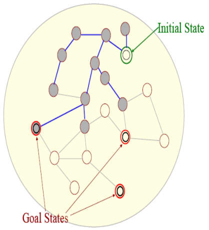
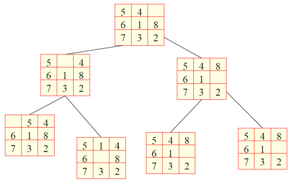
Artificial Intelligence

Module 2: Automated Problem Solving

PART 2.3: Problem Representation in AI



Dr. Chandra Prakash
Assistant Professor
Department of Computer Science and Engineering



(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Prof. Pallab Dasgupta, Prof. Partha Pratim Chakrabarti, Saikishor Jangiti

Module 2: Automated Problem Solving

- PART 2.1: Intelligent Agent & Environment
- PART 2.2: Complex Problems and AI
- PART 2.3: Problem Representation in AI
  - Problem Solving Agents
    - Intelligent Agent
  - Problem Solving Methods
    - Defining the problem as a State Space Search
    - Problem Decomposition
  - Problem Formulation by AI Search Methods
    - Modeling formulation of a problem to AI

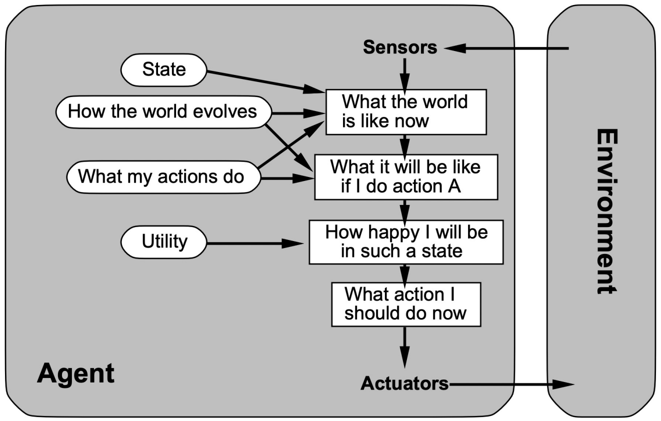
Type of Agents

Simple reflex agents

Model-based reflex agents

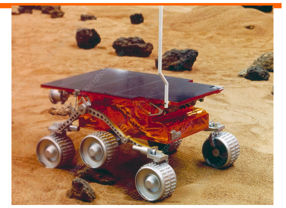
Goal-based agents

Utility-based agents

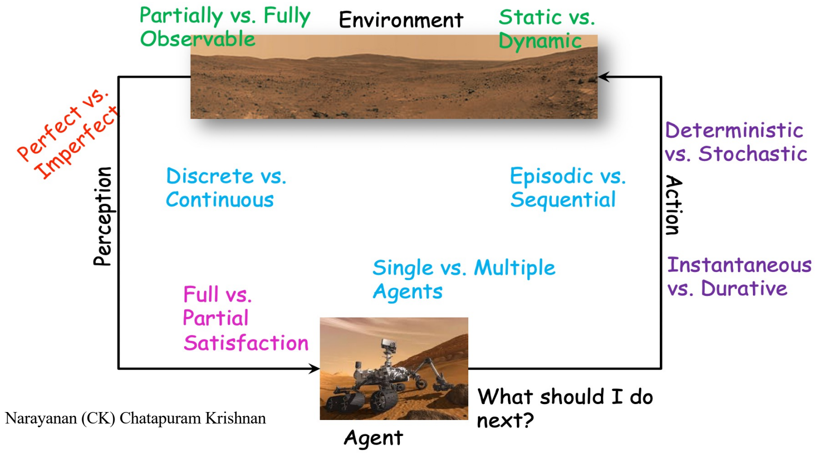


Design an Agent for The Mars Rover Sojourner

- Step 1: Aim of the agent ????
- The 25-lb, 6-wheeled robotic explorer needs to
  - travel on sandy, rocky terrain
  - perceive its surroundings
  - drive autonomously for short distances
  - communicate and transmit data
  - be remotely controlled
  - be transported easily
  - withstand extreme temperatures on Mars
  - carry necessary equipments
  - respond to events as they occur
  - decide what to do next



Step 2: Task Environment

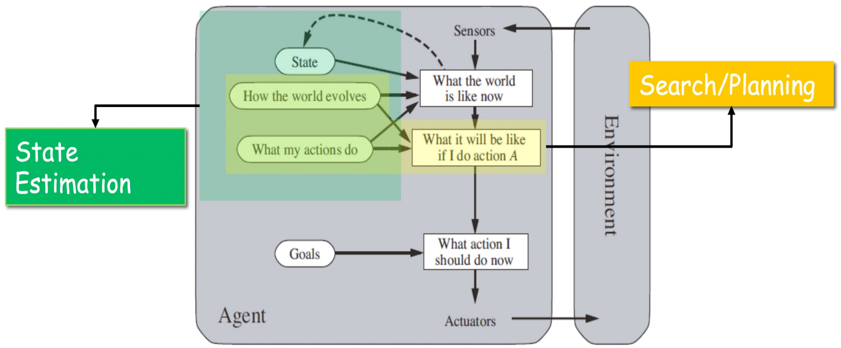


Credit : Narayanan (CK) Chatapuram Krishnan

Problem Solving Agents

- Reflex agents cannot work well in those environments
  - state/action mapping too large
  - take too long to learn
- Problem-solving agent
  - is one kind of goal-based agent
  - decides what to do by finding sequences of actions that lead to desirable states
- Formulation
  - Goal formulation (final state)
  - Problem formulation (decide what actions and states to consider)
- Search (look for solution i.e. action sequence)
- Execution (follow states in solution)
- Assume the environment is static, observable, discrete, deterministic

Intelligent Agent –Goal based

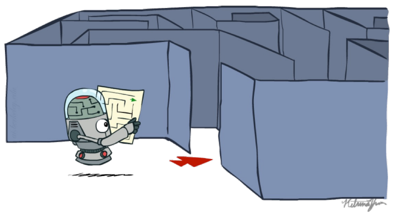


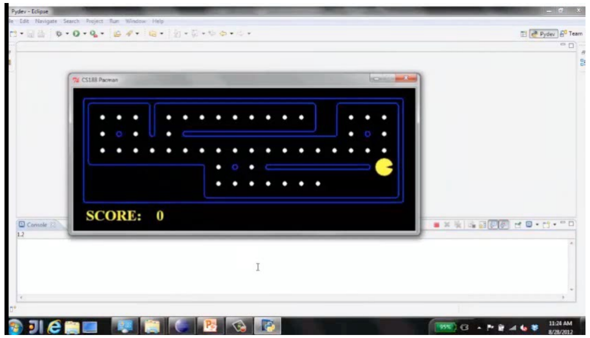
Intelligent agents may also learn or use knowledge to achieve their goals

We want

- Smart human :
  - not only good in 1 problem but in other areas
- Our aim is to solve all type of problems in the world
- Automated Problem solving approach
- Generalized Techniques for
  - Solving Large Classes of Complex Problems
  - if not clear how to start/ proceed :
    - Ask what is input and what is output

Problem solving agent : Pac-Man





L2D3, replanning.
Figure out how to get to next dot.
Clears board; non-optimal. Fast.

Search Problems

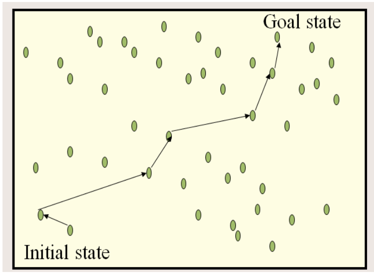
How to represent the environment/ world/ percepts for a problem ??

Methods of Problem Representation in AI

- Before a solution can be found, the prime condition is that the problem must be very precisely defined.
- The most common methods of problem representation in AI are:
  1. **State Space representation**
     - Includes the initial state S and all other states reachable from S by a sequence of actions
  2. **Problem Reduction**
     - Whether the problem can be decomposed into smaller problems?
     - Using the technique of problem decomposition, we can solve very large problems easily.
       - Example for decomposable problems
         - » $\int(x^2 +3x + \mathrm{Sin}2x.\mathrm{Cos}2x)dx$

Defining the problem: State Space representation

- State Space
  - provide all possible state, operations and the goals.
  - all information about the environment
  - All information necessary to make a decision for the task at hand.
  - If the entire state-space representation for a problem is given, it is possible to trace the path from the initial state to the goal state and identify the sequence of operations necessary for doing it.
  - **Limitation :**
    - not possible to visualize all states for a given problem.



Agent Classification in Terms of State Representations

Type	State representation	Focus
Atomic	States are indivisible;	No internal structure Search on atomic states;
Propositional (aka Factored)	States are made of state variables that take values (Propositional or Multi-valued or Continuous)	Search+inference in logical (prop logic) and probabilistic (bayes nets) representations
Relational	States describe the objects in the world and their interrelations	Search+Inference in predicate logic (or relational prob. Models)
First-order	functions over objects	Search+Inference in first order logic (or first order probabilistic models)

Spectrum of State Representations

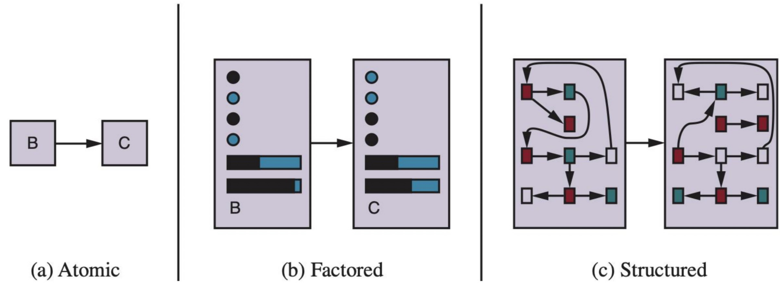
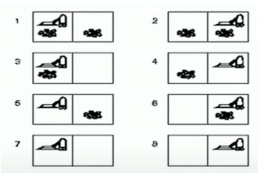


Illustration with Vacuum World

Atomic:
S1, S2…. S8, Each state is seen as an indivisible snapshot
All Actions are SXS matrices.

If you add a second roomba the state space doubles
If you want to consider noisiness of the rooms, the representation Quadruples.

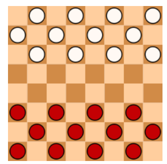


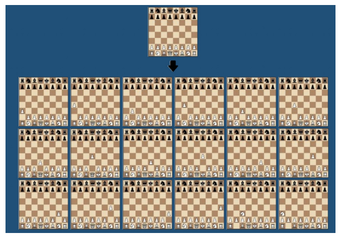
Relational:
World made of objects: Roomba; L-room, R-room
Relations: In (<robot>, <room>); dirty(<room>)
If you add a second roomba, or more rooms, only the objects increase.
If you want to consider noisiness, you just need to add one other relation

Propositional/Factored:
States made up of 3 state variables
Dirt-in-left-room T/F
Dirt-in-right-room T/F
Roomba-in-room L/R
Each state is an assignment of Values to state variables
23 Different states
Actions can just mention the variables they affect
Note that the representation is compact (logarithmic in the size of the state space)
If you add a second roomba, the representation increases by just one more state variable.
If you want to consider "noisiness" of rooms, we need two variables, one for each room

Representing States

- **State space:**
  - A state space can be organized as a graph:
    - nodes: states in the space
    - arcs: actions/operations
- The **size of a problem** is usually described in terms of the number of states (or the size of the state space) that are possible.
  - Tic-Tac-Toe has about 3^9 states.
  - Checkers has about 10^40 states.
  - Chess has about 10^120 states in a typical game.
    - Shannon number



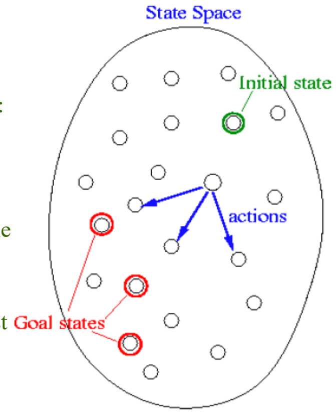


Atomic Agent

- Input
  - Set of states
  - Operators [and cost]
  - Start state
  - Goal state [Test]
    - **This is a hard part that is rarely tackled in AI, usually assuming that the system designer or user will specify the goal to be achieved.**
- Output
  - Path : start => a state satisfying goal test
  - May require shortest path

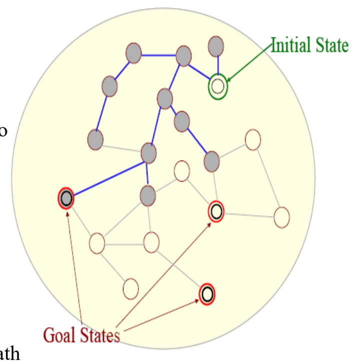
Representing States

- At any moment, the relevant world is represented as a state
  - **Initial (start) state: S**
  - **Possible action (or an operation)**
    - changes the current state to another state (if it is applied):
  - **State transition / Transition Model**
    - An action can be taken (applicable) only if the its precondition is met by the current state
    - For a given state, there might be more than one applicable actions
  - **Goal state/ Goal Test :**
    - a state satisfies the goal description or passes the goal test
  - **Dead-end state:**
    - a non-goal state to which no action is applicable



Formalizing Search in a State Space

- A state space is a graph, (V, E) where
  - V is set of nodes and E is set of arcs
- **Node:** corresponds to a state
- **Arc:** corresponds to an applicable action/operation.
- **node generation:** making explicit a node by applying an action to another node which has been made explicit
- **node expansion:** generate **all** children of an explicit node by applying **all** applicable operations to that node
- One or more nodes are designated as **start nodes**
- A **goal test** predicate is applied to a node to determine if its associated state is a goal state
- A **solution** is a sequence of operations that is associated with a path in a state space from a start node to a goal node
- The **cost of a solution** is the sum of the arc costs on the solution path



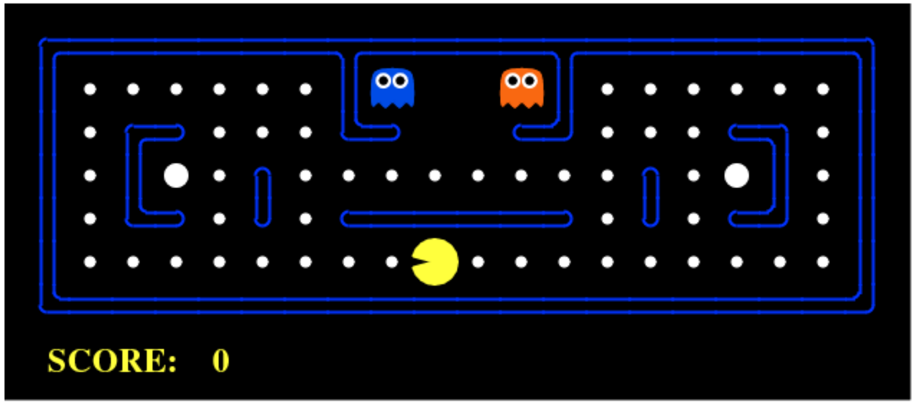
Problem Solving in AI

- A process or procedure used to find out the solution to a specific problem.
- To build a system to solve a particular problem we need to do 4 things.
  1. Define the problem precisely
     - This definition must include precise specifications of what the initial situations will be as well as what the final situations constitute acceptable solution to the problem.
  2. Analyze the problem
     - A few important features of the problem can help in the selection of various possible techniques for solving the problem.
  3. Isolate and represent the task knowledge that is necessary to solve the problem
  4. Choose the best problem-solving techniques and apply it to the particular problem.

Problem Solving Stages

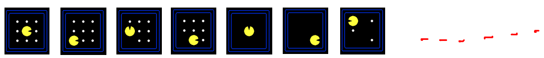
1. Assumptions
2. Solution steps
3. State Space Representations
   - Initial state
   - Final state
   - Operators that can be applied
   - Abbreviations
   - State space representation of the given problem
   - Operators and Conditions.

Example : Pac-Man

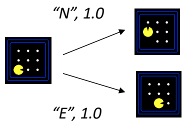


Pac-Man Example : Search Problems

- A search problem consists of:
  - A state space



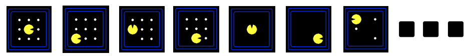
  - A successor function
    - (with actions, costs)



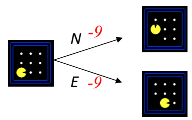
  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Search Problems

- A search problem consists of:
  - A state space $\mathcal{S}$



  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model $Result(s,a)$
  - A goal test $G(s)$
    - s has no dots left
  - Action cost $c(s,a,s')$
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost



- A solution is an action sequence that reaches a goal state
- An optimal solution has least cost among all solutions

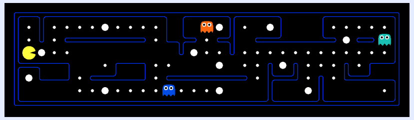
Pacman *agent program* in Python

```
class GoWestAgent(Agent):

  def getAction(self, percept):
    if Directions.WEST in percept.getLegalPacmanActions():
      return Directions.WEST
    else:
      return Directions.STOP
```

Pac-Man Example: What's in a State Space?

The world state includes every last detail of the environment

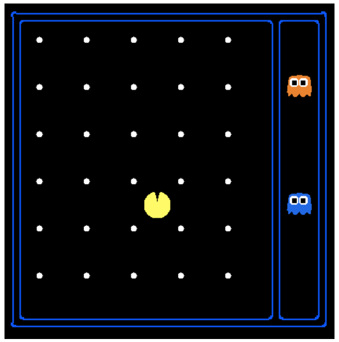


A search state keeps only the details needed for planning (abstraction)

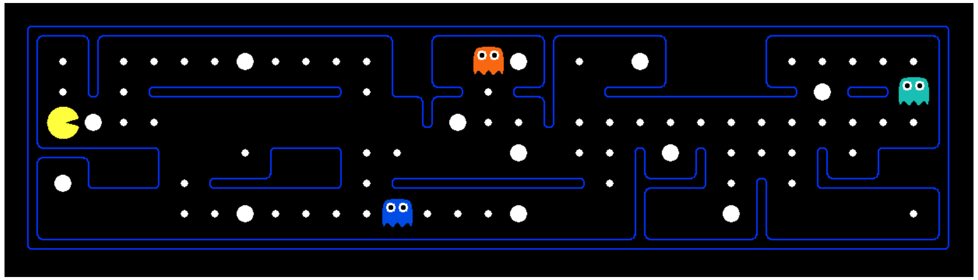
- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

State Space Sizes?

- World state: 12x10 grid
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120 \text{x} (2^{30}) \text{x} (12^2) \text{x} 4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120 \text{x} (2^{30})$

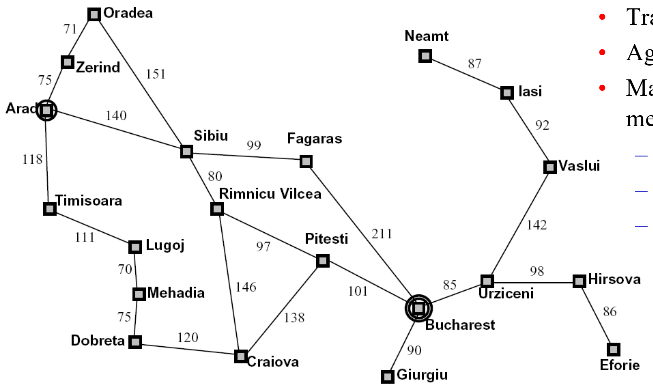


Quiz: Safe Passage



- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)

Example : Romania



- Traveling in Romania
- Agent is holidaying in Arad.
- Many factors in its performance measure
  - Tour the famous places
  - Try local cuisine
  - Souvenir Shopping

Problem Solving – Atomic Agents

- Atomic Agents
  - States are indivisible
  - Searching through the states to reach the goal.

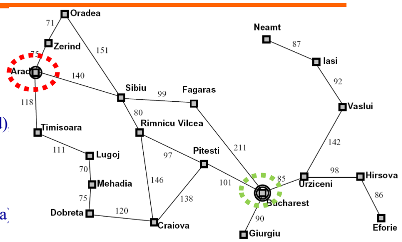
function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **returns** an action
  static: *seq*, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null
    problem, a problem formulation

  state ← UPDATE-STATE(*state*, *percept*)
  if *seq* is empty **then**
    goal ← FORMULATE-GOAL(*state*)
    problem ← FORMULATE-PROBLEM(*state*, *goal*)
    seq ← SEARCH(*problem*)
  action ← RECOMMENDATION(*seq*, *state*)
  seq ← REMAINDER(*seq*, *state*)
  return *action*

Problem Solving Agents – Formulate

A problem can be formulated using five components

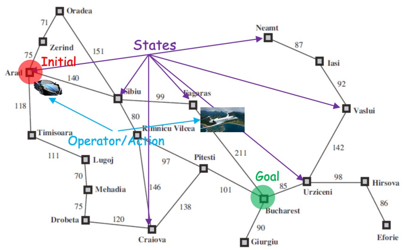
1. **Initial State** :
   - The agent's starting state. E.g., In(Arad)
2. **Possible Actions** :
   - Set of applicable actions in a given state. E.g., from the state s = In(Arad) applicable actions
     - ACTIONS(s) → {Go(Sibiu), Go(Timisoara), Go(Zerind)}
3. **Transition Model** :
   - The resulting state of an action in a given state, modeled as RESULT(s, a
     - RESULT(In(Arad), Go(Sibiu)) = In(Sibiu)
4. **Goal Test** :
   - Determines whether a given state is a goal state.
     - IsGoal(In(Bucharest)) = Yes
5. **Path Cost** :
   - A function that assigns a numeric cost to each path. A path is a series of actions. Each action is given a cost depending on the problem.
     - cost(In(Arad), go(Sibiu)) = 140 kms



- State space: Cities
- Successor function: Roads: Go to adjacent city with cost = distance
- Start state: Arad
- Goal test: Is state == Bucharest?
- Path Cost: ?
- Solution: Finding the sequence of actions - Search

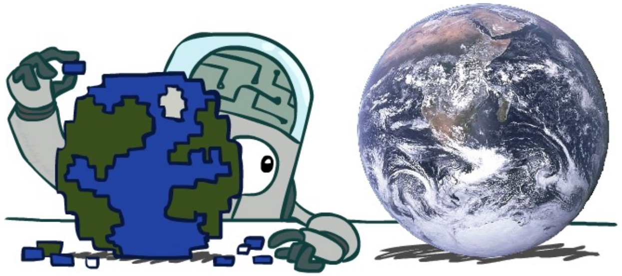
Problem Solving Agents :Traveling in Romania

- In the above definition
  - Many possible states – In(Arad), the agent has many possibilities in real world, e.g., can be in the middle of Arad, near an airport, shopping center, hotel, etc.
  - Many possible actions – Agent can take several actions while driving from Arad to Sibiu. It can halt at a place, take a detour, turn the radio on.
  - However, they are irrelevant for finding the solution of path to Bucharest
  - Hence, can be abstracted
- Abstraction – Process of removing detail from a representation
- **Problem Formulation:**
  - Process of deciding what actions and states to consider, given a goal



- **Assumptions about the Environment**
  - Observable: Agent always knows the current state
  - Discrete: At any given state, only finitely many actions to choose from
  - Known: Knows which states are reached by each action
  - Deterministic: Each action would always have the same resulting state

Search Problems Are Models

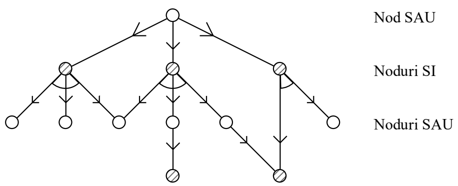


Problem Representation in AI

- Before a solution can be found, the prime condition is that the problem must be very precisely defined. The most common methods of problem representation in AI are:
  1. State Space representation
     - Includes the initial state S and all other states that are reachable from S by a sequence of actions
  2. Problem Reduction
     - Whether the problem can be decomposed into smaller problems?
     - Using the technique of problem decomposition, we can often solve very large problems easily.
       - Example for decomposable problems
       - $\int(x2 +3x + Sin2x.Cos2x)dx$

AND/OR graph representation

- Problem decomposition into sub-problems
  - AND/OR graph
  - Solved node
  - Unsolvable node
  - Problem solution

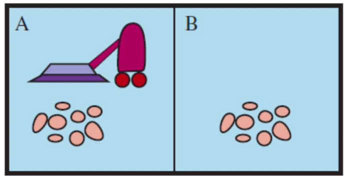


Problem Formulation - Examples

- Toy Problems :
  - Intended to illustrate or exercise various problem solving methods. Useful in research for comparing the performance of algorithms
    - Vacuum World, 8-Puzzle, 8-Queens Problem,
    - Cryptarithmetic, Missionaries and Cannibals
- Real-World Problems :
  - Problems for which solutions can be impacting people's daily lives.
    - Route finding, Touring problems
    - Traveling salesman problem,
    - VLSI layout, Robot navigation,
    - Assembly sequencing,
    - Protein Design, Internet Searching

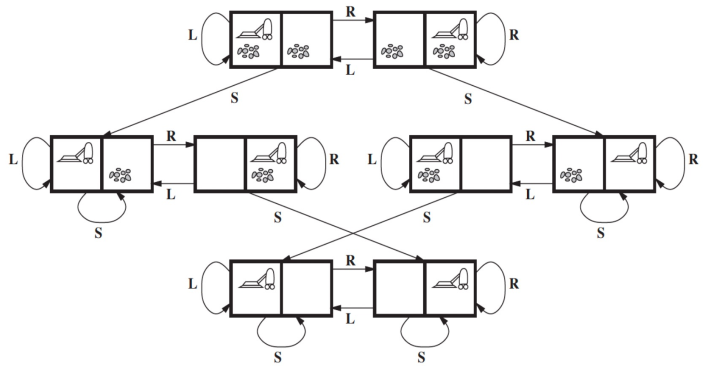
Modeling Formulate – Toy Problem Example

- **Possible States** – There are two locations and two possibilities of dirt. Total of 8 possible states:
- $2x2^2=8$

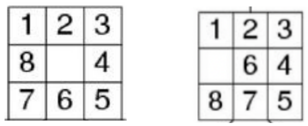


1. State
   - Robot and dirt locations
   - Initial State
     - Any state with position of agent and dirt mentioned
2. Actions
   - Three possible actions, Left, Right, Suck
3. Transition Model
   - Left would move the agent to left location, except when the agent is in leftmost location, there would be no effect.
   - Similarly for Right action. Suck would remove the dirt, if any.
4. Goal Test
   - Checks whether all locations are clean or no Dirt
5. Path cost
   - Each step costs 1, so the path cost is total number of steps

Formalizing problem in a State Space: Toy Example



Problem Formulation Example: 8-puzzle

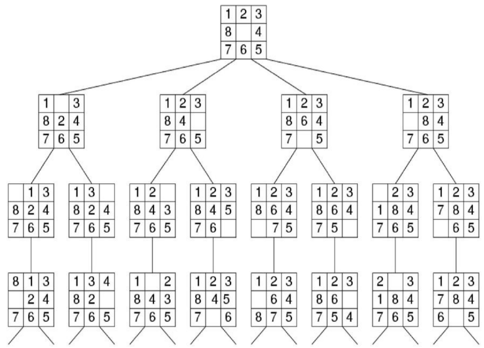


Initial State Goal State

a) Initial state
b) Final/ Goal State
c) Operator
   i. *Move(left)*
   ii. *Move(right)*
   iii. *Move(Up)*
   iv. *Move(Down)*

   – *Belongs to sliding-block puzzles*
   – *NP - Complete problem.*

Problem Formulation Example: 8-puzzle



- Possible states = 9! / 2

1. Initial State :
   – Any permutation of 1-8 numbers with a blank
2. Actions :
   – Movement of blank space either by Left, Right, Up or Down
3. Transition Model :
   – The resulting state after moving the blank space will replace the digit
4. Goal Test :
   – Check whether the state matches the goal configuration
5. Path cost :
   – Each step costs 1, so the path cost is total number of steps

Problem Formulation : Water Jug Problem

Define the Problem Accurately.

– You are given two jugs, a 4 gallon one and a 3 gallon one.
– Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water.
– Discuss how exactly 2 gallon of water can be filled into 4 gallon of jug by employing the state space representations.

Goal: 2 Gallons Water in the 4 Gallon Jug

Example : Step1-Assumptions

In order to solve the given problem we can make the following assumptions without affecting the problem.

1. We can fill the jugs with the help of a pump.
2. We can pour water from any jug to the ground.
3. We can transfer water from one jug to the other.
4. No external measuring devices are available.

Example : Step2- Solution Steps

State Space Representation

1. Initial and final states
2. Operators
3. Abbrivations
4. State space representation of the given problem

State No.	Operator	State (G4,G3)

5. Operators and Conditions
6. Production rules for Water Jug Problem

State Space Representation

1. Initial & Final States

- The state space for this problem can be described as the set of ordered pairs of integers (x,y)
  - such that x = 0, 1,2, 3 or 4 and y = 0,1,2 or 3;
  - – x represents the number of gallons of water in the 4-gallon jug and
  - – y represents the quantity of water in 3-gallon jug
- The start state is (0,0)
- The goal state is (2,n) for any n.
- Attempting to end up in a goal state.

Goal: 2 Gallons Water in the 4 Gallon Jug

2. Operators

Let us define the following 4 operators.

– FILL (jug): where jug = 3 gallon or 4 gallon, fill the jug fully with water.
– EMPTY (jug): where jug = 3 gallon or 4 gallon, empty the jug by pouring the water to ground.
– POUR (jug1, jug2): pour the water from jug1 to jug 2 until it is just filled
– TRANSFER (jug1, jug2): pour the water from jug1 to jug 2 completely.

State Space Representation

3. Abbreviations

G3 ➔ 3 Gallon Jug

G4 ➔ 4 Gallon Jug

4. State space representation of the given problem

State No.	Operator	State (G4,G3)
0 (Initial)	---------	(0, 0)
1	FILL (G3)	(0, 3)
2	TRANSFER (G3, G4)	(3, 0)
3	FILL (G3)	(3, 3)
4	POUR (G3, G4)	(4, 2)
5	EMPTY (G4)	(0, 2)
6 (Final)	TRANSFER (G3,G4)	(2, 0)

State Space Representation

5. Operators and Conditions

Operators		Conditions
FILL (G3)	:	G3 was not full
FILL (G4)	:	G4 was not full
EMPTY (G4)	:	G4 was not empty
TRANSFER (G3,G4)	:	G3 was not empty & G4 was not full
POUR (G3, G4)	:	G3 was not empty & G4 was not full

Production rules :

Sl No	Current state	Next State	Descritpion
1	(x,y) if x < 4	(4,y)	Fill the 4 gallon jug
2	(x,y) if y <3	(x,3)	Fill the 3 gallon jug
3	(x,y) if x > 0	(x-d, y)	Pour some water out of the 4 gallon jug
4	(x,y) if y > 0	(x, y-d)	Pour some water out of the 3-gallon jug
5	(x,y) if x>0	(0, y)	Empty the 4 gallon jug
6	(x,y) if y >0	(x,0)	Empty the 3 gallon jug on the ground
7	(x,y) if x+y >= 4 and y >0	(4, y-(4-x))	Pour water from the 3 –gallon jug into the 4 –gallon jug until the 4-gallon jug is full
8	(x, y) if x+y >= 3 and x>0	(x-(3-y), 3)	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if x+y <=4 and y>0	(x+y, 0)	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if x+y <= 3 and x>0	(0, x+y)	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	(0,2)	(2,0)	Pour the 2 gallons from 3-gallon jug into the 4-gallon jug
12	(2,y)	(0,y)	Empty the 2 gallons in the 4-gallon jug on the ground

Water Jug Problem : Solution

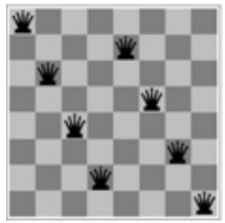
We can list the steps that may be followed to reach the goal state.

1. Fill the 3 gallon jug with water
2. Pour the water from 3 gallon jug to 4 gallon jug.
3. Again fill the 3 gallon jug with water
4. Pour the water carefully from 3 gallon jug to 4 gallon jug such that it is just filled.
5. Empty the 4 gallon jug
6. Transfer the water from 3 gallon jug to 4 gallon jug.
7. Stop.

Gallons in the 4-gallon jug	Gallons in the 3-gallon jug	Rule applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5 or 12
0	2	9 0r 11
2	0	

Problem Formulation: The 8-Queens

- Incremental formulation vs. complete-state formulation
- States-I-1: 0-8 queens on board
- Successor function-I-1:
  - add a queen to any square
  - # of possible states = (64*63*…*57=)
- States-I-2:
  - 0-8 non-attacking queens on board
- Successor function-I-2:
  - add a queen to a non-attacking square in the left-most empty column
  - # of possible states = 2057 --- ???
- Goal test: 8 queens on board, none attacked
- Path cost: of no interest (since only the final state count)



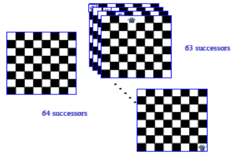
The Eight Queens Problem

- One strategy: guess at a solution
  - There are 4,426,165,368 ways to arrange 8 queens on a chessboard of 64 squares
- An observation that eliminates many arrangements from consideration
  - No queen can reside in a row or a column that contains another queen
    - Now: only 40,320 (8!) arrangements of queens to be checked for attacks along diagonals

N queens problem formulation

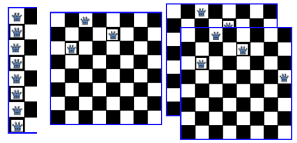
Formulation 1

- States: Any arrangement of 0 to 8 queens on the board
  - Initial state: 0 queens on the board
- Successor function: Add a queen in any square
- Goal test: 8 queens on the board, none are attacked



Formulation 2

- States: Any arrangement of 8 queens on the board
  - Initial state: All queens are at column 1
- Successor function: Change the position of any one queen
- Goal test: 8 queens on the board, none are attacked

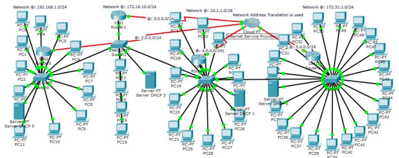


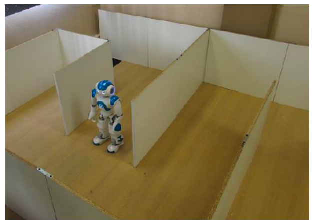
Formulation 3

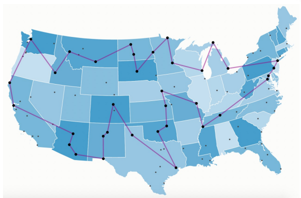
- States: Any arrangement of k queens in the first k rows such that none are attacked
  - Initial state: 0 queens on the board
- Successor function: Add a queen to the (k+1)th row so that none are attacked.
- Goal test : 8 queens on the board, none are attacked

Formulate – Real World Example

- Route finding problem, e.g., Google Maps
- Routing Video Streams in Computer Networks
- Traveling Salesman problem
- Robot Navigation





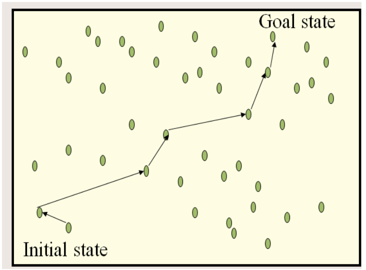


Formulate – Real World Example

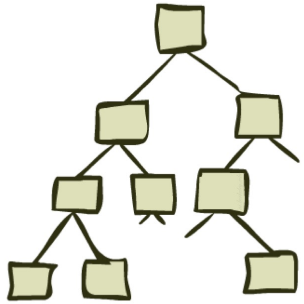
- **Airline Travel problem**
  - the task of a travel agent to book cheapest and fastest flight route from City A to City B.
  - E.g., Need to Travel from Delhi to Los Angeles within Rs. 70,000
  - Possible States – Each state is a location and current time.
    1. Initial State – Specified by User's query
    2. Actions – Flight from current location at a particular time with enough time for within-airport transfer if needed
    3. Transition Model – The destination location and arrival time
    4. Goal test – Are we at final destination at the specified time?
    5. Path Cost – Flight cost + Duration + Waiting time + Immigration, etc.

Searching for Solutions

- Solution for above Problems:
  - A sequence of possible actions starting at the initial state and reaching the destination specified
- Search Algorithms:
  - Given problem formulation as input, these algorithms would output the sequence of actions
  - Search Trees
    - Where states are nodes and actions are edges. The initial state will be the root node and possible actions are branches

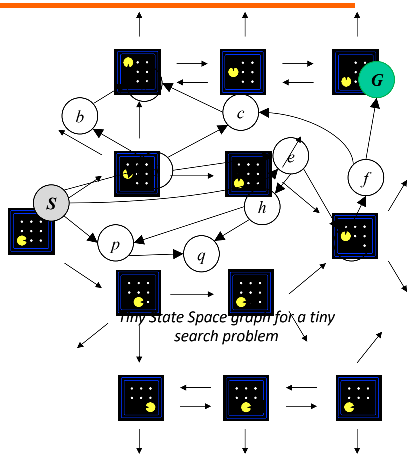


State Space Graphs and Search Trees



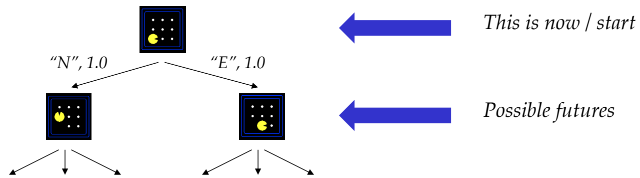
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea
  - Find solution without writing most of the graph down.
  - really care about the start and what you can do from the start -> search tree



Tiny State Space graph for a tiny search problem

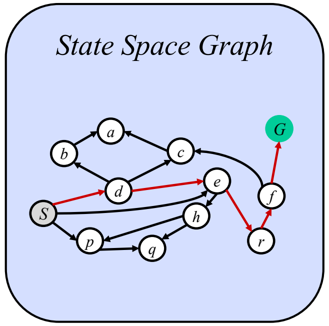
Search Trees



- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

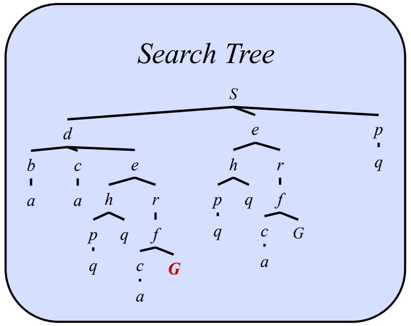
- Different plans that achieve the same state, will be different nodes in the tree.
- Every plan in the tree.
- Search ignores most of the tree.

State Space Graphs vs. Search Trees



Each NODE in the search tree is an entire PATH in the state space graph.

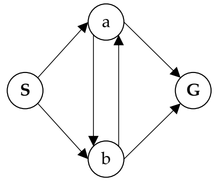
We construct both on demand – and we construct as little as possible.

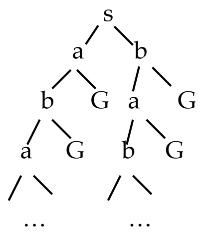


State Space Graphs vs. Search Trees

Consider this 4-state graph:

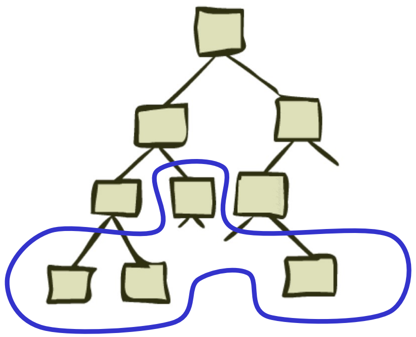
How big is its search tree (from S)?





Important: Lots of repeated structure in the search tree!

Tree Search



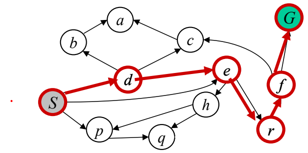
That's the abstraction. Now talk about algorithm.

General Tree Search

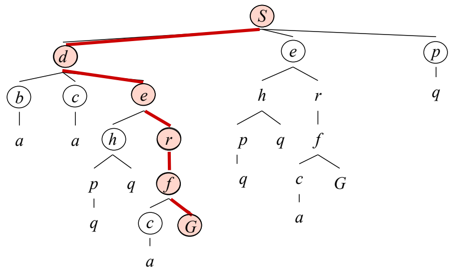
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains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

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Example: Tree Search



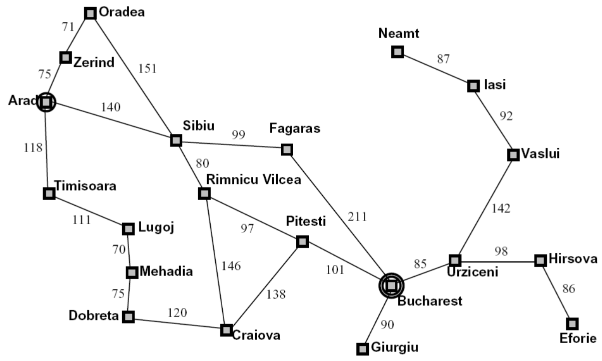
- **Frontier:**
  - Set of all leaf nodes available for expansion at any given point
- **Search Strategy:**
  - Choosing which state to expand next



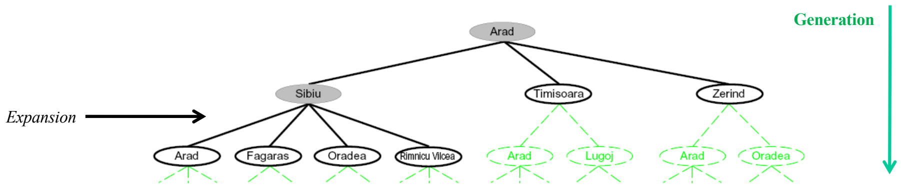
Fringe

~~s~~
~~s →d~~
s →e
s →p
s →d →b
s →d →c
~~s →d →e~~
s →d →e →h
~~s →d →e →r~~
~~s →d →e →r →f~~
s →d →e →r →f →c
~~s →d →e →r →f →G~~

Search Example: Romania

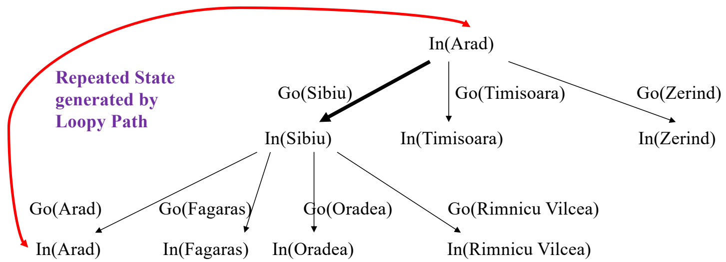


Searching for a solution with a Tree search Algorithm



- Frontier: Set of all leaf nodes available for expansion at any given point
  - Search Strategy: Choosing which state to expand next
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

Tree Search Algorithm: Romania



Redundant Path: More than one way to reach a state from another.
- E.g., Arad – Sibiu (140 kms) and Arad-Zerind-Oradea-Sibiu (297 kms)
- Loopy Path is a special case

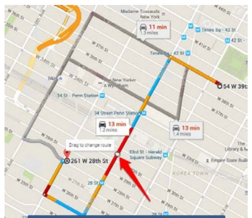
Revision

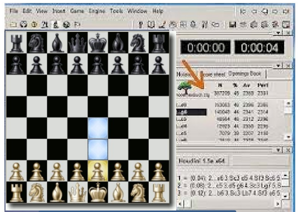
- Description of the problem
  - Define a state space that contains all the possible configurations of the relevant objects.
  - Specify one or more states within that space that describe possible situations from which the problem solving process may start (initial state)
  - Specify one or more states that would be acceptable as solutions to the problem. (goal states)
  - Specify a set of rules that describe the actions (operations) available.
    - What are unstated assumptions?
    - How general should the rules be?
    - How much knowledge for solutions should be in the rules?

Can Solution Steps be ignored or undone?

- Suppose we are trying to prove a math theorem.
  - We can prove a lemma. If we find the lemma is not of any help, we can still continue.
- 8-puzzle problem
- Chess: A move cannot be taken back.
- Important classes of problems:
  - Ignorable (e.g. theorem proving) in which solution steps can be ignored
  - Recoverable (e.g. 8-puzzle) in which solution steps can be undone.
  - Irrecoverable (e.g. chess) in which solution steps cannot be undone

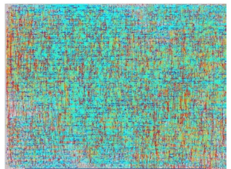
COMPLEX PROBLEMS & SOLUTIONS


Path Finding

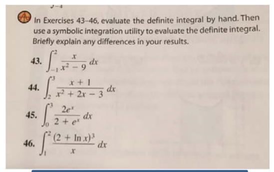

Chess Playing


Robot Assembly

COMPLEX PROBLEMS & SOLUTIONS


VLSI Chip Design


Time-Table Scheduling



Symbolic Integration

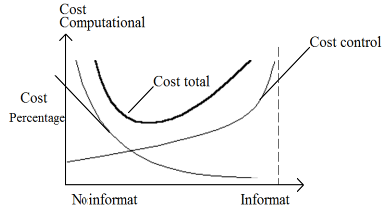
Measuring Search Algorithm Performance

- Completeness :
  - Is the algorithm guaranteed to find a solution when there is one?
- Optimality :
  - Does the algorithm find the optimal solution?
- Time Complexity :
  - How long does it take to find the solution?
- Space Complexity :
  - How much memory is needed to perform the search?

- **Time and space complexity**
  - Search in AI is represented by initial state, actions and transitions which usually result in infinite Nodes and Edges in a graph.Hence, the complexity is rather measured by
    - Branching Factor (b): Maximum number of successors of any node
    - Depth (d) of the shallowest goal, i.e., number of steps from initial node
    - Maximum length (m) of any path in state space (may be ∞)

Measuring Search Algorithm Performance

- Effectiveness of an algorithm can be measured by:
  - Search Cost:
    - Time spent by the algorithm in finding a solution (can also include memory usage)
  - Solution Cost:
    - Path cost of the solution
  - Total Cost:
    - Search Cost + Solution Cost (if not in same units, convert them)



Search Algorithms

- Uninformed Search Algorithms
  - No additional information about states beyond what is provided in problem formulation
  - Generate successors and distinguish a goal state from a non-goal state
- Informed Search Algorithms
  - Strategies that know if one non-goal state is more promising than another non-goal state
  - Also called Heuristic Search strategies

Summary

- Four steps for designing a program to solve a problem:
  1. Define the problem precisely
  2. Analyse the problem
  3. Identify and represent the knowledge required by the task
  4. Choose one or more techniques for problem solving and apply those techniques to the problem.

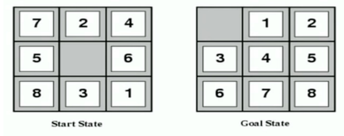
Next :

- Module 3: Search Strategies
  - PART 3.1: Search
  - PART 3.2: Uninformed Search
  - PART 3.3: Informed/Heuristic Search
  - PART 3.4: Beyond Classical Search
  - PART 3.5: Problem reduction
  - PART 3.6: Adversarial Search

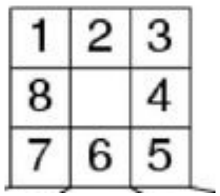
Explore

- Problem Formulation by AI Search Methods for the following famous problems :
  - Tic-Tac-Toe
  - 8-puzzel problem
  - 8 Queens Problem
  - Missionaries and Cannibals
  - Tower of Hanoi
  - Travelling Salesman Problem (TSP)
  - Rubik's Cube

- # of states
- State :
- Initial State:
- Action :
- Goal test :
- path cost :



Search Tree



References

- Artificial intelligence : A Modern Approach, Prentice Hall by Stuart Russell, Peter Norvig,
- Artificial Intelligence by Elaine Rich & Kevin Knight, Third Ed, Tata McGraw Hill
- Artificial Intelligence and Expert System by Patterson
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE