today’s topics:
• heuristic search (part 1 of 2)

Recap
The last lecture introduced
• Basic problem solving techniques:
  – Breadth-first search
  – Depth-first search
• Breadth-first search is complete but expensive.
• Depth-first search is cheap but incomplete
• Can’t we do better than this?
• That is what this lecture is about

Overview
Aims of this lecture:
• show how basic search (depth 1st, breadth 1st) can be improved;
• introduce:
  – depth limited search;
  – iterative deepening.
• show that even with such improvements, search is hopelessly unrealistic for real problems.

Algorithmic Improvements
• Are then any algorithmic improvements we can make to basic search algorithms that will improve overall performance?
• Try to get optimality and completeness of breadth 1st search with space efficiency of depth 1st.
• Not too much to be done about time complexity :-( 
Depth Limited Search

- Depth first search has some desirable properties — space complexity.
- But if wrong branch expanded (with no solution on it), then it won't terminate.
- Idea: introduce a depth limit on branches to be expanded.
- Don't expand a branch below this depth.

- General algorithm for depth limited search:
  
  depth limit = max depth to search to;
  agenda = initial state;
  while agenda not empty do
      take node from front of agenda;
      new nodes = apply operations to node;
      if goal state in new nodes then {
          return solution;
      }
      if depth(node) < depth limit then {
          add new nodes to front of agenda;
      }
      
- For the 8-puzzle setup as:
  
  ![Diagram](a.png)
  ![Diagram](b.png)
  ![Diagram](c.png)

  - the search will be as follows:
  
  ![Diagram](a.png)
  ![Diagram](b.png)
  ![Diagram](c.png)

  Discarded before generating node 7

© 1998 Morgan Kaufman Publishers
• So, when we hit the depth bound, we don’t add any more nodes to the agenda.
• Then we pick the next node off the agenda.
• This has the effect of moving the search back to the last node above depth limit that that is “partly expanded”.
• This is known as chronological backtracking.
• The effect of the depth limit is to force the search of the whole state space down to the limit.
• We get the completeness of breadth-first (down to the limit), with the space cost of depth first.

• Unfortunately, if we choose a max depth for d.l.s. such that shortest solution is longer, d.l.s. is not complete.
• Iterative deepening an ingenious complete version of it.
• Basic idea is:
  – do d.l.s. for depth 1; if solution found, return it;
  – otherwise do d.l.s. for depth 2; if solution found, return it;
  – otherwise, . . .
• So we repeat d.l.s. for all depths until solution found.

• General algorithm for depth limited search:
  depth limit = 1;
  repeat {
    result = depth_limited_search(
      max depth = depth limit;
      agenda = initial node;
    );
    if result contains goal then {
      return result;
    }
    depth limit = depth limit + 1;
  } until false; /* i.e., forever */
• Calls d.l.s. as subroutine.
• Note that in iterative deepening, we *re-generate nodes on the fly*. Each time we do call on depth limited search for depth \( d \), we need to regenerate the tree to depth \( d - 1 \).

• Isn’t this inefficient?

• Tradeoff time for memory.

• In general we might take a little more time, but we save a lot of memory.

• Now for breadth-first search to level \( d \):

\[
N_{bf} = 1 + b + b^2 + b = \frac{b^{d+1} - 1}{b-1}
\]

• In contrast a complete depth-limited search to level \( j \):

\[
N_{df}^j = \frac{b^{j+1} - 1}{b-1}
\]

• (This is just a breadth-first search to depth \( j \).)

• In the worst case, then we have to do this to depth \( d \), so expanding:

\[
N_{id} = \sum_{j=0}^{d} \frac{b^{j+1} - 1}{b-1} = \frac{b^{d+2} - 2b - bd + d + 1}{(b-1)^2}
\]

• For large \( d \):

\[
\frac{N_{id}}{N_{bf}} = \frac{b}{b-1}
\]

• So for high branching and relatively deep goals we do a small amount more work.

• Example: Suppose \( b = 10 \) and \( d = 5 \).

  Breadth first search would require examining 111,111 nodes, with memory requirement of 100,000 nodes.

  Iterative deepening for same problem: 123,456 nodes to be searched, with memory requirement only 50 nodes.

  Takes 11% longer in this case.

For the 8-puzzle setup as:

\[
\begin{array}{ccc}
2 & 8 & 3 \\
1 & 6 & 4 \\
7 & 5 & \end{array}
\]

• What would iterative deepening search look like?

• Well, it would explore the search space:
In the following way.
States would be expanded in the order:
1. 1
2. 1, 2, 3, 4
3. 1, 2, 5, 3, 6, 7, 8, 4, 9.
4. 1, 2, 5, 10, 11, 3, 6, 13, 17, 4, 15, 8, 16, 17, 4, 9, 18, 19.
5. ...
Note that these are the states visited, not the nodes on the agenda.

Bi-directional Search

- Suppose we search from the goal state backwards as well as from initial state forwards.
- Involves determining predecessor nodes to goal, and then looking at predecessor nodes to this, ...
- Rather than doing one search of $b^d$, we do two $b^{d/2}$ searches.
- Much more efficient.

Example:
Suppose $b = 10$, $d = 6$.
Breadth first search will examine $b^d$ nodes.
Bidirectional search will examine $2b^{d/2}$ nodes.
- Can combine different search strategies in different directions.
- For large $d$, is still impractical!
Summary

- This lecture has looked at some more efficient techniques than breadth first and depth first search.
  - depth-limited search;
  - iterative-deepening search; and
  - bidirectional search.
- These all improve on depth-first and breadth-first search.
- However, all fail for big enough problems (too large state space).
- Next lecture, we will look at approaches that cut down the size of the state-space that is searched.