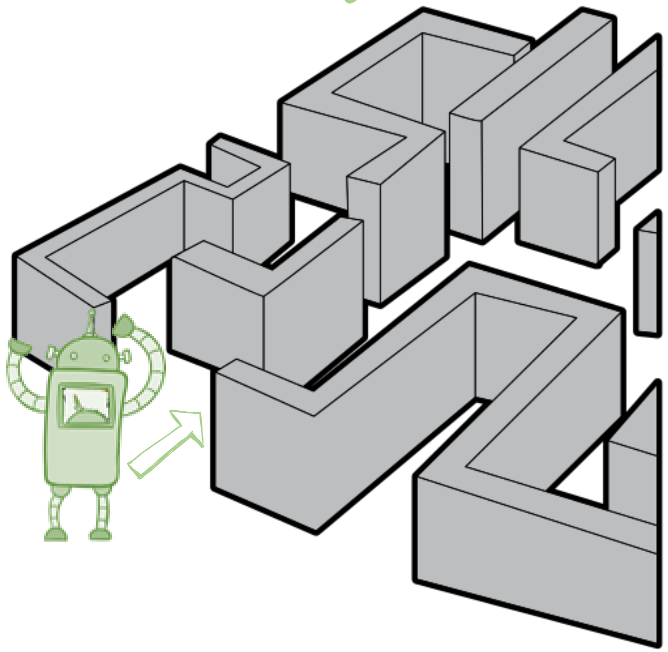
CS5491: Artificial Intelligence

Search



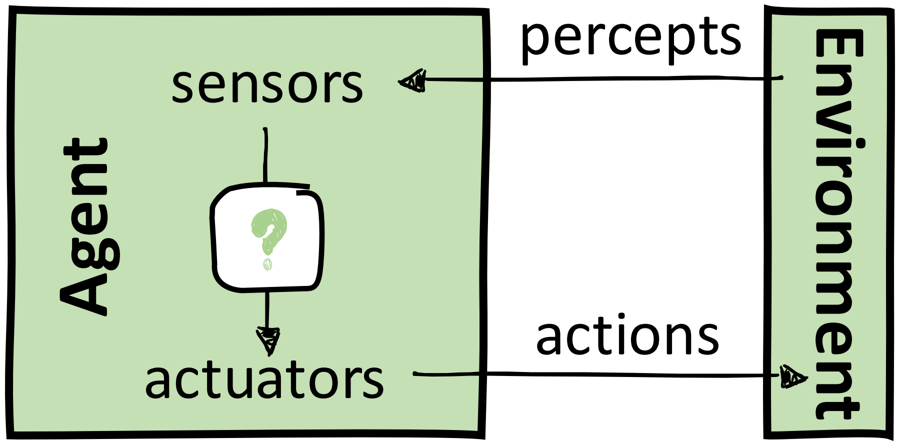
Instructor: Kai Wang

Recap: Intelligent Agents

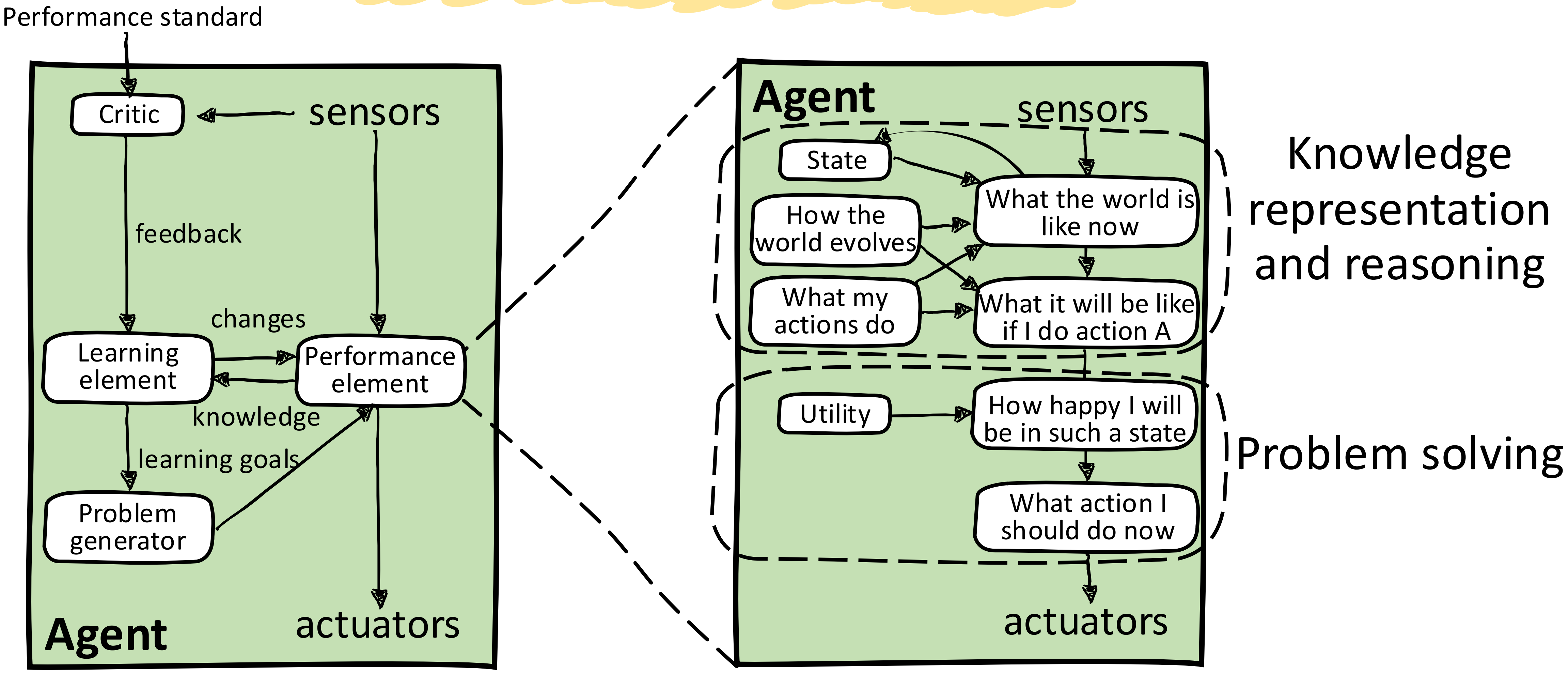
- Agent: perceiving its environment through sensors and acting upon that environment through actuators.

- Agent = architecture + program

- Agent types: simple reflex agents, model-based reflex agents, goal-based agents, utility-based agents, learning agents



Recap: The Big Picture

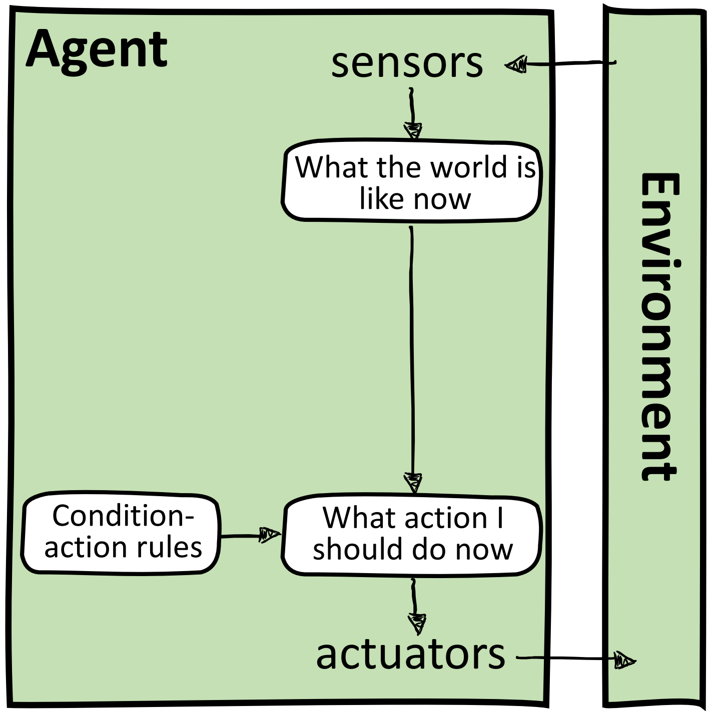


Agents with learning

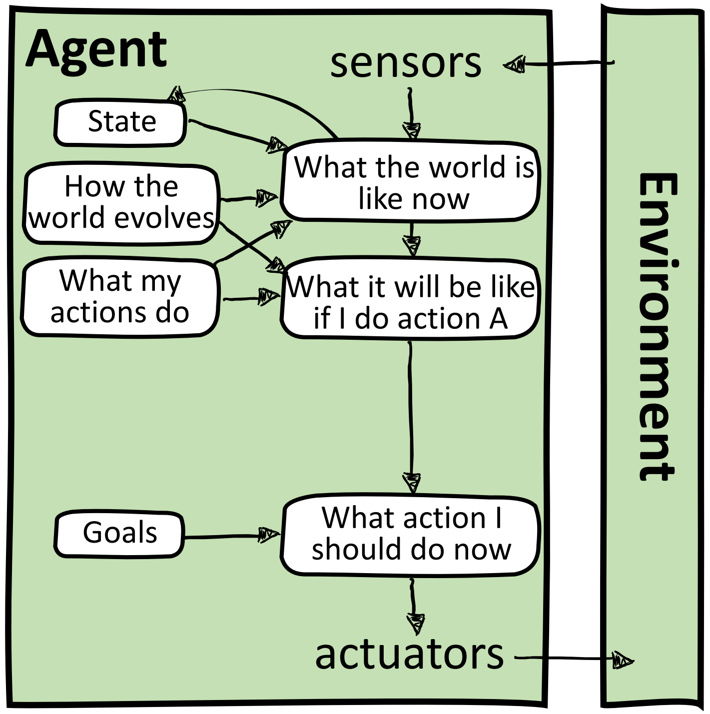
Agents without learning

Reflex vs. Problem-solving Agents

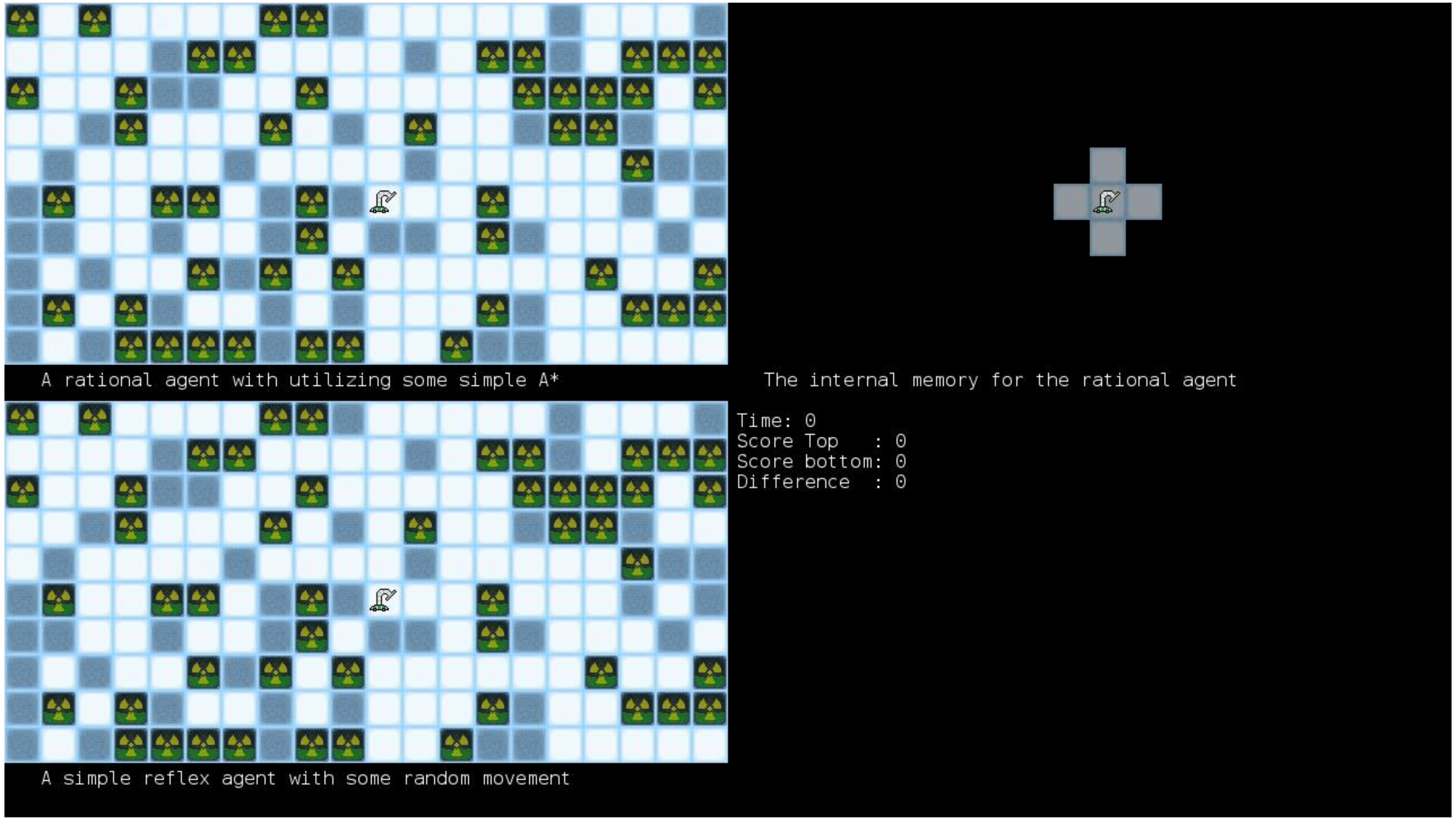
Reflex agents



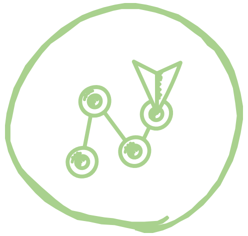
Problem-solving agents



Reflex vs. Problem-solving Agents



Today

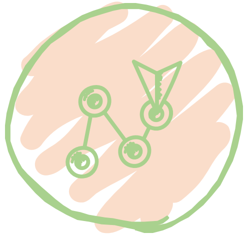


Search problems

Uninformed search

Informed search

Today

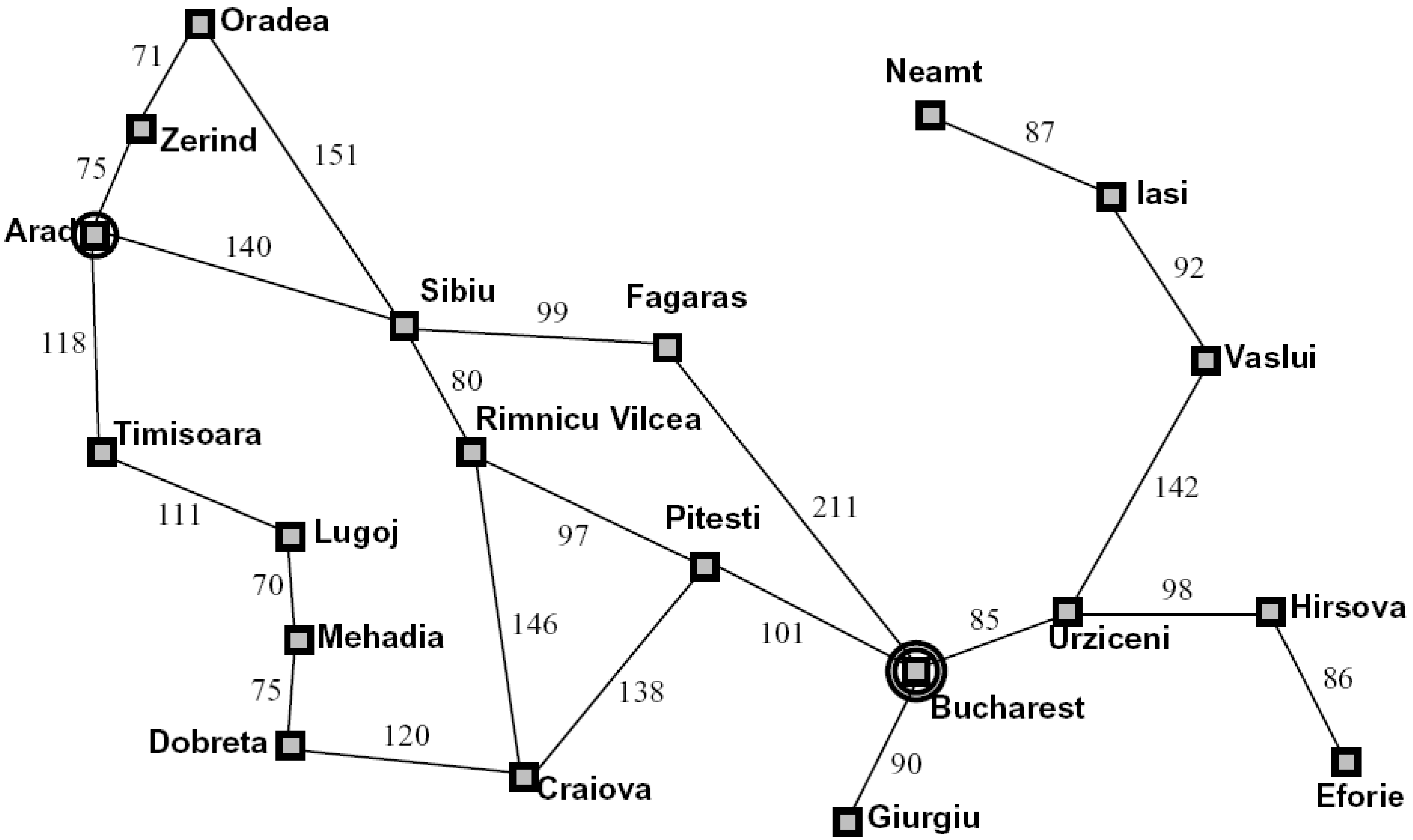


Search problems

Uninformed search

Informed search

Example: Holiday in Romania



The Search Problem

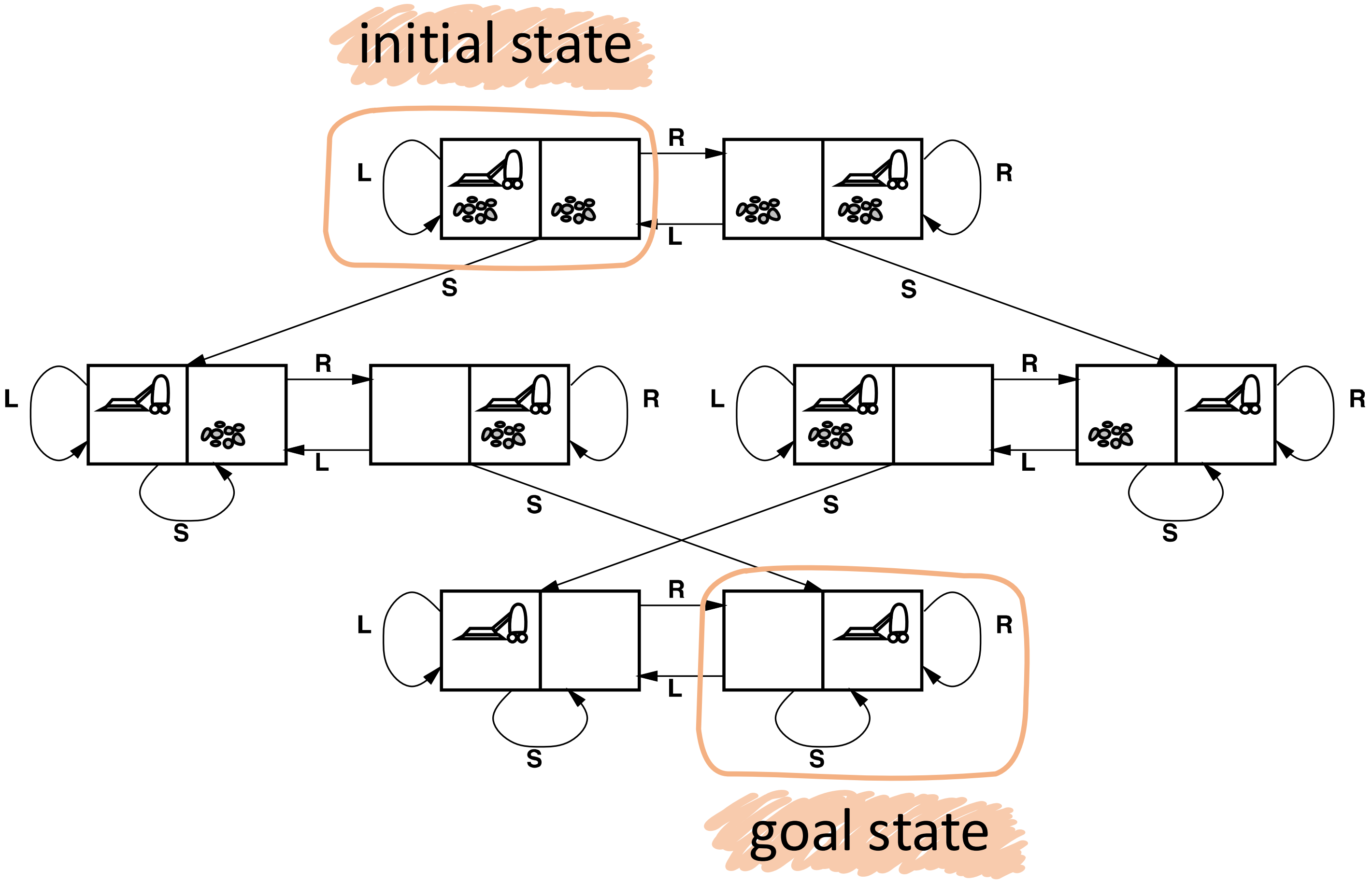
A search problem is defined as finding a solution sequence of actions which transforms the start state to a goal state.

Search problem

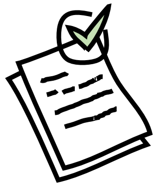
- State space: all possible configurations, e.g., cities
- Initial states: e.g., Arad
- Goal states: e.g., Bucharest
- Successor function: roads (actions) that go to adjacent cities, e.g., successor(Arad) = {Zerind, Sibiu, Timisoara}
- Cost: distance of a road, 75 for Arad to Zerind

Example: Vacuum World

- State space
  - integer dirt and robot locations
  - 2 cells * 2 positions * 2 possibilities for dirt = 8 states
- Initial state
- Goal state
  - states that everything gots clean
- Successor function:
  - actions: suck(S), move left(L), move right(R), no_op(hits the wall)
  - transitions: arcs in the digraph
- Cost: 1 per action (0 for no_op)



Example: Vacuum World



Clicker question: What is the number of states if we have n cells?

$n * 4^n$

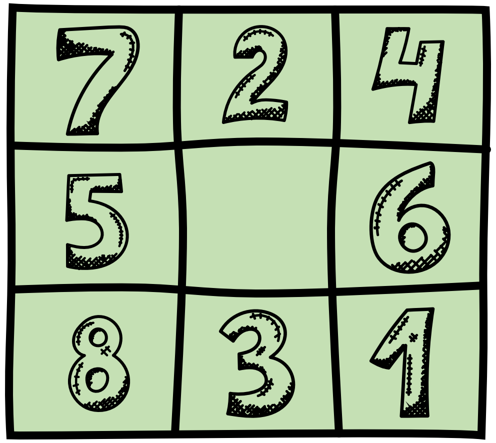
$n * 2^n$

$n * n^2$

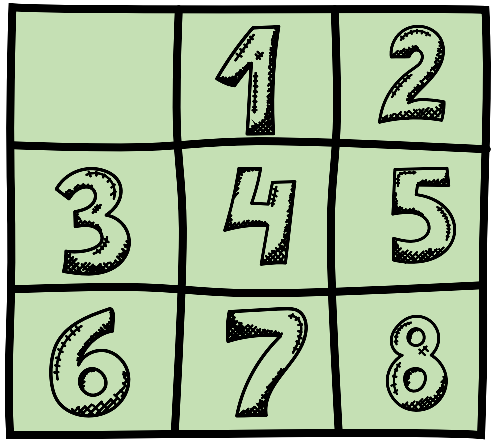
$n * n^4$

Example: 8-Puzzle

- State space
  - integer locations of tiles
- Initial state
- Goal state
- Successor function:
  - actions: move blank left, right, up, down
  - transitions: effect of the actions
- Cost: 1 per move



initial state



goal state

Art in Formulating a Search Problem

Decide only those propertiers that matter & how to represent

Initial state, goal state, possible intermediate states, state space sizes

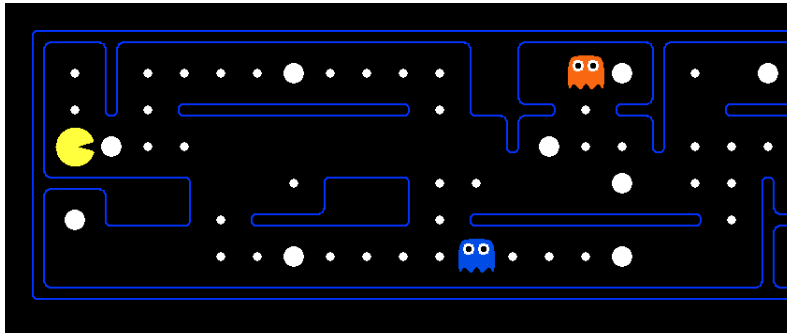
Decide which actions are possible & how to represent

Actions and transition model

Decide which action is the next

Path cost function

Hard Task: Sepecifying a State Space



Problem: Pathing

- Agent posistion

Problem: Eating all dots

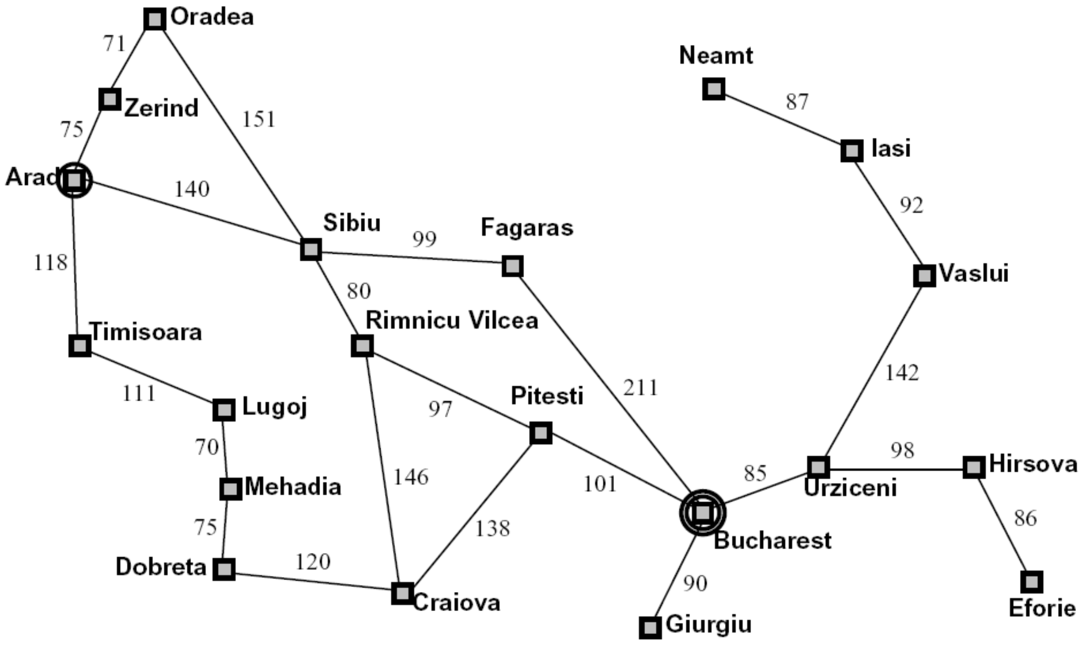
- Agent posistion
- Dot booleans

Problem: Eating all dots and keeping the ghosts perma-scared

- Agent posistion
- Dot booleans
- Power pellet booleans
- Ghost scared time

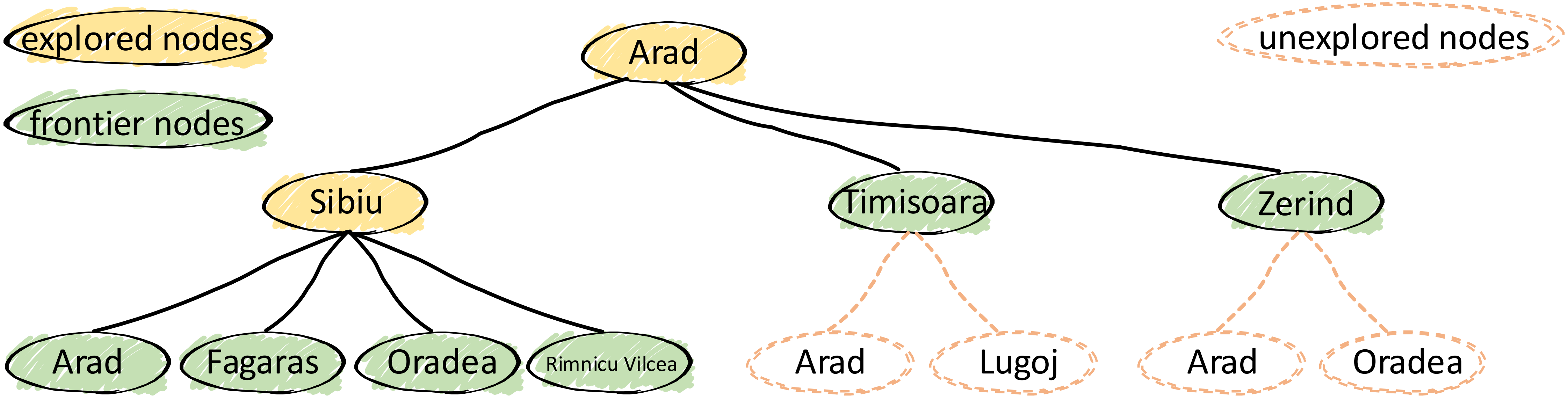
State Space Graph

A state space graph is a mathematical representation of a search problem – its node is an abstracted world configuration and its arc represents successors.



Search Tree

A search tree is a "what if" tree of paths and their outcomes – its node corresponds to a path that achieves the state showing in this node and its arc represents successors.



State Space Graph vs. Search Tree

For both, we never build its full version in memory (it’s too big). We construct both on demand and construct as little as possible.

State space graph

- Each node: a state
- Each state occurs only once.
- Good at dealing with repeated states.

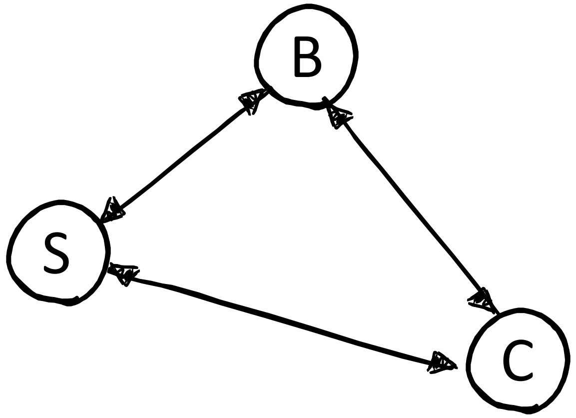
Search tree

- Each node: an entire path in the state-space graph
- Each state could occur repeatedly.
- Fail to detect repeated states → infinite size of search tree

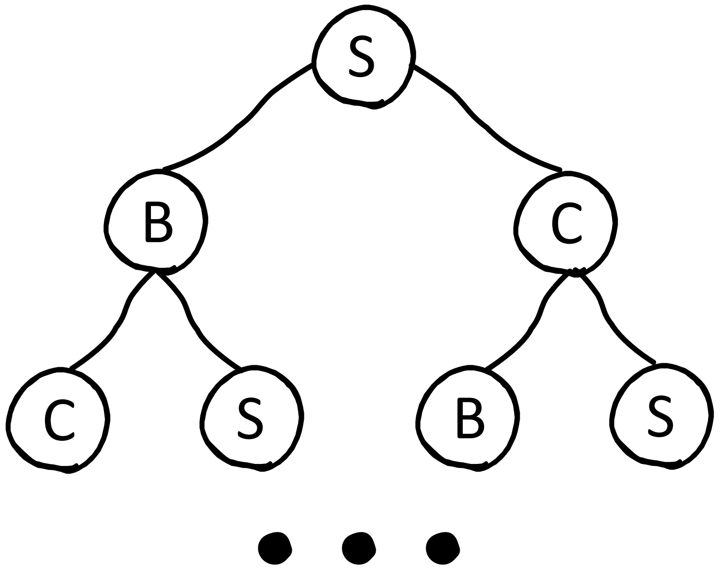
State Space Graph vs. Search Tree

For both, we never build its full version in memory (it's too big). We construct both on demand and construct as little as possible.

State space graph



Search tree



Tree Search

function TREE-SEARCH (problem, strategy) **returns** a solution, or failure

Initialize the frontier using the initial state of problem

loop do

if the frontier is empty **then return** failure

choose a leaf node according to strategy and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

else expand the node and add the resulting nodes to the frontier

end

General Tree Search

function GRAPH-SEARCH (problem, strategy) **returns** a solution, or failure

Initialize the frontier using the initial state of problem

Initialize the explored set to be empty

loop do

if the frontier is empty **then return** failure

choose a leaf node according to strategy and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

else *add the node to the explored set*, expand the node and add the resulting nodes to the frontier *only if not in the frontier or explored set*

end

Search Strategy/Algorithm

A search strategy or algorithm is the order of node expansion.

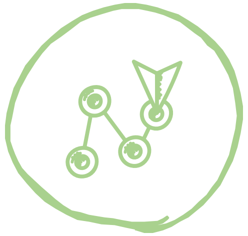
Evaluation of search algorithm

- Completeness: does it always find a solution if one exists?
- Optimality: does it always find a least path cost solution?
- Time complexity: maximum number of nodes expanded
- Space complexity: maximum number of nodes in memory

Measuring complexity

- b: maximum branching factor (finite)
- d: depth of the goal node with the least cost (finite)
- m: maximum depth of path length (potentially infinite)

Today



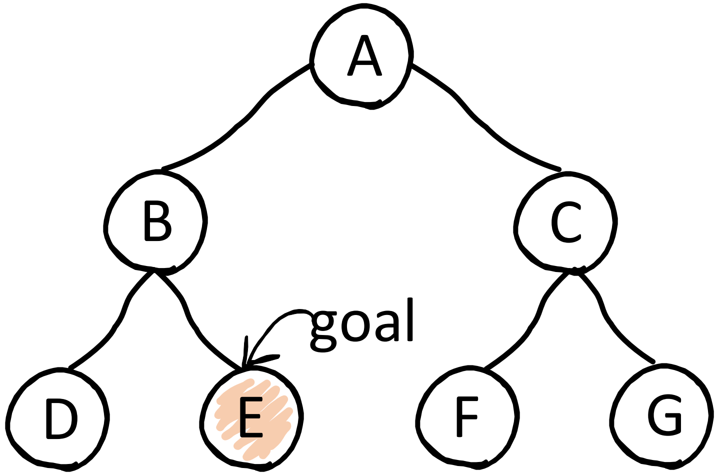
Search problems

Uninformed search

Informed search

Breadth-First Search

- Key idea: expand shallowest unexpanded node
- Implementation: frontier is a FIFO (First-In-First-Out) queue, i.e., new successors go at end



expand node	nodes list
	{A}
A	{B,C}
B	{C,D,E}
C	{D,E,F,G}
D	{E,F,G}
E	{F,G}

Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance of Breadth-First Search

- Completeness: yes if b is finite
- Optimality: only of costs are all 1 (more on costs later)
- Time complexity: $1 + b + b^2 + \cdots + b^d + b(b^d - 1) \sim O(b^d)$
- Space complexity: $b^{d-1} + b^d \sim O(b^d)$

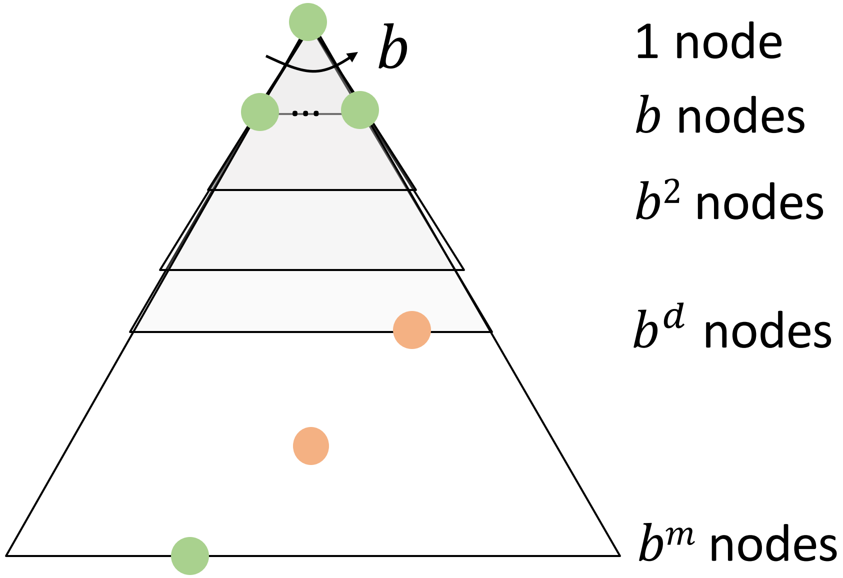


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance of Breadth-First Search

- Exponential time complexity – cannot solve any problems but those with the smallest instances.

- Exponential space complexity is a bigger problem.

depth	nodes	time	memory
4	11110	11ms	10.6MB
6	10^6	1.1s	1GB
8	10^8	2min	103GB
10	10^{10}	3hours	10TB
12	10^{12}	13 days	1PB
14	10^{14}	3.5 years	99PB

Assuming
b=10,
1M nodes/sec,
1000 bytes/node

Performance Comparison

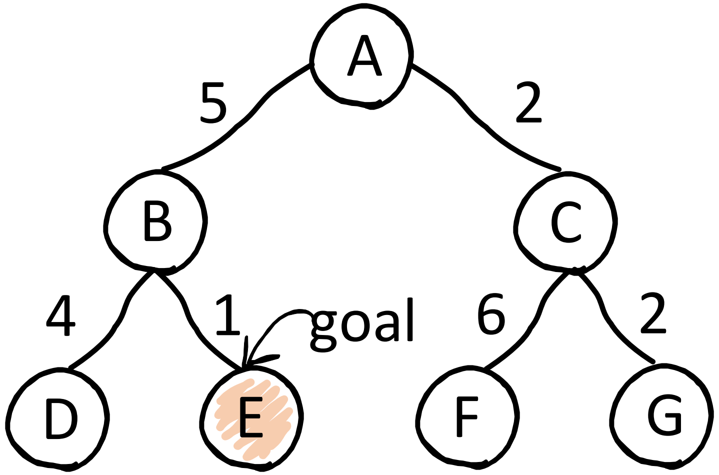
	completeness	optimality	time complexity	space complexity
Breadth-First	yes	yes, if all costs are 1	$O(b^d)$	$O(b^d)$

b: branching factor (finite), d: goal depth, m: maximum

Uniform Cost Search

- Key idea: expand cheapest unexpanded node

- Implementation: frontier is a priority queue ordered by path cost, lowest first



expand node	nodes list
	{A}
A	{C,B}
C	{G,B,F}
G	{B,F}
B	{E,F,D}
E	{F,D}

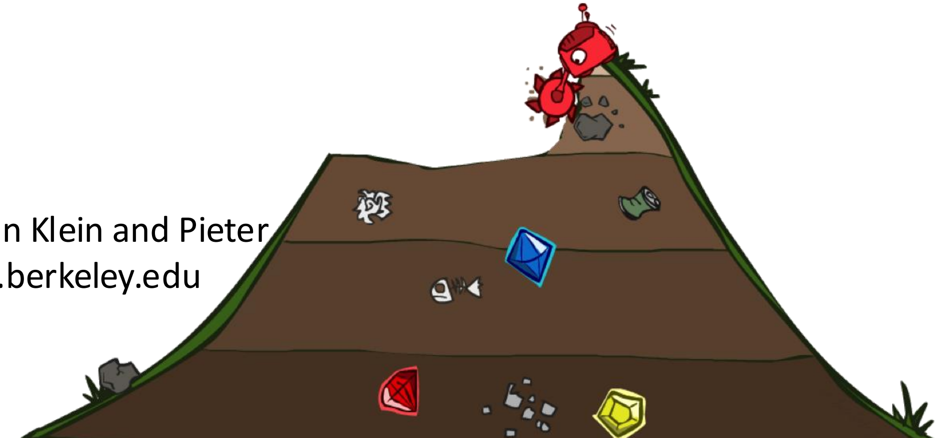


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance of Uniform Cost Search

- Completeness: yes if step cost $\geq \epsilon$ ($\epsilon > 0$) and the best solution has a finite cost

- Optimality: yes (proof via A* later)

- Time complexity: # of nodes whose costs are less than that of the optimal solution $O(b^{1+\lceil\frac{C^*}{\epsilon}\rceil})$, where C^* is the cost of the optimal solution.

- Space complexity: has roughly the last tier, $O(b^{1+\lceil\frac{C^*}{\epsilon}\rceil})$

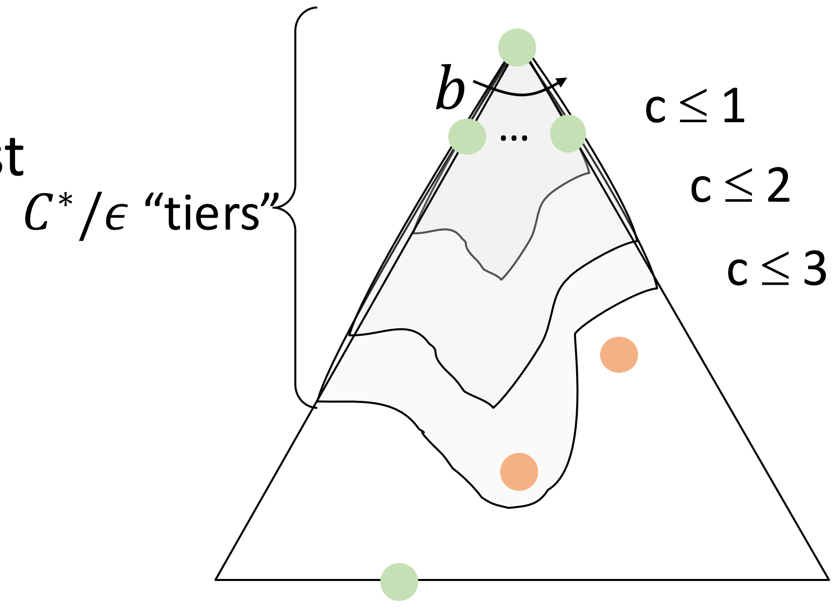


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance Comparison

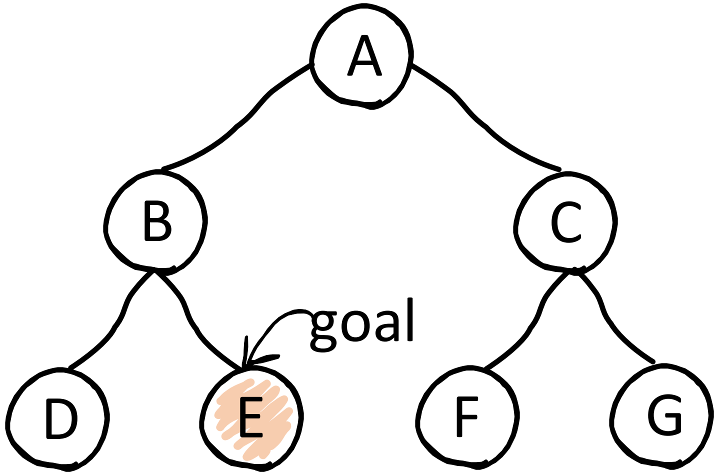
	completeness	optimality	time complexity	space complexity
breadth-first	yes	yes, if all costs are 1	$O(b^d)$	$O(b^d)$
uniform cost	yes	yes	$O(b^{1+\lceil \frac{C^*}{\epsilon} \rceil})$	$O(b^{1+\lceil \frac{C^*}{\epsilon} \rceil})$

b: branching factor (finite), d: goal depth, m: maximum

Depth-First Search

- Key idea: expand deepest unexpanded node

- Implementation: frontier is a LIFO (Last-In-First-Out) queue, i.e., new successors at front



expand node	nodes list
	{A}
A	{B,C}
B	{D,E,C}
D	{E,C}
E	{C}

Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance of Depth-First Search

- Completeness: no as it fails in infinite-depth spaces or spaces with loops (repeated states we mentioned)
- Optimality: no as it finds the leftmost solution regardless of depth or cost
- Time complexity: $O(b^m)$, which is terrible if $m \gg d$, but faster than breadth-first if solutions are dense.
- Space complexity: $O(bm)$, only siblings on path to root

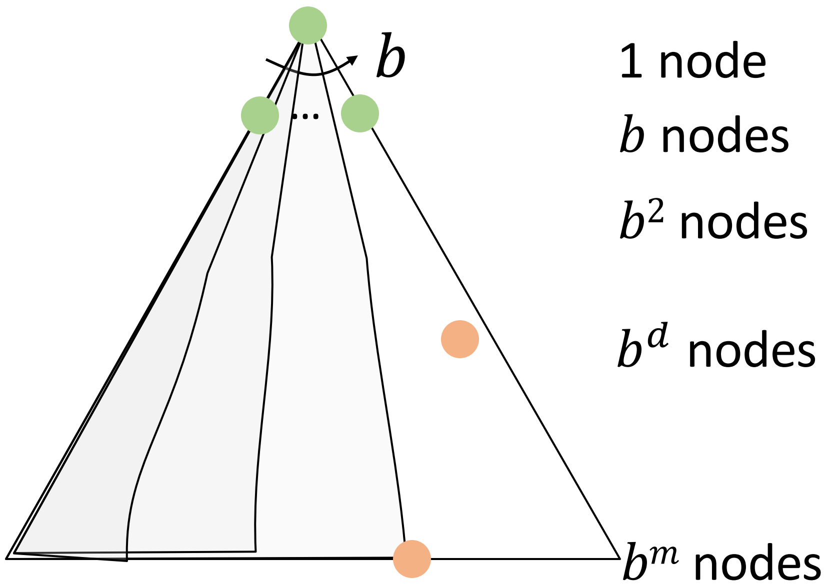


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance Comparison

	completeness	optimality	time complexity	space complexity
breadth-first	yes	yes, if all costs are 1	$O(b^d)$	$O(b^d)$
uniform cost	yes	yes	$O(b^{1+\lceil\frac{C^*}{\epsilon}\rceil})$	$O(b^{1+\lceil\frac{C^*}{\epsilon}\rceil})$
depth-first	no	no	$O(b^m)$	$O(bm)$

b: branching factor (finite), d: goal depth, m: maximum

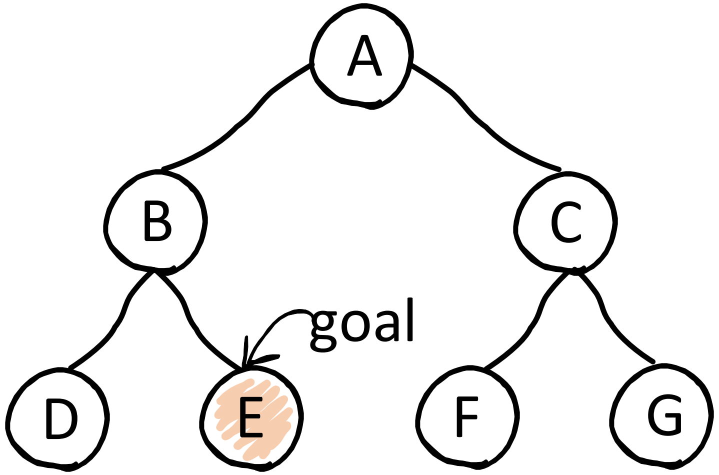
Iterative Deepening Search

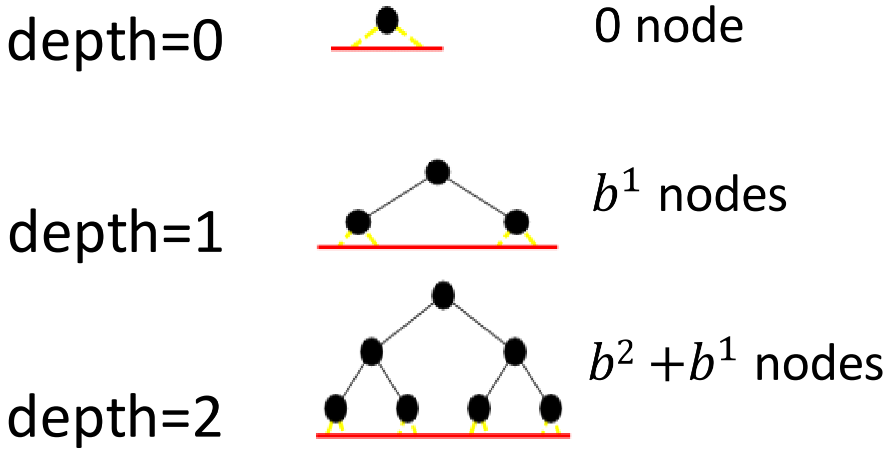
- Key idea: get DFS's space advantage with BFS's time/shallow-solution advantages

- Implementation:

for depth limit $l = 0,1, \cdots$

  Perform a depth-first search with maximum depth l





Performance of Iterative Deepening Search

- Completeness: yes, like breadth-first search
- Optimality: yes if all costs are the same
- Time complexity: $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d \sim O(b^d)$, like breadth-first search
- Space complexity: $O(bd)$

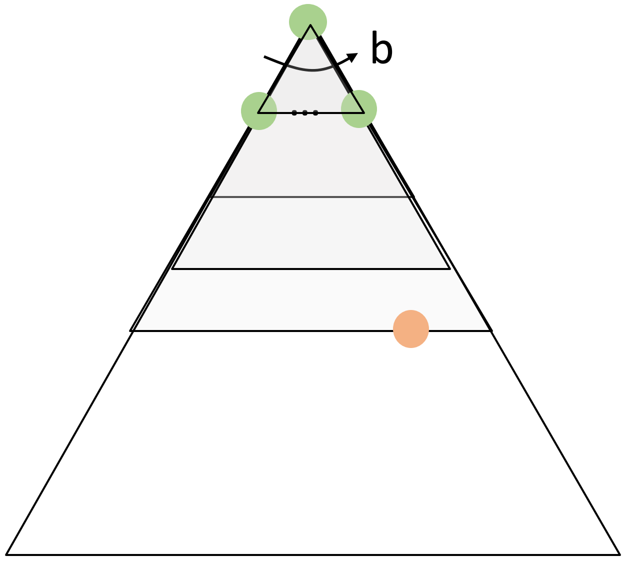


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Performance Comparison

	completeness	optimality	time complexity	space complexity
breadth-first	yes	yes, if all costs are 1	$O(b^d)$	$O(b^d)$
uniform cost	yes	yes	$O(b^{1+\lceil \frac{C^*}{\epsilon} \rceil})$	$O(b^{1+\lceil \frac{C^*}{\epsilon} \rceil})$
depth-first	no	no	$O(b^m)$	$O(bm)$
iterative deepening	yes	yes, if all costs are 1	$O(b^d)$	$O(bd)$

b: branching factor (finite), d: goal depth, m: maximum

Today

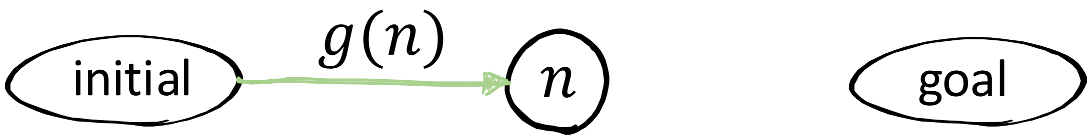
Search problems

Uninformed search

Informed search

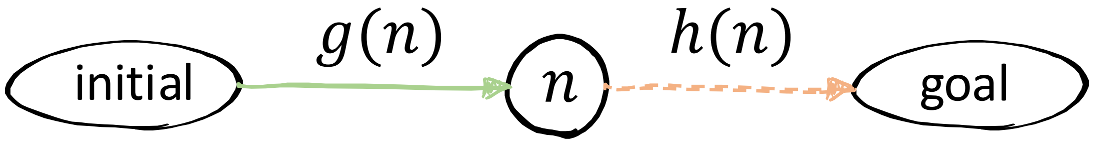
Uninformed vs. Informed Search

Uninformed search



- Only estimates the path cost $g(n)$ from the initial node to the current node n in the frontier.

Informed search



- Estimates the path cost $g(n)$ and a heuristic $h(n)$ from the current node n to the goal.
- Can be much faster than uninformed search.

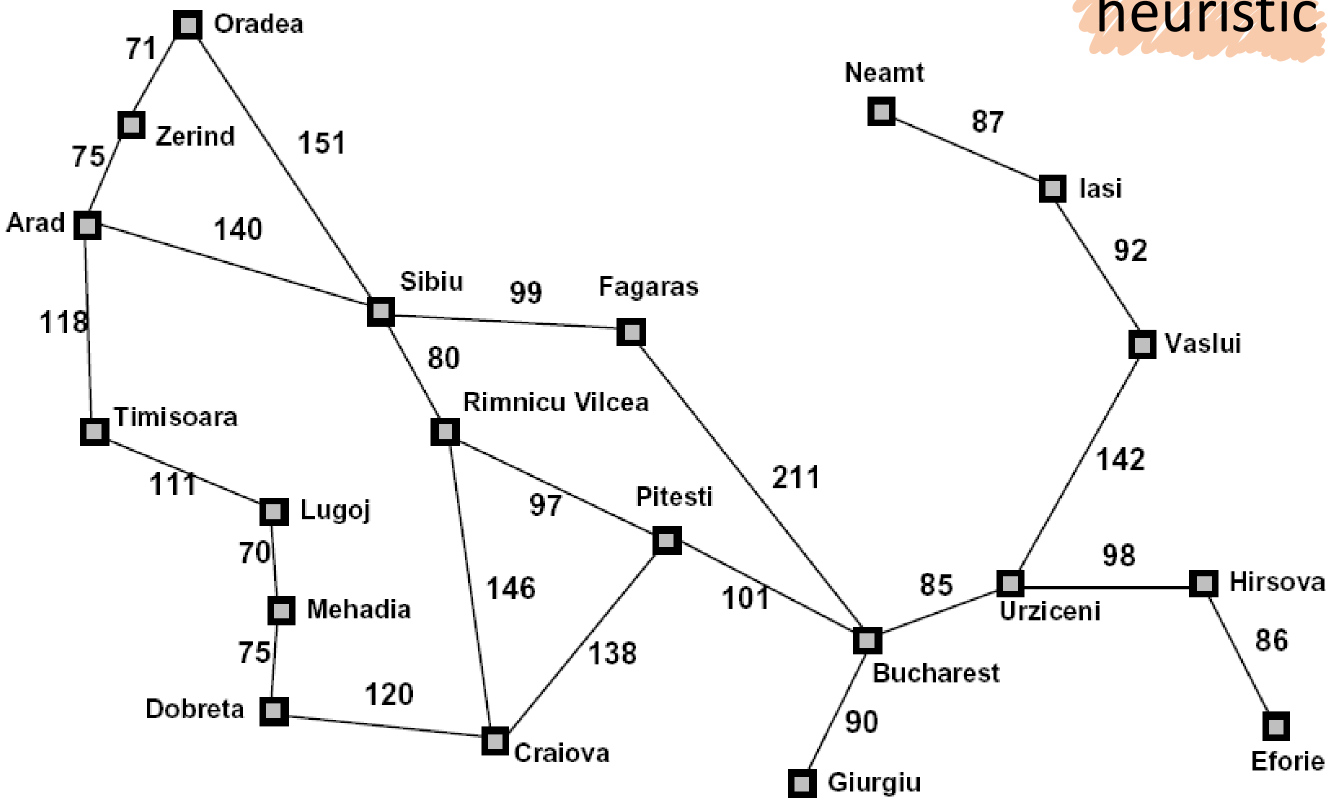
Heuristic Function

A heuristic $h(n)$ is an estimate of the cost of the cheapest path from node n to a goal node.

- $h(n)$ is arbitrary, non-negative, and problem-specific
- If n is a goal node, then $h(n) = 0$.
- $h(n)$ must be easy to compute (without search), e.g., Manhatten/Euclidean distance for pathing.

Example Heuristic

heuristic

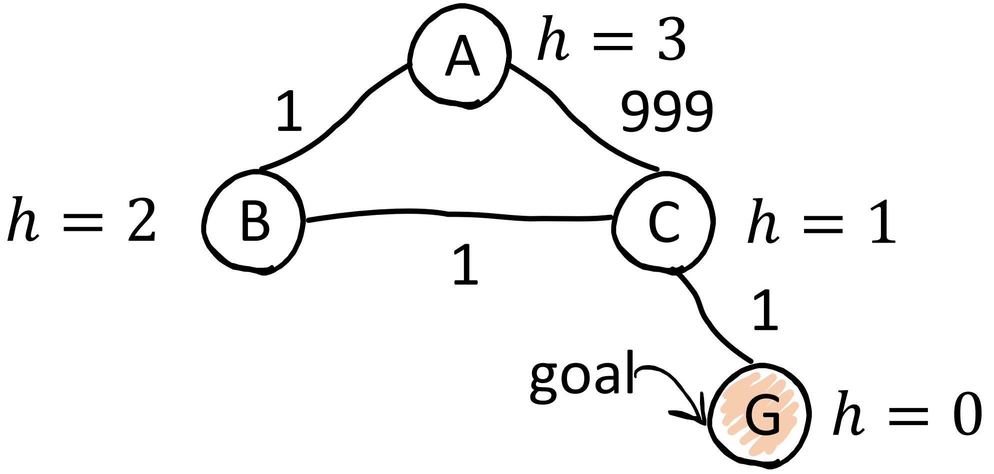


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Uniform Cost Search→Greedy Search

- Key idea: replace $g(n)$ in uniform cost search with $h(n)$, i.e., expand the node that seems closest according to $h(n)$.

- Implementation: frontier is a priority queue ordered by $h(n)$, lowest first



expand node	nodes list
	{A}
A	{C,B}
C	{G,B}
G	{B}

Obviously not optimal!

Performance of Greedy Search

- Completeness: yes, if the space graph is finite and does not contain cycles. No if there are cycles, as it gets suck in a loop due to $h(n)$.
- Optimality: no, as it may choose a sub-optimal path due to inaccurate $h(n)$.
- Time complexity: $O(b^m)$, like a badly-guided DFS.
- Space complexity: $O(b^m)$

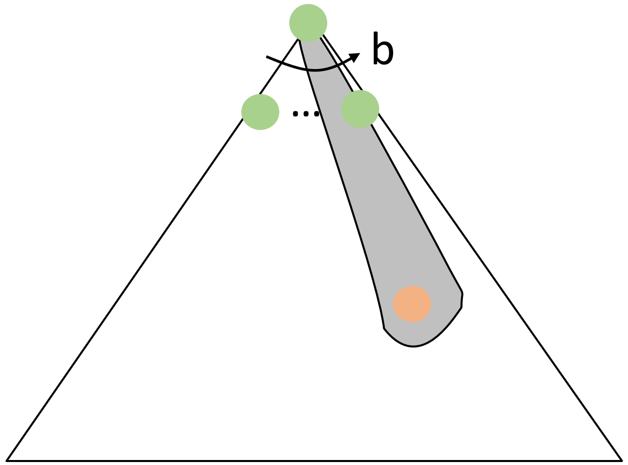
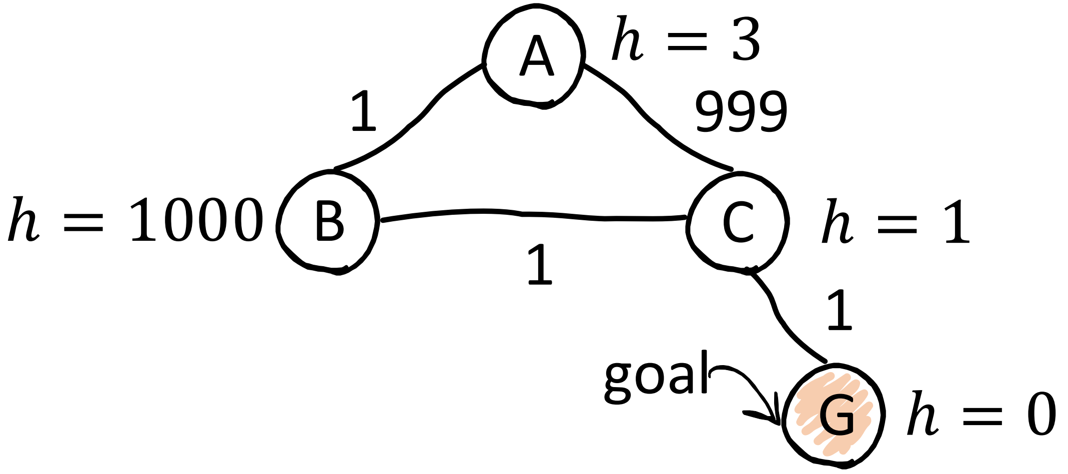


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Greedy Search→A Search

- Key idea: replace $h(n)$ in greedy search with $g(n) + h(n)$, i.e., expand the node that seems cheapest according to $g(n) + h(n)$.

- Implementation: frontier is a priority queue ordered by $g(n) + h(n)$, lowest first



expand node	nodes list
	{A}
A	{C,B}
C	{G,B}
G	{B}

Still not optimal!

A Search→A* Search

Problem

The heuristic function $h(n)$ overestimates!

A* search as the solution

Put constraints on $h(n)$ to make an admissible and optimistic heuristic!

Admissible Heuristic

- A heuristic $h(n)$ is admissible (optimistic) if

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

where $h^*(n)$ is the true cost from node n to a nearest goal.

- Theorem: If the heuristic $h(n)$ is admissible, A* search is optimal. (proof later)

Admissible Heuristic

Clicker question: For the 8-puzzle example, which of the following are admissible heuristics?

$h(n)$ = # of tiles in wrong position

$h(n) = 0$

$h(n) = 1$

$h(n)$ = sum of Manhattan distance between each tile and its goal location

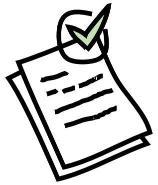
7	2	4
5		6
8	3	1

initial state

	1	2
3	4	5
6	7	8

goal state

Admissible Heuristic



Clicker question: In general, which of the following are admissible heuristics? $h^*(n)$ is the true optimal cost from n to goal.

$$h(n) = h^*(n)$$

$$h(n) = \max(2, h^*(n))$$

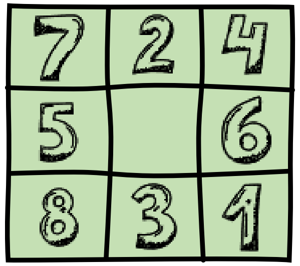
$$h(n) = \min(2, h^*(n))$$

$$h(n) = h^*(n) - 2$$

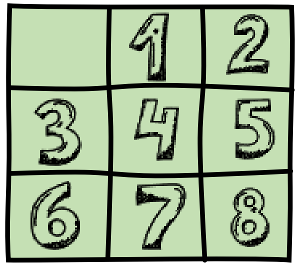
$$h(n) = sqrt(h^*(n))$$

Heuristics for Creating Admissible Heuristics

- Define a relaxed problem by simplifying or dropping requirements on the original problem.
- The relaxed problem can be solved without search.
- The cost of the optimal solution to the relaxed problem is an admissible heuristic for the original problem.



initial state



goal state

- $h(n)$ = number of tiles in wrong position
Relaxed: allow tiles to fly to their destination in one step.
- $h(n)$ = sum of Manhattan distance between each tile and its goal location
Relaxed: allow tiles to move on top of other tiles.

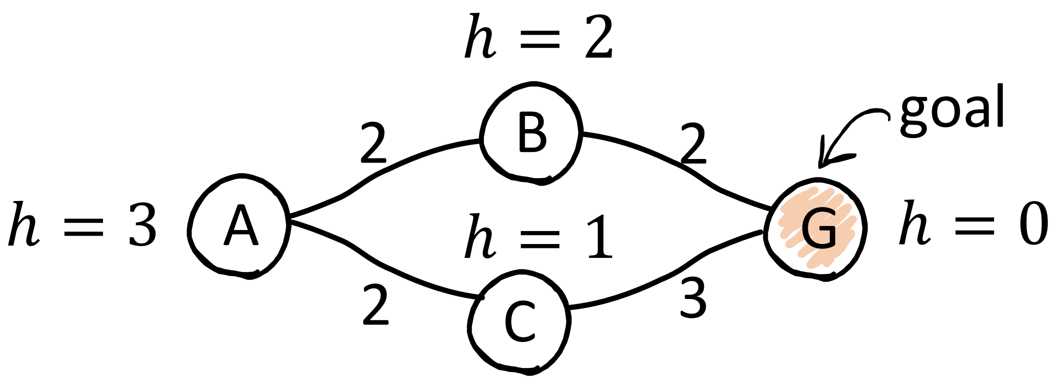
Comparing Admissible Heuristics

A heuristic function $h_2(n)$ domintes $h_1(n)$ if for all n

$$h_1(n) \leq h_2(n) \leq h^*(n).$$

- Heuristic functions as close to $h^*(n)$ as possible, but not over $h^*(n)$, are preferred.
- However, a better heuristic function, say $h_2(n)$, usually requires more complex computation. In this case, we could use a simpler and faster heuristic to spend more time on expanding more nodes.

When Should A* Stop?



Stop when we enqueue a goal

expand node	nodes list
	{A}
A	{C,B}
C	{B,G}

The path A->C->G is not optimal!

Stop when we deque a goal

expand node	nodes list
	{A}
A	{C,B}
C	{B,G}
B	{G}
G	

Performance of A* Search (vs. Uniform Cost)

Uniform cost search

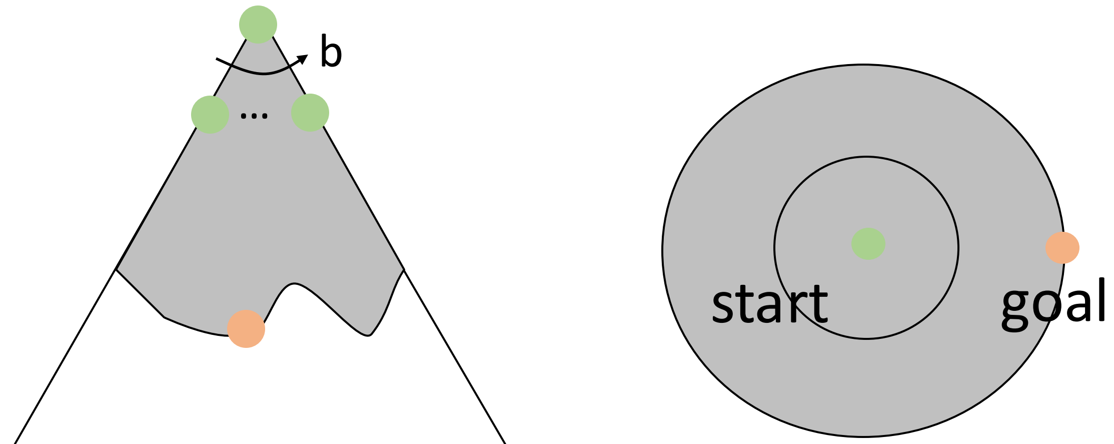
- Completeness: yes if step cost $\geq \epsilon$ ($\epsilon > 0$) and the best solution has a finite cost
- Optimality: yes
- Time complexity: # of nodes whose costs are less than that of the optimal solution $O(b^{1+\lceil \frac{C^*}{\epsilon} \rceil})$, where C^* is the cost of the optimal solution.
- Space complexity: has roughly the last tier, $O(b^{1+\lceil \frac{C^*}{\epsilon} \rceil})$

A* search

- Completeness: yes if step cost $\geq \epsilon$ ($\epsilon > 0$) and b is finite
- Optimality: yes
- Time complexity: $O(b^m)$. If the heuristic $h(n) = 0$, and the costs are the same, A* is just like breadth-first search.
- Space complexity: $O(b^m)$. A* maintains a frontier which grows with the size of the tree.

Performance of A* Search (vs. Uniform Cost)

Uniform cost search



A* search

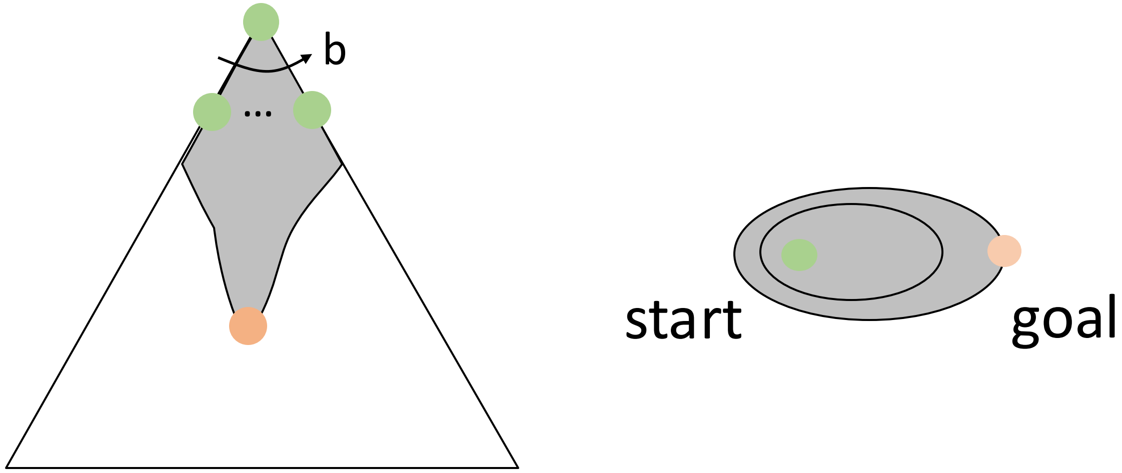


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

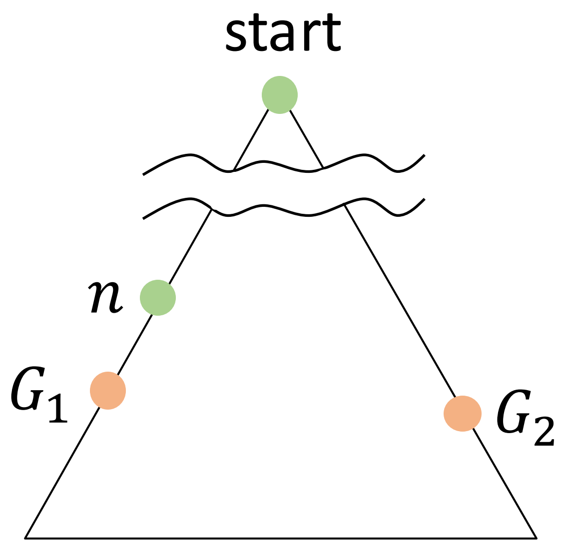
Optimality of A* Search

Proof sketch

- Assume that a sub-optimal goal node G_2 has been generated and is in the priority queue and n is an unexpanded node on a shortest path to an optimal goal G_1.
- $f(G_2) = g(G_2)$ (as $h(G_2) = 0$)

  $> g(G_1)$ (as G_2 is suboptimal)

  $\geq f(n)$ ($h(n)$ is admissible)
- A* will never select G_2 for expansion before n.
- All ancestors of G_1 expand before n.
- G_1 expands before G_2. Therefore, A* is optimal.



No Weakness?

Problem

A* consumes lots of memory $\sim O(b^m)$, especially for large problems.

Two alternative solutions

- Iterative deepening A*
- Beam search

Iterative Deepening A*

- Key idea: get DFS's space advantage by repeating DFS. Unlike iterative deepening search constrained by the maximum depth at t-th iteration, here only nodes with $f(n) \leq f_{t-1}^*$ are expanded. f_{t-1}^* is the smallest cost at last iteration.

- Implementation:

  for iteration $t = 0,1,\cdots$

  Perform a depth-first search by only expanding n with $f(n) \leq f_{t-1}^*$

- Completeness: yes

- Optimality: yes

- Time complexity: $O(b^m)$.

- Space complexity: $O(bd)$, linear!

Beam Search

- Key idea: Fixing the size of the priority queue to be k. Only keeping the top-k nodes. Beam search is general technique, not only applicable to A*.
- Implementation: frontier is a priority queue of size k ordered by $g(n) + h(n)$, discarding the largest ones.

- Completeness: no
- Optimality: no
- Time complexity: $O(b^m)$.
- Space complexity: $O(k)$, constant!

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- …

- Language analysis
- Machine translation
- Speech recognition

Goals

- Describe and formulate a search problem.
- Determine properties of different search algorithms.
- Implement both uninformed and informed search algorithms.
- Given a scenario, explain why it is appropriate or not to use a search algorithm.
- Construct admissible heuristics for appropriate problems.
- Formally prove the optimality of A*.

Important This Week

- Do some exercises on the textbook, and ask questions (optionally).
- Practice the tutorial this week for map coloring.