## UNIT V

Advanced Topics: Game Playing: Minimax search procedure-Adding alpha-beta cutoffs- Expert System: Representation-Expert System shells-Knowledge Acquisition. Robotics: Hardware- Robotic Perception-Planning-Application domains

## Advanced Topics:

**GAME PLAYING**

* The term *Game* means a sort of *conflict* in which n individuals or groups (known as players) participate.
* Game theory denotes games of *strategy*.
* John von Neumann is acknowledged as father of game theory. Neumann defined Game theory in 1928 and 1937 and established the mathematical framework for all subsequent theoretical developments.
* Game theory allows decision-makers (players) to cope with other decision-makers (players) who have different purposes in mind. In other words, players determine their own strategies in terms of the strategies and goals of their opponent. Games are integral attribute of human beings.
* Games engage the *intellectual faculties* of humans.
* If computers are to mimic people they should be able to play games.

## Definition of Game

* A game has at least *two players*.
* Solitaire is not considered a game by game theory.
* The term 'solitaire' is used for single-player games of concentration.
* An instance of a game begins with a player choosing from a set of specified *(game rules)*

alternatives. This choice is called a *move*.

* After first move, the new situation determines which player to make next move and alternatives available to that player.
	+ In many board games, the next move is by other player.
	+ In many multi-player card games, the player making next move depends on who dealt, who took last trick, won last hand, etc.
* Minimax - The least good of all good outcomes.
* Maximin - The least bad of all bad outcomes**.**

The primary game theory is the Mini-Max Theorem. This theorem says :

"If a Minimax of one player corresponds to a Maximin of the other player, then that outcome is the best both players can hope for."



## MINIMAX SEARCH PROCEDURE

The minimax search procedure is a depth first, depth limited search procedure. The idea is to start at the current position and use the plausible move generator to generate the set of possible successor positions.

Consider two players, zero sum, non-random, perfect knowledge games. Examples: Tic-Tac-Toe, Checkers, Chess, Go, Nim, and Othello.

## Formalizing Game

A general and a Tic-Tac-Toe game in particular. Consider 2-Person, Zero-Sum, Perfect Information

Both players have access to complete information about the state of the game. No information is hidden from either player.

Players alternately move. Apply Iterative methods

Required because search space may be large for the games to search for a solution. Do search before each move to select the next best move.

Adversary Methods

Required because alternate moves are made by an opponent , who is trying to win, are not controllable.

Static Evaluation Function f(n)

‡ Used to evaluate the "goodness" of a configuration of the game. It estimates board quality leading to win for one player.

Example: Let the board associated with node n then

If f(n) = large +ve value means the board is good for me and bad for opponent.

If f(n) = large -ve value means the board is bad for me and good for opponent. If f(n) near 0 Means the board is a neutral position.

If f(n) = +infinity means a winning position for me.

If f(n) = -infinity means a winning position for opponent.

* Games hold an inexplicable fascination for many people, and the notion that computers might play games has existed at least as long as computers.
* Charles Babbage the nineteenth century computer architect, thought about programming his analytical engine to play chess and later of building a machine to play tic tac toe.
* Two of the pioneers of the science of information and computing contributed to the fledgling computer game playing literature.
* Claude Shannon [1950] wrote a paper in which he described mechanisms that could be used in a program to play chess.
* A few years later, Alan Turing described a chess playing program, although he never built it.

There are two main reasons that games appeared to be a good domain in which to explore machine intelligence.

1. They provide a structured task in which it is very easy to measure success or failure.
2. They did not obviously require large amounts of knowledge. For example in chess:
3. The average branching factor is around 35.
4. In a average game, each player might makes 50 moves.
5. So in order to examine complete game tree, we would have to examine 35 100 positions.

## Adversarial Search

Competitive environments, in which the agents’ goals are in conflict, give rise to adversarial search problems- often known as games. In AI, “games” are usually of a rather specialized kind

* in which there are two agents whose actions must alternate and in which the utility values at the end of the game are always equal and opposite.

A game can be formally defined as a kind of search problem with the following components:

* + The initial state, which includes the board position and identifies the player to move.
	+ A successor function, which returns a list of (move, state )pairs, each indicating a legal move and the resulting state.
	+ A terminal test, which determines when the game is over. States where the game has ended are called terminal states.
	+ A utility function, which gives a numeric value for the terminal states.

The initial state and the legal moves for each side define the game tree for the game. The following figure shows part of the game tree for tic-tac-toe. From the initial state, MAX has nine possible moves. Play alternates between MAX’s placing an X and MIN’s placing an O until we reach leaf nodes corresponding to terminal states such that one player has three in a row or all the squares are filled. Game tree (2-player, deterministic, turns)

## Minimax

Given a game tree, the optimal strategy can be determined by examining the minimax value of each node, which we write as MINIMAX-VALUE(n). The minimax value of a node is the utility of being in the corresponding state, assuming that both players play optimally from there to the end of the game.

* Perfect play for deterministic games
* Idea: choose move to position with highest minimax value

= best achievable payoff against best play

* E.g., 2-ply game:

***CST71 Artificial Intelligence, Dept of CSE, MVIT***

Here, the first MIN node, labeled B, has three successors with values 3,12, and 8, so its minimax value is 3. The minimax algorithm computes the minimax decision from the current state.

## Minimax algorithm

**Properties of minimax**

* Complete? Yes (if tree is finite)
* Optimal? Yes (against an optimal opponent)
* Time complexity? O(bm):the maximum depth of the tree is m, and there are b legal moves at each point
	+ Space complexity? O(bm) (depth-first
* With "perfect ordering," time complexity = O(bm/2)

 doubles depth of search

* exploration)
* For chess, b ≈35, m ≈100 for "reasonable" games

 exact solution completely infeasible

**EXAMPLE: *Game playing with Mini-Max STEP:1***



## Game Playing with Mini-Max - Tic-Tac-Toe STEP:2



**STEP:3**



## STEP:4



**Static Evaluation:**

## ‘+1’ for a win, ‘0’ for a draw

**Criteria ‘+1’ for a Win, ‘0’ for a Draw**



## Back-up the Evaluations:

**Level by level, on the basis of opponent's turn Up : One Level**

## ALPHA-BETA PRUNING

The problem with minimax procedure is that the number of game states it has to examine is exponential in the number of moves. We can cut it in half using the technique called alpha-

beta pruning. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Consider again the two-ply game tree. The steps are explained in the following figure. The outcome is that we can identify the minimax decision without ever evaluating two of the leaf nodes.

## α-β pruning example

The value of the root node is given by

MINIMAX-VALUE(root) = max(min(3,12,8), min(2,x,y), min(14,5,2))

= max(3,min(2,x,y),2)

=max(3,z,2) where z<=2

=3

x and y: two unevaluated successors z: minimum of x and y

## Properties of α-β

* Pruning does not affect final result
* Good move ordering improves effectiveness of pruning
* A simple example of the value of reasoning about which computations are relevant (a form of meta reasoning)

## Why is it called α-β?

* α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
* β is the value of the best (i.e., lowest-value) choice found so far at any choice point along the path for *min*

 If *v* is worse than α, *max* will avoid it prune that branch

Alpha-beta search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN respectively. The effectiveness of alpha-beta pruning is highly dependent on the order in which the successors are examined. The algorithm is given below:

The α-β algorithm


## EXPERT SYSTEM ITS REPRESENTATION

Expert Systems (ES), also called Knowledge-Based Systems (KBS) or simply Knowledge Systems (KS), are computer programs that use expertise to assist people in performing a wide variety of functions, including diagnosis, planning, scheduling and design. ESs are distinguished from conventional computer programs in two essential ways (Barr, Cohen et al. 1989):

* 1. Expert systems reason with domain-specific knowledge that is symbolic as well as numerical;
	2. Expert systems use domain-specific methods that are heuristic (i.e., plausible) as well as algorithmic.

The technology of expert systems has had a far greater impact than even the expert systems business. Expert system technology has become widespread and deeply embedded. As expert system techniques have matured into a standard information technology, the most important recent trend is the increasing integration of this technology with conventional information processing, such as data processing or management information systems.

## The Building Blocks of Expert Systems

Every expert system consists of two principal parts: the knowledge base; and the reasoning, or inference, engine.

* + - The *knowledge base* of expert systems contains both factual and heuristic knowledge. *Factual knowledge* is that knowledge of the task domain that is widely shared, typically found in textbooks or journals, and commonly agreed upon by those knowledgeable in the particular field.
		- *Heuristic knowledge* is the less rigorous, more experiential, more judgmental knowledge of performance.
		- In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic.
		- It is the knowledge of good practice, good judgment, and plausible reasoning in the field. It is the knowledge that underlies the "art of good guessing."
		- *Knowledge representation* formalizes and organizes the knowledge. One widely used representation is the *production rule*, or simply *rule*.
		- A rule consists of an IF part and a THEN part (also called a *condition* and an *action*). The IF part lists a set of conditions in some logical combination.
		- The piece of knowledge represented by the production rule is relevant to the line of reasoning being developed if the IF part of the rule is satisfied; consequently, the THEN part can be concluded, or its problem-solving action taken.
		- Expert systems whose knowledge is represented in rule form are called *rule-based systems*.
		- Another widely used representation, called the *unit* (also known as *frame*, *schema*, or *list structure*) is based upon a more passive view of knowledge.
		- The unit is an assemblage of associated symbolic knowledge about an entity to be represented. Typically, a unit consists of a list of properties of the entity and associated values for those properties.

Since every task domain consists of many entities that stand in various relations, the properties can also be used to specify relations, and the values of these properties are the names of other units that are linked according to the relations. One unit can also represent knowledge that is a "special case" of another unit, or some units can be "parts of" another unit.

The *problem-solving model,* or *paradigm*, organizes and controls the steps taken to solve the problem. One common but powerful paradigm involves chaining of IF-THEN rules to form a line of reasoning. If the chaining starts from a set of conditions and moves toward some conclusion, the method is called *forward chaining*. If the conclusion is known (for example, a goal to be achieved) but the path to that conclusion is not known, then reasoning backwards is called for, and the method is *backward chaining*. These problem-solving methods are built into program modules called *inference engines* or *inference procedures* that manipulate and use knowledge in the knowledge base to form a line of reasoning.

* + - The most important ingredient in any expert system is knowledge. The power of expert systems resides in the specific, high-quality knowledge they contain about task domains.
		- AI researchers will continue to explore and add to the current repertoire of knowledge representation and reasoning methods. But in knowledge resides the power.
		- Because of the importance of knowledge in expert systems and because the current knowledge acquisition method is slow and tedious, much of the future of expert systems depends on breaking the knowledge acquisition bottleneck and in codifying and representing a large knowledge infrastructure.

## EXPERT SYSTEM SHELL

* + - A rule-based, expert system maintains a separation between its Knowledge-base and that part of the system that executes rules, often referred to as the ***expert system shell***.
		- The system shell is indifferent to the rules it executes. This is an important distinction, because it means that the expert system shell can be applied to many different problem domains with little or no change.
		- It also means that adding or modifying rules to an expert system can effect changes in program behavior without affecting the controlling component, the system shell.
		- The language used to express a rule is closely related to the language ***subject matter experts*** use to describe problem solutions.
		- When the subject matter expert composes a rule using this language, he is, at the same time, creating a written record of problem knowledge, which can then be shared with others.
		- Thus, the creation of a rule kills two birds with one stone; the rule adds functionality or changes program behavior, and records essential information about the problem domain in a human-readable form. Knowledge captured and maintained by these systems ensures continuity of operations even as subject matter experts (i.e., mathematicians, accountants, physicians) retire or transfer.
		- Furthermore, changes to the Knowledge-base can be made easily by subject matter experts without programmer intervention, thereby reducing the cost of software maintenance and helping to ensure that changes are made in the way they were intended.
		- Rules are added to the knowledge-base by subject matter experts using text or graphical editors that are integral to the system shell. The simple process by which rules are added to the knowledge-base is depicted in Figure 1.



Finally, the expert system never forgets, can store and retrieve more knowledge than any single human being can remember, and makes no errors, provided the rules created by the subject matter experts accurately model the problem at hand.

## EXPERT SYSTEM ARCHITECTURE

An expert system is, typically, composed of two major components, the Knowledge-base and the Expert System Shell.

The Knowledge-base is a collection of rules encoded as *metadata* in a file system, or more often in a relational database. The Expert System Shell is a problem-independent component housing facilities for creating, editing, and executing rules. A software architecture for an expert system is illustrated in Figure 2.



The **shell** portion includes software modules whose purpose it is to,

* + - * Process requests for service from system users and application layer modules;
			* Support the creation and modification of business rules by subject matter experts;
			* Translate business rules, created by a subject matter experts, into machine-readable forms;
			* Execute business rules; and
			* Provide low-level support to expert system components (e.g., retrieve metadata from and save metadata to knowledge base, build Abstract Syntax Trees during rule translation of business rules, etc.).

## ROBOTICS:

**Definition**

"A reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks"

## A robot has these essential characteristics:

* + - * **Sensing:** First of all your robot would have to be able to sense its surroundings. It would do this in ways that are not similar to the way that you sense your surroundings. Giving your robot [sensors](http://www.thetech.org/exhibits_events/online/robotics/universal/page09.html): light sensors (eyes), touch and pressure sensors (hands), [chemical](http://www.nature.com/nsu/010607/010607-3.html)

sensors (nose), [hearing and sonar sensors](http://www.militaryaudiology.org/) (ears), and [taste sensors](http://www.nature.com/nsu/020107/020107-3.html) (tongue) will give your robot awareness of its environment.

* + - * **Movement:** A robot needs to be able to move around its environment. Whether rolling on wheels, walking on legs or propelling by thrusters a robot needs to be able to move. To count as a robot either the whole robot moves, like the Sojourner or just parts of the robot moves, like the Canada Arm.
			* **Energy:** A robot needs to be able to power itself. A robot might be solar powered, electrically powered, battery powered. The way your robot gets its energy will depend on what your robot needs to do.
			* **Intelligence:** A robot needs some kind of "smarts." This is where programming enters the pictures. A programmer is the person who gives the robot its 'smarts.' The robot will have to have some way to receive the program so that it knows what it is to do.

## ROBOTICS HARDWARE

* + - 1. **Sensors**

Sensors are the perceptual interface between robots and their environments. Passive sensors. Such as cameras are true observers of the environment: they capture signals that are generated by other sources in the environment. Active sensors, such as sonar, send energy into the environment.

Many mobile robots make use of range finders, which are sensors that measure the distance to nearby objects. One common type is the sonar sensor, also known as and ultrasonic transducer.

Some range sensors measure very short or very long distances. Close-range sensors include tactile sensors such as whiskers, bumb panels, and touch-sensitive skin. At the other end of the spectrum is the Global Positioning System (GPS), which measures the distance to satellites that emit pulsed signals.

## Effectors

Effectors are the means by which robots move and change the shape of their bodies. To understand the design of effectors,, it will help to talk about motion and shape in the abstract, using the concept of a degree of freedom(DOF).

The dynamic state of a robot includes one additional dimension for the rate of change of each kinematic dimension.

The arm in Figure 25.3(a) has exactly six degrees of freedom. Created by five revolute joints that generate rotational motion and one prismatic joint that generates sliding motion.

The car has 3 effective degrees of freedom but 2 controllable degress of freedom. We say a robot is nonholonomic if it has more effective DOF s than controllable DOFs and holonomic if the two numbers are the same.

For mobile robots, there exists a range of mechanisms for locomotion, including wheels,tracks and legs. Differential drive robots possess two independently actuated wheels.

## ROBOTIC PERCEPTION

Perception is the process by which robots map sensor measurements into internal representations of the environment. Perception is difficult because in general the sensors are noisy, and the environment is partially observable, unpredictable, and often dynamic. As a rule of thumb, good internal representations have three properties:

* + - contain enough information for the robot to make the right decisions,
		- structured such that can be updated efficiently
		- Natural in the sense that internal variable correspond to natural state variables in the physical world.

# Localization:

* + - Localization is a generic example of robot perception. It is the problem of determining where things are.
		- Localization is one of the most pervasive perception problems in robotics, because knowledge about where things are is at the core of any successful physical interaction.
		- For example , robot manipulators must know the location of objects they manipulate. Navigating robots must know where they are in order to find their way to goal locations.
		- The localization problem comes in three flavours of increasing difficulty. If the initial pose of the object to be localized is known, localizations is a **tracking** problem.
		- Tracking problems are characterized by bounded uncertainty. More difficult is the global localization problem, in which the initial location of the object is entirely unknown.
		- Global localization problems turn into tracking problems once the object of interest has been localized, but they also involve phases where the robot has to manage very broad uncertainties.
		- Finally, we can be mean to our robot and “KIDNAP” the object it is attempting to localize. Localization under such devious conditions is known as the Kidnapping problem.
		- Kidnapping is often used to test the robustness of a localization technique under extreme conditions.

# Mapping:

In the literature, the robot mapping problem is often referred to as **simultaneous localization and mapping**, abbreviated as SLAM. Not only must the robot construct a map, it must do so without knowing where it is. SLAM is one of the core problems in robotics.

## PLANNING TO MOVE

In robotics, decisions ultimately involve motion of effectors.

* + - The **point-to-point motion problem** is to deliver the robot or its end-effectors to a designated target location.
		- A greater challenge is the **complaint motion problem**, in which a robot moves while being physical contact with an obstacle .
		- An example of compliant motion is a robot manipulator that screws in a light bulb , or a robot that pushes a box across a table top.
		- We begin by finding a suitable representation in which motion planning problems can be described and solved.
		- It turns out that the configuration space – the space of robot states defined by location orientation, and joint angles – is a better place to work than the original 3D space.
		- The **path planning problem** is to find a path from one configuration to another in configuration space.

The literature on robot path planning distinguishes a range of different techniques specifically aimed at finding paths in high-dimensional continuous spaces.

The major families of approaches are known as **cell decomposition and skeletonization**. Each reduces the continuous path-planning problem to a discrete graph search problem by identifying some canonical states and paths within the free space.