Artificial Intelligence

Informed (heuristic) Search

Dr. Priyadarshan Dhabe,
Ph.D (IIT Bombay),
Assistant Professor in Information Technology, VIT, Pune

Syllabus-

- **Informed (heuristic) Search strategies**

Generate and Test, Hill climbing, Best First search, A* and AO* Algorithm, Constraint satisfaction, Game playing: minimax search, Alpha –Beta cutoffs, waiting for quiescence.

What are informed (heuristic) search strategies?

-Definition:- Search techniques which "**do have additional information**" about states beyond that provided in problem definition.
-They can measure "goodness" of successors using heuristic function.
-Heuristics works as a tour guide and guide the search.

-**Example1:-** nearest neighbor heuristic function for travelling salesman problem
-**Example2:-** No of misplaced tiles in 8-puzzle

Broad Search categories

- **Strong search** techniques=Blind /Uninformed search techniques
- **Weak search** Techniques= Heuristic search Techniques

Generate and Test

- Is the simplest of all the heuristic techniques and contains following steps

Algorithm: Generate-and-Test

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

- Is a depth first search algorithm and a complete solution must be generated before they can be tested.
- If "**generation of solution**" is done systematically then this procedure will find the solution.

Generate and Test

- **Generation of solution** can be done in 2 ways

1. **Systematically:-** doing exhaustive search of search space.
2. **Randomly:-** It generates solutions randomly and in this form it is also called as " *British Museum Algorithm*" (finding a book by randomly walking)

A practical middle way is to use systematic approach and avoiding exploration of some branches which are not/less promising the goals. This can be done with the help of heuristic function.

Generate and Test-implementation

- The most straight forward way of implementing generate and test is a depth first search with backtracking.
- When a path doesn't look promising, algorithm may decide to backtrack and pursue another path.

Hill Climbing

Characteristics

- No pre-existing path
- Discover a new path
- There can be multiple paths

Two Types of hill climbing Algorithms

1. Simple Hill Climbing
2. Steepest Ascent Hill climbing

Simple Hill Climbing

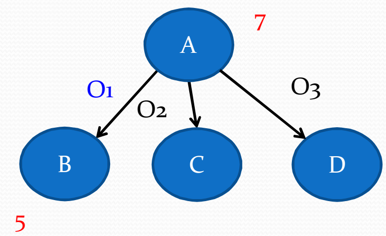
Algorithm: Simple Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   (b) Evaluate the new state.
      (i) If it is a goal state, then return it and quit.
      (ii) If it is not a goal state but it is better than the current state, then make it the current state.
      (iii) If it is not better than the current state, then continue in the loop.

A is current state with heuristic value 7

Let 3 operators are available in A viz. O1, O2 and O3

Applied **O1** and generated node B with heuristic value 5



Simple Hill Climbing

- **Drawbacks-**
- It immediately decide to move to a successor node, if it is found <u>better</u> than the current state.
- **Exercise:- Try 8-puzzle with simple hill climbing**

Steepest Ascent Hill Climbing

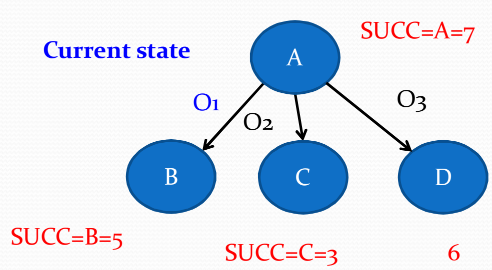
Ascent- means higher position

- Also called as **gradient search**.
- It examines all possible moves from the current state and selects the best one as the next current state.

Algorithm: Steepest-Ascent Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   (a) Let *SUCC* be a state such that any possible successor of the current state will be better than *SUCC*.
   (b) For each operator that applies to the current state do:
      (i) Apply the operator and generate a new state.
      (ii) Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to *SUCC*. If it is better, then set *SUCC* to this state. If it is not better, leave *SUCC* alone.
   (c) If the *SUCC* is better than current state, then set current state to *SUCC*.

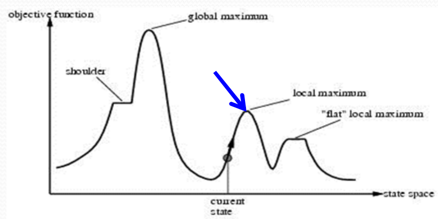
Steepest Ascent Hill Climbing



Since SUCC is **better** than the current state A, thus state C will be marked as **next current state** and search progresses

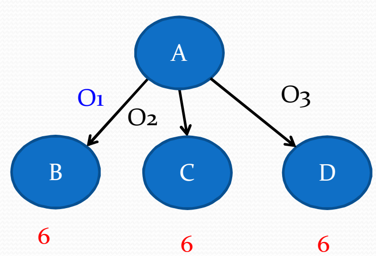
Hill Climbing-Comments

- Both methods may fail to find a solution since they are reached at a state from which no better states are generated.
- This can happen if program is reached in either a local maximum, a plateau or a ridge.
- Local maximum:- a state that is better than all its neighbors but is not better than other states farther away.



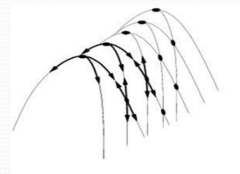
Hill Climbing-Comments

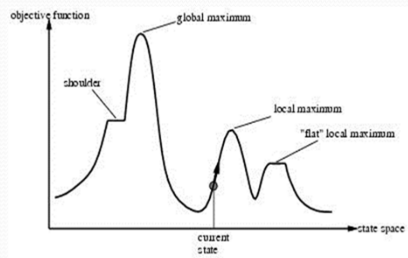
- **A plateau:-** is a flat area of the search space in which a whole set of neighboring states have the same value. On plateau it is not possible to determine best direction to move.



Hill Climbing-Comments

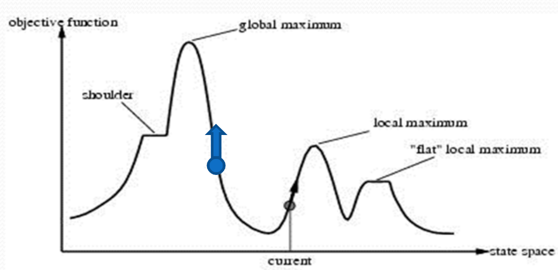
- **A ridge:-** is a sequence of **local maxima** difficult to traverse and can be trapped in.





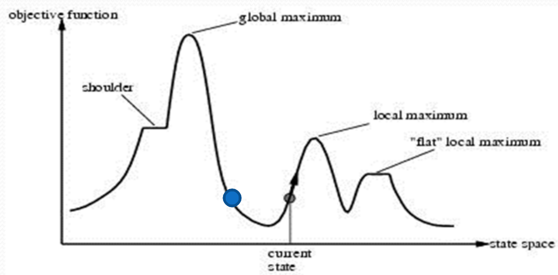
Hill Climbing-Dealing with local maxima

- Backtrack to earlier node and try going to different direction.
- For this purpose we need to maintain the list of paths already visited.



Hill Climbing-Dealing with a plateau

- Make a big jump in some direction to try to get a new section of search space.



Hill Climbing-Dealing with a ridge

- Apply two or more rules before doing the test. This corresponds to moving in several directions at once.

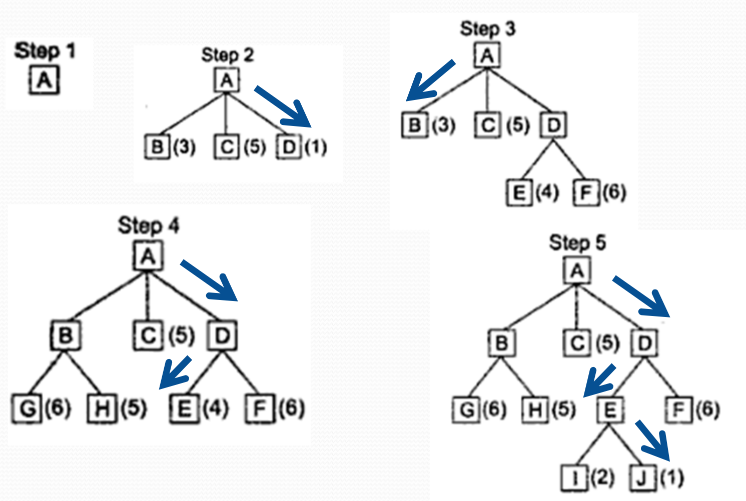
-Use of local information

Hill climbing algorithms are called **LOCAL METHODs**, since they decide what to do next? By looking at the "immediate" consequences of its choice rather than exhaustively exploring all the consequences"

Best First search

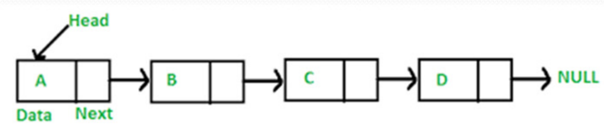
- It is **combination** of Depth First search (DFS) and Breadth First search (BFS).
  - DFS is good since it allows to find a solution without exploring all the branches.
  - BFS good since it does not trapped in dead-end paths.
- **Combination of DFS and BFS:-** Follow a single path at a time, but switch paths whenever some competing path looks **more promising** than the current one.

Best First search



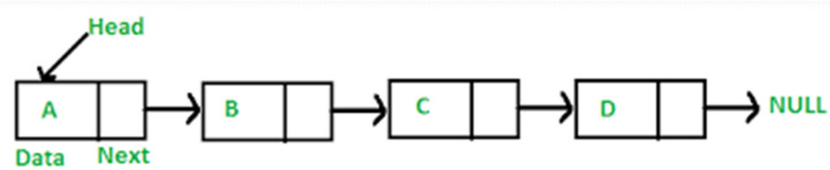
Best First search- implementation

- We need **two linked lists** for implementation of Best-First –Search
  - **1. OPEN**
  - **2. CLOSED**
- **1. OPEN:-** Contains nodes that are generated and have had heuristic function applied to them but have not been examined (i.e their successors are not created) (unprocessed). It is a priority queue and nodes with highest priority (best heuristic value) are most promising node.



Best First search- implementation

2. CLOSED:- Contains nodes that have already been examined (processed). For graph search we need this list to check if this node is generated before or not.



Best First search- implementation

- **OPEN-** contains nodes which are generated and heuristic function applied but not processed (i.e their childs are not generated)

  Choice of implementation

  1. Priority queue
  2. Ordinary liked list
- **CLOSED**-Contains nodes which are generated and processed (their childs are created)
- Both linked list shares the **same node structure**

Best First search- heuristic function

- Heuristic function:-

  $f'-$ is the heuristic function that measure the merit of a node (which is approximation of f)

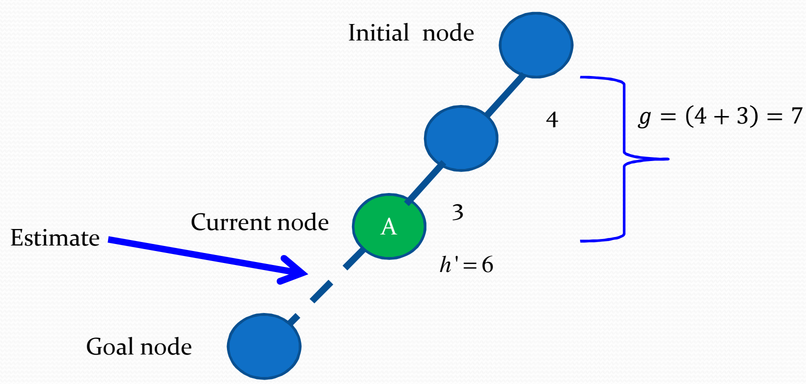
  $f'(b) = g(\mathrm{b}) + h'(b)$

$g-$ is the cost getting initial node to the current node

g is exact sum (not an estimate) of costs of all the arc.

h'- is the estimate of cost from reaching the current state to goal state

Best First search-heuristic value



$f'(A)$ =g + h'=7+6=13

Best First search-Algorithm

Abstract Working:-

-It proceeds in steps, expanding one node at each step, until it generates a node that corresponds to a goal state.

-At each step, it picks most promising node that have been generated but not expanded.

-It generates successors of the nodes, applies heuristic function to them and adds on the list of open nodes, after checking to see if any of them have been generated before.

Best First search-Algorithm

Algorithm: Best-First Search

1. Start with *OPEN* containing just the initial state.
2. Until a goal is found or there are no nodes left on *OPEN* do:
   (a) Pick the best node on *OPEN*.
   (b) Generate its successors.
   (c) For each successor do:
      (i) If it has not been generated before, evaluate it, add it to *OPEN*, and record its parent.
      (ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already, have.

Algorithm source: Rich and Knight, Artificial Intelligence

Best First search-Algorithm

- Provides a basic approach to search the graphs, but it is not a "detailed algorithm" and thus, not a systematic .
- The best-first search is a simplification of A* algorithm, developed by Hart et.al
- Reference:- Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4*. **4**(2): 100–107. doi:10.1109/TSSC.1968.300136.
- A* is proposed as an extension of Dijikstra's algorithm (1959) of shortest path- Wikipedia

Best First search Algorithm node structure

Current Node (C)	Father pointer of Node (C)	g(C)	f''(C)= g(C)+h'(C)	PTR to next node

Why g(C) need to be saved as a separate field????

A*-Algorithm

- **Applications of A* Algorithm-** Finding short path
  - Gaming
  - Route planning
  - Traffic navigational system
  - Robot navigation

A*-Algorithm

Algorithm source: Rich and Knight, Artificial Intelligence

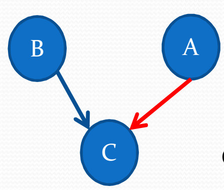
Algorithm: A*

1. Start with *OPEN* containing only the initial node. Set that node's *g* value to 0, its *h'* value to whatever it is, and its *f'* value to *h'* + 0, or *h'*. Set *CLOSED* to the empty list.
2. Until a goal node is found, repeat the following procedure: If there are no nodes on *OPEN*, report failure. Otherwise, pick the node on *OPEN* with the lowest *f'* value. Call it *BESTNODE*. Remove it from *OPEN*. Place it on *CLOSED*. See if *BESTNODE* is a goal node. If so, exit and report a solution (either *BESTNODE* if all we want is the node or the path that has been created between the initial state and *BESTNODE* if we are interested in the path). Otherwise, generate the successors of *BESTNODE* but do not set *BESTNODE* to point to them yet. (First we need to see if any of them have already been generated.) For each such *SUCCESSOR*, do the following:
   (a) Set *SUCCESSOR* to point back to *BESTNODE*. These backwards links will make it possible to recover the path once a solution is found.
   (b) Compute *g*(*SUCCESSOR*) = *g*(*BESTNODE*) + the cost of getting from *BESTNODE* to *SUCCESSOR*.
   (c) See if *SUCCESSOR* is the same as any node on *OPEN* (i.e., it has already been generated but not processed). If so, call that node *OLD*. Since this node already exists in the graph, we can throw *SUCCESSOR* away and add *OLD* to the list of *BESTNODE*'s successors. Now we must decide whether *OLD*'s parent link should be reset to point to *BESTNODE*. It should be if the path we have just found to *SUCCESSOR* is cheaper than the current best path to *OLD* (since *SUCCESSOR* and *OLD* are really the same node). So see whether it is cheaper to get to *OLD* via its current parent or to *SUCCESSOR* via *BESTNODE* by comparing their *g* values. If *OLD* is cheaper (or just as cheap), then we need do nothing. If *SUCCESSOR* is cheaper, then reset *OLD*'s parent link to point to *BESTNODE*, record the new cheaper path in *g*(*OLD*), and update *f'*(OLD).

A*-Algorithm

→ New path

- **2.C- Successor is on OPEN**



SUCCESSOR=C

OLD=C

OPEN→

-	B	
……	C	…..
	f'=8	

Compare New and old paths and do the changes on OPEN using best path

OLD path f '(C) = g(C) +h'(C)= [g(B)+g(B to C)] + h'(C)

New Path f '(C) =g(C) +h'(C)= [g(A)+g(A to C)] + h'(C)

A*-Algorithm

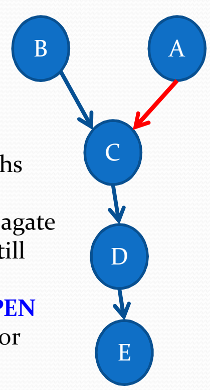
Algorithm source: Rich and Knight, Artificial Intelligence

(d) If *SUCCESSOR* was not on *OPEN*, see if it is on *CLOSED*. If so, call the node on *CLOSED OLD* and add *OLD* to the list of *BESTNODE's*, successors. Check to see if the new path or the old path is better just as in step *2(c)*, and set the parent link-and g and f' values appropriately. If we have just found a better path to *OLD*, we must propagate the improvement to *OLD's* successors. This is a bit tricky. *OLD* points to its successors. Each successor in turn points to its successors, and so forth, until each branch terminates with a node that either is still on *OPEN* or has no successors. So to propagate the new cost downward, do a depth-first traversal of the tree starting at *OLD*, changing each node's g value (and thus also its f' value), terminating each branch when you reach either a node with no successors or a node to which an equivalent or better path has already been found.[4] This condition is easy to check for. Each node's parent link points back to its best known parent. As we propagate down to a node, see if its parent points to the node we are coming from. If so, continue the propagation. If not, then its g value already reflects the better path of which it is part. So the propagation may stop here. But it is possible that with the new value of g being propagated downward, the path we are following may become better than the path through the current parent. So compare the two. If the path through the current parent is still better, stop the propagation. If the path we are propagating through is now better, reset the parent and continue propagation.

(e) If *SUCCESSOR* was not already on either *OPEN* or *CLOSED*, then put it on *OPEN*, and add it to the list of *BESTNODE's* successors. Compute $f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR)$.

A*-Algorithm

- **2.d- Successor is on CLOSED**



CLOSED--

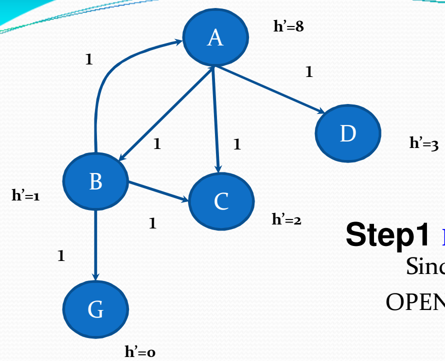
-	B	
……	C	…..
	f'=8	

Compare new and old paths and change **g** and **f'** of **SUCCESSOR=C** and propagate this new info downwards till reached at

1. A node still it is on **OPEN**
2. A node has no successor

A* ALGORITHM TRACE

A- is initial state, G- is goal node



Open--

-	
A	
f'=8	

CLOSED--EMPTY

initial

Step1 **BESTNODE=A**

Since A is not goal node , generate its successors

OPEN--**EMPTY**

CLOSED--

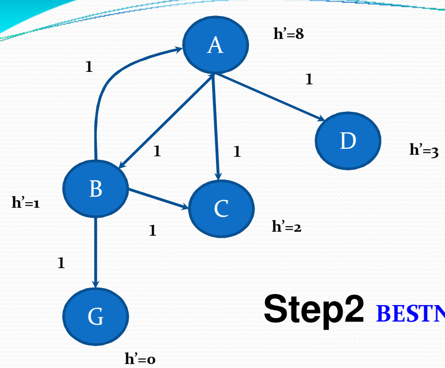
-	
A	
f'=8	

Open--

A	A	A
B	C	D
f '=2	f '=3	f '=4

CLOSED--

-	
A	
f'=8	



Open--

A	A	A
B	C	D
f '=2	f '=3	f '=4

CLOSED-

-	
A	
f'=8	

Step2 **BESTNODE=B**

Open--

A	A
C	D
f '=3	f '=4

CLOSED--

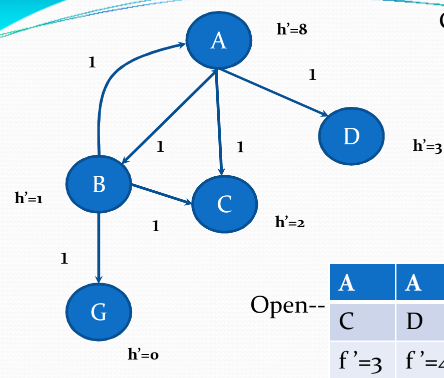
-	A
A	B
f'=8	f'=2

Since B is not goal node , generate its successors

Open--

	A	A
	C	D
	f '=3	f '=4

C- is already on OPEN, but new path to C is worse, thus don't change the parent of C



CLOSED--

-	A
A	B
f'=8	f'=2

Generate A, which is already there on CLOSED, But new path is worse than old one, so don't change parent of A

Generate node G, it is not on OPEN or CLOSED, thus add it to OPEN

Open--

A	A	B
C	D	G
f '=3	f '=4	f '=2

CLOSED--

-	A
A	B
f'=8	f'=2

Step3 **BESTNODE=G**

CLOSED--

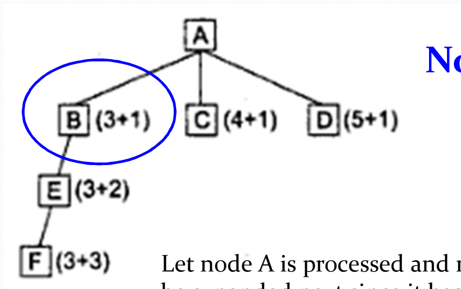
-	A	B
A	B	G
f'=8	f'=2	f '=2

Since G is goal node, output f'(G) and a path from A to G, using father pointer s on CLOSED

A*-Algorithm-Problem

- Solve **8-puzzle** using A*-Algorithm. Use no. of misplaced tiles as heuristics and assume path of reaching from a current state to successor is always 1.
- **Drawbacks of A*:-**
- It is suitable for **small** state spaces, since it creates very large number of nodes on OPEN and CLOSED, thus machine may run out of memory

A* - If h’ underestimates h

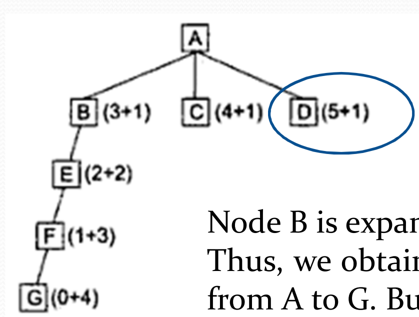


Node B is under estimated

Let node A is processed and nodes B, C and D are on OPEN. B will be expanded next since it has lowest f’=4. Let B has only successor E with value 5. Now, we have a tie to choose a node out of E and C. But, we will favor E, since it is on the same path. Let E also has only successor F with value 6. It means that, now C looks more promising and algorithm will explore branch passing through C. We wasted time and efforts in pursuing branch through B, since h’(B) underestimates h.

A*- If h’ Overestimates h

Node D is over estimated



Node B is expanded, then E, then F and then G. Thus, we obtained goal with a path length of 4 from A to G. But, if there is a direct link from D to G, then such a solution will never find it, since node D is overestimated and it looks so bad that we will find another worse solution than exploring D. This will happen if node D is overestimated

Admissibility of an algorithm

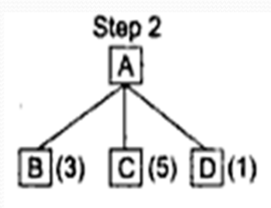
- A search algorithm is called "***Admissible***", if it ***never overestimate*** the cost of reaching to the goal (*h'*) .
- **Graceful decay of admissibility**:-

if h' rarely overestimates h by δ, then A* algorithm will rarely find a solution whose cost is more than δ greater than the cost of optimal solution.

Heuristic is admissible if for every node n
h(n) ≤ c(n), where C(n) is the actual/true evaluation of node n.
i.e it should not overestimate a node n.

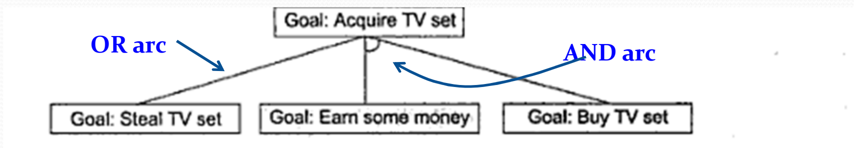
A* is **optimal** if the heuristic is **admissible**

Definition of "OR- Graphs"



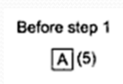
- Consider the **graph** shown. There are 3 alternate ways (branches) to solve a problem. Such graphs are called as OR graph and alternate paths are called as OR- path/Arc
- A*- Algorithm is used to search OR-Graphs

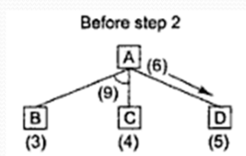
AND-OR- Graphs



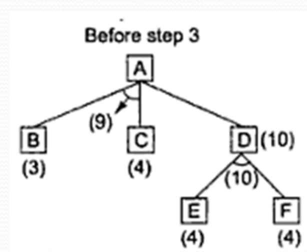
- **AND-OR Graphs:- is another structure useful in representing problems that can be decomposed into set of smaller problems, all of them need to be solved. This decomposition generate AND arcs and OR arcs, hence the name.**
- **AO* algorithm is used to search AND-OR graphs**

AO* Algorithm-working

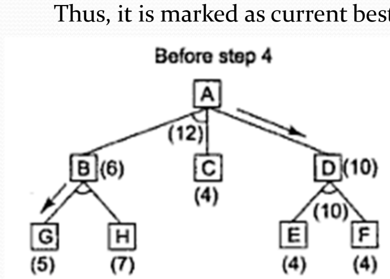




Cost of each arc is 1



Path through D looks mores promising. Thus, it is marked as current best path.



Now, path through AND arc B-C looks mores promising. Thus, it is marked as current best path.

Now, again path through D looks mores promising. Thus, it is marked as current best path.

This process continues till the solution is found or all paths leads to dead end

AO* Algorithm

- AO* uses a structure called **GRAPH**, representing the part of search graph **explicitly generated** so far.
- Each node in the **GRAPH** point to immediate successor and predecessor.
- Each node will have **h'** value associated with it and we are not storing **g** value, since there can be many paths to a node.
- Let **FUTILITY** be maximum cost with which we want to get solution, otherwise we abandon the search.

AO* Algorithm

Algorithm: AO*

1. Let *GRAPH* consist only of the node representing the initial state. (Call this node *INIT.*) Compute *h'(INIT)*
2. Until *INIT* is labeled *SOLVED* or until *INIT's h'* value becomes greater than *FUTILITY*, repeat the following procedure:
   (a) Trace the labeled arcs from *INIT* and select for expansion one of the as yet unexpanded nodes that occurs on this path. Call the selected node *NODE.*
   (b) Generate the successors of *NODE.* If there are none, then assign *FUTILITY* as the *h'* value of *NODE.* This is equivalent to saying that *NODE* is not solvable. If there are successors, then for each one (called *SUCCESSOR)* that is not also an ancestor of *NODE* do the following:
      (i) Add *SUCCESSOR* to *GRAPH.*
      (ii) If *SUCCESSOR* is a terminal node, label it *SOLVED* and assign it an *h'* value of 0.
      (iii) If *SUCCESSOR* is not a terminal node, compute its *h'* value.

AO* Algorithm

(c) Propagate the newly discovered information up the graph by doing the following: Let *S* be a set of nodes that have been labeled *SOLVED* or whose *h'* values have been changed and so need to have values propagated back to their parents. Initialize 5 to *NODE*. Until *S* is empty, repeat the, following procedure:

(i) If possible, select from *S* a node none of whose descendants in *GRAPH* occurs in *S*. If there is no such node, select any node from *S*. Call this node *CURRENT*, and remove it from *S*.

(ii) Compute the cost of each of the arcs emerging from *CURRENT*. The cost of each arc is equal to the sum of the *h'* values of each of the nodes at the end of the arc plus whatever the cost of the arc itself is. Assign as *CURRENT'S* new *h'* value the minimum of the costs just computed for the arcs emerging from it.

(iii) Mark the best path out of *CURRENT* by marking the arc that had the minimum cost as computed in the previous step.

(iv) Mark *CURRENT SOLVED* if all of the nodes connected to it through the new labeled arc have been labeled *SOLVED*.

(v) If *CURRENT* has been labeled *SOLVED* or if the cost of *CURRENT* was just changed, then its new status must be propagated back up the graph. So add all of the ancestors of *CURRENT* to *S*.

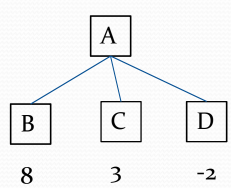
Game playing using minimax search

- In the game playing, we consider two players and a **score** associated with the game.
- A player who is trying to minimize the score (say -100) is called as minimizing player.
- A maximizing player will try to increase score (say +100) and hence the name to the algorithm.
- They plays moves alternately.
- Winning
  - Maximizing player is say +10
  - Minimizing player is -10

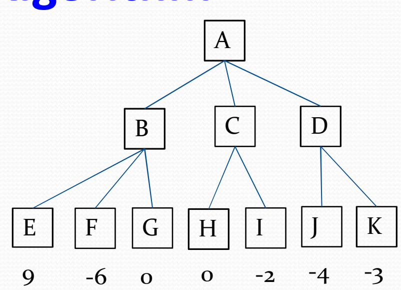
Minimax search Algorithm

- It is depth-first and depth limited search algo.
- The ides is to start at the current position and use plausible move generator to generate set of possible successor positions.
- Apply static evaluation function to those positions and choose the best one. We can back the value up to starting position, which represent current position in the game.

Minimax search Algorithm

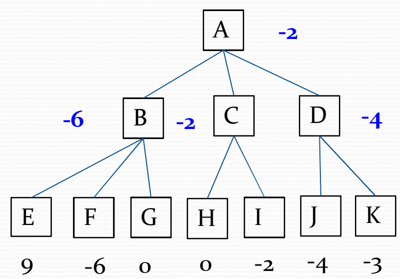


One-ply search



Two-ply search

Minimax search Algorithm



Maximizing ply

Minimizing ply

Backing up the values of a Two-ply search

A depth limit m, is a m-ply search

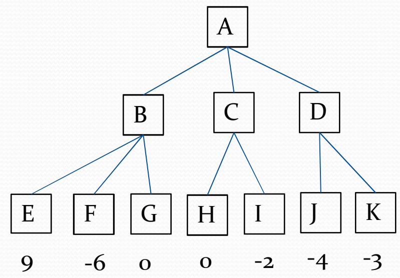
Minimax search Algorithm

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        bestValue := −∞
        for each child of node
        v := minimax(child, depth − 1, FALSE)
        bestValue := max(bestValue, v)
        return bestValue
    else (* minimizing player *)
        bestValue := +∞
        for each child of node
        v := minimax(child, depth − 1, TRUE)
        bestValue := min(bestValue, v)
    return bestValue
```

Call this function as
minimax(initialNode, depth, TRUE)

Source:-https://en.wikipedia.org/wiki/Minimax

Minimax search Algorithm trace

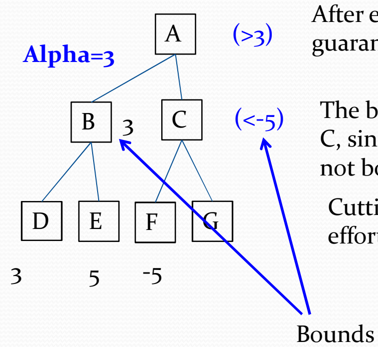


Pl. see the trace of algorithm given with the slides

Minimax search – Alpha-Beta cutoff's

- The efficiency of Depth first search (like minimax algorithm) can be improved by using branch and bound techniques, where a bound (number) is assigned to each branch and the branch is selected with the best bound.
- For a 2-player game, we need two bounds, one for each player and called **Alpha** and **Beta**.
- **Alpha-** is a threshold value representing lower bound on maximizing node.
- **Beta-** an upper bound on the value of a minimizing node.
- The updated strategy is called **Alpha-Beta pruning**

Minimax search – Alpha-Beta cutoff's



After examining node F, we know that opponent is guaranteed a score of -5, regardless of score of G.

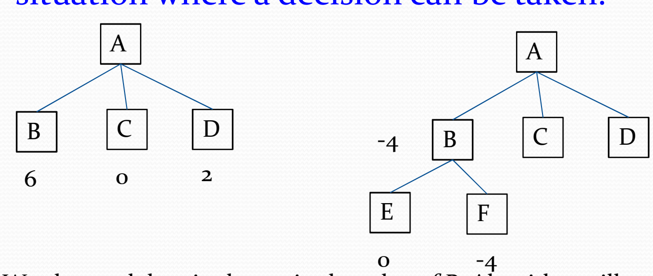
The branch through B looks better than through C, since it guarantees a +3 score. Thus, we need not bother to explore G.

Cutting down a single node saves considerable efforts in say, 6-ply search.

Alpha- is a threshold value representing lower bound on maximizing node.
Beta- an upper bound on the value of a minimizing node.

Minimax search – Waiting for quiescence

- Quiescence- means quietness or inactivity
- Waiting for Quiescence is additional refinement to Minimax search algorithm.
- It remains inactive till the search enters into a stable situation where a decision can be taken.



We observed drastic change in the value of B. Algorithm will continue the search till no drastic change occurs from one level to another is called " waiting for quiescence".

Minimax search – Waiting for quiescence

- Horizon effect:- in which an inevitable (unavoidable) bad event can be delayed by tactics until it does not appear in the portion of the game tree.

Constraint satisfaction

- Is the heuristic search technique, used to find solution that satisfy given "Constraints" and hence the name.
- The most-constrained variable heuristic to select which variable to assign a value.
- **Constraint satisfaction algorithm** uses following steps
  - 1. Constraints are **propagated** by using rules of that problem.
  - 2. A value is **guessed** for a variable whose value is still unknown.

Constraint satisfaction

- Constraint satisfaction in **AI** and **Operations Research**, is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy. A solution is therefore a set of values for the variables that satisfies all constraints—that is, a point in the feasible region.
- **Crypt arithmetic Puzzle:-**

```
   S E N D
 + M O R E
--------------
  M O N E Y
```

General Rules:
1. Each alphabet takes only one number from 0 to 9 uniquely.
2. Two single digit numbers sum can be maximum 19 with carryover. So carry over in problems of two number addition is always 1.
3. Try solve left most digit in the

Constraint satisfaction

```
  5  4  3  2  1
     S  E  N  D
+    M  O  R  E
     c3 c2 c1
----------------------
  M  O  N  E  Y
```

1. From Column 5, **M=1**, since it is only carry-over possible from sum of 2 single digit number in column 4.
2. To produce a carry from column 4 to column 5 'S + M' is atleast 9 so **'S=8or9'** so **'S+M=9or10'** & so **'O = 0 or 1'**. But 'M=1', so **'O = 0'**.
3. If there is carry from Column 3 to 4 then 'E=9' & so 'N=0'. But 'O = 0' so there is no carry & **'S=9'** & **'c3=0'**.
4. If there is no carry from column 2 to 3 then 'E=N' which is impossible, therefore there is carry & **'N=E+1'** & **'c2=1'**.
5. If there is carry from column 1 to 2 then **'N+R=E mod 10'** & 'N=E+1' so 'E+1+R=E mod 10', so 'R=9' but 'S=9', so there must be carry from column 1 to 2. Therefore **'c1=1'** & **'R=8'**.
6. To produce carry 'c1=1' from column 1 to 2, we must have 'D+E=10+Y' as Y cannot be 0/1 so D+E is atleast 12. As D is atmost 7 & E is atleast 5 (D cannot be 8 or 9 as it is already assigned). N is atmost 7 & 'N=E+1' so 'E=5or6'.

Reference:- http://aicryptarithmetics-pallavi.blogspot.in/2010/10/cryptarithmetics.html

5 4 3 2 1
S E N D
+ M O R E
c3 c2 c1

M O N E Y

SOLUTION:

9 5 6 7
+ 1 0 8 5
----
1 0 6 5 2

VALUES:
S=9
E=5
N=6
D=7
M=1
O=0
R=8
Y=2

7. If E were 6 & D+E atleast 12 then D would be 7, but 'N=E+1' & N would also be 7 which is impossible. Therefore '**E=5**' & '**N=6**'.

8. D+E is atleast 12 for that we get '**D=7**' & '**Y=2**'.

Try some problems from
http://www.campusgate.co.in/2015/10/cryptarithmetic-solution-solved-examples.html