EC-350 AI and Decision Support Systems

Week 3

Solving Problems by Searching

Dr. Arslan Shaukat

NUST

Acknowledgement: Lecture slides material from Stuart Russell

Problem Solving Agent

- Problem solving agent decides what to do by finding sequence of actions that lead to desirable states and hence solution
- 2 types of search algorithms:
  - *Uninformed search algorithms: that are given no information about the problem other than its definition.*
  - *Informed search algorithms: can do quite well given some guidance on where to look for solutions*
- Intelligent agents are supposed to maximize their performance measure.
- Achieving this is sometimes simplified if the agent can adopt a **goal** and aim at satisfying it

Example

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest

- Formulate goal:
  - *be in Bucharest*
- Formulate problem:
  - *states: various cities*
  - *actions: drive between cities*
- Find solution:
  - *sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest*

Goal & Problem Formulation

- Goals help organize behaviour by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider
- Goal formulation, based on current situation and the agent's performance measure, is the first step in problem solving
- Problem Formulation is the process of deciding what actions and states to consider, given a goal
- In general, an agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that leads to a state of known value, and then choosing the best sequence

Example

One possible route
Arad -> Sibiu -> Ramnincu Valcea -> Pitesti -> Bucharest

Search and Solution

- The process of looking for sequence of actions to arrive at a goal is called search
- Search algorithm takes problem as input and returns solution in form of action sequence
- Once a solution is found, the recommended actions can be carried out. This is called execution.
- Formulate, search, execute design for agent

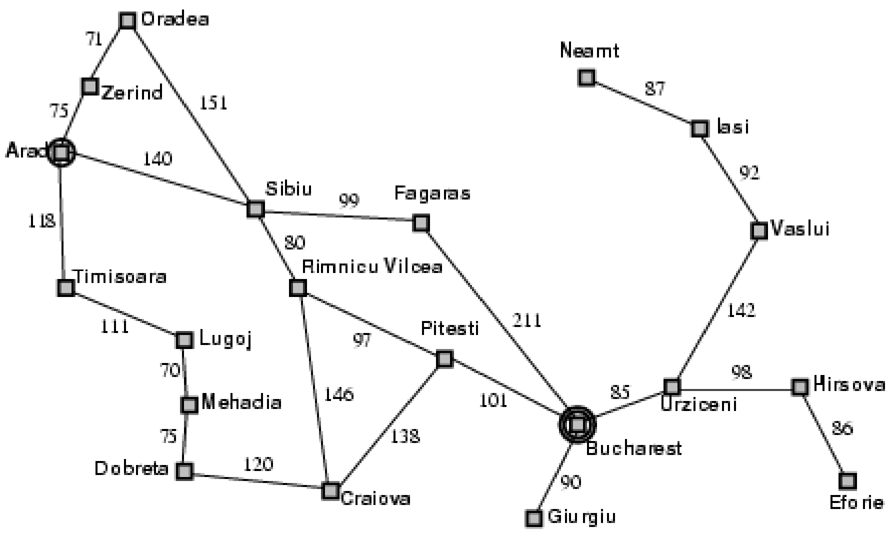
Problem Definition

- A Problem can be defined by the following 5 components:
  - *Initial state defines the start state*
  - *Actions (s) A description of the possible actions available to the agent. Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s*
  - *Transition model/Result (s, a) returns the state that results from doing action a in state s*

Problem Definition (cont'd)

  - *Goal Test (s) a function, when a state is passed to it, returns True or False value if it is a goal or not*
  - *Path Cost an additive function which assigns a numeric cost to each path. This function also reflect agent's own performance measure.*
    - There may be step cost also if a path contains more than one steps. It may be denoted by c(s, a, s'), where a is action and s & s' are the current and new states respectively.
- A path or solution in the state space is the sequence of states connected by sequence of actions
- Together, the initial state, actions and new states implicitly defines the State space of the problem

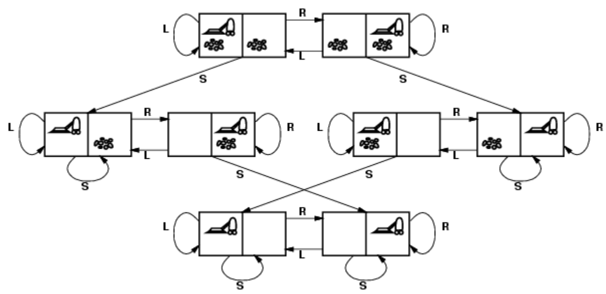
Find a Route – Arid to Bucharest



Problem Formulation - Example

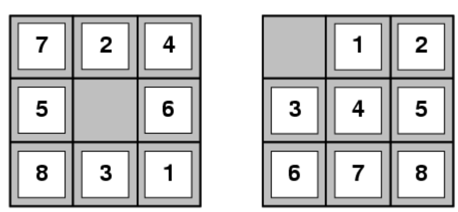
- Problem Description: find an optimal path from Arad to Bucharest
- Initial State= *In(Arad)*
- Actions (s) = set of possible actions agent can take
  {goto(Zerind), goto(Sibiu), goto(Timisora)}
- Result (s, a): Result (*In(Arad)*, *Go(Zerind))* = *In(Zerind)* .
- Goal Test: determine if at goal
  - *can be explicit, e.g., In(Bucharest)*
- Path Cost: cost of each step added together
  - *e.g., sum of distances, number of actions executed, etc.*
  - *The step cost is assumed to be* ≥ 0
- A solution is a sequence of actions leading from the initial state to a goal state, solution quality is measured by path cost

Vacuum World State Space Graph



- states? The agent is in one of two locations, 2*2*2 = 8 possible world states
- initial state: any state can be designated as initial state
- Actions (a): {<left>, <right>, <suck>, <noop>}
- Result(s,a): <right clean>, <left clean>
- goal test: no dirt at all locations
- path cost: 1 per action

Example: The 8-puzzle

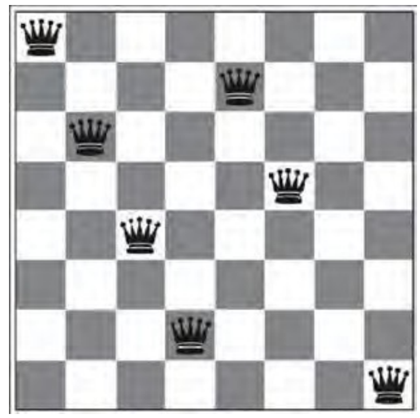


Start State Goal State

- States: locations of each tile and blank,
  9!/2=181,440 reachable world states
- Initial state: any state can be designated
- Action(s): {<left>, <right>, <up>, <down>}
- Result(s,a): new state after taking any of above actions
- goal test: matches the goal state (given)
- path cost: 1 per move

Example: 8-Queen Problem

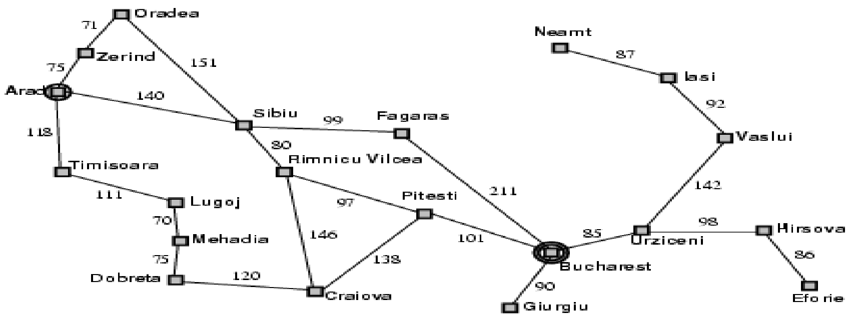
States: any arrangement of 0 to 8 queens on the board
Initial State: no queen on the board
Actions: add a queen at any square
Result: new board state with queen added
Goal test: 8 queens on the board – none attacked
Path Cost: 1 per move

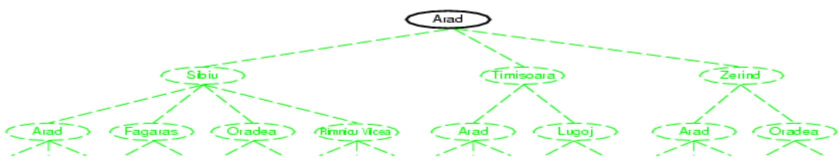


Searching for Solutions

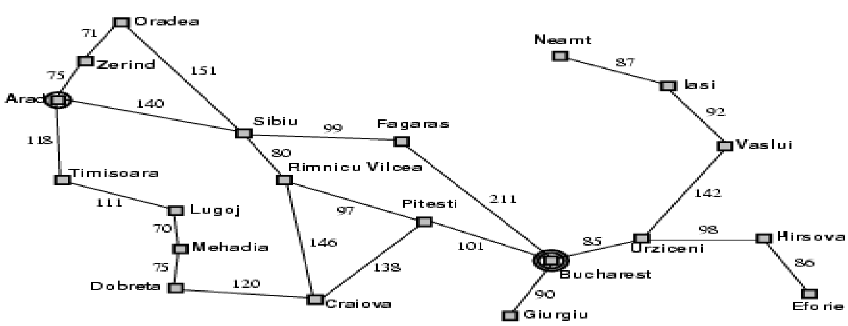
- We solve problems by searching the state space
- State space is represented by a graph or a tree
- We start at root node, check if it is a goal state
- If not, apply the Result/successor function to generate new state (node)
- The choice of which state to expand is determined by the search strategy

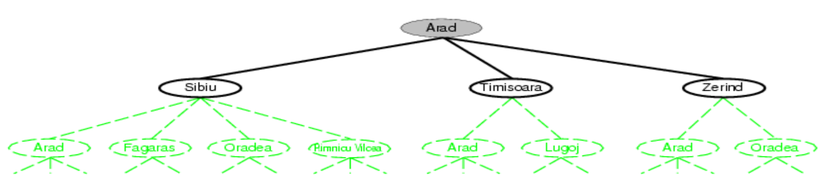
Tree Search Example



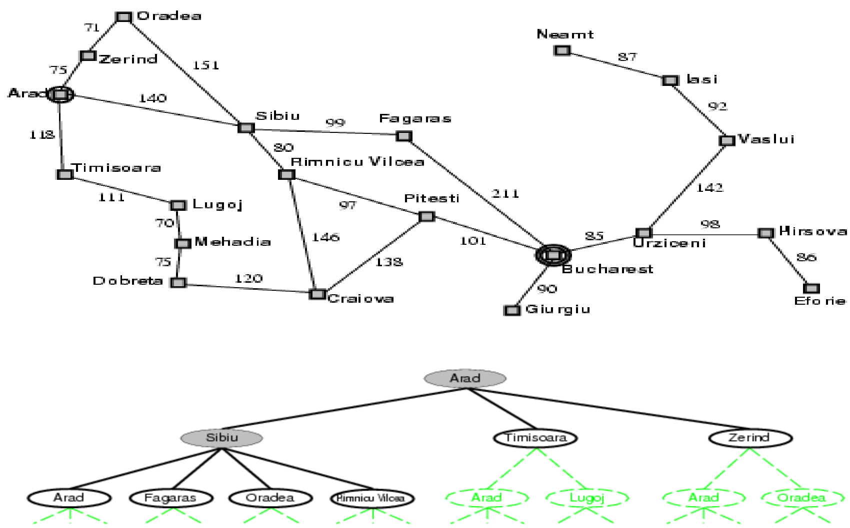


Tree Search Example





Tree Search Example



Assumptions about Node

- Node is a data structure with five components
  - *State: the state in the state space to which node correspond*
  - *Parent node: the node which generated this node*
  - *Action: the action that was applied to parent to generate this*
  - *Path-cost: the cost, denoted by g(n), from initial state to this node*
  - *Depth: the number of steps along the path from initial state*

Fringe

- We shall use a data structure called "fringe"
- This will contain the collection of nodes which has been generated but not expanded
- The search strategy would decide next node to be expanded from this list
- Assume collection of nodes is represented as a *Queue*.

Search Strategies Evaluation

- Strategies are evaluated along the following dimensions:
  - *completeness: does it always find a solution if one exists?*
  - *time complexity: how long does it take*
  - *space complexity: how much memory is needed*
  - *optimality: does it always find a least-cost solution?*
- Time and space complexity are measured in terms of
  - *b: maximum branching factor of the search tree*
  - *d: depth of the shallowest goal-node*
  - *m: maximum depth of the tree (may be ∞)*

Measuring Performance

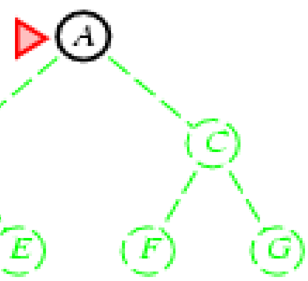
- Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory
- Total cost = search cost + path cost

Uninformed Search Strategies

- Uninformed (Blind) search strategies use only the information available in the problem definition
  - *Breadth-first Search*
  - *Uniform-cost search*
  - *Depth-first search*
  - *Depth-limited search*
  - *Iterative deepening search*

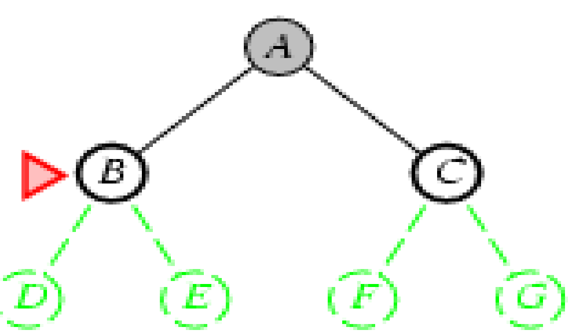
Breadth-first Search

- Expand shallowest unexpanded node first
- Implementation:
  - *Put the newly expanded node at the back of the fringe*
  - *use the fringe as a Queue, FIFO*
  - *Fringe = A*



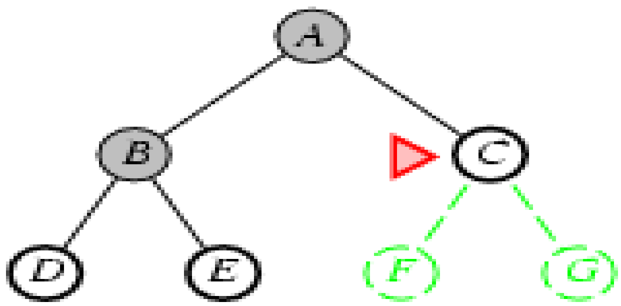
Breadth-first Search

- Expand shallowest unexpanded node first
- Check if A is goal, it is not, expand A
- Fringe = B,C



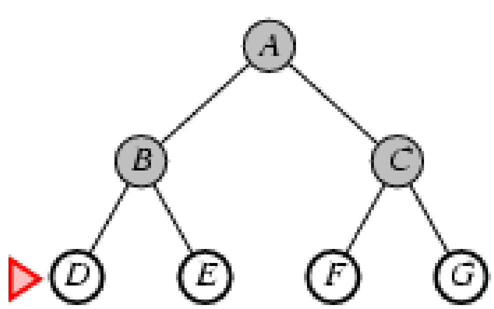
Breadth-first Search

- Check if B is goal, it is not, expand B
- Fringe = C, D, E
- Check if C is goal, it is not, expand C
- Fringe = D, E, F, G



Breadth-first Search

- Expand shallowest unexpanded node
- Check if D is goal, it is not, no children
- Fringe = E, F, G
- Search goes on until goal is found. Once it is found, the search terminates



Properties of Breadth-first Search

- Complete: Yes (if b is finite)
- Time: $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space: $O(b^{d+1})$ (keeps every node in memory)
- Optimal: Yes (if cost = 1 per step)

- Space is the bigger problem (more than time)

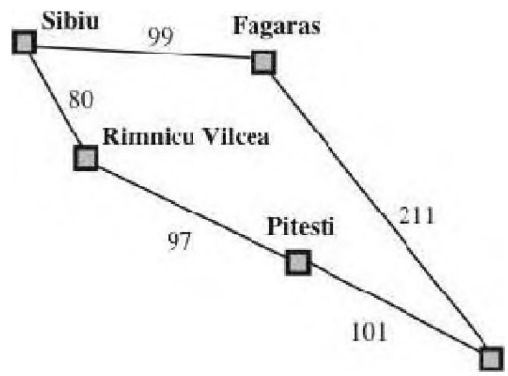
Breadth-first Search

Depth	Nodes	Time	Memory
2	1111	1 Secs	1 MB
4	111,100	11 Secs	106 MB
6	10^7	19 Mins	10 GB
8	10^9	31 Hrs	1 TB
10	10^11	120 Days	101 TB
12	10^13	35 Yrs	10 PB
14	10^15	3,523 Yrs	1 XB

Time & Memory requirements for BFS. The numbers shown assume branching factor b = 10; 10,000 nodes per second expansion; 1,000 bytes per node

Uniform-cost Search

- Expands a node with the lowest path cost
- Equivalent to breadth-first if all step costs are equal
- Implementation:
  - *fringe = queue ordered by path cost*



Assignment # 1

- Deliverables:
  - *Part I to be solved on paper*
  - *Print out of Code in Part II with Comments and Output*

- Due after 2 weeks (Wed, 25^{th} Oct)