SEARCH TECHNIQUE

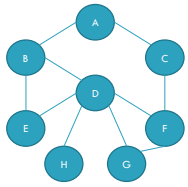
Unit 3

Outline

- Searching
- Uninformed Search Techniques
  - Breadth First Search
  - Uniform Cost Search
  - Depth First Search
  - Backtracking Search
  - Depth Limited Search
  - Iterative Deepening Depth First Search
  - Bidirectional Search
  - Search Strategy Comparison
- **Informed Search Techniques**
  - **Best First Searching**
    - **Greedy Search**
    - **A* Search**
  - **Local Search Algorithm & Optimization**
  - **Iterative Improvement Algorithm**
    - **Hill Climbing Search**
    - **Simulated Annealing Search**
    - **Local Beam Search**
    - **Genetic Algorithm**
  - **Adversarial Search Techniques**
    - **Mini-max Procedure**
    - **Alpha Beta Procedure**

Searching

- Step in Problem Solving
- Searching is Performed through the State Space
- Searching accomplished by constructing a search tree



Searching: Steps

- Check whether the current state is the goal state or not
- Expand the current state to generate the new sets of states
- Choose one of the new states generated for search which entire depend on the selected search strategy
- Repeat the above steps until the goal state is reached or there are no more states to be expanded

Searching: Criteria to Measure Performance

- **Completeness:** Ability to find the solution if the solution exists
- **Optimality:** Ability to find out the highest quality solution among the several solutions
  - Should maintain the information about the number of steps or the path cost from the current state to the goal state
- **Time Complexity:** Time taken to find out the solution
- **Space Complexity:** Amount of Memory required to perform the searching

Searching: Types

- Blind Search or Uninformed Search
- Informed Search or Heuristic Search

Searching: Evolution Function

- A number to indicate how far we are from the goal
- Every move should reduce this number or if not never increase
- When this number becomes zero, the problem is solved (there may be some exceptions)

8 Puzzle Games

1	2	3
8		4
7	6	5

- Its Goal State
- Evolution Function = 0

2	8	3
1	4	
7	6	5

- Its Initial State
- Evolution Function = -4

Searching: Problem Classification

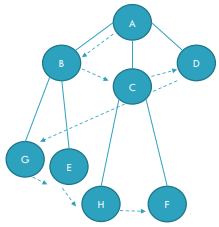
- **Ignorable**: Intermediate actions can be ignored. Example: Water Jug Problem
- **Recoverable**: The actions can be implemented to go the initial states. Example: 8 Puzzle Games
- **Irrecoverable**: The actions can't be implemented to reach the previous state. Example: Tic-Tac-Toe
- **Decomposable**: The problem can be broken into similar ones. Example: Bike Racing

Uninformed Search

- Search provided with problem definition only and no additional information about the state space
- Expansion of current state to new set of states is possible
- It can only distinguish between goal state and non-goal state
- Less effective compared to Informed search

Breadth First Search

- Root node is expanded first
- Then all the successors of the root node are expanded
- Then their successors are expanded and so on.
- Nodes, which are visited first will be expanded first (FIFO)
- All the nodes of depth 'd' are expanded before expanding any node of depth 'd+1'



Breadth First Search: Four Criteria

- Completeness
  - d: depth of the shallowest goal
  - b: branch factor
  - This search strategy finds the shallowest goal first
  - Complete, if the shallowest goal is at some finite depth
- Optimality
  - If the shallowest goal nodes were available, it would already have been reached
  - Optimal, if the path cost is a non-decreasing function of the path of the node

Breadth First Search: Four Criteria

13

- Time Complexity
  - For a search tree a branching factor 'b' expanding the root yields 'b' nodes at the first level.
  - Expanding 'b' nodes at first level yields b^2 nodes at the second level.
  - Similarly, expanding the nodes at $(d+1)^{th}$ level yields b^d node at d^{th} level
  - If the goal is in d^{th} level, in the worst case, the goal node would be the last node in the d^{th} level
  - Hence, We should expand (b^d-1) nodes in the d^{th} level (Except the goal node itself which doesn't need to be expanded)
  - So, Total number of nodes generated at d^{th} level = $b(b^d-1)$ $=b^{d+1}-b$
  - Again, Total number of nodes generated = $1+b+b^2+\ldots+b^{d+1}-b$ $=O(b^{d+1})=O(b^d)$
  - Hence, time complexity is $O(b^{d+1})$ where, b= branching factor and d= level of goal node in the search table

Breadth First Search: Four Criteria

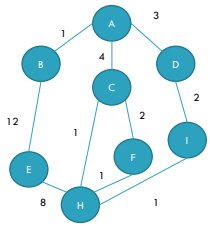
14

- Space Complexity
  - Same as time complexity
  - i.e. $O(b^{d+1})$
  - Since each node has to be kept in the memory
- Disadvantages
  - Memory Wastage
  - Irrelevant Operations
  - Time Intensive
  - It doesn't assure the optimal cost solution

Uniform Cost Search

15

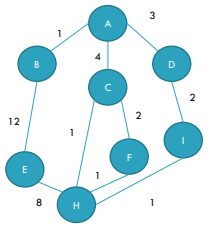
- It expands the lowest cost mode on the fringe
- The first solution is guaranteed to be the cheapest one because a cheaper one would have expanded earlier and so would have been found first
- Required Condition: A to H
  - ABEH=21, ACH=5, ACFH=7, ADIH=6



Uniform Cost Search

16

- Solution: Required Operation
  - Expand A→ Yield B, C, D With AB=1, AC=4, AD=3
  - Expand B→ Yield E with ABE=13 As ABE>AC and ABE>AD
  - Expand D→ Yield I with ADI=5 As ADI>AC
  - Expand C→ Yield H and F with ACH=5 and ACF=6
  - Solution Achieved
- If all step costs are equal, it is identical breadth first search



Uniform Cost Search

17

- Disadvantages
  - Doesn't care about the number of steps a path has but only about their cost
  - It might get stuck in an infinite loop if it expands a node that has a zero cost action leading back to same state

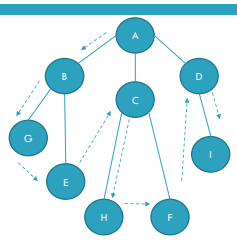
Uniform Cost Search: Four Criteria

18

- Completeness
  - Complete, if the cost of every step is greater than or equal to some small positive constant E
- Optimality
  - The same ensures optimality
- Time Complexity
  - $O(b^{C*/E})$
  - Where C*→ cost of optimal path and E→ small positive constant
  - This complexity is much greater than that of Breadth first search
- Space Complexity
  - $O(b^{C*/E})$

Depth First Search

- Expands the nodes at the deepest level of the search tree (LIFO)
- When a dead end is reached, the search backup to the next node that still has unexplored successors



Depth First Search: Four Criteria

- Completeness
  - Can get stuck going down the wrong path
  - It will always continue downwards without backing up
  - If the path chose get infinitely down, even when shallow solution exists
  - Not complete
- Optimality
  - The strategy might return a solution path that is longer than the optimal solution, if it starts with an unlucky path
  - Not optimal

Depth First Search: Four Criteria

- Space Complexity
  - It needs to store a single path from root to a leaf node and the remaining unexpanded sibling nodes for each node in the path
  - For a search tree of branching factor 'b' and maximum tree depth 'm', only the storage of b_{m+1} node is required
  - Hence,
    Space Complexity= O(b.m+1)
    = O(bm)
- Time Complexity
  - $O(b^m)$, in the worst case, since in the worst case all the b^m nodes of the search tree would be generated
  - Hence,
    Time Complexity= $O(b^m)$

Backtracking Search

- It uses still less memory
- Only one successor is generated at a time rather than all
- Each partially expanded nodes remember which node to expand next
- Completeness: Not Complete
- Optimality: Not Optimal
- Time Complexity= $O(b^m)$
- Space Complexity= O(m)

Depth Limited Search

- Modification of depth first search
- Depth first search with predetermined limit 'l'
- After the nodes at the level 'l' are explored, the search backtracks without going further deep
- Hence, it solves the infinite path problem of the depth first search strategy
- Completeness: Complete except at additional source of incompleteness if l>d
- Optimality: Optimal except at l>d
- Time Complexity=$O(b^l)$
- Space Complexity=O(bl)

Exercises

- A farmer has to cross a river with his Fox, Goose, and Grain. Each trip, his boat can only carry himself and one of his possessions. How can he cross the rivers if an unguarded fox eats the goose and an unguarded goose the grain.
  - Perform Depth-first-search(DFS)
  - Perform Breath-First-Search(BFS)
  - Construct a complete Search Tree.

Exercise

- Three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks and the boat, if there are missionaries present on the bank (or the boat), they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.
  - Perform DFS
  - Perform BFS
  - Construct a complete search tree

Iterative Deepening Depth First Search

- Finds the best limit by gradually increasing depth limit l first to 0, then to 1, 2 and so on
- Combines the benefits of the depth first and breadth first search
- The complex part is to choose good depth limit
- This strategy addresses the issue of good depth limit by trying all possible depth limits
- The process is repeated until goal is found at depth limit 'd' which is the depth of shallowest goal
- Completeness: as of Breadth First Search i.e. Complete if branching factor is finite
- Optimality: as of Breadth First Search i.e. optimal if the path cost is non decreasing function of depth
- Time Complexity= $O(b^d)$
- Space Complexity= $O(b^d)$

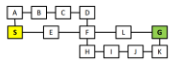
Bidirectional Search

- Performs two simultaneous searches, one forward from initial state and the other backward from the last state
- Search stops when the two traversals meet in the middle
- Completeness: Complete if both searches are B.F.S. and b is finite
- Optimality: Optimal if both searches are B.F.S.
- Time Complexity
  - For B.F.S. is $O(b^{d+1})$
  - If B.F. Bidirectional Search is used then the complexity = $O(b^{d/2})$
  - Since the forward and backward searches have to go halfway only
- Space Complexity= $O(b^{d/2})$

Bidirectional Search

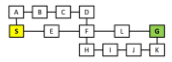
BFS

Forward:
– (<S>)→(<SA>,<SE>)→(<SE>,<SAB>)→**(<SAB>,<SEF>)**
Backward:
– (<G>)→(<GK>,<GL>)→(<GL>,<GKJ>)→**(<GKJ>,<GLF>)**



DFS

Forward:
– (<S>) →(<SA>,<SE>) →(<SAB>,<SE>) →(<SABC>,<SE>) →(<SABCD>,<SE>) →**(<SABCDF>,<SE>)**
Backward:
– (<G>) →(<GK>,<GL>) →(<GKJ>,<GL>) →(<GKJI>,<GL>) →(<GKJIH>,<GL>) →**(<GKJIHF>,<GL>)**



Informed Search

- Strategy of problem solving where problem specific knowledge is known along with problem definition
- These search find solutions more efficiently by the use of heuristics
- Heuristic is a search technique that improves the efficiency of the search process
- By eliminating the unpromising states and their descendants from consideration, heuristic algorithms can find acceptable solutions

Informed Search

- Heuristics are fallible i.e. they are likely to make mistakes as well
- It is the approach following an informed guess of next step to be taken
- It is often based on experience or intuition
- Heuristic have limited information and hence can lead to suboptimal solution or even fail to find any solution at all

Best First Search

- A node is selected for expansion based on evaluation function f(n)
- A node with lowest evaluation function is expanded first
- The measure i.e. evaluation function must incorporate some estimate of the cost of the path from a state to the closest goal state
- The algorithm may have different evaluation function, one of such important function is the heuristic function h(n)
  where, h(n) is the estimated cost of the cheapest path from node n to the goal

Best First Search: Types

- Greedy Best First Search
- A* Search

Greedy Best First Search

- The node whose state is judged to be the closest to the goal state is expanded first
- At each step it tries to be as close to the goal as it can
- It evaluates the nodes by using heuristic function
  hence, f(n)=h(n)
  where, h(n)=0, for the goal node
- This search resembles depth first search in the way that it prefers to follow a single path all the way to the goal or if not found till the dead end and returns back up

Greedy Best First Search

- Completeness
  - Can start down an infinite path and never return to any possibilities
  - Not complete
- Optimality
  - Looks for immediate best choice and doesn't make careful analysis of long term options
  - May give longer solution even if shorter solution exists
  - Not optimal
- Space Complexity
  - $O(b^m)$ where, m is the maximum depth of search space, since all nodes have to be kept in memory
- Time Complexity
  - $O(b^m)$

A* Search

- Evaluates node by combining g(n), the cost to reach the node and h(n) the cost to get from node to goal
  f(n)=g(n)+h(n)
  where f(n) is the estimated cost of the cheapest solution through node n
- To find the cheapest solution, the first try node is the mode with lowest g(n)+h(n)

A* Search

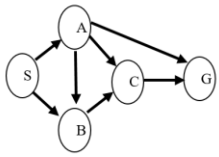
- Admissible Heuristic: h(n) is admissible if it never overestimates cost to reach the solution
  example: h_{SLD} (straight line distance) as g(n) is exact, so f(n) is never overestimated

A* Search

- Optimality
  - Optimal if h(n) is admissible
- Completeness
  - Complete if h(n) is admissible
- Space Complexity
  - $O(b^d)$ if h(n) is admissible
- Time Complexity
  - $O(b^d)$ if h(n) is admissible

A* Search

- Example…



	S	A	B	C	G
S		1	4		
A			2	5	12
B				2	
C					3
G					

State	H(n)
S	7
A	6
B	2
C	1
G	0

Local Search Algorithm and Optimization

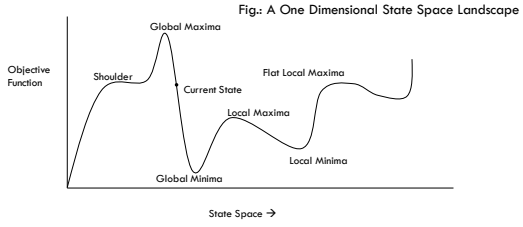
- Optimization
  - Aim to find the best state according to an objective function
- Local Search Algorithm
  - It operates using a single current state rather than multiple path and generally move only to neighbour of that state

Iterative Improvement Algorithm

- Consider that all the states are laid down on the surface of a landscape
- The height of a point on a landscape corresponds to process to move around the landscape trying to find the highest peaks, which are the optimal solutions
- Algorithm is suitable for problems where the path of the goal is irrelevant and only final configuration matters

Iterative Improvement Algorithm

Fig.: A One Dimensional State Space Landscape



Iterative Improvement Algorithm: Types

- Hill Climbing Search
- Simulated Annealing Search
- Local Beam Search
- Genetic Algorithm

Hill Climbing Search

43

- Moves continuously in the direction of increasing value (uphill)
- Doesn't maintain a search tree so the current node data structure needs only record the state and its objective function value
- Hill climbing doesn't look ahead beyond the immediate neighbours of the current state
- Also called greedy local search sometimes because it grabs a good neighbour state without thinking about where to go next

Hill Climbing Search

44

- it often makes very rapid progress towards the solution because it is usually quite easy to improve a bad state
- One move is selected and all other nodes are rejected and are never considered
- Halts if there is no successor

Hill Climbing Search: Problems

45

- Local Maxima
  - Peak that is higher than each of its neighbouring states but lower than the global maxima
  - Hill climbing halts whenever a local maxima is reached
- Plateau
  - An area of the state space landscape where the evaluation function is flat
  - Can be flat local maxima where no uphill exists or shoulder from which it is possible to progress
  - A hill climbing search might be unable to find its way off the plateau
- Ridges
  - A special kind of local maxima which is the result of a sequence of local maxima that is very difficult for greedy algorithms to navigate
  - It is an area of search space that is higher than the surrounding areas and that itself is at a slope

Hill Climbing Search: Solution to the Problems

46

- Local Maxima
  - Backtrack to some earlier node and try going to different direction
- Plateau
  - Make a big jump in some direction to try to get a new section of the search space
  - If rule apply single small steps, apply them several times in the same direction
- Ridges
  - Apply two or more rules such as bidirectional search before doing the test
  - Moving in several directions at once

Simulated Annealing Search

47

- Rather than starting from a different initial state all over again, when a current state shows no progress in the technique
- This search takes some downhill steps so that it can escape that particular local maxima and continue with other peaks in the state space
- A random pick is made for the move
  - If it improves the situation, it is accepted straight away
  - If it worsen the situation, it is accepted with some probability less than 1 which decreases exponentially with the badness of the move i.e. for bad moves the probability is low and for comparatively less bad one, it's higher

Simulated Annealing Search

48

- The degree of badness of the move is determined by the amount ΔE, by which the evaluation is worsened
- The probability also depends on the value of a objective function parameter 'T'
- For low value of T, probability is high and vice versa
- Hence, bad moves are more likely to be allowed at the beginning only
- This method is more common in VLSI layout problem solving, factory scheduling and travelling salesman problems

Local Beam Search

- A path based algorithm
- Keeps track of k-states rather than just one
- Begins with k randomly generated states, at each step, all the successors of all k states are generated
- If any one is the goal, the algorithm halts
- Can quickly become concentrated in a small region of the state space

Genetic Algorithm

- A variant of stochastic beam search in which successor states are generated by combining two parent states, rather than by modifying a single state
- It begins with a set of k randomly generated states called the population
- Each state or individual is represented as a string over a finite alphabet most commons 0s and 1s
- Fitness function defines the fitness value of each state

Genetic Algorithm

- Cross over point is randomly chosen for crossing of strings, which yields offspring for each generation
- Each individual of next generation offspring is subjected to random mutation with a small independent probability
- The primary advantage of GA comes from crossover operation
- GA combines an uphill tendency with random exploration and exchange of information among parallel search threads
- GA has widespread impact on optimization problem such as circuit layout and job scheduling

Genetic Algorithm

Steps in Genetic Algorithm

1. Represent the problem variable as a chromosome of a fixed length, choose the size of a chromosome population *N*, the crossover probability *pc* and the mutation probability *pm*.
2. Define a fitness function to measure the fitness of an individual chromosome in the problem domain.
3. Randomly generate an initial population of chromosomes of size *N*: *x*1, *x*2,., *xN*
4. Calculate the fitness of each individual chromosome: *f*(*x*1), *f*(*x*2), . . . , *f*(*xN*)
5. Select a pair of chromosomes for mating from the current population based on their fitness.
6. Create a pair of offspring chromosomes by applying the genetic operators – **crossover** and **mutation.**
7. Place the created offspring chromosomes in the new population.
8. Repeat *Step 5* until the size of the new chromosome population becomes equal to the size of the initial population, *N*.
9. Replace the initial (parent) chromosome population with the new (offspring) population.
10. Go to *Step 4*, and repeat the process until the termination criterion is satisfied

GA : Case Study

Example of Selection

Evolutionary Algorithms is to maximize the function $f(x) = x^2$ with x in the integer interval $[0, 31]$, i.e., $x = 0, 1, \ldots 30, 31$.

1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to 31.
2. Assume that the population size is 4.
3. Generate initial population at random. They are chromosomes or genotypes; e.g., 01101, 11000, 01000, 10011.
4. Calculate fitness value for each individual.
   (a) Decode the individual into an integer (called phenotypes),
   $01101 \rightarrow 13$; $11000 \rightarrow 24$; $01000 \rightarrow 8$; $10011 \rightarrow 19$;
   (b) Evaluate the fitness according to $f(x) = x^2$,
   $13 \rightarrow 169$; $24 \rightarrow 576$; $8 \rightarrow 64$; $19 \rightarrow 361$.
5. Select parents (two individuals) for crossover based on their fitness in p_i. Out of many methods for selecting the best chromosomes, if **roulette-wheel** selection is used, then the probability of the i^{th} string in the population is $p_i = F_i / (\sum_{j=1}^{n} F_j)$, where
   F_i is fitness for the string i in the population, expressed as $f(x)$
   p_i is probability of the string i being selected,
   n is no of individuals in the population, is population size, $n=4$
   $n * p_i$ is expected count

53

GA : Case Study

String No	Initial Population	X value	Fitness Fi f(x) = x²	pi	Expected count N * Prob i
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Sum			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

The string no 2 has maximum chance of selection.

6. Produce a new generation of solutions by picking from the existing pool of solutions with a preference for solutions which are better suited than others:

We divide the range into four bins, sized according to the relative fitness of the solutions which they represent.

Strings	Prob i	Associated Bin
0 1 1 0 1	0.14	0.0 ... 0.14
1 1 0 0 0	0.49	0.14 ... 0.63
0 1 0 0 0	0.06	0.63 ... 0.69
1 0 0 1 1	0.31	0.69 ... 1.00

By generating 4 uniform (0, 1) random values and seeing which bin they fall into we pick the four strings that will form the basis for the next generation.

Random No	Falls into bin	Chosen string
0.08	0.0 ... 0.14	0 1 1 0 1
0.24	0.14 ... 0.63	1 1 0 0 0
0.52	0.14 ... 0.63	1 1 0 0 0
0.87	0.69 ... 1.00	1 0 0 1 1

54

GA : Case Study

7. Randomly pair the members of the new generation
   Random number generator decides for us to mate the first two strings together and the second two strings together.

8. Within each pair swap parts of the members solutions to create offspring which are a mixture of the parents :
   For the first pair of strings: 01101 , 11000
   – We randomly select the crossover point to be after the fourth digit. Crossing these two strings at that point yields:
   01101 ⇒ 0110|1 ⇒ 01100
   11000 ⇒ 1100|0 ⇒ 11001
   For the second pair of strings: 11000 , 10011
   – We randomly select the crossover point to be after the second digit. Crossing these two strings at that point yields:
   11000 ⇒ 11|000 ⇒ 11011
   10011 ⇒ 10|011 ⇒ 10000

55

GA : Case Study

9. Randomly mutate a very small fraction of genes in the population :
   With a typical mutation probability of per bit it happens that none of the bits in our population are mutated.

10. Go back and re-evaluate fitness of the population (new generation) :
    This would be the first step in generating a new generation of solutions. However it is also useful in showing the way that a single iteration of the genetic algorithm has improved this sample.

String No	Initial Population (chromosome)	X value (Pheno types)	Fitness $f(x) = x^2$	Prob i (fraction of total)	Expected count
1	01100	12	144	0.082	0.328
2	11001	25	625	0.356	1.424
3	11011	27	729	0.415	1.660
4	10000	16	256	0.145	0.580
Total (sum)			1754	1.000	4.000
Average			439	0.250	1.000
Max			729	0.415	1.660

Observe that :
1. Initial populations : At start step 5 were
   01101, 11000, 01000, 10011
   After one cycle, new populations, at step 10 to act as initial population
   01100, 11001, 11011, 10000
2. The total fitness has gone from 1170 to 1754 in a single generation.
3. The algorithm has already come up with the string 11011 (i.e x = 27) as a possible solution.

56

Genetic Algorithm

57

Genetic algorithms are used to solve many large problems including:
- -Scheduling
- - Transportation
- -Chemistry, Chemical Engineering
- - Layout and circuit design
- -Medicine
- -Data Mining and Data Analysis
- -Economics and Finance
- -Networking and Communication
- - Game etc.

Adversarial Search Techniques

58

- Often known as Games or Game Playing
- Used in competitive multi-agent environments
- Based on game theory
- Deterministic and fully observable environments 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
- This creates adversarial situation

Optimal Decision in Adversarial Search

59

- A game can be defined as a kind of search problem with the following components:
  - Initial State identifying the initial position in the game and identification of the first player
  - Successor Function returning a list of (move, state) pairs
  - Terminal Test which determine that the game is over
  - Utility function which gives a numeric value for the terminal states.

Minimax Algorithm

60

- Max is considered as the first player in the game and Min as the second player
- This algorithm computes the minimax decision from the current state
- It uses a recursive computation of minimax values of each successor state directly implementing some defined function
- The recursion proceeds from the initial node to all the leaf nodes
- Then the minimax values are backed up through the tree as the recursion unwinds
- It performs the depth first exploration of a game tree in a complete way

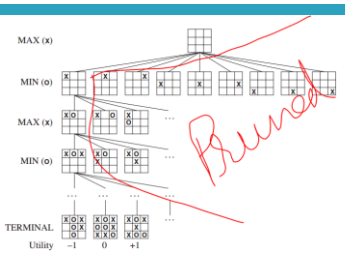
Alpha Beta Pruning

61

- Minimax algorithm has to examine exponentially increasing number of moves
- As the exponential rise can't be avoided Pruning cut it into halves
- By not considering a large part of the tree number of states to be calculated is cut down
- When applied to a standard minimax tree, alpha beta pruning returns the same move as minimax would, but prunes away the branches which couldn't possibly influence the final decision
- Alpha beta pruning could be applied to the trees of any depth

Alpha Beta Pruning

62



References

63

- Russell, S. and Norvig, P., 2011, Artificial Intelligence: A Modern Approach, Pearson, India.
- Rich, E. and Knight, K., 2004, Artificial Intelligence, Tata McGraw hill, India.

64 Thank You

Any Queries?

Now, Search for yourself.