Unit III

Adversarial search Methods-Game playing-Important concepts

- Adversarial search: Search based on Game theory- Agents- Competitive environment
- ***According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.'***
- Such Conflicting goal- adversarial search
- Game playing technique- Those games- Human Intelligence and Logic factor- Excluding other factors like Luck factor
  - Tic-Tac-Toe, Checkers, Chess – Only mind works, no luck works

Adversarial search Methods-Game playing-Important concepts

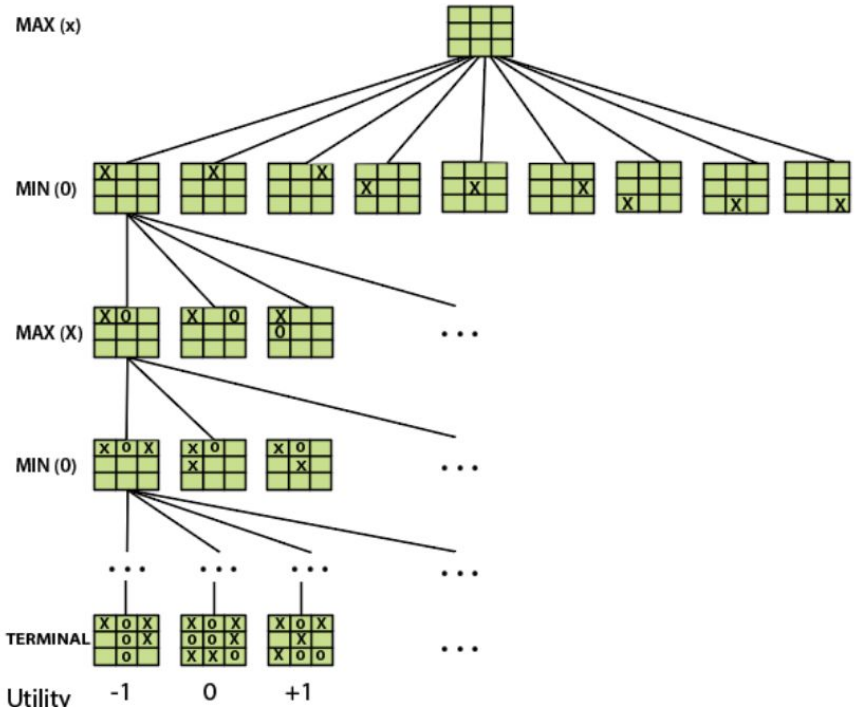
We are opponents- I win, you loose.

- **Techniques required to get the best optimal solution (Choose Algorithms for best optimal solution within limited time)**
  - **Pruning:** A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.
  - **Heuristic Evaluation Function:** It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

Game playing and knowledge structure- Elements of Game Playing search

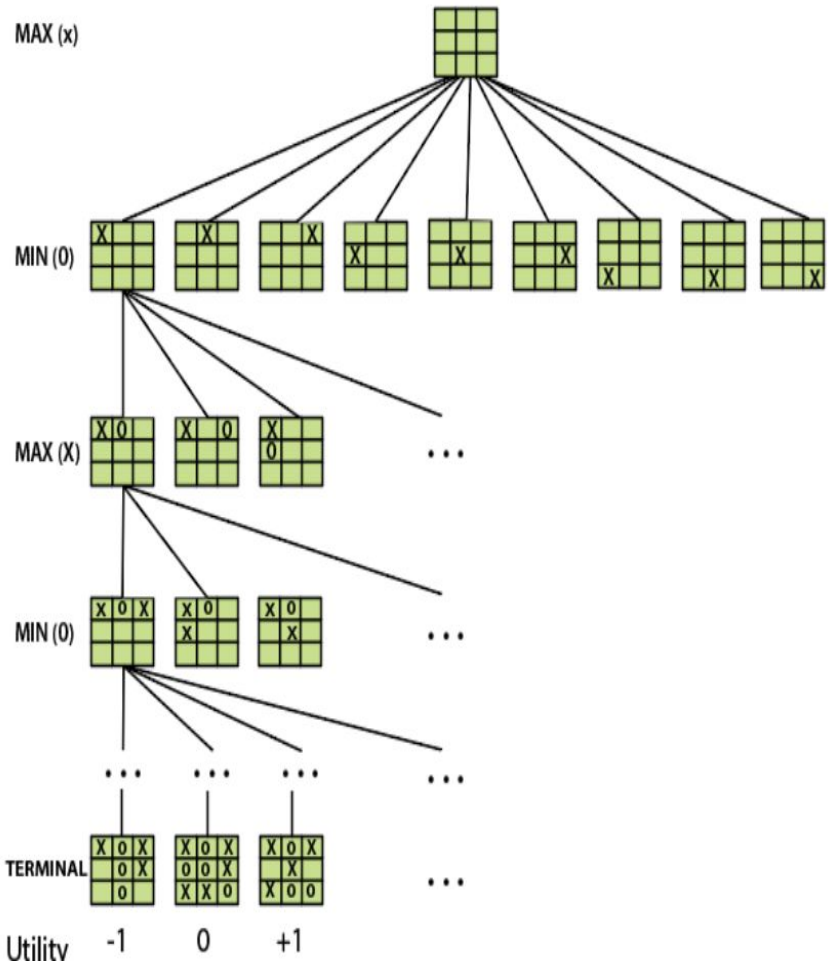
- To play a game, we use a game tree to know all the possible choices and to pick the best one out. There are following elements of a game-playing:
- **S_0:** It is the initial state from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves to be used in a state.
- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended and returns true.
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**. The price which the winner will get i.e.
- **(-1):** If the PLAYER loses.
- **(+1):** If the PLAYER wins.
- **(0):** If there is a draw between the PLAYERS.

- ***For example**, in **chess, tic-tac-toe,** we have two or three possible outcomes. Either to win, to lose, or to draw the match with values **+1,-1 or 0.***
- ***Game Tree for Tic-Tac-Toe***
  - Node: Game states, Edges: Moves taken by players



Game playing and knowledge structure- Elements of Game Playing search

- **INITIAL STATE (S_0):** The top node in the game-tree represents the initial state in the tree and shows all the possible choice to pick out one.
- **PLAYER (s):** There are two players, **MAX and MIN**. **MAX** begins the game by picking one best move and place **X** in the empty square box.
- **ACTIONS (s):** Both the players can make moves in the empty boxes chance by chance.
- **RESULT (s, a):** The moves made by **MIN** and **MAX** will decide the outcome of the game.
- **TERMINAL-TEST(s):** When all the empty boxes will be filled, it will be the terminating state of the game.
- **UTILITY:** At the end, we will get to know who wins: **MAX** or **MIN,** and accordingly, the price will be given to them**.**



Game as a search problem

- **Types of algorithms in Adversarial search**
  - In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game. It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time.
    - **Minmax Algorithm**
    - **Alpha-beta Pruning**

Game Playing vs. Search

- Game vs. search problem
- "Unpredictable" opponent □ specifying a move for every possible opponent reply
- Time limits □ unlikely to find goal, must approximate

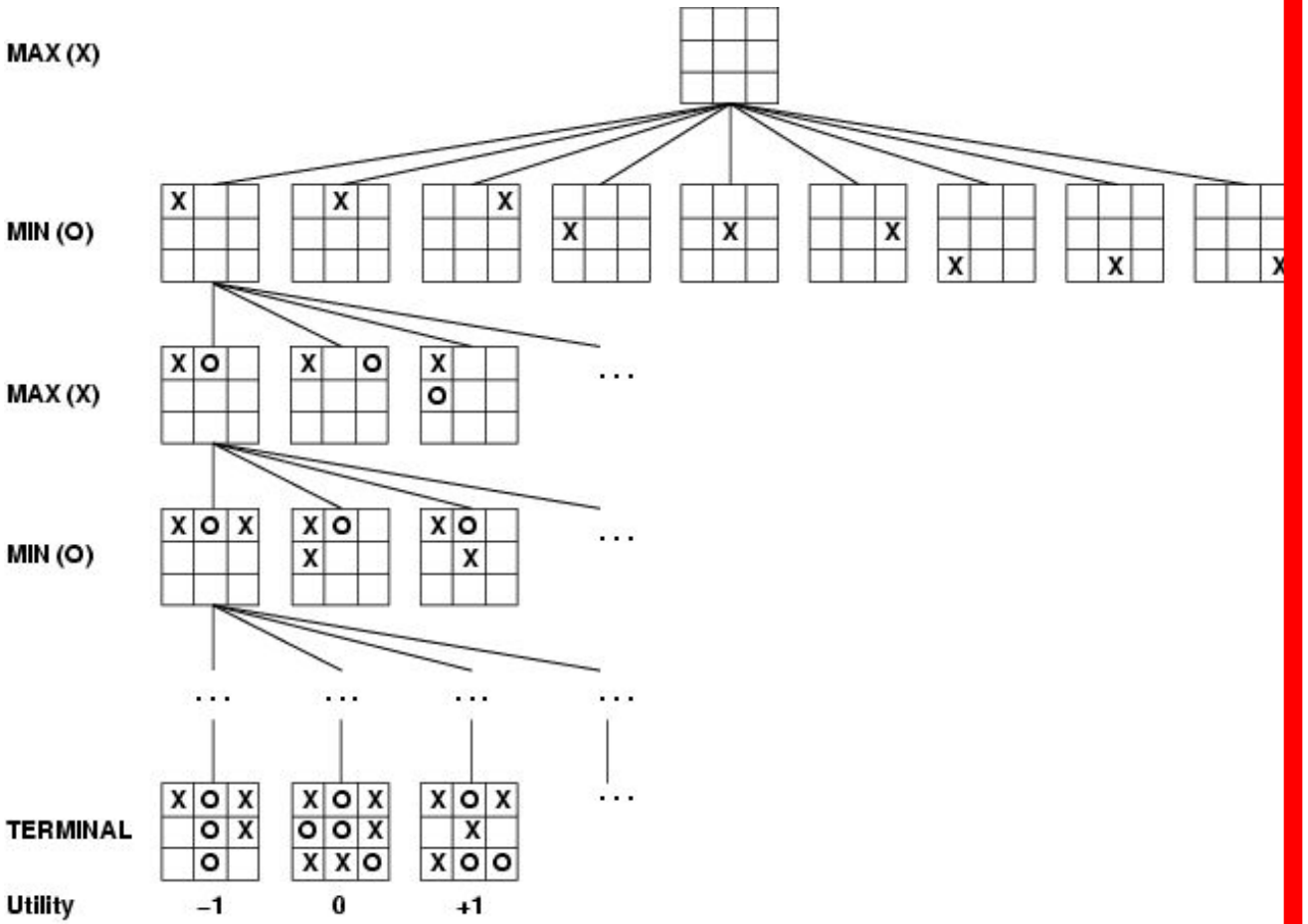
Game Playing

- Formal definition of a game:
  - Initial state
  - Successor function: returns list of *(move, state)* pairs
  - Terminal test: determines when game over

    Terminal states: states where game ends
  - Utility function (objective function or payoff function): gives numeric value for terminal states

We will consider games with 2 players (**Max and Min**);
Max moves first.

Game Tree Example: Tic-Tac-Toe

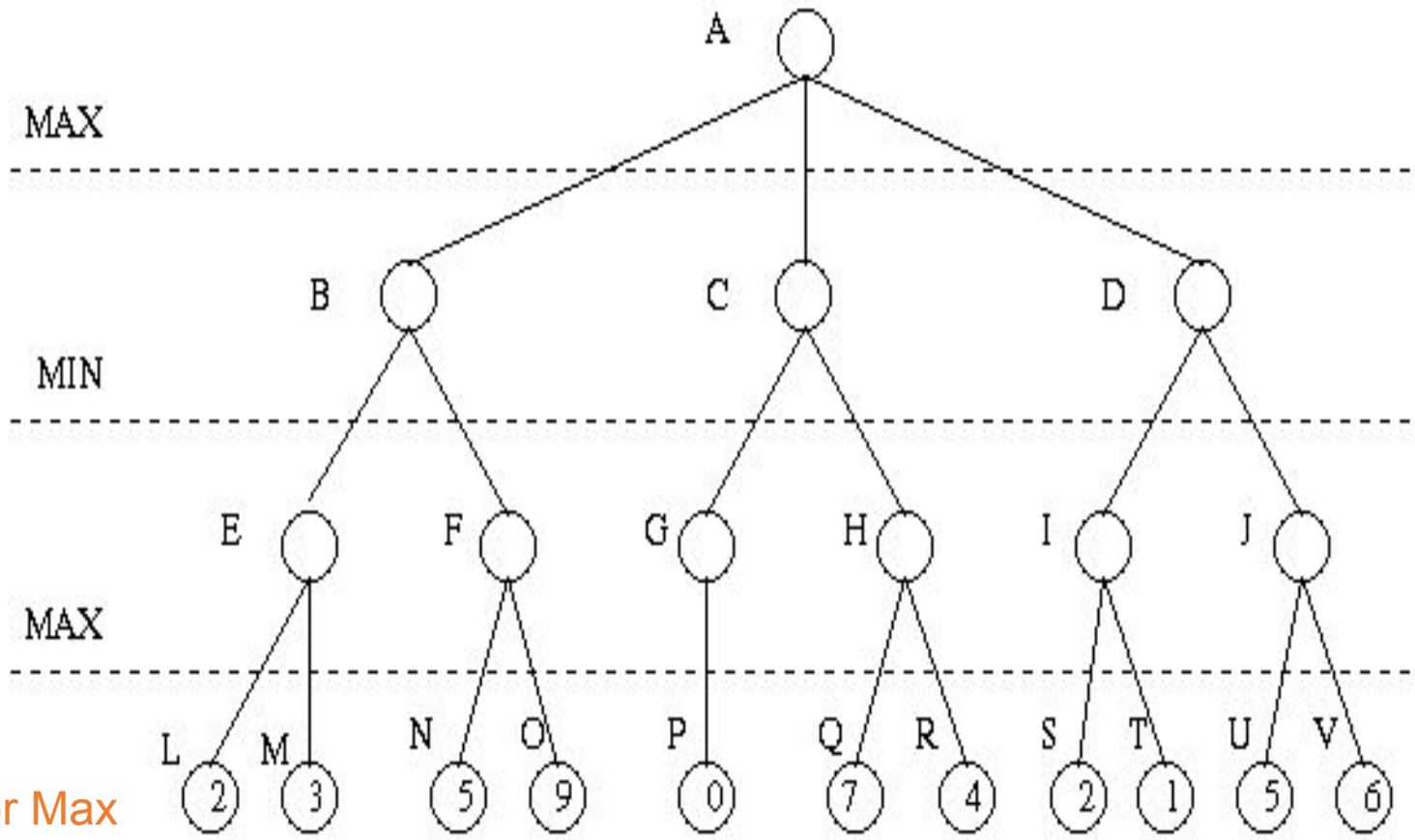


Tree from Max's perspective

Minimax Algorithm

- Minimax algorithm
  - Perfect play for deterministic, 2-player game
  - Max tries to maximize its score
  - Min tries to minimize Max's score (Min)
  - Goal: move to position of highest minimax value
    - Identify best achievable payoff against best play

Minimax Algorithm



Payoff for Max

Minimax Rule

- Goal of game tree search: to determine **one move** for Max player that **maximizes** the **guaranteed payoff** for a given game tree for MAX

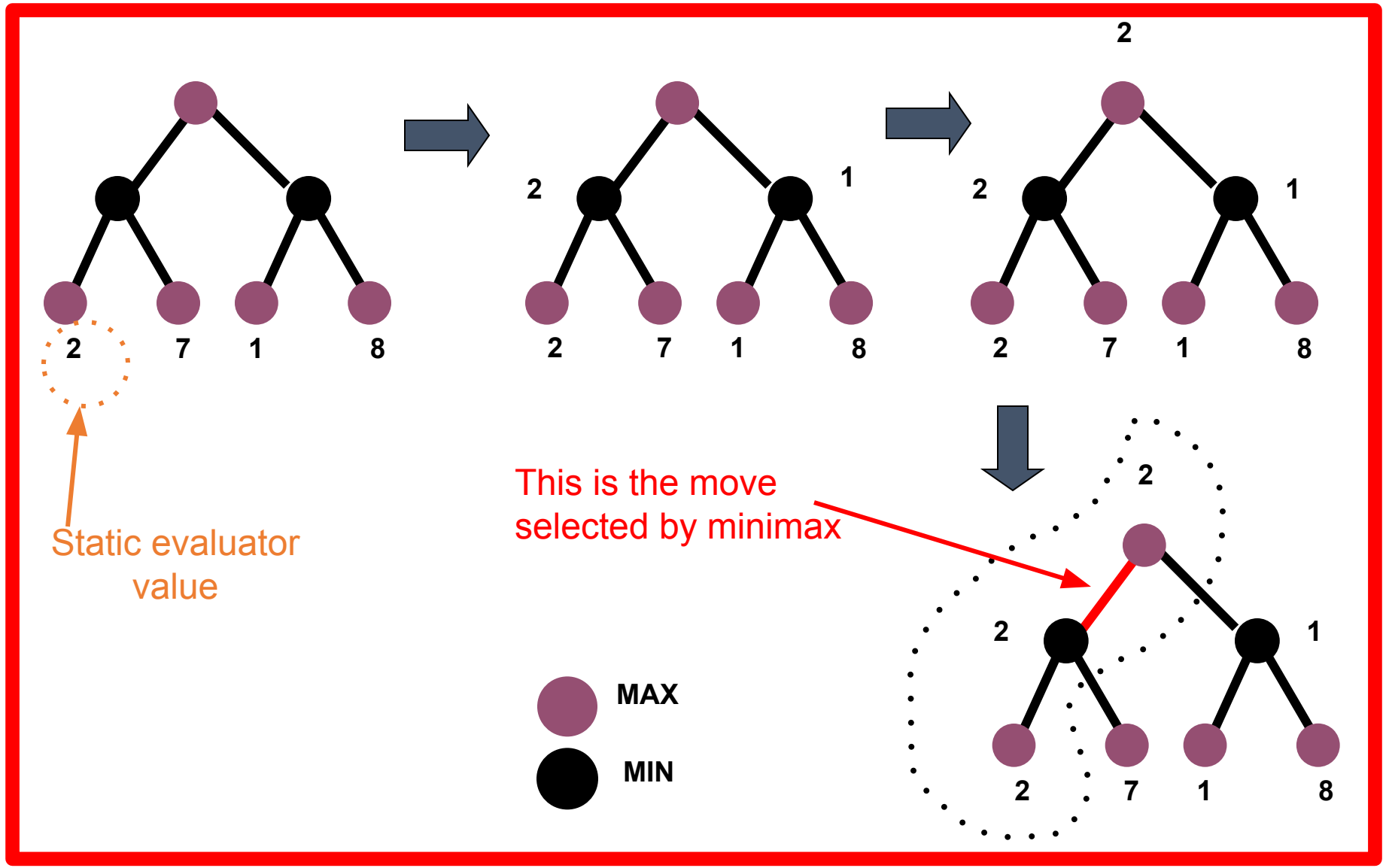
  Regardless of the moves the MIN will take
- The value of each node (Max and MIN) is determined by (back up from) the values of its children
- MAX plays the worst case scenario:

  Always assume MIN to take moves to maximize his pay-off (i.e., to minimize the pay-off of MAX)
- For a MAX node, the backed up value is the **maximum** of the values associated with its children
- For a MIN node, the backed up value is the **minimum** of the values associated with its children

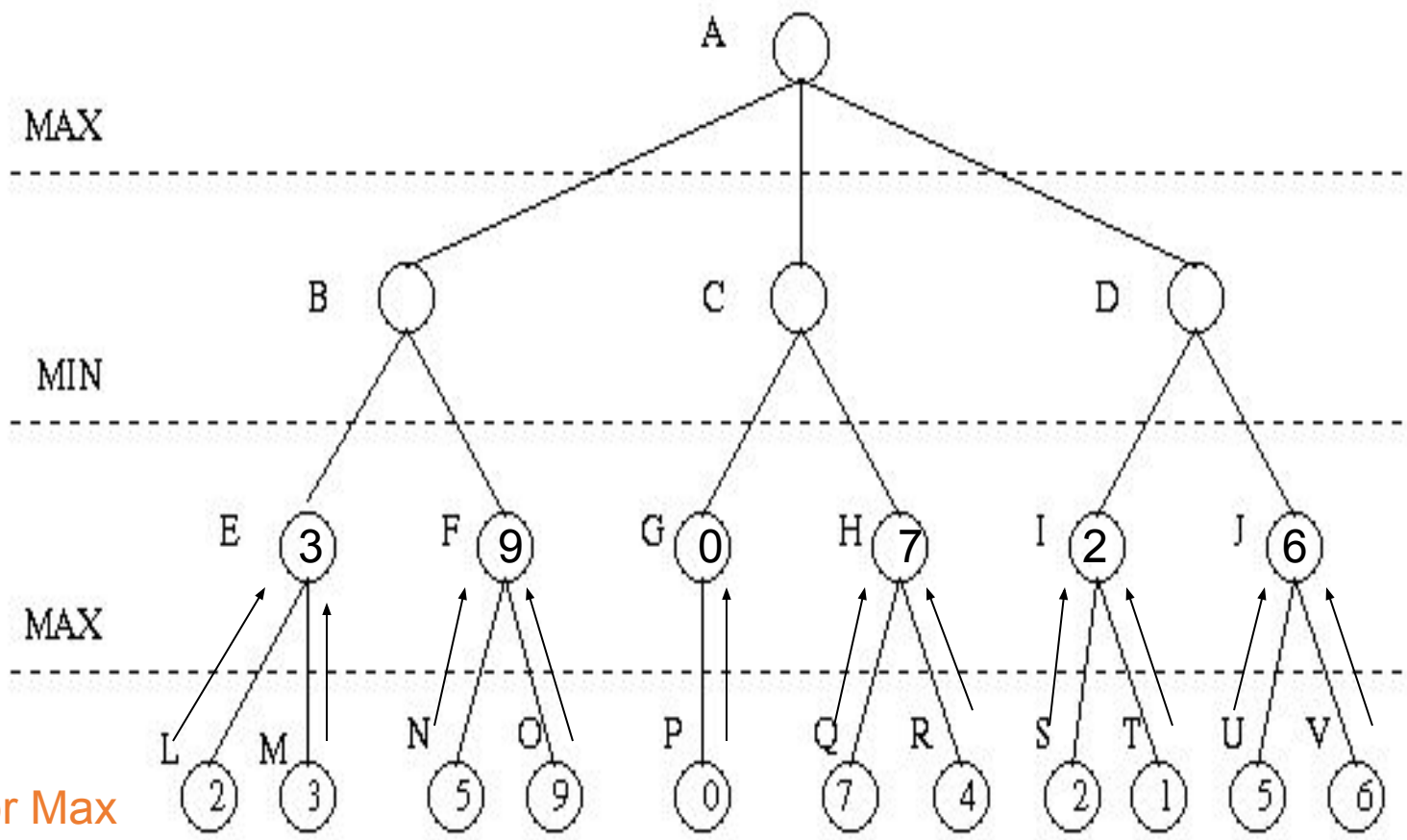
Minimax procedure

1. Create start node as a MAX node with current board configuration
2. Expand nodes down to some depth (i.e., ply) of lookahead in the game.
3. Apply the evaluation function at each of the leaf nodes
4. Obtain the “back up" values for each of the non-leaf nodes from its children by Minimax rule until a value is computed for the root node.
5. Pick the operator associated with the child node whose backed up value determined the value at the root as the move for MAX

Minimax Search

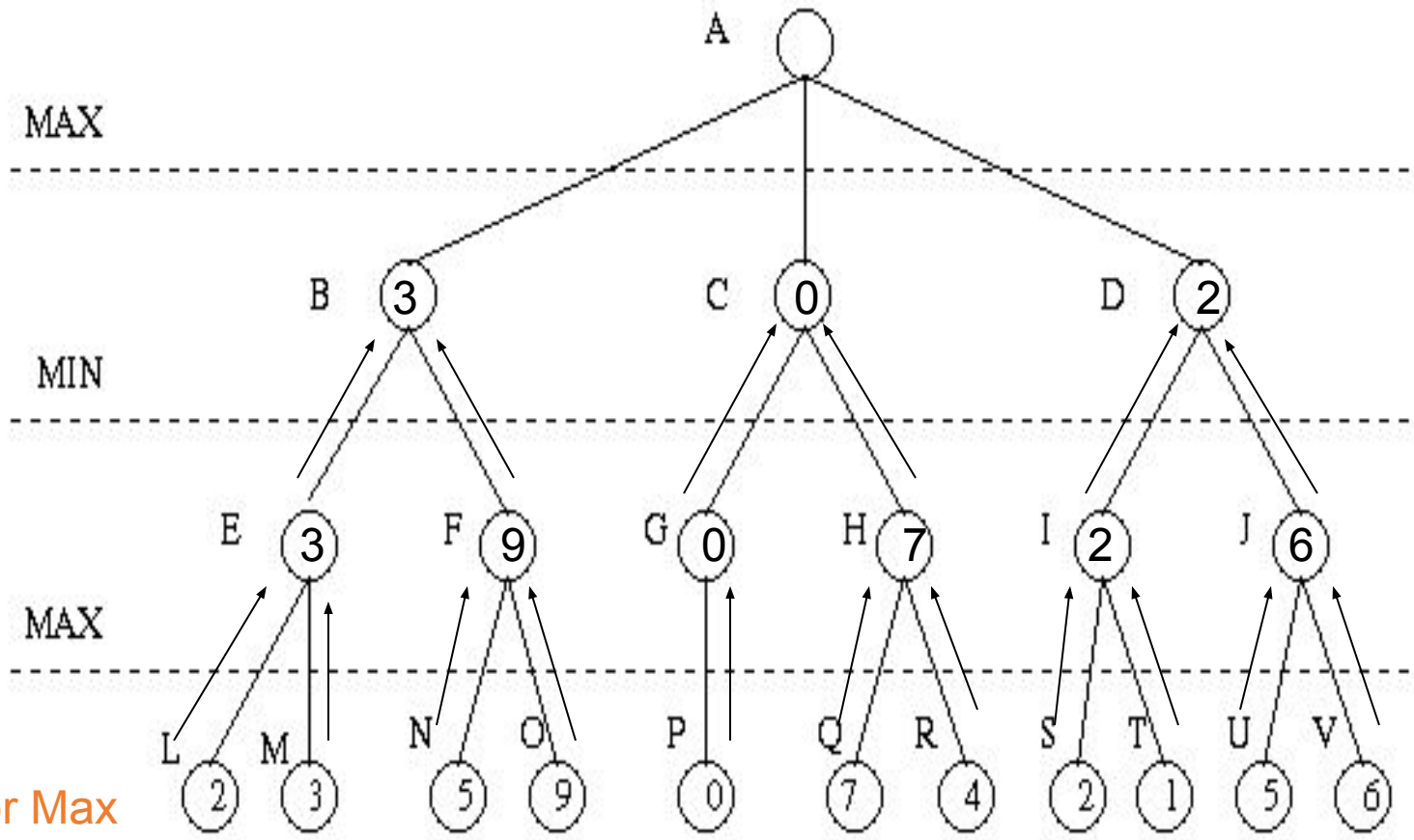


Minimax Algorithm (cont'd)



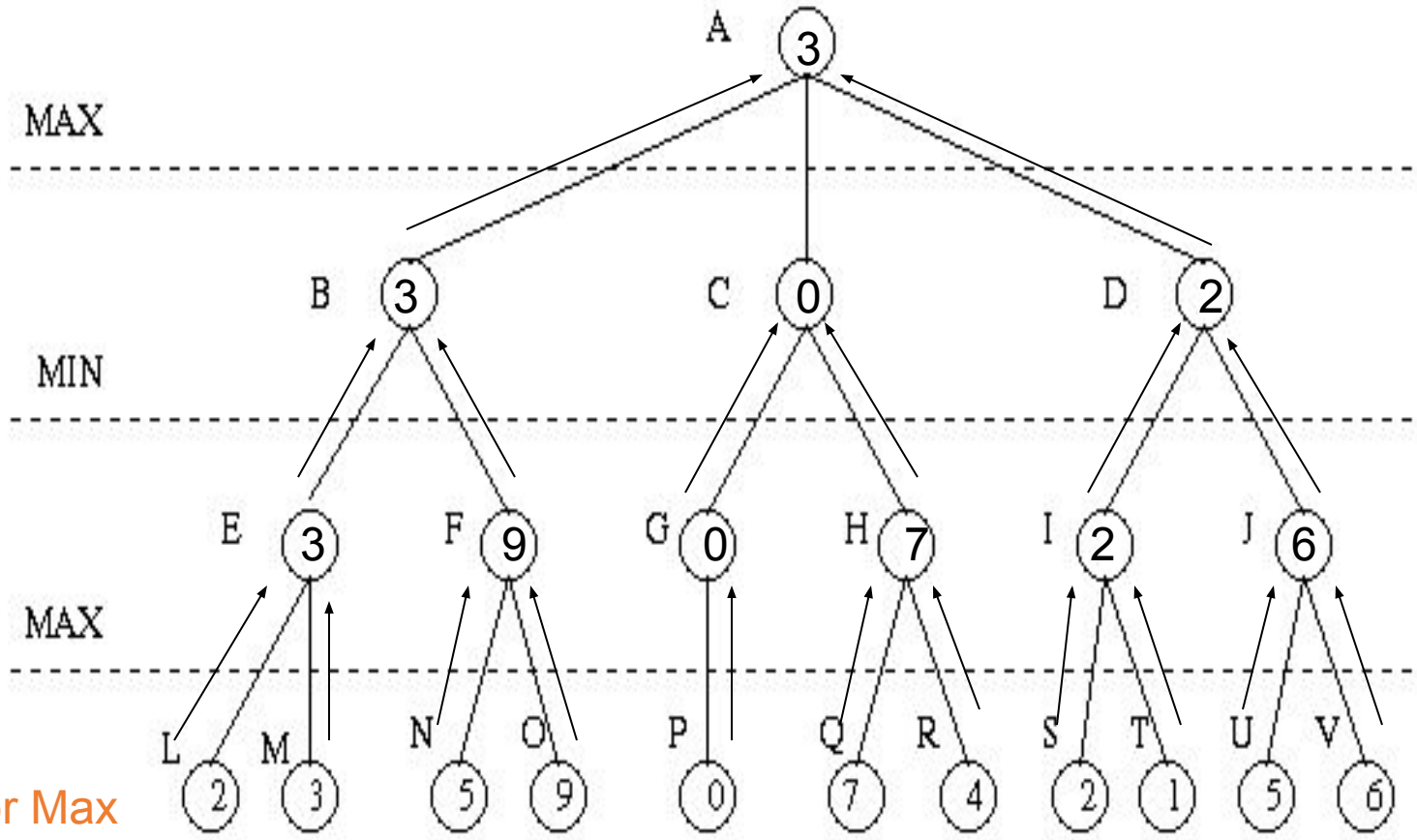
Payoff for Max

Minimax Algorithm (cont'd)



Payoff for Max

Minimax Algorithm (cont'd)



Payoff for Max

Minimax Algorithm (cont'd)

- Properties of minimax algorithm:
- Complete? Yes (if tree is finite)
  Optimal? Yes (against an optimal opponent)
  Time complexity? $O(b^m)$

- Space complexity? O(bm) (depth-first exploration, if it generates all successors at once)

m – maximum depth of tree; b branching factor

m – maximum depth of the tree; b – legal moves;

Minimax Algorithm

- Limitations
  - Not always feasible to traverse entire tree
  - Time limitations
- Key Improvement
  - Use evaluation function instead of utility
    - Evaluation function provides estimate of utility at given position

Unit 2 List of Topics

- Searching techniques – Uninformed search – General search Algorithm
- Uninformed search Methods – Breadth First Search
- Uninformed search Methods – Depth First Search
- Uninformed search Methods – Depth limited Search
- *Uniformed search Methods- Iterative Deepening search*
- *Bi-directional search*
- *Informed search- Generate and test, Best First search*
- *Informed search-A* Algorithm*
- *AO* search*
- *Local search Algorithms-Hill Climbing, Simulated Annealing*
- *Local Beam Search*
- *Genetic Algorithms*
- *Adversarial search Methods-Game playing-Important concepts*
- *Game playing and knowledge structure.*
- *Game as a search problem-Minimax Approach*
- *Minimax Algorithm*
- *Alpha beta pruning*
- *Game theory problems*

Alpha Beta Pruning

•Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

•As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree.

•Since we cannot eliminate the exponent, but we can cut it to half.

•Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**.

•This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.

•Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire

Alpha Beta Pruning

- The two-parameter can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Alpha Beta Pruning

Condition for Alpha-beta pruning:

- The main condition which required for alpha-beta pruning is:
  $\alpha >= \beta$

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Alpha Beta Pruning

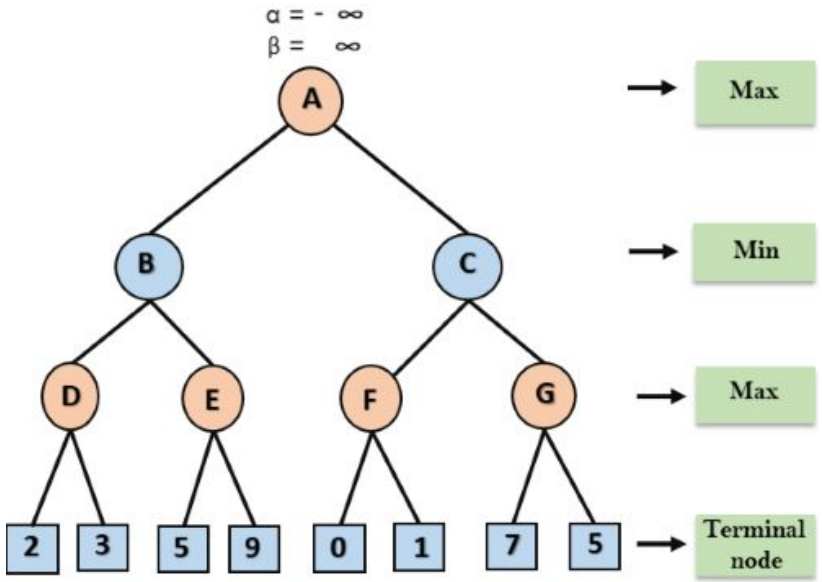
```
function minimax(node, depth, alpha, beta, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
  maxEva= -infinity
  for each child of node do
  s= minimax(child, depth-1, alpha, beta, False)
 maxEva= max(maxEva, s)
 alpha= max(alpha, maxEva)
  if beta<=alpha
 break
 return maxEva
else                    // for Minimizer player
  minEva= +infinity
for each child of node do
 s= minimax(child, depth-1, alpha, beta, tru
      e)
  minEva= min(minEva, s)
  beta= min(beta, minEva)
   if beta<=alpha
 break
return minEva
```

Alpha Beta Pruning

Working of Alpha-Beta Pruning:

- Let's take an example of two-player search tree to understand the working of Alpha-beta pruning
- **Step 1:** At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.

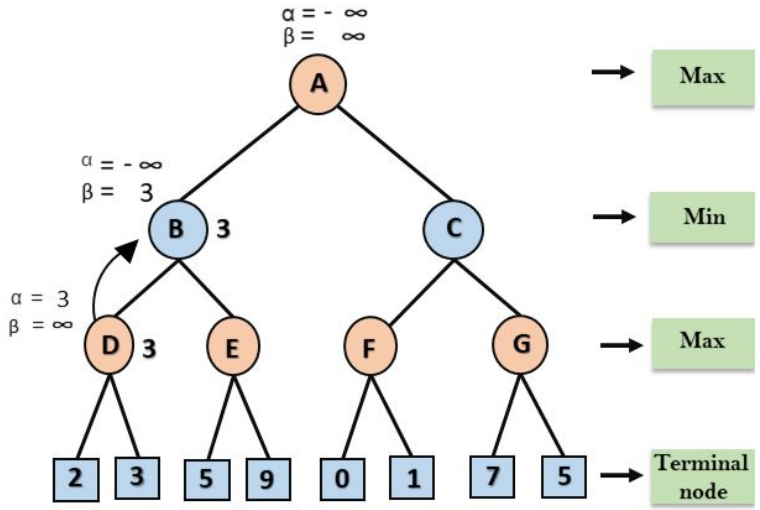


Alpha Beta Pruning

Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

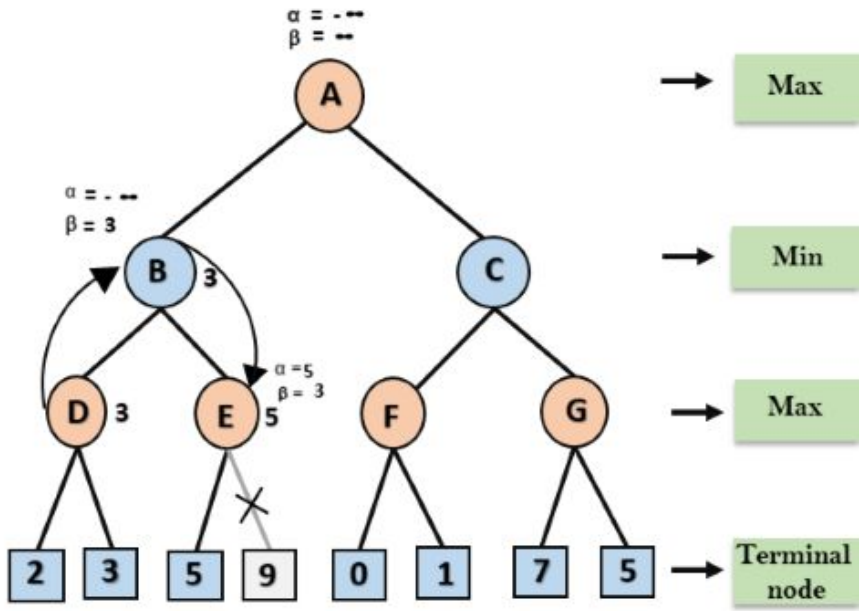
Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.

In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.



Alpha Beta Pruning

Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.

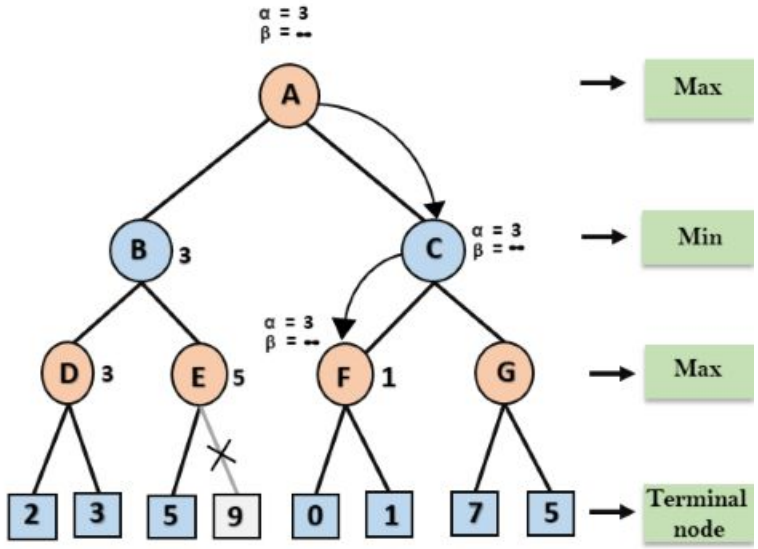


Alpha Beta Pruning

Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

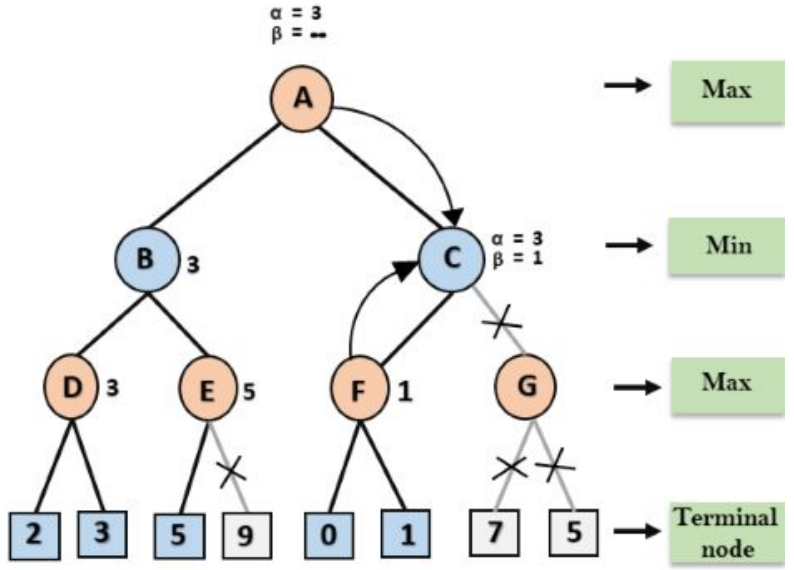
At node C, α=3 and β= +∞, and the same values will be passed on to node F.

Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



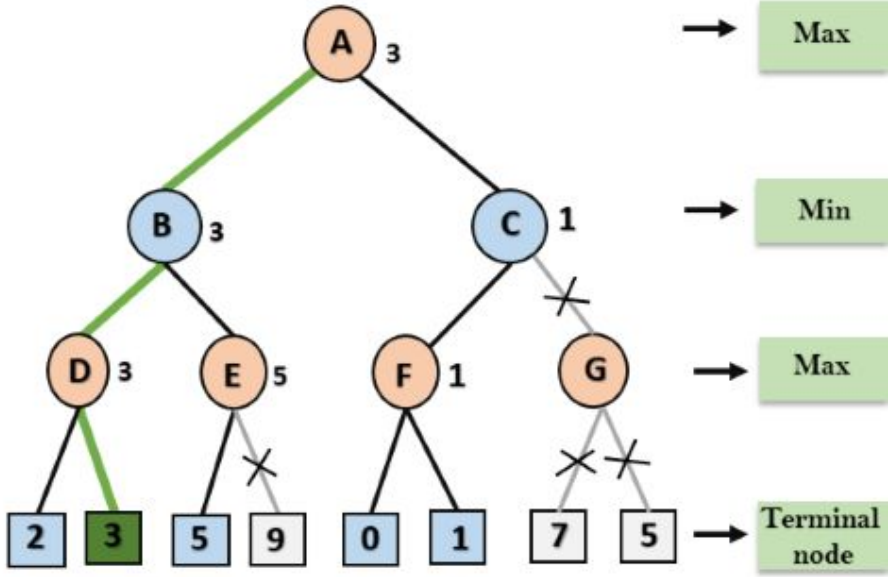
Alpha Beta Pruning

- **Step 7:** Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Alpha Beta Pruning

Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Alpha Beta Pruning

Move Ordering in Alpha-Beta pruning:

- The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.
- It can be of two types:
- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Unit 2 List of Topics

- Searching techniques – Uninformed search – General search Algorithm
- Uninformed search Methods – Breadth First Search
- Uninformed search Methods – Depth First Search
- Uninformed search Methods – Depth limited Search
- *Uniformed search Methods- Iterative Deepening search*
- *Bi-directional search*
- *Informed search- Generate and test, Best First search*
- *Informed search-A* Algorithm*
- *AO* search*
- *Local search Algorithms-Hill Climbing, Simulated Annealing*
- *Local Beam Search*
- *Genetic Algorithms*
- *Adversarial search Methods-Game playing-Important concepts*
- *Game playing and knowledge structure.*
- *Game as a search problem-Minimax Approach*
- *Minimax Algorithm*
- *Alpha beta pruning*
- *Game theory problems*

What is Game Theory?

- It deals with Bargaining.
- The whole process can be expressed Mathematically
- Based on Behavior Theory, has a more casual approach towards study of Human Behavior.
- It also considers how people Interact in Groups.

Game Theory Definition

•Theory of rational behavior for interactive decision problems.

• In a *game*, several agents strive to maximize their (expected) utility index by choosing particular courses of action, and each agent's final utility payoffs depend on the profile of courses of action chosen by *all* agents.

•The interactive situation, specified by the set of participants, the possible courses of action of each agent, and the set of all possible utility payoffs, is called a *game*;

• the agents 'playing' a game are called the *players*.

Definitions

Definition: *Zero-Sum Game* – A game in which the payoffs for the players always adds up to zero is called a zero-sum game.

Definition: *Maximin strategy* – If we determine the least possible payoff for each strategy, and choose the strategy for which this minimum payoff is largest, we have the maximin strategy.

A Further Definition

Definition: *Constant-sum and nonconstant-sum game* – If the payoffs to all players add up to the same constant, regardless which strategies they choose, then we have a constant-sum game. The constant may be zero or any other number, so zero-sum games are a class of constant-sum games. If the payoff does not add up to a constant, but varies depending on which strategies are chosen, then we have a non-constant sum game.

Game theory: assumptions

(1) Each decision maker has available to him two or more well-specified choices or sequences of choices.

(2) Every possible combination of plays available to the players leads to a well-defined end-state (win, loss, or draw) that terminates the game.

(3) A specified payoff for each player is associated with each end-state.

Game theory: assumptions (Cont)

(4) Each decision maker has perfect knowledge of the game and of his opposition.

(5) All decision makers are rational; that is, each player, given two alternatives, will select the one that yields him the greater payoff.

Rules, Strategies, Payoffs, and Equilibrium

- A **game** is a contest involving two or more decision makers, each of whom wants to win
  - **Game theory** is the study of how optimal strategies are formulated in conflict
- A player's payoff is the amount that the player wins or loses in a particular situation in a game.
- A players has a dominant strategy if that player's best strategy does not depend on what other players do.
- A two-person game involves two parties (X and Y)
  - A **zero-sum game** means that the sum of losses for one player must equal the sum of gains for the other. Thus, the overall sum is zero

Payoff Matrix - Store X

- Two competitors are planning radio and newspaper advertisements to increase their business. This is the payoff matrix for store X. A negative number means store Y has a positive payoff

		GAME PLAYER Y's STRATEGIES	
		Y_1 (Use radio)	Y_2 (Use newspaper)
GAME PLAYER X's STRATEGIES	X_1 (Use radio)	3	5
	X_2 (Use newspaper)	1	–2

Game Outcomes

Game Outcomes

STORE *X*'s STRATEGY	STORE *Y*'s STRATEGY	OUTCOME (% CHANGE IN MARKET SHARE)
X_1 (use radio)	Y_1 (use radio)	*X* wins 3 and *Y* loses 3
X_1 (use radio)	Y_2 (use newspaper)	*X* wins 5 and *Y* loses 5
X_2 (use newspaper)	Y_1 (use radio)	*X* wins 1 and *Y* loses 1
X_2 (use newspaper)	Y_2 (use newspaper)	*X* loses 2 and *Y* wins 2

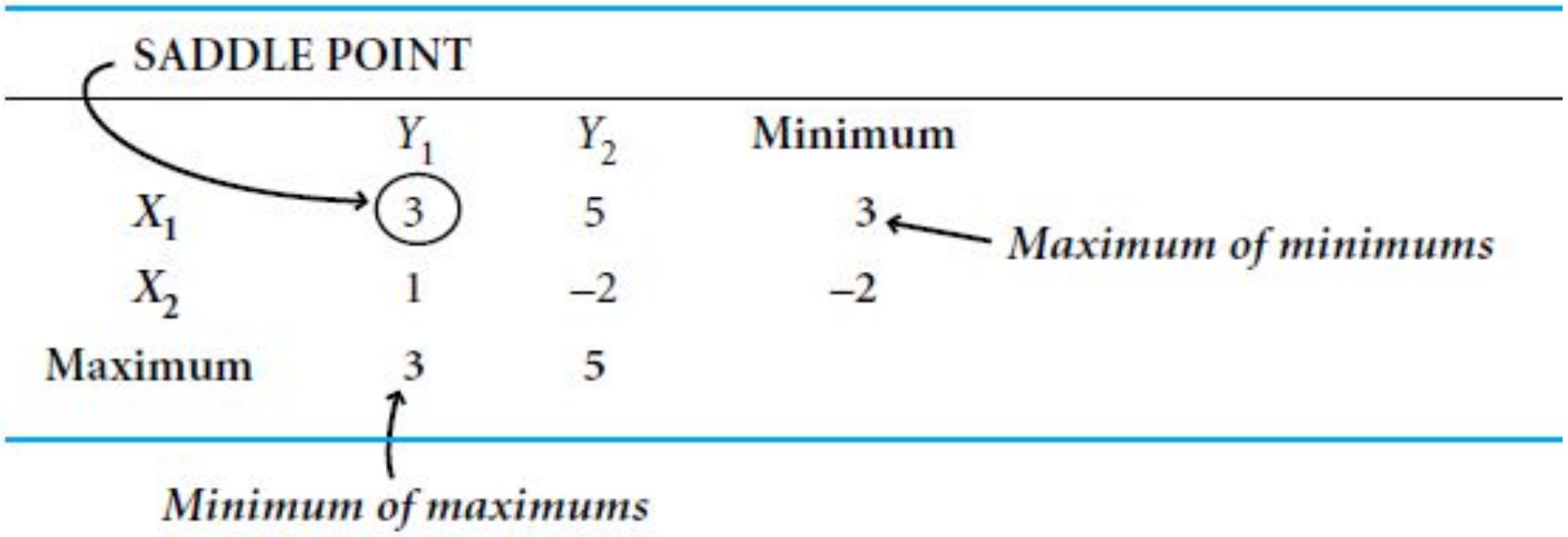
Minimax Criterion

- Look to the “cake cutting problem” to explain
  - Cutter – maximize the minimum the Chooser will leave him
  - Chooser – minimize the maximum the Cutter will get

Chooser □ Cutter	Choose bigger piece	Choose smaller piece
Cut cake as evenly as possible	Half the cake minus a crumb	Half the cake plus a crumb
Make one piece bigger than the other	Small piece	Big piece

Minimax Criterion

- If the upper and lower values are the same, the number is called the **value of the game** and an equilibrium or **saddle point condition** exists
- The value of a game is the average or expected game outcome if the game is played an infinite number of times
- A saddle point indicates that each player has a **pure strategy** i.e., the strategy is followed no matter what the opponent does



Saddle Point

- Von Neumann likened the solution point to the point in the middle of a saddle shaped mountain pass
  - It is, at the same time, the maximum elevation reached by a traveler going through the pass to get to the other side and the minimum elevation encountered by a mountain goat traveling the crest of the range

Pure Strategy - Minimax Criterion

		Player Y's Strategies		Minimum Row Number
		Y1	Y2	
Player X's strategies	X1	10	6	6
	X2	-12	2	-12
Maximum Column Number		10	6	

Mixed Strategy Game

- When there is no saddle point, players will play each strategy for a certain percentage of the time
- The most common way to solve a mixed strategy is to use the expected gain or loss approach
- A player plays each strategy a particular percentage of the time so that the expected value of the game does not depend upon what the opponent does

	Y1 P	Y2 1-P	Expected Gain
X1 Q	4	2	4P+2(1-P)
X2 1-Q	1	10	1P+10(1-p)
	4Q+1(1-Q)	2Q+10(1-q)	

Mixed Strategy Game
: Solving for P & Q

4P+2(1-P) = 1P+10(1-P)

or: P = 8/11 and 1-p = 3/11

Expected payoff:

1P+10(1-P)

=1(8/11)+10(3/11)

$EP_X = 3.46$

4Q+1(1-Q)=2Q+10(1-q)

or: Q=9/11 and 1-Q = 2/11

Expected payoff:

$EP_Y = 3.46$

Mixed Strategy Game : Example

- Using the solution procedure for a mixed strategy game, solve the following game

	Y_1	Y_2
X_1	4	2
X_2	0	10

Mixed Strategy Game

Example

- This game can be solved by setting up the mixed strategy table and developing the appropriate equations:

		Y_1	Y_2	
		P	$1 - P$	**Expected gain**
X_1	Q	4	2	$4P + 2(1 - P)$
X_1	$1 - Q$	1	10	$0P + 10(1 - P)$
Expected gain		$4Q + 0(1 - Q)$	$2Q + 10(1 - Q)$	

Mixed Strategy Game: Example

The equations for Q are

$$4Q + 0(1 - Q) = 2Q + 10(1 - Q)$$
$$4Q = 2Q + 10 - 10Q$$
$$12Q = 10 \text{ or } Q = {}^{10}/_{12} \text{ and } 1 - Q = {}^{2}/_{12}$$

The equations for P are

$$4P + 2(1 - P) = 0P + 10(1 - P)$$
$$4P + 2 - 2P = 10 - 10P$$
$$12P = 8 \text{ or } P = {}^{8}/_{12} \text{ and } 1 - P = {}^{4}/_{12}$$

Two-Person Zero-Sum and Constant-Sum Games

Two-person zero-sum and constant-sum games are played according to the following basic assumption:

Each player chooses a strategy that enables him/her to do the best he/she can, given that his/her opponent *knows the strategy he/she is following*.

A two-person zero-sum game has a saddle point if and only if

Max (row minimum) = min (column maximum)

all *all*

rows *columns*

(1)

Two-Person Zero-Sum and Constant-Sum Games (Cont)

If a two-person zero-sum or constant-sum game has a saddle point, the row player should choose any strategy (row) attaining the maximum on the right side of (1). The column player should choose any strategy (column) attaining the minimum on the right side of (1).
In general, we may use the following method to find the optimal strategies and value of two-person zero-sum or constant-sum game:

Step 1 Check for a saddle point. If the game has none, go on to step 2.

Two-Person Zero-Sum and Constant-Sum Games (Cont)

Step 2 Eliminate any of the row player's dominated strategies. Looking at the reduced matrix (dominated rows crossed out), eliminate any of the column player's dominated strategies and then those of the row player. Continue until no more dominated strategies can be found. Then proceed to step 3.

Step 3 If the game matrix is now 2 x 2, solve the game graphically. Otherwise, solve by using a linear programming method.

Zero Sum Games

- Game theory assumes that the decision maker and the opponent are rational, and that they subscribe to the maximin criterion as the decision rule for selecting their strategy
- This is often reasonable if when the other player is an opponent out to maximize his/her own gains, e.g. competitor for the same customers.
- Consider:

  Player 1 with three strategies S1, S2, and S3 and Player 2 with four strategies OP1, OP2, OP3, and OP4.

Zero Sum Games (Cont)

- The value 4 achieved by both players is called the *value* of the game
- The intersection of S2 and OP2 is called a *saddle point*. A game with a saddle point is also called a game with an *equilibrium solution*.
- At the saddle point, neither player can improve their payoff by switching strategies

Zero Sum Games- To do problem!

Let's take the following example: Two TV channels (1 and 2) are competing for an audience of 100 viewers. The rule of the game is to simultaneously announce the type of show the channels will broadcast. Given the payoff matrix below, what type of show should channel 1 air?

		Channel 1		
		Movie	Reality Show	Stand-up Comedy
Channel 2	Movie	37	17	62
	Reality Show	47	60	52
	Stand-up Comedy	40	16	72

Two-person zero-sum game – Dominance property

dominance method Steps (Rule)	
Step-1:	1. If all the elements of Column-i are greater than or equal to the corresponding elements of any other Column-j, then the Column-i is dominated by the Column-j and it is removed from the matrix. eg. If Column-2 $\geq$ Column-4, then remove Column-2
Step-2:	1. If all the elements of a Row-i are less than or equal to the corresponding elements of any other Row-j, then the Row-i is dominated by the Row-j and it is removed from the matrix. eg. If Row-3 $\leq$ Row-4, then remove Row-3
Step-3:	Again repeat Step-1 & Step-2, if any Row or Column is dominated, otherwise stop the procedure.

Two-person zero-sum game – Dominance property- To do problem!

Player A \ Player B	B1	B2	B3	B4
A1	3	5	4	2
A2	5	6	2	4
A3	2	1	4	0
A4	3	3	5	2

Solution

		Player B	
		$B3$	$B4$
Player A	$A2$	2	4
	$A4$	5	2

The Prisoner's Dilemma

- The prisoner's dilemma is a universal concept. Theorists now realize that prisoner's dilemmas occur in biology, psychology, sociology, economics, and law.
- The prisoner's dilemma is apt to turn up anywhere a conflict of interests exists -- and the conflict need not be among sentient beings.
- Study of the prisoner's dilemma has great power for explaining why animal and human societies are organized as they are. It is one of the great ideas of the twentieth century, simple enough for anyone to grasp and of fundamental importance (...).
- The prisoner's dilemma has become one of the premier philosophical and scientific issues of our time. It is tied to our very survival (W. Poundstone,1992, p. 9).

Prisoner's Dilemma

- Two members of a criminal gang are arrested and imprisoned.
  - They are placed under solitary confinement and have no chance of communicating with each other
- The district attorney would like to charge them with a recent major crime but has insufficient evidence
  - He has sufficient evidence to convict each of them of a lesser charge
  - If he obtains a confession from one or both the criminals, he can convict either or both on the major charge.

Prisoner's Dilemma

- The district attorney offers each the chance to turn state's evidence.
  - If only one prisoner turns state's evidence and testifies against his partner he will go free while the other will receive a 3 year sentence.
  - Each prisoner knows the other has the same offer
  - The catch is that if both turn state's evidence, they each receive a 2 year sentence
  - If both refuse, each will be imprisoned for 1 year on the lesser charge

A game is described by

- The number of players
- Their strategies and their turn
- Their payoffs (profits, utilities etc) at the outcomes of the game

Game Theory Definition

Payoff matrix

Normal- or strategic form

Player B

Player A	Left	Right
Top	3, 0	0, -4
Bottom	2, 4	-1, 3

Game Playing

How to solve a situation like this?

- The most simple case is where there is a optimal choice of strategy no matter what the other players do; *dominant strategies*.
- Explanation: For Player A it is always better to choose Top, for Player B it is always better to choose left.
- A *dominant strategy* is a strategy that is best no matter what the other player does.

Nash equilibrium

- If Player A's choice is optimal given Player B's choice, and B's choice is optimal given A's choice, a pair of strategies is a *Nash equilibrium*.
- When the other players' choice is revealed neither player like to change her behavior.
- If a set of strategies are best responses to each other, the strategy set is a *Nash equilibrium*.

Payoff matrix
Normal- or strategic form

Player B

Player A	Left	Right
Top	1, 1	2, 3*
Bottom	2, 3*	1, 2

Solution

- Here you can find a *Nash equilibrium*; Top is the best response to Right and Right is the best response to Top. Hence, (Top, Right) is a *Nash equilibrium*.
- But there are two problems with this solution concept.

Problems

- A game can have several *Nash equilibriums*. In this case also (Bottom, Left).
- There may not be a *Nash equilibrium* (in pure strategies).

Payoff matrix
Normal- or strategic form

Player B

Player A	Left	Right
Top	1, -1	-1, 1
Bottom	-1, 1	1, -1

Nash equilibrium in mixed strategies

- Here it is not possible to find strategies that are best responses to each other.
- If players are allowed to randomize their strategies we can find s solution; a *Nash equilibrium* in mixed strategies.
- An equilibrium in which each player chooses the optimal frequency with which to play her strategies given the frequency choices of the other agents.

The prisoner's dilemma

Two persons have committed a crime, they are held in separate rooms. If they both confess they will serve two years in jail. If only one confess she will be free and the other will get the double time in jail. If both deny they will be hold for one year.

Prisoner's dilemma
Normal- or strategic form

Prisoner B

Prisoner A	Confess	Deny
Confess	-2, -2	0, -4
Deny	-4, 0	-1, -1*

Solution

Confess is a *dominant strategy* for both. If both Deny they would be better off. This is the dilemma.

Nash Equilibrium – To do Problems!

	HENRY		
JANE		L	R
	U	8,7	4,6
	D	6,5	7,8

	McD (1)		
KFC (1)		L	R
	U	9,9*	1,10
	D	10,1	2,2

	COKE		
PEPSI		L	R
	U	6,8*	4,7
	D	7,6	3,7

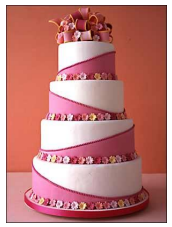
	B		
A		L	R
	U	7,6*	5,5
	D	4,5	6,4

GAME PLAYING & MECHANISM DESIGN

Mechanism Design is the design of games or reverse engineering of games; could be called *Game Engineering*

Involves inducing a game among the players such that in some equilibrium of the game, a desired social choice function is implemented

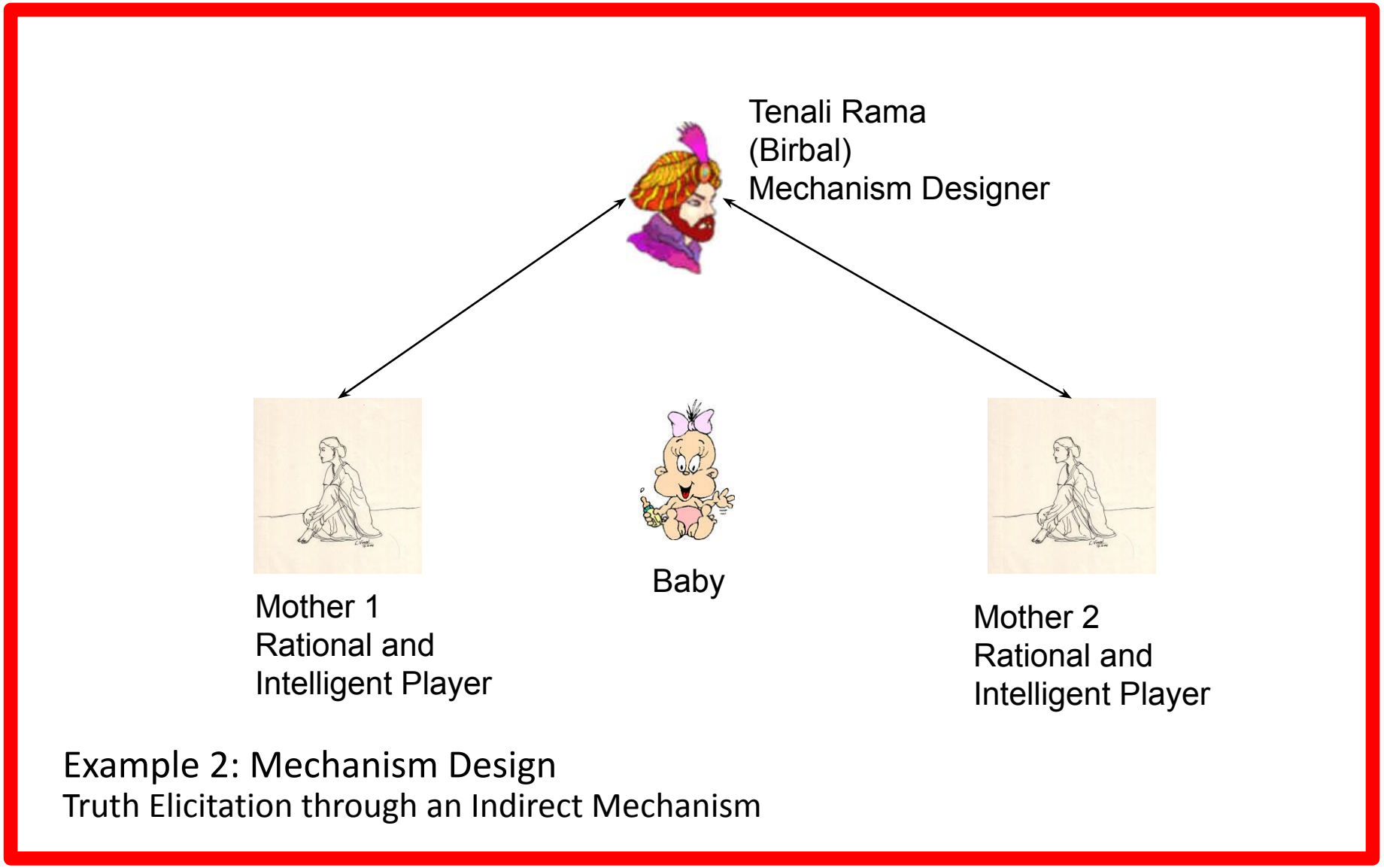
Mother
Social Planner
Mechanism Designer



Kid 1
Rational and
Intelligent

Kid 2
Rational and
Intelligent

Example 1: Mechanism Design
Fair Division of a Cake



Example 2: Mechanism Design
Truth Elicitation through an Indirect Mechanism

One Seller, Multiple Buyers, Single Indivisible Item

Example: B1: 40, B2: 45, B3: 60, B4: 80

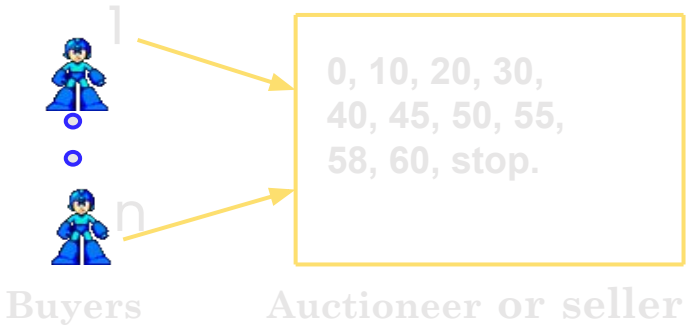
Winner: whoever bids the highest; in this case B4

Payment: Second Highest Bid: in this case, 60.

Vickrey showed that this mechanism is Dominant Strategy Incentive Compatible (DSIC) ;Truth Revelation is good for a player irrespective of what other players report

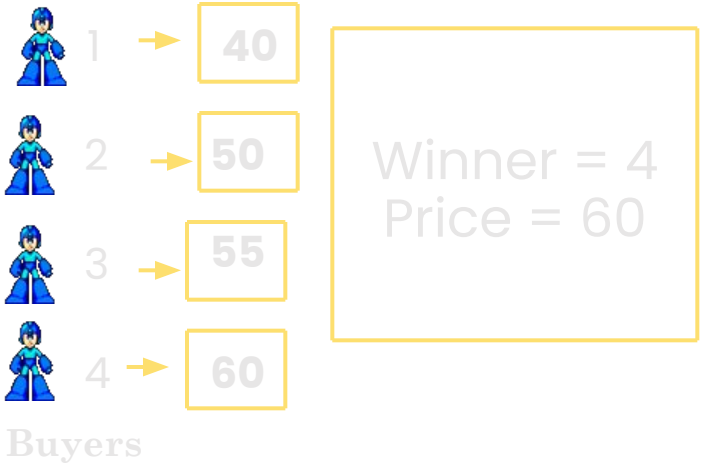
MECHANISM DESIGN: EXAMPLE 3 : VICKREY AUCTION

English Auction

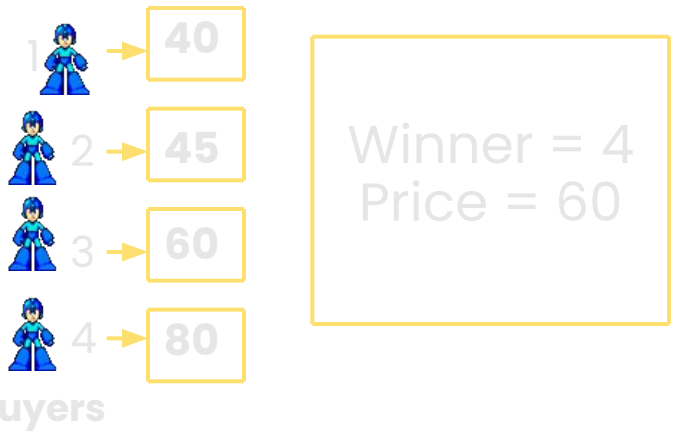


Dutch Auction

First Price Auction



Vickrey Auction



Four Basic Types of Auctions

Simple reflex agents

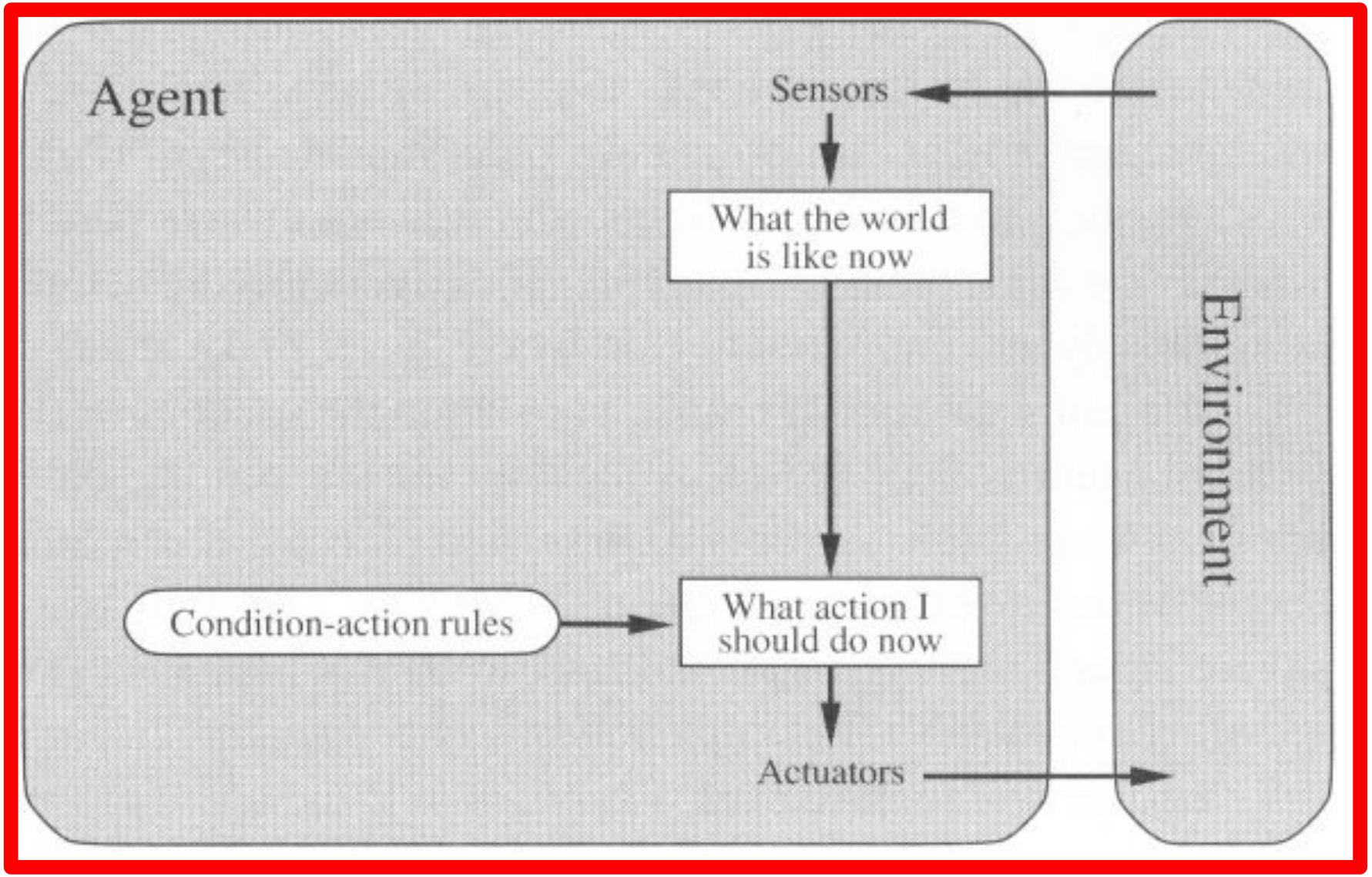
- It uses just ***condition-action rules***
  - The rules are like the form “if … then …”
  - efficient but have narrow range of applicability
  - Because knowledge sometimes cannot be stated explicitly
  - Work only
    - if the environment is fully observable

Simple reflex agents

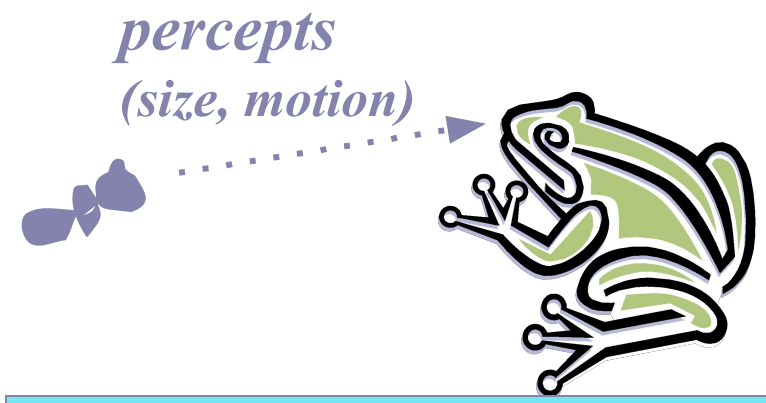

```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

Simple reflex agents



A Simple Reflex Agent in Nature



RULES:

(1) If small moving object,
then activate SNAP

(2) If large moving object,
then activate AVOID and inhibit SNAP

ELSE (not moving) then NOOP

needed for completeness

Action: SNAP or AVOID or NOOP

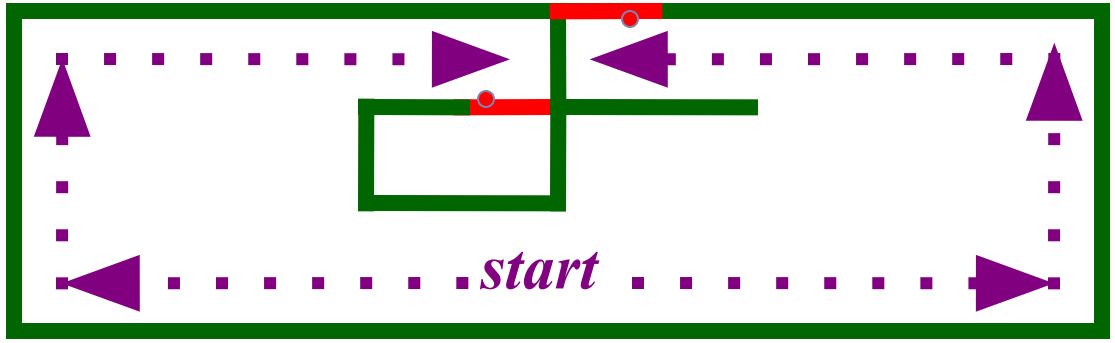
Model-based Reflex Agents

- For the world that is partially observable
  - the agent has to keep track of an internal state
    - That depends on the percept history
    - Reflecting some of the unobserved aspects
    - E.g., driving a car and changing lane
- Requiring two types of knowledge
  - How the world evolves independently of the agent
  - How the agent's actions affect the world

Example Table Agent With Internal State

IF	THEN
Saw an object ahead, and turned right, and it's now clear ahead	Go straight
Saw an object Ahead, turned right, and object ahead again	Halt
See no objects ahead	Go straight
See an object ahead	Turn randomly

Example Reflex Agent With Internal State
Wall-Following



Actions: left, right, straight, open-door

Rules:

1. If open(left) & open(right) and open(straight) then choose randomly between right and left
2. If wall(left) and open(right) and open(straight) then straight
3. If wall(right) and open(left) and open(straight) then straight
4. If wall(right) and open(left) and wall(straight) then left
5. If wall(left) and open(right) and wall(straight) then right
6. If wall(left) and door(right) and wall(straight) then open-door
7. If wall(right) and wall(left) and open(straight) then straight.
8. (Default) Move randomly

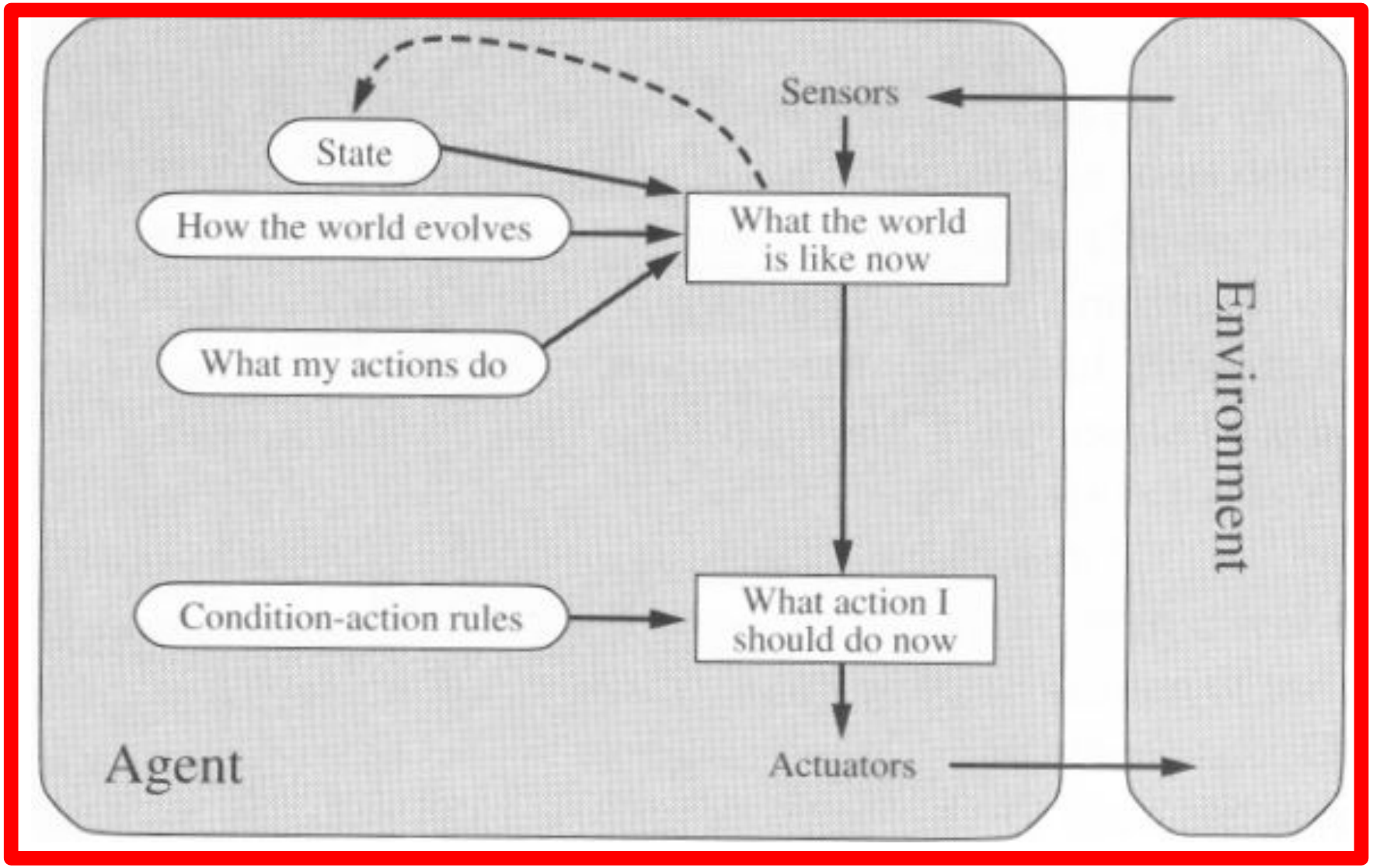
Model-based Reflex Agents


```
function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```

The agent is with memory

Model-based Reflex Agents



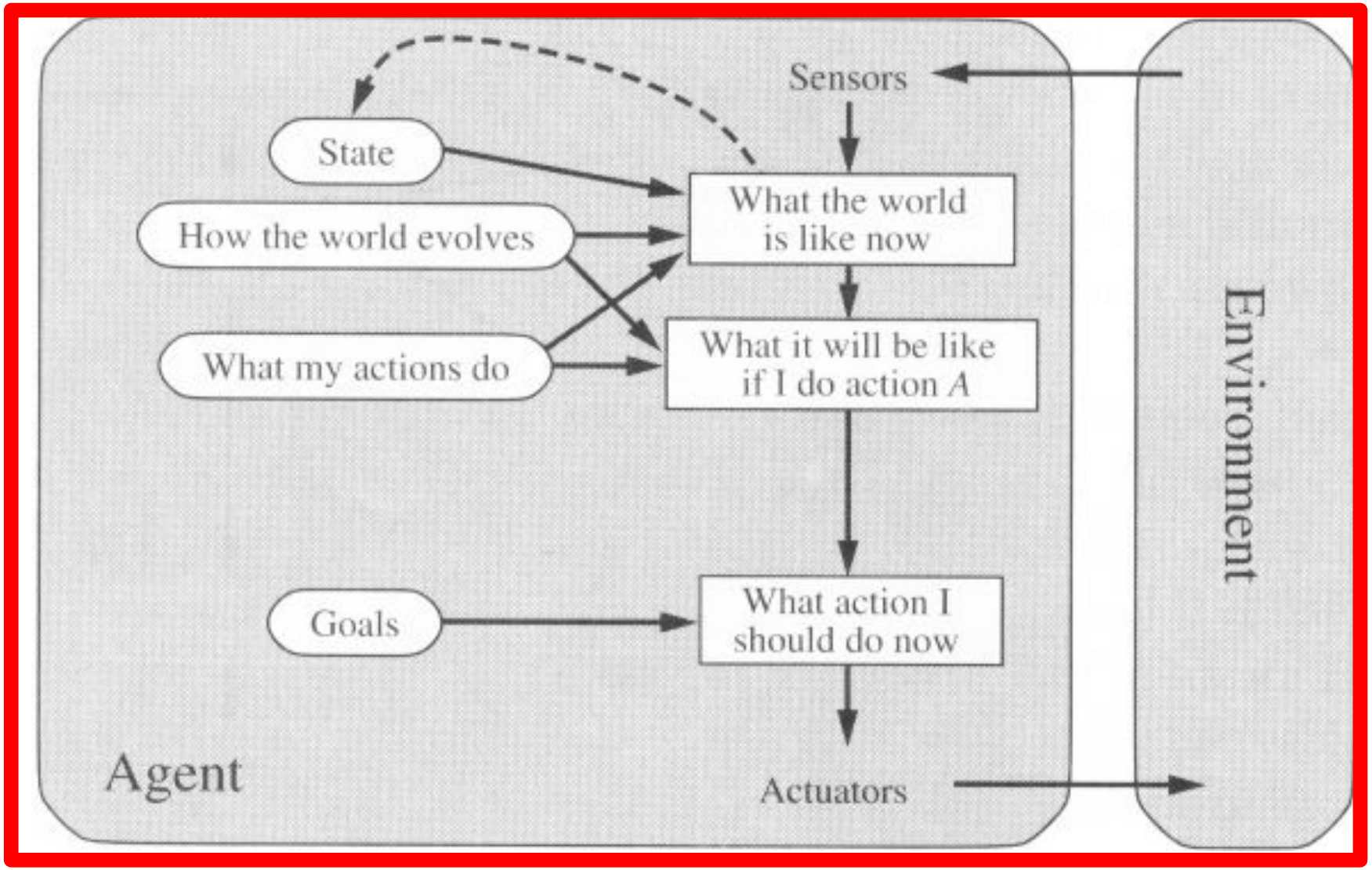
Goal-based agents

- Current state of the environment is always not enough
- The goal is another issue to achieve
  - Judgment of rationality / correctness
- Actions chosen goals, based on
  - the current state
  - the current percept

Goal-based agents

- Conclusion
  - Goal-based agents are less efficient
  - but more flexible
    - Agent Different goals different tasks
  - Search and planning
    - two other sub-fields in AI
    - to find out the action sequences to achieve its goal

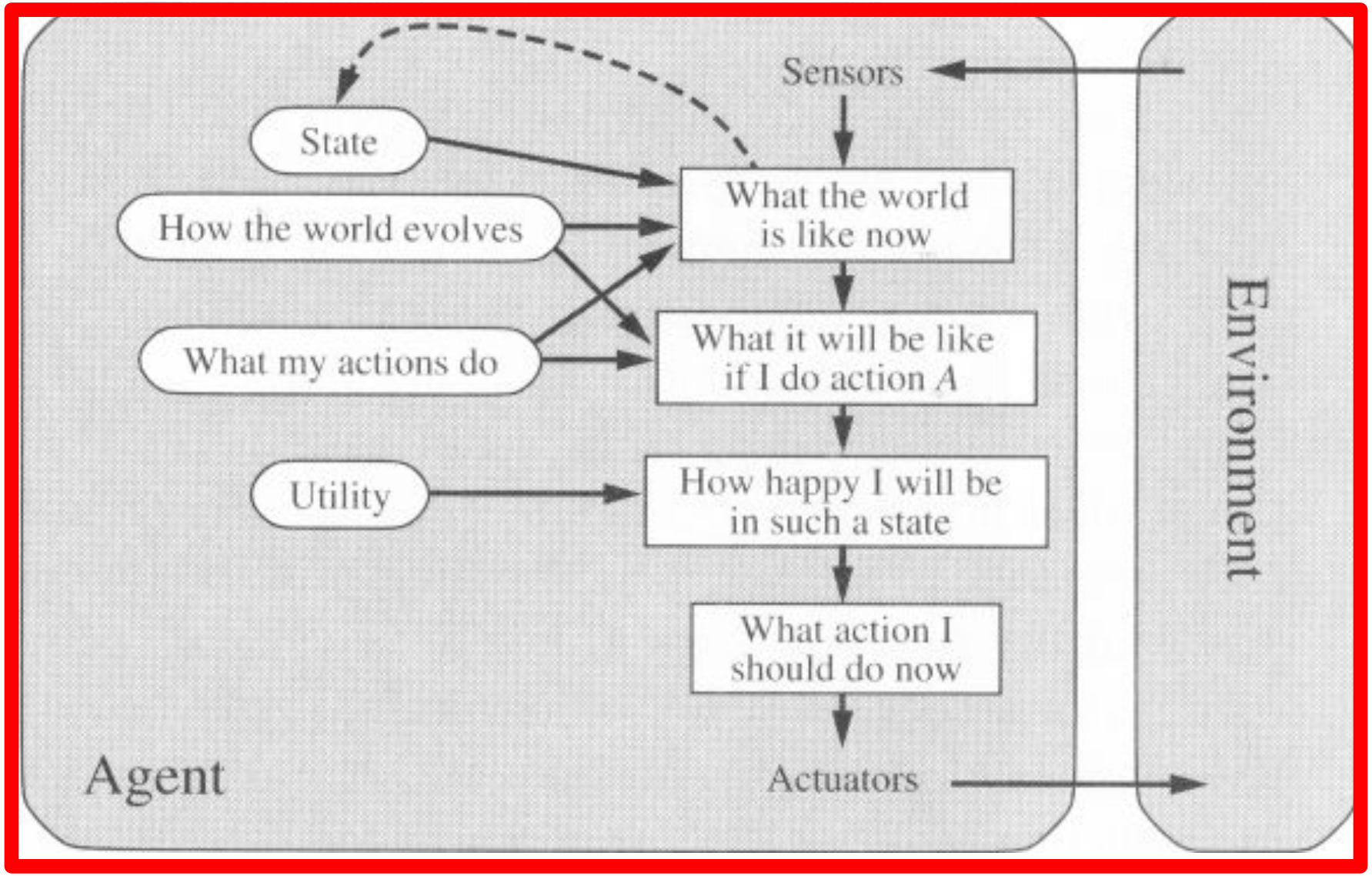
Goal-based agents



Utility-based agents

- Goals alone are not enough
  - to generate **high-quality** behavior
  - E.g. meals in Canteen, good or not ?
- Many action sequences the goals
  - some are better and some worse
  - If goal means success,
  - then **utility** means the degree of success (how successful it is)

Utility-based agents(4)



Utility-based agents

- it is said state A has higher utility
  - If state A is more preferred than others
- Utility is therefore a function
  - that maps a state onto a real number
  - the degree of success

Utility-based agents (3)

- Utility has several advantages:
  - When there are conflicting goals,
    - Only some of the goals but not all can be achieved
    - utility describes the appropriate trade-off
  - When there are several goals
    - None of them are achieved **certainly**
    - utility provides a way for the decision-making

Learning Agents

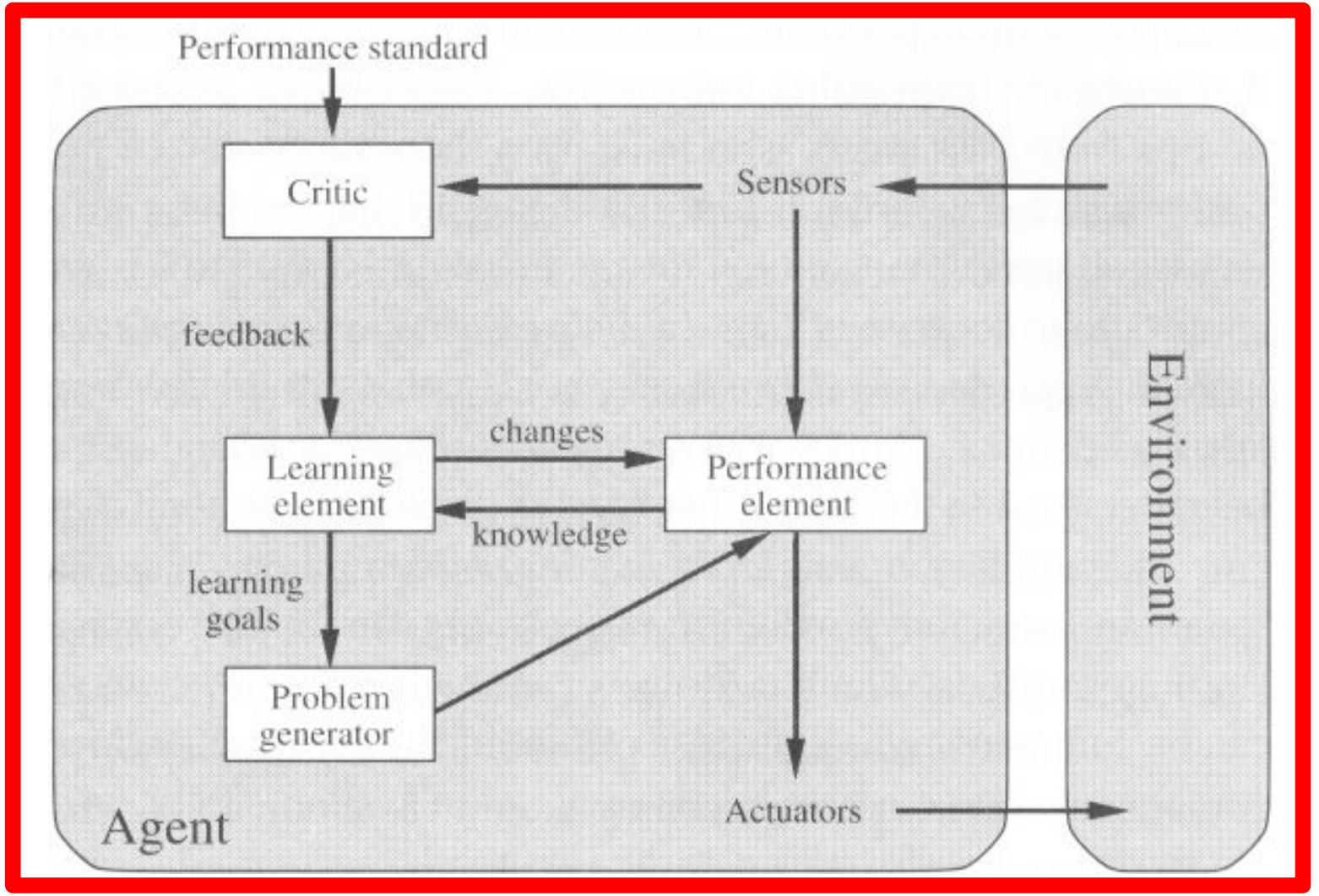
- After an agent is programmed, can it work immediately?
  - No, it still need teaching
- In AI,
  - Once an agent is done
  - We teach it by giving it a set of examples
  - Test it by using another set of examples
- We then say the agent learns
  - A learning agent

Learning Agents

- Four conceptual components
  - Learning element
    - Making improvement
  - Performance element
    - Selecting external actions
  - Critic
    - Tells the Learning element how well the agent is doing with respect to fixed performance standard.

      (Feedback from user or examples, good or not?)
  - Problem generator
    - Suggest actions that will lead to new and informative experiences.

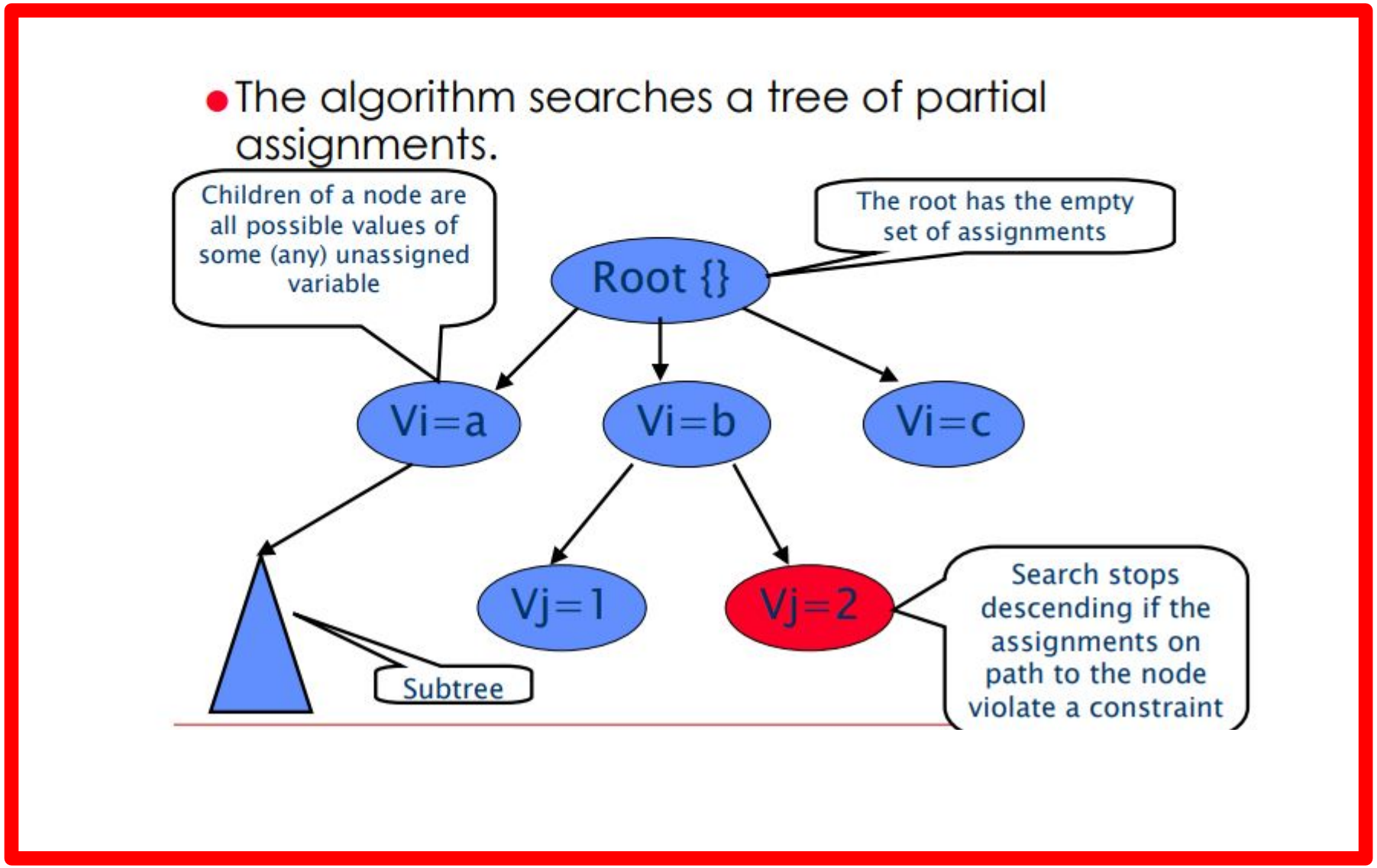
Learning Agents



Constraint Satisfaction Problem

- Search Space is constrained by a set of conditions and dependencies
- Class of problems where the search space is constrained called as CSP
- For example, Time-table scheduling problem for lecturers.
- Constraint,
  - Two Lecturers can not be assigned to same class at the same time
- To solve CSP, the problem is to be decomposed and analyse the structure
- Constraints are typically mathematical or logical relationships

Solving Constraint Satisfaction Problem



CSP

- Any problem in the world can be mathematically represented as CSP
- Hypothetical problem does not have constraints
- Constraint restricts movement, arrangement, possibilities and solutions
- Example: if only concurrency notes of 2,5,10 rupees are available and we need to give certain amount of money, say Rs. 111 to a salesman and the total number of notes should be between 40 and 50.
- Expressed as, Let n be number of notes,
  - $40 < n < 50$
- and
  - $c_1X_1 + c_2X_2 + c_3X_3 = 111$

CSP

- Types of Constraints
  - Unary Constraints - Single variable
  - Binary Constraints - Two Variables
  - Higher Order Constraints - More than two variables
- CSP can be represented as Search Problem
- Initial state is empty assignment, while successor function is a non-conflicting value assigned to an unassigned variables
- Goal test checks whether the current assignment is complete and path cost is the cost for the path to reach the goal state
- CSP Solutions leads to the final and complete assignment with no exception

Cryptarithmetic puzzles

- Cryptarithmetic puzzles are also represented as CSP
- Example: MIKE + JACK = JOHN
- Replace every letter in puzzle with single number (number should not be repeated for two different alphabets)
- The domain is *{ 0,1, ... , 9 }*

- Often treated as the ten-variable constraint problem
- where the constraints are:
  - All the variables should have a different value
  - The sum must work out

Cryptarithmetic puzzles

- M * 1000 + I * 100 + K * 10 + E + J * 1000 + A * 100 + C * 10 + K = J * 1000 + O * 100 + H * 10 + N
- Constraint Domain is represented by Five-tuple and represented by,
  - *D = {var, f , O, dv, rg}*
- *Var* stands for set variables, *f* is set functions, *O* stands for the set of legitimate operators to be used, *dv* is domain variable and *rg* is range of function in the constraint
- Constraint without conjunction is referred as Primitive constraint (for Eg., $x < 9$)
- Constraint with conjunction is called as non-primitive constraint or a generic constraint (For Eg., $x < 9$ and $x > 2$)

Crypt arithmetic puzzles – Solved Example


```
  TO
  GO
  ---
 OUT
 ------
  21
  81        G = 8/9
  ---
 102
 ------     2 + G = U + 10
```

Var	Value
T	2
O	1
G	8
U	0

Cryptarithmetic puzzles

- SEND + MORE = MONEY

c4 c3 c2 c1

```
   S  E  N  D
   M     O  R  E
   ----------------------
M O     N  E  Y
  9 5 6 7
  1  0 8 5
 ----------
1  0 8 52
```

D + E = Y + 10C1
N + R + C1 = E + 10C2
E + O + C2 = N + 10C3
S + M + C3 = O + 10C4
C4 = M

CSP- Room Coloring Problem
-CSP as a Search Problem

- Let *K* for Kitchen, *D* for Dining Room, *H* is for Hall, B_2 and B_3 are bedrooms 2 and 3, MB_1 is master bedroom, *SR* is the store Room, *GR* is Guest Room and *Lib* is Library
- Constraints
  - All bedrooms should not be colored red, only one can
  - No two adjacent rooms can have the same color
  - The colors available are red, blue, green and violet
  - Kitchen should not be colored green
  - Recommended to color the kitchen as blue
  - Dining room should not have violet color

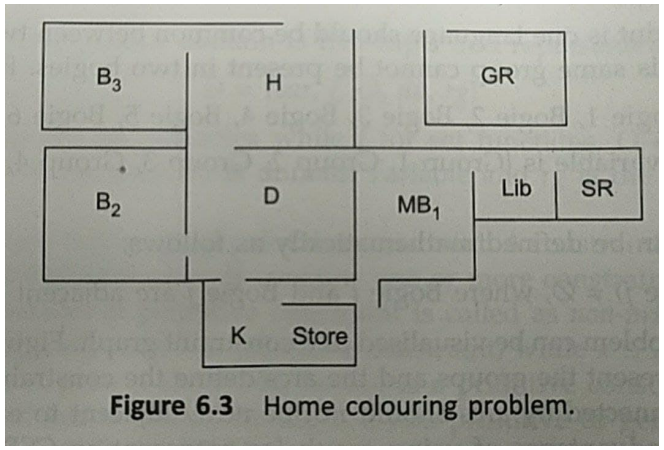


Figure 6.3 Home colouring problem.

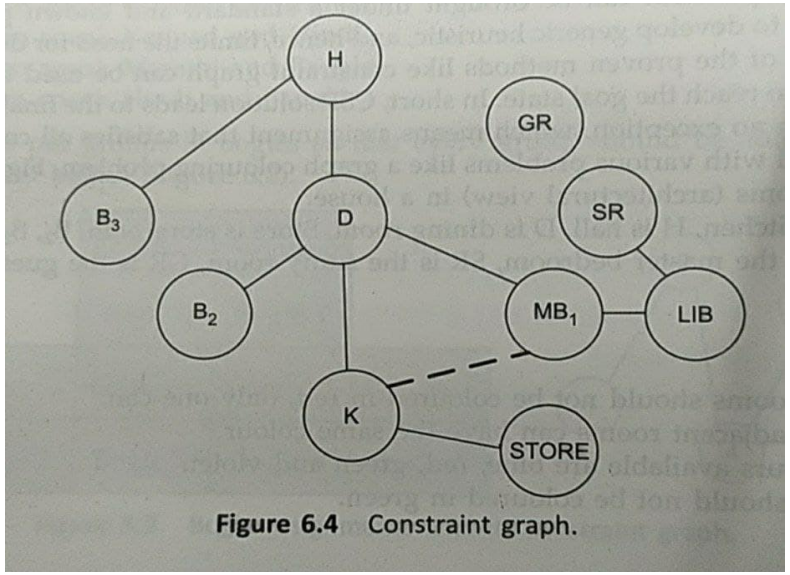


Figure 6.4 Constraint graph.

Room Coloring Problem – Representation as a Search Tree

- Soft Constraints are that they are cost-oriented or preferred choice
- All paths in the Search tree can not be accepted because of the violation in constraints

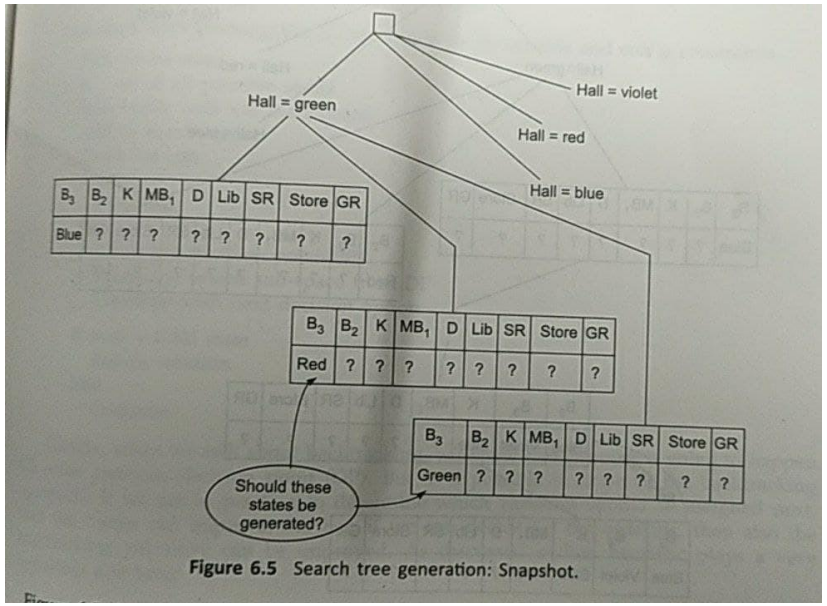


Figure 6.5 Search tree generation: Snapshot.

Backtracking Search for CSP

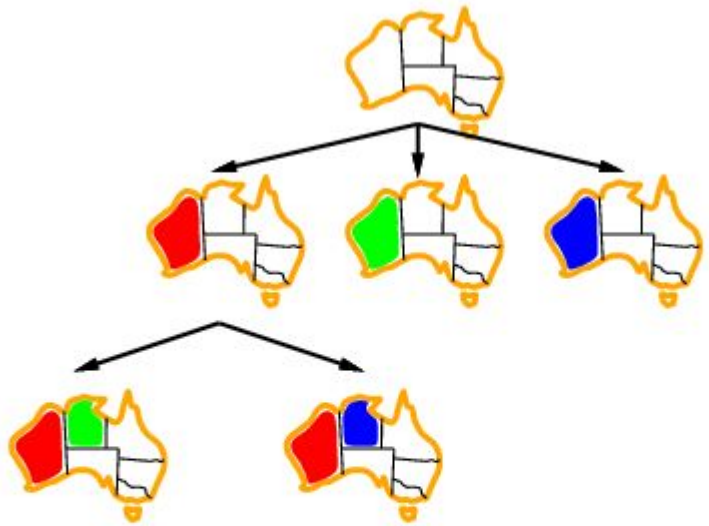
- Assignment of value to any additional variable within constraint can generate a legal state (Leads to successor state in search tree)
- Nodes in a branch backtracks when there is no options are available

Example: Map-Coloring

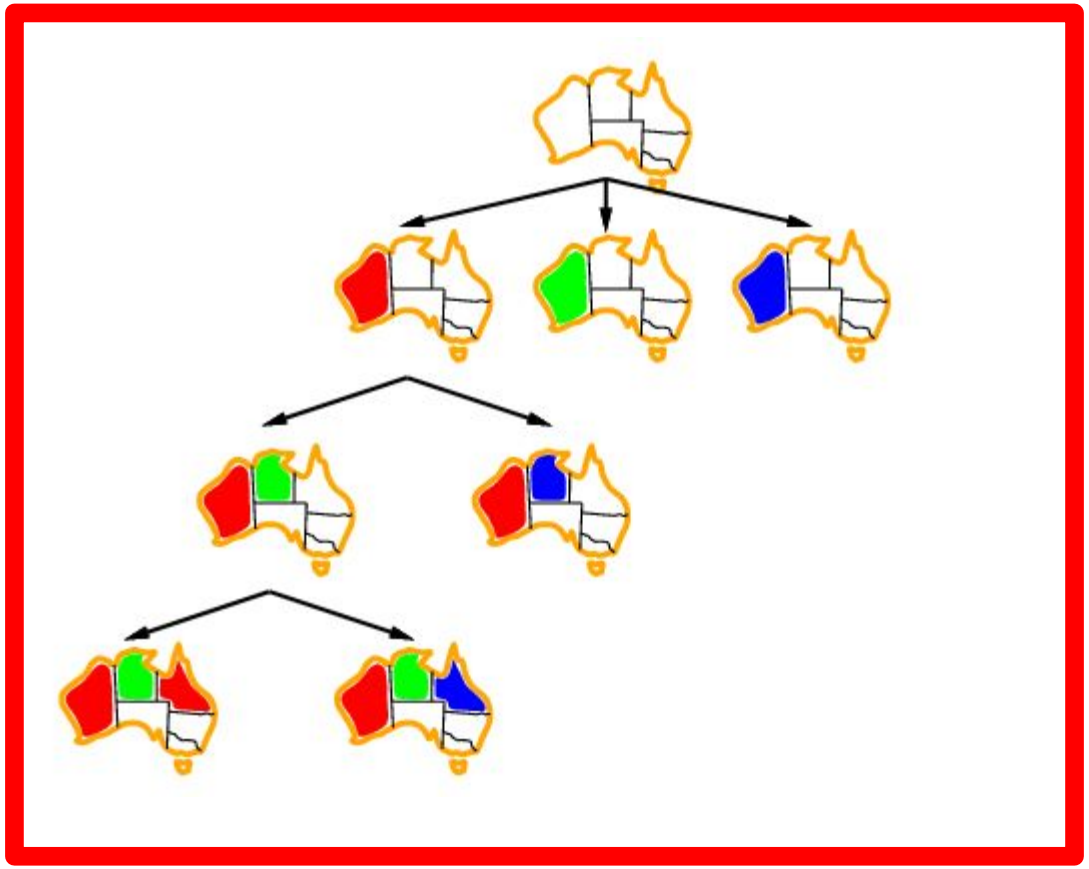
- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
  e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

Backtracking example

Backtracking example



Backtracking example



Algorithm for Backtracking

Pick initial state

R = set of all possible states

Select state with var assignment

Add to search space

check for con

If Satisfied

Continue

Else

Go to last Decision Point (DP)

Prune the search sub-space from DP

Continue with next decision option

If state = Goal State

Return Solution

Else

Continue

CSP-Backtracking, Role of heuristic

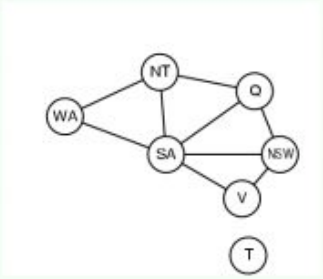
- Backtracking allows to go to the previous decision-making node to eliminate the invalid search space with respect to constraints
- Heuristics plays a very important role here
- If we are in position to determine which variables should be assigned next, then backtracking can be improved
- **Heuristics** help in deciding the initial state as well as subsequent selected states
- Selection of a variable with minimum number of possible values can help in simplifying the search
- This is called as Minimum Remaining Values Heuristic (MRV) or Most Constraint Variable Heuristic
- Restricts the most search which ends up in same variable (which would make the backtracking ineffective)

Heuristic

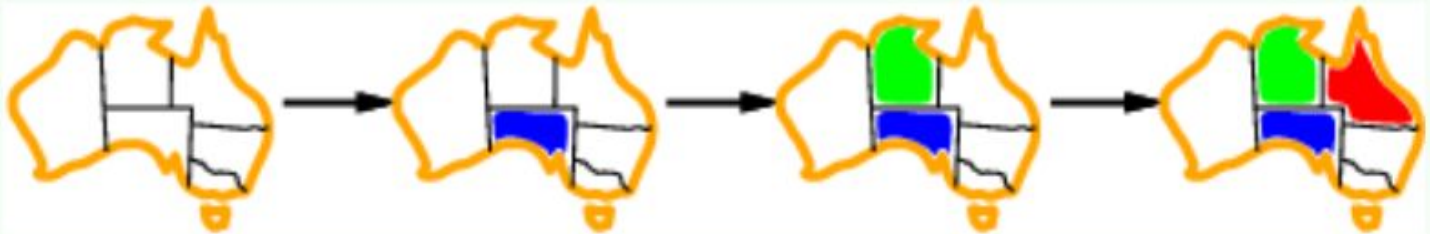
- MRV cannot have hold on initial selection process
- Node with maximum constraint is selected over other unassigned variables - Degree Heuristics
- By degree heuristics, branching factor can not be reduced
- Selection of variables are considered not the values for it, so the order in which the values of particular variable can be arranged is tackled by least constraining value heuristic

Heuristic – Most Constraining Variable

- Tie-breaker among most constrained variables

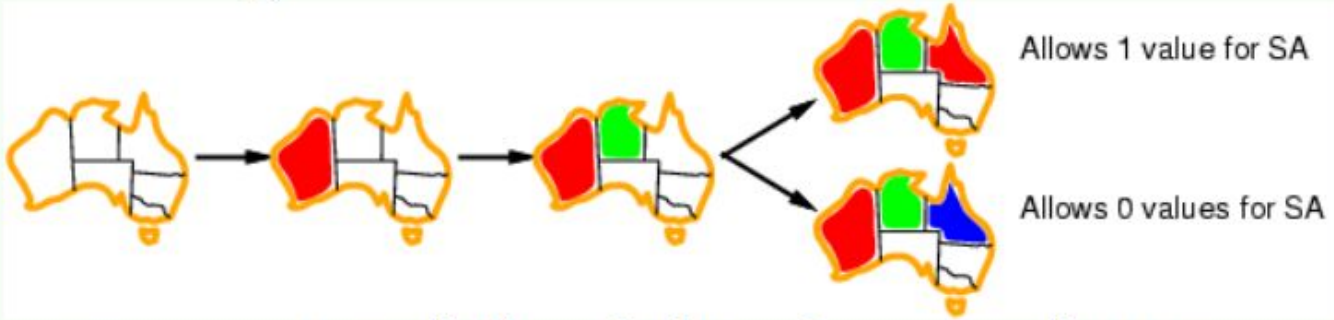


- Most constrain*ing* variable:
  - choose the variable with the most constraints on remaining variables (most edges in graph)



Heuristic- Minimum Remaining Values

- *Given a variable*, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables



- Leaves maximal flexibility for a solution.

Forward Checking

- To understand the forward checking, we shall see 4 Queens problem
- If an arrangement on the board of a queen *x* , hampers the position of $queens_{x+1}$, then this forward check ensures that the queen *x* should not be placed at the selected position and a new position is to be looked upon

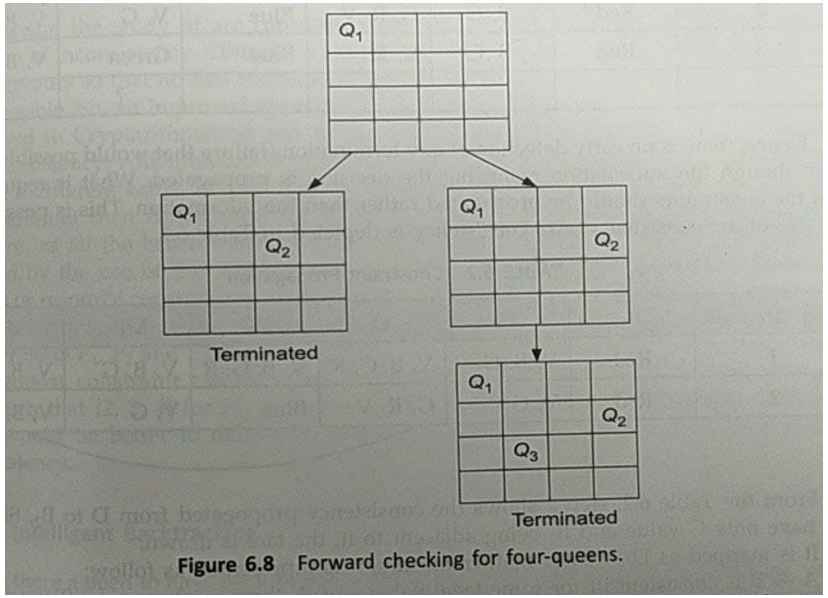


Figure 6.8 Forward checking for four-queens.

Forward Checking

- Q_1 and Q_2 are placed in row 1 nad 2 in the left sub-tree, so, search is halted, since No positions are left for Q_3 and Q_4
- Forward Checking keeps track of the next moves that are available for the unassigned variables
- The search will be terminated when there is no legal move available for the unassigned variables

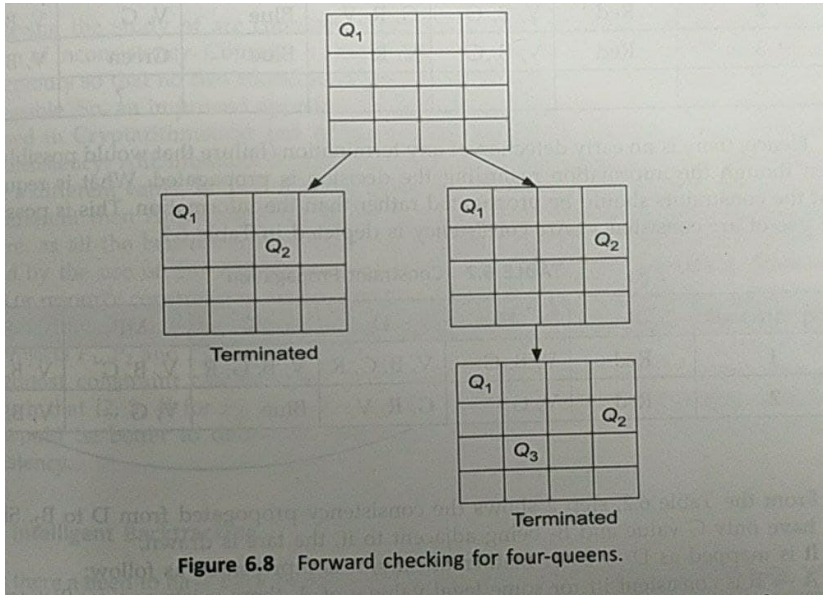


Figure 6.8 Forward checking for four-queens.

CSP- Room Coloring Problem
-CSP as a Search Problem

- Let *K* for Kitchen, *D* for Dining Room, *H* is for Hall, B_2 and B_3 are bedrooms 2 and 3, MB_1 is master bedroom, *SR* is the store Room, *GR* is Guest Room and *Lib* is Library
- Constraints
  - All bedrooms should not be colored red, only one can
  - No two adjacent rooms can have the same color
  - The colors available are red, blue, green and violet
  - Kitchen should not be colored green
  - Recommended to color the kitchen as blue
  - Dining room should not have violet color

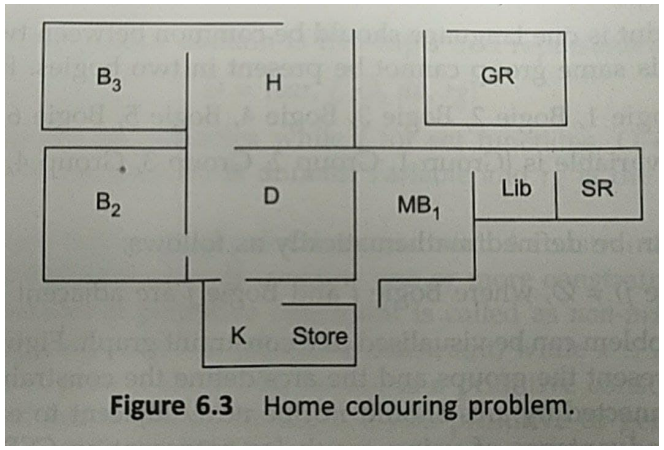


Figure 6.3 Home colouring problem.

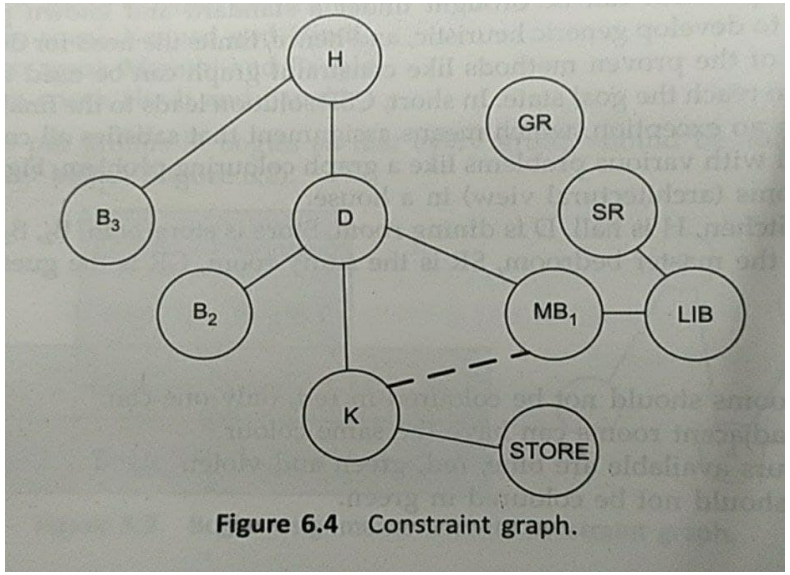


Figure 6.4 Constraint graph.

Forward Checking – Room Coloring Problem

- For Room Coloring problem, Considering all the constraints the mapping can be done in following ways;
  - At first, B_2 is selected with Red (R). Accordingly, R is deleted from the adjacent nodes
  - Kitchen is assigned with Blue (B). So, B is deleted form the adjacent Nodes
  - Furthermore, as MB_1 is selected green, no color is left for *D*.

TABLE 6.1 Decision and Information Propagation

Step number	B_3	H	D	K	MB_1	B_2
1	Red	V, B, G	V, B, G, R	V, B, G, R	V, B, G	V, B, G
2	Red	V, B, G	G, R, V	Blue	V, G	V, B, G
3	Red	V, B, G	V, R	Blue	Green	V, B
			?		?	

Constraint Propagation

- There is no early detection of any termination / failure that would possible occur even though the information regarding the decision is propagated
- Constraint should be propagated rather than the information

TABLE 6.2 Constraint Propagation

Step number	B_3	H	D	K	MB_1	B_2
1	Red	V, B, G	V, B, G, R	V, B, G, R	V, B, G	V, B, G
2	Red	V, G	G, R, ~~V~~	Blue	V, G	V, B, G

Constraint Propagation

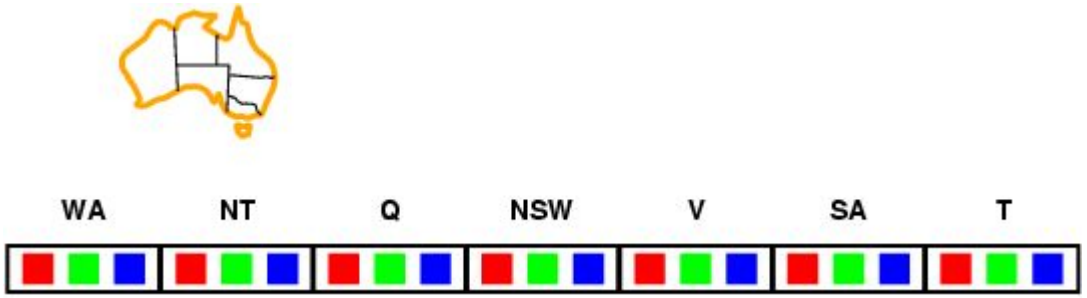
- Step 2 shows the consistency propagated from D to B_2
- Since D is can have only *G* value and B_2 being adjacent to it, the arc is drawn
- It is mapped as $D \rightarrow B_2$ or Mathematically,
  - $A \rightarrow B$ is consistent $\leftrightarrow$ $\forall$ legal value $a \in A$, $\exists$
    - non-conflicting value b $\in B$
- Failure detection can take place at early stage

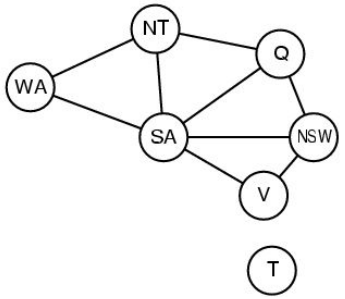
Algorithm for Arc Assignment

- Algorithm for arc assignment is:
  - Let C be the variable which is being assigned at a given instance
  - X will have some value from D{} where D is domain
  - For each and every assigned variable, that is adjacent to X, Say X^j
    - 1Perform forward check (remove values from domain D that conflict the decision of the current assignment)
    - 2For every other variable X^{jj} that are adjacent or connected to
    - X^j;
      - iRemove the values from D from X^{jj} that can't be taken as further unassigned variables
    - iiRepeat step 2, till no more values can be removed or discarded
  - Inconsistency is considered and constraints are propagated in Step (2)

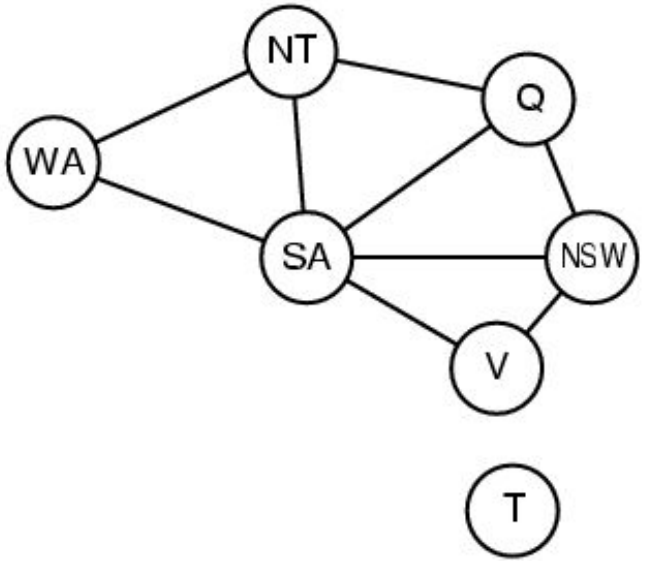
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



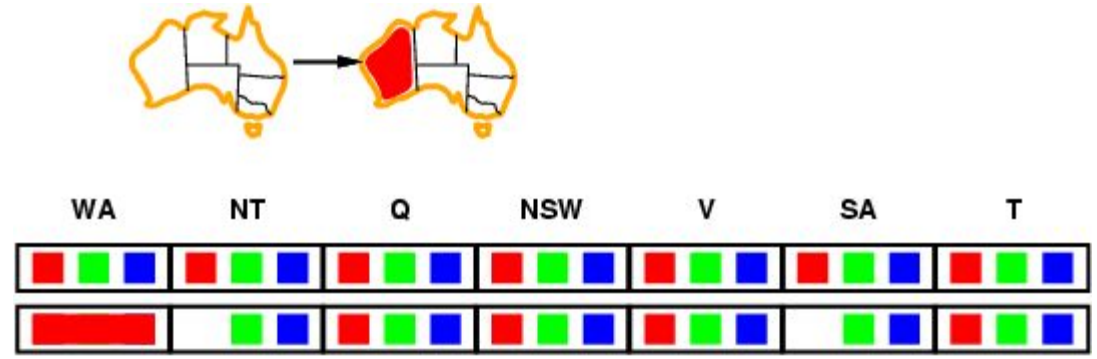


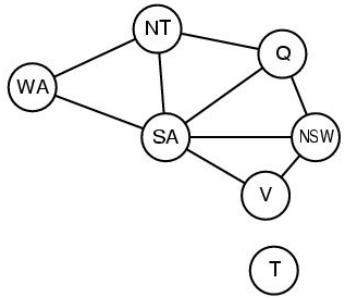
WA	RGB
NT	GB
SA	B
Q	R
NSW	G
Y	R
T	RGB



Forward checking

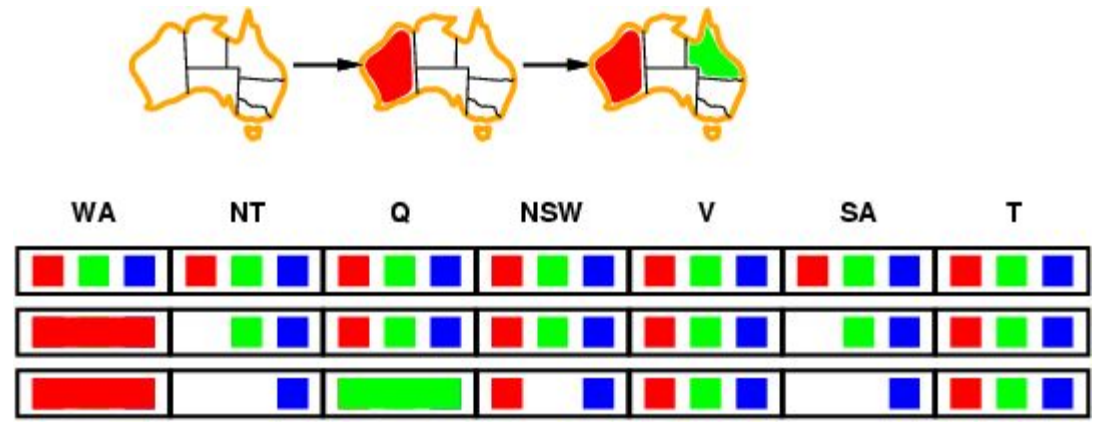
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

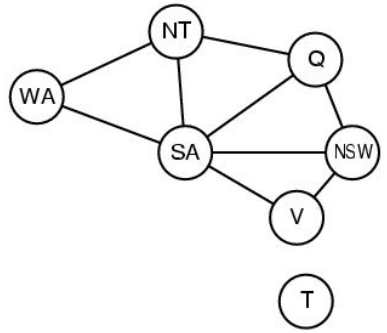




Forward checking

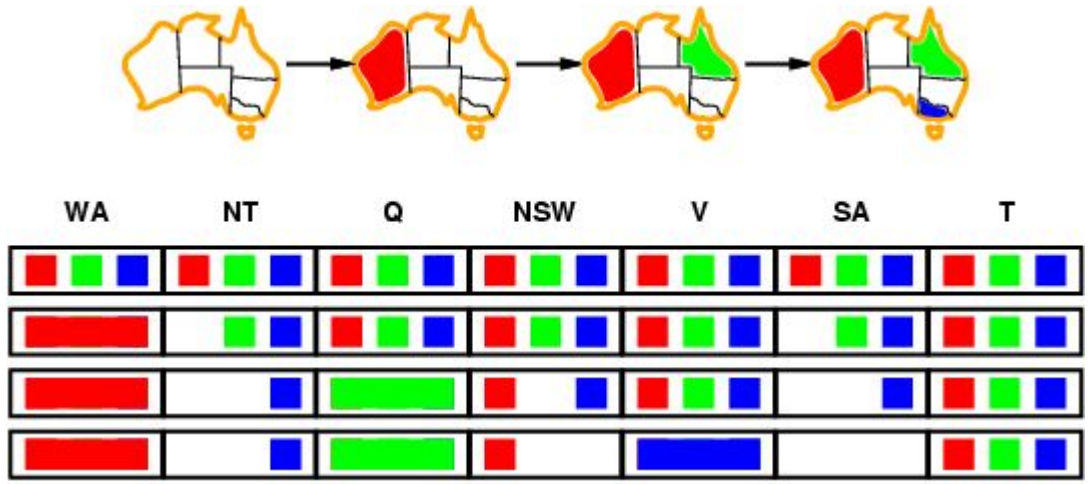
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

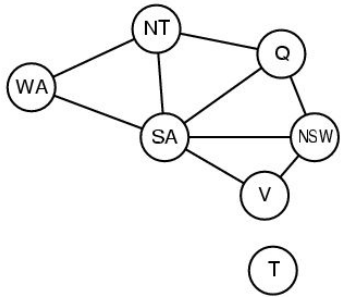




Forward checking

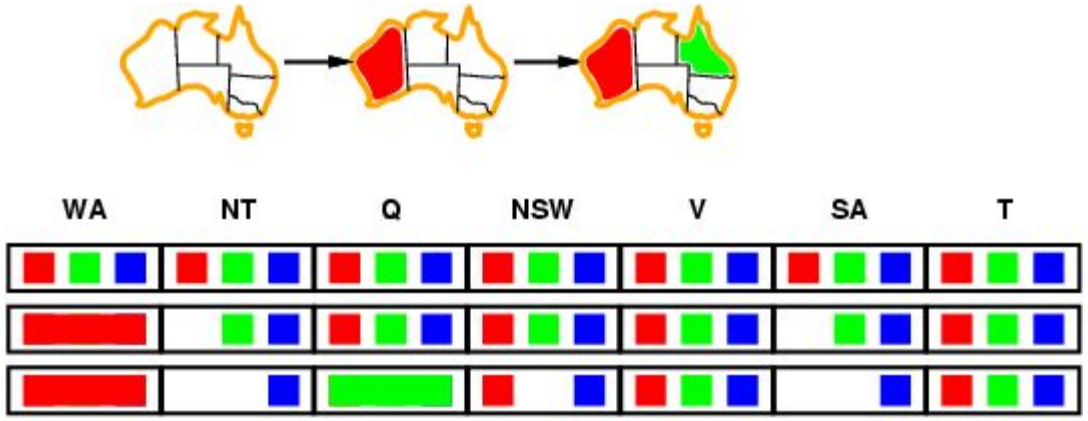
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

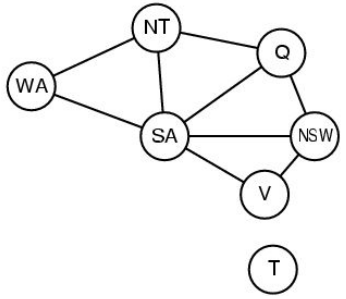




Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



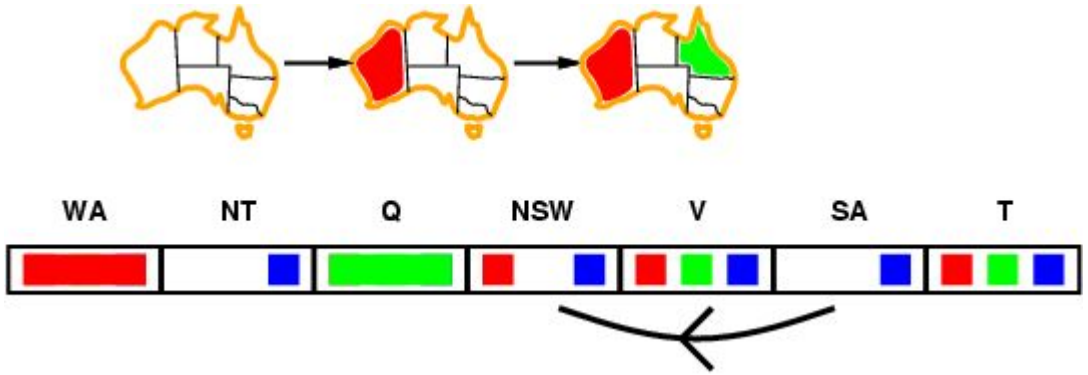


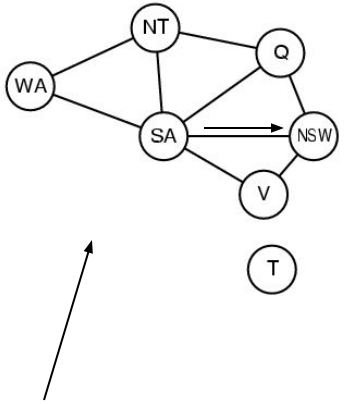
- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \square Y$ is consistent iff

  for every value x of X there is some allowed y



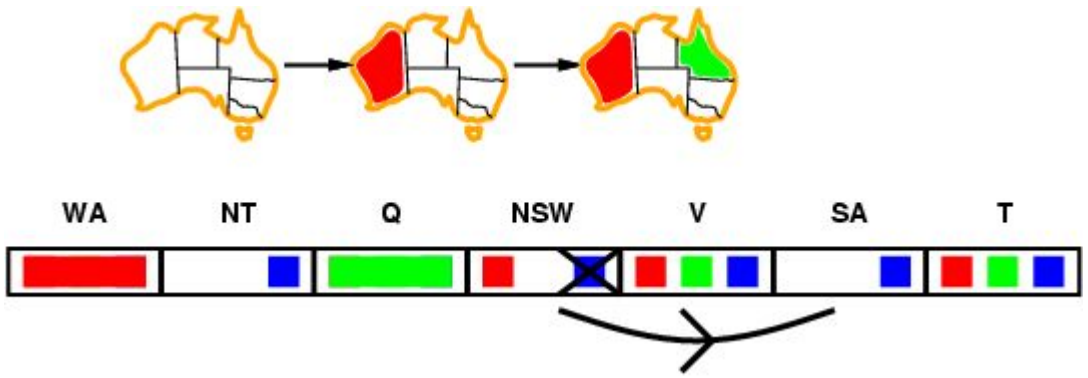


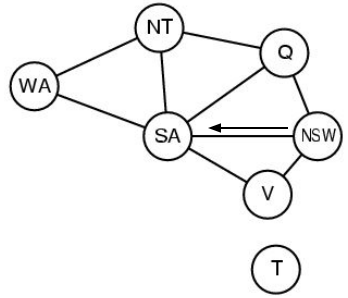
constraint propagation propagates arc consistency on the graph.

Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* □ *Y* is consistent iff

  for every value *x* of *X* there is some allowed *y*

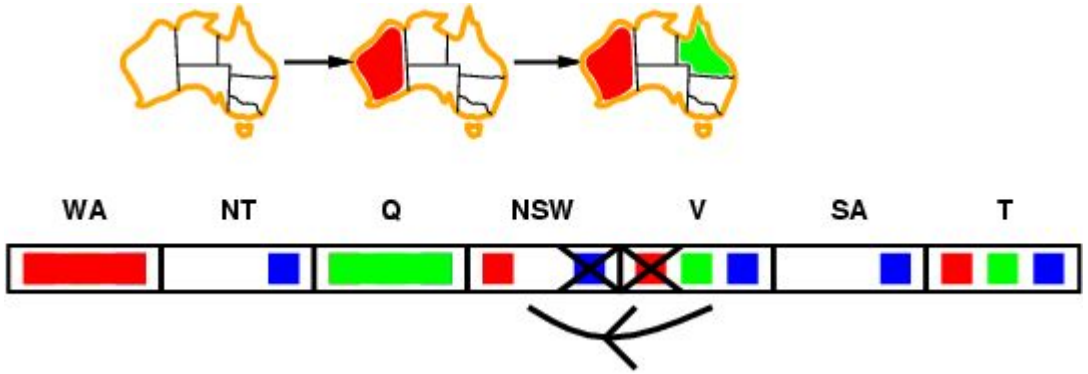


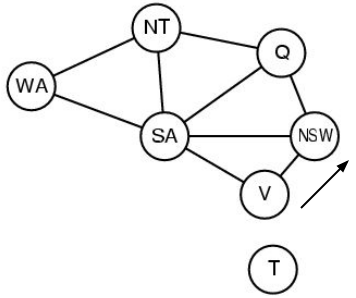


Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* □ *Y* is consistent iff

  for every value *x* of *X* there is some allowed *y*

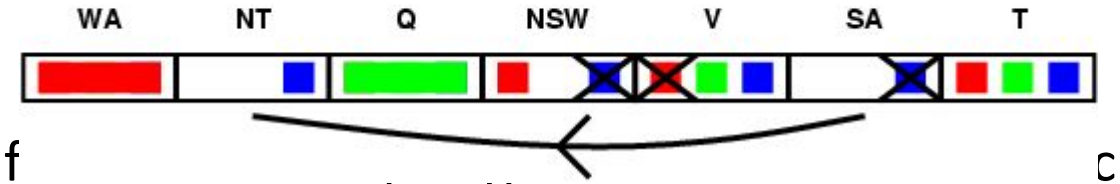


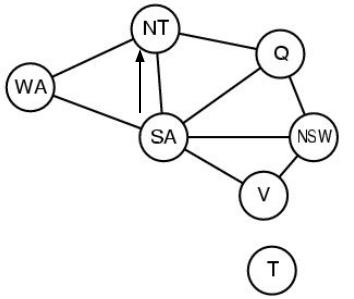


- If *X* loses a value, neighbors of *X* need to be rechecked

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \square Y$ is consistent iff
  for every value x of X there is some allowed y





- If checked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

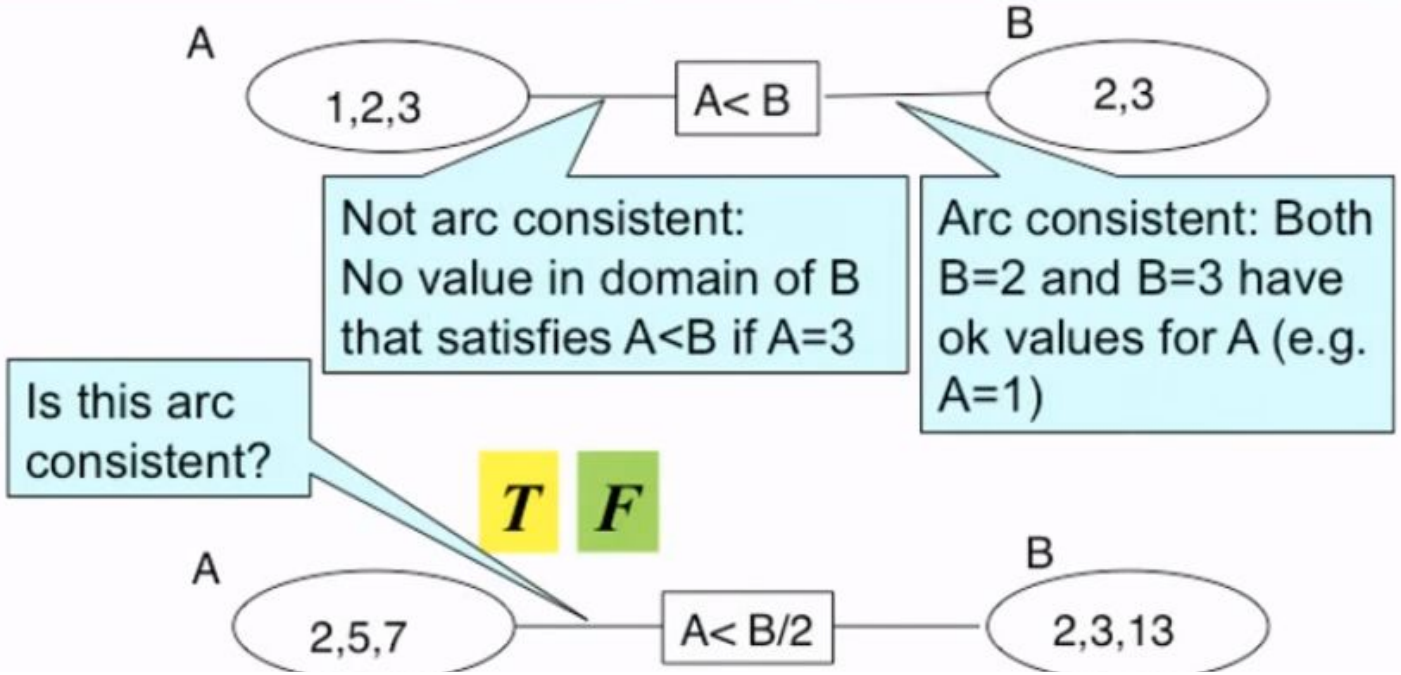
- Time complexity: $O(n^2d^3)$

Arc Consistency

Definition:

An arc <x, r(x,y)> is arc consistent if for each value x in dom(X) there is some value y in dom(Y) such that r(x,y) is satisfied.

A network is arc consistent if all its arcs are arc consistent.

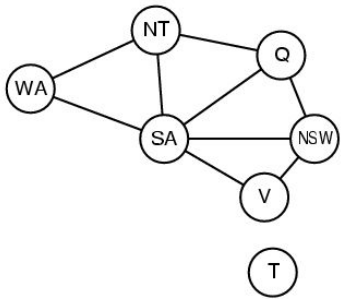


Intelligent Backtracking

- Conflict set is maintained using forward checking and maintained
- Considering the 4 Queens problem, Conflict needs to be detected by the user of conflict set so that a backtrack can occur
- Backtracking with respect to the conflict set is called as conflict-directed back jumping
- Back jumping approach can't actually restrict the earlier committed mistakes in some other branches

Intelligent Backtracking

- **Chronological backtracking:** The BACKGRACKING-SEARCH in which, when a branch of the search fails, back up to the preceding variable and try a different value for it. (The most recent decision point is revisited).
  - e.g: Suppose we have generated the partial assignment {Q=red, NSW=green, V=blue, T=red}.
  - When we try the next variable SA, we see every value violates a constraint.
  - We back up to T and try a new color, it cannot resolve the problem.



- **Intelligent backtracking:** Backtrack to a variable that was responsible for making one of the possible values of the next variable (e.g. SA) impossible.

Conflict set for a variable: A set of assignments that are in conflict with some value for that variable.
(e.g. The set {Q=red, NSW=green, V=blue} is the conflict set for SA.)
Backjumping method: Backtracks to the most recent assignment in the conflict set.
(e.g. backjumping would jump over T and try a new value for V.)

Thank You