Single Agent Se

Lecture 2
Search on Trees, Best-first search

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Tree Algorithms

Initial Assumptions

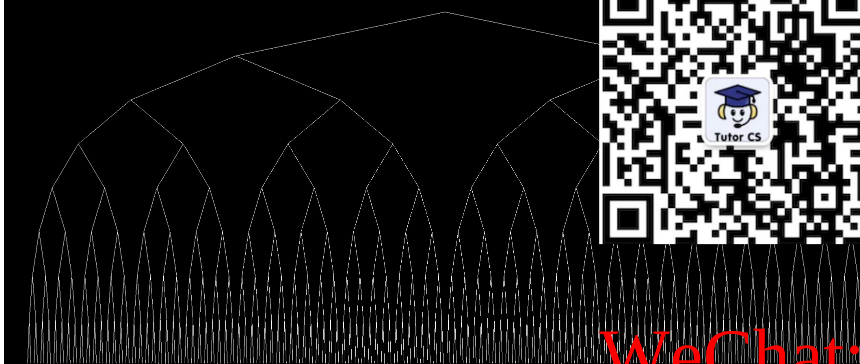
- State space is a tree (no cycles or transpositions)
  - Branching Factor b
  - Depth d
  - Optimal solution cost C*
  - Single Goal state
- Uniform edge costs (assume 1)

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

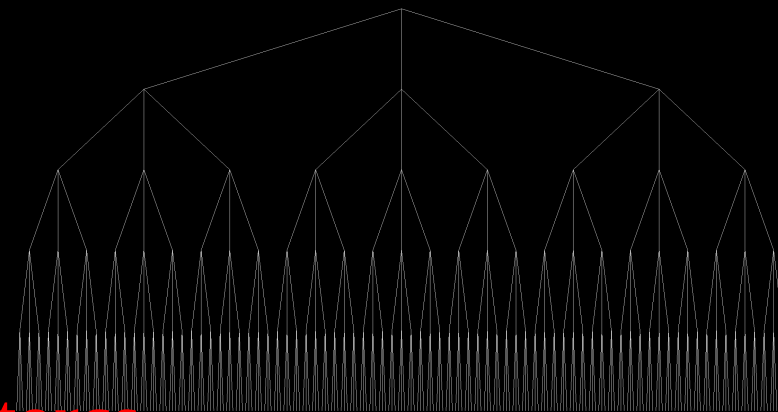
Initial Metrics

- Complete (finds solution if it exists)
- Optimal (finds optimal solution)
- Solution Quality (as ratio of optimal)
- Time Complexity
- Space Complexity

How many nodes in this tree?



How many nodes in this tree?



$$N(b,d) = 1 + b + b^2 + ... + b^{d-1} + b^d$$
$$bN(b,d) = b + b^2 + b^3 + ... + b^d + b^{d+1}$$
$$bN(b,d) - N(b,d) = b^{d+1} - 1$$
$$(b-1)N(b,d) = b^{d+1} - 1$$
$$N(b,d) = \frac{b^{d+1} - 1}{(b-1)}$$
$$N(b,d) = \frac{b(b^d) - 1}{(b-1)}$$
$$N(b,d) = \frac{b}{b-1}(b^d) - \frac{1}{b-1}$$
$$N(b,d) \approx \frac{b}{b-1}(b^d)$$

Algorithm 1: BFS (tree)

```
bool BFS(state from, state to)
{
  queue.push_back(from);
  while (queue.size() != 0)
  {
    if (env->GoalTest(queue.front(), to)) // later than necessary
      return true;

    env->GetSuccessors(queue.front(), succ);
    queue.pop_front();
    for (int x = 0; x < succ.size(); x++)
    {
      queue.push_back(succ[x]);
    }
  }
  return false;
}
```

BFS (tree) Complexity

- Complete - yes
- Optimal - yes
- Solution Quality - (optimal)
- Time Complexity - $O(b^d)$
- Space Complexity - $O(b^d)$

- Is it possible to do better?

Algorithm 2: DFS (tree)

```
bool DFS(state from, state to)
{
  queue.push_back(from);

  while (queue.size() != 0)
  {
      if (env->GoalTest(queue.back(), to))
        return true;

      env->GetSuccessors(queue.back(), succ);
      queue.pop_back();
      for (int x = succ.size()-1; x >= 0; x--)
      {
        queue.push_back(succ[x]);
      }
  }
  return false;
}
```


Implementation

- What do you keep in memory for BFS?
  - Have to keep copy of each state
- What do you keep in memory for DFS?
  - Recursive formulation
  - Only have to keep a single copy of the state
  - Apply & undo moves to state

DFS (tree) Complexity

- Complete - yes
- Optimal - yes (*What if there is more than 1 goal?*)
- Solution Quality - (optimal)
- Time Complexity - $O(b^d)$
- Space Complexity - $O(b \cdot d)$

- Is it possible to do better?

DFS (general) Complexity

- Complete - yes
- Optimal - no
- Solution Quality - $O(b^d/d)$ (worst case
- Time Complexity - $O(b^d)$ (or worse wi
- Space Complexity - $O(b^d)$ (worst case

- Is it possible to do better?

Still assume unit edge costs

DFID

- Run DFS but bound the depth of the tree
  - 1, 2, 3…*d*

DFID Complexity

- Complete - yes
- Optimal - yes
- Solution Quality - (optimal)
- Time Complexity - $O(b^d)$
- Space Complexity - $O(b \cdot d)$

- Is it possible to do better?

DFID Time Complexity

$$DFID(b,d) = \sum_{i=0}^{d} N(b,d)$$

$$DFID(b,d) = \frac{b}{b-1}b^0 + \cdots + \frac{b}{b-1}b^{d-1} + \frac{b}{b-1}b^d$$

$$DFID(b,d) = \frac{b}{b-1}(b^0 + \cdots + b^{d-1} + b^d)$$

$$DFID(b,d) \approx \left(\frac{b}{b-1}\right)^2 (b^d)$$

$$DFID(b,d) = O(b^d)$$

DFID Time Optimality

- DFID is a time-optimal brute-force algorithm
- Proof by contradiction
  - b^d nodes at the level of the solution
  - If algorithm A uses less than b^d exp
    not expand some node at level d
  - Create new problem -- swap goal a
    unexpanded node -- then A won't fi

DFID Space Optimality

- O(d) space
- How much space must an algorithm use?
- If it has b^d time, must have $\log(b^d)$ space
  - $d \log(b)$ -- assume b is constant
- If algorithm is a FSM and runs K steps, each step must have unique state
  - At least a counter to distinguish between states

Underlying Assumptions

- What about problems with cycles?
- Pathfinding in a grid?
  - BFS - $O(r^2)$
  - DFID - $O(4^r)$
- Removing short cycles?
- Difference between explicit/implicit (mark nodes)
- See how to improve this later in grids

Reconstruct Path

- How to reconstruct DFS path?
  - Path is sitting on stack
- How to reconstruct BFS path?
  - Can't do it with naive implementation

Activity

- DFS, DFID, BFS Demo
  - Run algorithms independently
    - Change goal and observe behav
  - Run DFS and DFID
    - Best goal for DFS?
    - Best goal for DFID?
  - How does branching factor impact performance?

Best First Search

Best First Search

- Broad class of algorithms
- Can handle much more general problems
- Many different metrics of "best"
  - f(n) often represents the priority of a node

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Best First Search

- Open List
  - All states that have been generated, but not expanded
- Closed List
  - All states that have been expanded
- Generate a state
  - Parent's successors generated, placed on open
- Expand a state
  - State taken from open
  - Successors generated
  - Put on closed

Best First Search Notes

- Open and Closed lists aren't actually lists
  - Open is usually a priority queue
    - Backed by a hash table for looku
  - Closed is usually a hash table
- Implementation:
  - Start with lists (slow)
  - Then implement faster data structures

Pseudo-Code (1)

- Best-First Algorithm Pseudo-Code
- Put start on OPEN
- While(OPEN is not empty)
  - Pop best node n from OPEN
  - if (n == goal) return path(n, goal)
    - for each child of n // generate children
    - Update(n) // see next slide
- Return NO PATH

Pseudo-Code (2)

- Update(n)
  - if n on closed
    - skip
  - if n on open
    - if found shorter path
      - update cost on open
  - else
    - add n to open

Algs as Best First Search

- DFS is Best First Search when
  - f(n) = depth (larger before smaller)
- BFS is Best First Search when
  - f(n) = depth (smaller before larger)

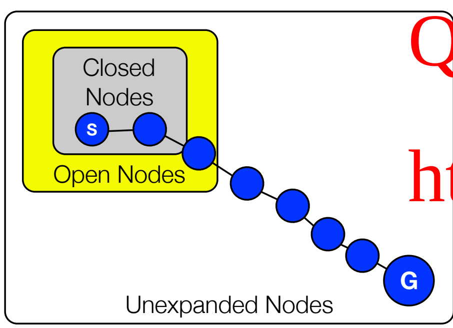
Dijkstra Algorithm

- g-cost is path cost to a node [written g(n)]
- Dijkstra is best-first search
  - f(n) = g(n)

Dijkstra Performance

- Complete - yes (Finite graph or minimum edge cost)
- Optimal - yes (Non-negative edges [for now])
- Solution Quality - (optimal)
- Time Complexity - $O(b^d)$ [?]
- Space Complexity - $O(b^d)$ [?]

High-Level View



Dijkstra Optimality

- Property #1
  - g-costs along any path are monotonically increasing
    - Assume non-negative costs
    - Therefore adding a node to a path, increases the cost

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Dijkstra Optimality

- Property #2
  - A node on the optimal path (to any state) is always on the open list wit cost from the start. (Proof by induct
    - Initially start is on OPEN
    - Assume step n; step n+1:
      - If node at n+1 is on optimal pa successor will be on open
      - If not, the previous node will still be on OPEN