AI Unit - 2 pdf - Lecture notes 2

Artificial Intelligence (SRM Institute of Science and Technology)

Studocu is not sponsored or endorsed by any college or university
Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Unit- II

18CSC305J– Artificial Intelligence

Dr. R. Rajkamal

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Search Techniques

References: **Artificial Intelligence – A Modern Approach 4e, 2020, by Stuart Russell and Peter Norvig**
Artificial Intelligence 3rd edition, 2009, Elaine Rich, Kevin Knight Shivsankar Nair
ARTIFICIAL INTELLIGENCE: Building Intelligent Systems, 2015 PARAG KULKARNI, PRACHI JOSHI

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

General Search Algorithms

Terminologies

State space : the set of all states reachable from the initial state by *any* sequence of actions
- *When several operators can apply to each state, this gets large very quickly*
- Might be a proper subset of the set of configurations

Path : a sequence of actions leading from one state to another state

Frontier : those states that are available for *expanding*(for applying legal actions to)

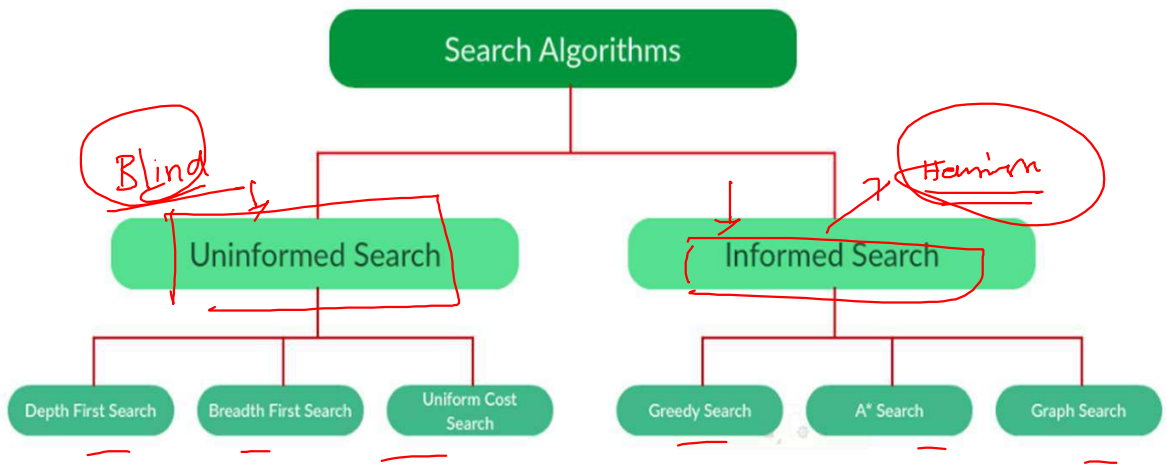
Solution : a path from the initial state to a state that satisfies the goal test

Problem space : what to solve?

Searching strategy: strategy for controlling the search

Search tree : tree representation of search space, showing possible solutions from initial state

General Search Algorithms



Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

General Search Algorithms

Blind search → traversing the search space until the goal nodes is found (might be doing exhaustive search).

•Techniques : **Breadth First, Depth first, Depth Limited, Iterative Deepening search, Uniform Cost search.**

•Guarantees solution.

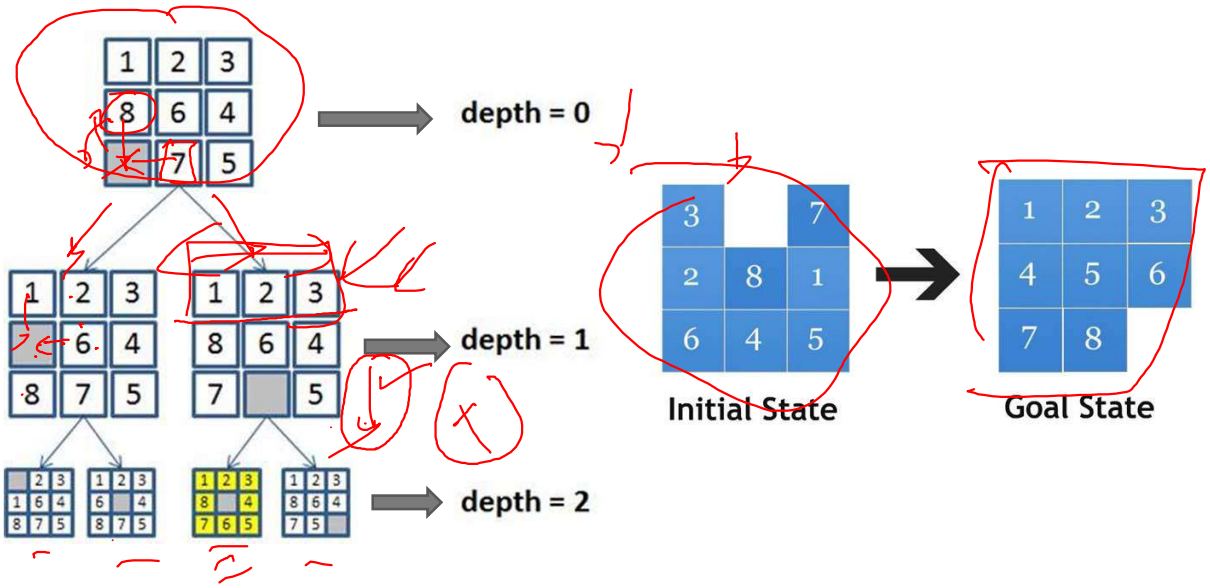
Heuristic search → search process takes place by traversing search space with applied rules (information).

•Techniques: **Greedy Best First Search, A* Algorithm**

•There is no guarantee that solution is found.

A blind search is usually uninformed. It doesn't have any specific knowledge about the problem whereas a **heuristic search is that which has information about the problem and therefore uses logic in decision making.**

8 – Puzzle Example

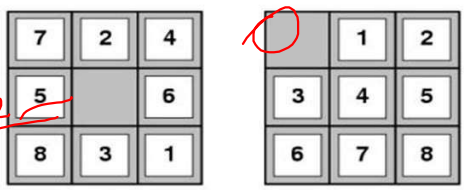


This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

8 – Puzzle Example

Example: 8-puzzle



Start State Goal State

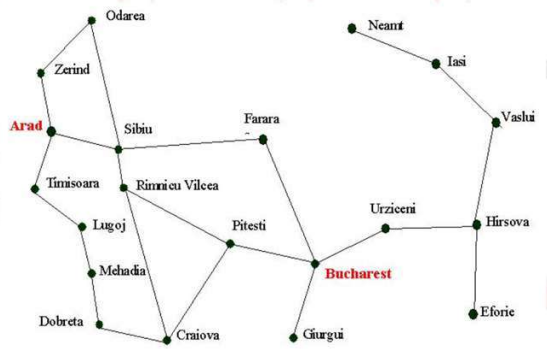
- States:
  - location of each tile and the blank
- Initial state: any,
- Actions:
  - blank moves Left, Right, Up or Down
- Transition model:
  - Given a state and action, returns the resulting state
- Goal test: Goal configuration
- Path cost: Each step costs 1

Breadth-First Search (BFS)

Travel Salesman Problem

Arad to Bucharest

- Breadth-first search goes through **the tree level by level, visiting all of the nodes** on the top level first, then all the nodes on the second level, and so on.
- It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.



Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Breadth-First Search

- Using a breadth-first strategy we expand the root level first and then we expand all those nodes (i.e. those at level 1) before we expand any nodes at level 2.

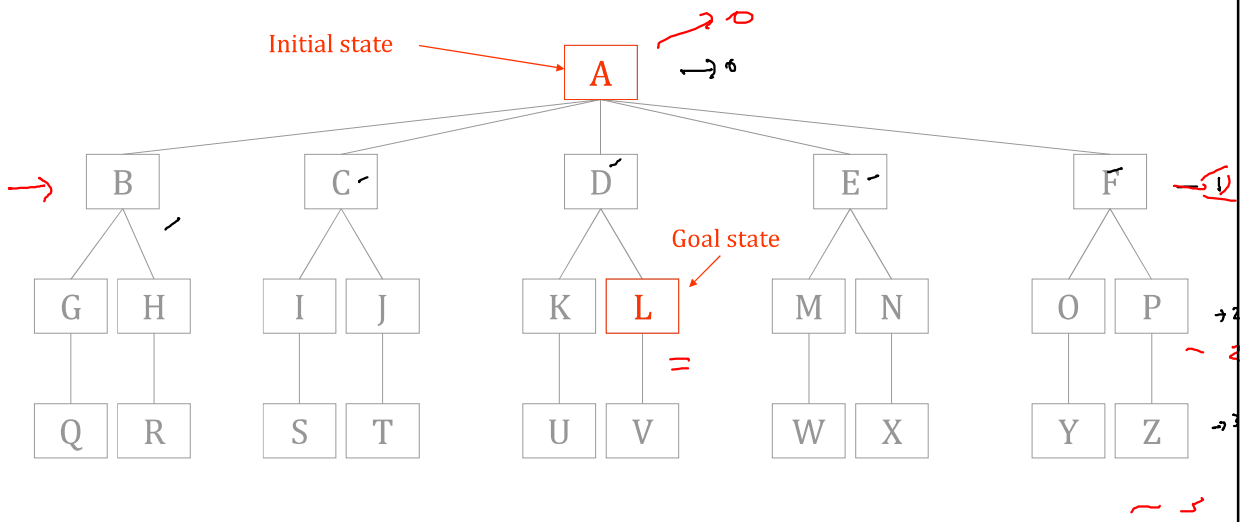
- Or to put it another way, all nodes at level d are expanded before any nodes at level d+1.

Function BREADTH-FIRST-SEARCH(problem) returns a solution or failure

Return GENERAL-SEARCH(problem,ENQUEUE-AT-END)

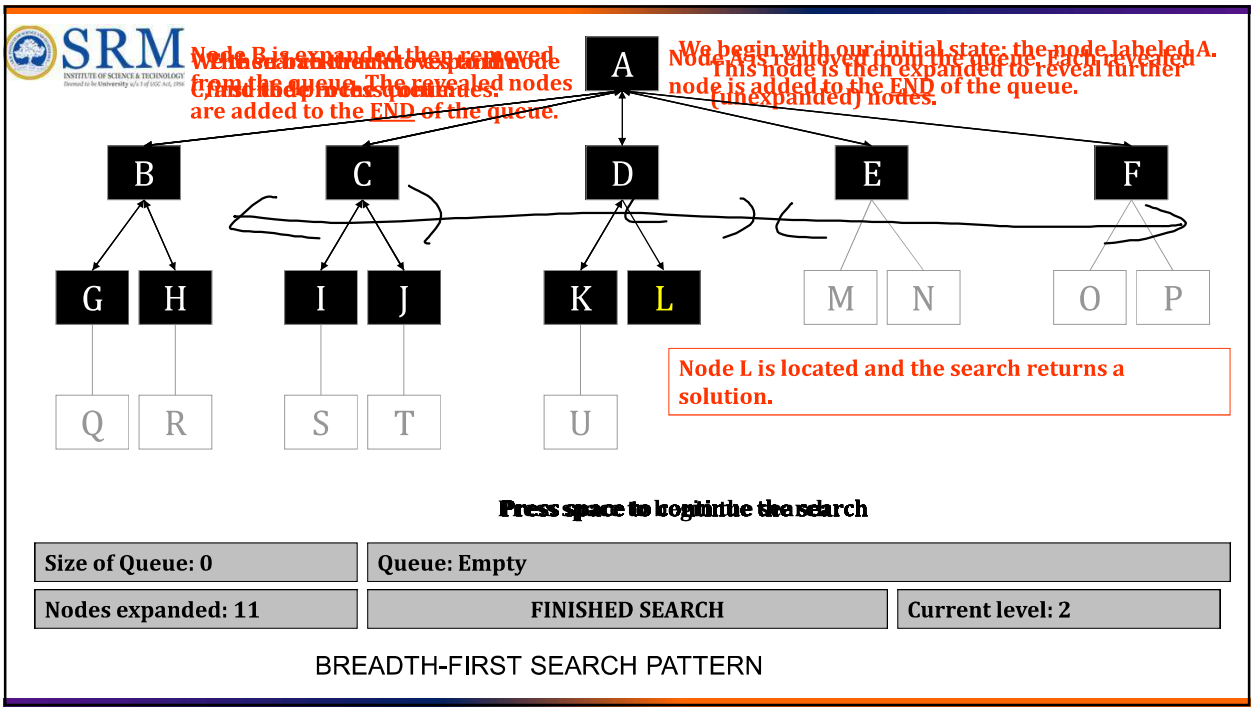
- Consider example having 26 nodes

The example node set



This document is available free of charge on studocu

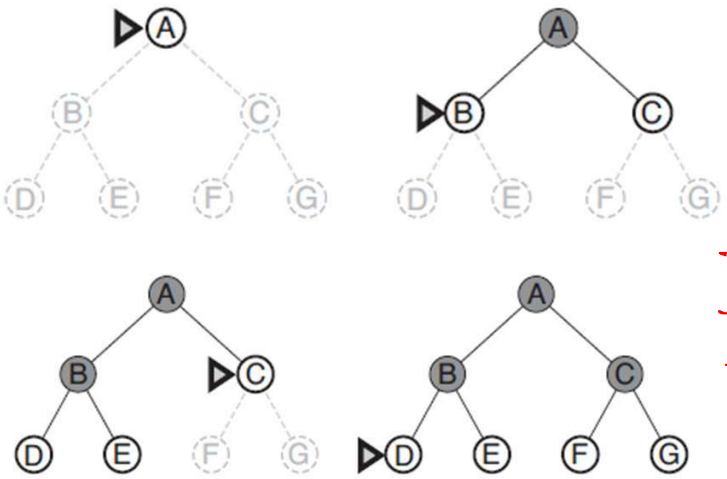
Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

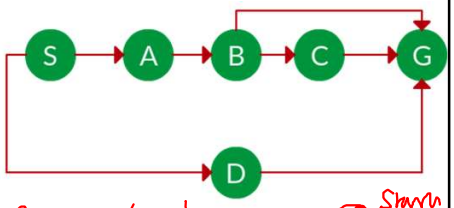


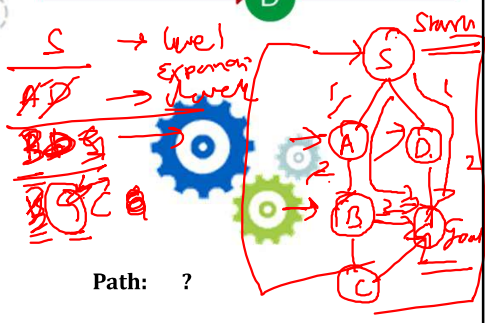
BREADTH-FIRST SEARCH PATTERN

Breadth-First Search (BFS)

Apply BFS move from node S to node G







Path: ?

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Breadth-First Search (BFS)

- This strategy has the benefit of being complete (if there's a solution, it will be found), and optimal as long as the shallowest solution is the best solution.
- However, the way breadth-first search achieves this is by **keeping all of the leaf nodes in memory,** which requires a prohibitive amount of memory when searching anything more than a very small tree.
- The time complexity of breadth-first search is O(b^d) where b is the branching factor (2 for the binary trees below) and d is the depth of the solution.

2^2

Uniform Cost Search

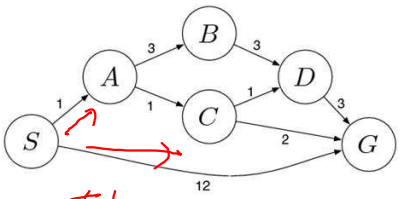
- Uniform Cost Search as it sounds searches in branches which are more or less the same in cost.
- Uniform Cost Search **again demands the use of a priority queue.**
- The priority queue used here is similar with the priority being the **cumulative cost up to the node.**

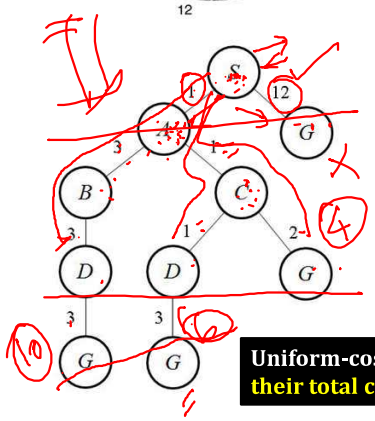
- **Insert the root into the queue**
- **While the queue is not empty**
  Dequeue the maximum priority element from the queue
  (If priorities are same, alphabetically smaller path is chosen)
  - **If the path is ending in the goal state, print the path and exit**
    Else
    Insert all the children of the dequeued element, with the cumulative costs as priority

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Uniform Cost Search





Each element of the **priority queue is written as [path, cumulative cost].**

Initialization: { [S , 0] }

Iteration 1: { [S -> A , 1] , [S -> G , 12] }

Iteration 2: { [S -> A ->C , 2] , [S ->A ->B , 4] , [S ->G , 12] }

Iteration 3:
{ [S ->A ->C ->D , 3] , [S ->A ->B , 4] , [S ->A ->C ->G , 4] , [S ->G , 12] }

Iteration 4:
{ [S ->A ->B , 4] , [S ->A ->C->G , 4] , [S->A->C->D->G , 6] , [S->G , 12] }

Iteration 5:
{ [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->A->B->D , 7] , [S->G , 12] }

Iteration 6: gives the final output as S->A->C->G.

Uniform-cost search does not care about the *number* of steps a path has, but only about their total cost.

Uniform Cost Search

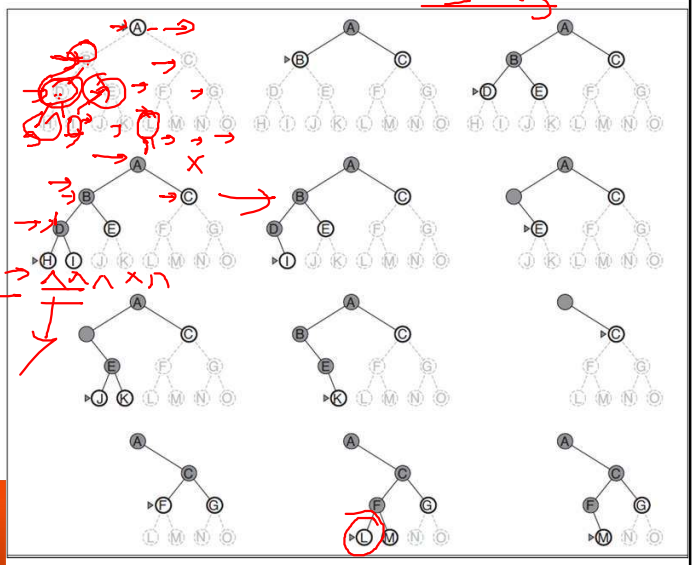
- The algorithm **returns the first path encountered**. It does not search for all paths.
- The algorithm returns a path which is optimal in terms of cost.
- At any given point in the execution, the algorithm never expands a node which has a cost greater than the cost of the shortest path in the graph.
- **The elements in the priority queue have almost the same costs at a given time, and thus the name Uniform Cost Search.**
- It may seem as if the elements don't have almost the same costs. But when applied on a much larger graph it is certainly so.

Uniform Cost Search can also be used as Breadth First Search if all the edges are given a cost of 1.

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Depth-first search

- Breadth-first-search uses a FIFO queue, **depth-first search uses a LIFO queue.**
- A LIFO queue means that the most recently generated node is chosen for expansion.
- This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn, was the deepest unexpanded node when it was selected.



Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Depth-limited search

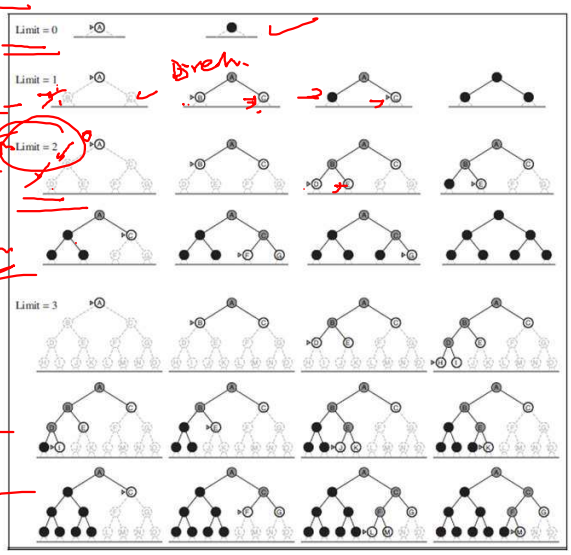
- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit .
- That is, nodes at depth are DEPTH-LIMITED treated as if they have no successors.
- This approach is called depth-limited search. The SEARCH depth limit solves the infinite-path problem.

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Iterative deepening search

- Iterative deepening search (or iterative deepening depth-first search) is a general strategy, DEEPENING SEARCH often used in combination with depth-first tree search, that finds the best depth limit.
- It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.



The iterative deepening search algorithm, which repeatedly applies depth limited search with increasing limits. It terminates when a solution is found or if the depth limited search returns failure, meaning that no solution exists.

Uninformed (blind) Search Techniques

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
- **Time complexity:** How long does it take to find a solution?
- **Space complexity:** How much memory is needed to perform the search?
- **Optimality:** Does the strategy find the optimal solution?

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes^c	Yes	No	No	Yes^c

b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit.

Superscript: a complete if b is finite; b complete if step costs ≥ for positive ; c optimal if step costs are all identical; d if both directions use breadth-first search.

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Uninformed (blind) Search Techniques

Uninformed search methods have access only to the problem definition.

- **Breadth-first search** expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space complexity.

- **Uniform-cost search** expands the node with lowest path cost, g(n), and is optimal for general step costs.

-- **Depth-first search** expands the deepest unexpanded node first. It is neither complete nor optimal, but has linear space complexity. **Depth-limited search** adds a depth bound.

- **Iterative deepening search** calls depth-first search with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to breadth-first search, and has linear space complexity.

Informed (Heuristic) Search Techniques

INFORMED SEARCH strategy

- uses problem-specific knowledge beyond the definition of the problem itself

- can find solutions more efficiently than can an uninformed strategy.

- **Best First Search**
- **Greedy best-first search**
- **A* search**
- **Memory-bounded heuristic search**

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Informed (Heuristic) Search Techniques

Best First Search

explores a graph by expanding the most promising node chosen according to a specified rule.

- A node is selected for expansion based on an **evaluation function, f(n).**
- The evaluation function is construed as a cost estimate, so the node with the **lowest evaluation is expanded first.**
- The implementation of best-first graph search is identical to that for **uniform-cost search except for the use of f instead of g to order the priority queue.**
- Evaluation function is used to decide which among the various available nodes is the most promising (or 'BEST') before traversing to that node.

Best First Search

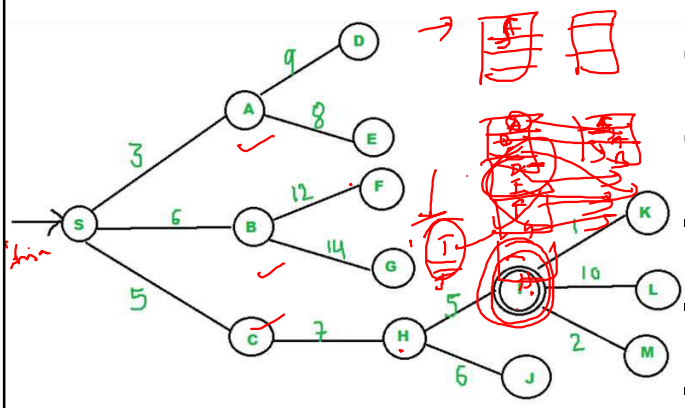
- The Best first search uses the concept of a **Priority queue and heuristic search.**
- To search the graph space, the BFS method uses two lists for tracking the traversal.
- **An 'Open' list which keeps track of the current 'immediate' nodes available for traversal and**
- **'CLOSED' list that keeps track of the nodes already traversed.**

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until GOAL node is reached
   a) If OPEN list is empty, then EXIT the loop returning 'False'
   b) Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also capture the information of the parent node
   c) If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
   d) If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
   e) Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

Best First Search



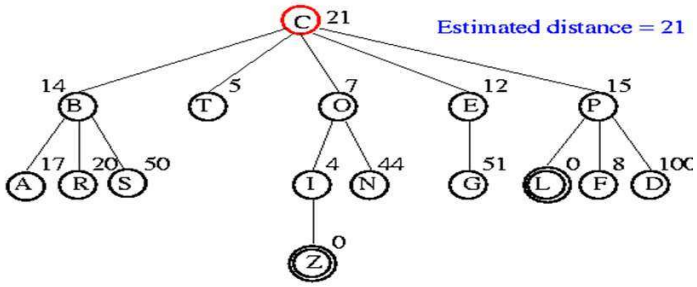
The time complexity of the algorithm is given by O(n*logn)

- We start from source "S" and search for goal "I" using given costs and Best First search.
- Priority Queue (pq) initially contains s. We remove s from and process unvisited neighbors of S to pq.
- pq now contains {A, C, B} **(C is put before B because C has lesser cost)**
- We remove A from pq and process unvisited neighbors of A to pq. pq now contains {C, B, E, D}
- We remove C from pq and process unvisited neighbors of C to pq. pq now contains {B, H, E, D}
- We remove B from pq and process unvisited neighbors of B to pq. pq now contains {H, E, D, F, G}
- We remove H from pq. Since our goal "I" is a neighbor of H, we return.

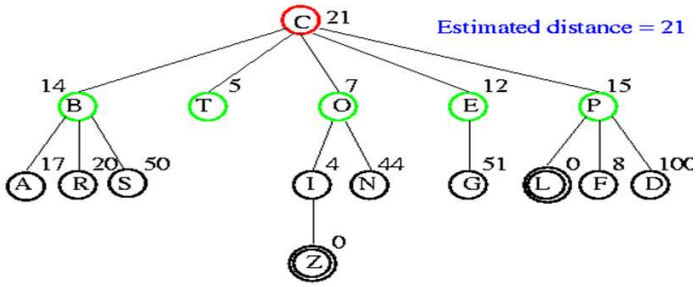
This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Example

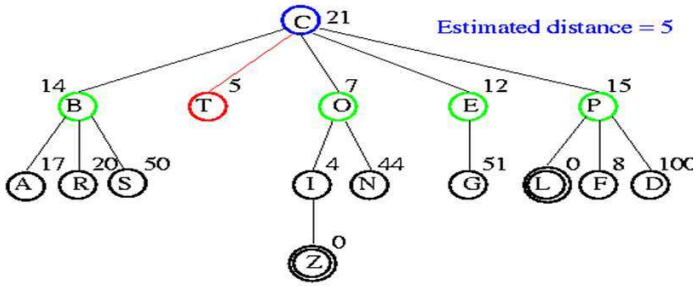


Example

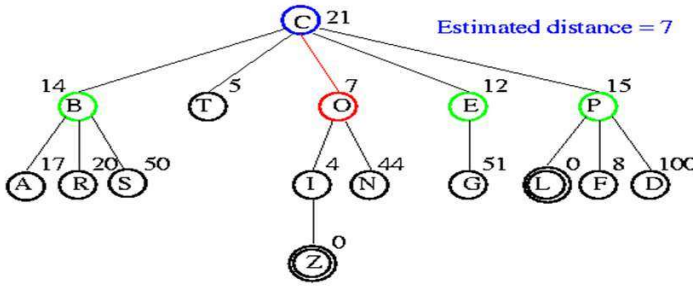


Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Example



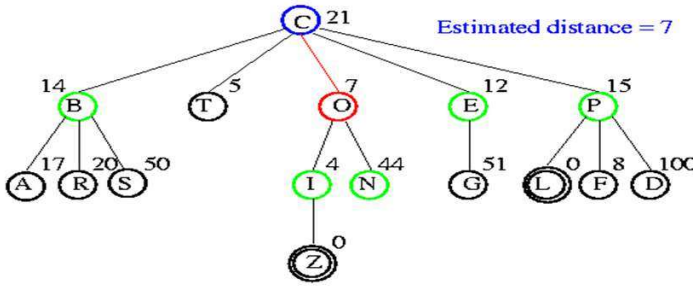
Example



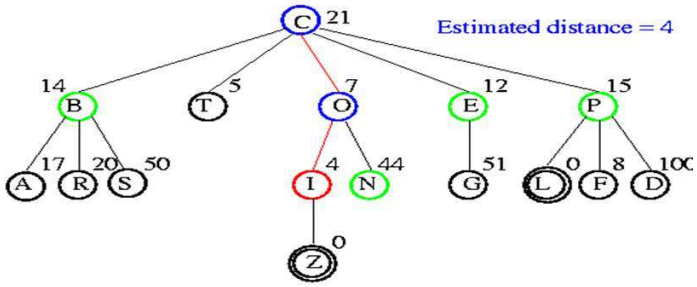
This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Example

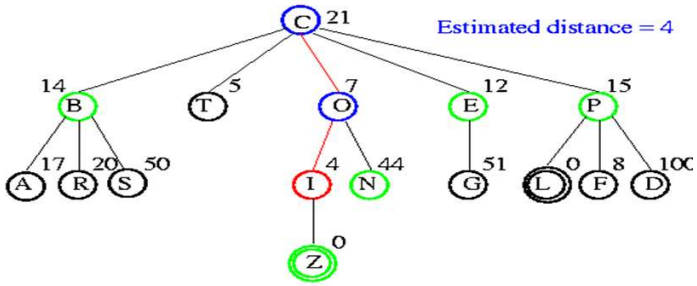


Example

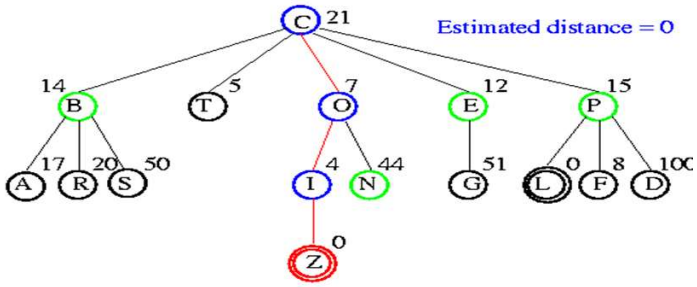


Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Example



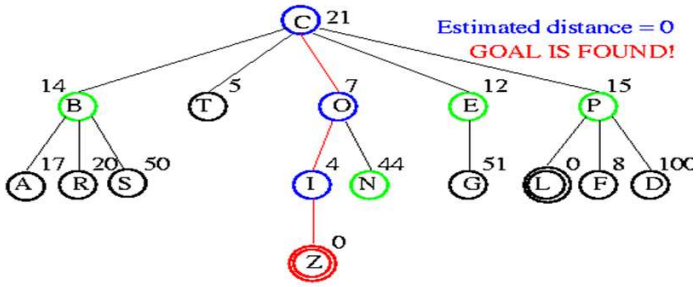
Example



This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Example



Best First Search

- The two variants of Best First Search are **Greedy Best First Search** and **A* Best First Search.**
- The Greedy BFS algorithm selects the path which appears to be the best, it can be known as the combination of depth-first search and breadth-first search.
- Greedy BFS makes use of Heuristic function and search and allows us to take advantages of both algorithms.

h(n) f(n)

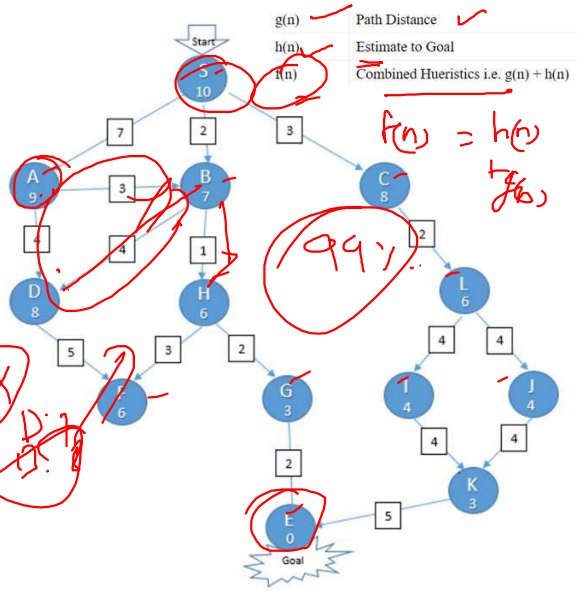
This distance is an approximation of how close we are to the goal from a given node and is denoted by the heuristic function h(n). This heuristic value is mentioned within each node.

The sum of the distance from the start node to each of these immediate next node is denoted by the function g(n).

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Greedy Best First Search

- At any point, the decision on which city to go next is governed by our evaluation function. The city which gives the least value for this evaluation function will be explored first.
- The only difference between Greedy BFS and A* BFS is in the evaluation function.
- For Greedy BFS the evaluation function is f(n) = h(n) while for A* the evaluation function is f(n) = g(n) + h(n).
- Essentially, since A* is more optimal of the two approaches as it also takes into consideration the total distance travelled so far i.e. g(n).



Greedy Best First Search

Greedy BFS with evaluation function f(n) = h(n)			
Step 1 - Start by adding the start node (S) to the open list with the path distance as 0			
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
S	10		
Repeat the next steps until the OPEN List is empty or the Goal node is moved to the CLOSED list			

Step 2 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list			
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
A	9	S	
B	7		
C	8		

Step 2 (b) - Re-order the list in ascending order of the combined hueristic value			
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
B	7	S	
C	8		
A	9		

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Greedy Best First Search

Step 3 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
C	8	S	
A	9	B	S
D	8		
H	6		

Step 3 (b) - Re-order the list in ascending order of the combined hueristic value

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
H	6	S	
C	8	B	S
D	8		
A	9		

Step 4 (b) - Re-order the list in ascending order of the combined hueristic value

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
G	3	S	
F	6	B	S
C	8	H	B
D	8		
A	9		

Step 4 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
C	8	S	
D	8	B	S
A	9	H	B
F	6		
G	3		

Greedy Best First Search

Step 5 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
F	6	S	
C	8	B	S
D	8	H	B
A	9	G	H
E	0		

Step 5 (b) - Re-order the list in ascending order of the combined hueristic value

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
E	0	S	
F	6	B	S
C	8	H	B
D	8	G	H
A	9		

Step 6 (a) - Move the first node in the OPEN list to the CLOSED list and expand it's immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
F	6	S	
C	8	B	S
D	8	H	B
A	9	G	H
		E	G

EXIT returning 'True' as the Goal node (E) is moved to the CLOSED list. Backtrack the closed list to get the optimal path (E --> G --> H --> B --> S)

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

A* Search

- QueueingFn is sort-by-f
  - f(n) = g(n) + h(n)
- Note that UCS and Best-first both improve search
  - UCS keeps solution cost low
  - Best-first helps find solution quickly
- A* combines these approaches

A* Search

A* BFS with evaluation function f(n) = ~~h(n) +~~ g(n)

Step 1 - Start by adding the start node (S) to the open list with the path distance as 0

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
S	0	10	10		

Repeat the next steps until the OPEN List is empty or the Goal node is moved to the CLOSED list

Step 2 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
A	7	9	16	S	
B	2	7	9		
C	3	8	11		

Step 2 (b) - Re-order the list in ascending order of the combined hueristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
B	2	7	9	S	
C	3	8	11	B	S
A	7	9	16		

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

A* Search

Step 3 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
A	7	9	16	B	S
D	6	8	14		
H	3	6	9		

Step 3 (b) - Re-order the list in ascending order of the combined hueristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
H	3	6	9	S	
C	3	8	11	B	S
D	6	8	14		
A	7	9	16		

A* Search

Step 4 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
D	6	8	14	B	S
A	7	9	16	H	B
F	6	6	12		
G	5	3	8		

Step 4 (b) - Re-order the list in ascending order of the combined hueristic value

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
G	5	3	8	S	
C	3	8	11	B	S
F	6	6	12	H	B
D	6	8	14		
A	7	9	16		

Step 5 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
F	6	6	12	B	S
D	6	8	14	H	B
A	7	9	16	G	H
E	7	0	7		

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

A* Search

Step 5 (b) - Re-order the list in ascending order of the combined hueristic value

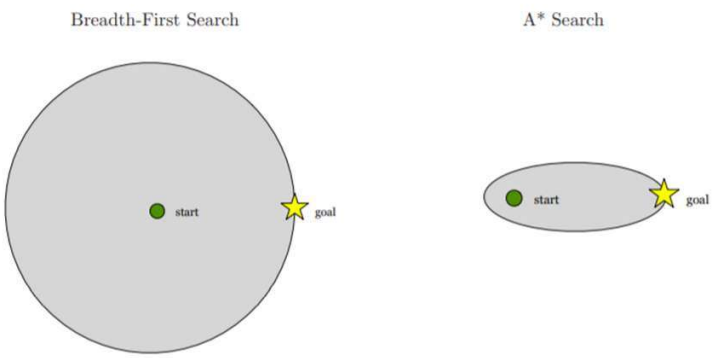
OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
E	7	0	7	S	
C	3	8	11	B	S
F	6	6	12	H	B
D	6	8	14	G	H
A	7	9	16		

Step 6 (a) - Move the first node in the OPEN list to the CLOSED list and expand it's immediate successors by adding them to the OPEN list

OPEN				CLOSED	
Node	g(n)	h(n)	f(n)	Node	Parent Node
C	3	8	11	S	
F	6	6	12	B	S
D	6	8	14	H	B
A	7	9	16	G	H
				E	G

EXIT returning 'True' as the Goal node (E) is moved to the CLOSED list. Backtrack the closed list to get the optimal path (E --> G --> H --> B --> S)

A* Search



- Breadth-first search explores a much larger portion of the search space while A* search explores only the portion that appears most promising.

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Greedy BFS Vs A*

- Even though you would find that both Greedy BFS and A* algorithms find the path equally efficiently, number of steps, you may notice that the A* algorithm is able to come up with is a more optimal path than Greedy BFS.
- Both Greedy BFS and A* are Best first searches but Greedy BFS is neither complete, nor optimal whereas A* is both complete and optimal.
- However, **A* uses more memory than Greedy BFS**, but it guarantees that the path found is optimal.

Memory Bounded Heuristic Search

- To reduce memory- Iterative deepening to the heuristic search.

  1) RBFS (recursive best-first search).

  2) Iterative Deepening Search

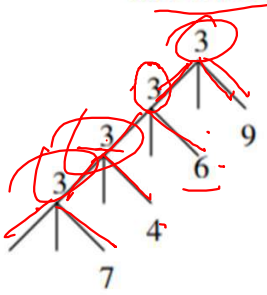
  3) MA* (Memory-bounded A*) and SMA*(simplified memory MA*)

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Memory Bounded Heuristic Search

Recursive Best-First search
- similar to recursive depth-first-search

Idea: keep expanding nodes along "best" path
keep track of nodes seen so far
when we expand a node, we replace the cost on each node of path with the cost of the current best child.



Since 7 is not the smallest thing we've seen so far, we back track to smallest thing and forget tree that backed up over.

Memory Bounded Heuristic Search

Iterative Deepening A* (IDA*)
- Similar to Iterative Deepening Search, but cut off at (g(n)+h(n)) > max instead of depth > max
- At each iteration, cutoff is the first f-cost that exceeds the cost of the node at the previous iteration

Simple Memory Bounded A* (SMA*)
- Set max to some memory bound
- If the memory is full, to add a node drop the worst (g+h) node that is already stored
- Expands newest best leaf, deletes oldest worst leaf

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Local Search Algorithms & Optimization

The informed and uninformed search expands the nodes systematically in two ways:

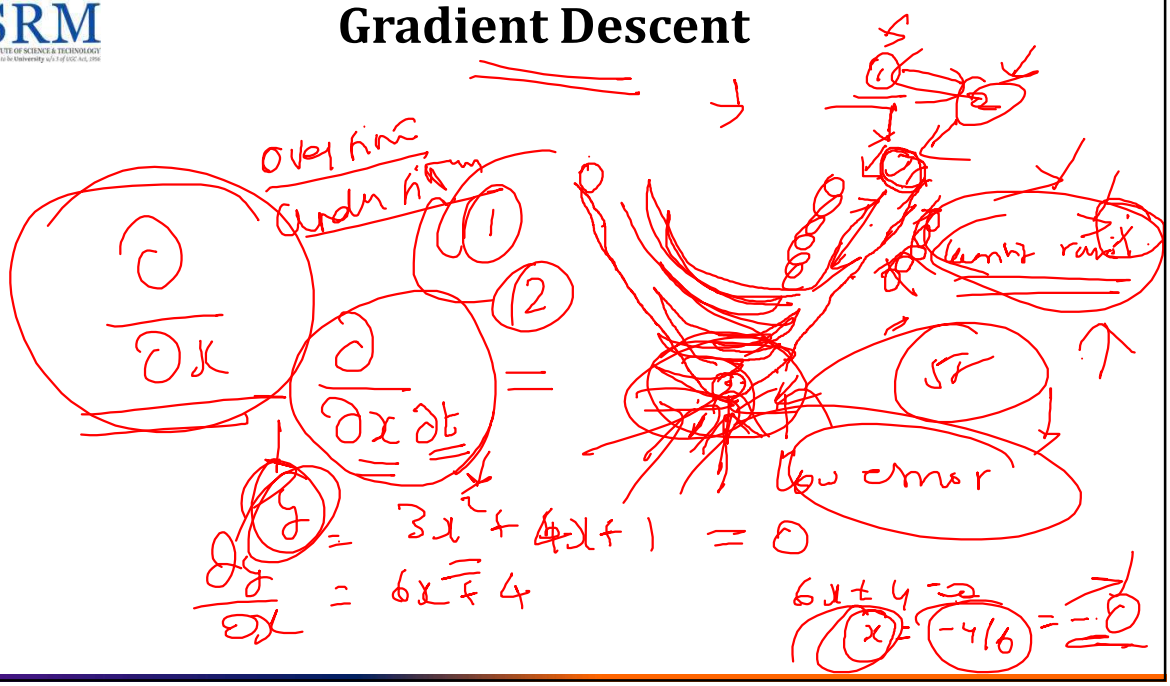
- ❑ keeping different paths in the memory and
- ❑ selecting the best suitable path, leads to a solution state required to reach the goal node

- ➢ **"local search algorithms"** where the **path cost does not matters, and only focus on solution-state needed to reach the goal node.**
- ➢ A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbors of that node generally.

Optimization Problem

An objective function is a function whose value is either minimized or maximized in different contexts of the optimization problems. In the case of search algorithms, an objective function can be the path cost for reaching the goal node, etc.

Gradient Descent



Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Local Search Algorithms & Optimization

Local search algorithms are not systematic, still they have the following advantages:

1. **Local search algorithms use a very little or constant amount of memory as they operate only on a single path.**
2. **Most often, they find a reasonable solution in large or infinite state spaces where the classical or systematic algorithms do not work.**

The local search algorithm works for pure optimized problems. A pure optimization problem is one where all the nodes can give a solution. But the target is to find the best state out of all according to the objective function. Unfortunately, the pure optimization problem fails to find high-quality solutions to reach the goal state from the current state.

Types of local searches:
- **Hill-climbing Search**
- **Simulated Annealing**

Hill Climbing

- Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.
- Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, mathematical optimization problems implies that hill-climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs.

Example-Travelling salesman problem where we need to minimize the distance traveled by the salesman.

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Hill Climbing

Step 1 : Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2 : Loop until the solution state is found or there are no new operators present which can be applied to the current state.

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

b) Perform these to evaluate new state

i. If the current state is a goal state, then stop and return success.

ii. If it is better than the current state, then make it current state and proceed further.

iii. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3 : Exit.

Hill Climbing

Features ← Inputs

- Variant of **generate and test** algorithm

1. Generate possible solutions.
2. Test to see if this is the expected solution.
3. If the solution has been found quit else go to step 1.

- Uses the **Greedy approach**

Gradient Descent

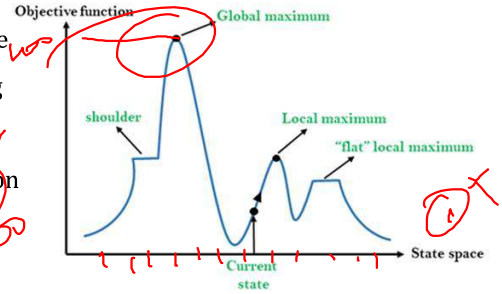
At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Hill Climbing

Hill climbing cannot reach the optimal / best state (global maximum) if it enters Local Maximum or Plateaus or ridges

Local maximum : At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.

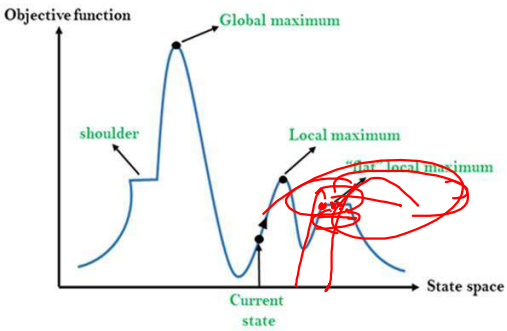


To overcome local maximum problem :
Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

Hill Climbing

Plateau : On plateau all neighbors have same value .
Hence, it is not possible to select the best direction.

To overcome plateaus :
Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region



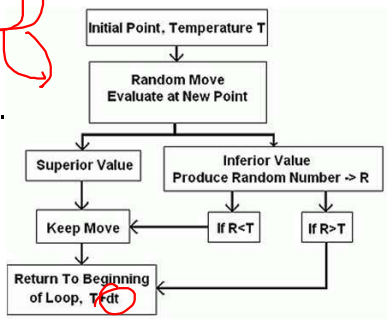
Ridge : It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.

To overcome Ridge :
In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

This document is available free of charge on studocu
Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Simulated Annealing

- Pure hill climbing is not complete, but pure random search is inefficient.
- Simulated annealing offers a compromise.



Inspired by **annealing process of gradually cooling a liquid until it changes to a low-energy state.**

- Very similar to hill climbing, except include a user-defined temperature schedule.
- When temperature is "high", allow some random moves.
- When temperature "cools", reduce probability of random move.
- If T is decreased slowly enough, guaranteed to reach best state.

Important Concepts of Game theory

Game theory attempts to find the state of equilibrium. In equilibrium, each player of the game adopts a strategy, which he is unlikely to change. Equilibrium depends on the field of application, although it may coincide.

Game Equilibrium

Nash Equilibrium

Equilibrium is about selection of a stable state. In case of Nash equilibrium, it involves multiple players (two or more) and the following rules:

1. The participants (players) in the game are aware of the equilibrium strategies of the other participants.
2. None of the participant can gain by selecting or changing the strategy unilaterally (that would affect only one player).
3. If a participant has taken up or selected a specific strategy, then none of the other participants can get any gain by means of change in their strategy (where the other participants do not have any change in their strategy).

Mohan

		Confess	Deny
Rohan	Confess	3/3	1/5
	Deny	5/1	2/2

Pay off Matrix

Prisoner's Dilemma Problem

Example

Rohan

1. If confess, when Mohan has also confessed: 3 days of suspension
2. If confess, when Mohan has denied: 1 day of suspension
3. If deny, when Mohan has confessed: 5 days of suspension
4. If deny, when Mohan has denied: 2 days of suspension

Dr. R. Rajkamal 60

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Game As Search Problem

- Any intelligent game results in to several possible moves from all the involved players.
- At any stage of the game, several options are available to a player for making the next move.
- Optimally correct choice in a state may result in superior gains.

Several AI search techniques such as iterative deepening, A*, DFS, secondary search, etc. have been successfully applied to different categories of games.

- Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.
- It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

Minimax Algorithm

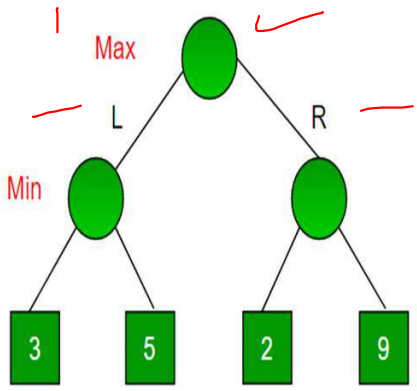
- In Minimax the two players are called maximizer and minimizer.
- The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Every board state has a value associated with it.
- In a given state, if the maximizer has upper hand then, the score of the board will tend to be some positive value.
- If the minimizer has the upper hand in that board state then it will tend to be some negative value.
- The values of the board are calculated by some heuristics which are unique for every type of game.

This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

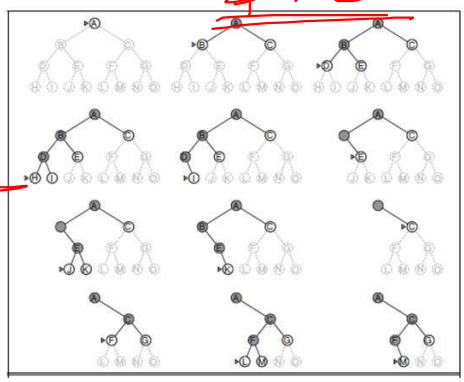
Minimax Algorithm

- Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown.
- Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level.
- **Which move you would make as a maximizing player considering that your opponent also plays optimally?**



Minimax Algorithm

- There are two players one is called Maximizer and other is called Minimizer.
- **Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.**
- This algorithm **applies DFS**, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the **terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.**



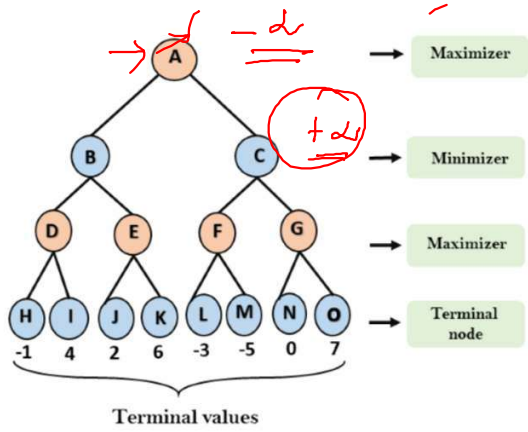
Depth First Search

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Minimax Algorithm

Step-1:

- In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.
- In the tree diagram, let's take A is the initial state of the tree.
- **Suppose maximizer takes first turn which has worst-case initial value = -infinity, and minimizer will take next turn which has worst-case initial value = +infinity.**



Minimax Algorithm

Step 2:

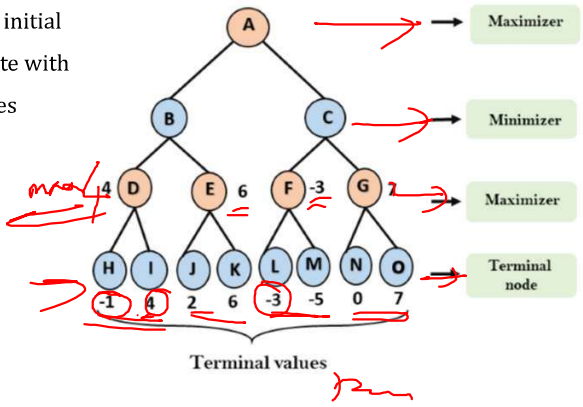
- Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values.
- It will find the maximum among the all.

For node D max(-1,-∞) => max(-1,4)= 4

For Node E max(2, -∞) => max(2, 6)= 6

For Node F max(-3, -∞) => max(-3,-5) = -3

For node G max(0, -∞) = max(0, 7) = 7



This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

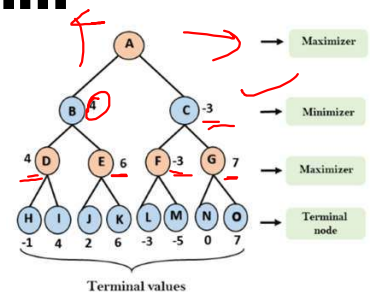
Minimax Algorithm

Step 3:

In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

For node B= min(4,6) = 4

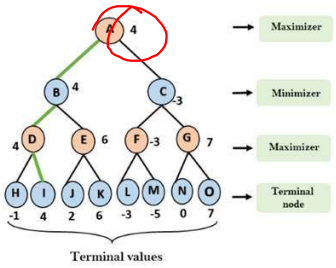
For node C= min (-3, 7) = -3



Step 4:

- Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.
- In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

For node A max(4, -3)= 4



Minimax Algorithm

Complete-

Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.

Optimal-

Min-Max algorithm is optimal if both opponents are playing optimally.

Time complexity-

As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is O(bm), where b is branching factor of the game-tree, and m is the maximum depth of the tree.

Space Complexity-

Space complexity of Mini-max algorithm is also similar to DFS which is O(bm).

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Alpha-Beta Pruning

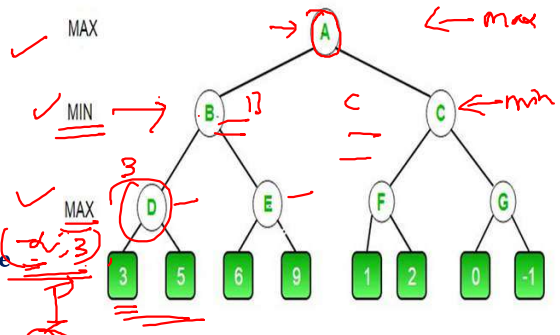
- Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm.
- It reduces the computation time by a huge factor.
- **This allows us to search much faster and even go into deeper levels in the game tree.**
- It cuts off branches in the game tree which need not be searched because there already exists a better move available.
- **It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.**
- **Alpha is the best value that the maximizer currently can guarantee at that level or above.**
  Beta is the best value that the minimizer currently can guarantee at that level or above.

Parameters to be passed to minmax algorithm

1. Player (maximising or minimising player)
2. Position (the node being evaluated)
3. Alpha (value of alpha at the node)
4. Beta (value of beta at the node)

Alpha-Beta Pruning

- The initial call starts from A.
- **The value of alpha here is -INFINITY and the value of beta is +INFINITY.**
- These values are passed down to subsequent nodes in the tree.
- **At A the maximizer must choose max of B and C, so A calls B first.**
- **At B it the minimizer must choose min of D and E and hence calls D first.**
- At D, it looks at its left child which is a leaf node. This node returns a value of 3. Now the value of alpha at D is max(-INF, 3) which is 3.

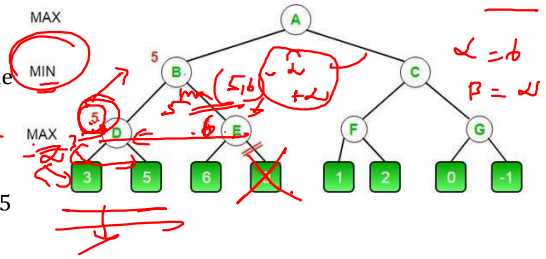


So far this is how our game tree looks. The 9 is crossed out because it was never computed.

This document is available free of charge on studocu
Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Alpha-Beta Pruning

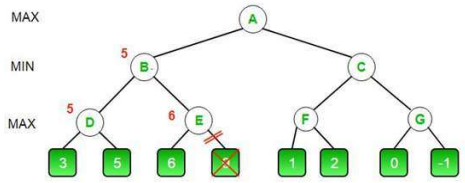
- To decide whether its worth looking at its right node or not, it checks the condition **beta<=alpha**.
- This is false since beta = +INF and alpha = 3. So it continues the search.
- D now looks at its right child which returns a value of 5.
- At D, alpha = max(3, 5) which is 5. Now the value of node D is 5
- D returns a value of 5 to B.
- At B, beta = min(+INF, 5) which is 5.
- The minimizer is now guaranteed a value of 5 or lesser.
- B now calls E to see if he can get a lower value than 5.
- At **E** the values of alpha and beta is not -INF and +INF but instead -INF and 5 respectively, because the value of beta was changed at **B** and that is what **B** passed down to **E**



- **Now E looks at its left child which is 6.**
- **At E, alpha = max(-INF, 6) which is 6.**
- **Here the condition becomes true. B**
- **beta is 5 and alpha is 6.**
- **So beta<=alpha is true.**
- **Hence it breaks and E returns 6 to B**

Alpha-Beta Pruning

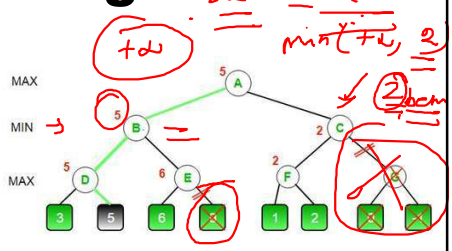
- **Note how it did not matter what the value of E's right child is.**
- **It could have been +INF or -INF, it still wouldn't matter, We never even had to look at it because the minimizer was guaranteed a value of 5 or lesser.**
- **So as soon as the maximizer saw the 6 he knew the minimizer would never come this way because he can get a 5 on the left side of B.**
- **This way we didnt have to look at that 9 and hence saved computation time.**
- E returns a value of 6 to B. At B, beta = min(5, 6) which is 5.
- The value of node B is also 5. B returns 5 to A.
- At A, alpha = max(-INF, 5) which is 5.
- Now the maximizer is guaranteed a value of 5 or greater.
- A now calls C to see if it can get a higher value than 5.
- At C, alpha = 5 and beta = +INF. C calls F
- At F, alpha = 5 and beta = +INF. F looks at its left child which is a 1. alpha = max(5, 1) which is still 5.



Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

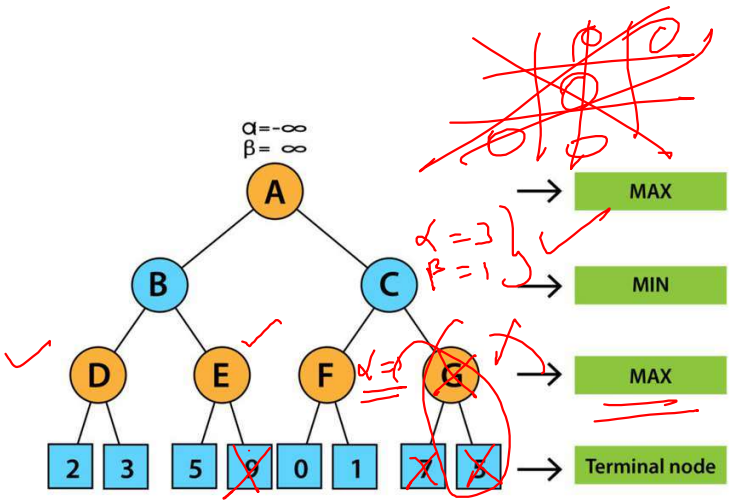
Alpha-Beta Pruning

- F looks at its right child which is a 2.
- Hence the best value of this node is 2. Alpha still remains 5.
- F returns a value of 2 to C. At C, beta = min(+INF, 2).
- The condition beta <= alpha becomes true as beta = 2 and alpha = 5. So it breaks and it does not even have to compute the entire sub-tree of G.
- The intuition behind this break off is that, at C the minimizer was guaranteed a value of 2 or lesser.
- But the maximizer was already guaranteed a value of 5 if he choose B. So why would the maximizer ever choose C and get a value less than 2 ? Again you can see that it did not matter what those last 2 values were. We also saved a lot of computation by skipping a whole sub tree.
- C now returns a value of 2 to A. Therefore the best value at A is max(5, 2) which is a 5.
- Hence the optimal value that the maximizer can get is 5



- This is how our final game tree looks like.
- As you can see G has been crossed out as it was never computed.

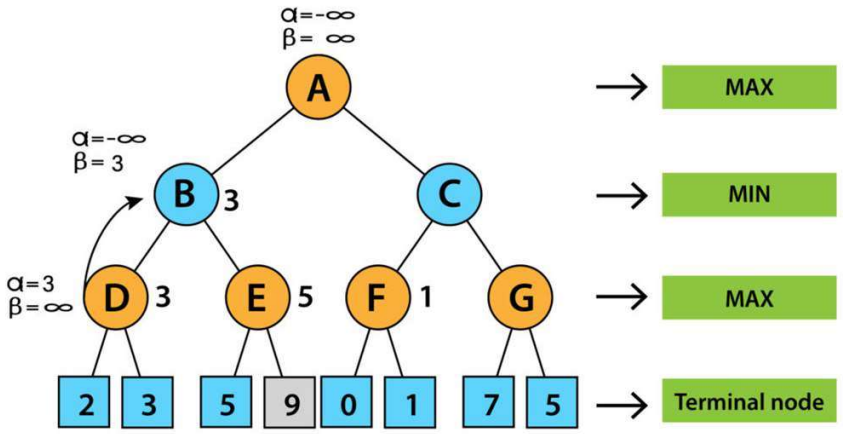
Alpha-Beta Pruning - Solve



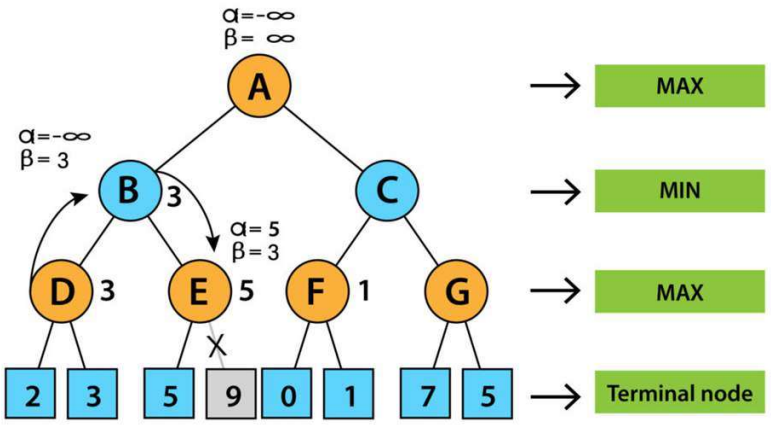
- **First start with the initial move.**
- **Initially define the alpha and beta values as the worst case i.e. α = -∞ and β= +∞.**
- **Prune the node only when alpha becomes greater than or equal to beta.**

This document is available free of charge on studocu
Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Alpha-Beta Pruning - Solve

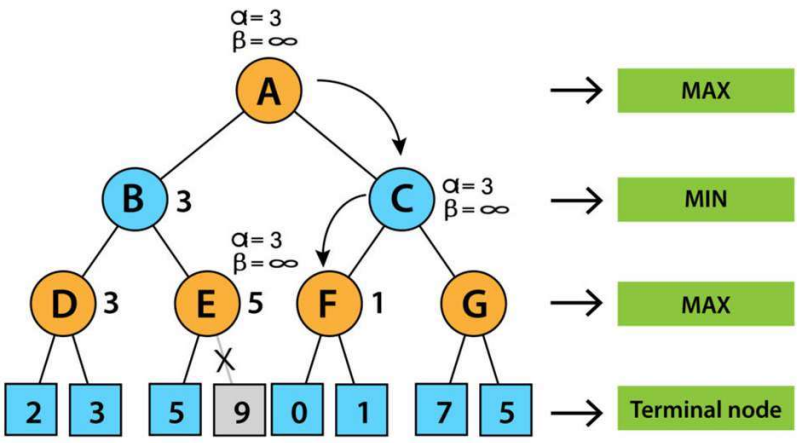


Alpha-Beta Pruning - Solve

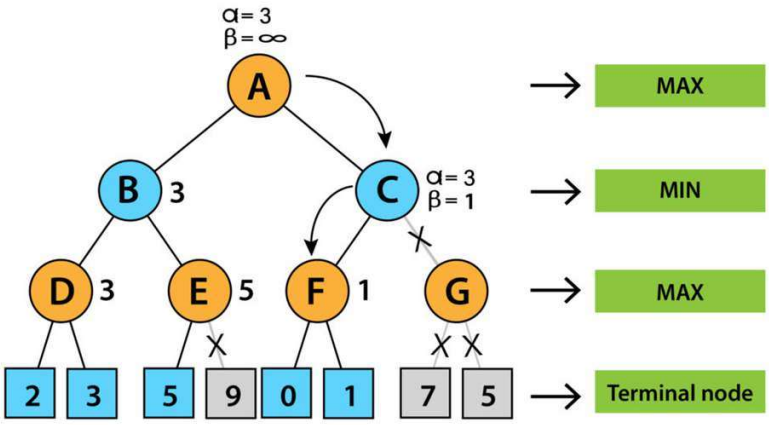


Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Alpha-Beta Pruning - Solve



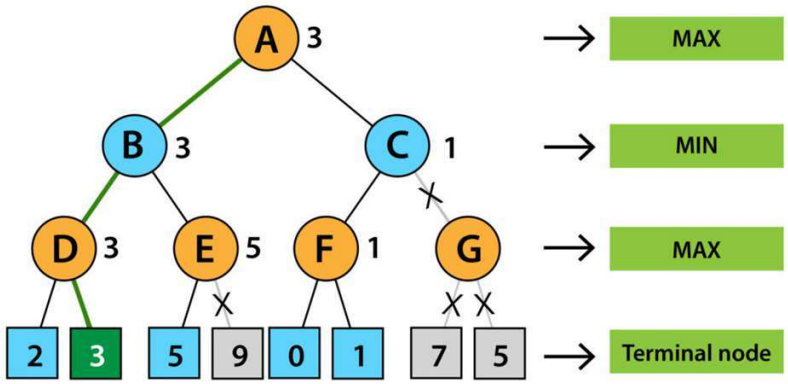
Alpha-Beta Pruning - Solve



This document is available free of charge on studocu

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)

Alpha-Beta Pruning - Solve



References

Artificial Intelligence – A Modern Approach 4e, 2020, by Stuart Russell and Peter Norvig

Artificial Intelligence 3rd edition, 2009, Elaine Rich, Kevin Knight Shivsankar Nair

ARTIFICIAL INTELLIGENCE: Building Intelligent Systems, 2015 PARAG KULKARNI, PRACHI JOSHI

Downloaded by SUPRIYA DUBEY DUBEY (supriyad@srmist.edu.in)