Problem Solving, Search and Control Strategies

Problem Solving, Search and Control Strategies

1. **General Problem Solving**
    - Problem solving definitions:
    - problem space,
    - problem solving,
    - state space,
    - state change,
    - structure of state space,
    - problem solution,
    - problem description;
    - Examples of problem definition.

Problem Solving, Search and Control Strategies

- **Problem definitions:**

A *problem* is defined by its *elements* and their *relations*.

To provide a formal description of a problem, we need to do following:

a. Define a *state space* that contains all the possible configurations of the relevant objects, including some impossible ones.
b. Specify one or more states, that describe possible situations, from which the problem-solving process may start. These states are called *initial states*.
c. Specify one or more states that would be acceptable solution to the problem. These states are called *goal states*.
d. Specify a set of *rules* that describe the *actions* (*operators*) available.

The problem can then be solved by using the *rules*, in combination with an appropriate *control strategy*, to move through the *problem space* until a *path* from an *initial state* to a *goal state* is found.

Problem Solving, Search and Control Strategies

- Problem definitions:

  This process is known as ***search***.

  - Search is fundamental to the problem-solving process.
  - Search is a general mechanism that can be used when more direct method is not known.
  - Search provides the framework into which more direct methods for solving subparts of a problem can be embedded.

  A very large number of AI problems are formulated as search problems.

Problem Solving, Search and Control Strategies

Problem Space

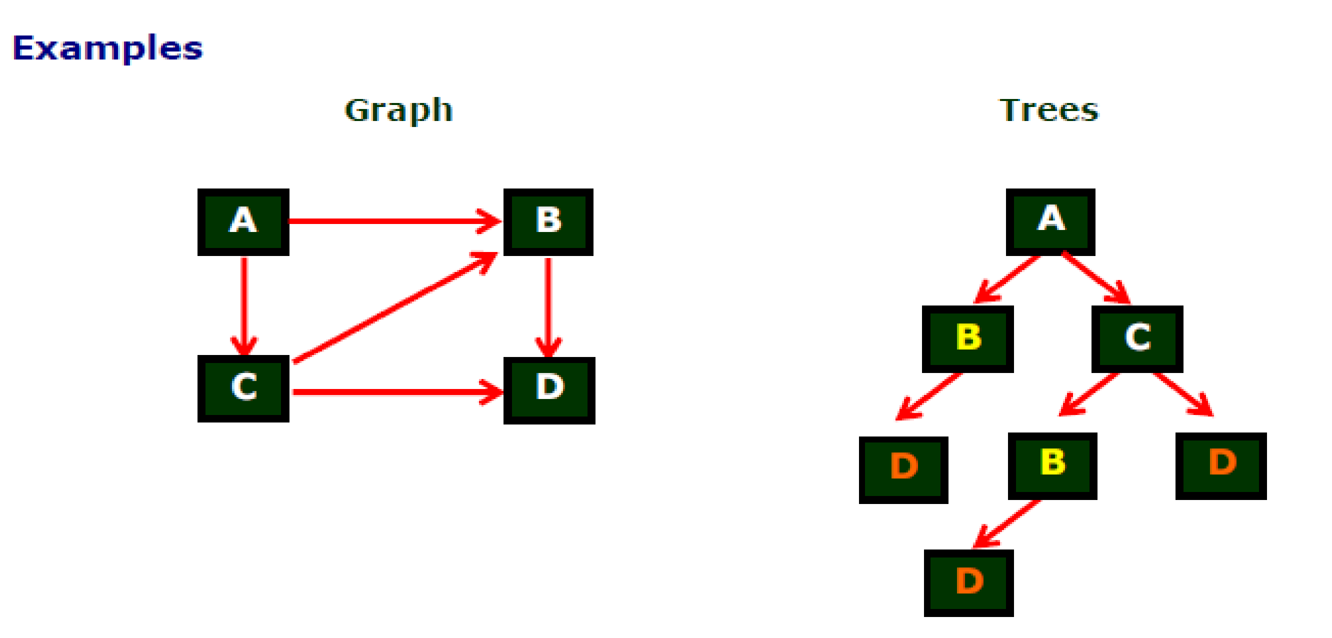
- A *problem space* is represented by directed *graph*, where *nodes* represent *search state* and *paths* represent the *operators* applied to change the *state*.

- To simplify a search algorithms, it is often convenient to logically and programmatically represent a problem space as a ***tree***.
- A tree usually decreases the complexity of a search at a ***cost***. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph; e.g., node **B** and node **D** shown in example below.

Problem Solving, Search and Control Strategies

Problem Space

A **tree** is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph with no cycles is a tree.



Problem Solving, Search and Control Strategies

- ***States***

A state is a representation of elements at a given moment. A problem is defined by its elements and their relations. At each instant of a problem, the elements have specific descriptors and relations; the descriptors tell - how to select elements ?
Among all possible states, there are two special states called :

- *Initial state* *is the start point*
- *Final state* *is the goal state*

Problem Solving, Search and Control Strategies

- **State Change:** Successor Function

A *Successor Function* is needed for state change.
The successor function moves one state to another state.

Successor Function :

- ◇ Is a description of possible actions; a set of operators.
- ◇ Is a transformation function on a state representation, which converts that state into another state.
- ◇ Defines a relation of accessibility among states.
- ◇ Represents the conditions of applicability of a state and corresponding transformation function

Problem Solving, Search and Control Strategies

- **State Space**

A *State space* is the set of all states reachable from the *initial state*. Definitions of terms :

◇ A *state space* forms a *graph* (or map) in which the *nodes* are states and the *arcs* between nodes are actions.

◇ In *state space*, a *path* is a sequence of states connected by a sequence of actions.

◇ The *solution* of a problem is part of the map formed by the *state space*.

Problem Solving, Search and Control Strategies

- ## **Structure of a State Space**

The *Structures* of *state space* are *trees* and *graphs*.

- Tree is a hierarchical structure in a graphical form; and
- Graph is a non-hierarchical structure.

◇ **Tree** has only one path to a given node;
i.e., a *tree* has one and only one path from any point to any other point.

◇ **Graph** consists of a set of nodes (vertices) and a set of edges (arcs).
Arcs establish relationships (connections) between the nodes; i.e., a graph has several paths to a given node.

◇ **operators** are directed *arcs* between nodes.

Search process explores the *state space*. In the worst case, the search explores all possible *paths* between the *initial state* and the *goal state*.

Problem Solving, Search and Control Strategies

- **Problem Solution**

  In the *state space*, a *solution is a path* from the *initial state* to a *goal state* or sometime just a *goal state*.

  ◇ A Solution cost function assigns a numeric cost to each path; It also gives the cost of applying the operators to the states.

  ◇ A Solution quality is measured by the path cost function; and An optimal solution has the lowest path cost among all solutions.

  ◇ The solution may be any or optimal or all.

  ◇ The importance of cost depends on the problem and the type of solution asked.

Problem Solving, Search and Control Strategies

1. Examples of Problem Definitions

- **Example 1 :**

  A game of 8–Puzzle

  ◇ State space : configuration of **8 - tiles** on the board

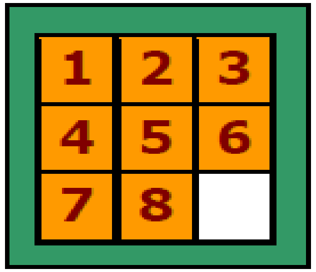
  ◇ Initial state : any configuration

  ◇ Goal state : tiles in a specific order

  ◇ Action : "blank moves"

  - Condition: the move is within the board
  - Transformation: blank moves Left, Right, Up, Dn

  ◇ Solution : optimal sequence of operators



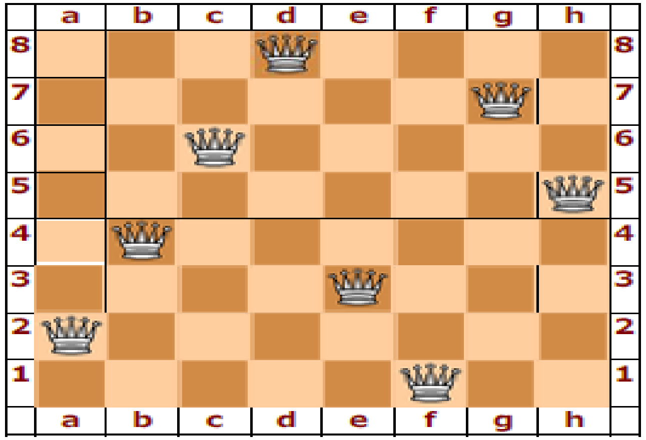
Solution

Problem Solving, Search and Control Strategies

- **Example 2 :**

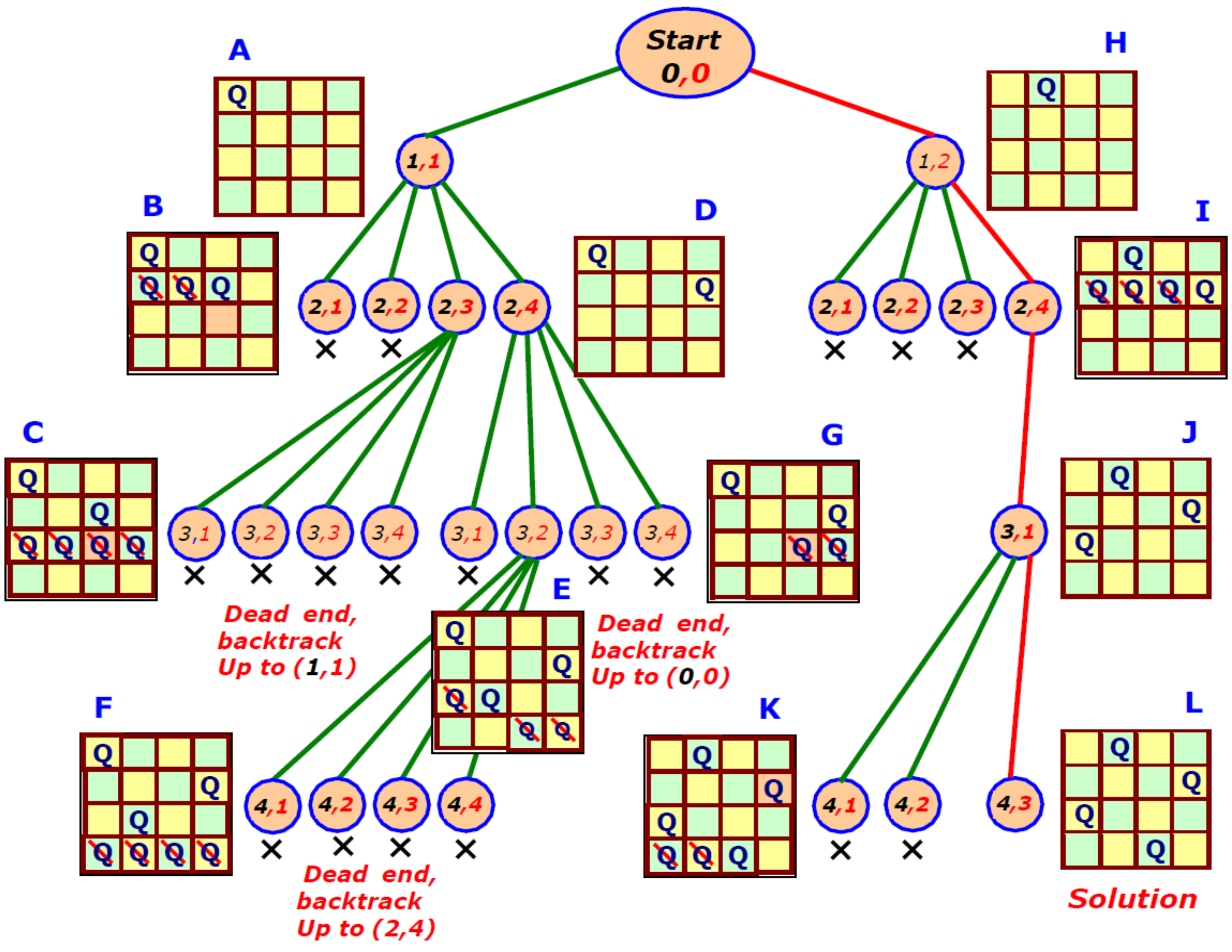
A game of n - queens puzzle; n = 8

- State space : configurations **n = 8** queens on the board with only one queen per row and column
- Initial state : configuration without queens on the board
- Goal state : configuration with **n = 8** queens such that no queen attacks any other
- Operators or actions : place a queen on the board.
  - Condition: the new queen is not attacked by any other already placed
  - Transformation: place a new queen in a particular square of the board
- Solution : one solution (cost is not considered)



One Solution

Example : Backtracking to solve **N = 4** Queens problem.



Hierarchical Representation of Search Algorithms

A representation of most search algorithms is illustrated below. It begins with two types of search - Uninformed and Informed.

Uninformed Search : Also called *blind, exhaustive or brute-force* search, uses no information about the problem to guide the search and therefore may not be very efficient.

Informed Search : Also called *heuristic* or *intelligent* search, uses information about the problem to guide the search, usually guesses the distance to a goal state and therefore efficient, but the search may not be always possible.

always possible.

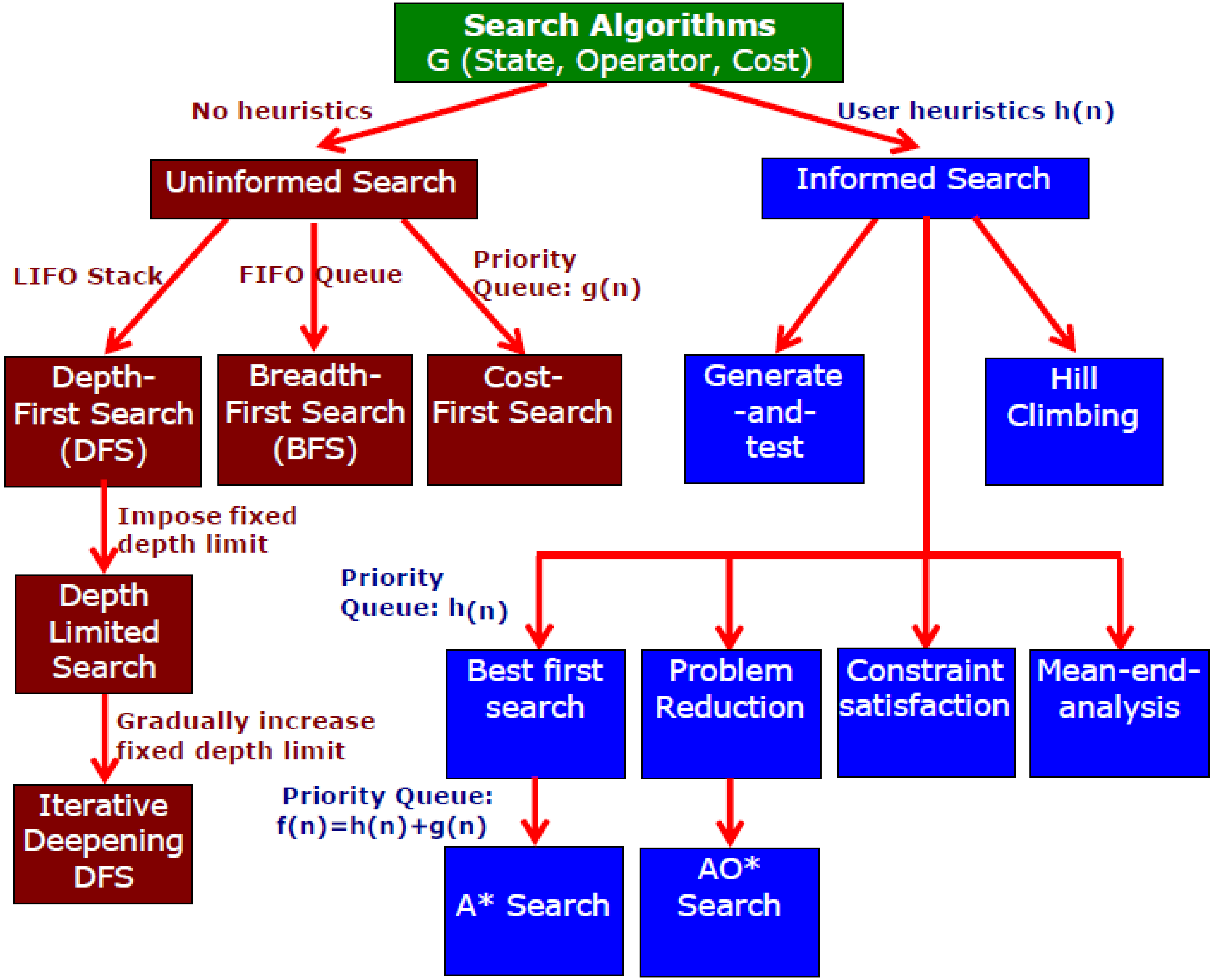


Fig. Different Search Algorithms

Depth-First Search (DFS)

Here explained the Depth-first search tree, the backtracking to the previous level, and the Depth-first search algorithm

- DFS explores a path all the way to a leaf before backtracking and exploring another path.
- **Example:** Depth-first search tree

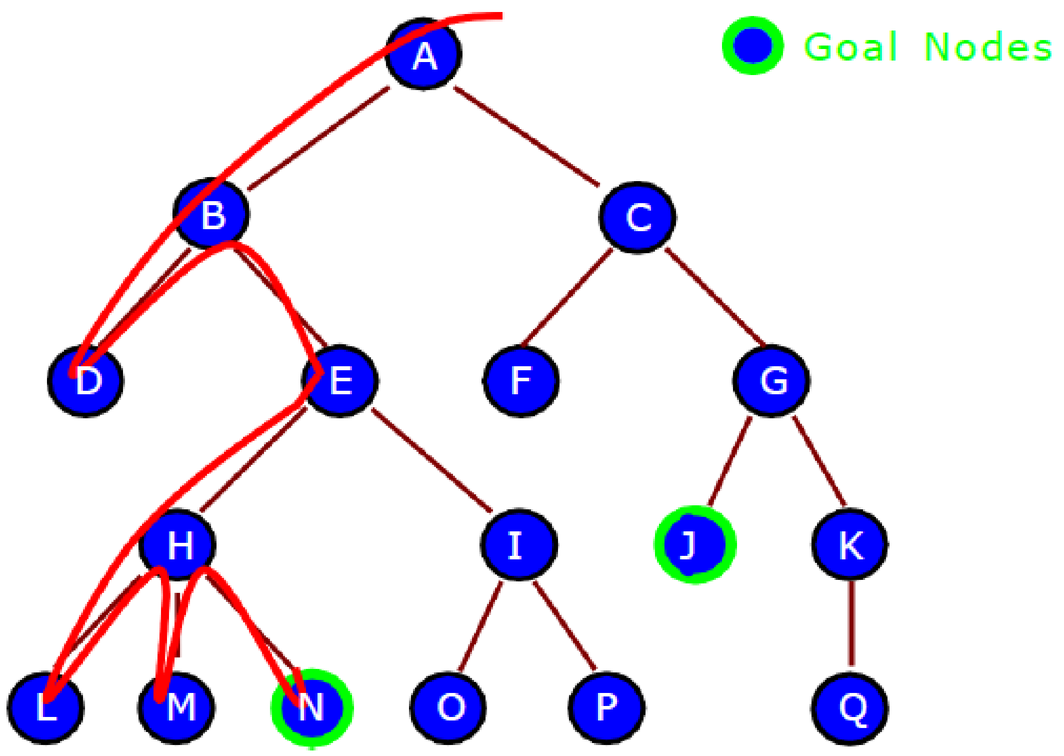


Fig. Depth-first search (DFS)

A B D E H L M N I O P C F G J K Q

- After searching node **A**, then **B**, then **D**, the search *backtracks* and tries another path from node **B .**

- The goal node **N** will be found before the goal node **J**.

◇ **Algorithm –** Depth-first search

- Put the root node on a stack;
  while (stack is not empty)
  { remove a node from the stack;
  if (node is a goal node) return success;
  put all children of node onto the stack; }
  return failure;

Note :

‡ At every step, the stack contains some nodes from each level.

‡ The stack size required depends on the branching factor **b**.

‡ Searching level **n**, the stack contains approximately **b * n** nodes.

‡ When this method succeeds, it does not give the path.

‡ To hold the search path the algorithm required is "*Recursive depth-first search*" and stack size large.