps)
- Each transition contributes to found solution
- Predictable behavior with regard to run time

Disadvantages

- No guarantee for optimality of solution
- No guarantee for optimality of number of necessary transitions

Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can't make it better (no fringe!)
- New successor function: local changes



- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing

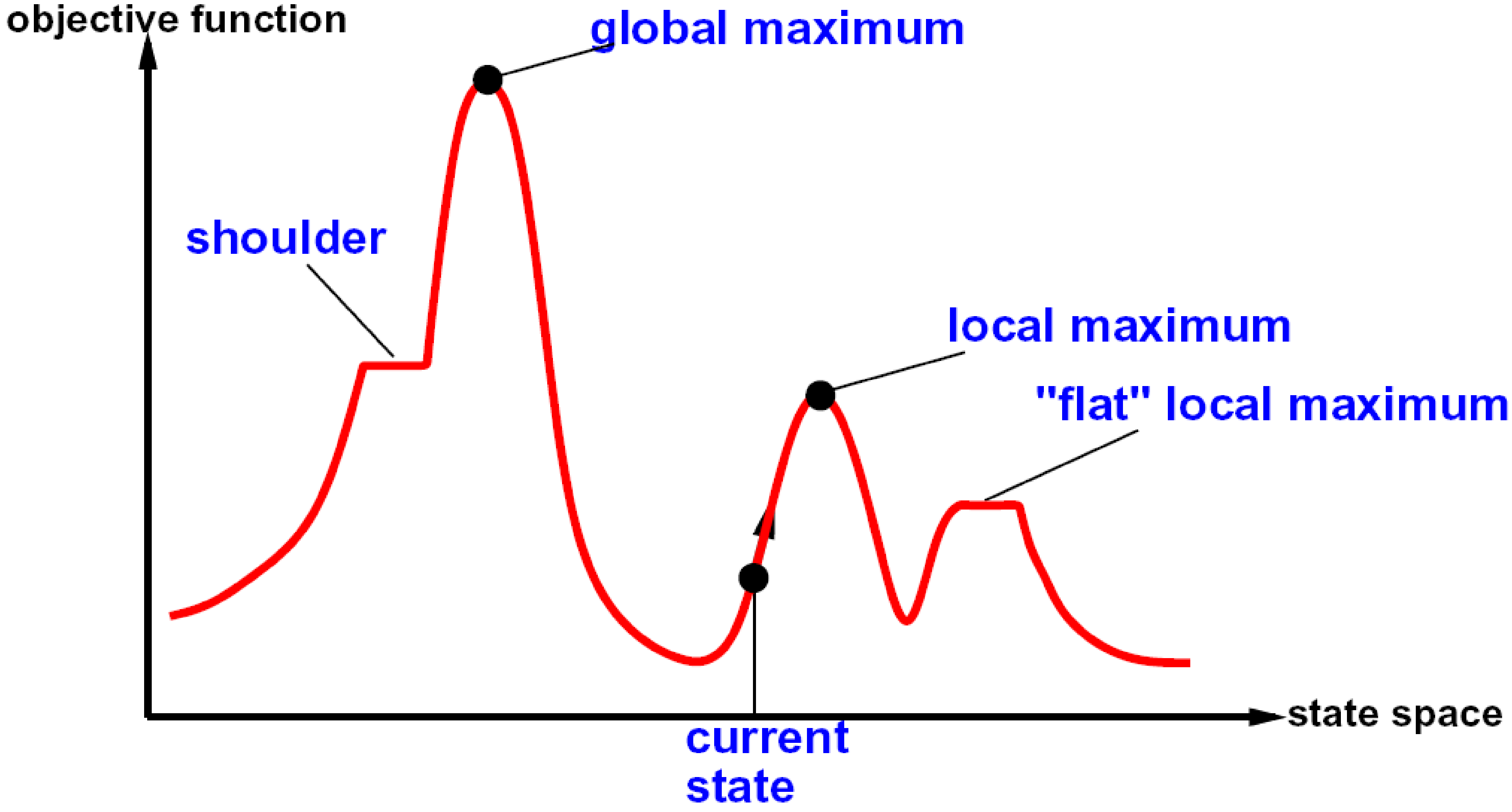
- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What's bad about this approach?

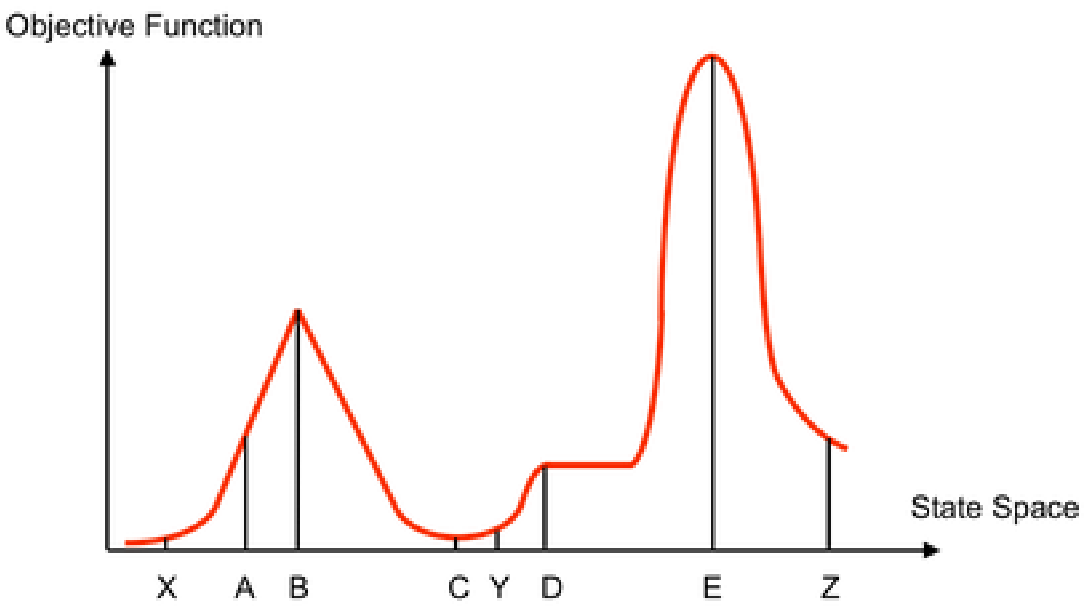
- What's good about it?

CALGARY

Hill Climbing Diagram



Hill Climbing Quiz



Starting from X, where do you end up ?

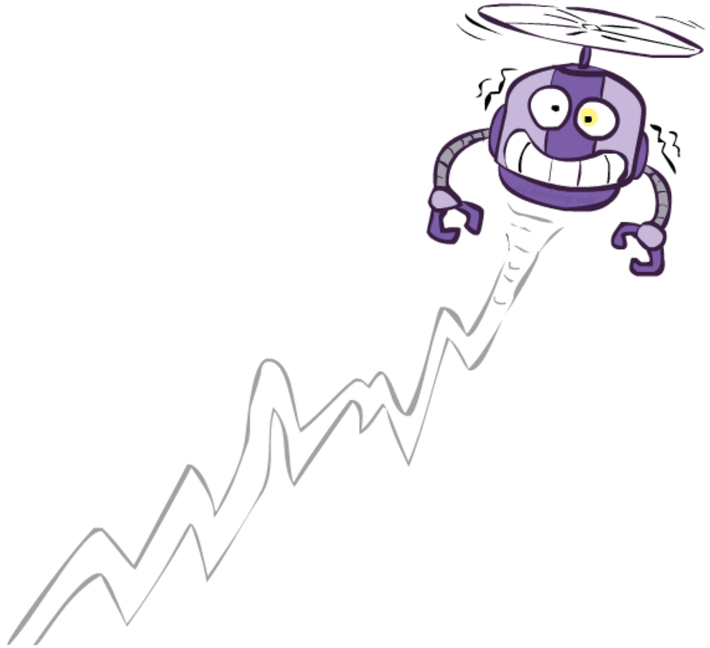
Starting from Y, where do you end up ?

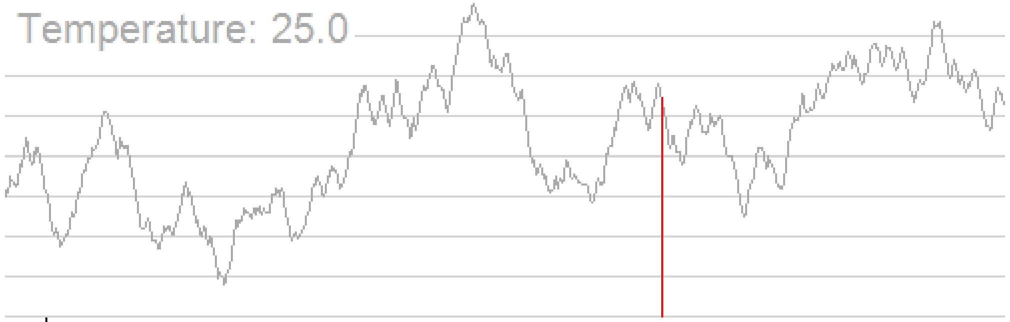
Starting from Z, where do you end up ?

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

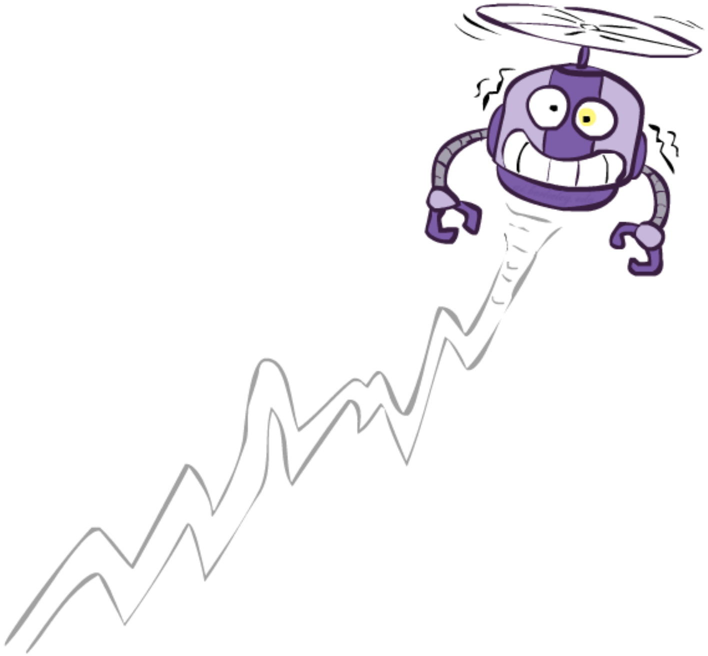
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```





Simulated Annealing

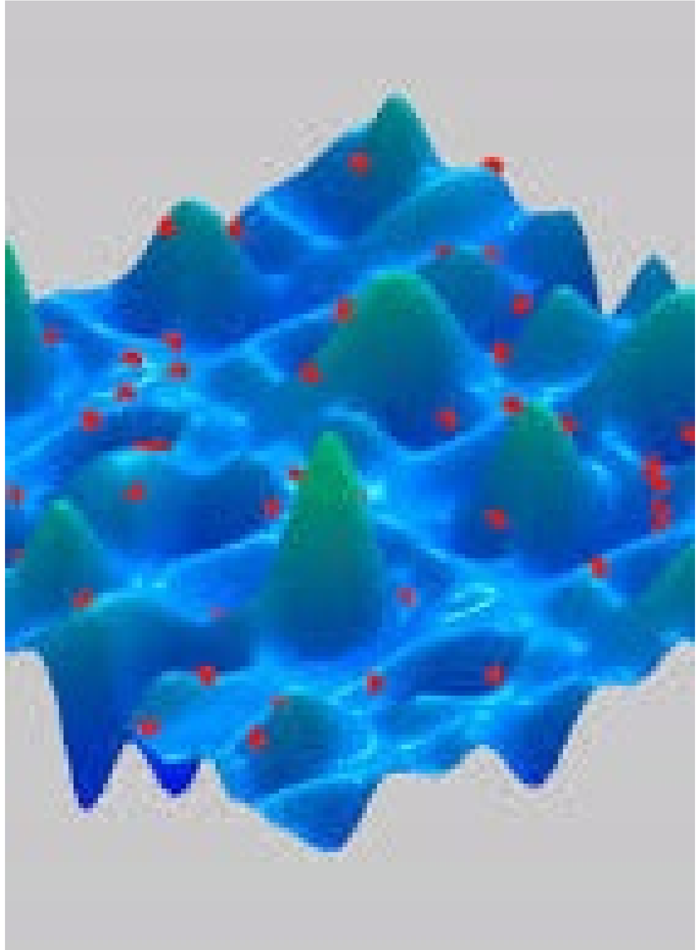
- Theoretical guarantee:
  - Stationary distribution: $p(x) \propto e^{\frac{E(x)}{kT}}$
  - If T decreased slowly enough, will converge to optimal state!



- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

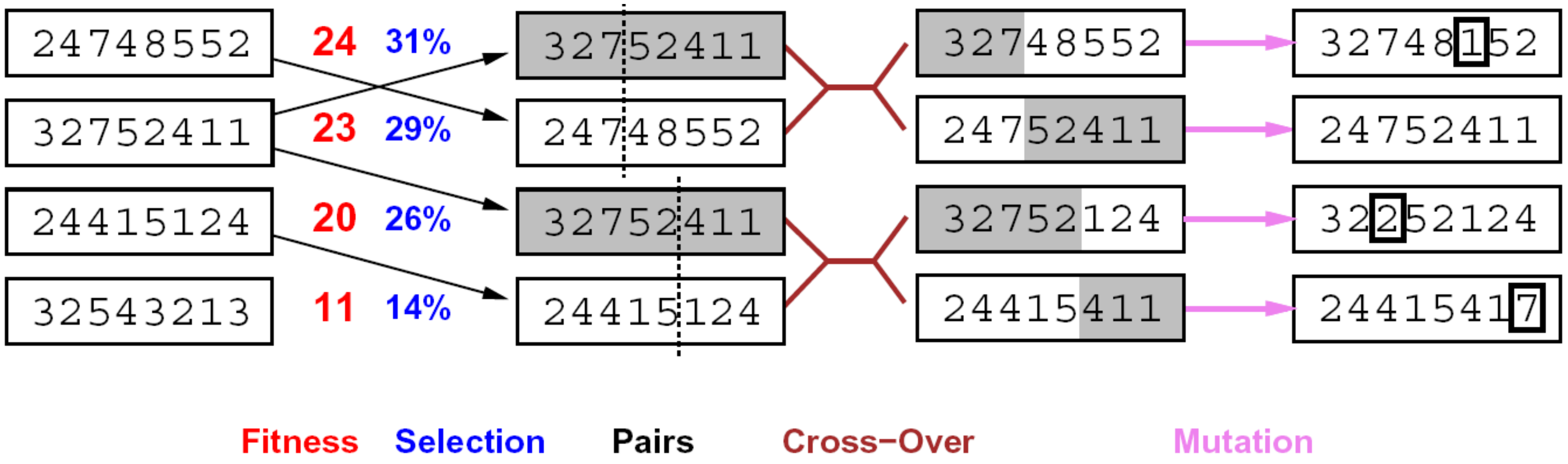
UNIVERSITY OF CALGARY

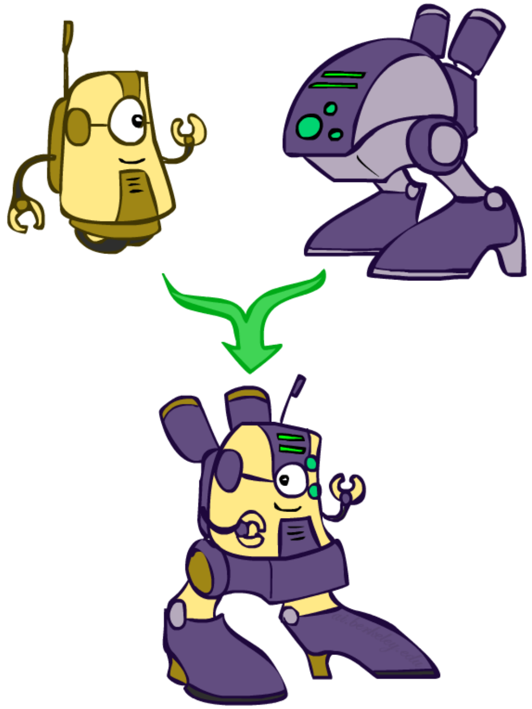
Particle Swarm Optimization



- Design complexity grows.
- Think of particles as having ‘gravity’. The better the solution the more ‘gravity’.
- Particles also have momentum.
- Have many particles.
- Each step, particles follow their current direction of change with influence of the nearby local optima and global optima.
- Less touchy to parameters and good at exploration. Often cooling principle included to help find best at end.
- Challenges with discrete problems.

Genetic Algorithms (Common Set-Based Search)





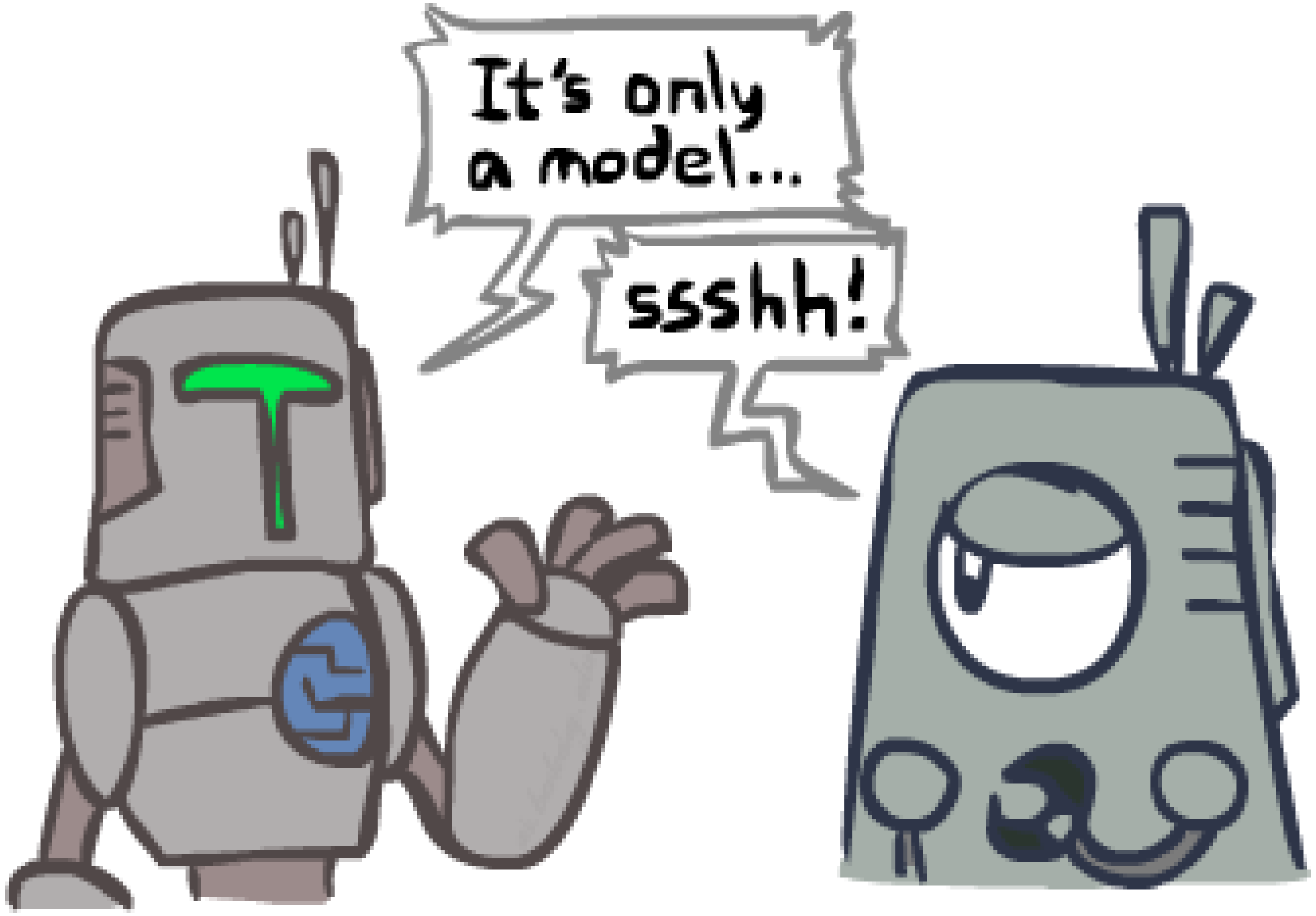
- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

UNIVERSITY OF CALGARY

Search Summary

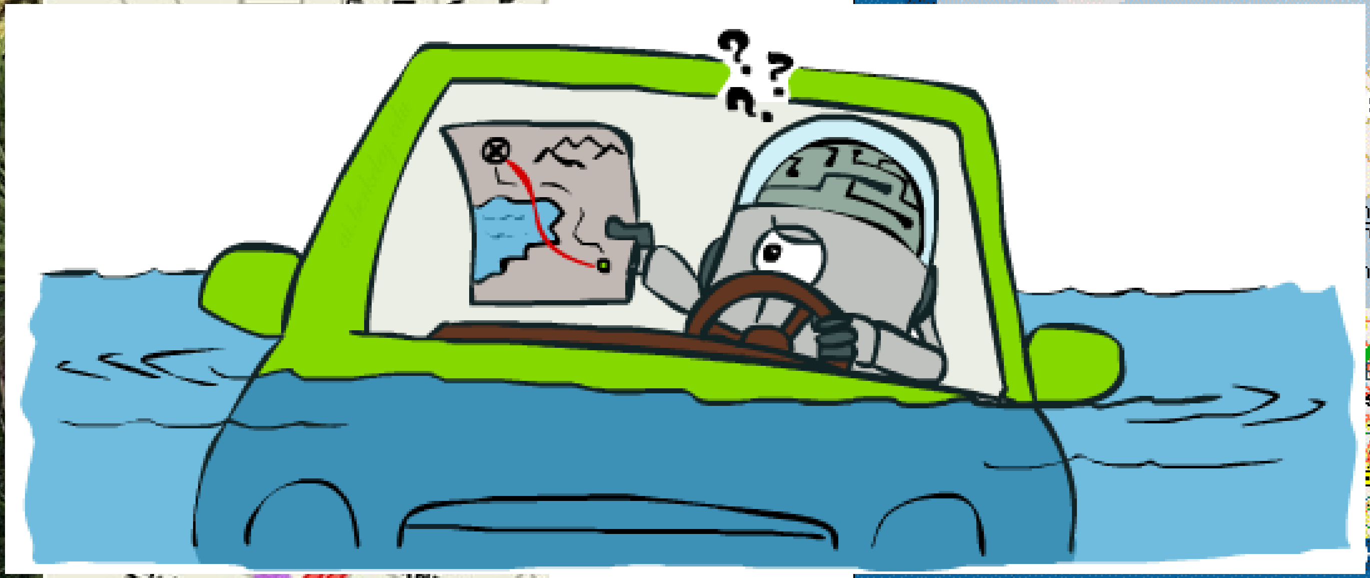
Search and Models

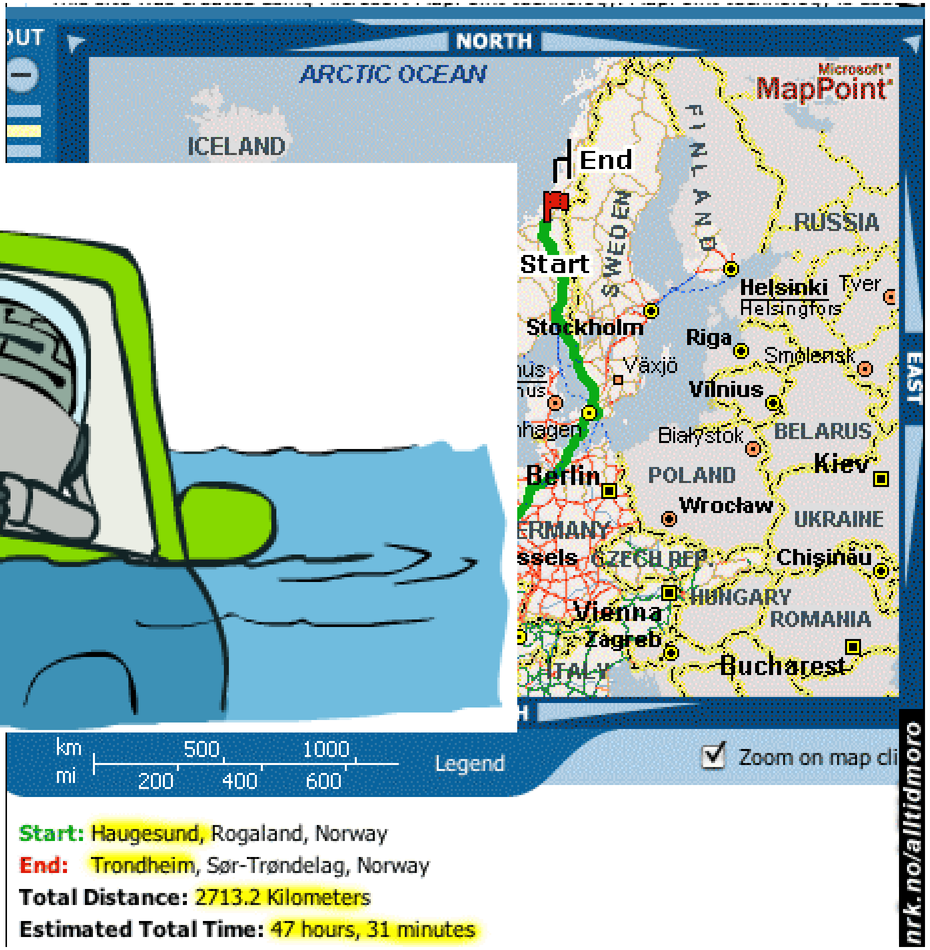
- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - Planning is all "in simulation"
  - Your search is only as good as your models…



UNIVERSITY OF CALGARY

Search Gone Wrong?





UNIVERSITY OF CALGARY

Onward to … Knowledge Representation

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY