Artificial Intelligence

Jeff Clune
Assistant Professor
Evolving Artificial Intelligence Laboratory

University of Wyoming
AI Challenge One

- Was it hard?

- Now imagine beating Kasparov, playing Jeopardy, or driving 100,000 miles on city/highway roads!
AI Challenge One
CS Picnic

Computer Science
Annual Picnic!
Saturday September 19th from 3-6pm
at Washington Park!
AI Challenge Two!

• Due Sept. 20th
  • < one week!
Repeated States

- Visiting a state more than once can lead to infinite loops.
- "algorithms that forget their history are doomed to repeat it"
- Solution: don't expand states already visited
  - Use a hash table (good implementation = constant time)
Graph Search (vs. Tree)

function `TREE-SEARCH(problem)` 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 and remove it from the frontier
  if the node contains a goal state then return the corresponding solution
  expand the chosen node, adding the resulting nodes to the frontier

function `GRAPH-SEARCH(problem)` 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 and remove it from the frontier
  if the node contains a goal state then return the corresponding solution
  add the node to the explored set
  expand the chosen node, adding the resulting nodes to the frontier
  only if not in the frontier or explored set
Graph Search

function GRAPH-SEARCH(problem) 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 and remove it from the frontier
  if the node contains a goal state then return the corresponding solution
  add the node to the explored set
  expand the chosen node, adding the resulting nodes to the frontier
  only if not in the frontier or explored set

In pairs: try it and record the order of nodes visited (goal: bucharest)

Expand nodes in the order added (FIFO).

Add nodes clockwise (start at midnight)
Graph Search

function GRAPH-SEARCH(problem) 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 and remove it from the frontier
  if the node contains a goal state then return the corresponding solution
  add the node to the explored set
  expand the chosen node, adding the resulting nodes to the frontier only if not in the frontier or explored set

What is the path you found to Bucharest?
Graph Search creates Trees out of Graphs
Tree Search

**Function**

```plaintext
function TREE-SEARCH(problem) 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 and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
    end loop
```

In pairs: try it and record the order of nodes visited

Expand nodes in the order added (FIFO).
Add nodes clockwise (start at midnight)
Search Strategy

- order in which nodes to expand next
- search algorithms are basically all the same
  - but with different search strategies
Evaluating Algorithms

• **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
• **Optimality**: Does the strategy find the optimal solution, as defined on page 68?
• **Time complexity**: How long does it take to find a solution?
• **Space complexity**: How much memory is needed to perform the search?

(lowest cost)
Problem Difficulty (for Graphs)

- **b**: branching factor
  - maximum number of successors per node
    - e.g. chess: 35 (average, not max)
- **d**: depth
  - shallowest goal node
- **m**: maximum length of any path in the state space

What are b, d, & m?
Time & Space Complexity

- Time: number of nodes generated during search
- Space: number of nodes stored in memory
- Unless specified, it’s the worst case
Power of Algorithms

- **Hardware:**
  - Moore’s law

- **Algorithms**
  - Computational benchmark over 15 years
  - 43,000,000-fold increase
  - 1,000-fold due to hardware
  - 43,000-fold due to better algorithms
  - Source: President’s Council of Advisors on Science and Technology. 2010. http://goo.gl/B8kmM

Image from Ray Kurzweil via Wikimedia Commons
Personal Example

- 2001 (Dot Com Boom, Silicon Valley)
- Website for friends to share ratings of books, movies, etc.
- Knew programming, but not algorithms!
- Site grew popular, grew slow....
Searching Algorithms

systematic ways of looking for things

cheese in a maze  lost keys in a house
Searching Graphs

Systematically traversing the nodes
- either until all are explored
- or until a goal is reached
Rules For A

- Cheese can be anywhere
- Number nodes 1 - N
- each unique
- can teleport to neighbors of visited nodes for free

Try it!
One Possibility
Another Possibility
Another Possibility

What is the best strategy if location is random?
Rules For B & C

- Same numbering scheme
- B: Cheese is at ends
- C: Cheese is at beginnings
- Calculate best, average, & worst case

Try it!
Rules For B & C

- Same numbering scheme
- B: Cheese is at ends
- C: Cheese is at beginnings
- Calculate best, average, & worst case

Try it!
Rules For C

B: Ends

• Best Strategy: Deep First
  • best case: 3
  • average case: 7.5
  • worst case: 12 (N)

• Broad First:
  • best case: 9
  • average case: 10.5
  • worst case: 12 (N)

C: Start

• Best Strategy: Broad First
  • best case: 1
  • average case: 2.5
  • worst case: 4

• Deep First:
  • best case: 1
  • average case: 5.5
  • worst case: 10
Searching Graphs

• Two foundational algorithms
  • Depth-First Search (DFS)
  • Bread-First Search (BFS)
• Both work on
  • undirected and directed graphs
    - today we focus on undirected
  • trees
Overview

Breadth-First Search

Depth-First Search

Overview

Breadth-First Search

Depth-First Search

Depth-First Search: Walkthrough

- start = green
- color
  - gray = visited, not left/exhausted
  - black = left/exhausted (optional)
  - white = unexplored

Done!

Example adapted from Dr. Naveen Garg: http://goo.gl/6msFy
Depth-First Search: Implementation

visited = [0] * V

DFS(v):
    visited[v] = 1
    for node in connectedTo(v):
        if !visited[node]:
            DFS(w)
Depth-First Search: Implementation

visited = [0] * V

DFS(ν):

visited[ν] = 1

for node in connectedTo(ν):

if !visited[node]:

DFS(w)

What would happen without visited[]?
Depth-First Search

Spanning tree
path to every vertex in a connected graph
Depth-First Search: Non-Recursive Implementation

• Optional challenge
  • Implement without recursion

• Stack or queue?
Depth-First Search: Complexity (Graph)

- time:
  - $O(b^m)$
- space:
  - $O(b^m)$

visited = [0] * V
DFS(v):
  visited[v] = 1
  for node in connectedTo(v):
    if !visited[node]:
      DFS(w)
Depth-First Search: Complexity (Tree)

- time:
  - $O(b^m)$
- space:
  - $O(b^m)$

- size of example space:
  - Chess: $\sim 10^{120}$ (Shannon 1950)
  - Branching factor 30 (50 moves/100 ply typical)
  - Go: branching factor $\sim 361$