[chapter 2: Problem Solving]
Artificial Intelligence (CSC 355)

Arjun Singh Saud

Central Department of Computer Science & Information Technology
Tribhuvan University

Problem Solving:

Problem solving, particularly in artificial intelligence, may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. Problem-solving methods divide into special purpose and general purpose. A special-purpose method is tailor-made for a particular problem and often exploits very specific features of the situation in which the problem is embedded. In contrast, a general-purpose method is applicable to a wide variety of problems. One general-purpose technique used in AI is means-end analysis—a step-by-step, or incremental, reduction of the difference between the current state and the final goal.

Four general steps in problem solving:

- Goal formulation
  - What are the successful world states
- Problem formulation
  - What actions and states to consider given the goal
- Search
  - Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
- Execute
  - Give the solution perform the actions.

Problem formulation:

A problem is defined by:

- An initial state: State from which agent start
- Successor function: Description of possible actions available to the agent.
- Goal test: Determine whether the given state is goal state or not
- Path cost: Sum of cost of each path from initial state to the given state.

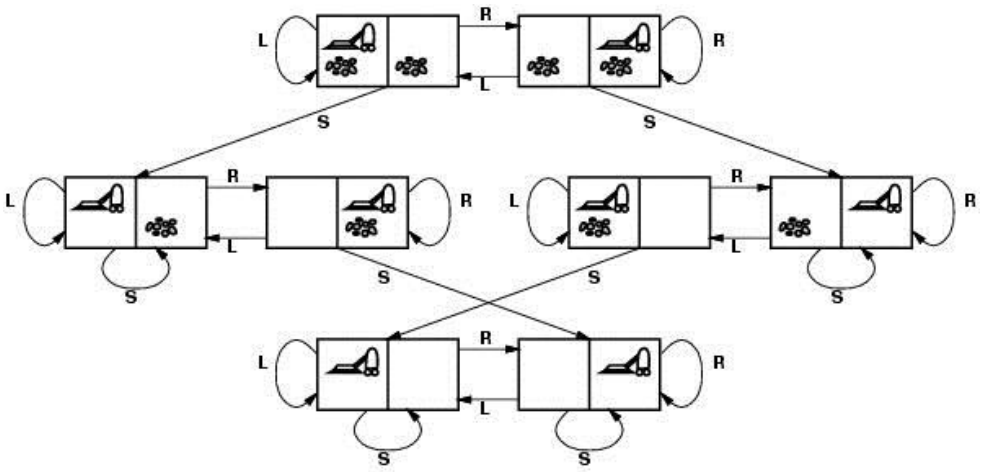
A solution is a sequence of actions from initial to goal state. Optimal solution has the lowest path cost.

State Space representation

The state space is commonly defined as a directed graph in which each node is a state and each arc represents the application of an operator transforming a state to a successor state.

A **solution** is a path from the initial state to a goal state.

State Space representation of Vacuum World Problem:

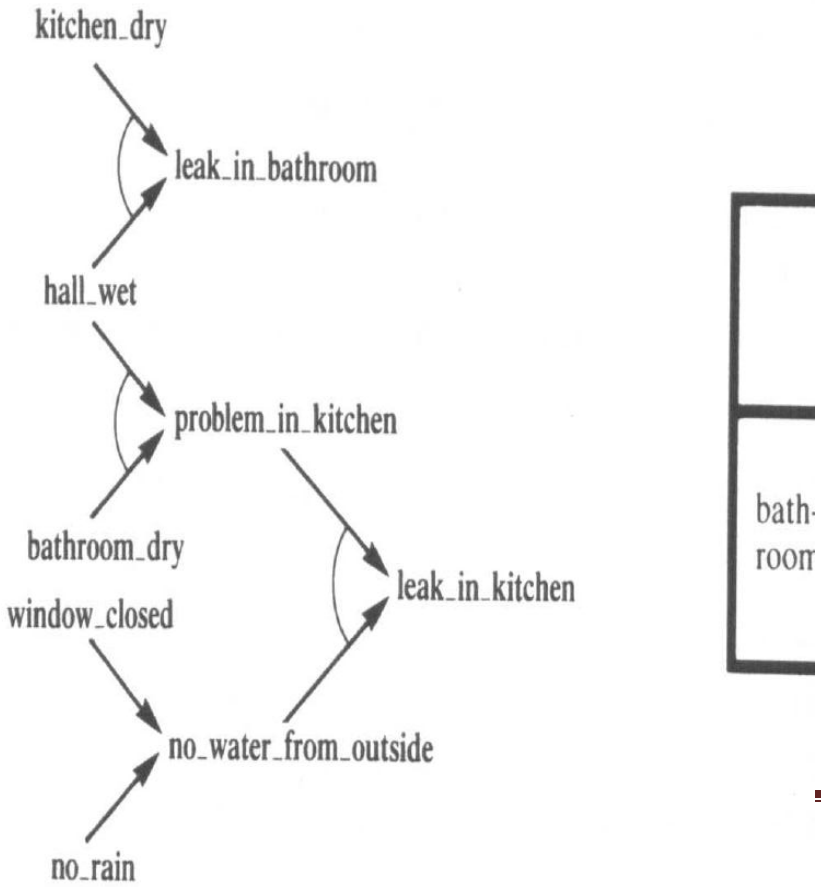


States?? two locations with or without dirt: 2 x 2^2=8 states.
Initial state?? Any state can be initial
Actions?? {*Left, Right, Suck*}
Goal test?? Check whether squares are clean.
Path cost?? Number of actions to reach goal.

For following topics refer Russell and Norvig's Chapter 3 from pages 87-96.

Problem Types: Toy Problems & Real World Problems
Well Defined Problems

Water Leakage Problem:



If
hall _wet and kitchen_dry
then
leak_in_bathroom
If
hall_wet and bathroom_dry
then
problem_in_kitchen
If
window_closed or no_rain
then
no_water_from_outside

Production System:

A **production system** (or **production rule system**) is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, termed **productions**, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system.

Productions consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN"). If a production's precondition matches the current state of the world, then the production is said to be *triggered*. If a production's action is executed, it is said to have *fired*. A production system also contains a database, sometimes called worcking memory, which maintains data about current state or knowledge, and a rule interpreter. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered.

The underlying idea of production systems is to represent knowledge in the form of condition-action pairs called production rules:

If the condition C is satisfied then the action A is appropriate.

Types of production rules

Situation-action rules
If it is raining then open the umbrella.
Inference rules
If Cesar is a man then Cesar is a person

Production system is also called ***ruled-based system***

Architecture of Production System:

Short Term Memory:

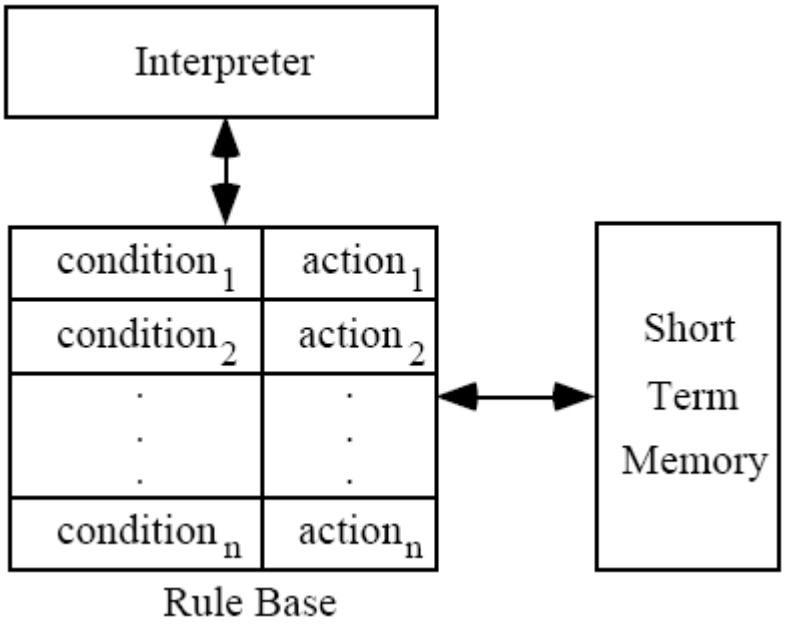
- Contains the description of the current state.

Set of Production Rules:

- Set of condition-action pairs and defines a single chunk of problem solving knowledge.

Interpreter:

- A mechanism to examine the short term memory and to determine which rules to fire (According to some strategies such as DFS, BFS, Priority, first-encounter etc)



The execution of a production system can be defined as a series of recognize-act cycles: ***Match*** **–**memory contain matched against condition of production rules**,** this produces a subset of production called **conflict set**. ***Conflict resolution*** **–**one of the production in the conflict set is then selected***, Apply the rule.***

Consider an example:

Problem: Sorting a string composed of letters a, b & c.
Short Term Memory: cbaca
Production Set:

1. ba → ab
2. ca → ac.
3. cb → bc

Interpreter: Choose one rule according to some strategy.

Iteration #	Memory	Conflict Set	Rule fired
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	ø	halt

Production System: The water jug problem

Problem:

There are two jugs, a 4-gallon one and a 3-gallon one. Neither jug has any measuring markers on it. There is a pump that can be used to fill the jugs with water.
How can you get exactly n (0, 1, 2, 3, 4) gallons of water into one of the two jugs ?

Solution Paradigm:

- build a simple production system for solving this problem.
- represent the problem by using the state space paradigm.

State = (x, y); where: x represents the number of gallons in the 4-gallon jug; y represents the number of gallons in the 3-gallon jug. x $\varepsilon\{0, 1, 2, 3, 4\}$ and y $\varepsilon\{0, 1, 2, 3\}$.

The initial state represents the initial content of the two jugs.

For instance, it may be (2, 3), meaning that the 4-gallon jug contains 2 gallons of water and the 3-gallon jug contains three gallons of water.

The goal state is the desired content of the two jugs.

The left hand side of a production rule indicates the state in which the rule is applicable and the right hand side indicates the state resulting after the application of the rule.

For instance;

(x, y) such that $x < 4 \rightarrow (4, y)$ represents the production

If the 4-gallon jug is not full then fill it from the pump.

The rule base contains the following production rules:

1. (x, y) such that $x < 4 \rightarrow (4, y)$; Fill the 4-gallon jug from pump
2. (x, y) such that $y < 3 \rightarrow (x, 3)$; Fill the 3-gallon jug from pump
3. (x, y) such that $x > 0 \rightarrow (0, y)$; Empty the 4-gallon jug on the ground
4. (x, y) such that $y > 0 \rightarrow (x, 0)$; Empty the 3-gallon jug on the ground
5. (x, y) such that $x + y \geq 4, x < 4, y > 0 \rightarrow (4, y - (4 - x))$
   ; Completely fill the 4-gallon jug from the 3-gallon jug
6. (x, y) such that $x + y \geq 3, x > 0, y < 3 \rightarrow (x - (3 - y), 3)$
   ; Completely fill the 3-gallon jug from the 4-gallon jug
7. (x, y) such that $x + y \leq 4, y > 0 \rightarrow (x+y, 0)$
   ; Empty the 3-gallon jug into the 4-gallon jug
8. (x, y) such that $x + y \leq 3, x > 0 \rightarrow (0, x + y)$
   ; Empty the 4-gallon jug into the 3-gallon jug

The short term memory contains the current state (x, y).

Let us consider the initial situation (0, 0) and the goal situation (n, 2)

short term memory : (0, 0)

1. Match: 1, 2. 2. Conflict resolution: select rule 2. 3. Apply the rule

short term memory becomes (0, 3)

1. Match: 1, 4, 7 2. Conflict resolution: select rule 7 3. Apply the rule

short term memory becomes (3, 0)

1. Match:1, 2, 3, 6, 8 2. Conflict resolution: select rule 2 3. Apply the rule

short term memory becomes (3, 3)

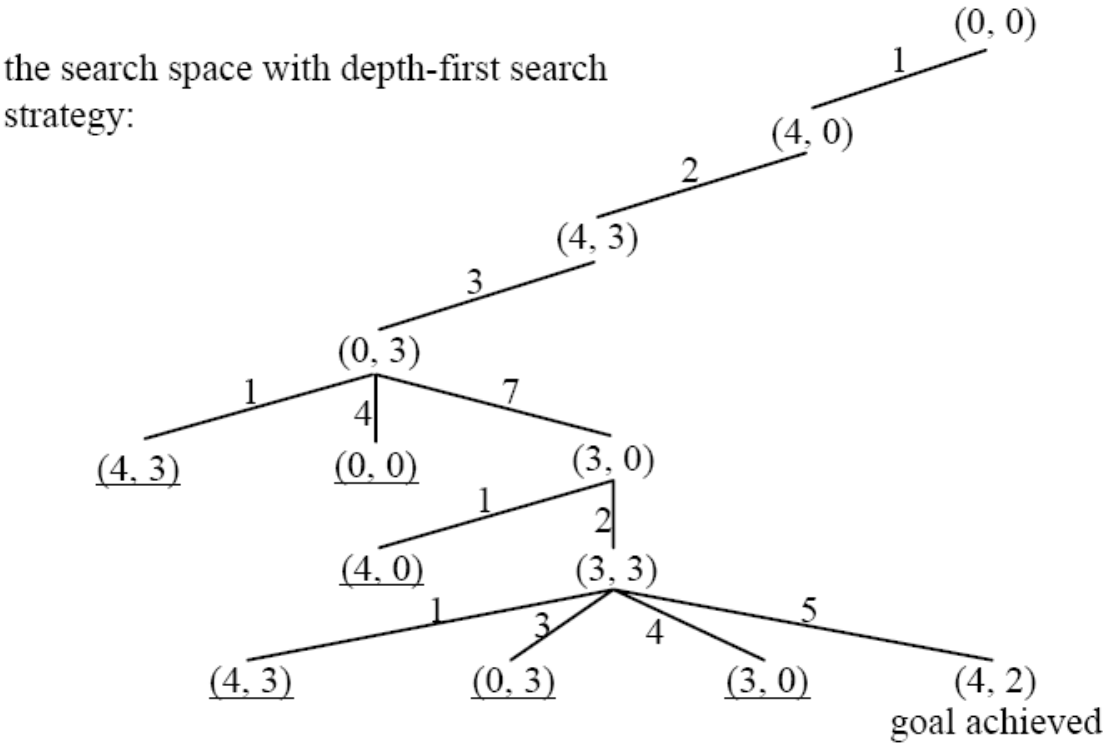
1. Match: 1, 3, 4, 5 2. Conflict resolution: select rule 5 3. Apply the rule

short term memory becomes (4, 2) Goal achieved

The sequence of the applied rules:

Fill the 3-gallon jug from pump
Empty the 3-gallon jug into the 4-gallon jug
Fill the 3-gallon jug from pump
Fill the 4-gallon jug from the 3-gallon jug

The Water Jug Problem: Representation



Constraint Satisfaction Problem:

A **Constraint Satisfaction Problem** is characterized by:

- a *set of variables* {x1, x2, .., xn},
- for each variable xi a *domain* Di with the possible values for that variable, and
- a set of *constraints*, i.e. relations, that are assumed to hold between the values of the variables. [These relations can be given intentionally, i.e. as a formula, or extensionally, i.e. as a set, or procedurally, i.e. with an appropriate generating or recognizing function.] We will only consider constraints involving one or two variables.

The constraint satisfaction problem is to find, for each i from 1 to n, a value in Di for xi so that all constraints are satisfied. Means that, we must find a value for each of the variables that satisfies all of the constraints.

A CS problem can easily be stated as a sentence in first order logic, of the form:

```
(exist x1)..(exist xn) (D1(x1) & .. Dn(xn) => C1..Cm)
```

A CS problem is usually represented as an undirected graph, called *Constraint Graph* where the nodes are the variables and the edges are the binary constraints. Unary constraints can be disposed of by just redefining the domains to contain only the values that satisfy all the unary constraints. Higher order constraints are represented by hyperarcs. In the following we restrict our attention to the case of unary and binary constraints.

Formally, a constraint satisfaction problem is defined as a triple $\langle X, D, C \rangle$, where *X* is a set of variables, *D* is a domain of values, and *C* is a set of constraints. Every constraint is in turn a pair $\langle t, R \rangle$, where *t* is a tuple of variables and *R* is a set of tuples of values; all these tuples having the same number of elements; as a result *R* is a relation. An evaluation of the variables is a function from variables to values, $v : X \rightarrow D$. Such an evaluation satisfies a constraint $\langle (x_1, \ldots, x_n), R \rangle$ if $(v(x_1), \ldots, v(x_n)) \in R$. A solution is an evaluation that satisfies all constraints.

Constraints

- A constraint is a relation between a **local** collection of variables.
- The constraint restricts the values that these variables can simultaneously have.
- For example, **all-diff(X1, X2, X3)**. This constraint says that X1, X2, and X3 must take on different values. Say that {1,2,3} is the set of values for each of these variables then:

  X1=1, X2=2, X3=3 OK X1=1, X2=1,X3=3 NO

The constraints are the key component in expressing a problem as a CSP.

- The constraints are determined by how the variables and the set of values are chosen.
- Each constraint consists of;
  1. A set of variables it is over.
  2. A specification of the sets of assignments to those variables that satisfy the constraint.
- The idea is that we break the problem up into a set of distinct conditions each of which have to be satisfied for the problem to be solved.

Example: In N-Queens: Place N queens on an NxN chess board so that queen can attack any other queen.

- No queen can attack any other queen.
- Given any two queens Qi and Qj they cannot attack each other.
- Now we translate each of these individual conditions into a separate constraint.
  - Qi cannot attack Qj(i ≠j)
    - Qi is a queen to be placed in column i, Qj is a queen to be placed in column j.

- The value of Qi and Qj are the rows the queens are to be placed in.

- Note the translation is dependent on the representation we chose.
- **Queens can attack each other**,
    1. *Vertically*, if they are in the same column---this is impossible as Qi and Qj are placed in different columns.
    2. *Horizontally*, if they are in the same row---we need the constraint Qi≠Qj.
    3. *Along a diagonal*, they cannot be the same number of columns apart as they are rows apart: we need the constraint |i-j| ≠|Qi-Qj| (| | is absolute value)
- **Representing the Constraints;**
    1. Between every pair of variables (Qi,Qj) (i ≠j), we have a constraint Cij.
    2. For each Cij, an assignment of values to the variables Qi= A and Qj= B, satisfies this constraint if and only if;

A ≠B

| A-B| ≠|i-j|

- **<u>Solutions:</u>**
    - A solution to the N-Queens problem will be any assignment of values to the variables $Q_1,\ldots,Q_N$ that satisfies all of the constraints.
    - Constraints can be over any collection of variables. In N-Queens we only need binary constraints---constraints over pairs of variables.

More Examples: Map Coloring Problem (Discussed in class)

Refer Russell and Norvig's Chapter 5 from pages 165-169. Also have a brief look on page 172-173 for forward checking that we have discussed in class.