Graph Search

Many problems in computer science correspond to searching for a path in a graph, given a start node and goal criteria:
- Route planning – Mapquest
- Packet-switching
- VLSI layout
- 6-degrees of Kevin Bacon
- Program synthesis
- Speech recognition
  - We’ll discuss these last two later...

General Graph Search Algorithm

Open – some data structure (e.g., stack, queue, heap)
Criteria – some method for removing an element from Open
• Search( Start, Goal_test, Criteria)
  • insert(Start, Open);
  • repeat
    • if (empty(Open)) then return fail;
    • select Node from Open using Criteria;
    • if (Goal_test(Node)) then return Node;
    • for each Child of node do
      • if (Child not already visited) then Insert( Child, Open );
    • Mark Node as visited;
  • end

Depth-First Graph Search

Open – Stack
Criteria – Pop
• DFS( Start, Goal_test)
  • push(Start, Open);
  • repeat
    • if (empty(Open)) then return fail;
    • Node := pop(Open);
    • if (Goal_test(Node)) then return Node;
    • for each Child of node do
      • if (Child not already visited) then push(Child, Open);
    • Mark Node as visited;
  • end

Breadth-First Graph Search

Open – Queue
Criteria – Dequeue (FIFO)
• BFS( Start, Goal_test)
  • enqueue(Start, Open);
  • repeat
    • if (empty(Open)) then return fail;
    • Node := dequeue(Open);
    • if (Goal_test(Node)) then return Node;
    • for each Child of node do
      • if (Child not already visited) then enqueue(Child, Open);
    • Mark Node as visited;
  • end

Two Models

1. Standard Model: Graph given explicitly with \( n \) vertices and \( e \) edges.
   > Search is \( O(n + e) \) time in adjacency list representation
2. AI Model: Graph generated on the fly.
   > Time for search need not visit every vertex.
Planning Example

• A huge graph may be implicitly specified by rules for generating it on-the-fly
  • Blocks world:
    › vertex = relative positions of all blocks
    › edge = robot arm stacks one block

AI Comparison: DFS versus BFS

• Depth-first search
  › Does not always find shortest paths
  › Must be careful to mark visited vertices, or you could go into an infinite loop if there is a cycle

• Breadth-first search
  › Always finds shortest paths – optimal solutions
  › Marking visited nodes can improve efficiency, but even without doing so search is guaranteed to terminate

Is BFS always preferable?

DFS Space Requirements

• Assume:
  › Longest path in graph is length \( d \)
  › Highest number of out-edges is \( k \)
• DFS stack grows at most to size ??
• For \( k=10, d=15 \), size is ??

BFS Space Requirements

• Assume
  › Distance from start to a goal is \( d \)
  › Highest number of out edges is \( k \) BFS
• Queue could grow to size
  › For \( k=10, d=15 \), size is ???

Conclusion

• In the AI Model, DFS is hugely more memory efficient, if we can limit the maximum path length to some fixed \( d \).
  › If we knew the distance from the start to the goal in advance, we can just not add any children to stack after level \( d \)
  › But what if we don’t know \( d \) in advance?

Problem: Large Graphs

• It is expensive to find optimal paths in large graphs, using BFS or Dijkstra’s algorithm (for weighted graphs)
• How can we search large graphs efficiently by using “commonsense” about which direction looks most promising?
Plan a route from 9th & 50th to 3rd & 51st

Best-First Search

• The Manhattan distance ($\Delta x + \Delta y$) is an estimate of the distance to the goal
  › It is a search heuristic
• Best-First Search
  › Order nodes in priority to minimize estimated distance to the goal
• Compare: BFS / Dijkstra
  › Order nodes in priority to minimize distance from the start

Open – Heap (priority queue)
Criteria – Smallest key (highest priority)
h(n) – heuristic estimate of distance from n to closest goal

Best-First Search

Obstacles

• Best-FS eventually will expand vertex to get back on the right track

Non-Optimality of Best-First

Path found by Best-first

Shortest Path
Improving Best-First

- Best-first is often tremendously faster than BFS/Dijkstra, but might stop with a non-optimal solution
- How can it be modified to be (almost) as fast, but guaranteed to find optimal solutions?
  - One of the first significant algorithms developed in AI
  - Widely used in many applications

A*

- Exactly like Best-first search, but using a different criteria for the priority queue:
  - minimize (distance from start) + (estimated distance to goal)
  - priority \( f(n) = g(n) + h(n) \)
  - \( g(n) \) = true distance from start
  - \( h(n) \) = heuristic distance to goal

Optimality of A*

- Suppose the estimated distance is always less than or equal to the true distance to the goal
  - heuristic is a lower bound
- Then: when the goal is removed from the priority queue, we are guaranteed to have found a shortest path!
- Everything else has a higher estimated cost

A* in Action

Applications of A*: Planning

- A huge graph may be implicitly specified by rules for generating it on-the-fly
- Blocks world:
  - vertex = relative positions of all blocks
  - edge = robot arm stacks one block

Blocks World

- Blocks world:
  - distance = number of stacks to perform
  - heuristic lower bound = number of blocks out of place
  - # out of place = 2, true distance to goal = 3
Application of A*: Speech Recognition

• (Simplified) Problem:
  › System hears a sequence of 3 words
  › It is unsure about what it heard
    • For each word, it has a set of possible “guesses”
      E.g.: Word 1 is one of {“hi”, “high”, “I”}
  › What is the most likely sentence it heard?

Speech Recognition as Shortest Path

• Convert to a shortest-path problem:
  › Utterance is a “layered” DAG
  › Begins with a special dummy “start” node
  › Next: A layer of nodes for each word position, one
    node for each word choice
  › Edges between every node in layer i to every node
    in layer i+1
    • Cost of an edge is smaller if the pair of words frequently
      occur together in real speech
      - Technically: - log probability of co-occurrence
  › Finally: a dummy “end” node
  › Find shortest path from start to end node

Summary: Graph Search

• Depth First
  › Little memory required
  › Might find non-optimal path
• Breadth First
  › Much memory required
  › Always finds optimal path
• Dijkstra’s Short Path Algorithm
  › Like BFS for weighted graphs
• Best First
  › Can visit fewer nodes
  › Might find non-optimal path
• A*
  › Can visit fewer nodes than BFS or Dijkstra
  › Optimal if heuristic estimate is a lower-bound