Module 2

Problem Solving using Search- (Single agent search)

Version 1 CSE IIT, Kharagpur

2.1 Instructional Objective

- The students should understand the state space representation, and gain familiarity with some common problems formulated as state space search problems.
- The student should be familiar with the following algorithms, and should be able to code the algorithms
    - greedy search
    - DFS
    - BFS
    - uniform cost search
    - iterative deepening search
    - bidirectional search
- Given a problem description, the student should be able to formulate it in terms of a state space search problem.
- The student should be able to understand and analyze the properties of these algorithms in terms of
    - time complexity
    - space complexity
    - termination
    - optimality
- Be able to apply these search techniques to a given problem whose description is provided.
- The student should understand how implicit state spaces can be unfolded during search.
- Understand how states can be represented by features.

The student will learn the following strategies for uninformed search:

- Breadth first search
- Depth first search
- Iterative deepening search
- Bidirectional search

For each of these they will learn

- The algorithm
- The time and space complexities
- When to select a particular strategy

At the end of this lesson the student should be able to do the following:

- Analyze a given problem and identify the most suitable search strategy for the problem.
- Given a problem, apply one of these strategies to find a solution for the problem.

Lesson 3

Introduction to State Space Search

Version 1 CSE IIT, Kharagpur

2.2 State space search

- Formulate a problem as a state space search by showing the legal problem states, the legal operators, and the initial and goal states .
- A state is defined by the specification of the values of all attributes of interest in the world
- An operator changes one state into the other; it has a precondition which is the value of certain attributes prior to the application of the operator, and a set of effects, which are the attributes altered by the operator
- The initial state is where you start
- The goal state is the partial description of the solution

2.2.1 Goal Directed Agent

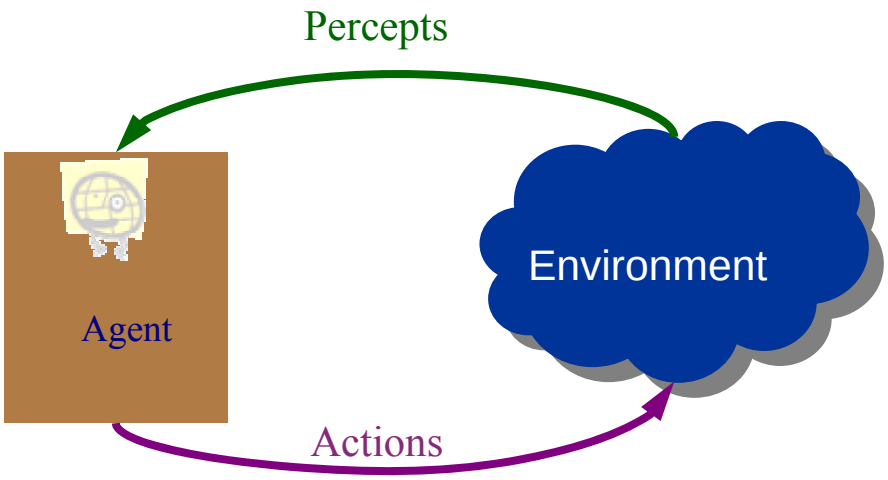

Figure 1

We have earlier discussed about an intelligent agent. Today we will study a type of intelligent agent which we will call a goal directed agent.

A goal directed agent needs to achieve certain goals. Such an agent selects its actions based on the goal it has. Many problems can be represented as a set of states and a set of rules of how one state is transformed to another. Each state is an abstract representation of the agent's environment. It is an abstraction that denotes a configuration of the agent.
Initial state : The description of the starting configuration of the agent
An action/ operator takes the agent from one state to another state. A state can have a number of successor states.
A plan is a sequence of actions.

A goal is a description of a set of desirable states of the world. Goal states are often specified by a goal test which any goal state must satisfy.

Let us look at a few examples of goal directed agents.

1. 15-puzzle: The goal of an agent working on a 15-puzzle problem may be to reach a configuration which satisfies the condition that the top row has the tiles 1, 2 and 3. The details of this problem will be described later.
2. The goal of an agent may be to navigate a maze and reach the HOME position.

The agent must choose a sequence of actions to achieve the desired goal.

2.2.2 State Space Search Notations

Let us begin by introducing certain terms.
An initial state is the description of the starting configuration of the agent
An action or an operator takes the agent from one state to another state which is called a successor state. A state can have a number of successor states.
A plan is a sequence of actions. The cost of a plan is referred to as the path cost. The path cost is a positive number, and a common path cost may be the sum of the costs of the steps in the path.

Now let us look at the concept of a search problem.
Problem formulation means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another.
Search is the process of considering various possible sequences of operators applied to the initial state, and finding out a sequence which culminates in a goal state.

2.2.3 Search Problem

We are now ready to formally describe a search problem.
A search problem consists of the following:

- S: the full set of states
- s_0 : the initial state
- A:S→S is a set of operators
- G is the set of final states. Note that G $\subseteq$S

These are schematically depicted in Figure 2.

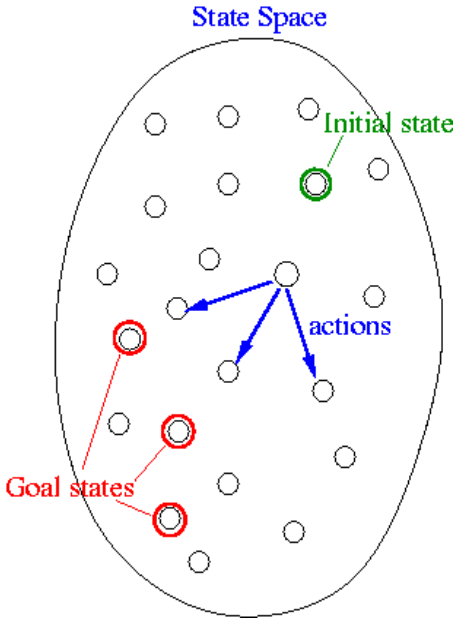

Figure 2

The search problem is to find a sequence of actions which transforms the agent from the initial state to a goal state g∈G. A search problem is represented by a 4-tuple {S, s_0, A, G}.

S: set of states

$s_0 \in S$: initial state

A: S→ S operators/ actions that transform one state to another state

G : goal, a set of states. $G \subseteq S$

This sequence of actions is called a solution plan. It is a path from the initial state to a goal state. A *plan* P is a sequence of actions.

$P = \{a_0, a_1, \ldots, a_N\}$ which leads to traversing a number of states $\{s_0, s_1, \ldots, s_{N+1} \in G\}$. A sequence of states is called a path. The cost of a path is a positive number. In many cases the path cost is computed by taking the sum of the costs of each action.

Representation of search problems

A search problem is represented using a directed graph.

- The states are represented as nodes.
- The allowed actions are represented as arcs.

Searching process

The generic searching process can be very simply described in terms of the following steps:

Do until a solution is found or the state space is exhausted.

1. Check the current state
2. Execute allowable actions to find the successor states.
3. Pick one of the new states.
4. Check if the new state is a solution state
   If it is not, the new state becomes the current state and the process is repeated

2.3 Examples

2.3.1 Illustration of a search process

We will now illustrate the searching process with the help of an example. Consider the problem depicted in Figure 3.

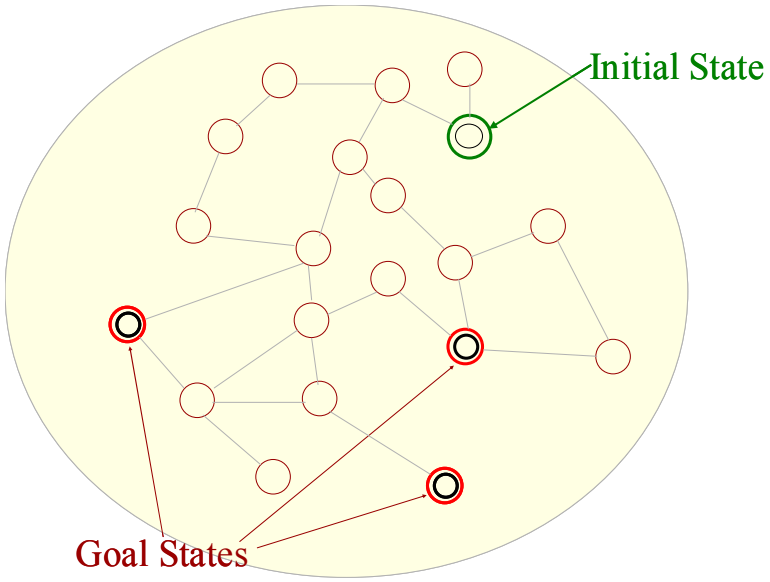

Figure 3

s_0 is the initial state.
The successor states are the adjacent states in the graph.
There are three goal states.

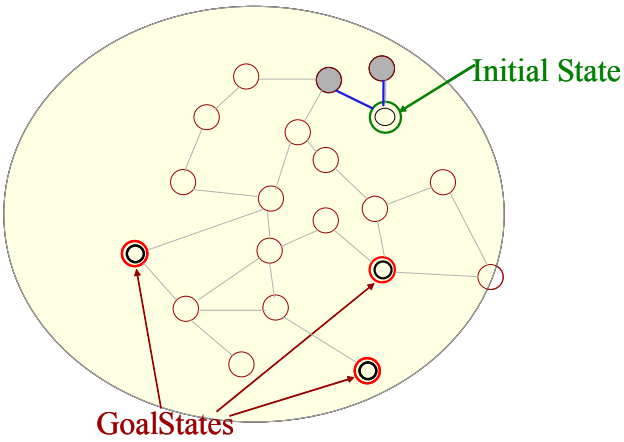

Figure 4

The two successor states of the initial state are generated.

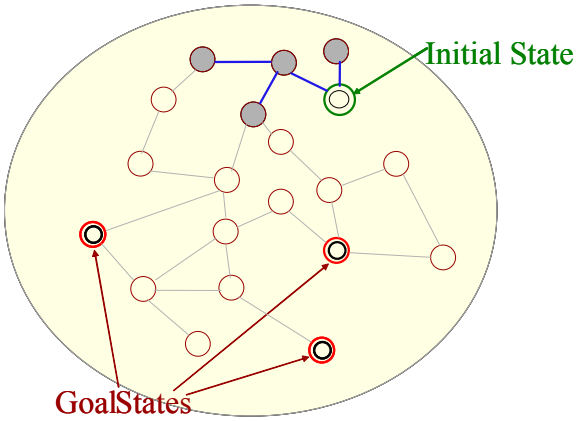

Figure 5

The successors of these states are picked and their successors are generated.

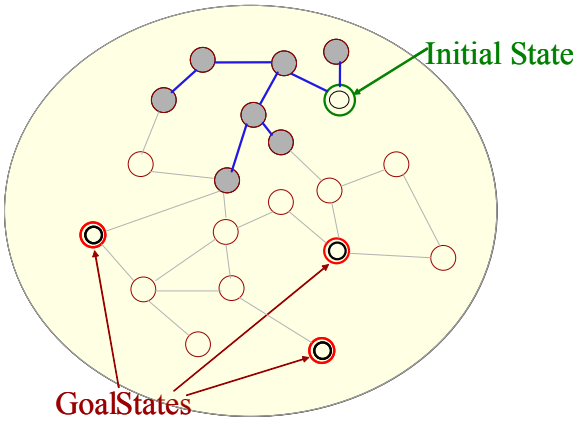

Figure 6

Successors of all these states are generated.

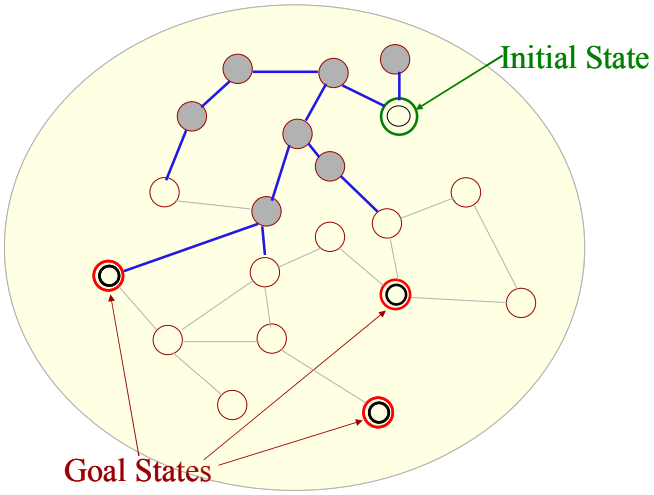

Figure 7

The successors are generated.

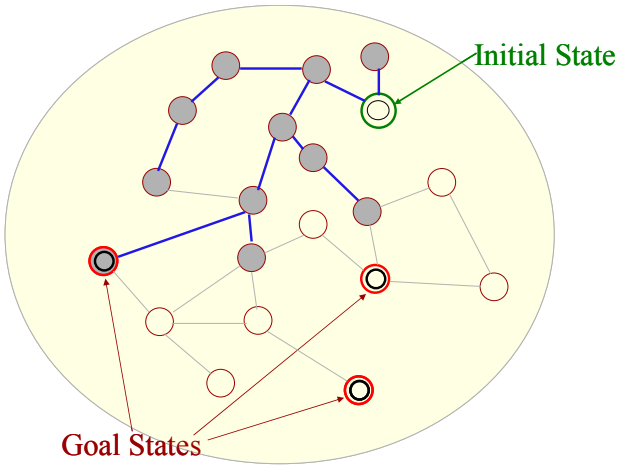

Figure 8

A goal state has been found.

The above example illustrates how we can start from a given state and follow the successors, and be able to find solution paths that lead to a goal state. The grey nodes define the search tree. Usually the search tree is extended one node at a time. The order in which the search tree is extended depends on the search strategy.

We will now illustrate state space search with one more example – the pegs and disks problem. We will illustrate a solution sequence which when applied to the initial state takes us to a goal state.

2.3.2 Example problem: Pegs and Disks problem

Consider the following problem. We have 3 pegs and 3 disks.
Operators: one may move the topmost disk on any needle to the topmost position to any other needle

In the goal state all the pegs are in the needle B as shown in the figure below..

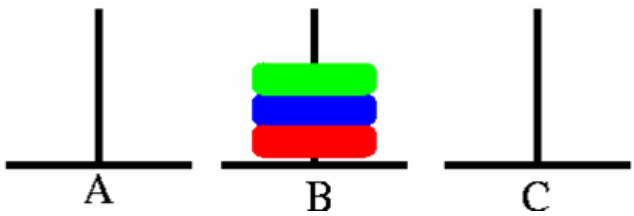

Figure 9

The initial state is illustrated below.

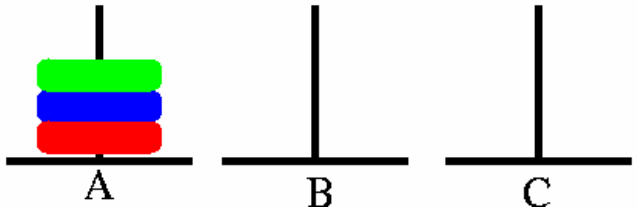

Figure 10

Now we will describe a sequence of actions that can be applied on the initial state.

Step 1: Move A → C

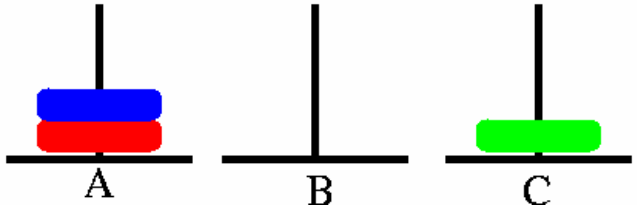

Figure 11

Step 2: Move A → B

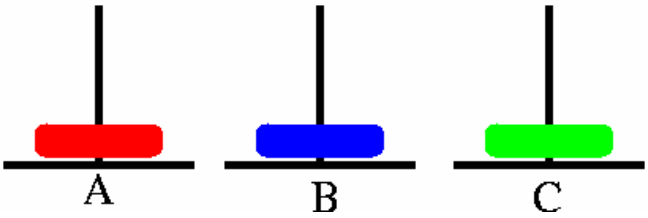

Figure 12

Step 3: Move A → C

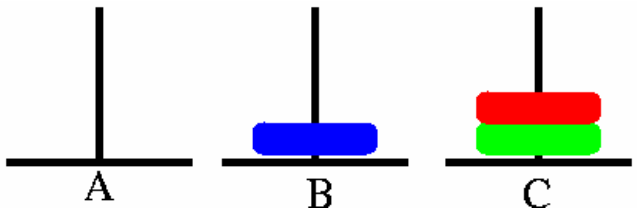

Figure 13

Step 4: Move B→ A

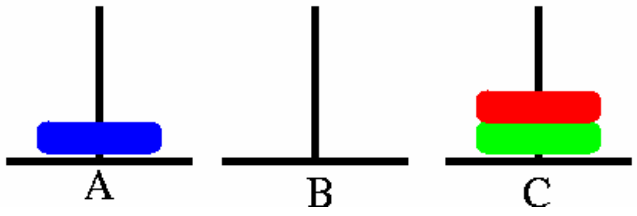

Figure 14

Step 5: Move C → B

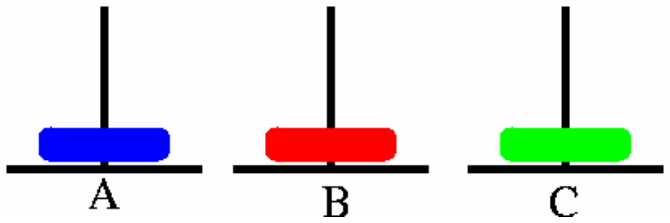

Figure 15

Step 6: Move A → B

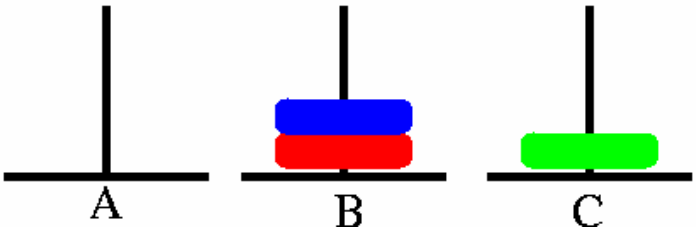

Figure 16

Step 7: Move C→ B

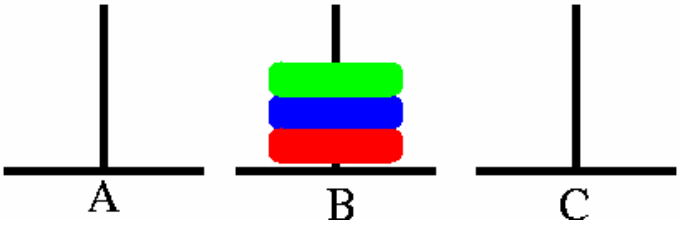

Figure 17

We will now look at another search problem – the 8-queens problem, which can be generalized to the N-queens problem.

2.3.4 8 queens problem

The problem is to place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal

The picture below on the left shows a solution of the 8-queens problem. The picture on the right is not a correct solution, because some of the queens are attacking each other.

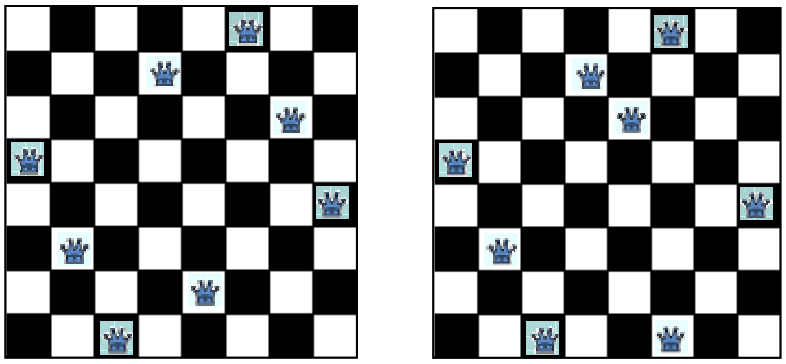

Figure 18

How do we formulate this in terms of a state space search problem? The problem formulation involves deciding the representation of the states, selecting the initial state representation, the description of the operators, and the successor states. We will now show that we can formulate the search problem in several different ways for this problem.

N queens problem 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

The initial state has 64 successors. Each of the states at the next level have 63 successors, and so on. We can restrict the search tree somewhat by considering only those successors where no queen is attacking each other. To do that we have to check the new queen against all existing queens on the board. The solutions are found at a depth of 8.

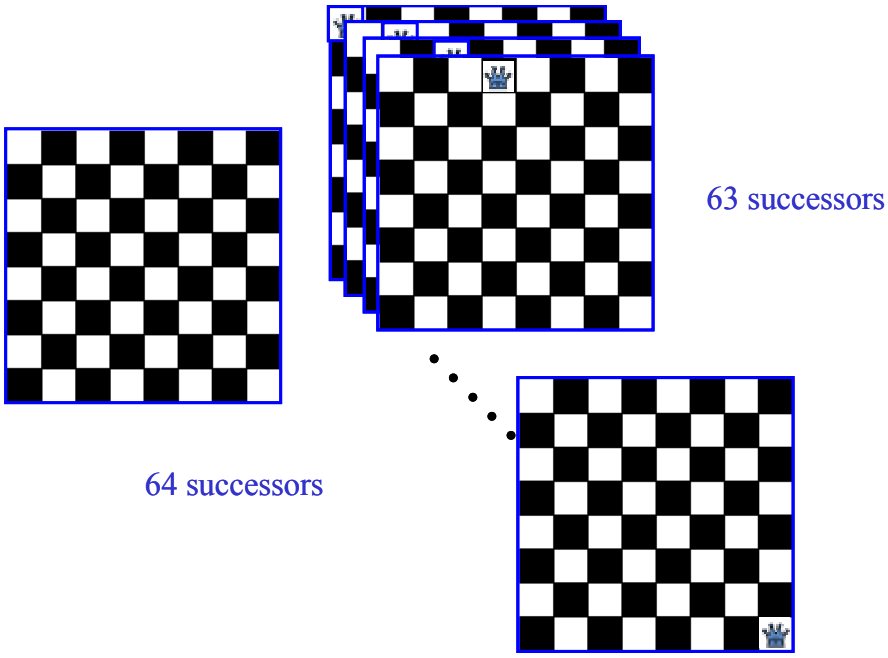

Figure 19

N queens problem 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

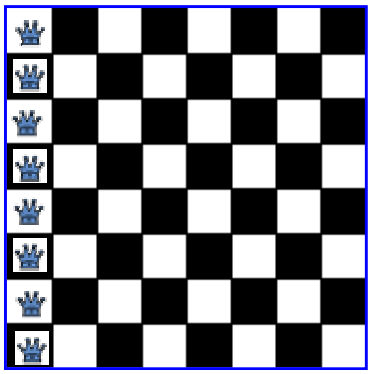

Figure 20

If we consider moving the queen at column 1, it may move to any of the seven remaining columns.

N queens problem 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

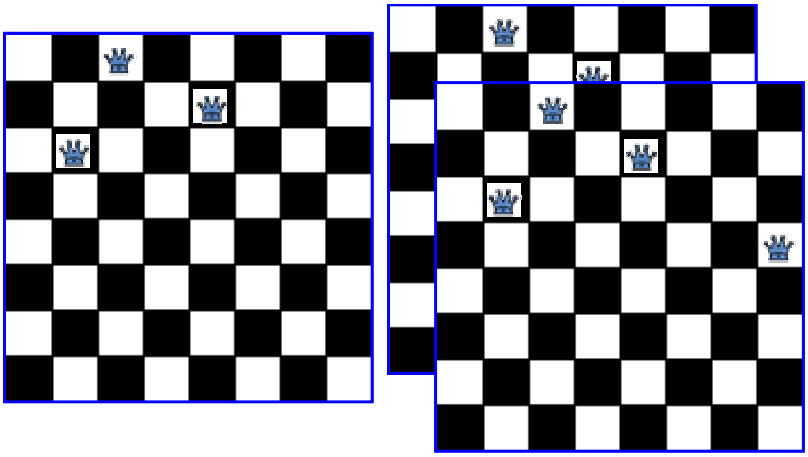

Figure 21

We will now take up yet another search problem, the 8 puzzle.

2.3.5 Problem Definition - Example, 8 puzzle

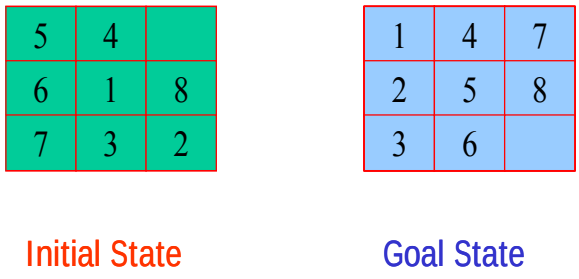

Figure 22

In the 8-puzzle problem we have a 3×3 square board and 8 numbered tiles. The board has one blank position. Bocks can be slid to adjacent blank positions. We can alternatively and equivalently look upon this as the movement of the blank position up, down, left or right. The objective of this puzzle is to move the tiles starting from an initial position and arrive at a given goal configuration.

The 15-puzzle problems is similar to the 8-puzzle. It has a 4×4 square board and 15 numbered tiles

The state space representation for this problem is summarized below:

States: A state is a description of each of the eight tiles in each location that it can occupy.
Operators/Action: The blank moves left, right, up or down
Goal Test: The current state matches a certain state (e.g. one of the ones shown on previous slide)
Path Cost: Each move of the blank costs 1

A small portion of the state space of 8-puzzle is shown below. Note that we do not need to generate all the states before the search begins. The states can be generated when required.

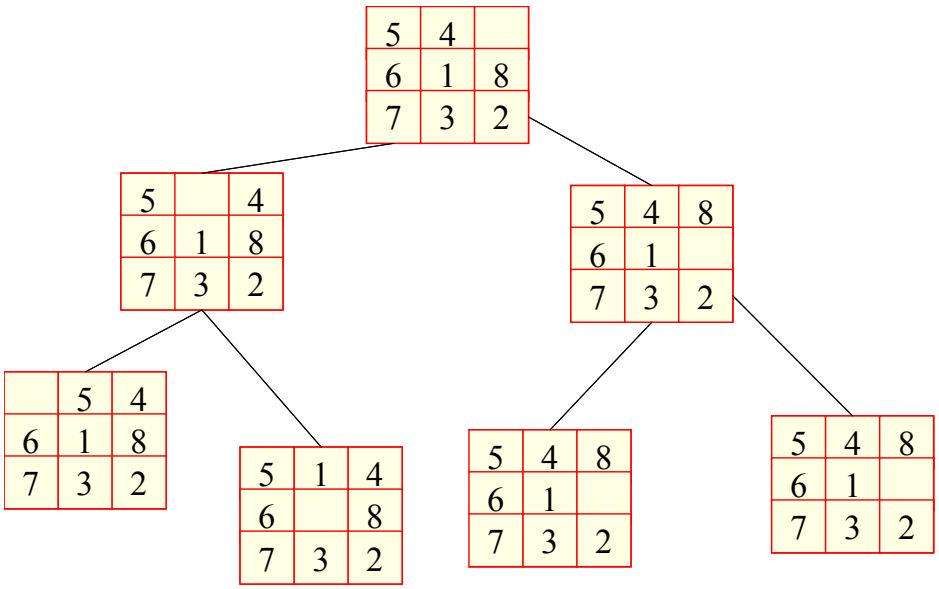

Figure 23

8-puzzle partial state space

2.3.6 Problem Definition - Example, tic-tac-toe

Another example we will consider now is the game of tic-tac-toe. This is a game that involves two players who play alternately. Player one puts a X in an empty position. Player 2 places an O in an unoccupied position. The player who can first get three of his symbols in the same row, column or diagonal wins. A portion of the state space of tic-tac-toe is depicted below.

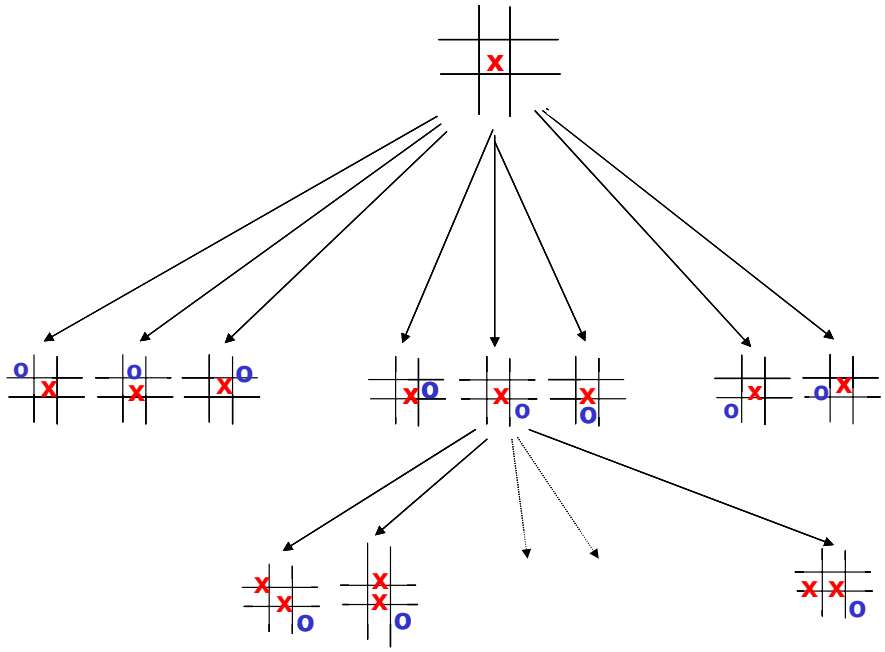

Figure 24

Now that we have looked at the state space representations of various search problems, we will now explore briefly the search algorithms.

2.3.7 Water jug problem

You have three jugs measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You need to measure out exactly one gallon.

- Initial state: All three jugs are empty
- Goal test: Some jug contains exactly one gallon.
- Successor function:Applying the
  - **action** *tranfer* to jugs i and j with capacities Ci and Cj and containing Gi and Gj gallons of water, respectively, leaves jug i with max(0, Gi−(Cj −Gj)) gallons of water and jug j with min(Cj, Gi+ Gj) gallons of water.
  - Applying the **action** *fill* to jug i leaves it with Ci gallons of water.
- Cost function: Charge one point for each gallon of water transferred and each gallon of water filled

Explicit vs Implicit state space

The state space may be explicitly represented by a graph. But more typically the state space can be implicitly represented and generated when required. To generate the state space implicitly, the agent needs to know

- the initial state
- The operators and a description of the effects of the operators

An operator is a function which "expands" a node and computes the successor node(s). In the various formulations of the N-queen's problem, note that if we know the effects of the operators, we do not need to keep an explicit representation of all the possible states, which may be quite large.