1.8 ADVERSARIAL SEARCH

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**. ○ Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI:

	Deterministic	**Chance Moves**
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent. o One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as ply. o Chess and tic-tac-toe are examples of a Zero-sum game.

Zero-sum game: Embedded thinking

The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:

- What to do.
- How to decide the move

- Needs to think about his opponent as well ○ The opponent also thinks what to do

Each of the players is trying to find out the response of his opponent to their actions. This requires embedded thinking or backward reasoning to solve the game problems in AI.

Formalization of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space. ○ **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

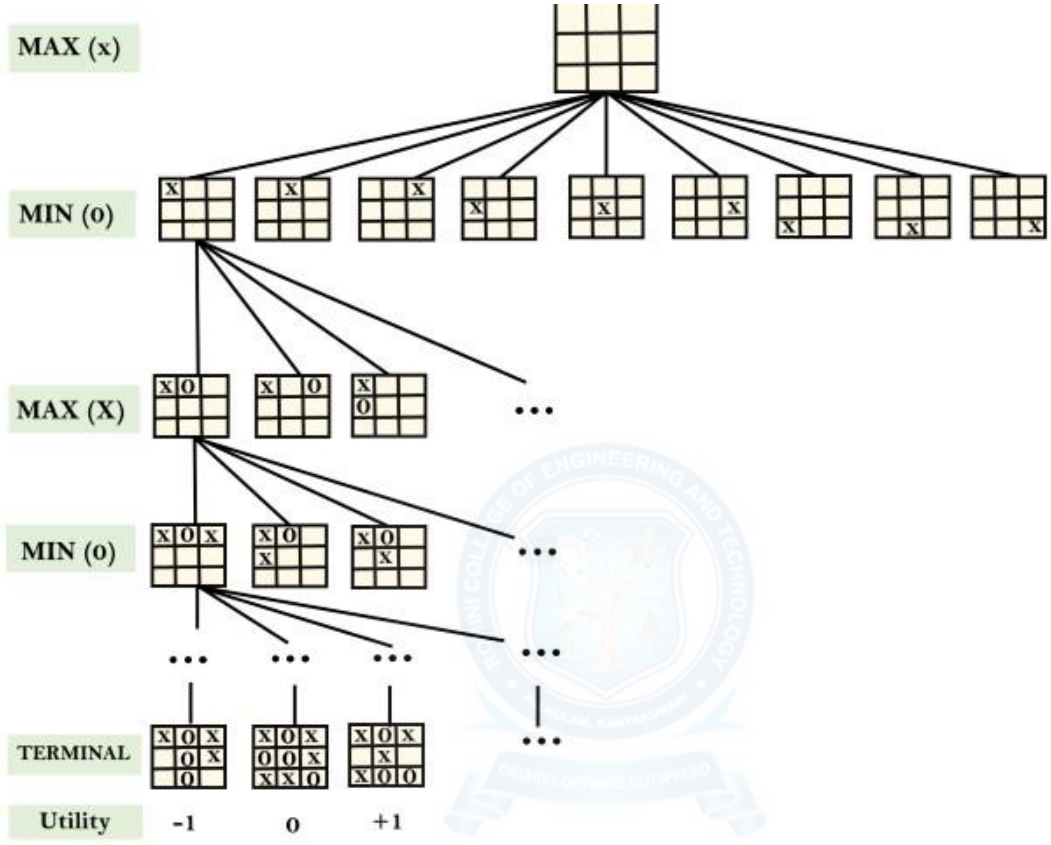
Game tree:

A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, actions function, and result Function.

Example: Tic-Tac-Toe game tree:

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

- There are two players MAX and MIN. ○ Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree ○ MIN minimizes the result.



Example Explanation: ○ From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.

- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.

- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

Hence adversarial Search for the minimax procedure works as follows:

- It aims to find the optimal strategy for MAX to win the game. ○ It follows the approach of Depth-first search. ○ In the game tree, optimal leaf node could appear at any depth of the tree. ○ Propagate the minimax values up to the tree until the terminal node discovered.

In a given game tree, the optimal strategy can be determined from the minimax value of each node, which can be written as MINIMAX(n). MAX prefer to move to a state of maximum value and MIN prefer to move to a state of minimum value then:

For a state S MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s) & \text{If TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If PLAYER}(s) = \text{MIN}. \end{cases}$$

1.9 CONSTRAINT SATISFACTION PROBLEM

The goal is to discover some problem state that satisfies a given set of constraints.

Basic workflow of constraint satisfaction problem

- We need to analyze the problem perfectly
- We need to derive the constraints given in the problem
- Then we need to derive a solution form a given constraint.
- Find whether we have reached the foal state, If we have not reached the goal state, we need to make a guess and that guess has to be added as the new constraint.
- After adding new constraint, again we go for the evaluation of the solution. Now we need to solve the problem by using this added new constraint, again we have to check, whether we have reached the goal state, if yes, solution found

Algorithm : Constraint satisfaction

1. Propagate available constraints. To do this , first set OPEN to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty:
    a. Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB
    b. If this is different from the set that was assigned the last time OB was examined or if it this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
    c. Remove OB from OPEN.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the solution of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:

a. Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
b. Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

Example :

Cryptarithmatic puzzles,

SEND	DONALD	CROSS
+MORE	+GERALD	+ROADS
……	……..	…….
MONEY	ROBERT	DANGER

Algorithm Working :

Consider the crypt arithmetic problem shown in the below fig.

Problem:

SEND

+MORE

……

MONEY

Assign decimal digit to each of the letters in such a way that the answer to the problem is correct to the same letter occurs more than once, it must be assign the same digit each time. No two different letters may be assigned the same digit. Consider the crypt arithmetic problem.

Constraints:-

1. No two digit can be assigned to same letter.

2. Only single digit number can be assign to a letter.
3. No two letters can be assigned same digit.
4. Assumption can be made at various levels such that they do not contradict each other.
5. The problem can be decomposed into secured constraints. A constraint satisfaction approach may be used.
6. Any of search techniques may be used.
7. Backtracking may be performed as applicable us applied search techniques.
8. Rule of arithmetic may be followed.

Initial state of problem.

D=? E=? Y=? N=? R=? O=? S=? M=? C1=? C2=?

C1 ,C 2, C3 stands for the carry variables respectively.

Goal State: the digits to the letters must be assigned in such a manner so that the sum is satisfied.

- The Solution process proceeds in cycles.
- At each cycle, 2 important things are done
    i. Constraints are propagated by using rules that correspond to the properties of arithmetic.
    ii. A value is guessed for some letter whose value is not determined

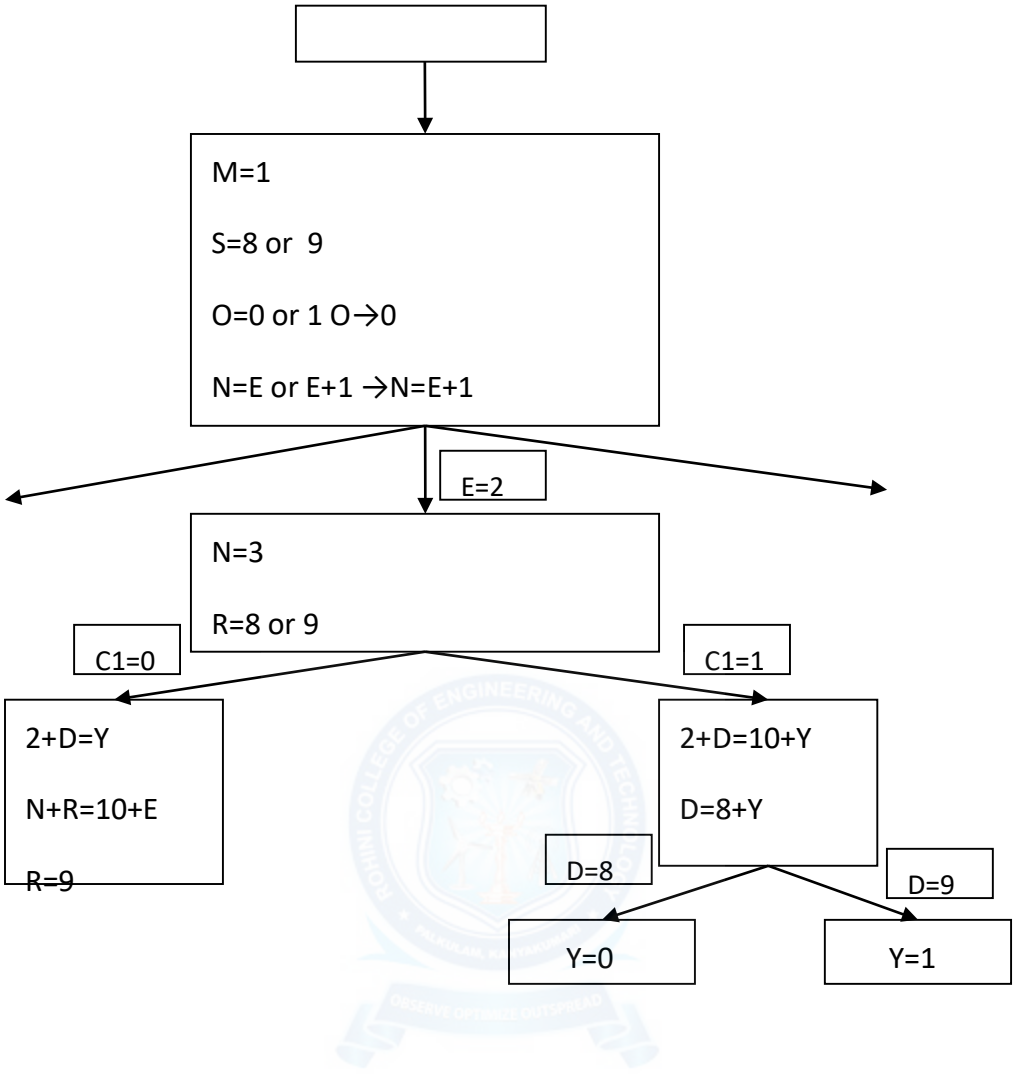


Fig Solving cryptarithmetic problem

C1,C2,C3 and C4 indicate the carry bits outs of the columns, numbering from the right.

Rules propagating constraints generate the following additional constraints:

- M=1, since two single digit numbers plus a carry cannot total more than 19.
- S=8 or 9, since S+M+C3>9 (to generate the carry) and M=1, S+1+C3>9, so S+C3>8 and C3 is atmost 1.
- O=0, since S+M(1)+C3(<=1) must be atleast 10 to generate a carry and it can be at most 11. But M is already 1, so O must be 0.

- N=E or E+1, depending on the value of C2.But N cannot have the same value as E. So N=E+1 and C2 is 1.
- In order to for C2 to be 1, the sum of N+R+C1 must be greater than 8
- N+R cannot be greater than 18, even with a carry in, so E cannot be 9.

Assume that no more constraints can be generated. To progress at this point, guessing happens. If E is assigned the value 2. Now the next cycle begins.

Constraint propagator shows that :

- N=3, since N=E+1
- R=8 or 9, since R+N(3)+C1(1 or 0)=2 or 12. But since N is already 3, the sum of these nonnegative numbers cannot be less than 3. Thus R+3+(0 or 1)=12 and R=8 or 9.
- 2+D=Y or 2+D=10+Y, from the sum in the rightmost column.

Assuming no more constraint generation then guess is required.

Suppose C1 is chosen to guess a value for 1, then we reach dead end as shown in the fig , when this happens , the process will be backtrack and C1=0.

Algorithm for constraint satisfaction in which chronological backtracking is used when guessing leads to an inconsistent set of constraints.

Constraints are generated are left alone if they are independent of the problem and its cause. This approach is called dependency directed backtracking (DDB).