Uninformed Search

Introduction to Artificial Intelligence

Dr. Robin Burke
Problem domain

- Static
  - only our actions change the world
- Deterministic
  - actions always work the way we expect
- Fully-observable
  - we always know everything we need about the world

To solve

- World = state representation
- Solution = sequence of operators
  - transform the initial state into the goal state
State Space

- The collection of all world states
- Connected by edges which are operations
- Solution
  - a path through the state space
  - leads from initial state to goal state
- Problem solving = graph search
Farmer State Space

(S,S,S,S) → (N,N,N,N)

Illegal nodes

(S,S,S,N) → (N,N,S,N)

(M, N, S, S, S) → (M, N, S, S, N)

(M, N, S, N, S) → (M, N, S, N, N)

(M, N, N, S, S) → (M, N, N, S, N)

(M, N, N, N, S) → (M, N, N, N, N)

(M, S, S, S, S) → (M, S, S, S, N)

(M, S, S, N, S) → (M, S, S, N, N)

(M, S, N, S, S) → (M, S, N, S, N)

(M, N, S, S, S) → (M, N, S, S, N)

(M, N, S, S, N) → (M, N, S, S, N)

(M, N, S, N, S) → (M, N, S, N, S)

(M, N, N, S, S) → (M, N, N, S, N)

(M, N, N, S, N) → (M, N, N, S, N)

(M, N, N, N, S) → (M, N, N, N, N)

(M, S, N, S, S) → (M, S, N, S, N)

(M, S, N, S, N) → (M, S, N, S, S)

(M, S, N, N, S) → (M, S, N, N, S)

(M, N, N, S, S) → (M, N, N, S, N)

(M, N, N, S, N) → (M, N, N, S, N)

(M, N, N, N, S) → (M, N, N, N, S)

(M, N, N, N, N) → (M, N, N, N, N)
Search Tree

- For better bookkeeping
  - transform the state space into a search tree

- Nodes
  - not just states but
  - state, parent, action, cost, depth
    - I said path, but you really don’t want a million copies of the same path
    - (parent, edge) are sufficient to reconstruct
Search Tree (ignore illegal states)

N1: (S,S,S,S), {}

N2: (N,S,N,S), (N1, MGN)

N3: (S,S,N,S), (N2, MFS)

N4: (N,N,N,S), (N3, MWN)

N5: (S,N,S,S), (N4, MGS)

N6: (N,N,S,N), (N5, MCN)

N7: (S,N,S,N), (N6, MFS)

N8: (N,N,N,N), (N7, MGN)

N9: (N,S,N,N), (N3, MCN)

N10: (S,S,S,N), (N9, MGS)

N11: (N,N,S,N), (N10, MWN)

N12: (S,N,S,N), (N11, MFS)

N13: (N,N,N,N), (N12, MGN)
Basic idea

- **Given**
  - a set of nodes $N$
  - a successor function $f$ that
    - takes a node $n$
    - returns all of the nodes $S$ reachable from $n$ in a single action

- **Algorithm**
  - pick $n$ from $N$ (somehow)
  - $s = state(n)$
  - $p = path(n)$
  - $S = f(s)$
  - for all $s', a \in S$
    - if $s$ is solution, done
    - if $s$ is illegal, discard
    - else
      - create new node $n_a = <s', (n,a)>$
    - $N = N - n + n_a$
  - repeat
Search strategies

- All of the interesting action is
  - maintenance of the active nodes
    - “search frontier”

- Classes of strategies
  - uninformed search
    - no knowledge of the search space is assumed
    - do not prefer one state over another
  - informed search
    - use knowledge of the search space
    - heuristic search
Uninformed Strategies

- Depth-first search
- Breadth-first search
- Iterative deepening
Characterizing search strategies

- Completeness
  - does it always find the answer?

- Time
  - how long does it take?
  - worst-case run time performance / complexity

- Space
  - how much memory do we need?
  - worst-case space complexity

- Parameters
  - b: branching factor of the search space
    - choices at each node
  - d: depth of the solution
    - # of steps
  - m: maximum depth of the tree
    - could be infinite
Depth-first search

- Unexamined nodes
  - last-in first-out stack

- Meaning
  - grab the top node on the node list
  - replace with its children
  - grab the top one of these
  - etc.
Example: path planning

- states = positions
- initial state
  - v5
- goal state
  - v6
Search Tree?
DFS

- Start with search node $n_1$
- Add successors to stack
  - *assume we enumerate clockwise from right*
  - $n_2, n_3, n_4$
- Pop off the most recent one
  - $n_4$
- Expand that node
- etc.
path: v5, v1, v2, v4, v3, v6
Properties of DFS

- **Complete?**
  - Not always
    - fails in infinite-depth spaces
  - Must avoid repeated states along path

- **Time?**
  - $O(b^m)$: terrible if $m$ is much larger than $d$
  - but if solutions are dense, may be OK

- **Space?**
  - $O(bm)$, i.e., linear space, if no repeated states
  - but closed list makes it $O(b^m)$ in worst case

- **Optimal?**
  - No
  - Returns first answer, not necessarily shortest path
Breadth-first search

- Unexamined nodes
  - first-in first-out queue

- Meaning
  - grab the first thing on the queue
  - put its children on the end
  - grab the next thing

- Effect
  - we don’t examine any paths of length \( k \)
  - until we’ve looked at all paths of length \( k-1 \)
  - achieves optimality
open

n1: v2, n4
n13: v3, n4
n7: v6, n3
n8: v3, n3
n6: v7, n3
n5: v6, n2
n4: v1, n1
n3: v4, n1
n2: v7, n1
n1: v5, init

closed

n1: v5, init

n2: v7, n1
n3: v4, n1
n4: v1, n1
n5: v6, n2

path: v5, v7, v6
Properties of BFS

- **Complete**
  - If $b$ is finite

- **Time**
  - $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$
  - Looks better than $O(b^m)$

- **Space**
  - Worst case = all nodes in memory
  - $O(b^{d+1})$

- **Optimal**
  - Yes, for uniform cost case

Often a bigger problem than time
Iterative Deepening

- How to get optimality of BFS
  - without the space penalty
- Counter-intuitive idea
  - create a depth limit for DFS
    - DFS(k)
    - fails if solution is deeper than k
  - do DFS(k) over and over again
    - as k increases from 1 to d
- Seems inefficient
  - throw away bk nodes after DFS(k)
  - re-generate them for DFS(k+1)
k = 1

open

closed

n1: v5, init

n2: v7, n1

n3: v4, n1

n4: v1, n1

failure
path: v5, v4, v6
Wasted time?

- Number of nodes generated in a depth-limited search to depth $d$ with branching factor $b$:
  \[ N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d \]

- Number of nodes generated in an iterative deepening search to depth $d$ with branching factor $b$:
  \[ N_{IDS} = (d+1)b^0 + db^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d \]

- For $b = 10$, $d = 5$,
  - $N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$
  - $N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$

- Overhead = $(123,456 - 111,111)/111,111 = 11\%$

- In general
  - $N_{IDS}$ is dominated by $b^d$ term
  - $O(b^d)$
Properties of IDS

- Complete?
  - Yes

- Time?
  - $O(b^d)$
  - same as BFS, maybe some overhead

- Space?
  - $O(bm)$, i.e., linear space, if no repeated states

- Optimal?
  - Yes, for uniform cost
Bi-directional search

- In some (but not all) search spaces
  - steps are reversible
  - goal state unique
- Counter-example
  - get to work
- If so
  - we can work backwards from end state
  - as well as forward from initial state
  - two BFS operations
Properties of Bidirectional BFS

- Complete?
  - Yes

- Time?
  - $O(b^{d/2})$
  - can be a big savings

- Space?
  - $O(b^{d/2})$
  - space still a problem

- Optimal?
  - Yes, for uniform cost
Paths with costs

- What if different operations have different costs / benefits?
  - distance
  - $$
  - $$$
  - energy
  - *in games, health*
Costs

- If costs are not uniform
  - then search must take this into account
  - otherwise, optimality is lost
  - shortest in terms of # operations may be very expensive

Question

- how to organize search to take cost into account?
Thursday

- Read Chapter 4.1-4.2