Artificial Intelligence: Knowledge Representation

Using Search in Problem Solving

- Intro
- Basic Search Techniques
- Heuristic Search

Intro: Search and AI

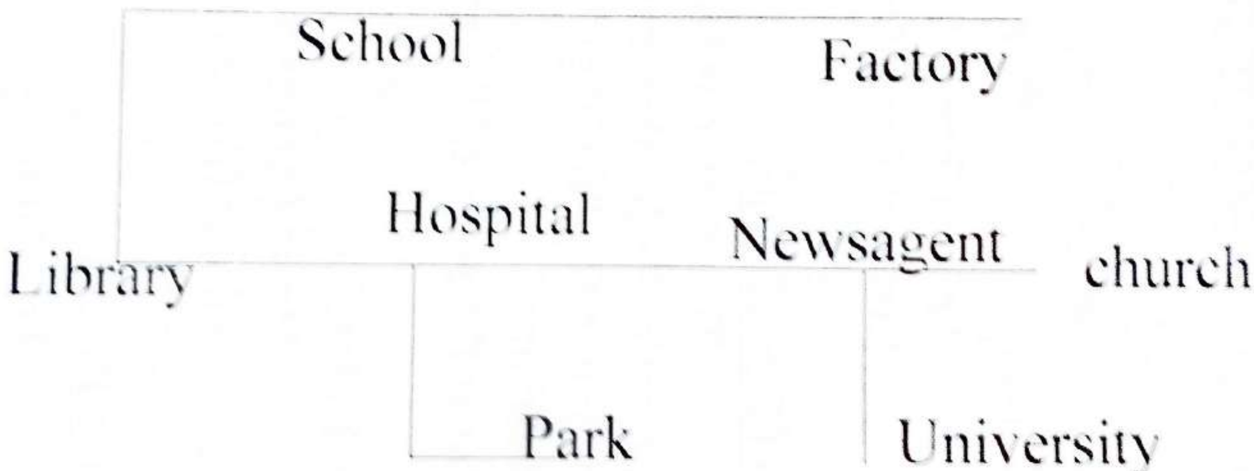
- In solving problems, we sometimes have to search through many possible ways of doing something.
  - We may know all the possible actions our robot can do, but we have to consider various sequences to find a sequence of actions to achieve a goal.
  - We may know all the possible moves in a chess game, but we must consider many possibilities to find a good move.
- Many problems can be formalised in a general way as search problems.

Search and Problem Solving

- Search problems described in terms of:
  - An initial state. (e.g., initial chessboard, current positions of objects in world, current location)
  - A target state.(e.g., winning chess position, target location)
  - Some possible actions, that get you from one state to another. (e.g. chess move, robot action, simple change in location).
- Search techniques systematically consider all possible action sequences to find a *path* from the initial to target state.

Simple Example

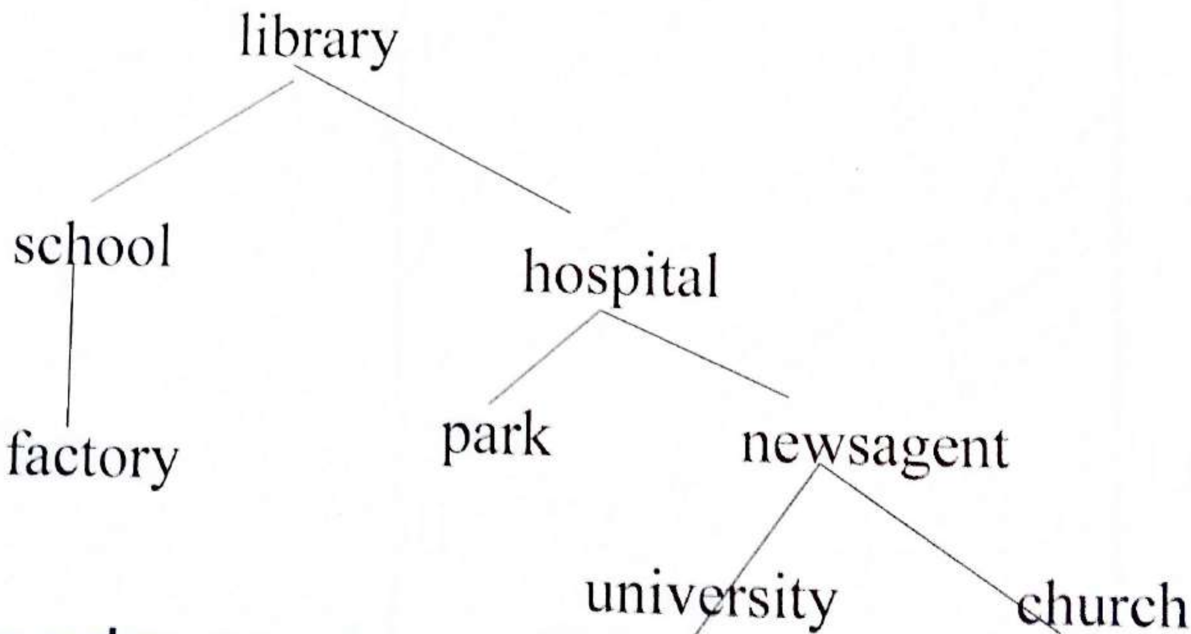
- Easiest to first look at simple examples based on searching for route on a map.



- How do we systematically and exhaustively search possible routes, in order to find, say, route from library to university?

Search Space

- The set of all possible states reachable from the initial state defines the *search space*.
- We can represent the search space as a tree.



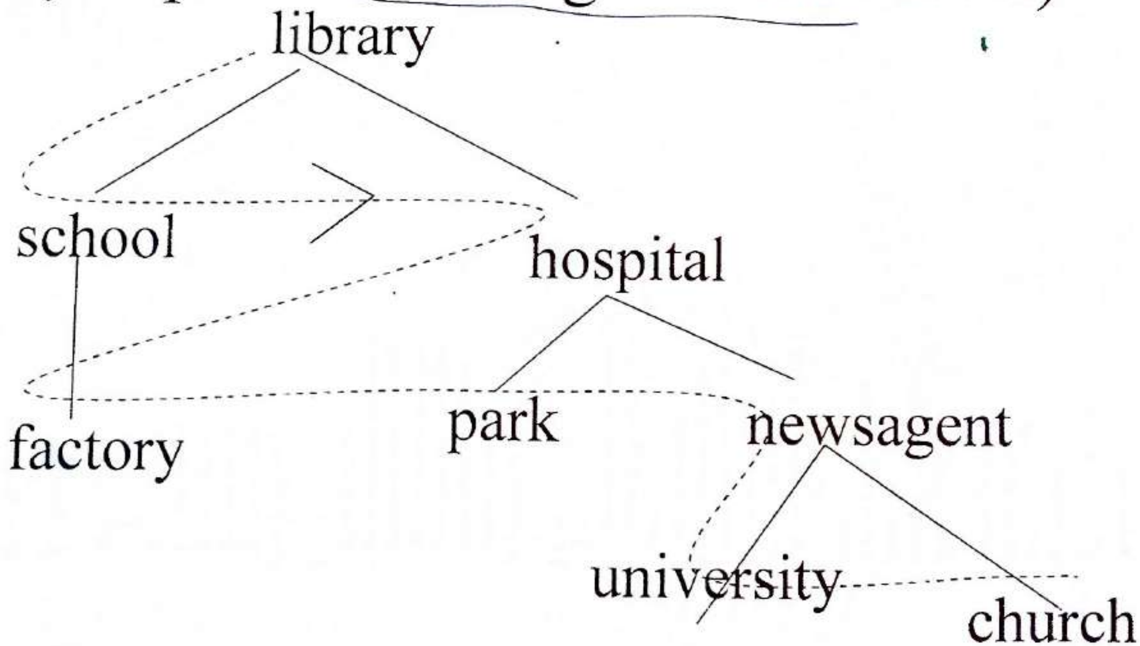
- We refer to nodes connected to and "under" a node in the tree as "successor nodes".

Simple Search Techniques

- How do we search this tree to find a possible route from library to University?
- May use simple systematic search techniques, which try every possibility in systematic way.
- Breadth first search - Try shortest paths first.
- Depth first search - Follow a path as far as it goes, and when reach dead end, backup and try last encountered alternative.

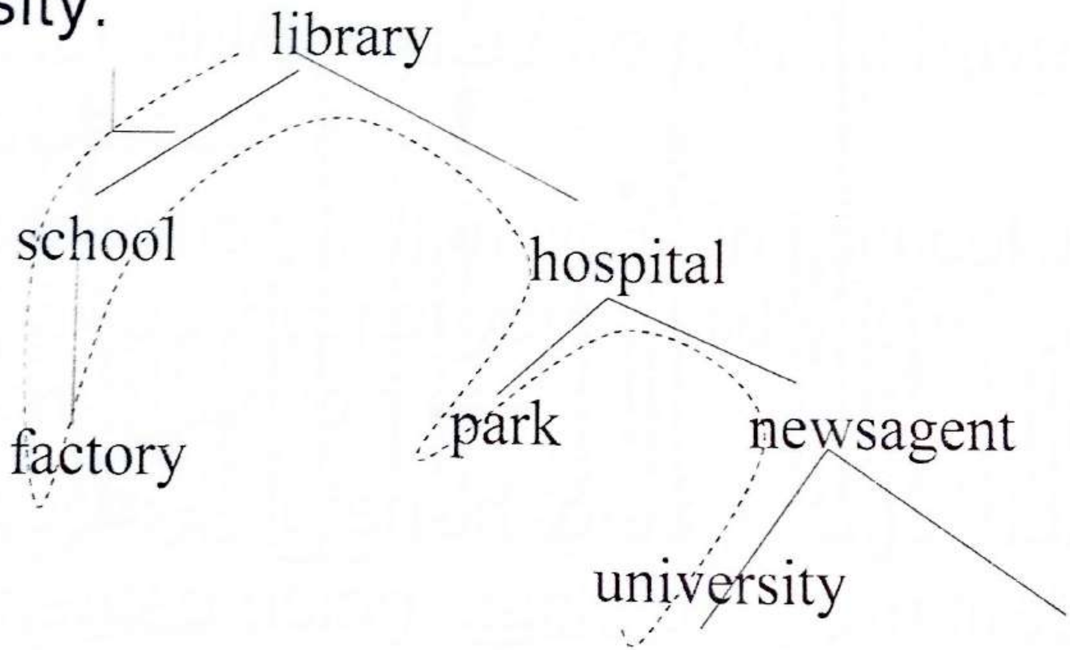
Breadth first search

Explore *nodes* in tree order: library, school, hospital, factory, park, newsagent, uni, church.
(conventionally explore left to right at each level)



Depth first search

- Nodes explored in order: library, school, factory, hospital, park, newsagent, university.



Algorithms for breadth first and depth first search.

- Very easy to implement algorithms to do these kinds of search.
- Both algorithms keep track of the list of nodes found, but for which routes from them have yet to be considered.
  - E.g., [school, hospital] -have found school and hospital in tree, but not yet considered the nodes connected to these.
- List is sometimes referred to as an agenda. But implemented using stack for depth first, queue for breadth first.

Algorithm for breadth first:

- Start with queue = [initial-state] and found=FALSE.
- While queue not empty and not found do:
  - Remove the first node N from queue.
  - If N is a goal state, then found = TRUE.
  - Find all the successor nodes of N, and put them on the end of the queue.

Algorithm for depth first:

- Start with stack = [initial-state] and found=FALSE.
- While stack not empty and not found do:
  - Remove the first node N from stack.
  - If N is a goal state, then found = TRUE.
  - Find all the successor nodes of N, and put them on the top of the stack.

Note: Detailed workthrough of algorithms and discussion of trees/graphs in textbook.

Or.. To solve a puzzle

- "You are given two jugs, a 4 gallon one, and a 3 gallon one. Neither has any measuring markers on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water in the 4 gallon jug?"
- How do we represent the problem state? Can represent just as pair or numbers.
  - {4, 1} means 4 gallons in 4 gallon jug, 1 gallon in 3 gallon jug.
- How do we represent the possible actions.
  - Can give simple rules for how to get from old to new state given various actions.

Jug actions

- 1. Fill 4-gallon jug. {X, Y} -> {4, Y}
- 2. Fill 3-gallon jug. {X, Y} -> {X, 3}
- 3. Empty 4 gallon jug into 3 gallon jug. {X, Y} -> {0, X+Y} (but only OK if X+Y <= 3)
- 4. Fill the 4 gallon jug from the 3 gallon jug. {X, Y} -> {4, X+Y-4} (if X+Y > 4)
- etc (full set given in textbook}

Search Tree for Jugs

{0, 0}

Fill 4 gallon　　Fill 3 gallon

{4, 0}　　{0, 3}

Fill 3 gallon　　Fill 3 gallon from 4 gallon

{4, 3}　　{1, 3}

.. And so on.

So..

- To solve a moderately complex puzzle what we can do is:
  - Express it in terms of search.
  - Decide how "problem state" may be expressed formally.
  - Decide how to encode primitive actions as rules for getting from one state to another.
  - Use a standard tree/graph search algorithm/program, which uses uses a general "successor state" function which you define for your problem.

Heuristic search algorithms.

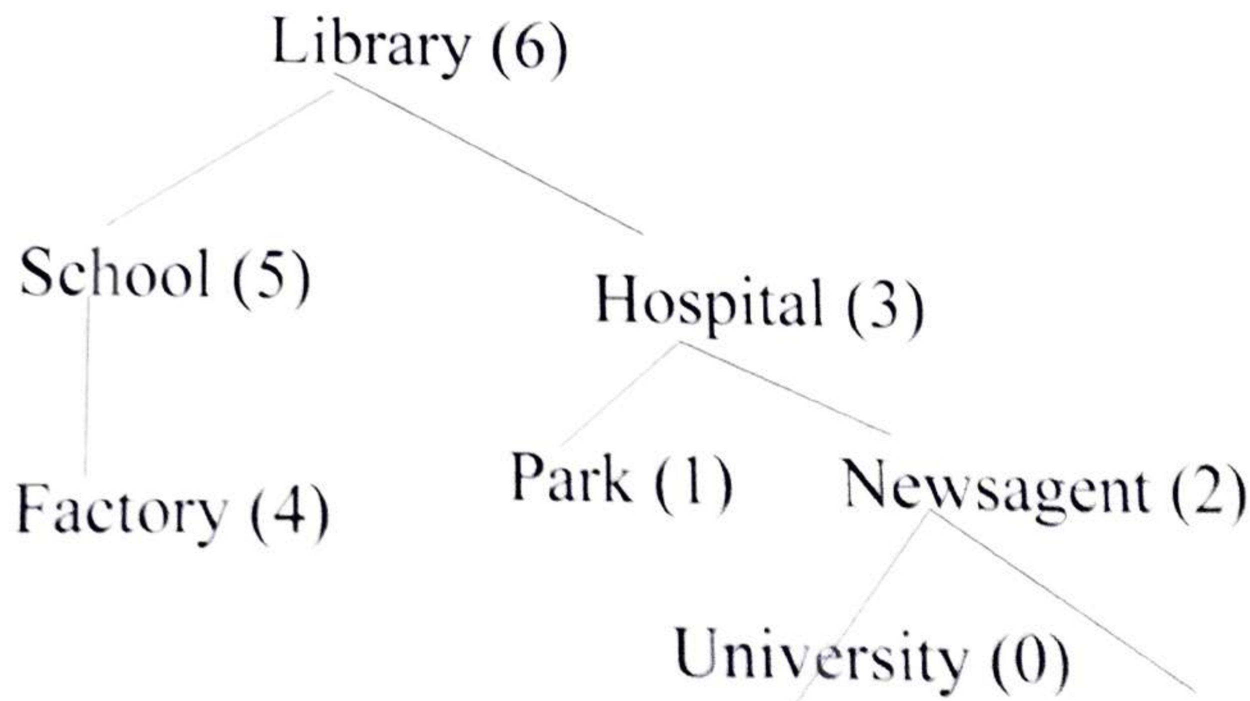
- Depth first and breadth first search turn out to be too inefficient for really complex problems.
- Instead we turn to "heuristic search" methods, which don't search the whole search space, but focus on promising areas.
- Simplest is best first search. We define some "heuristic evaluation function" to say roughly how close a node is to our target.
  - E.g., map search: heuristic might be "as the crow flies" distance based on map coords,
  - Jug problem: How close to 2 gallons there are in 4 gallon jug.

Best first search algorithm

- Best first search algorithm almost same as depth/breadth.. But we use a priority queue, where nodes with best scores are taken off the queue first.
- While queue not empty and not found do:
  - Remove the BEST node N from queue.
  - If N is a goal state, then found = TRUE.
  - Find all the successor nodes of N, assign them a score, and put them on the queue..

Best first search

- Order nodes searched: Library, hospital, park, newsagent, university.



Other heuristic search methods

- Hill climbing: always choose successor node with highest score.
- A*: Score based on predicted total path “cost”, so sum of
  - actual cost/distance from initial to current node,
  - predicted cost/distance to target node.

Summary

- General search methods can be used to solve complex problems.
- Problems are formulated in terms of initial and target state, and the primitive actions that take you from one state to next.
- May need to use *heuristic* search for complex problems, as search space can be too large.