

# Artificial Intelligence

# Fahad Sherwani

## Course Contents

- Introduction
- Intelligent Agents
- Search
  - Problem solving through search
  - Uninformed search
  - Informed search
  - Local search and constraint Satisfaction
- Games
  - games as search problems
- Knowledge and Reasoning
  - reasoning agents
  - propositional logic
  - predicate logic
  - planning
  - knowledge-based systems
  - uncertain knowledge and reasoning
- Machine Learning
  - learning from observation
  - reinforcement learning
  - neural networks
  - deep learning
- (Natural Language Processing)
- (Robotics)
- (Philosophical, Ethical, Social Issues with AI)
- Conclusions

# AI

↳ aims to replicate human intelligence

↳ Reasoning → how knowledge is acquired, represented and stored

↳ Learning → learn and store things, so won't make same mistake  
how intelligent behaviour is generated and learned

↳ Problem Solving ↴

↳ Perception → to reason about past and plan for future

## STRONG AI

↳ machine that superseeds human intelligence

↳ if it can

- do typical human tasks
- apply a wide range of background and knowledge
- has some degree of self-consciousness

↳ it is indistinguishable from that of a human

## WEAK AI

↳ use of software to accomplish tasks that do not encompass the full range of human cognitive abilities

↳ does not achieve self-awareness

↳ demonstrates few of human level cognitive abilities

## AI Goals

↳ replicate human intelligence

↳ solve knowledge intensive tasks

↳ make intelligent connection b/w perception and action

- Enhance human-human, human-computer and computer to computer interaction / communication.
- Engineering based AI Goal
- Develop concepts, theory and practice of building intelligent machines
- Emphasis is on system building.
- Science based AI Goal
- Develop concepts, mechanisms and vocabulary to understand biological intelligent behavior.
- Emphasis is on understanding intelligent behavior.

→ Think human like

## COGNITIVE SCIENCE

- Goal is not just to produce human-like behavior but to produce a sequence of steps of the reasoning process, similar to the steps followed by a human in solving the same task.

→ Think Rationally

## LAWS OF THOUGHT

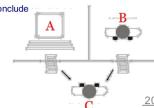
- Goal is to formalize the reasoning process as a system of logical rules and procedures for inference.

# Turing Test

- The art of creating machines that perform functions requiring intelligence when performed by people; that it is the study of, how to make computers to do things which at the moment people do better.
- A Behaviorist approach, is not concerned with how to get results but to the similarity to what human results are...

## Example : Turing Test

- 3 rooms contain: a person, a computer, and an interrogator.
- The interrogator can communicate with the other 2 by teletype (to avoid the machine imitate the appearance or voice of the person).
- The interrogator tries to determine which is the person and which is the machine.
- The machine tries to fool the interrogator to believe that it is the human, and the person also tries to convince the interrogator that it is the human.
- If the machine succeeds in fooling the interrogator, then conclude that the machine is intelligent.
- Goal is to develop systems that are human-like.



20

## 1. What is the common definition of "AI"? Do you agree?

Artificial Intelligence (AI) is the simulation of human intelligence processes by machines, especially computer systems. Yes, I agree with this definition.

## 2. Do you know any AI application?

One example of an AI application is virtual personal assistants like Siri, Alexa, or Google Assistant.

## 3. Should artificial intelligence simulate natural intelligence?

Yes, the goal of AI is often to simulate or replicate aspects of natural human intelligence.

## 4. What are the criticisms on the AI research? Do you agree?

Some criticisms include concerns about job displacement, ethical implications, biases in algorithms, and potential misuse of AI technologies. Yes, I agree with these criticisms.

## 5. What is the relation between AI and logic? AI and philosophy? Logic and philosophy?

AI often relies on logical reasoning as a foundational component, and there are philosophical implications regarding the nature of intelligence and consciousness in AI research.

## 6. Explain the meaning of logic? reasoning? ontology?

Logic: The study of principles of correct reasoning. Reasoning: The mental process of drawing conclusions or making inferences from available information. Ontology: The branch of philosophy that deals with the nature of being, existence, and reality.

## 7. What is Natural Language Processing? And how is it related to AI?

Natural Language Processing (NLP) is a subfield of AI that focuses on the interaction between computers and humans through natural language. It involves tasks such as text understanding, language generation, and machine translation.

## 8. Why and how are Probabilistic and statistical methods used in AI?

Probabilistic and statistical methods are used to deal with uncertainty and incomplete information in AI systems. They help in decision-making, pattern recognition, and learning from data.

## 9. What are the major research approaches/schools in AI? Which one do you think is more productive?

Some major approaches include symbolic AI (based on logic and rules), connectionism/neural networks, evolutionary computation, and Bayesian networks. Each has its strengths and weaknesses, but the neural network approach has seen significant productivity and success in recent years.

# CMP 2

## Agents

- ↳ Anything that perceives environment through sensors
- ↳ And acts upon that environment through actuators

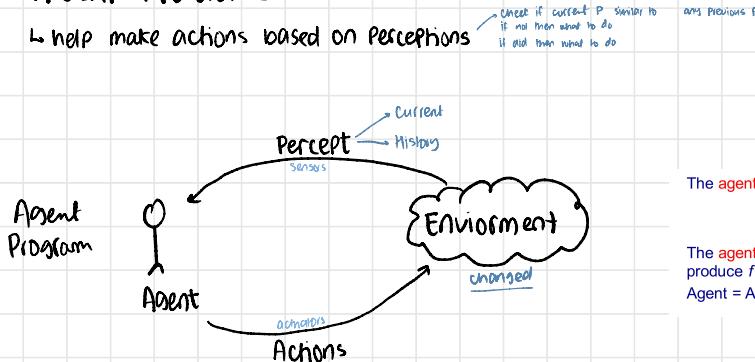
**Human agent:** Eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators.

**Robotic agent:** Cameras and infrared range finders for sensors; various motors for actuators.

**Software agent:** Software agent is a computer program that acts for a user or other program: an agreement to act on one's behalf.

## Agent Programs

- ↳ help make actions based on perceptions



Agent → Percepts → Decision → Actions

The **agent function** maps from percept histories to actions:  
 $[f: P^* \rightarrow A]$

The **agent program** runs on the physical **architecture** to produce  $f$   
**Agent** = Architecture + Program

when designing an agent  
we keep in mind

P E A S  
 ↓ ↓ ↓  
 Performance Environment Actuators Sensors

## Goals of Agents

- ↳ High Performance
- ↳ Optimized Result
- ↳ Rational Action  
 ↳ right action

### Agent: Part-picking robot

- Performance measure: Percentage of parts in correct bins
- Environment: Conveyor belt with parts, bins
- Actuators: Jointed arm and hand
- Sensors: Camera, joint angle sensors

### Agent: automated taxi driver

- Performance measure: Safe, fast, legal, comfortable trip, maximize profits
- Environment: Roads, other traffic, people and objects in/around the street
- Actuators: Steering wheel, accelerator, brake, signal, horn
- Sensors: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

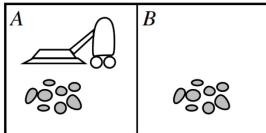
### Agent: Interactive English tutor

- Performance measure: Maximize student's score on test
- Environment: Set of students
- Actuators: Screen display (exercises, suggestions, corrections)
- Sensors: Keyboard

### Agent: Medical diagnosis system

- Performance measure: Healthy patient, minimize costs, lawsuits
- Environment: Patient, hospital, staff
- Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
- Sensors: Keyboard (entry of symptoms, findings, patient's answers)

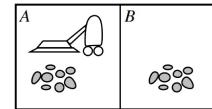
## Vacuum-cleaner world



Percepts: location and contents, e.g., [A, Dirty]

Actions: *Left, Right, Suck, DoNothing*

## A vacuum-cleaner Agent



Tabulation of an agent function of the vacuum-cleaner

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
[ ]	[ ]

function REFLEX-VACUUM-AGENT([location,status]) returns action

```

if status = Dirty then return Suck
else if location = A then return Right
else if location = B then return Left

```

## Rational Agents

An agent should strive to "do the right thing", based on what it can perceive and the actions it can perform. The right action is the one that will cause the agent to be most successful.

Performance measure: an objective criterion for success of an agent's behavior.

E.g., *performance measure* of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.

**Rational Agent:** For each possible percept

sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

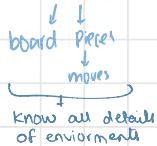
An agent is **autonomous** if its behavior is determined by its own experience (with ability to learn and adapt).

# Environment Types

## 1. Fully Observable

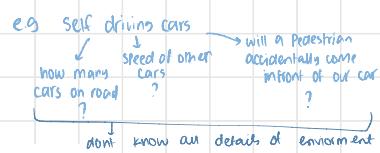
- ↳ sensors capture all relevant info from the environment

e.g chess, tic tac toe



VS

## 2. Partially Observable



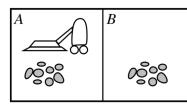
Tic Tac Toe is Fully Observable. Cards are Partially Observable.

## 3. Deterministic

- ↳ changes in the environment are predictable
- ↳ the next state is completely determined by the current state

VS

## 4. Stochastic → Non deterministic



Vacuum is deterministic. Taxi driver is stochastic.

## 5. Episodic

- ↳ independent perceiving-action episodes
- ↳ agents experience is divided into atomic 'episodes'
- ↳ the choice of action in each episode depends on the episode itself
- ↳ takes one step

VS

## 6. Sequential → non Episodic



Robot is Episodic. Taxi driver is sequential.

## 7. Static

- ↳ no changes in the environment while the agent is thinking

VS

## 8. Dynamic



can think and drive



can't think and move pieces together

Taxi driver is dynamic. Chess is static.

## 9. Discrete

- ↳ limited no. of distinct percepts/actions

VS

## 10. Continuous



continuous

Chess has a finite number of distinct states, thus it is discrete; however the Taxi-driving is not. ??

24

## 11. Single Agent

VS

## 12. Multiagents

- ↳ interaction and collaboration among agents
- ↳ competitive, cooperative
- ↳ an agent operating by itself in an environment



Crossword is Single agent, while Chess is a multi-agent environment.

2

## Environment Types

*Task Environment*

Partially observable  
Stochastic  
Sequential  
Deterministic  
Episodic  
Static  
Discrete

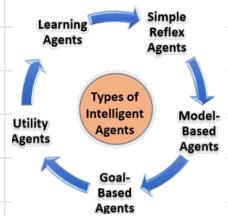
Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Dynamic	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

## Agent design

- The environment type largely determines the agent design

- **Partially observable** => agent requires **memory** (internal state)
- **Stochastic** => agent may have to prepare for **contingencies**
- **Multi-agent** => agent may need to behave **randomly**
- **Static** => agent has time to compute a rational decision
- **Continuous time** => continuously operating **controller**

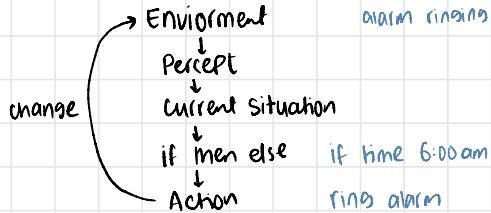
# AGENT TYPES



## 1. SIMPLE REFLEX AGENTS

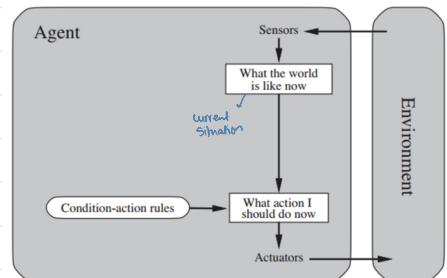
- ↳ acts only on the basis of current perception
- ↳ ignores rest of the percept history → hence very limited knowledge
- ↳ if-else rules  
→ rule matches since it's fully observable
- ↳ partially observable environment

e.g. sneeze, alarm clock



## CONS

- ↳ infinite loops are unavoidable
- ↳ very limited knowledge
- ↳ no perception about prev or next state



## 2. MODEL BASED REFLEX AGENTS

- ↳ acts on the basis of current perception + perception history

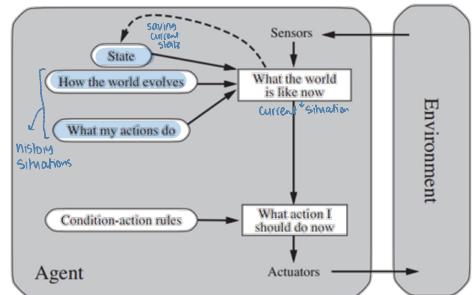
↳ state is evaluated based on  
how it changed from previous state

↳ partially observable environment

↳ stores percept history

e.g. cleaner robot, telephone answering machine

- Know how world evolves
  - Overtaking car gets closer from behind
- How agents actions affect the world
  - Wheel turned clockwise takes you right
- Model base agents update their state



### 3. Goal based Agents

↳ Known goal → agent decides actions based on known goal

↳ Focuses on Goal

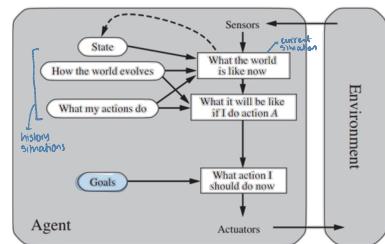
## ↳ Searching and Planning

- find best route
  - safe and fun route

How to get to linear  
 • which bike  
 • where to stop  
 • where fuel stations

## ↳ Partially Observable Environment

e.g GPS system to find a path to a certain destination



## 4. Utility based Agents

<sup>object</sup> ↳ acts on the basis of utilities / Preferences

↳ Focuses on Utility not goal

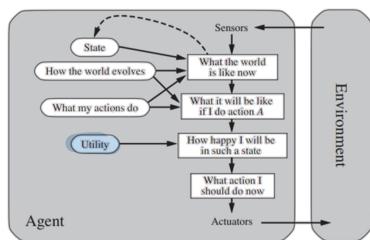
Utility function

↳ finds Optimally better state for

### ↳ Partially Observable Environment

e.g GPS system finding fastest path

to a certain destination



## 5. Learning Agents

↳ adapts its actions based on feedback

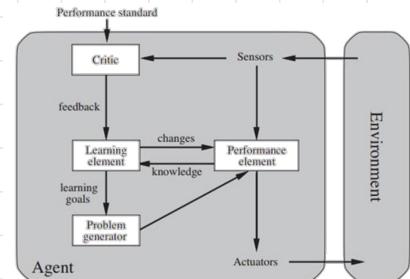
Learning element: makes improvements from past events

↳ Performance element: takes **Percept** and then decide on actions

↳ Critic: gives learning element feedback → what needs to be modified

↳ Problem generator: suggests actions that lead to new and informative experiences

e.g. human agent



## Summary

- Agents interact with environments through actuators and sensors
- The agent function describes what the agent does in all circumstances.
- The performance measure evaluates the environment sequence.
- A perfectly rational agent maximizes expected performance.
- Agent programs implement (some) agent functions PEAS descriptions define task environments.
- Environments are categorized along several dimensions:
  - Observable? deterministic? episodic? static? discrete? single-agent?
- Several basic agent architectures exist:
  - Simple Reflex, Model based, goal-based, utility-based, Learning

# Searching

↳ falls under AI

## Searching in AI

- Optimization.** Search algorithms are the basis for many optimization and planning methods.
- Solve a problem by searching for a solution.** Search strategies are important methods for many approaches to problem-solving.
- Problem formulation.** The use of search requires an abstract formulation of the problem and the available steps to construct solutions.

### Searching for Solutions

#### Traversal of the search space

- From the initial state to a goal state.
- Legal sequence of actions as defined by successor function.

#### General procedure

- Check for goal state
- Expand the current state
  - Determine the set of reachable states
  - Return "failure" if the set is empty
- Select one from the set of reachable states
- Move to the selected state

#### A search tree is generated

- Nodes are added as more states are visited

### Problem Formulation

To solve a problem by search, we need to first formulate the problem.

HOW?

Our textbook suggest the following schema to help us formulate problems:

- State
- Initial state
- Actions or Successor Function
- Goal Test
- Path Cost
- Solution

### Search Terminology

#### Search Tree

- Generated as the search space is traversed
- The search space itself is not necessarily a tree, frequently it is a graph
- The tree specifies possible paths through the search space

#### Expansion of nodes

- As states are explored, the corresponding nodes are expanded by applying the successor function
  - this generates a new set of (child) nodes
- The fringe (frontier/queue) is the set of nodes not yet visited
  - newly generated nodes are added to the fringe

#### Search strategy

- Determines the selection of the next node to be expanded
- Can be achieved by ordering the nodes in the fringe
  - e.g. queue (FIFO), stack (LIFO), "best" node w.r.t. some measure (cost)

## State Space Search

→ no of states in which the problem can go

$$S: \{ S, A, \text{Action}(s), \text{Result}(s,a), \text{Cost}(s,a) \}$$

S: start state, goal state, intermediate states

A: all possible actions  
e.g. for 8 Puzzled Problem  
Action: up, down, left, right

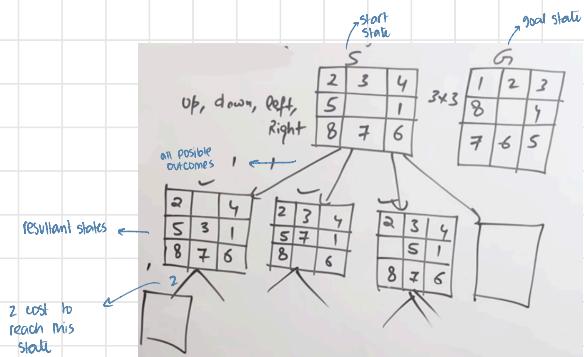
Actions(s): the chosen actions

Result(s,a): the resultant state

Cost(s,a): get minimum cost

↳ Precisely

↳ analyse



# 8 PUZZLE PROBLEM

↳ Blind Search → Uninformed

↳ BFS

↳ 4 moves (up, down, left, right)

## Problem Formulation (The 8-Puzzle Example)

**State:** The location of the eight tiles, and the blank



**Initial State:**  $\{(7,0), (2,1), (4,2), (5,3), (1,4), (6,5), (8,6), (3,7), (1,8)\}$

**Successor Function:** one of the four actions (blank moves Left, Right, Up, Down).

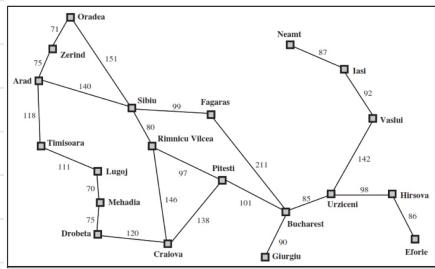
**Goal Test:** determine if a given state is a goal state.

**Path Cost:** each step costs 1

**Solution:**  $\{(\_,0),(1,1),(2,2),(\_,3,3),(4,4),(\_,5,5),(\_,6,6),(\_,7,7),(\_,8,8)\}$

11

## Problem Formulation (The Romania Example)



**Solution:** a sequence of actions leading from the initial state to a goal state  
(Arad → Sibiu → Rimnicu Vilcea → Pitești → Bucharest)

8

## Problem Formulation (Real-life Applications)

### Route Finding Problem



Car Navigation



Airline travel planning



Military operation planning

- States

- locations

- Initial state

- starting point

- Successor function (operators)

- move from one location to another

- Goal test

  - arrive at a certain location

- Path cost

  - may be quite complex
  - money, time, travel comfort, scenery,

12

## Problem Formulation (Real-life Applications)

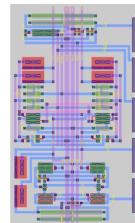
### Travel Salesperson Problem



- States
  - locations / cities
  - illegal states
    - each city may be visited only once
    - visited cities must be kept as state information
- Initial state
  - starting point
  - no cities visited
- Successor function (operators)
  - move from one location to another one
- Goal test
  - all locations visited
  - agent at the initial location
- Path cost
  - distance between locations

## Problem Formulation (Real-life Applications)

### VLSI layout Problem

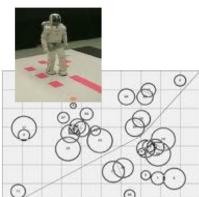


- States
  - positions of components, wires on a chip
- Initial state
  - incremental: no components placed
  - complete-state: all components placed (e.g. randomly, manually)
- Successor function (operators)
  - incremental: place component, route wire
  - complete-state: move component, move wire
- Goal test
  - all components placed
  - components connected as specified
- Path cost
  - maybe complex
    - distance, capacity, number of connections per component

13

## Problem Formulation (Real-life Applications)

### Robot Navigation



- States
  - locations
  - position of actuators
- Initial state
  - start position (dependent on the task)
- Successor function (operators)
  - movement, actions of actuators
- Goal test
  - task-dependent
- Path cost
  - maybe very complex
    - distance, energy consumption

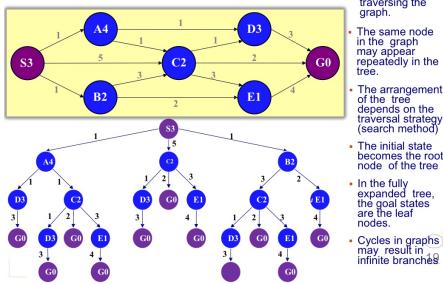
## Problem Formulation (Real-life Applications)

### Automatic Assembly Sequencing



- States
  - location of components
- Initial state
  - no components assembled
- Successor function (operators)
  - place component
- Goal test
  - system fully assembled
- Path cost
  - number of moves

### Traversing a Graph as Tree



### Prim's Vs Kruskal's

#### Prim's Algorithm

It starts to build the Minimum Spanning Tree from any vertex in the graph.

It traverses one node more than one time to get the minimum distance.

Prim's algorithm has a time complexity of  $O(V^2)$ ,  $V$  being the number of vertices and can be improved up to  $O(E + \log V)$  using Fibonacci heaps.

Prim's algorithm gives connected component as well as it works only on connected graph.

Prim's algorithm runs faster in dense graphs.

#### Kruskal's Algorithm

It starts to build the Minimum Spanning Tree from the vertex carrying minimum weight in the graph.

It traverses one node only once.

Kruskal's algorithm's time complexity is  $O(E \log V)$ ,  $V$  being the number of vertices.

Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components

Kruskal's algorithm runs faster in sparse graphs.

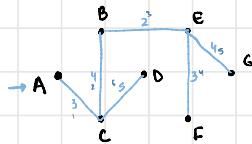
## 1. PRIM'S ALGO

↳ start with any node

↳ disconnectedly move to smallest neighbour

↳ no cycles

↳ edges =  $n - 1$



$$MST = 21$$

### Prim's Algorithm

Prim's Algorithm includes 4 Steps:

**Step 1:** Draw an  $n \times n$  ( $n \times n$ ) vertices' matrix and label them as  $V_1, V_2, \dots, V_n$  along with the given weights of the edges.

**Step 2:** Starting from vertex  $V_1$ , and connect it to its nearest neighbor by searching in row 1.

**Step 3:** Consider  $V_1$  and  $V_i$  as one subgraph and connect as step 2 while do not forming any circuit.

**Step 4:** Continue this process until we get the MST having  $n$  vertices and  $(V_n - 1)$  edges.

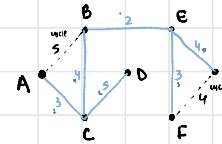
## 2. KRUSKAL ALGO

↳ start with smallest edge

↳ disconnectedly mark smallest edge

↳ no cycles

↳ edges :  $n - 1$



$$MST = 21$$

state  
Initial  
Succ  
Goal  
Path cost  
Solt

### Kruskal's Algorithm

Kruskal's Algorithm includes 4 Steps:

**Step 1:** List all the edges of the graph in order of increasing weights.

**Step 2:** Select the smallest edge of the graph.

**Step 3:** Select the next smallest edge that do not makes any circuit.

**Step 4:** Continue this process until all the Vertices are explored and  $(V_n - 1)$  edges have been selected

# SEARCHING

## Uniformed Search

- ↳ no of steps unknown
- ↳ search without info ↪ start state → goal state like brute force
- ↳ time consuming
- ↳ more complexity
- ↳ always optimal solution

↳ BFS

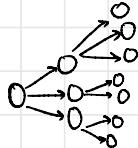
↳ Uniformed cost search

↳ DFS

↳ Depth limited search

↳ Iterative Deepening

↳ Bi-directional search



Blind Search

Heuristic Search

## Informed search

- ↳ agent has background info of the problem
- ↳ search with info
- ↳ quick solution
- ↳ less complexity
- ↳ NOT always optimal

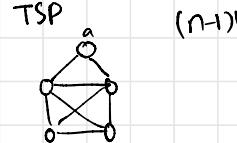
↳ Best First Search

↳ Heuristics

↳ Memory Boundred Search

↳ Iterative Improvement search

↳ A\*



→ QUEUE FIFO

## 1. Breadth-First-Search (BFS)

- ↳ Start from top, put in Queue
- ↳ Put adj vertices in Queue
- ↳ take out vertex when explored take it out of queue
- ↳ move to next vertex in Queue
- ↳ Repeat till all vertices explored

moves  
Uniformed  
Search

stack LIFO

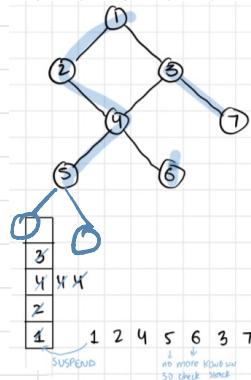
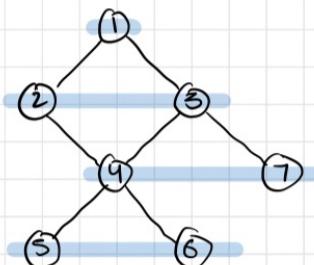
## 2. Depth First Search (DFS)

- ↳ Start from top, write it
- ↳ move to adj vertex
- ↳ Suspend prev vertex to stack
- ↳ Repeat till no adj vertex
- ↳ back track by checking stack
- ↳ Repeat till no vertices left

10/13



1 2 3 4 7 5 6



### CON

NOT Suitable for searching large graphs

### Algorithm (Informal)

1. Enqueue the root/initial node (**Queue Structure**).
2. Dequeue a node and examine it.
  1. If the element sought is found in this node, quit the search and return a result.
  2. Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
4. Repeat from Step 2.

### Depth-First Search

1. Start
2. Push Root Node to Stack
  - Mark Root Node as Visited
  - Print Root Node as Output
3. Check Top of the Stack If Stack is Empty, Go to Step 6
4. Else, Check Adjacent Top of the Stack
  - If Adjacent is not Visited
  - Push Node to Stack
  - Mark Node as Visited
  - Print Node as Output
  - Else Adjacent Visited
5. Go to Step 3
6. Stop

### 3. Uniformed-Cost-Search

↳ visits next node which has least total cost from root

↳ until a goal state is reached

#### Steps

1. Start from start node

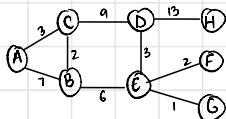
2. find min cost node from that node

3. Repeat 2 until

↳ Goal node found

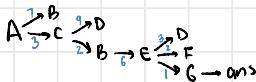
↳ Goal node not found, but reached last node

repeat



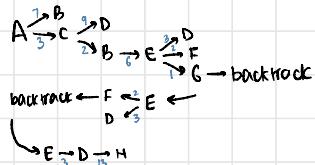
↳ checks all  
↳ gets smallest

#### Case 1: Goal node G



ANS:  $A \rightarrow C \rightarrow B \rightarrow E \rightarrow G$   
= 12

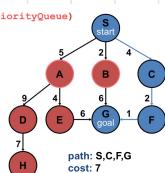
#### Case 2 : Goal node H



ANS:  $A \rightarrow C \rightarrow B \rightarrow E \rightarrow G \rightarrow E \rightarrow F \rightarrow E \rightarrow D \rightarrow H$   
= 33

generalSearch(problem, priorityQueue)  
# of nodes tested: 6, expanded: 5

Closed	Open Node
S	(S)
B	(B,2,C,4,A,5)
C	(C,4,A,5,G,8)
A	(A,5,F,6,G,9)
F	(F,6,G,8,E,9,D,14)
G	(G,8,E,9,D,14)



Similar to BFS  
↳ an evaluation of cost  
for each reachable node

### 4. DEPM-Limited Search

↳ solves DFS' infinite loop problem

- Sometimes a depth limit can be inferred or estimated from the problem description
- In other cases, a good depth limit is only known when the problem is solved
- must keep track of the depth

#### Termination conditions

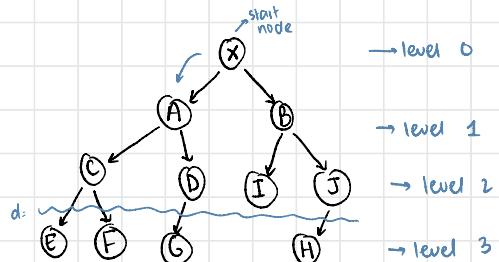
↳ failure value: There is no solution

↳ Cutoff failure: Terminates on reaching pre-determined depth

#### Steps

1. Same as DFS

2. don't move past pre-determined depth



Pre-determined depth = 2

#### Case 1: Goal node J

ANS:  $X \rightarrow A \rightarrow C \rightarrow D \rightarrow B \rightarrow I \rightarrow J$

#### Case 2: Goal node H

ANS: No solution as H is in  
level 3 and Pre-determined depth is 2

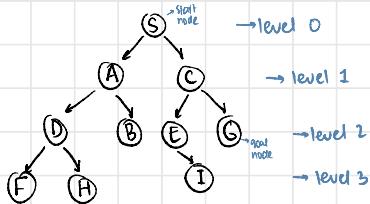
Similar to DFS  
but with a limit

## 5. Iterative Deepening DFS

- ↳ applies LIMITED-DEPTH with increasing depth limit
- ↳ combines adv of BFS and DFS

### Steps

1. DFS to depth 1
2. Increase depth, apply DFS
3. Repeat 2 till goal node found



Iteration	d	[S]
1	0	S
2	1	S → A → C
3	2	S → A → D → B → C → E → G → found

## 6. Bi-directional Search

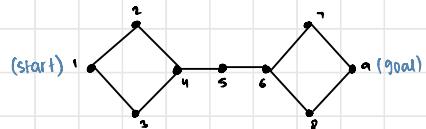
- ↳ both search forward from initial state and backwards from goals

Motivation:  $b^{dh} + b^{d/2} = 2b^{dh}$  is much less than  $b^d$

### Steps

1. start BFS/DFS from start and goal node
2. when both searches meet stop

↳ solution found



## PRO

- ↳ includes benefits of both DFS/BFS
- ↳ fast searching
- ↳ less memory req → uses only linear space

## CON

- ↳ Repeats Process ↳ as many states are expanded multiple times

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

# Evaluation of Search Strategies

↳ defined by picking the order of node expansion

↳ evaluated along the following dimensions

Properties	Breadth First	Depth First	Uniform Cost	Iterative Deepening	Depth Limited	Bi-directional
Complete	Yes	No ↑ finite for loops	Yes	Yes	No	Yes ↑ if BFS
Time Complexity	$O(b^{d+1})$	$O(b^m)$		$O(b^d)$	$O(b^L)$	$2b^{d/2}$
Space Complexity	$O(b^{d+1})$	$O(bm)$		$O(bd)$	$O(bL)$	
Optimality	Yes	No	Yes	Yes	No ↑ if $L_{cd}$	Yes ↑ if BFS

b: branching factor  
d: depth of goal  
m: max depth of static space

## When to use what

### BFS

↳ shallow solutions

↳ of little depth

### UCS

↳ actions have varying costs

↳ least cost solution is req

### Iterative Deepening DFS

↳ large search spaces

↳ unknown depth

### Depth First Search

↳ many solutions exist

↳ know/estimate the depth of solution

Breadth-first search (BFS) and depth-first search (DFS) are the foundation for all other search techniques.

## Properties of Uniform-cost Search (UCS)

Completeness: Yes (if  $b$  is finite, and step cost is positive)

Time Complexity: much larger than  $b^d$ , and just  $b^d$  if all steps have the same cost.

Space Complexity: as above

Optimality: Yes

Criterion	Uniform-Cost
Complete?	Yes
Time	$O(b(C/d))$
Space	$O(b(C/d))$
Optimal?	Yes

b Branching Factor  
d Depth of the goal/tree

Requires that the goal test being applied when a node is removed from the nodes list rather than when the node is first generated while its parent node is expanded.

## Breadth First Search

- ↳ special case of uniformed cost search
- ↳ when all edge costs are positive and identical
- ↳ expands the shallowest node ↳ only optimal if all costs equal
- ↳ only optimal if all costs equal

VS

## Uniform Cost Search

- ↳ considers overall path cost
- ↳ optimal for any cost function
- ↳ non-zero, positive
- ↳ gets stuck down in trees
- ↳ low path cost, but no goal node

both are complete for non-extreme problems

- ↳ finite no. of branches
- ↳ strictly positive search function

## Depth First

VS

## Breadth First

- ↳ goes off into one branch until it reaches leaf node
- ↳ not good if goal is on another branch
- ↳ neither complete or optimal
- ↳ uses lesser space than breadth first

- ↳ checks all alternatives
- ↳ complete and optimal
- ↳ memory intensive

For a very large tree

- ↳ may take excessively long time to find even a very nearby goal

- ↳ memory req. maybe excessive

### Iterative Deepening Search

When searching a binary tree to depth 7:

- DFS requires searching 255 nodes
- Iterative deepening requires searching 502 nodes
- Iterative deepening takes only about twice as long

When searching a tree with branching factor of 4 (each node may have four children):

- DFS requires searching 21845 nodes
- Iterative deepening requires searching 29124 nodes
- Iterative deepening takes about 4/3 = 1.33 times as long

The higher the branching factor, the lower the relative cost of iterative deepening depth first search

↳ new branch factor  
↳ worse IDS

### Iterative Deepening Search

The nodes in the bottom level (level d) are generated once, those on the next bottom level are generated twice, and so on

$$N_{IDS} = (d)b + (d-1)b^2 + \dots + (1)b^d$$

Time complexity =  $b^d$

Compared with BFS:

$$N_{BFS} = b + b^2 + \dots + b^d + (b^{d+1} - b)$$

- Suppose  $b = 10$ ,  $d = 5$ ,

$$\begin{aligned} N_{IDS} &= 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456 \\ N_{BFS} &= 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111 \end{aligned}$$

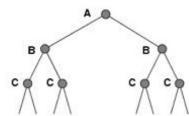
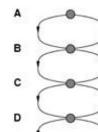
- IDS behaves better in case the search space is large and the depth of goal is unknown.

only informed search that worries about cost

### Improving Search Methods

Make algorithms more efficient

- avoiding repeated states



Use additional knowledge about the problem

- properties ("shape") of the search space
  - more interesting areas are investigated first
- pruning of irrelevant areas
  - areas that are guaranteed not to contain a solution can be discarded

To discover

## Heuristics → aka Rule of Thumb

↳ A technique designed to solve a problem quickly

↳ Comes from experience

↳ helps you think through things ↳ process of elimination  
process of trial and error

↳ Will give quick solution

But not necessarily optimal

## Heuristic function

↳ Are Problem specific

↳ Mostly employs straight-line distance  
for route finding and similar problems

↳ Often better than DFS

although worse time complexities are  
equally bad or worse

Properties	Greedy Best First search	A*	SMA*	RBFS
Completeness		Yes	Yes	
Time Complexity		Exponential		
Space Complexity		Keeps all nodes in memory		
Optimality		Yes	Yes	Yes

### Greedy best-first search

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost f(n) = h(n) */
    frontier = Heap.new(initialState)
    explored = Set.new()
    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)
        if goalTest(state):
            return SUCCESS(state)
        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)
    return FAILURE
```

### A\* Algorithm

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost: f(n) = g(n) + h(n) */
    frontier = Heap.new(initialState)
    explored = Set.new()
    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)
        if goalTest(state):
            return SUCCESS(state)
        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)
    return FAILURE
```

### SMA\* pseudocode

```
function SMA*(problem) returns a solution sequence
    inputs: problem, a problem
    static: Queue, a queue of nodes ordered by f-cost
    Queue ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
    loop do
        if Queue is empty then return failure
        if Queue desire high-cost mode in Queue
        if GOAL-TEST(node) then return success
        i ← NEXT-SUCCESSOR(i)
        if it is not a goal and at maximum depth then
            f(i) ← ∞
            f(i) ← MAX(f(i),g(i)+h(i))
            if all of i's successors have been generated
            then delete i from the list of its ancestors if necessary
            if SUCCESSOR(i) is in memory then remove i from Queue
            if memory is full then
                delete shallowest, highest-cost node in Queue
                remove it from its parent's successor list
                insert its parent on Queue if necessary
                insert i in Queue
        end
```

## Greedy Best First Search

$O(k^m)$

- ↳ expands the node that appears to be closest to goal heuristically
- ↳ estimate of cost from  $n$  to goal using heuristic function  
 $h_{SLD}(n)$  = straight line distance from  $n$  to Bucharest

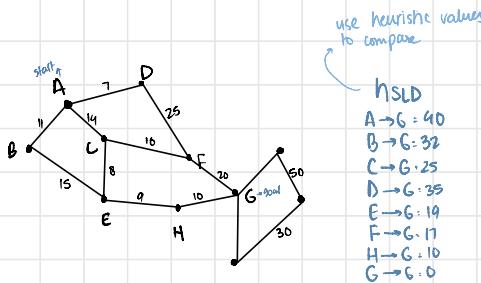
### Steps

1. Start from start node

2. find min  $h_{SLD}$  node from that node

3. Repeat 2 until

↳ Goal node found



$$A \xrightarrow{B=32} C \xrightarrow{25} D \xrightarrow{35}$$

$$A, C, B, D$$

$$C \xrightarrow{E=10} F \xrightarrow{11}$$

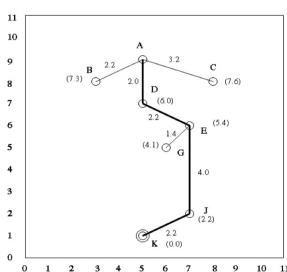
$$A, C, B, D, F, E$$

$F \rightarrow G \rightarrow 0 \rightarrow$  reached goal state

$A \rightarrow C \rightarrow F \rightarrow G$

### Solution to Greedy Best-First Exercise

Node	Coordinates	$h(n)$
A	(5.9)	8.0
B	(3.8)	7.3
C	(8.8)	7.6
D	(5.7)	6.0
E	(7.6)	5.4
F	(4.5)	4.1
G	(6.5)	4.1
H	(3.3)	2.8
I	(5.3)	2.0
J	(7.2)	2.2
K	(5.1)	0.0



## A\* Search

↳ combines greedy and UCS

$O(b^d)$

- ↳ avoids expanding paths that are already expensive
- ↳  $f(n) = g(n) + h(n)$
- ↳ Heuristics must be admissible

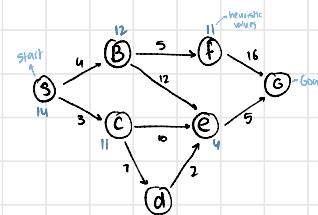
### Steps

1. Start from start node

2. Use  $f(n)$  function to find min heuristic cost node

3. Repeat 2 until

↳ Goal node found



$$\hookrightarrow f(S) = 0 + 14 = 14$$

$$\hookrightarrow S \rightarrow B : 4 + 12 = 16 \quad S \rightarrow C : 3 + 11 = 14 \quad \hookrightarrow S_B \rightarrow f: 4+5+11=20 \quad S_B \rightarrow e: 4+12+4=20$$

$$\hookrightarrow S_C \rightarrow d : 3+7+6=16 \quad S_C \rightarrow e : 3+10+4=17 \quad \hookrightarrow \text{smallest so use this}$$

$$\hookrightarrow S_C d \rightarrow e : 3+7+2+4=16$$

$$\hookrightarrow S_C d e \rightarrow G : 3+7+2+5 = 17$$

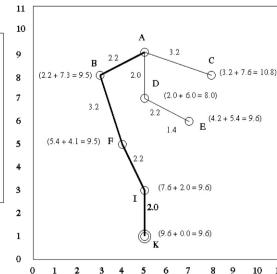
### CONS

↳ complexity problems

↳ huge memory req

### Solution to A\* Exercise

Node	Coordinates	$h(n)$
A	(5.9)	8.0
B	(3.8)	7.3
C	(8.8)	7.6
D	(5.7)	6.0
E	(7.6)	5.4
F	(4.5)	4.1
G	(6.5)	4.1
H	(3.3)	2.8
I	(5.3)	2.0
J	(7.2)	2.2
K	(5.1)	0.0



## Admissible Heuristics

$h(n)$  is admissible

if every node  $n$ ,  $h(n) \leq h^*(n)$  → underestimation

↪ it is optimistic → never overestimates the cost to reach the goal

Theorem-1: If  $h(n)$  is admissible, A\* using TREE-SEARCH is optimal.

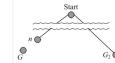
true cost to reach the goal state from  $n$

→ underestimation

## Optimality of A\* (Proof Admissible)

Recall that  $f(n) = g(n) + h(n)$

Now, suppose some suboptimal goal  $G_2$  has been generated and is in the fringe. Let  $n$  be an unexpanded node in the fringe such that  $n$  is on a shortest path to an optimal goal  $G$ .



We want to prove:

$f(n) < f(G)$

(then A\* will prefer  $n$  over  $G$ )

$f(G_2) = g(G_2)$  since  $h(G_2) = 0$

$g(G_2) > g(G)$  since  $G_2$  is suboptimal

$f(G) = g(G)$  since  $h(G) = 0$

Then  $f(G_2) > f(G)$  from above

$h(n) \leq h^*(n)$  since  $h$  is admissible

$g(n) + h(n) \leq g(n) + h^*(n)$

Then  $f(n) \leq f(G)$

Thus, A\* will never select  $G_2$  for expansion

In other words:

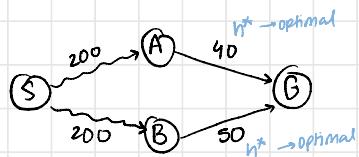
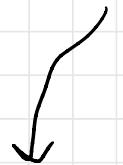
$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$ ,

since  $G_2$  is a goal on a non-optimal path ( $C^*$  is the optimal cost)

$f(n) = g(n) + h(n) \leq C^*$ , since  $h$  is admissible

$f(n) \leq C^* < f(G_2)$ , so  $G_2$  will never be expanded

↪ A\* will not expand goals on sub-optimal paths

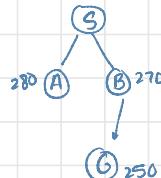


$h(n) \geq h^*(n) \rightarrow$  overestimation

$$\begin{array}{l} \text{Q)} h(A) = 80 \\ h(B) = 70 \end{array} \quad ] > h^*$$

$$\begin{array}{l} f(A) = 200 + 80 = 280 \\ f(B) = 200 + 70 = \underline{\underline{270}} \end{array}$$

$$\begin{aligned} f(G) &= g(G) + h(G) \\ &= \frac{g(G) + h^*(G)}{h} \\ &= \frac{700 + 50 + 0}{0} \\ &= 250 \end{aligned}$$



$h(n) \leq h^*(n) \rightarrow$  underestimation

$$\begin{array}{l} \text{Q)} h(A) = 30 \\ h(B) = 20 \end{array} \quad ] < h^*$$

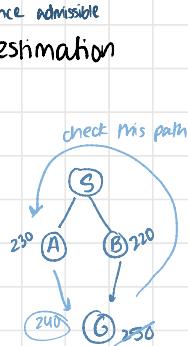
$$\begin{array}{l} f(A) = 200 + 30 = 230 \\ f(B) = 200 + 20 = 220 \end{array}$$

$$\begin{aligned} f(G) &= g(G) + h(G) \\ &= \frac{g(G) + h^*(G)}{h} \\ &= \frac{700 + 50 + 0}{0} = 250 \end{aligned}$$

$$\begin{aligned} f(G) &= g(G) + h(G) \\ &= \frac{g(G) + h^*(G)}{h} \\ &= \frac{200 + 40 + 0}{0} = 240 \end{aligned}$$

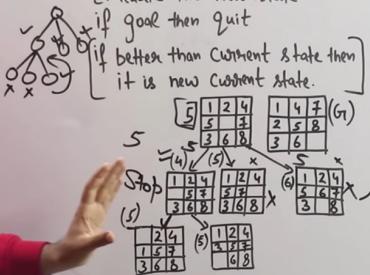
$$\begin{aligned} g(n) + h(n) &< g(n) + h^*(n) \\ 230 &< 250 \end{aligned}$$

so won't expand on B route



'Simple Hill Climbing Algorithm'(Local search Algo, greedy approach, No backtracks)

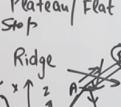
- Evaluate the initial state
  - Loop Until a solution is found or there are no operators left
    - Select and Apply a new operator
    - Evaluate the new state



Problems in Hill climbing

- i) Local Maximum
 

→ There may be a better move if you didn't take one step first. But it's not apparent so descent algorithm probably always give global answer.
- ii) Plateau / Flat Maximum
 

→ If all possibilities have same heuristic no move if steps
- iii) Ridge
 

**PROS**

- ↳ less SPACE complexity as it only stores 1 more

L only keeps a singular current state

↳ moves in the direction of inc elevation

Lterminates when peak value reached and no neighbour has a higher value

Simulated Annealing:- Checks all neighbors.

Simulated Annealing (SA) allows downward steps.

→ Annealing is a process in metallurgy where metals are slowly cooled to make them reach a state of low energy where they are very strong.

Advantages	Disadvantages
<ul style="list-style-type: none"> <li>→ Easy to code for</li> <li>Complex problems also</li> <li>→ Gives Good Sol<sup>n</sup>.</li> <li>→ Statistically guarantees finding optimal sol<sup>n</sup>.</li> </ul>	<ul style="list-style-type: none"> <li>→ Slow Process.</li> <li>→ Can't tell whether an optimal sol<sup>n</sup> is found.</li> <li>↳ Some other method is also reqd.</li> </ul>



## Simulated Annealing | Hill Climbing

- i) annealing schedule is maintained.
  - ii) Moves to worst States may be accepted.
  - iii) Best state found so far

↳ Space time greater than hill climbing

↳ SLOW

## ↳ backtracking

↳ Optimal Sol

↳ space time complexity less

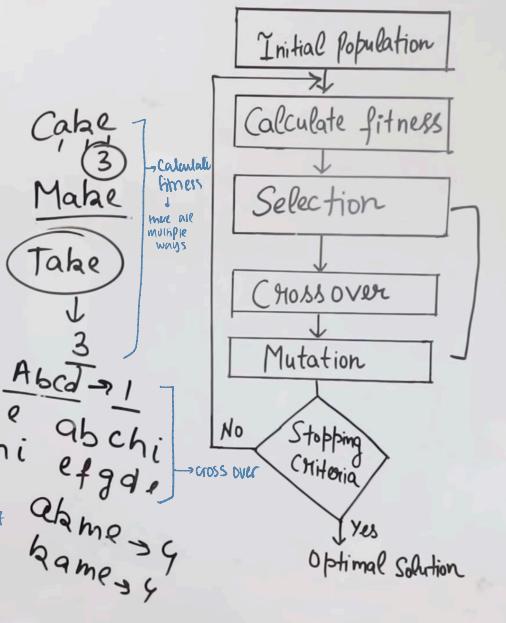
↳ fast

End backtracking

↳ may not give optimal sol

## Genetic Algorithm (John Holland)

- Abstraction of Real Biological Evolution.
- Solve Complex Problems (like NP Hard)
- focus on Optimization
- Population of possible solutions for a given problem
- From a group of individuals, the best will survive
- Phenotype  $\xrightarrow{\text{Encode}}$  Genotype  $\xleftarrow{\text{representation}}$



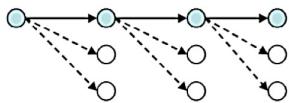
## Genetic Algorithm

Introduced in the 1970s by John Holland at University of Michigan

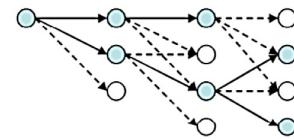
- ▶ begin with  $k$  randomly generated states (population)
- ▶ each state (individual) is a string over some alphabet (chromosome)
- ▶ fitness function (bigger number is better)
- ▶ crossover
- ▶ mutate (evolve?)

## Local Beam Search

- Start with  $k$  randomly generated states ( $\beta = 2$  or  $3$ )
- At each iteration, all the successors of all  $k$  states are generated
- If any one is a goal state, stop; else select the  $k$  best successors from the complete list and successors from the complete list and repeat



Greedy Local



Local Beam

# Memory Bounded Heuristic Search

↳ To solve memory problem of A\*

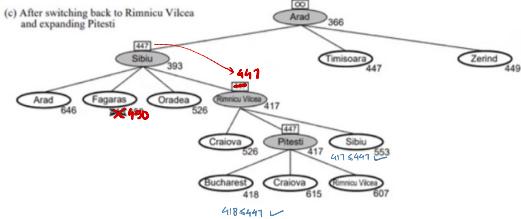
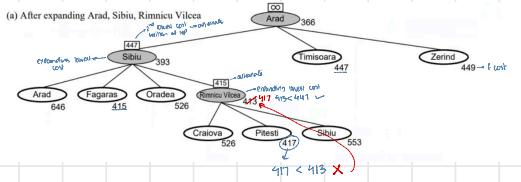
↳ Two types of Memory Bounded Heuristic search

↳ Recursive BFS

↳ SMA\*

## Recursive Best First Search (RBFS)

↳ BFS using only linear space

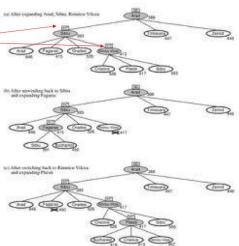


best alternative  
over fringe nodes,  
which are not children:  
*do I want to back up?*

RBFS changes its mind very often  
in practice.

This is because the  $f = g + h$   
become more accurate (less  
optimistic) as we approach the  
goal. Hence, higher level nodes  
have smaller  $f$ -values and will be  
explored first.

Problem? If we have more  
memory we cannot make use of it.  
Any idea to improve this?



Ans: Arad → Sibiu → Rim → Pitesti → Bucharest

# Simple Memory-Bounded A\* (SMA\*)

↳ finds optimal reachable solution given the memory constraint

→ but time can still be exponential

If there is a tie → eval f-values

We first delete the oldest nodes first

How it works:

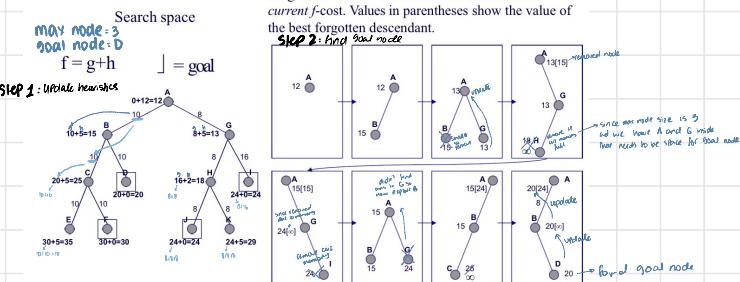
- Like A\*, it expands the best leaf until memory is full.
- Drops the worst leaf node - the one with the highest f-value.
- SMA\* then backs up the value of the forgotten node to its parent.

• This is like A\*, but when memory is full we delete the worst node (largest f-value).

• Like RBFS, we remember the best descendent in the branch we delete.

## Simple Memory-bounded A\* (SMA\*)

(Example with 3-node memory)



∞ is given to nodes that the path up to it uses all available memory.

Can tell when best solution found within memory constraint is optimal or not.

35

The Algorithm proceeds as follow

Let MaxNodes = 3

• Initially B and G are added to open list, then hit max.

• B is larger f value, so discard but save f(B)=15 at parent A

Add H, but f(H)=18. This is not a goal and we can never go deeper, so set f(H)=infinity and save at G.

• Generate next child I with f(I)=24, bigger other child of A. Now we have seen all children of G, so reset f(G) to 24.

• Regenerate B and child C. This is not a goal so f(C) is reset to infinity.

• Generate second child D with f(D)=20, backing up value to ancestors.

• D is a goal node, so search terminates.

## PROS

↳ uses bounded memory

↳ avoids repeated states if memory allows it

↳ optimal and complete if memory is high enough to store shallowest solution

↳ will use all memory

↳ higher the memory, higher the speed

## Relaxed Problems

↳ problems with fewer restrictions on actions

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then  $h_1(n)$  gives the shortest solution.

If the rules are relaxed so that a tile can move to any near square, then  $h_2(n)$  gives the shortest solution.

ishma hafeez  
notes

reprint  
tree