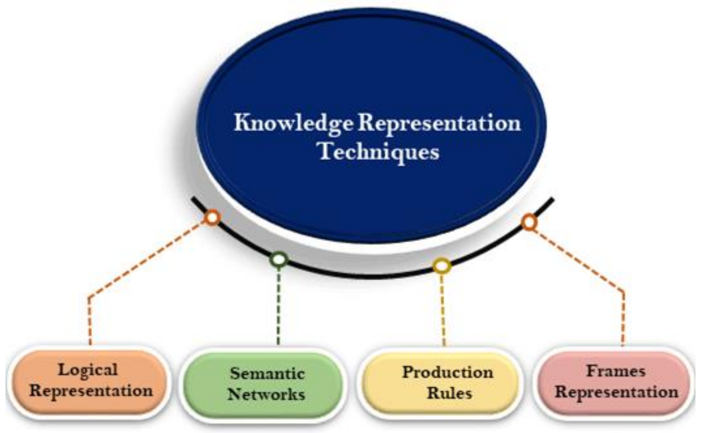
<u>Unit IV</u>

Structure Representation of Knowledge

Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. **Semantic Network Representation**
3. **Frame Representation**
4. Production Rules



1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorized into mainly two logics:

a. Propositional Logics
b. Predicate logics

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

2. Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

This representation consists of mainly two types of relations:

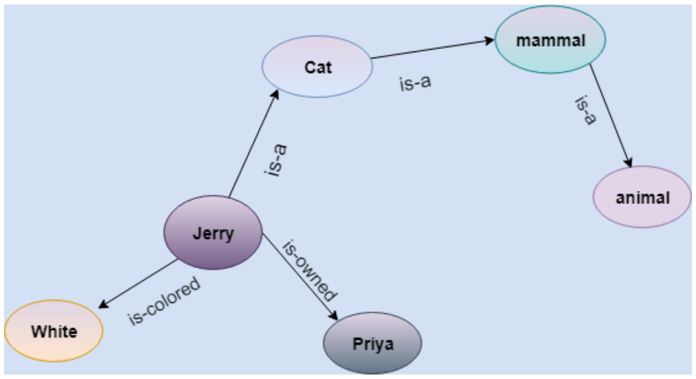
a. IS-A relation (Inheritance)
b. Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:

a. Jerry is a cat.
b. Jerry is a mammal
c. Jerry is owned by Priya.

d. Jerry is brown colored.
e. All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

Drawbacks in Semantic representation:

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.
5. These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Example: 1

Let's take an example of a frame for a book

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter

Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

Example:

- **IF (at bus stop AND bus arrives) THEN action (get into the bus)**
- **IF (on the bus AND paid AND empty seat) THEN action (sit down).**
- **IF (on bus AND unpaid) THEN action (pay charges).**
- **IF (bus arrives at destination) THEN action (get down from the bus).**

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

Conceptual Dependency in Artificial Intelligence

Conceptual Dependency:

In 1977, Roger C. Schank has developed a Conceptual Dependency structure. The Conceptual Dependency is used to represent knowledge of Artificial Intelligence. It should be powerful enough to represent these concepts in the sentence of natural language. It states that different sentence which has the same meaning should have some unique representation. There are 5 types of states in Conceptual Dependency:

1. Entities

2. Actions

3. Conceptual cases

4. Conceptual dependencies

5. Conceptual tense

Main Goals of Conceptual Dependency:

1. It captures the implicit concept of a sentence and makes it explicit.

2. It helps in drawing inferences from sentences.

3. For any two or more sentences that are identical in meaning. It should be only one representation of meaning.

4. It provides a means of representation which are language independent.

5. It develops language conversion packages.

Rules of Conceptual Dependency:

Rule-1: It describes the relationship between an actor and the event he or she causes.

Rule-2: It describes the relationship between PP and PA that are asserted to describe it.

Rule-3: It describes the relationship between two PPs, one of which belongs to the set defined by the other.

Rule-4: It describes the relationship between a PP and an attribute that has already been predicated on it.

Rule-5: It describes the relationship between two PPs one of which provides a particular kind of information about the other.

Rule-6: It describes the relationship between an ACT and the PP that is the object of that ACT.

Rule-7: It describes the relationship between an ACT and the source and the recipient of the ACT.

Rule-8: It describes the relationship between an ACT and the instrument with which it is performed. This instrument must always be a full conceptualization, not just a single physical object.

Rule-9: It describes the relationship between an ACT and its physical source and destination.

Rule-10: It represents the relationship between a PP and a state in which it started and another in which it ended.

Rule-11: It represents the relationship between one conceptualization and another that causes it.

Rule-12: It represents the relationship between conceptualization and the time at which the event occurred described.

Rule-13: It describes the relationship between one conceptualization and another, that is the time of the first.

Rule-14: It describes the relationship between conceptualization and the place at which it occurred.

Game Playing in Artificial Intelligence

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game. Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS (Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedure that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

Game playing is a popular application of artificial intelligence that involves the development of computer programs to play games, such as chess, checkers, or Go. The goal of game playing in artificial intelligence is to develop algorithms that can learn how to play games and make decisions that will lead to winning outcomes.

1. One of the earliest examples of successful game playing AI is the chess program Deep Blue, developed by IBM, which defeated the world champion Garry Kasparov in 1997. Since then, AI has been applied to a wide range of games, including two-player games, multiplayer games, and video games.

There are two main approaches to game playing in AI, rule-based systems and machine learning-based systems.

1. **Rule-based systems** use a set of fixed rules to play the game.
2. **Machine learning-based systems** use algorithms to learn from experience and make decisions based on that experience.

In recent years, machine learning-based systems have become increasingly popular, as they are able to learn from experience and improve over time, making them well-suited for complex games such as Go. For example, AlphaGo, developed by DeepMind, was the first machine learning-based system to defeat a world champion in the game of Go.

Game playing in AI is an active area of research and has many practical applications, including game development, education, and military training. By simulating game playing scenarios, AI algorithms can be used to develop more effective decision-making systems for real-world applications.

The most common search technique in game playing is **Minimax search procedure**. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

Minimax algorithm uses two functions –

MOVEGEN: It generates all the possible moves that can be generated from the current position.

STATICEVALUATION: It returns a value depending upon the goodness from the viewpoint of two-player

This algorithm is a two player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1 the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has make the best move. It is a recursive algorithm, as same procedure occurs at each level.

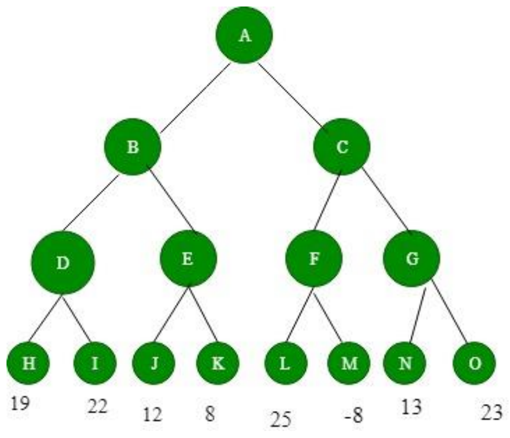


Figure 1: Before backing-up of values

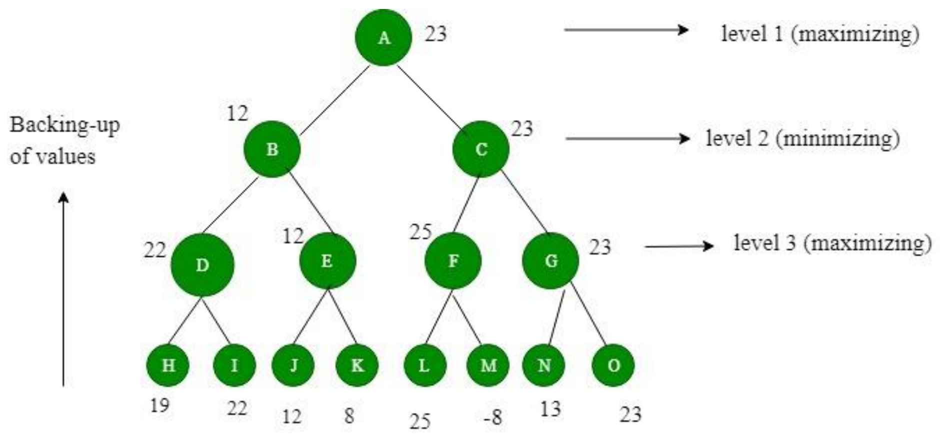


Figure 2: After backing-up of values We assume that PLAYER1 will start the game.

4 levels are generated. The value to nodes H, I, J, K, L, M, N, O is provided by STATICEVALUATION function. Level 3 is maximizing level, so all nodes of level 3 will take maximum values of their children. Level 2 is minimizing level, so all its nodes will take minimum values of their children. This process continues. The value of A is 23. That means A should choose C move to win.

Reference: Artificial Intelligence by Rich and Knight

Advantages of Game Playing in Artificial Intelligence:

1. **Advancement of AI:** Game playing has been a driving force behind the development of artificial intelligence and has led to the creation of new algorithms and techniques that can be applied to other areas of AI.
2. **Education and training:** Game playing can be used to teach AI techniques and algorithms to students and professionals, as well as to provide training for military and emergency response personnel.
3. **Research:** Game playing is an active area of research in AI and provides an opportunity to study and develop new techniques for decision-making and problem-solving.
4. **Real-world applications:** The techniques and algorithms developed for game playing can be applied to real-world applications, such as robotics, autonomous systems, and decision support systems.

Disadvantages of Game Playing in Artificial Intelligence:

1. **Limited scope:** The techniques and algorithms developed for game playing may not be well-suited for other types of applications and may need to be adapted or modified for different domains.
2. **Computational cost:** Game playing can be computationally expensive, especially for complex games such as chess or Go, and may require powerful computers to achieve real-time performance.

Introduction

We have already seen the MiniMax algorithm as well as need for improvement using alpha beta pruning in previous module. We will see how we can write a modified algorithm incorporating alpha beta pruning in this module. We will also see how we can improve the performance of the algorithm using some other measures.

The algorithm that we propose here is again a recursive algorithm with two more parameters. One is used for checking the threshold value against the nodes heuristic values and another is to set threshold for the next level; as we have different threshold values of Min and Max levels. The process of the cutoff is applied after each node is examined and children are returned with their MiniMax values.

If the ordering of the moves is perfect, they can save exponential amount of nodes to be explored. We will discuss an output of one such study at the end.

MiniMax with Alpha Beta Pruning

Now we will see how the Mini Max algorithm that we have presented in the previous module can be modified to include alpha beta pruning. We have already seen that the cutoff we apply at maximizing level is called beta and minimizing level is called alpha. While we are exploring a minimizing level, we can avoid a move early if we find that the values are greater than the current threshold as we know that opponent is unlikely to choose that move. Ruling out a move by us (a maximizing player), happens while searching at minimizing layer. Alpha cutoffs are applied *by the maximizing player*, cuts off *moves at minimizing level*. Similarly *Beta cutoffs are applied by minimizing player* (the opponent) cuts of *moves at maximizing* ply.

It is also important to note that alpha and beta cutoffs are not applied together but at alternative plies. Our recursive algorithm needs to pass both values at every level but at one level only one of them will be used. For example at maximizing level only beta is used, but when we call a minimizing level (next level) from it we need the alpha value as a threshold. Similarly, at minimizing level we only need alpha, but we also need to call next level as maximizing level from it and we need to pass beta to it. Thus both values are to be passed to each level.

We have designed MiniMax algorithm in a way that we do not need to differentiate between maximizing and minimizing level. We have just negated best values at alternate levels to create that effect. We would like to do the same while updating the algorithm for alpha beta pruning. To do that, we need to pass two threshold values, one which is current and another is next. The CurrentThreshold value is to be applied to current level to allow or cutoff moves while NextThreshold is to be applied when the next level is to be explored. Thus we will pass NextThreshold value to next level as CurrentThreshold. The current threshold value for current level becomes next threshold value for the next level as that threshold is to be applied to next to next level. Thus we will not be specifying which one is alpha and which one is beta. If the current level is maximizing level, the CurentThreshold is beta and NextThreshold is alpha and vice versa for minimizing level. Exactly like the heuristic values, we will negate these values too to remain in sync. As we are negating at each level exactly with the heuristic values, our testing of best value being greater or less than these threshold values makes sense.

You probably have noticed that the use of alpha or beta threshold value requires it to set in the previous level. For example let us take the case depicted in figure 22.1 which is same as 20.4.

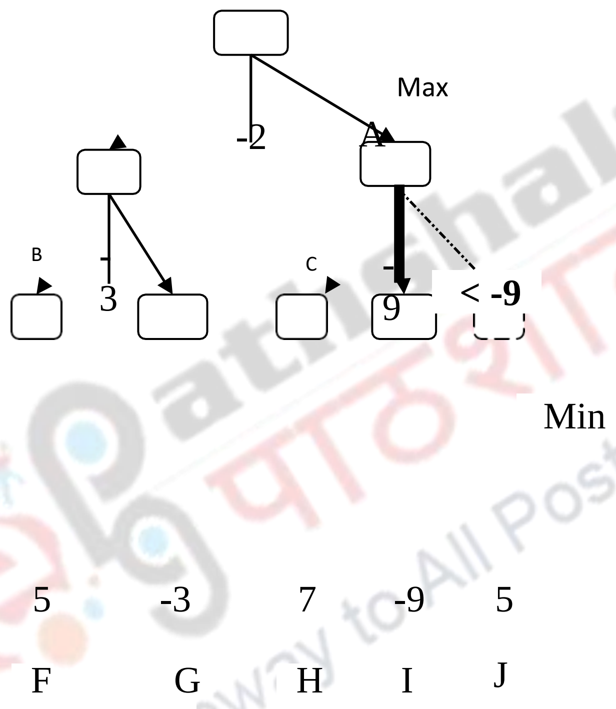


Figure 22.1 alpha cutoff

While we are exploring the second level and when we encounter -9, we know that we already have the other option of playing B to get -3 so the maximizing user will not choose this move (C). This value -3 was set during exploration of level 1. Thus this value -3 is the best so far node at level 3. The idea of alpha cutoff is to make sure that when a minimizing node guarantees to provide lesser value than -3, there is no point in exploring other siblings of that node. The value that we use for cutoff (-3) is decided at B, level 1, which is used as the threshold value at level 2.

Thus the threshold value applied at level n is the best value one gets at level n-1 for alpha cutoff. Similarly look at figure 22.2 which is essentially same as 22.1 to see if it is also true for the beta cutoff.

Our beta cutoff, -9 has come from a sibling H. when we get a better value than -9, we know this being minimizing ply, and the opponent is not going to play any move better than -9 as he already have the option of playing -9. Thus the decision made at level 3, is decided on the basis of threshold value calculated at the previous level. Thus the value for beta cutoff is applied at the minimzing ply but the threshold decided at the previous minimizing ply. While we are exploring children of I, the value found at previous level (-9) must be passed to it. Thus the same thing is true for minimizing ply as well!

Thus we can state a simple rule of thumb for this process, the best successor of the previous level is to be considered as the threshold value for current level.

That means that the NextThreshold is to be updated with the best value one gets during current ply while the CurrentThreshold will be used to check the further exploration at the same level. You can see that in figure 22.3, in case of level 1, the CurrentThreshold value is 3 which is passed to next level while we are exploring.

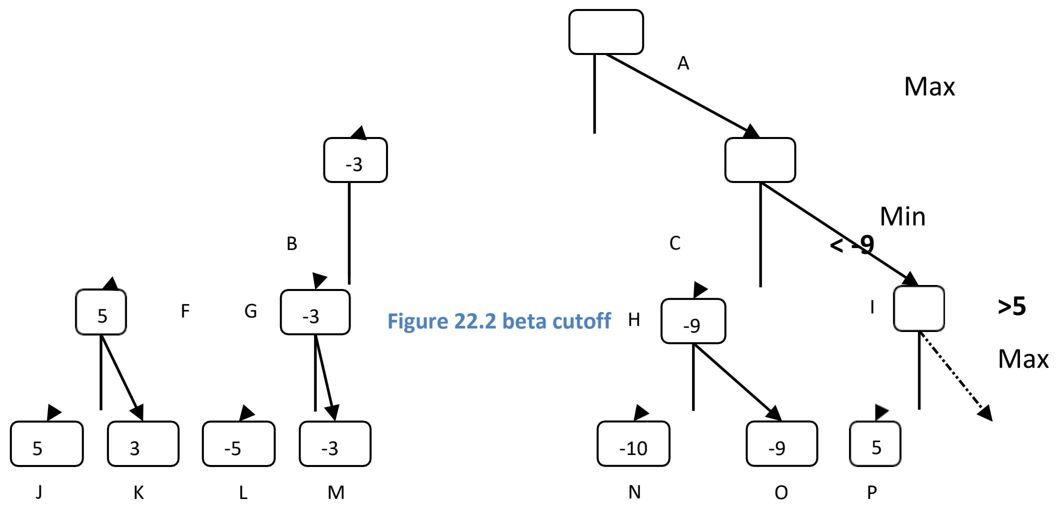
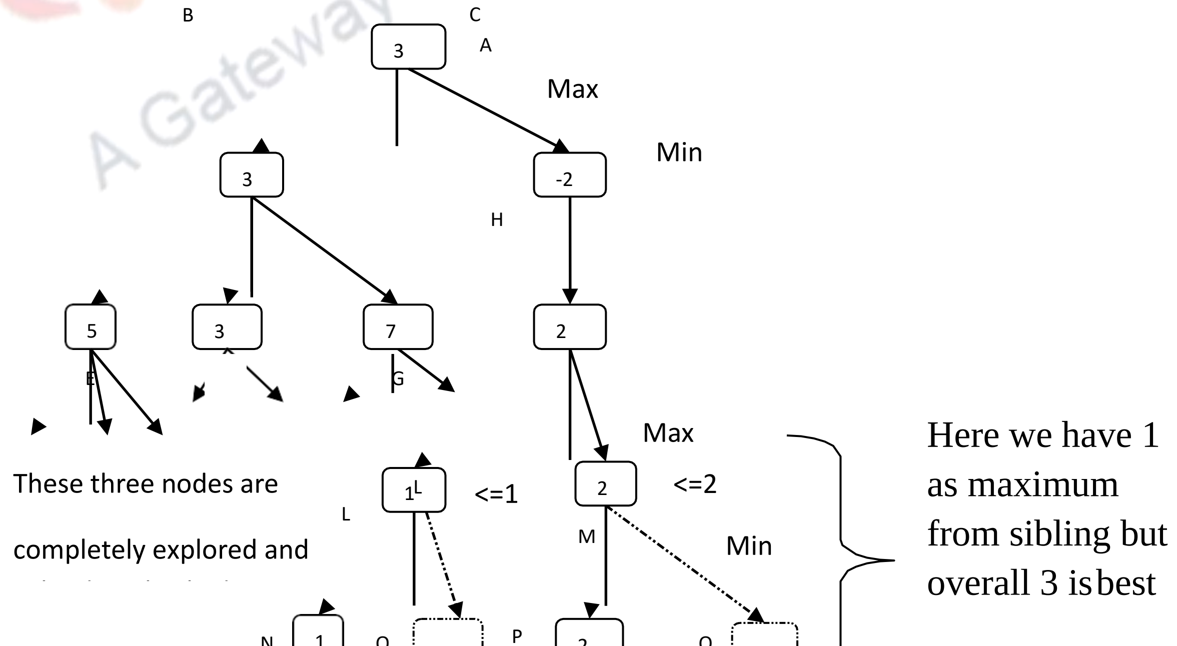


Figure 22.2 beta cutoff

Another point is also important to note. There are two different values to be considered. First the maximum value at parent level so far and maximum value overall.



possible so the
beta value is 3
and not 1

Figure 22.3 The cutoff is maximum possible value

Look at figure 22.3. This is a game tree with 5 levels till the leaves. We can call it 5 ply tree. The leaves are evaluated and their values are backed up one after another. The branch where the root node is B, is already explored with best value being 3. We are taking a case where C is being explored. We begin with the exploration of N, passing the value up to L, checking with threshold and omitting exploration of O, pass the value up to H and start exploring M and its child P, we apply SEF to P and yield 2, which we pass up. This is where we have taken the snap shot of the tree.

When L explored, we get 1 from N and thus it is less than 3 which is guaranteed from B so we do not explore O. now when we are exploring M and we get 2 from P, we have a problem. If we compare this node with the sibling L, we have a better value so we should explore as we are guaranteed to get 2 from this. Though the sibling provides an upper bound of 1, an inherited upper bound from A will be 3. That means that A already has a possibility to play B with a guarantee of 3. If we explore the sibling of P, called Q, we are guaranteed of maximum 2. If O has higher value, the opponent will choose P. Though this move is better than the sibling L, it is still worse than the best so far, B, and thus we will not explore other children of M. In figure 21.2 we can see that we can continue working on the same problem using 3 as the best score so far and eventually coming to the conclusion that we must pick up B as our next step. That means even when sibling is better, if it is not better than global best so far, we only will look at global best so far value. That means, we will also be passing this best so far down the tree while exploring with other options.

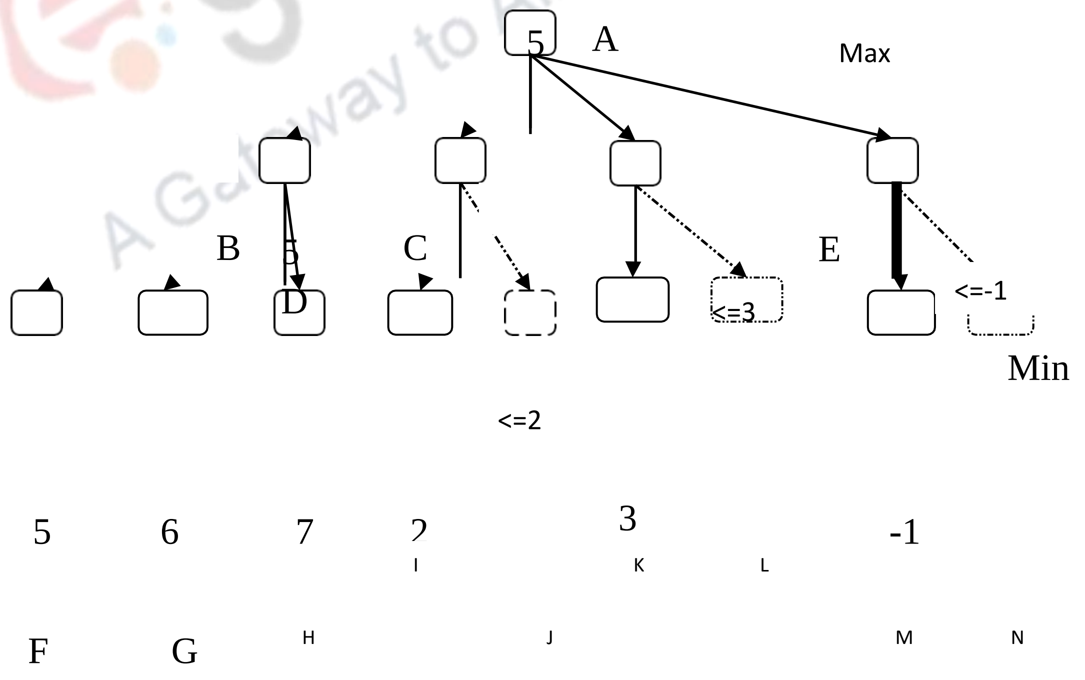


Figure 21.4 The further exploration of the game tree depicted in 21.1

One more important observation is about ordering the exploration. As we have explored the best node B, before C, D and E, it becomes easier for us to have alpha cutoffs at all of them. We are not only removing one child as it might seem from the figure, but more than one child if there are, and all children with complete sub tree beneath. The amount of save is almost 50%. What if we have explored E, D, C before B? We would not have that advantage. If we have explored E first, we will have to examine all other children of E. Assuming all other children are better than -1, we will have -1 from E. That will not help us alpha cutting K and other children of D. We might have some cutoff at C but again B will have to be completely explored to find that it is better than the rest. Thus the order of exploration is an important issue. If we can somehow know which nodes are more promising, we can certainly take better advantage of alpha beta cutoff.

Algorithm

Here is the algorithm.

```
Minimax (CurrentState, CurrentDepth, CurentPlayer, CurrentThreshold,
NextThreshold)

1. If Over (CurrentState, CurrentPlayer)
   {
           NodeInfo
           NF;
           NF.Path=
           NULL
           NF.Value =
           SEF(CurrentState,CurrentPlayer)
           return NF,
   }
2. Else

           ChildrenStatesList = PMG(CurrentState, CurrentPlayer)
3. If ChildrenStateList == []
   {
           NodeInfo
           NF; NF.Path
           = NULL;
           NF.Value =
           SEF(CurrentState,CurrentPlayer);
           return NF;
   }
4. Else
   for each Child in ChildrenStateList do
   {NodeInfoChildValues
= MiniMax(Child, CurrentDepth + 1, 1 – CurrentPlayer, -NextThreshold
           ,-CurrentTHreshold);ChildSEF = (-1) * ChildValues.Value;
   If ChildSEF>NextThreshold
           {
                   NextThreshold= ChildSEF;
                   Path = Child + ChildValues.Path;// add Child in the front of the
                   path
           }
```

```
        If
        CurrentThreshold>ChildSEF//
        Cutoff[
            NodeInfo
            NF;
            NF.Path
            =Path;
            NF.Value =
            NextThreshold;
            Return NF;
        }
    }
```

5. NodeInfo NF

```
NF.Path   =   Path;
NF.Value
=NextThreshold;
return NF;
```

The process

The process is exactly like conventional MiniMax described in the previous module additionally considering alpha beta cutoff. Let us try to understand.

We begin with our old Over function as well as structure NF. There is no change here so we will not describe it further. Step number 1, 2, 3 also are the same. Next part is different so important here.

For each successor of the current node, we first invoke MiniMax algorithm. We will increase the depth by one, change the player, and change both the threshold values position as well as sign. Why? Because the current threshold will become next threshold now and next threshold becomes current now. We have seen why we do that before. Next is assigning the value returning from the Minimax call to value ChildSEF. If this ChildSEF is better than current threshold value, it is a better node than what we have seen so far (this part is also similar to pervious algorithm, if we get a better node than the BEST, we will make it BEST). Also, the best path should travel through this node so we adjust the path accordingly.

The only logic left now is for cutoff. If our current threshold becomes greater than next threshold (the cutoff value), there is no point in exploring this node's children, so we do not explore other children and return immediately. If that does not happen with any one of the children and all children are exhausted, we will return with the best values like before.

Closely observe following statement

```
        If ChildSEF>NextThreshold
```

The value of the current node is more than what the threshold is. NextThreshold is best so far and thus we need to have a better best so far now;so the next statement.

```
NextThreshold = ChildSEF;
Path = Child + ChildValues.Path;// add Child in the front of the path
```

Remember our discussion. The threshold which is to be used in the next ply is set now. So we are setting the next threshold's value. We are setting next threshold's value right here.

Also observe another statement

```
If CurrentThreshold>ChildSEF // Cutoff
```

What are we doing now? The current threshold value is set up at the parent level, we are testing it if it is more than the node's child's SEF. If so, there is no point in exploring that node further. So we return immediately.

One last point; how do you think the algorithm will be

called initially? MiniMax(CurrentPosition, 0,

CurrentPlayer, 10,-10)

Why? It is because we need to pass maximum threshold value for CurrentThreshold and minimum for next threshold. Thus, we can have any node having any value more than -10 can be used to set threshold for next level. (It is very similar to setting minimum value when we want to get maximum of some numbers). The current threshold will be set in the next level so we have kept it as a maximum value now which will become a minimum value in the next level and any node with little better value can be accepted.

Futility cutoff

Some research is done on how much saving is possible using alpha beta cutoffs on a general purpose game tree. We have already seen in figure 21.4 that ordering determines the amount of saving. Knuth and Moore proved an interesting theorem. It says that if we have n nodes for a depth of length d without using alpha beta cutoff and we examine all leaves and back the values up. Now if we apply alpha beta cutoff and we also somehow manage to have the tree perfectly ordered. The result is that we can examine the leaves of the same tree for double the depth (2d) and back the values up.

This is indeed a huge saving. For example if a binary tree case, each level adds almost the same number of nodes that we have already explored. Thus even when we save a time for exploring just one level in a binary tree, it is double than without alpha beta cutoff. Having d more levels is 2^d; and for an m-ary tree it is m^d. This is really big saving.

An interesting proposal is not only to ignore nodes which are poorer, but also some of them who are just little better. For example, if we get a threshold value as 5 and if we get a node with a value 5.1, there is no point in exploring it as well as anything less than 5. This is known as futility cutoff.

Summary

We begun this module with understanding that alpha cutoffs are applied at minimizing ply but the threshold for which is decided at previous maximizing ply. Similarly beta cutoffs are applied at maximizing ply but the threshold for which is decided at previous minimizing ply. We followed this and decided two values; one value which is to be passed to next level for testing is set in this ply. One which is set in the previous ply is used to test if the heuristic value is going above threshold here. We use CurrentThreshold is one which we test our values against and NextThreshold which we set during exploration of this ply. The recursive procedure is quite similar to previous module except for passing CurrentThreshold and NextThreshold values. Both values are negated at every ply for the same reason we negate the heuristic values. Additionally, the current threshold is to be set in the next level and must act as NextThreshold in the next ply. Similary the NextThreshold value that we are setting here is to be used for testing in the next ply so should act as CurrentThreshold value. This switching over process happens continuously and thus the algorithm changes the position of both these values across calls. The saving achieved using alpha beta cutoff depends on right ordering. If the ordering is perfect, the saving is exponential. One can not only avoid worse paths but also paths which are just little better than current and thus can improve search time.