Mehran University of Engineering & Technology, Pakistan

Problems & Problem Spaces

Department of Software Engineering

Problems

There are 3 general categories of problems in AI:

a) Single-agent pathfinding problems.

b) Two-player games.

c) Constraint satisfaction problems.

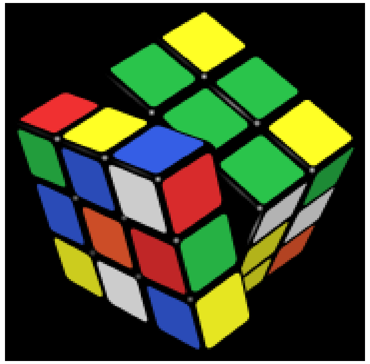
To solve a Problem:

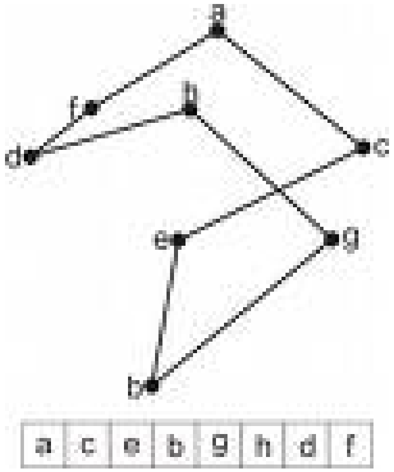
1. Define the problem precisely.
2. Analyze the problem.
3. Isolate and represent the task knowledge.
4. Choose the best problem-solving technique and apply it.

Dr. Isma F. Siddiqui *(isma.farah@faculty.muet.edu.pk)*

Single Agent Pathfinding Problems

- In these problems, in each case,
  - We have a ***single problem-solver*** making the decisions,
  - The task is to find a sequence of primitive steps: ***from the initial location*** to ***the goal location***.
- **Famous examples:**
  - Rubik's Cube (Erno Rubik, 1975).
  - Sliding-Tile puzzle.
  - Navigation - Travelling Salesman Problem.



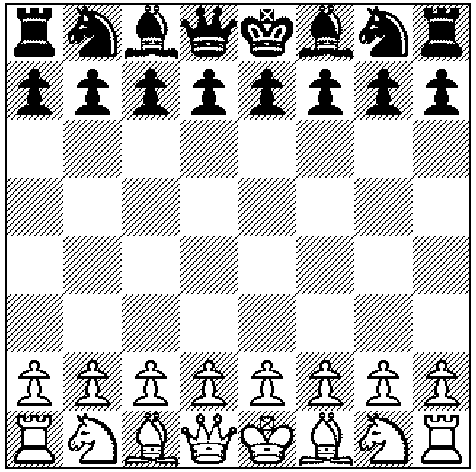


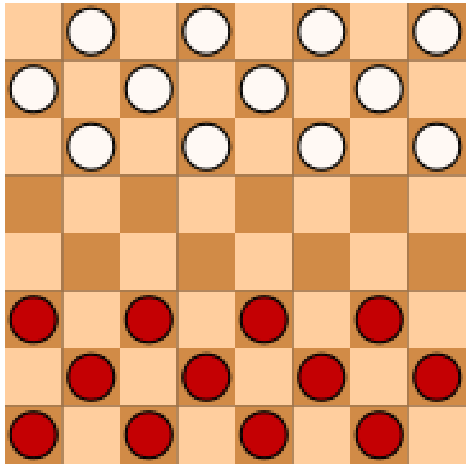
Dr. Isma F. Siddiqui *(isma.farah@faculty.muet.edu.pk)*

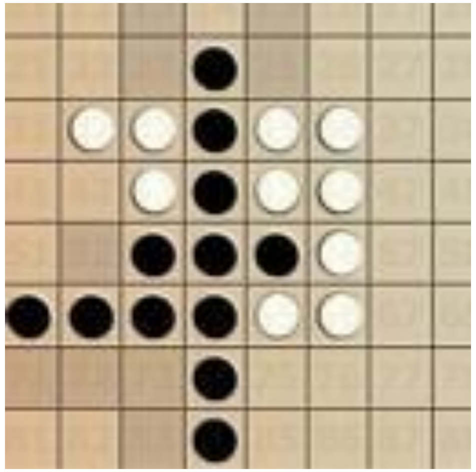
Two-Player Games

- In a ***two-player game***:
    - One must consider the ***moves of an opponent***,
    - The **ultimate goal** is a strategy that will **guarantee a win** whenever possible.

- Researchers are starting to consider more **complex games**, many of them involve **an element of chance *(Probability)***.
- The brilliant ***Chess, Checkers, and Othello players*** in the world are computer programs!







Mehran University of Engineering & Technology, Pakistan

Constraint-Satisfaction Problems

- In these problems, we also have ***a single-agent making all the decisions***, but here we are ***not*** concerned with ***the sequence of steps required to reach the solution***, but simply the solution itself.

- The task is to identify:
  - A ***state of the problem***, (all the ***constraints (limitations)*** of the problem are satisfied)

- **Famous Examples:**
  - Eight Queens Problem.

Department of Software Engineering

The Problem Spaces

- A ***problem space*** consists of:
  - A set of ***states of a problem***,
  - A set of ***operators*** that change the state.

- **State** : a symbolic structure that:
  - Represents **a single configuration of the problem** in a sufficient detail to allow problem solving to proceed.
- **Operator** : a function that:
  - Takes a state
  - Maps it to another state.

A problem (problem instance)

- It consists of:
  - A problem space
  - An initial state
  - A set of goal states
    - explicitly or implicitly characterized
  - A solution,
    - a sequence of operator applications leading from the initial state to a goal state

      or

    - a path through the state space from initial to final state.

Representing States

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

Representing States (Cont.)

The **state-space representation of a problem** is a **triplet (I, O, G)**

- where:
  - **I** - initial state,
  - **O** - a set of operators on states,
  - **G** - goal states
- A state space can be organized as a graph:
  - ***Nodes:*** states in the space
  - ***Arcs:*** actions/operations
- A ***solution*** is a path from the initial state to a goal state.
- 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.
  - **Example:** Chess has about **10^{120} states** in a typical game.

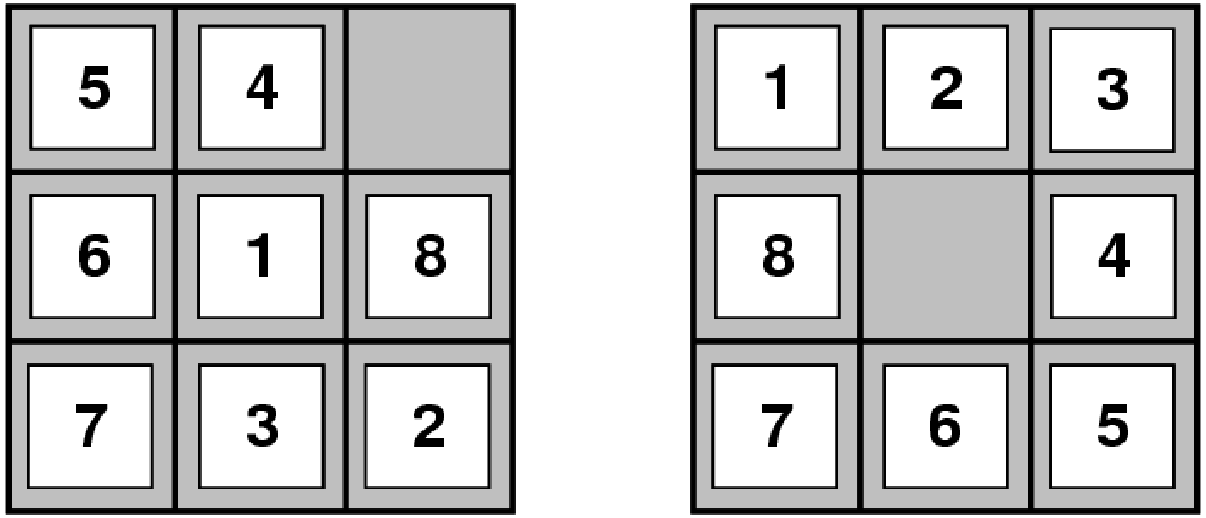
Some Example Problems

- Toy problems and micro-worlds
  - 8-Puzzle
  - Missionaries and Cannibals
  - Cryptarithmetic
  - Remove 5 Sticks
  - Traveling Salesman Problem (TSP)
- Real-world-problems

Mehran University of Engineering & Technology, Pakistan

8-Puzzle

- Given:
  - An initial configuration of 8 numbered tiles on a 3 x 3 board,
  - Move the tiles in such a way so as to produce a desired goal configuration of the tiles.



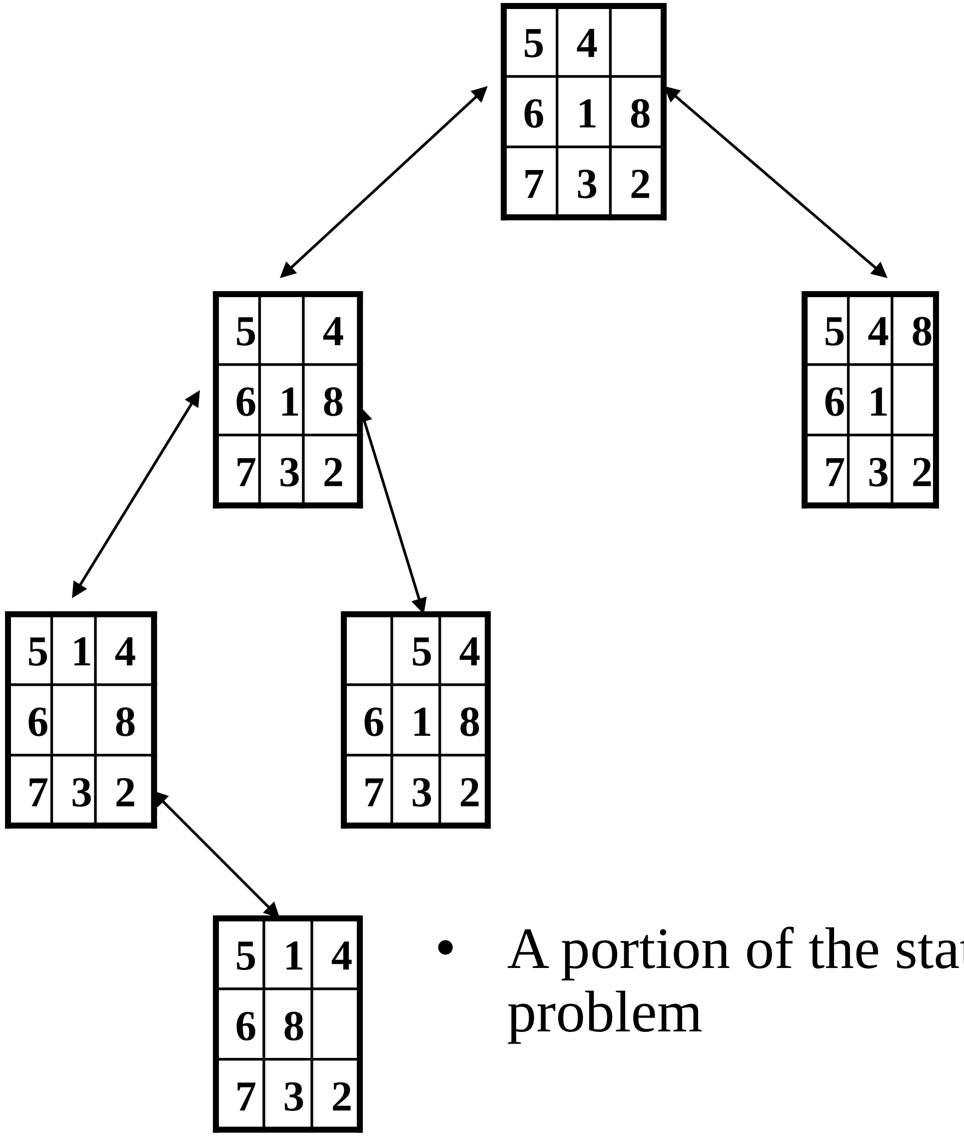
Start State Goal State

Department of Software Engineering

8-Puzzle

- ***State:*** **3 x 3** array configuration of the tiles on the board.
- ***Operators:*** Move Blank square **Left**, **Right**, **Up** or **Down**.
  - This is an efficient encoding of the operators than one in which each of four possible moves for each of the 8 distinct tiles is used.
- ***Initial State:*** A particular **configuration** of the board.
- ***Goal:*** A particular **configuration** of the board.
  - The state space is partitioned into **two subspaces**
  - NP-complete problem, requiring O(2^k) steps where k is the length of the solution path.
  - 15-puzzle problems (4 x 4 grid with 15 numbered tiles), and N-puzzles (N = n^2-1)

8-Puzzle

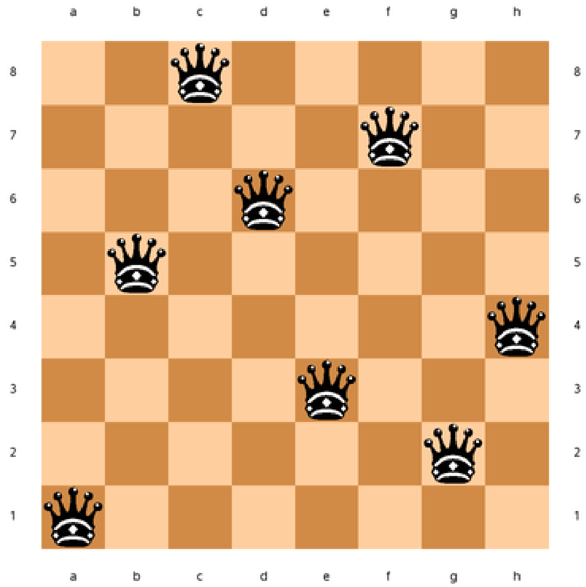


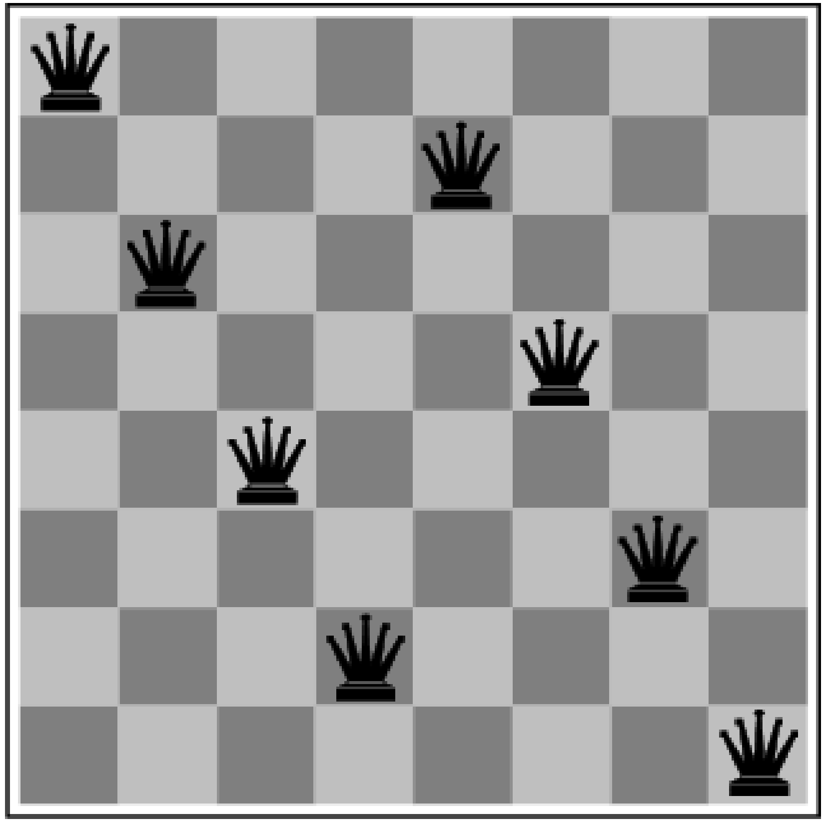
- A portion of the state space representation of a 8-Puzzle problem

Dr. Isma F. Siddiqui *(isma.farah@faculty.muet.edu.pk)*

The 8-Queens Problem

- Place ***eight queens*** on a chessboard such that **no queen** attacks any other!
  - Total # of states: 4.4x109
  - Total # of solutions: 12 (or 96)



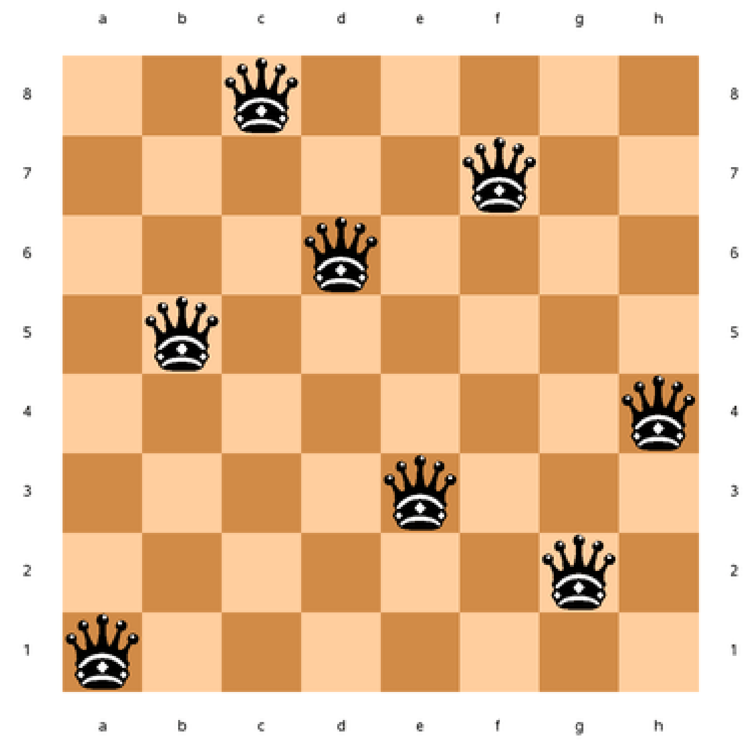


Dr. Isma F. Siddiqui *(isma.farah@faculty.muet.edu.pk)*

Mehran University of Engineering & Technology, Pakistan

Eight Queens Problem

- It is the problem of **placing eight chess queens** on ***an 8×8 chessboard***:
  - No two queens **threaten** each other
- A solution requires that:
  - No two queens share the same row, column, or diagonal.
  - An example of the '***n***' queens problem of placing '***n***' non-attacking queens on an '***n * n***' chessboard.
- Solutions:
  - All natural numbers '***n***' with the exception of ***n = 2*** and ***n = 3***



n	1	2	3	4	5	6	7	8	9	10	...
fundamental	1	0	0	1	2	1	6	12	46	92	...
all	1	0	0	2	10	4	40	92	352	724	...

Department of Software Engineering

Missionaries and Cannibals

Three cannibals and **three missionaries** come to a **crocodile infested river**. There is ***a boat*** on their side that can be used by **either one or two persons**. If cannibals outnumber the missionaries ***at any time***, the cannibals eat the missionaries. How can they ***use the boat to cross the river*** so that all missionaries ***survive*** ?

There are 3 missionaries, 3 cannibals, and 1 boat that can carry up to two people on one side of a river.

- ***Goal:*** Move all the missionaries and cannibals across the river.
- ***Constraint:*** Missionaries can never be outnumbered by cannibals on either side of river, or else the missionaries are killed.
- ***State:*** configuration of missionaries and cannibals and boat on each side of river.
- ***Operators:*** Move boat containing some set of occupants across the river (in either direction) to the other side.

Missionaries and Cannibals Solution

	A side		G side
0 Initial setup:	MMMCCC	B	-
1 Two cannibals cross over:	MMMC		B CC
2 One comes back:	MMMCC	B	C
3 Two cannibals go over again:	MMM		B CCC
4 One comes back:	MMMC	B	CC
5 Two missionaries cross:	MC		B MMCC
6 A missionary & cannibal return:	MMCC	B	MC
7 Two missionaries cross again:	CC		B MMMC
8 A cannibal returns:	CCC	B	MMM
9 Two cannibals cross:	C		B MMMCC
10 One returns:	CC	B	MMMC
11 And brings over the third:	-		B MMMCCC

State Space Example

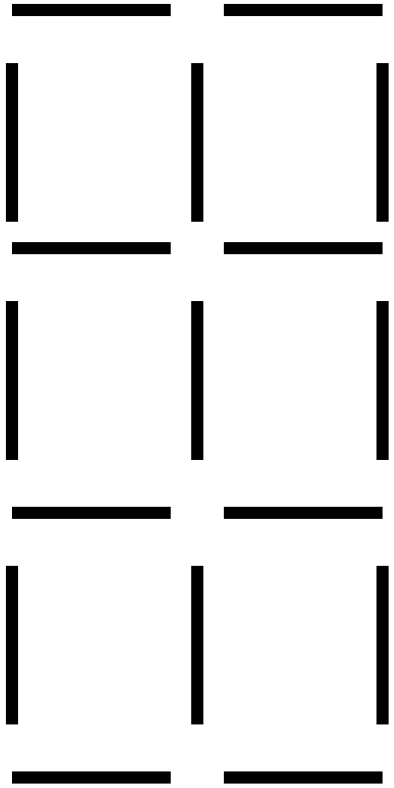
- 3 missionaries, 3 cannibals, 1 boat
- The canoe (boat) can hold at most two people
- Cannibals may never outnumber missionaries (on either side)
- Initial state is **(3, 3, 1)**, representing the number of missionaries, cannibals, boats on the initial side
- The goal state is **(0, 0, 0)**
- Operators are addition or subtraction of the vectors
  (1 0 1), (2 0 1), (0 1 1), (0 2 1), (1 1 1)
- Operators apply if result is between **(0 0 0)** and **(3 3 1)**

Another:

(331,220,321,300,311,110,221,020,031,010,021,000)

Remove 5 Sticks

Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.



Cryptarithmetic

- Find an assignment of digits **(0, ..., 9)** to letters so that a given arithmetic expression is true. examples: **SEND + MORE = MONEY** and

```
FORTY                            Solution:   29786
+   TEN                                        850
+   TEN                                        850
 -----                                       -----
 SIXTY                                       31486
F=2, O=9, R=7, etc.
```

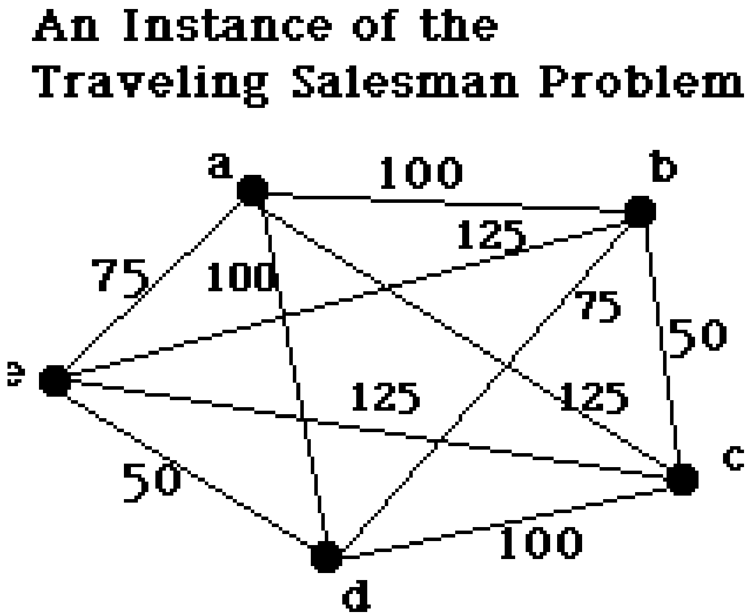
- Note: In this problem, the solution is ***NOT*** a sequence of actions that transforms the initial state into ***the goal state***, but rather the solution is simply finding ***a goal node*** that includes an assignment of digits to each of the ***distinct letters*** in the given problem.

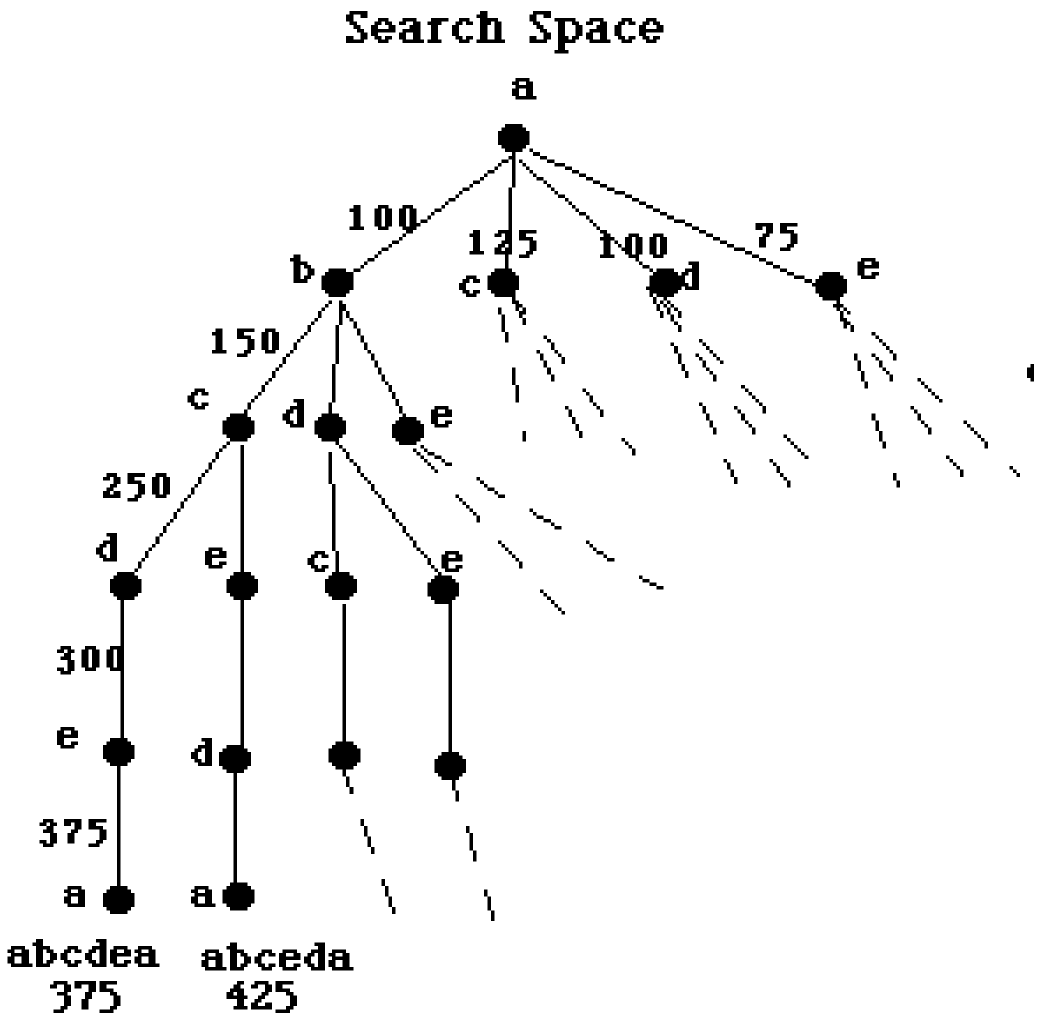
Traveling Salesman Problem

- Given a road map of n cities, find the shortest tour which visits every city on the map exactly once and then return to the original city ***(Hamiltonian circuit)***

- (Geometric version):
  - a complete graph of n vertices.
  - n!/2n legal tours
  - Find one legal tour that is shortest

State Space of TSP





Formalizing Search in a State Space

- A state space is a ***graph, (V, E)*** where V is a set of ***nodes*** and E is a set of ***arcs***, where each arc is directed from a node to another node
- ***node***: corresponds to a ***state***
  - state description
  - plus optionally other information related to the parent of the node, operation to generate the node from that parent, and other bookkeeping data)
- **arc:** corresponds to an applicable action/operation.
  - the source and destination nodes are called as ***parent (immediate predecessor) and child (immediate successor)*** nodes with respect to each other
  - ancestors((predecessors) and descendants (successors)
  - each arc has a *fixed, non-negative* **cost** associated with it, corresponding to the cost of the action

Formalizing Search in a State Space

- **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

Mehran University of Engineering & Technology, Pakistan

Formalizing Search in a State Space

- ***State-space search*** is the process of searching through a state space for a solution by making explicit a sufficient portion of an implicit state-space graph to include a goal node.
  - Hence, initially **V={S}**, where S is the **start node**; when S is expanded, its successors are generated and those nodes are added to V and the associated arcs are added to E. This process continues until a goal node is **generated (included in V)** and identified (by goal test)
- During search, a node can be in one of the three categories:
  - Not generated yet (has not been made explicit yet)
  - **OPEN:** generated but not expanded
  - **CLOSED:** expanded
  - Search strategies differ mainly on ***how to select an OPEN node*** for expansion at each step of search

Department of Software Engineering

A General State-Space Search Algorithm

- Node n
  - state description
  - parent (may use a back pointer) **(if needed)**
  - Operator used to generate n **(optional)**
  - Depth of n **(optional)**
  - Path cost from S to n **(if available)**
- **OPEN** list
  - initialization: {S}
  - node insertion/removal depends on specific search strategy
- **CLOSED** list
  - initialization: {}
  - organized by back pointers

Dr. Isma F. Siddiqui *(isma.farah@faculty.muet.edu.pk)*

A General State-Space Search Algorithm

```
open := {S}; closed :={ };
repeat
    n := select(open);        /* select one node from open for expansion */
  if n is a goal
        then exit with success;  /* delayed goal testing */
  expand(n)
          /* generate all children of n
             put these newly generated nodes in open (check duplicates)
             put n in closed (check duplicates) */
until open = {};
exit with failure
```