Uninformed Search

State Space Search

- N is the set of nodes of the graph. These correspond to states in the problem solving process.
- A is the set of arcs between nodes. These correspond to steps in the problem solving process.
- S, a non-empty subset of N, contains the start state(s) of the problem.
- G, a non-empty subset of N, contains the goal state(s) of the problem. The states in G are described using either a measurable property of the states encountered in the search, or a property of the path developed in the search.
- A solution path is a path through this graph from a node in S to a node in G.

Data-Driven vs. Goal-Driven Search

- Data-Driven or Forward-Chaining: Start with the data you have, and apply the rules until you reach the goal state.
- Goal-Driven or Backward-Chaining: Start at the goal, and work backwards.
- Both search the same space but, depending on the ‘shape’ of the problem, one might find the solution faster.

Recursion and backtracking

- The CLOSED list contains states already checked against the goal state.
- The OPEN list contains states which have been generated, but not yet checked.
- Can also use these lists to keep track of path information – see text for details.

Depth-first search

- If the present state S is not the goal, move it from the OPEN to the CLOSED list.
- Generate the children of S: S_1, S_2…S_n. Eliminate any which are already on OPEN or CLOSED. Add the remaining children to the *beginning* of the OPEN list.
- Make the first state on the OPEN list the present state.

Advantages and disadvantages of DFS

- If the solution path is known to be long, DFS won't waste time searching for easy solutions.
- Not guaranteed to find the shortest solution – or *any* solution, if search space is not finite.
- Often more efficient for spaces with high ***branching factor*** (i.e. average number of arcs from nodes)
- ***Worst case time complexity****: $O(b^m)$ where b is the branching factor and m is the maximum depth – infinite if tree is unbounded!*

Breadth-first search

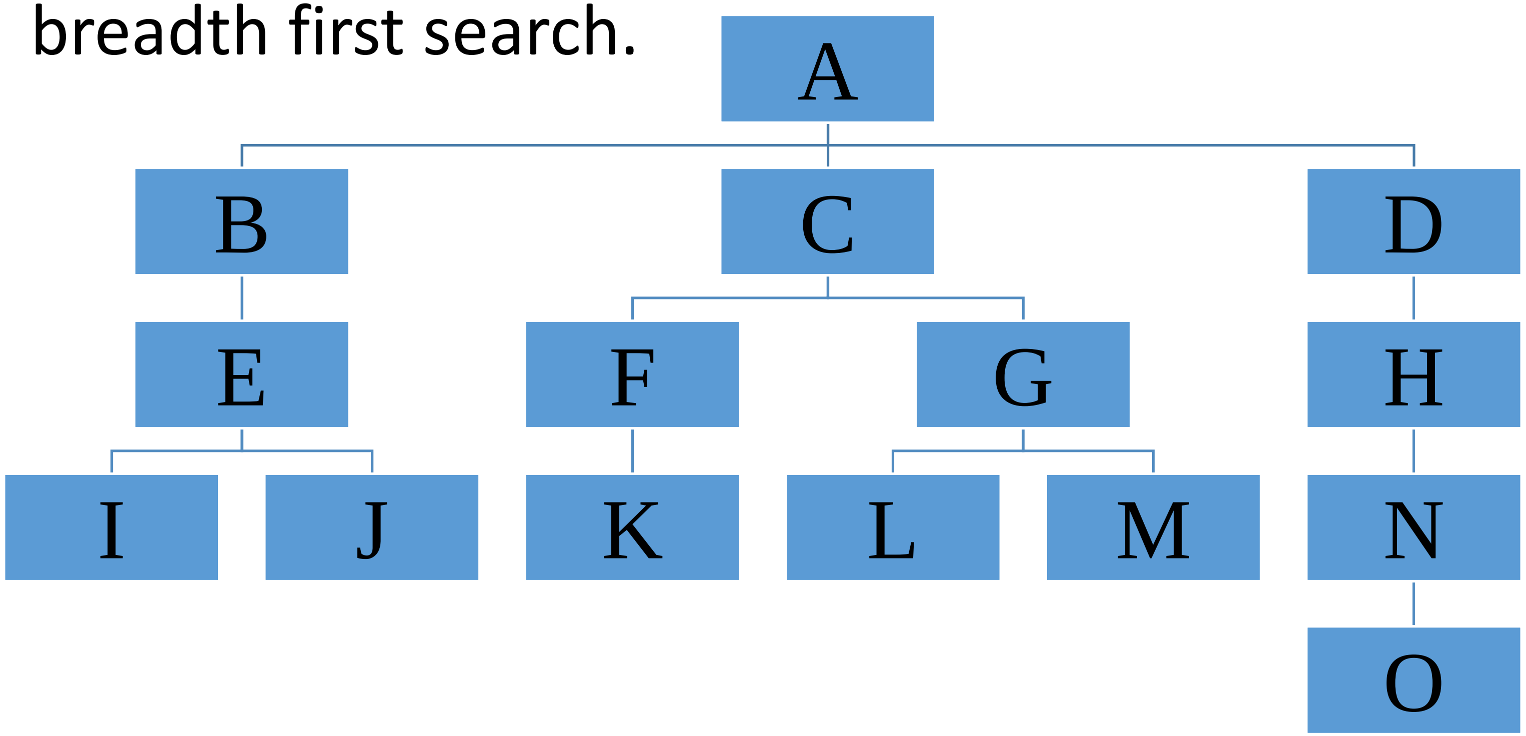
- If the present state S is not the goal, move it from the OPEN to the CLOSED list.
- Generate the children of S: S_1, S_2…S_n. Eliminate any which are already on OPEN or CLOSED. Add the remaining children to the *end* of the OPEN list.
- Make the first state on the OPEN list the present state.

Advantages and disadvantages of BFS

- Guaranteed to find shortest solution.
- If there's a high *branching factor*, then *combinatorial explosion* might occur (i.e. the OPEN list could get very large before getting very far in the search space)
- Time complexity $O(b^n)$, where b is the average branching factor, and n is the depth of the shortest solution path.

Exercise

- For this search space, give the order in which the nodes would be examined in depth first and breadth first search.



DFS with iterative deepening

- First, do DFS with *depth bound* at 1.
- If no solution is found, repeat with the depth bound at 2. And so on.
- Guaranteed to find the shortest path, while avoiding the potentially large OPEN lists of BFS.
- Worst case time efficiency on the same order of magnitude as BFS, $O(B^d)$, where B is the average branching factor, and d is the depth of the shortest-path solution.

Bidirectional search

- A combination of data-driven and goal driven search.
- Starts at both ends and tries to meet in the middle.
- The two searches can be DFS or BFS.
- Great if the two searches meet, but if they don’t, excessive search results.

Informed Search

Uninformed vs. informed search

- All uninformed strategies (BFS, DFS, IDFS, and others) have worst-case exponential time complexity.
- Only *informed* strategies reduce this complexity.

Best-first search

- Uses *heuristics* to **order** the states on the OPEN list.
- Heuristics are *domain-specific* rules-of-thumb that **evaluate states.**
- So, at each iteration, the search considers the ‘best’ or most promising state.
- Cost of evaluating the states (evaluating the heuristic) must be balanced against improvement in search efficiency.
- Heuristics are *fallible,* so they could steer the search away from a nearby solution.

Heuristics are good for:

- Problems without exact solutions, because of ambiguities in the problem statement or available data (e.g. medical diagnosis, vision).
- Problems with exact solutions, but the computational cost of exhaustive search is prohibitive (e.g. chess, theorem proving)

Typical heuristics:

- Estimate the distance to a solution for a given state (e.g. # of pieces the opponent has left, # of pieces in the correct positions).
- Give points for strategically important events (e.g. capture of castle might be worth 10 points to a pawn’s 5).
- Use pattern-matching to recognize good (or bad) parts of the current state (e.g. might spot a typical “problem situation” in a corner of the board).
- Can be combined

Exercise

What would be good heuristics for:

- Finding your way from one side of downtown to the other? (what are the nodes? How do you evaluate them?)
- Searching the UH catalog for elective courses you want to take? Nodes are courses, links are to same professor, same department, cross-listed courses, shared words in title.

Weighted linear function

$EVAL(s) = w_1f_1(s)+w_2f_2(s)\ldots w_nf_n(s)$

Where w are weights, f are individual heuristics functions.

PROBLEM: Assumes that functions are independent.

Exercise

Consider the square puzzle.

- What is the branching factor?
- What heuristics could you use for this search space?
- Assume the program can search a maximum number of nodes before moving (say, 10) – what does heuristic search buy us?

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	