A* SEARCH

CSE 511A: Introduction to Artificial Intelligence

UNINFORMED SEARCH

Generic uninformed search pseudo-code:

(1) Start with a tree that contains only the start state
(2) Pick an unexpanded fringe node n
(3) If fringe node n represents a goal state, then stop
(4) Expand fringe node n*
(5) Go to (2)

*generate neighboring nodes that aren't ancestors

BREADTH-FIRST SEARCH

Breadth-first search pseudo-code:

(1) Start with a tree that contains only the start state
(2) Pick an unexpanded fringe node n **with the smallest depth**
(3) If fringe node n represents a goal state, then stop
(4) Expand fringe node n*
(5) Go to (2)

*generate neighboring nodes that aren't ancestors

DEPTH-FIRST SEARCH

Depth-first search pseudo-code:

(1) Start with a tree that contains only the start state
(2) Pick an unexpanded fringe node n **with the largest depth**
(3) If fringe node n represents a goal state, then stop
(4) Expand fringe node n*
(5) Go to (2)

*generate neighboring nodes that aren't ancestors

UNIFORM-COST SEARCH

Uniform-cost search pseudo-code:

(1) Start with a tree that contains only the start state
(2) Pick an unexpanded fringe node n **with the smallest g(n)**
(3) If fringe node n represents a goal state, then stop
(4) Expand fringe node n*
(5) Go to (2)

*generate neighboring nodes that aren't ancestors

GREEDY BEST-FIRST SEARCH

Greedy best-first search pseudo-code:

(1) Start with a tree that contains only the start state
(2) Pick an unexpanded fringe node n **with the smallest h(n)**
(3) If fringe node n represents a goal state, then stop
(4) Expand fringe node n*
(5) Go to (2)

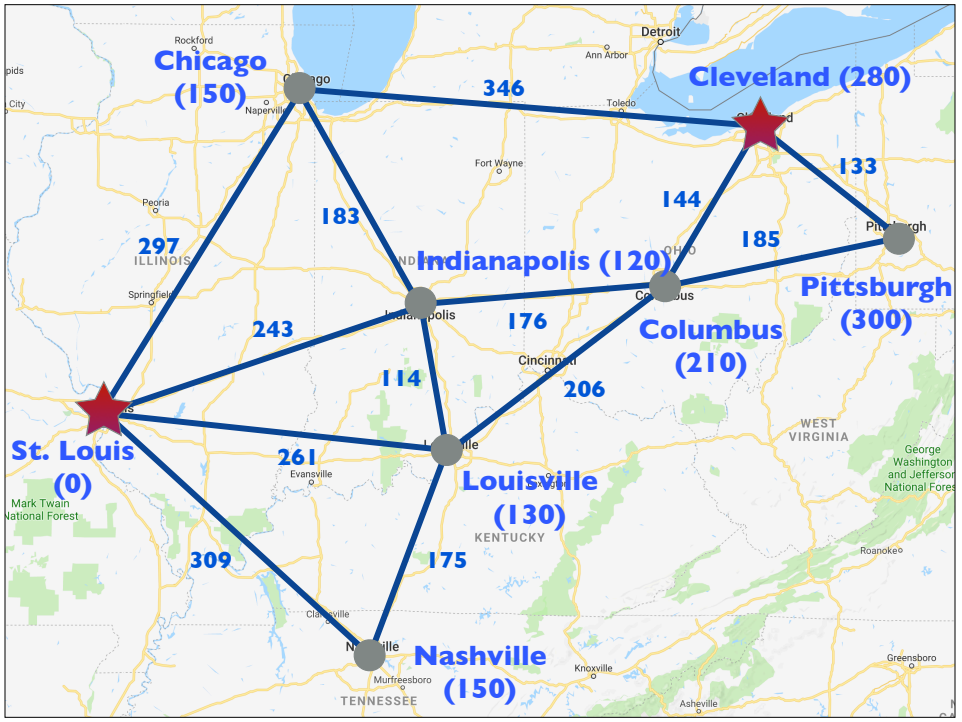
*generate neighboring nodes that aren't ancestors

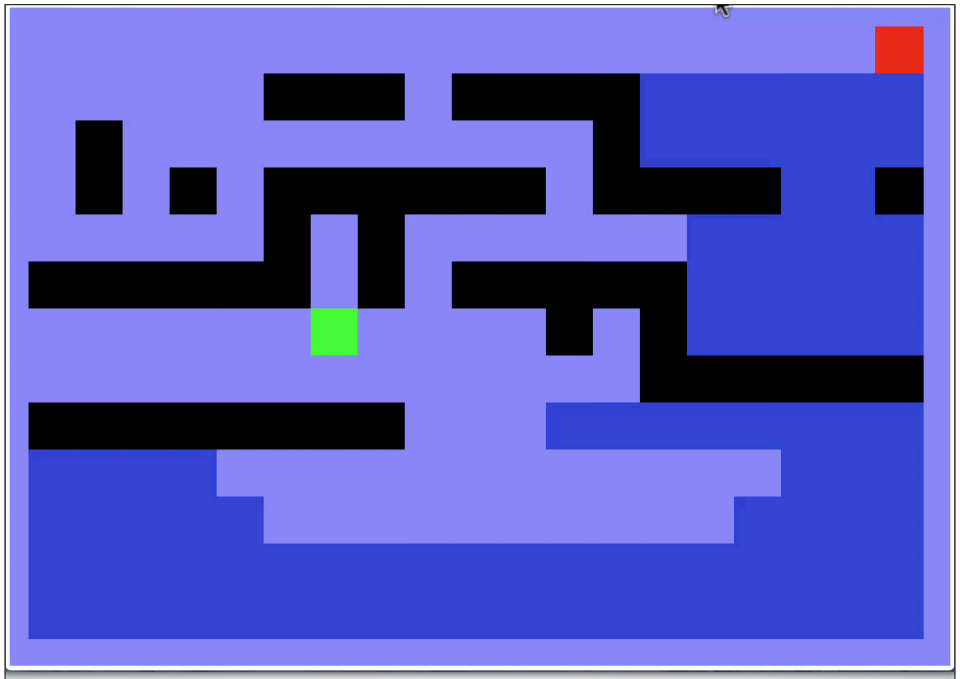
A* SEARCH

A* search pseudo-code:

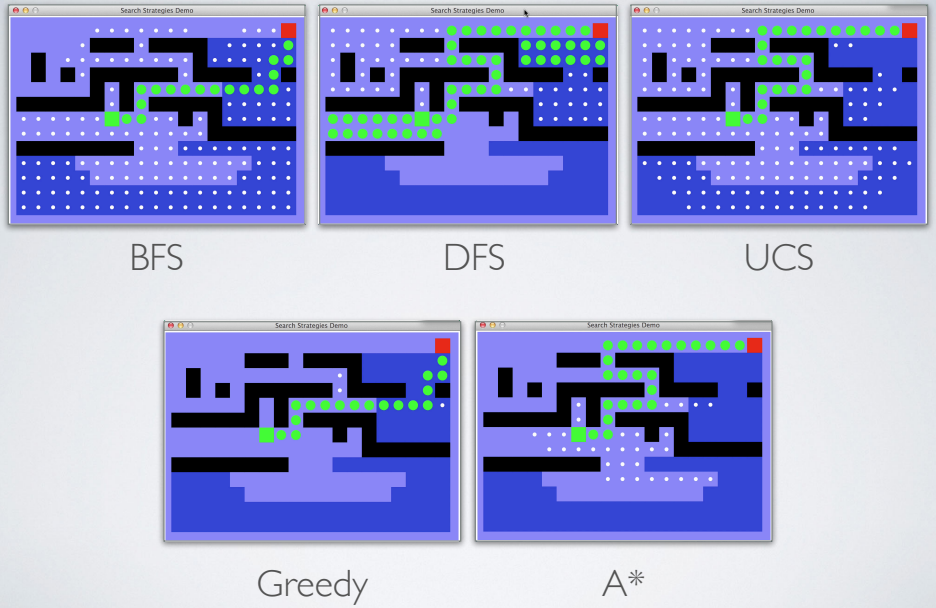
(1) Start with a tree that contains only the start state
(2) Pick an unexpanded fringe node n **with the smallest g(n) + h(n)**
(3) If fringe node n represents a goal state, then stop
(4) Expand fringe node n*
(5) Go to (2)

*generate neighboring nodes that aren't ancestors





Videos are from slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.



Videos are from slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.

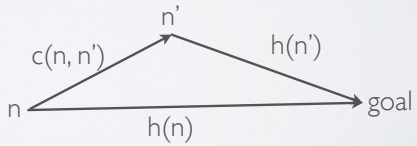
HEURISTICS

- Heuristics are **admissible** if they do not over-estimate the true distances

  $h(n) \leq c(n, goal)$

- Heuristics are **consistent** if they satisfy the triangle inequality

  $h(n) \leq c(n, n') + h(n')$
  $h(goal) = 0$



HEURISTICS

- Heuristics need to be computed very quickly
- Typically computed by relaxing the problem: add some states or actions.
- Examples:
  - Zero heuristics
  - Straight-line heuristics
  - Manhattan heuristics

PROPERTIES OF A*

- Consistent heuristics ⇒ Admissible heuristics
- Consistent heuristics ⇍ Admissible heuristics

PROPERTIES OF A*

- Consistent heuristics ⇒ Admissible heuristics
- Consistent heuristics ⇍ Admissible heuristics

- If A* uses consistent heuristics, then it is correct

PROPERTIES OF A*

- Consistent heuristics ⇒ Admissible heuristics
- Consistent heuristics ⇍ Admissible heuristics

- If A* uses consistent heuristics, then it is correct
- If A* uses admissible but inconsistent heuristics, then it is correct if it re-expand states

PROPERTIES OF A*

- If A* uses consistent heuristics, then it is efficient

  No other search algorithm that uses the same heuristics, which is correct and complete, can expand fewer nodes than A* (except for breaking ties at nodes n with f(n) = f(goal))

- If $h_1(n) \geq h_2(n)$ for all states n, then A* using $h_1(n)$ is at least as efficient as using $h_2(n)$

  A* using $h_1(n)$ expands at most the same number of states as A* using $h_2(n)$ (except for breaking ties at nodes n with f(n) = f(goal))

PROPERTIES OF A*

- Space and time complexities are like those for UCS
  - $O(b^{\lceil C^*/\varepsilon \rceil + 1})$ cost of optimal solution C^*, cost increment ε
- Two general approaches to use less memory:
  - Memory-bounded A* searches (e.g., Iterative-Deepening A*)
  - Local search

ITERATIVE DEEPENING DFS

Iterative Deepening DFS pseudo-code:

(1) *limit* = 0
(2) Run DFS with depth limit = *limit*
(3) If found goal, then stop
(4) *limit* = *limit* + 1
(5) Go to (2)

ITERATIVE DEEPENING A*

Iterative Deepening A* pseudo-code:

(1) *limit* = 0
(2) Run DFS with f-value limit = *limit*
(3) If found goal, then stop
(4) *limit* = $\min_n f(n)$, where $f(n) >$ *limit*
(5) Go to (2)

OTHER MEMORY-BOUNDED ALGORITHMS

- A bunch of other algorithms:
  - Recursive best-first search (RBFS)
  - Memory-bounded A* (MA*)
  - Simplified MA* (SMA*)
- Descriptions are in the textbook