Lecture 4-5A

Problem Solving by Searching

State space graphs and uninformed search

Textbook: **Artificial Intelligence, A Modern Approach, IV edition**
Authors: **Russell and Norvig**
Reading material: **Chapter 3, sections 3.1 to 3.4, pp. 63-84**

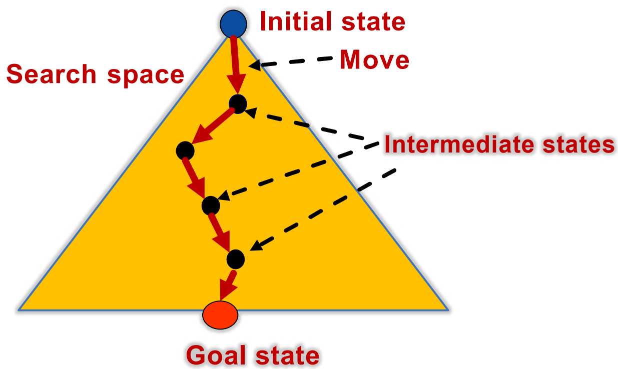
1

Outline

- State space graph
  - **state** is a set of variable-value pairs; graph-nodes are states
  - A move changes the value of a variable and transitions to new state
  - edge is a change in one or more values associated with variables
  - **goal** is when state meets the **final condition**
  - Components of a state-space graph: initial state, set of states, moves that change one or more states, goals
  - There is a cost associated with making a move; an edge may have varying weight
  - Total cost is sum of the cost of the path from initial state to the goal state
- Search strategies
  - exhaustive/blind search: depth first; breadth-first and their variations
- issues in graph-based search
  - how to prune the search space without sacrificing a solution

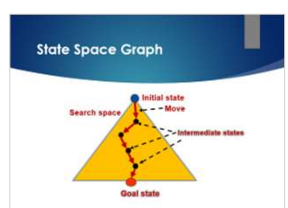
2

State Space Graph



3

AI Search



- Plays an important role in AI
- Key idea includes
  - *"mathematically sound estimate of distance of the current state to the goal state"*
  - *"prune search space and remain focused"*
  - *smaller the difference between estimated distance and minimal distance, better pruning is achieved*
- Applicability and advantages
  - real-world problems that require **exponential time**
  - handling very large search space where algorithmic approaches will consume unrealistic time

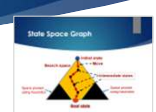
4

AI Search Problems

- Finding the best route from one node to another node towards a goal state based upon a well-defined notion such as
  - minimum time / minimum distance / minimum cost
- Finding the best strategy to win a game / solve a puzzle
- Finding the solution to a logical query
- Finding the solution under constrained conditions, such as
  - laying the printed circuit board and VLSI design without jumpers
  - putting the fixed number of workers on a complex task
  - carrying maximum passengers with minimum planes
  - adjusting traffic lights time to minimize delay and congestion
  - adjusting electric grid to minimize power outages

5

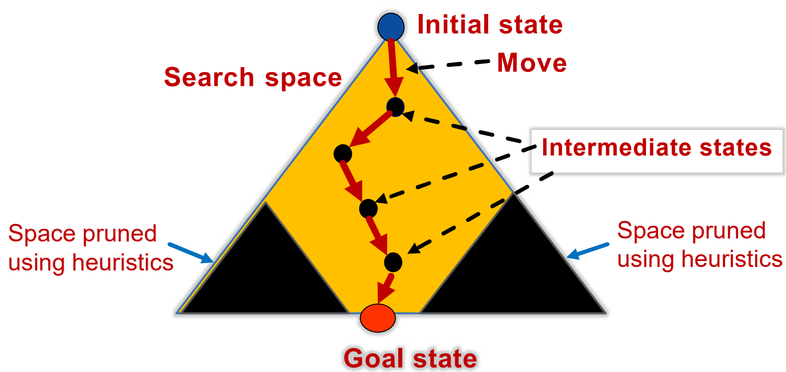
Heuristic Search Strategy



- Identify the problem
- Abstract and formulate the problem
  - extract the features for identifying the variables to form states
  - identify set of states, initial state, final state (goal state), and sets of incremental moves to take to next state
- Start with the initial state
- At every node, test if the 'goal state' has been reached
- Find the sequence of moves that will
  - take from initial state to goal state in a focused manner using heuristics functions
  - test and remove cyclic traversal – archive visited nodes in a set and perform membership test to avoid visiting visited nodes
  - minimize the total cost according to some criteria
  - search in reasonable time and memory as desired

6

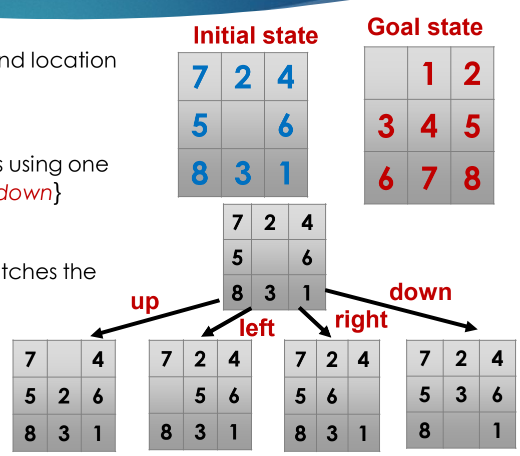
State Space Graph



7

Example: Sliding Puzzle

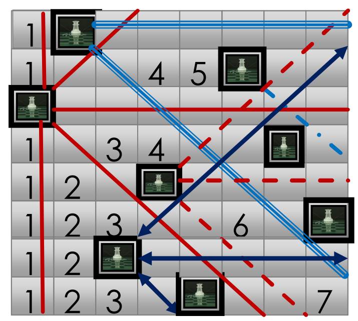
- States:
  - description of the eight tiles and location of the blank tile
- Successor function:
  - generates intermediate states using one of four moves: {*left, right, up, down*}
- Goal test:
  - checks if the current state matches the final state
- Path cost:
  - each move is unit cost
  - total cost = number of moves



8

Example: Eight Queens

- Place eight queens on a chess board such that no queen can attack another queen
  - naïve number of states: 64 * 63 * 62 * 61 * 60 * 59 * 58 * 57 = **1.78 * 10^{14}**
- Intelligent formulation with pruning
  - place one queen at a time
  - do not place a queen in square if it is attacked by other queens
  - new number of states is around 8 * 5 * 4 * 3 * 2 * 2 * 1 * 2 = **3840 states**



9

Traversal Planning

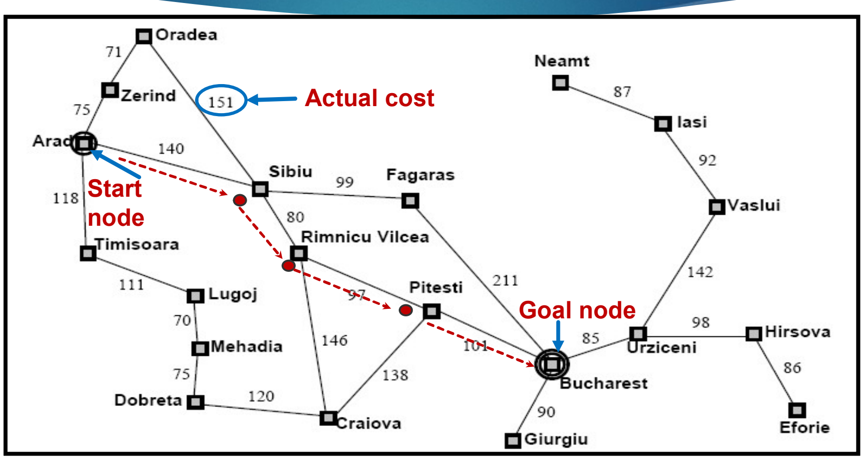


Figure taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

10

Search Strategies

- Exhaustive search (blind search)
  - uninformed no cost estimated used to reach goal state
  - **Breadth-first, uniform-cost, depth-first, limited depth-first, iterative deepening**
- Search space pruning (informed search using heuristics)
  - focused movement towards the goal
  - uses a heuristic cost-function that estimates of distance from the current state to the goal state using mathematical abstraction
  - total cost = actual cost of traversal from initial state to next state + estimated cost from next state to goal state
  - assumption: actual cost from current state to next state is known
- Two major types of heuristic searches called **best first search**

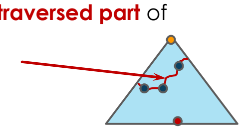
  Greedy **best first**, estimated cost from the current state to the goal state is minimal

  A* algorithm: total cost from any traversed node to goal-state is minimal. Hence, A* always finds the optimal path (at the cost of additional memory)

11

Terminology Used

- Complete : search finds out all the existing goal states / final states
- Incomplete: search misses out at least one goal state / final state
- Optimal search
  - finds out the least-cost solution by expanding the right nodes.
- Heuristics function: estimates distance from current state to goal state
- Pruning
  - search does not traverse a subset of nodes that either has a higher total cost or do not lead to goal-state based upon heuristic-function estimates
- Frontier (or fringe) – set of leaf nodes of the **currently traversed part** of the tree



12

General Tree Search Strategy

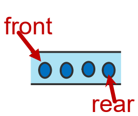
function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose one node from frontier according to the current strategy
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end

Note: Tree is a subset of directed acyclic graphs (DAGs). The algorithm can be generalized to any DAG traversal.

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

13

Generic Tree-search Algorithm

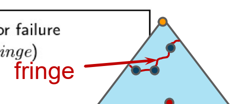


fringe
queue

Initially fringe is empty

function TREE-SEARCH(*problem, fringe*) **returns** a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if *fringe* is empty **then return** failure
    node ← REMOVE-FRONT(*fringe*)
    If goal_test(state(node)) then return(node)
    fringe ← INSERTALL(EXPAND(*node, problem*), *fringe*)

function EXPAND(*node, problem*) **returns** a set of nodes
  successors ← the empty set
  for each *action, result* **in** SUCCESSOR-FN[*problem*](STATE[*node*]) **do**
    s ← a new NODE
    PARENT-NODE[*s*] ← *node*; ACTION[*s*] ← *action*; STATE[*s*] ← *result*
    PATH-COST[*s*] ← PATH-COST[*node*] + STEP-COST(*node, action, s*)
    DEPTH[*s*] ← DEPTH[*node*] + 1
    add *s* to *successors*
  return *successors*



Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

14

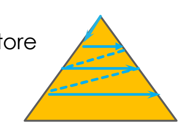
Search Strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated using the following criteria:
  - **completeness:** does it always find a solution if one exists?
  - **time complexity:** number of nodes generated and traversed
  - **space complexity:** maximum number of nodes in the fringe queue
  - **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 least-cost solution
  - **m:** maximum depth of the state-space (may be **very large**)

15

Exhaustive / Blind Search

- Does not use heuristic function. Traverses potentially every node.
- Breadth-first search (BFS)
  - search all nodes level by level ; uses a fringe queue to store all the children of the expanded unexplored node



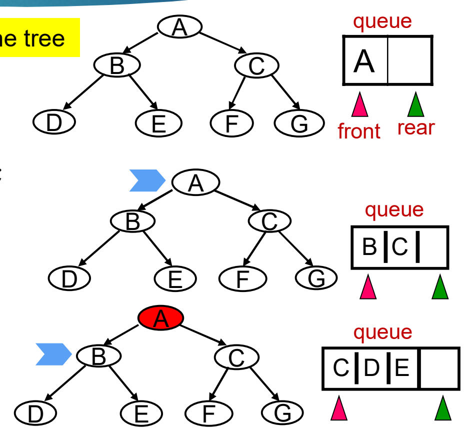
- Uniform cost search
  - a variation of breadth-first search that picks the shortest-path node during node expansion – replace queue by a priority queue
- Depth-first search (DFS)
  - search the leftmost children first (similar to pre-order traversal)
  - uses stack to store deferred right siblings
- Depth-limited search (DLS)
  - a variation of depth-first search that limits the depth being searched
- Iterative deepening search (IDS or IDDFS)
  - iterative deepening of depth and performing depth-first search

16

Breadth-first Search

Assumption: Goal state is the leaf node of the tree

1. Put the root node in the queue;
2. **Loop** % traverse level wise
   1. pick the next node from the queue;
   2. **If** it is a leaf node **then**
      If node is a goal-state **then**
      return(node) and **exit**;
   3. **else if** it a non-leaf node **then**
      find all the children of the node,
      and insert them in the queue; **}**
3. **until** the queue is empty
5. If queue is empty then return **failure**;



17

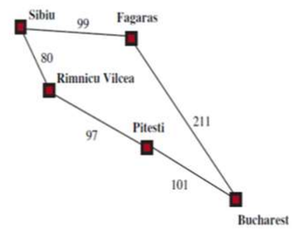
Properties of Breadth-first Search

- Complete? Yes (if branching factor b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d = O(b^{d+1})$ where d is depth of the tree
- Space? $O(b^{d+1})$ (keeps every unexpanded node in the queue)
- Optimal? Yes (if cost = 1 per step)

18

Uniform-cost Search

- Expand **least-cost unexpanded node**
  - use priority queue with **least path cost from root to the current node**
- Implementation:
  - **fringe = queue sorted in ascending order**
  - **goal-node test done before expansion and not after generation**



- Becomes breadth-first if step costs all equal
- Complete? Yes
- Time? # O($\mathbf{b}^{\text{ceiling}(C^*/\,\varepsilon)}$); C^* is cost of the optimal solution
- Space? # O($\mathbf{b}^{\text{ceiling}(C^*/\,\varepsilon)}$)
- Optimal? Yes – nodes expanded in increasing order of cost

Figure taken from the book “Artificial Intelligence – A modern Approach by Russell and Norwig, 4th edition

19

Uniform Cost Search Algorithm

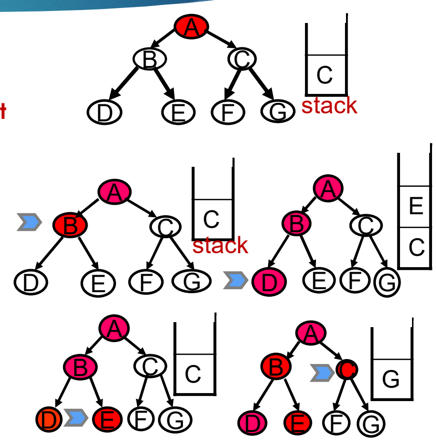
```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
         child ← CHILD-NODE(problem, node, action)
         if child.STATE is not in explored or frontier then
             frontier ← INSERT(child, frontier)
         else if child.STATE is in frontier with higher PATH-COST then
             replace that frontier node with child
```

Algorithm taken from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

20

Depth-first Search

- cn = root-node; goal-found = false;
- **loop**
  - **If (**cn is a leaf node **&&** goal-state(cn)**)**
    goal-found = true; **return(**cn**)** and **exit**
  - **elseif** (leaf(cn) **&&** ¬ empty(stack))
    cn = pop(stack);
  - **elseif** ¬ has-left-child**(**cn**)**
    cn = right-child (cn);
  - **else** push(stack, right-sibling(cn));
  - cn = left-child (cn);
- **until** ((empty(stack) **&&** leaf(cn))
- **If** (empty(stack)) && leaf(cn)) **return failure.**



21

Properties of Depth-first Search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - avoid repeated states along path using a lookup table of visited nodes
  - Generally, stops after finding the first solution
- Worst case Time? O(b^m). Excessive if m is much larger than d
  - generally faster than breadth-first because all branches are not explored
  - Best case time: O(bm) – when the goal is near the leftmost leaf node
- Space? O(bm), i.e., linear stack space guided by depth of the tree
- Optimal? No, solution may be at higher level on the right side

22

Depth-limited Search

- Depth-first search with depth-limit L < total depth of the tree,
  - nodes beyond depth L are not traversed
- Alleviates the problem of non-termination with indefinite branches
- Improves the search time
- **Problem with completeness**
  - can not find a goal-state if the goal-state is beyond depth L

23

Depth Limited Search Recursive Algorithm

function DEPTH-LIMITED-SEARCH(*problem*, *limit*) **returns** a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(*problem*.INITIAL-STATE), *problem*, *limit*)

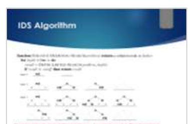
function RECURSIVE-DLS(*node*, *problem*, *limit*) **returns** a solution, or failure/cutoff
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  else if *limit* = 0 **then return** *cutoff*
  else
    cutoff_occurred? ← false
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← CHILD-NODE(*problem*, *node*, *action*)
      result ← RECURSIVE-DLS(*child*, *problem*, *limit* − 1)
      if *result* = *cutoff* **then** *cutoff_occurred?* ← true
      else if *result* ≠ *failure* **then return** *result*
    if *cutoff_occurred?* **then return** *cutoff* **else return** *failure*

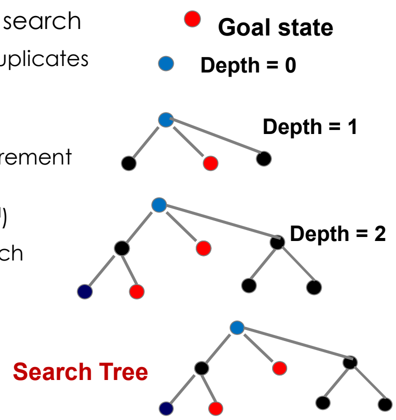
Algorithm taken from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

24

Iterative Deepening Search (IDS)

- Iteratively perform depth-limited search
  - Level = 0, level = 1, level = 2 … duplicates traversal of previous levels
- Advantages
  - Uses stack reducing space requirement to O(bd)
  - Traversed nodes limited by $O(b^d)$
  - much less than breadth-first search
  - Optimal like breadth-first search





25

IDS Algorithm

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
```

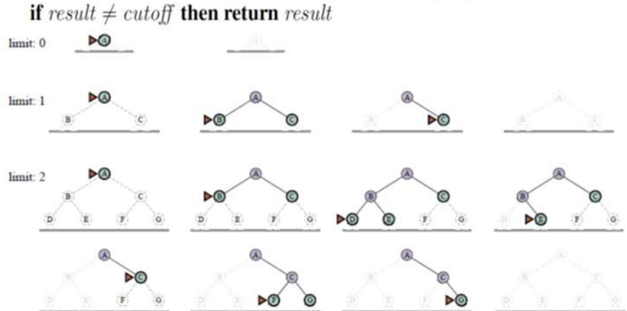


Figure taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

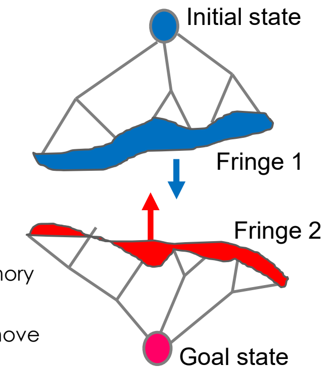
26

Bidirectional Search

- Start in both the directions
  - **goal-state → initial state and**

    initial-state → goal-state

  Meet somewhere halfway
- Expand until the two fringes intersect
- Complexity: $O(b^{d/2})$
- Disadvantages
  - lot of memory needed to keep fringes in memory for intersection
  - overhead of finding intersection after every move
  - backward moves from goal-states not always easy to figure out



27

Comparing Uninformed Search Techniques

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Table taken from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

28

Graph-based Search

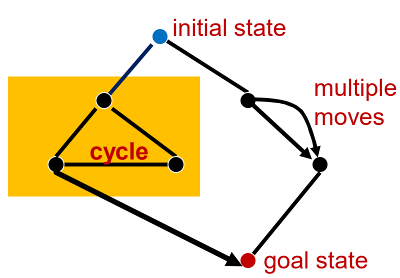
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
   closed ← an empty set ; fringe ← an empty set;
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
      if STATE[node] is not in closed then
         add STATE[node] to closed
         fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

29

Handling Repeated States

- Caused by
  - cycles in the search path
  - multiple moves between two states giving rise to duplications
- Causes looping and/or exponential explosion of states
- Solution
  - keep history of traversed nodes
  - **perform membership test** using hashing for fast search



30