COMS W4701: Artificial Intelligence

Lecture 2: Search Problems

Tony Dear, Ph.D.

Department of Computer Science

School of Engineering and Applied Sciences

Today

- Search problem formulation
- State space graphs and search trees

- Uninformed search: DFS, BFS, UCS
- Informed search: Greedy, A*

- Search heuristics: Admissibility, design
- Applet for self-studying: http://www.aispace.org/search/index.shtml

Problem-Solving Agents

- Goal-based agent whose goal is a **state**, or description of the current task environment
- States contain all relevant information, can be treated as black boxes

- Environment properties: **fully observable**, **single-agent, deterministic**, **static, discrete**
  - Percepts are trivial, since we see entire environment
  - Action results always known, go from one state to another state

- Agent wants to find an *action sequence* that will result in a *state sequence* to a goal

- This is the agent's solution to a **search problem**

Search Problems

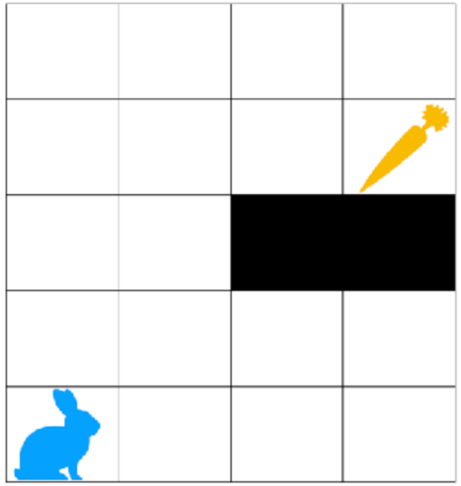
- State space S: Set of descriptions of the agent and environment
- Actions: (Finite) set of available actions in a state
  - Ex: $Actions(s_1) = \{a_1, a_2, a_3\}$

- Transition model: (Deterministic) mapping from (state, action) to a new state
  - Ex: $Result(s_1, a_1) = s_2$
- Action costs: Numerical cost for a (state, action, new state) transition
  - Ex: $Cost(s_1, a_1, s_2) = 10$

- Goal test (for goal states)
  - Ex: $IsGoal(s_1) = \text{False}$, $IsGoal(s_2) = \text{True}$

Example: Grid World Path Finding

- State space: Current coordinates of the rabbit
  - $S = \{(x, y) \mid 0 \leq x \leq 3, 0 \leq y \leq 4\}$

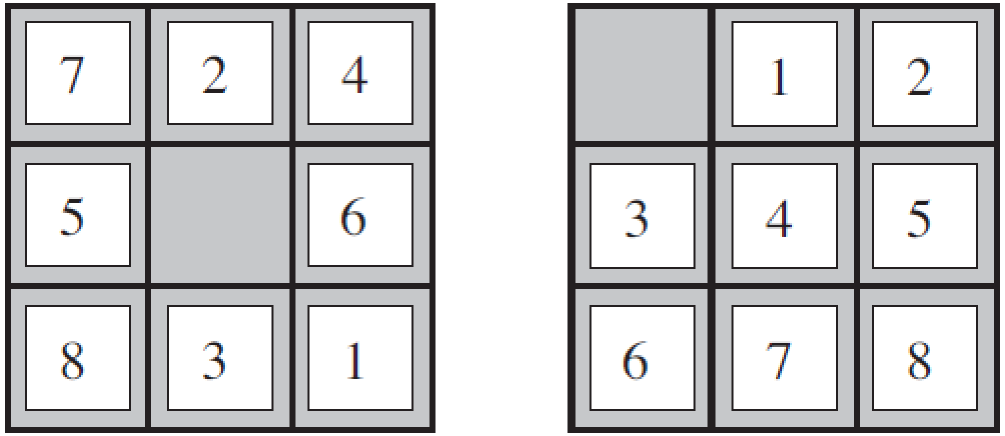


- Actions: $Actions\big((x, y)\big) = \{\mathrm{Up}, \mathrm{Down}, \mathrm{Left}, \mathrm{Right}\}$
- Costs: $Cost(s, a, s') = 1, \forall s, a, s'$

- Transition model: $Result((x, y), \text{Up}) = (x, y + 1)$, $Result((x, y), \text{Down}) = \cdots$
  - Should also account for walls and boundaries, e.g. $Result\big((0{,}0), Left\big) = (0{,}0)$

- Goal test: $In((3{,}3))$?

Search Problem Example: n-puzzle



Start State Goal State

- **State**: Locations of all tiles and blank
- **Action**: 4 possible directions for the blank tile
- **Action cost**: Each step taken costs 1
- **Goal test**: Is current state equal to goal state?

More Search Problems

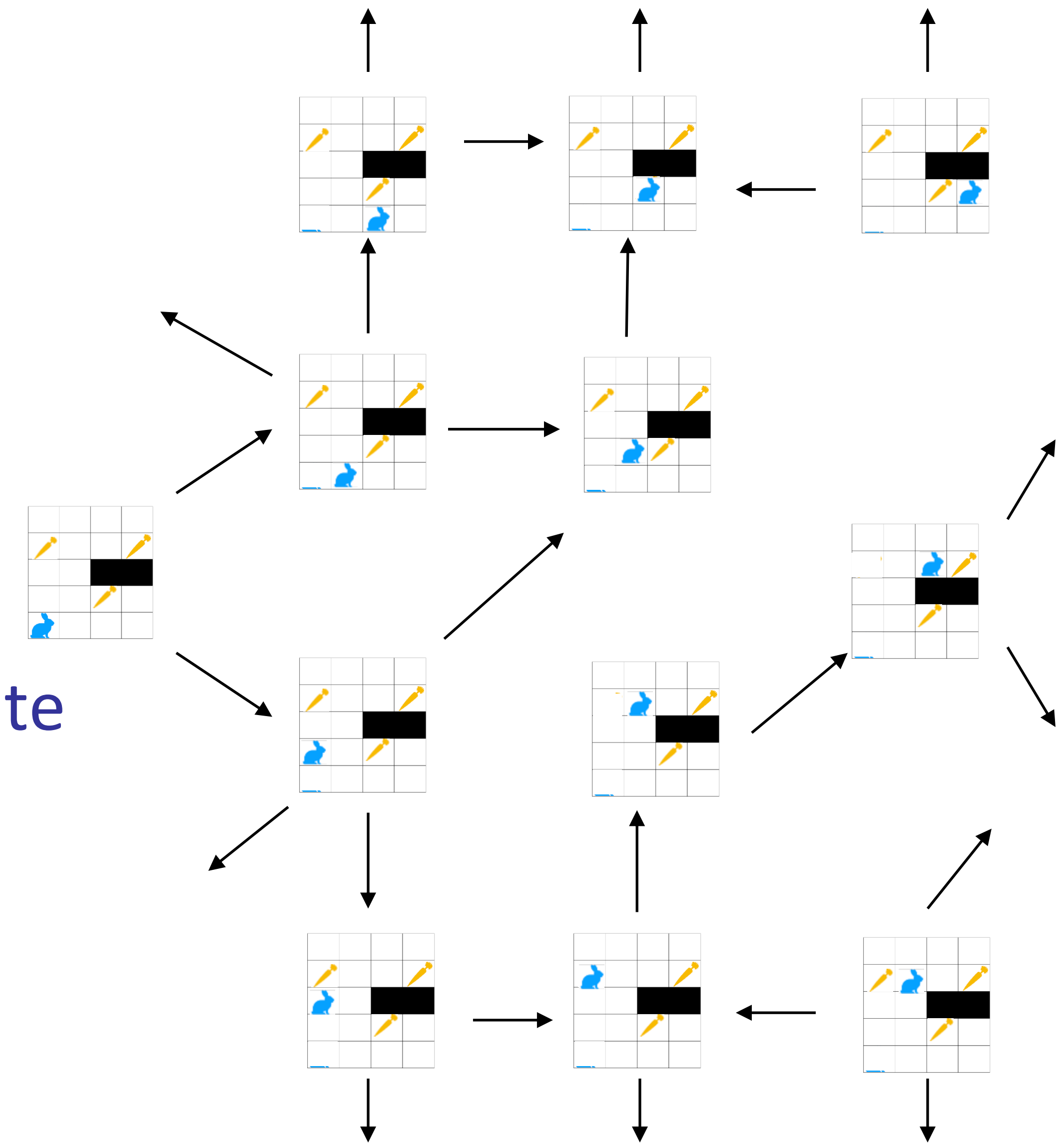
- Route-finding (e.g., vehicle navigation), robot navigation in the real world
- Touring problems (traveling salesperson)
- Layout and assembly sequencing problems

- Mathematical puzzles and proofs: Infinitely large state spaces!

- Knuth’s conjecture (1964): Starting with the number 4, use a combination of factorial, floor, and sqrt operations to reach any other desired integer
- States: All nonnegative integers

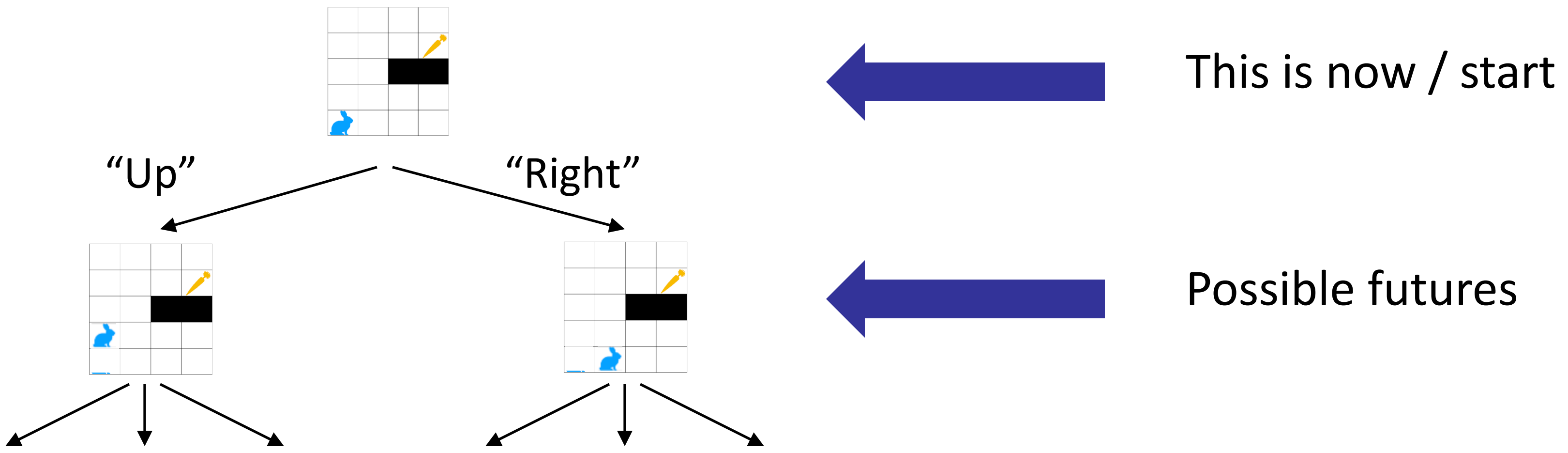
State Space Graphs

- **State space graph**: A mathematical representation of a search problem
  - *Vertices* are states; *edges* are actions
  - Each state occurs only once!
- *Paths* are sequences of actions/states
- A *solution* is a path from initial to goal state
- We can rarely build this full graph in memory—it can be very large or infinite



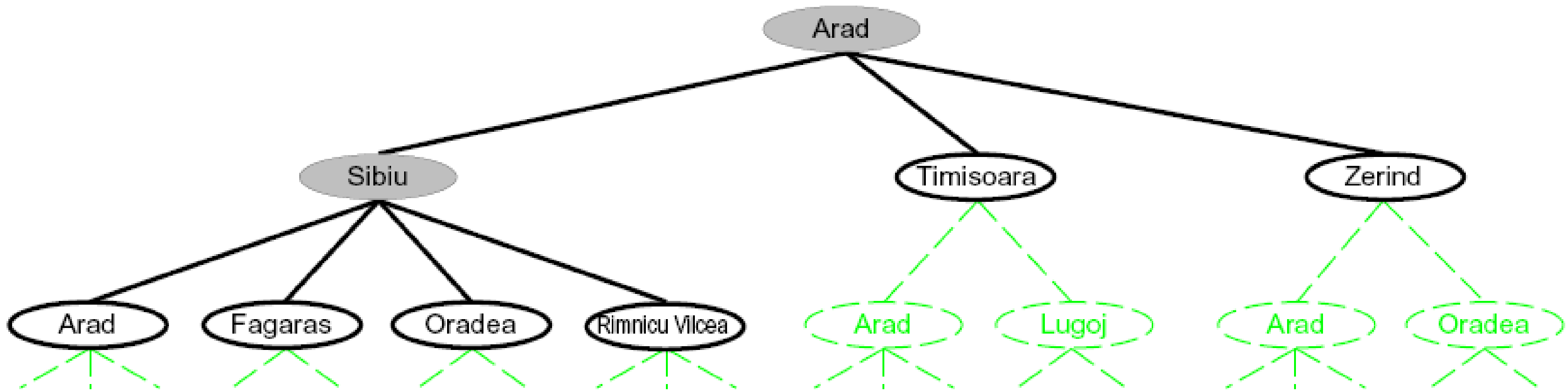
Search Trees

- Need a systematic way of performing search over a state space graph
- **Search tree**: Nodes are states, edges are actions; root is initial state



- Unlike state space graph, states can occur more than once
- Each node corresponds to a *unique* path from initial state

General Search Ideas



- From current node, **expand** and consider all possible actions
- Generate successor **nodes** for each resultant state according to transition function
  - Each node should track its corresponding state, parent, prior action, and total cost so far

- Successors are added to a **frontier** of possible next nodes to expand
- Frontier forms a boundary between explored and unexplored parts of tree

Node Expansion

- We still have some unanswered questions re: node expansion…

- How to select the next node from the frontier?
- **Best-first search**: Implement frontier as a priority queue; each node is assigned a priority according to an *evaluation function* $f(n)$ (lowest priority node popped first)
  - **Uninformed search**: $f(n)$ has no knowledge about how close a state is to goal

- Suppose we expand a node and a child node has already been expanded…
- We may want to consider it again if this new occurrence is through a cheaper path!
- Idea: Keep track of all *reached* nodes in a lookup table

Best-First Search

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s′ ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s′)
    yield NODE(STATE=s′, PARENT=node, ACTION=action, PATH-COST=cost)
```

Depth-First Search

- Idea: Expand the *deepest* node in the frontier (relative to start node)

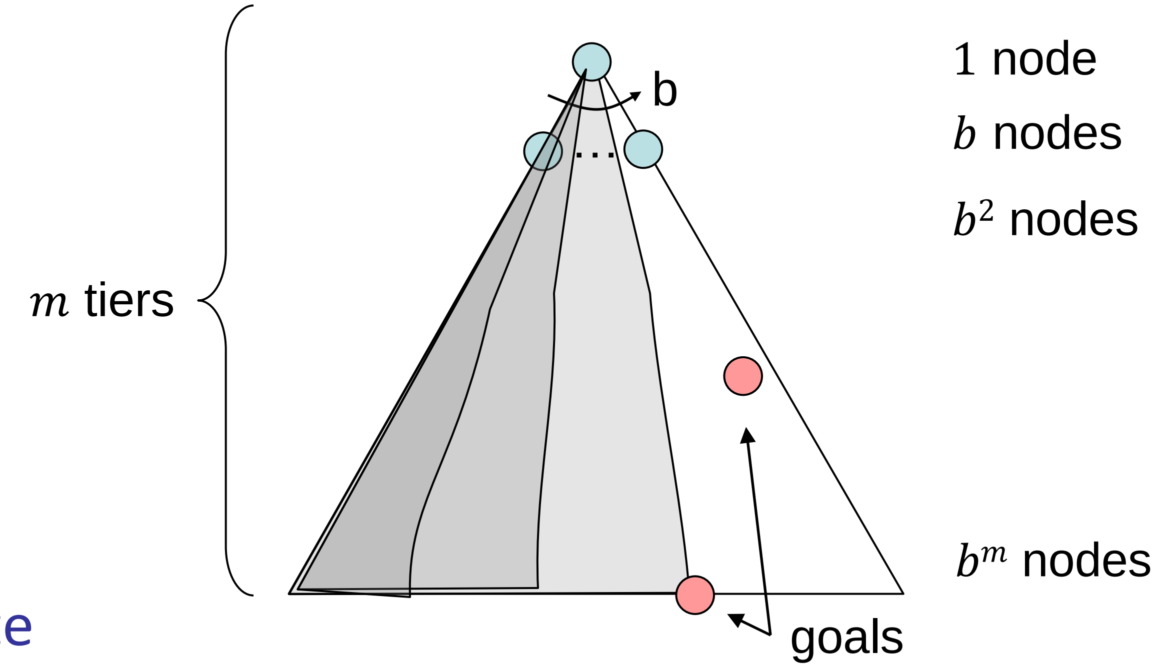
- Node priority can be implemented as the *negative of depth*
- Alternatively, the frontier can be implemented as a stack (LIFO)

- No consideration of true costs! DFS only "cares" about depth information

- Possible optimizations from best-first search base implementation:
- *Early goal test*: Check if node is goal upon insertion into frontier (instead of removal)
- Do not use a reached table, possibly exploring a node more than once

DFS Properties

- **Time complexity**: How many nodes to explore in the worst case? $O(b^m)$
- **Space complexity**: How many frontier nodes to keep in memory? $O(bm)$
  - Assumes no need for "reached" table
- **Completeness**: Not if state space is infinite
- **Optimality**: No, only returns first solution



- b is the *branching factor*
- m is the *maximum depth*
- Total nodes: $O(1 + b + b^2 + \cdots + b^m)$

Breadth-First Search

- Idea: Expand the *shallowest* node in the frontier (relative to start node)
- Node priority is exactly equal to the node *depth*
- Alternatively, the frontier can be implemented as a FIFO queue

- No consideration of true costs! BFS only “cares” about depth information
- BFS may be optimal if true costs are equivalent to depths

- Possible optimizations from best-first search base implementation:
- *Early goal test*: Check if node is goal upon insertion into frontier (instead of removal)
- Reached table can just be a set of states (no need to track costs)

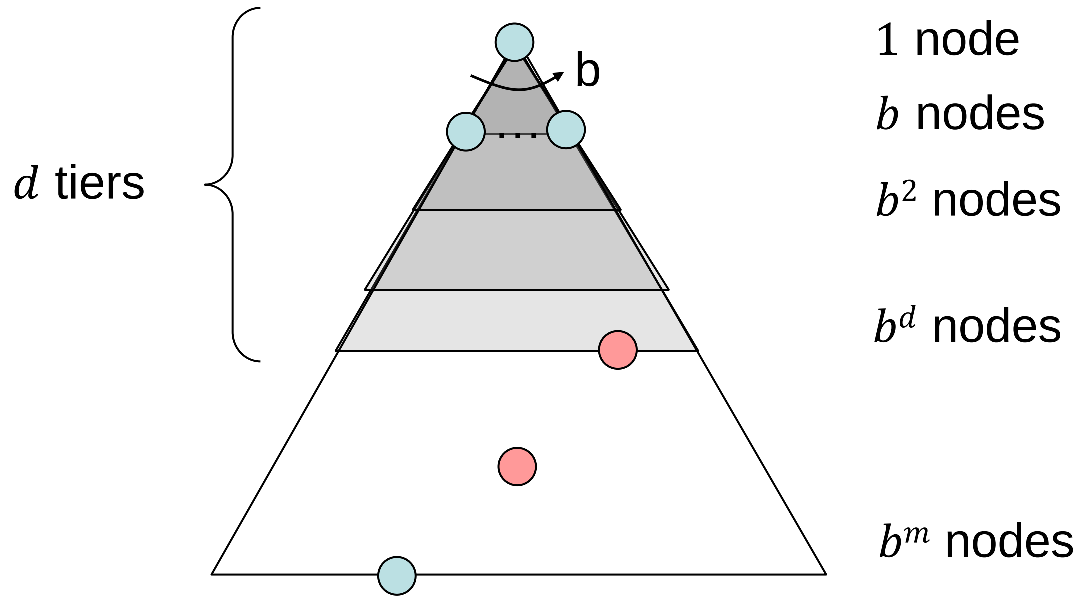
BFS Properties

- **Time complexity**: How many nodes to explore in the worst case? $O(b^d)$

- **Space complexity**: How many frontier nodes to keep in memory? $O(b^d)$

- **Completeness**: If solution exists, yes!

- **Optimality**: Only if costs are uniform

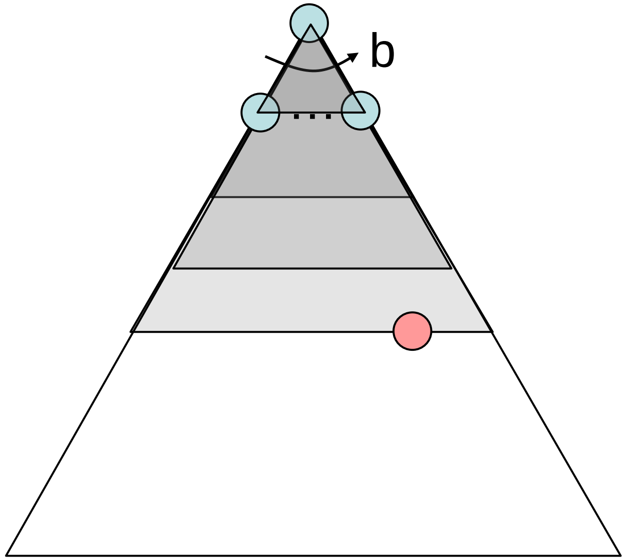


- d is depth of the shallowest solution
- May be significantly smaller than m
- Max frontier size is $O(b^d)$

Improving DFS and BFS

- **Depth-limited DFS**: Prevent DFS from going past a set depth l
- Time complexity $O(b^l)$, space complexity $O(bl)$
- Best if we know *diameter* of state space in advance

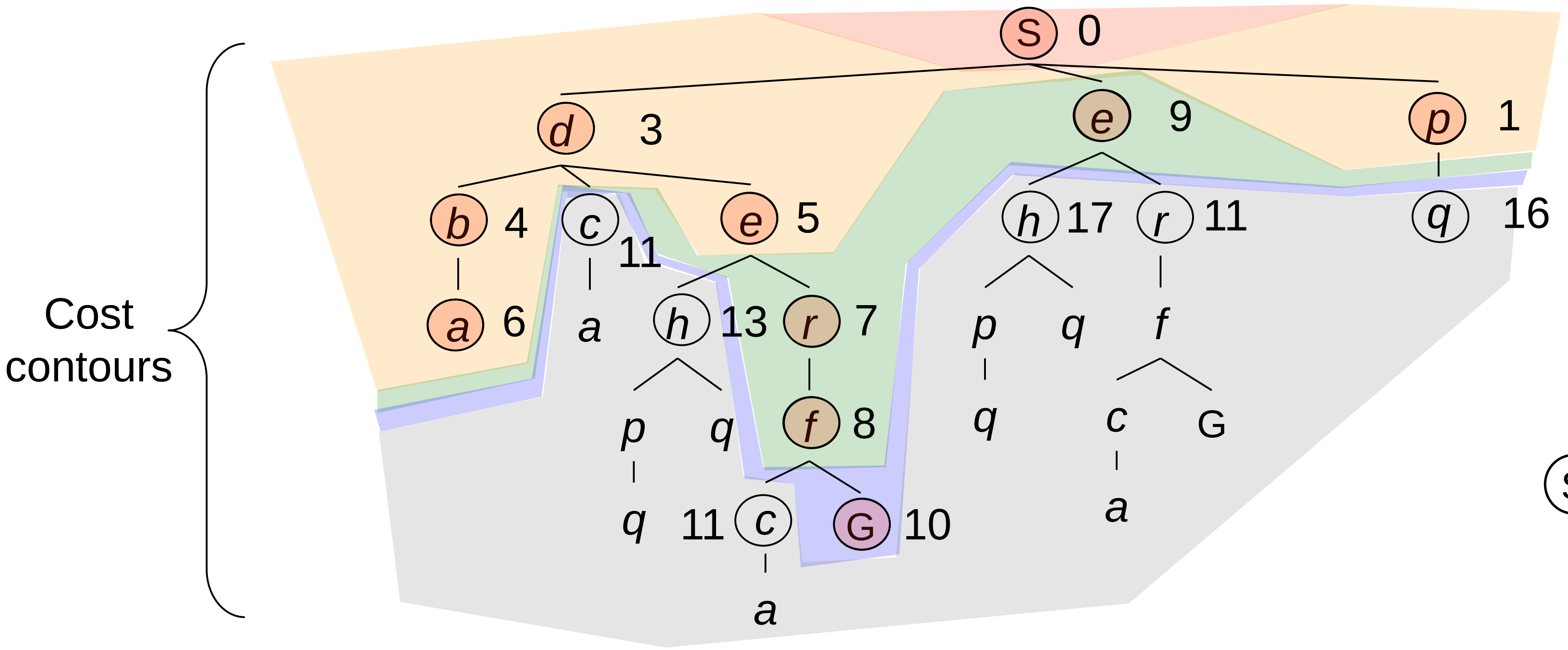
- **Iterative-deepening:** Iteratively do depth-limited search with increasing l: try $l = 0$, then $l = 1$, …
- Ends when l reaches d (depth of shallowest solution)
- Time complexity $O(b^d)$, space complexity $O(bd)$

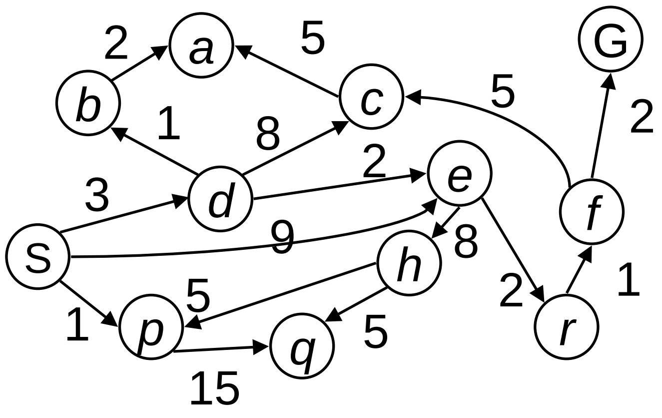


- Why is wasted effort in upper levels of search tree not a concern?

Uniform-Cost Search (Dijkstra)

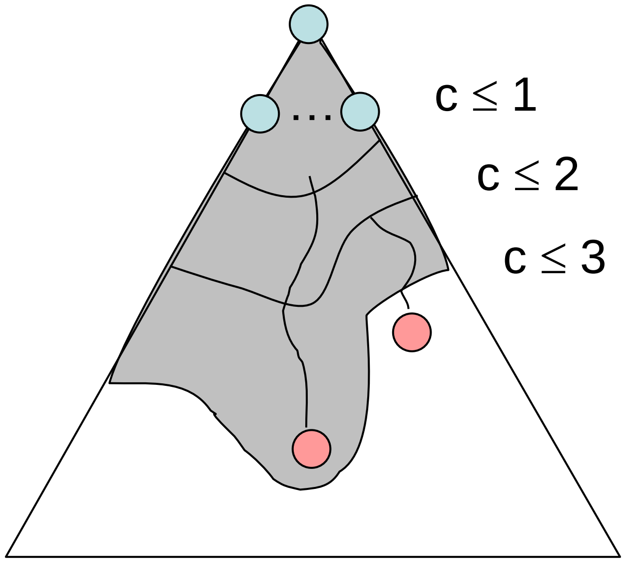
- Idea: Expand node that least increases total cost
- Evaluation function $f(n)$ is the *cumulative path cost*
- For UCS to be correct, we must use a reached table and conduct a late goal test

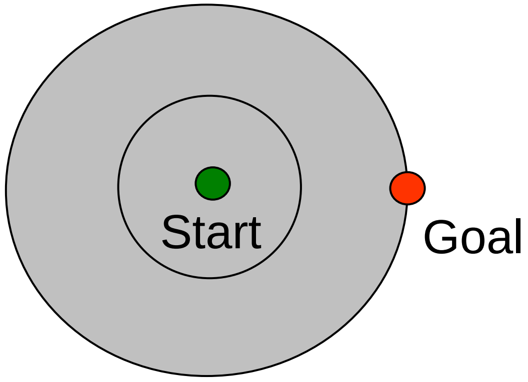




UCS Properties

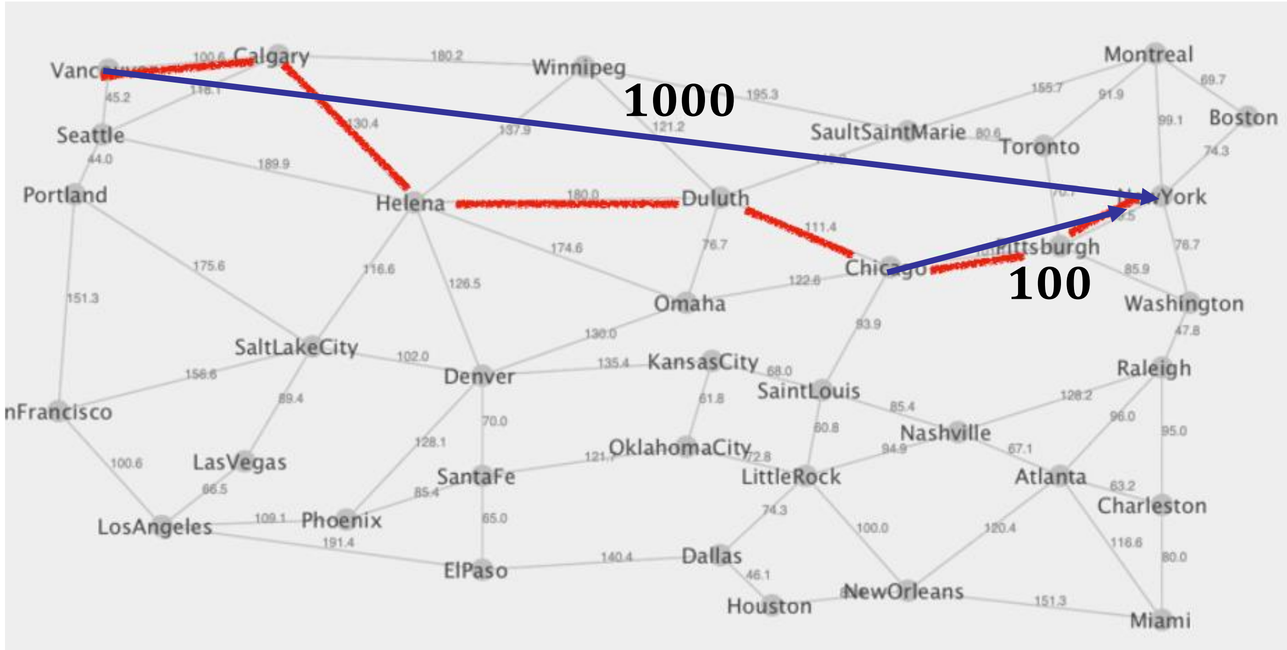
- Let C^* be the cost of optimal solution
- Let ϵ be lower bound on all possible costs
- $1 + C^*/\epsilon$ is the max depth to traverse before finding optimal solution
- **Time and space complexity**: $O(b^{1+C^*/\epsilon})$
- UCS is both **complete** and **optimal**





Informed (Heuristic) Search

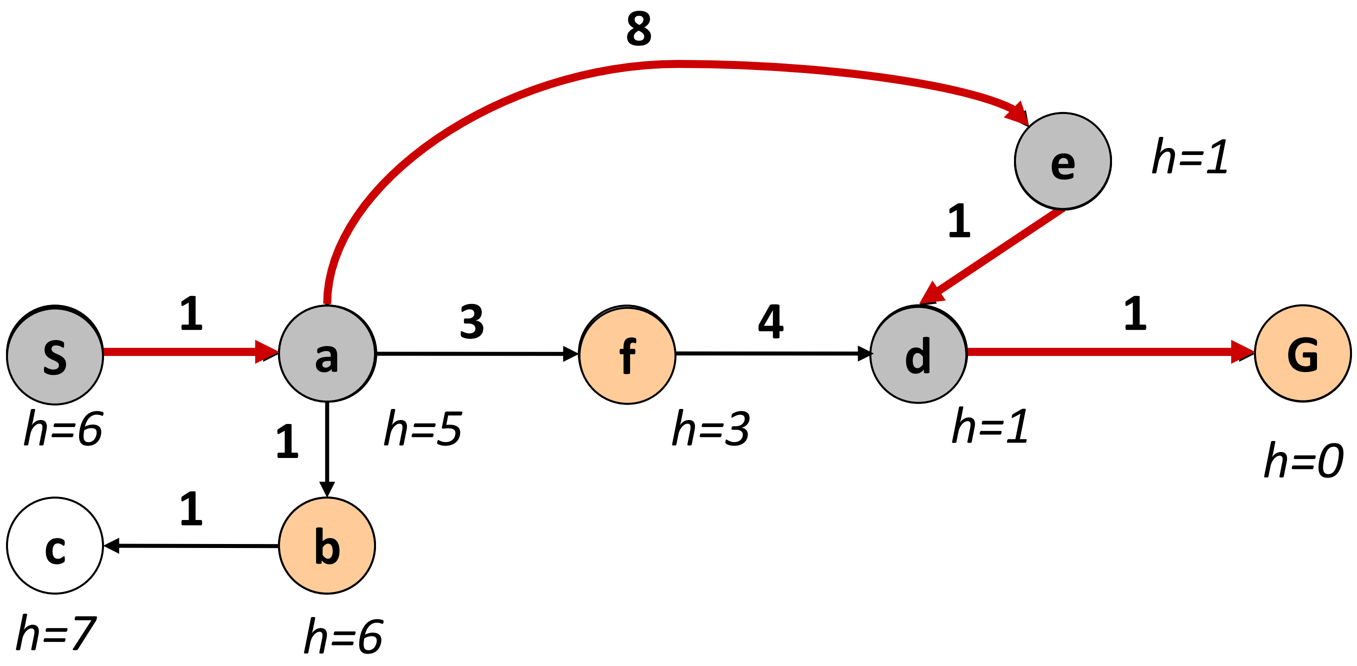
- Oftentimes we have additional, *domain-specific* **heuristics** that tell us how close a state is to a goal
- **Heuristic function** $h(n)$: Estimated cost of cheapest path from state at node n to a goal state
- Often come from *relaxed problems*, precomputed *subproblem* solutions, or learning from experience



Example: Euclidean or Manhattan distance on a map

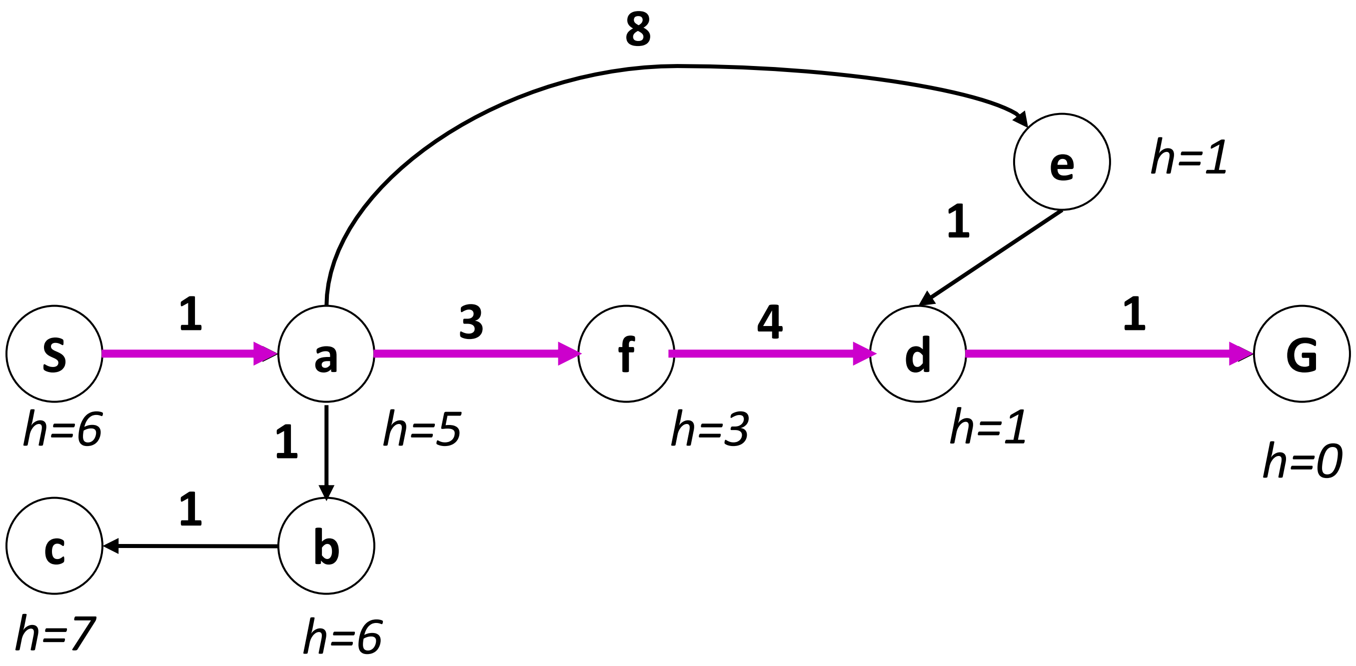
Greedy Best-First Search

- Idea: Expand node that *appears* closest to goal according to heuristic function
- Evaluation function is just the heuristic function! $f(n) = h(n)$
- As with DFS and BFS, there is no consideration of *true* costs



A* Search

- Idea: From a given node, estimate the *best* path that *continues* to the goal
- $f(n) = g(n) + h(n)$: Sum of path cost to n and estimated cost from n to goal
- Benefits of both UCS and greedy best-first search

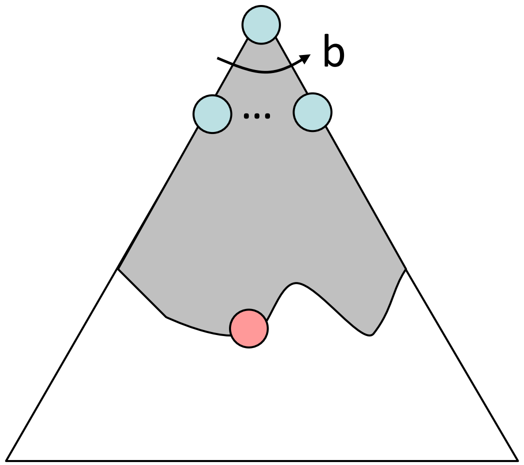


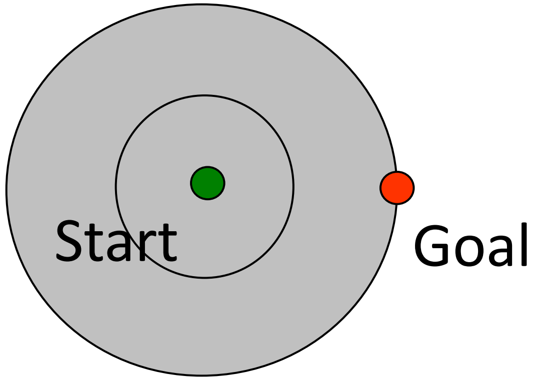
Node	$f(n) = g(n) + h(n)$
S	$6 = 0 + 6$
a	$6 = 1 + 5$
b	$8 = 2 + 6$
e	$10 = 9 + 1$
f	$7 = 4 + 3$
d	$9 = 8 + 1$
c	$10 = 3 + 7$
G	$9 = 9 + 0$

A* vs UCS vs BFS

- BFS expands search tree by increasing depth

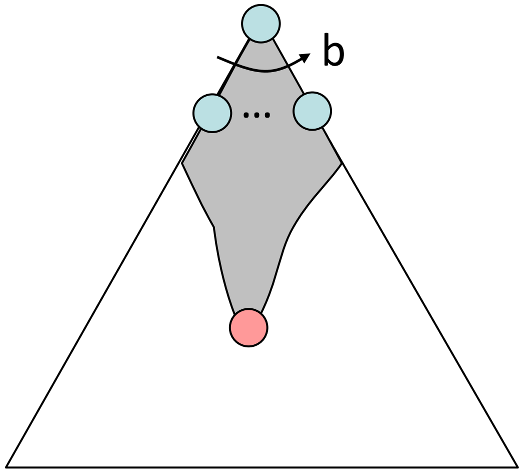
- UCS expands acc. to increasing g-cost
- Contours are "circular" around start state, if normalized by path costs

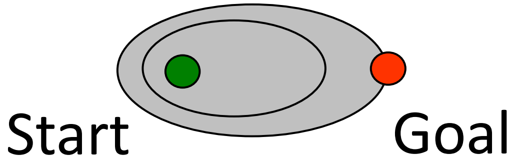




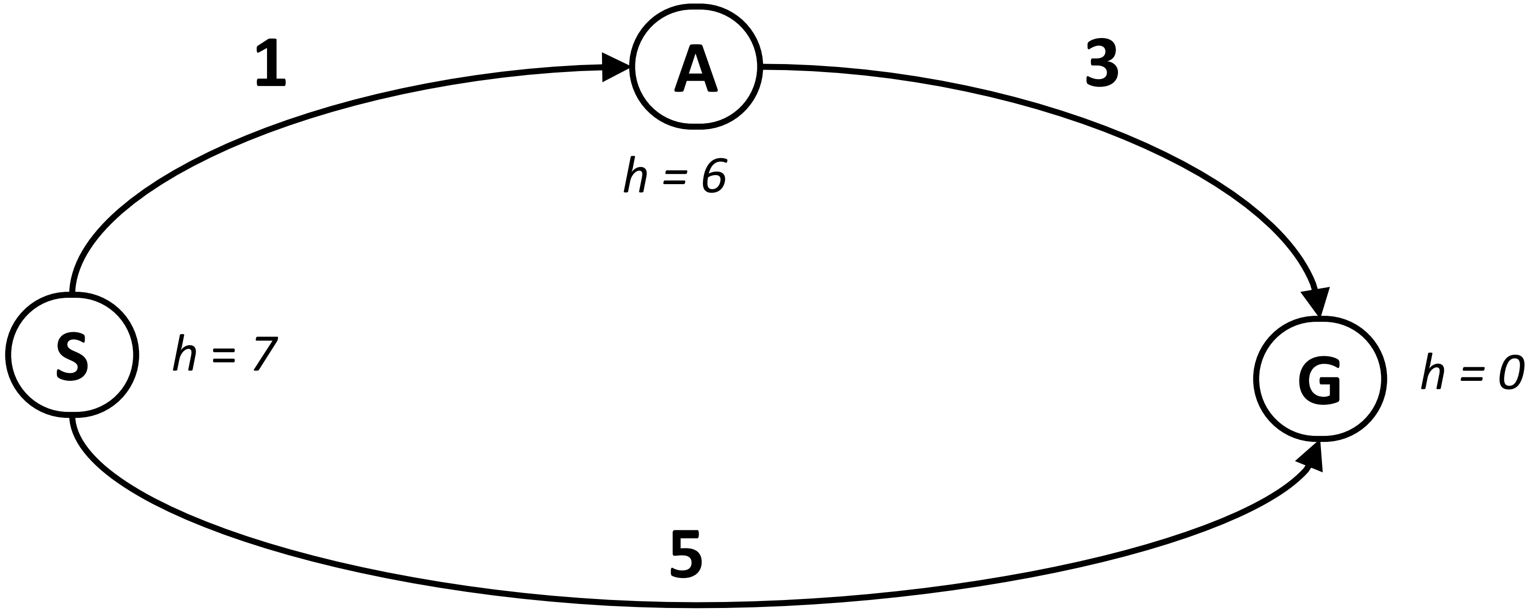
- A* expands acc. to increasing $g + h$ cost

- If heuristic is good, expanded states should show preference toward goal





When is A* Optimal?



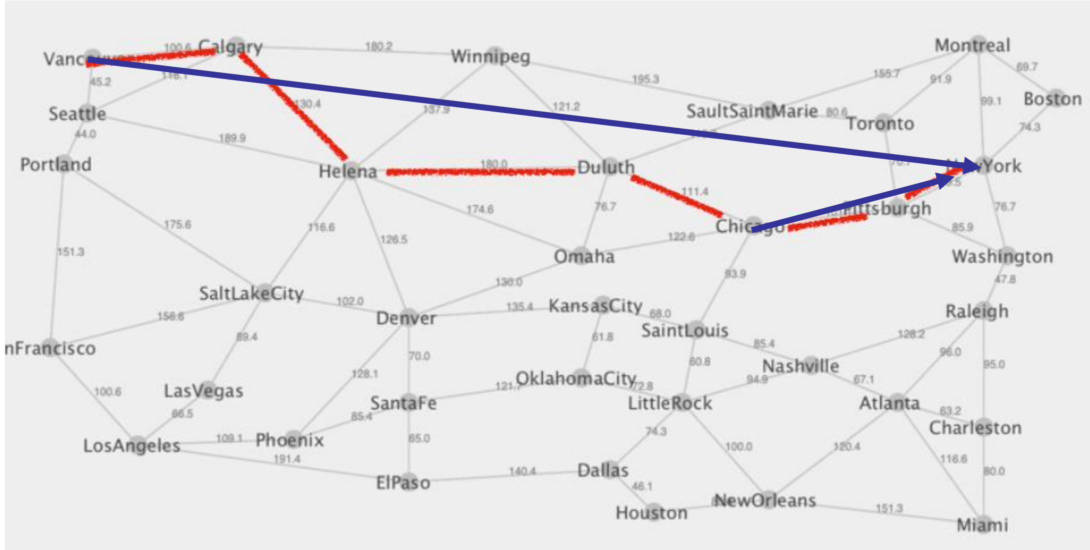
- What is the problem here?
- Heuristic along optimal path overestimated the true cost!
- Good heuristics should be optimistic—never overestimate true costs

Admissible Heuristics

- A heuristic h is **admissible** if $0 \leq h(n) \leq h^*(n)$ where $h^*(n)$ is true cost from n to goal

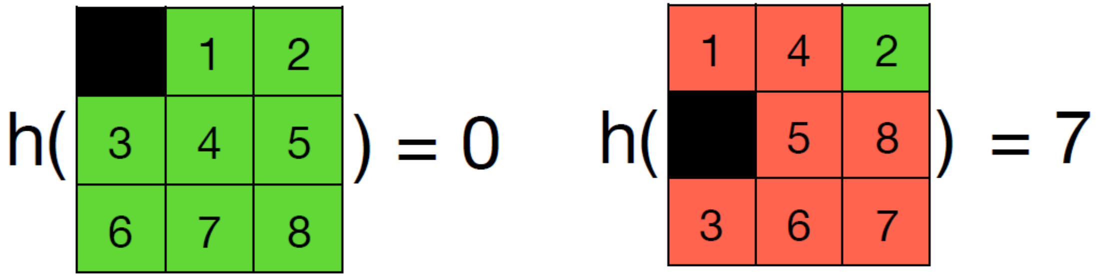
- Most heuristics derived from relaxed problems are admissible
- Same state space graph, but with added edges
- With fewer constraints or restrictions, problems are easier to solve

- Example: Euclidean distances



Misplaced Tiles Heuristic

- $h(n) =$ number of misplaced tiles, not including blank



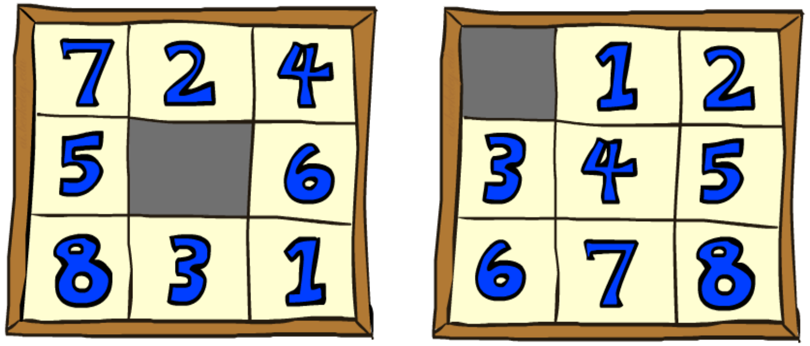
- Relaxed problem: Any tile can be correctly replaced with just one move

- Admissible because misplaced tiles will always require *at least* one move

Manhattan Distance Heuristic

- $h(n)$ = sum of Manhattan distances between current tile positions and goal positions

$$h(start) = 18$$



Start State Goal State

- Relaxed problem: Multiple tiles can simultaneously occupy same space
- Admissible because misplaced tiles will always require *at least* number of moves equal to Manhattan distance

Heuristic Domination

- For any node n, Manhattan distance heuristic $h_2(n)$ > misplaced tiles heuristic $h_1(n)$
- h_2 **dominates** h_1 if $h_2(n) \geq h_1(n)$ for all n

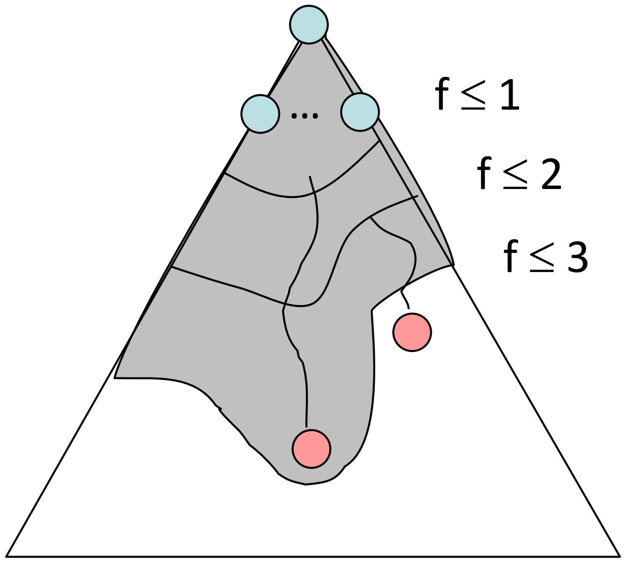
- A* search using h_2 will be more efficient and never expand more nodes than h_1
- h_2 reflects true costs more accurately

- Suppose we have collection of admissible heuristics $h_1, h_2, \ldots, h_m$
- The composite heuristic $h(n) = \max\{h_1(n), \ldots, h_m(n)\}$ is admissible and dominates all other heuristics!

Completeness and Optimality of A*

- A* is complete (same reason as UCS, BFS)
- If heuristic function is admissible, A* is also optimal!

- Suppose A* returns a suboptimal solution
- Then there exists some unexpanded node n on optimal path
- Since n was not expanded, $f(n) > C^*$ (optimal cost)



- By definition, $f(n) = g(n) + h(n)$: backward cost + heuristic
- Since h is admissible, $h(n) \leq h^*(n)$ (true cost-to-go), and so $f(n) \leq g(n) + h^*(n)$
- But $C^* = g(n) + h^*(n)$, meaning that $f(n) \leq C^*$. Contradiction!

Other A* Points

- Performance of A* depends entirely on choice of heuristic function

- Assuming an admissible heuristic, worst case time and space complexity is the same as UCS (all heuristics equal to 0)
- Otherwise, we can only improve from UCS

- Heuristics and the use of domain knowledge make A* distinctly more “intelligent” than its uninformed counterparts
- While still hard theoretically, problems become easier in practice

Satisficing Solutions*

- Like BFS or UCS, A* may suffer computationally intractable memory requirements
- Idea: Trade off admissibility for more accurate heuristics to reduce computation
- Return **satisficing solutions**—suboptimal, but "good enough"

- **Weighted A* search**: $f(n) = g(n) + \alpha h(n)$
- We can choose to place higher weight α on the heuristic
- Generalizes A* ($\alpha = 1$), UCS ($\alpha = 0$), and greedy best-first ($\alpha = \infty$)

- Suboptimality: If optimal solution has cost C^*, weighted A* solution may cost up to αC^*

Memory-Bounded Search*

- We can also consider A* variants that are more memory-efficient

- **Beam search**: Limit frontier size by discarding worst nodes past a given limit
- Alternatively, discard nodes with scores much smaller than best one

- **Iterative-deepening A*** (IDA*): Repeatedly run A* with increasing depth limit
- Nodes with higher f-cost than limit are treated as leaves
- Increment depth limit by smallest f-cost of "leaves" from previous iteration

- IDA* worst case: Each node has different f-cost, num iterations equal to num states

Summary

- Objective of search problems is to find action/state sequence to reach a goal state
- Represented by state space graphs; search algorithms follow a tree structure

- Uninformed search: No usage of information indicating closeness to goal
- Examples: Depth-first, breadth-first, depth-limited, iterative deepening, uniform-cost
- Generally suffer from lack of completeness or intractable memory usage

- Informed search: Domain-specific heuristics guide search toward goal
- Greedy best-first and A* search use a heuristic function to evaluate frontier nodes
- Optimal if heuristics are admissible: good, optimistic estimates of true costs