UNIT 3 INFORMED SEARCH OR HEURISTIC SEARCH

CONTENTS

1.0 INTRODUCTION

We have seen that uninformed search methods that systematically explore the state space and find the goal. They are inefficient in most cases. Informed search methods use problem specific knowledge, and may be more efficient. Informed Search will be able to unravel the factoring an effective way if we now have relevant information, clues or hints. The clues that assist solve the factor constitute heuristic information. Informed search could also be known as heuristic search.

According to George Polya *heuristic* is the study of the methods and rules of discovery and invention. In state space search, *heuristic* define the rules for choosing branches in a state space that are most likely to lead to an acceptable solution. There are two cases in AI searches when heuristics are needed:

The problem has no exact solution. For example, in medical diagnosis doctors use heuristic to choose the most likely diagnoses given a set of symptoms.
The problem has an exact solution but is too complex to allow for a *brute force* solution.

Key Point: Heuristics are fallible. Because they rely on limited information, they may lead to a suboptimal solution or to a dead end.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

Explain informed search
Mention other names of informed search
Describe best-first search
Describe greedy search
Solve simple problems on informed search.

3.0 MAIN CONTENT

3.1 What is Heuristic?

Heuristic search methods explore the search space "intelligently". That is, evaluating possibilities without having to investigate every single possibility.

Heuristic search is an AI search technique that employs heuristic for its moves. *Heuristic* is a rule of thumb that probably leads to a solution. Heuristic play a major role in search strategies because of exponential nature of the most problems. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number. In Artificial Intelligence, heuristic search has a general meaning, and a more specialized technical meaning. In a general sense, the term heuristic is used for any advice that is often effective, but is not guaranteed to work in every case.

Heuristic means “rule of thumb”. To quote Judea Pearl, “Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal”. In heuristic search or informed search, heuristics are used to identify the most promising search path.

3.1.1 Examples of Heuristic Function

A heuristic function at a node n is an estimate of the optimum cost from the current node to a goal. It is denoted by *h (n)*.
H (n) = estimated cost of the cheapest path from node n to a goal node

Example 1: We want a path from Kolkata to Guwahati Heuristic for Guwahati may be straight-line distance between Kolkata and Guwahati *h (Kolkata) = Euclidean Distance (Kolkata, Guwahati)*

Example 2: 8-puzzle: Misplaced Tiles Heuristics is the number of tiles out of place.

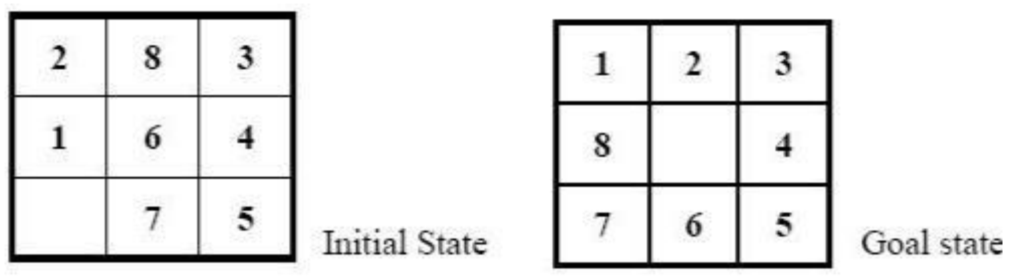


Figure 1: 8 puzzle

The first picture shows the current state *n*, and the second picture the goal state.

h (n) = 5 because the tiles 2, 8, 1, 6 and 7 are out of place.

Manhattan Distance Heuristic: Another heuristic for 8-puzzle is the Manhattan distance heuristic. This heuristic sums the distance that the tiles are out of place. The distance of a tile is measured by the sum of the differences in the x-positions and the y-positions.

For the above example, using the Manhattan distance heuristic, $h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6$

We will now study a heuristic search algorithm best-first search.

3.2 Best-First Search

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.

Judea Pearl described best-first search as estimating the promise of node *n* by a "heuristic evaluation function *f*(*n*) which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain.

Uniform Cost Search is a special case of the best first search algorithm. The algorithm maintains a priority queue of nodes to be explored. A cost function f (n) is applied to each node. The nodes are put in OPEN in the order of their f values. Nodes with smaller f (n) values are expanded earlier. The generic best first search algorithm is outlined below.

Best First Search

```
Let fringe be a priority queue containing the initial state
Loop if fringe is empty return failure Node      remove-first (fringe)
If Node is a goal
then return the path from initial state to
Node else generate all successors of Node,
and put the newly generated nodes into
fringe according to their f values End Loop
```

We will now consider different ways of defining the function f. This leads to different search algorithms.

3.2.1 Greedy Search

In greedy search, the idea is to expand the node with the smallest estimated cost to reach the goal.

We use a heuristic function
f (n) = h (n)
h (n) estimates the distance remaining to a goal.

A greedy algorithm is any algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding the global optimum. In general, greedy algorithms are used for optimization problems.

Greedy algorithms often perform very well. They tend to find good solutions quickly, although not always optimal ones.

The resulting algorithm is not optimal. The algorithm is also incomplete, and it may fail to find a solution even if one exists. This can be seen by running greedy search on the following example. A good heuristic for the route-finding problem would be straight-line distance to the goal.

S is the starting state, G is the goal state.

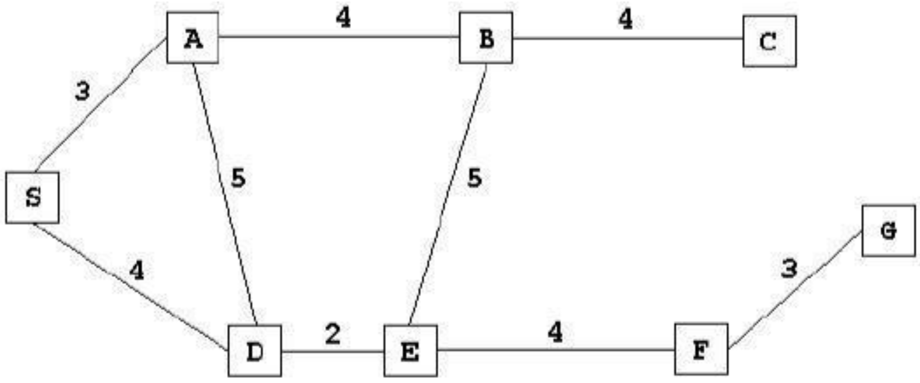


Figure 2 is an example of a route finding problem.

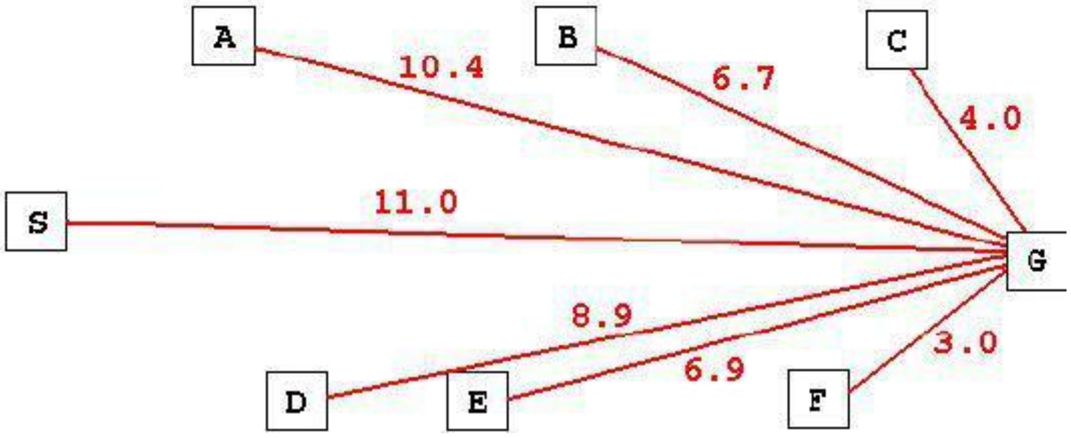
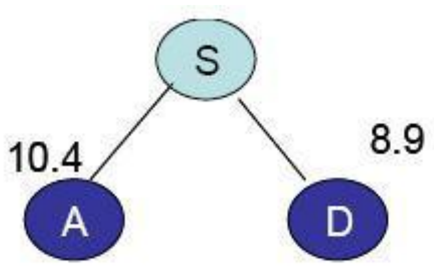


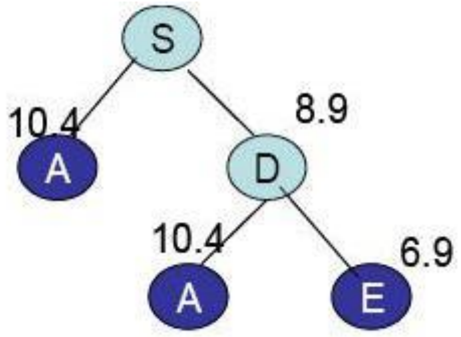
Figure 3 -The straight line distance heuristic estimates for the nodes.

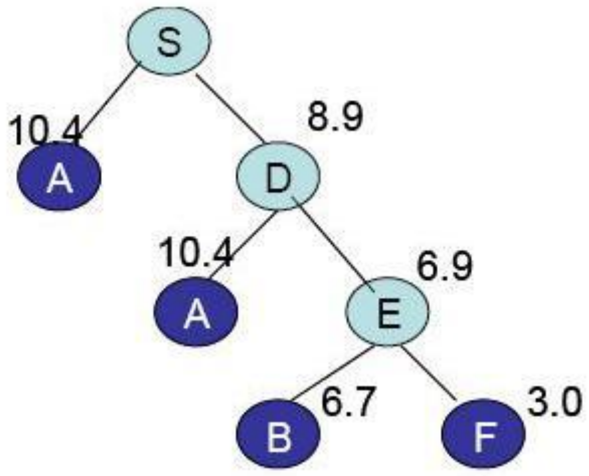
Let us run the greedy search algorithm for the graph given in Figure 2. The straight line distance heuristic estimates for the nodes are shown in Figure 3.

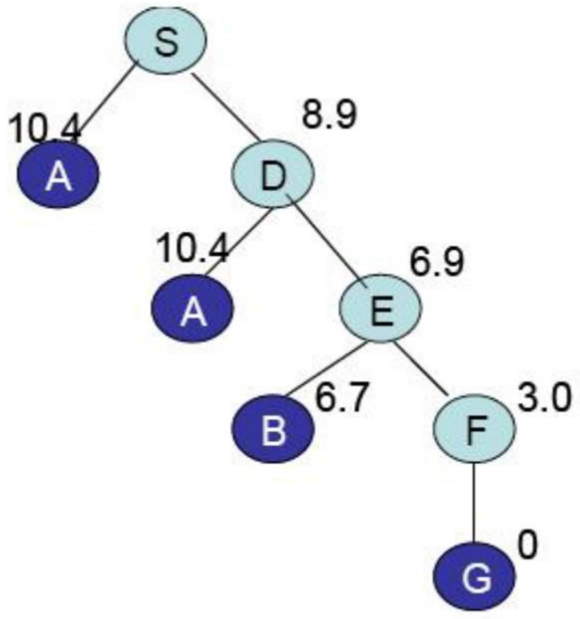
Step 1: S is expanded. Its children are A and D.



Step 2: D has smaller cost and is expanded next.







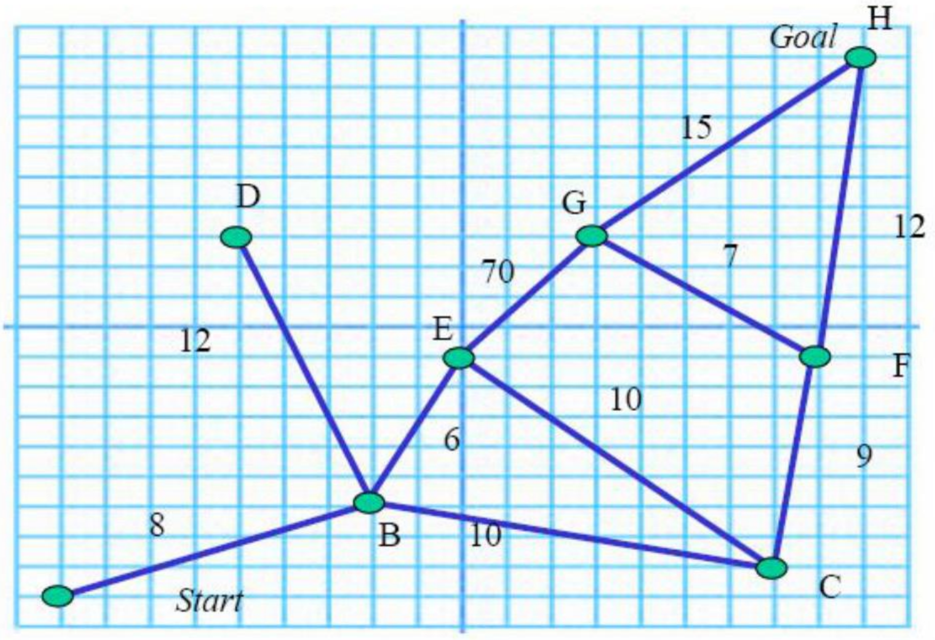


Figure 4

Greedy Best-First Search illustrated

We will run greedy best first search on the problem in Figure 2. We use the straight line heuristic. *h(n)* is taken to be the straight line distance from *n* to the goal position.

The nodes will be expanded in the following order:

A
B
E
G
H

The path obtained is A-B-E-G-H and its cost is 99
Clearly this is not an optimum path. The path A-B-C-F-H has a cost of 39.

3.2.2 A* Search

We will next consider the famous A* algorithm. This algorithm was given by Hart, Nilsson & Rafael in 1968.

A* is a best first search algorithm with
$f(n) = g(n) + h(n)$
where
$g(n)$ = sum of edge costs from start to n
$h(n)$ = estimate of lowest cost path from n to goal

$f(n)$ = actual distance so far + estimated distance remaining

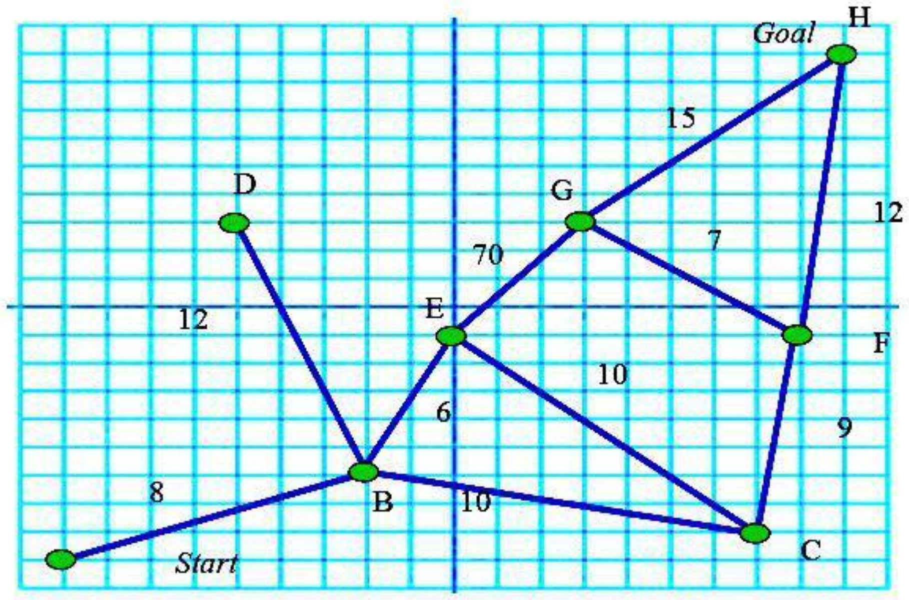
$h(n)$ is said to be admissible if it underestimates the cost of any solution that can be reached from n. If $C^*(n)$ is the cost of the cheapest solution path from n to a goal node, and if h is admissible,
$h(n) <= C^*(n)$.
We can prove that if $h(n)$ is admissible, then the search will find an optimal solution.

The algorithm A* is outlined below:

```
Algorithm A*
OPEN = nodes on frontier.    CLOSED = expanded nodes.
OPEN = {<s, nil>}
while OPEN is not empty
    remove from OPEN the node <n,p> with minimum f(n)
    place <n,p> on CLOSED
    if n is a goal node,
        return success (path p)
    for each edge connecting n & m with cost c
        if <m, q> is on CLOSED and {p|e} is cheaper than q
            then remove n from CLOSED,
                put <m,{p|e}> on OPEN
        else if <m,q> is on OPEN and {p|e} is cheaper than q
            then replace q with {p|e}
        else if m is not on OPEN
            then put <m,{p|e}> on OPEN
```

return failure

3.2.1 A* illustrated



The heuristic function used is straight line distance. The order of nodes expanded, and the status of Fringe is shown in the following table.

Steps	Fringe	Node expanded	Comments
1	A		
2	B(26.6)	A	
3	E(27.5), C(35.1), D(35.2)	B	
4	C(35.1), D(35.2), ~~C(41.2)~~ G(92.5)	E	C is not inserted as there is another C with lower cost.
5	D(35.2), F(37), G(92.5)	C	
6	F(37), G(92.5)	D	
7	H(39), G(42.5)	F	G is replaced with a lower cost node
8	G(42.5)	H	Goal test successful.

The path returned is A-B-C-F-H.
The path cost is 39. This is an optimal path.

3.2.2 A* search: properties

The algorithm A* is admissible. This means that provided a solution exists, the first solution found by A* is an optimal solution. A* is admissible under the following conditions:

- In the state space graph
    - Every node has a finite number of successors
    - Every arc in the graph has a cost greater than some $\varepsilon > 0$
- Heuristic function: for every node *n*, $h(n) \le h^*(n)$

A* is also complete under the above conditions.

A* is optimally efficient for a given heuristic – of the optimal search algorithms that expand search paths from the root node, it can be shown that no other optimal algorithn will expand fewer nodes and find a solution

However, the number of nodes searched still exponential in the worst case.

Unfortunately, estimates are usually not good enough for A* to avoid having to expand an exponential number of nodes to find the optimal solution. In addition, A* must keep all nodes it is considering in memory.

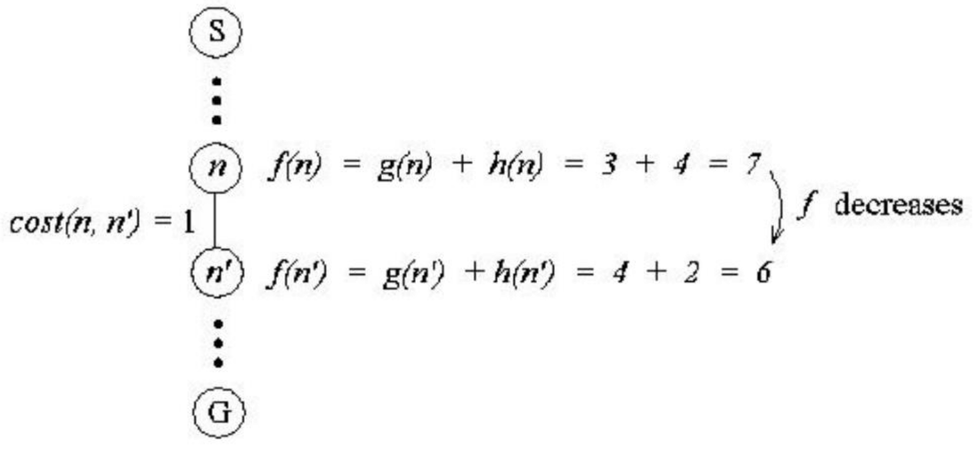
A* is still much more efficient than uninformed methods.

It is always better to use a heuristic function with higher values as long as it does not overestimate.

A heuristic is consistent if:

$h(n) <= cost(n, n') + h(n')$

For example, the heuristic shown below is inconsistent, because h(n) = 4, but $cost(n, n') + h(n') = 1 + 2 = 3$, which is less than 4. This makes the value of f decrease from node n to node n':


S
n
$f(n) = g(n) + h(n) = 3 + 4 = 7$
f decreases
$cost(n, n') = 1$
n'
$f(n') = g(n') + h(n') = 4 + 2 = 6$
G

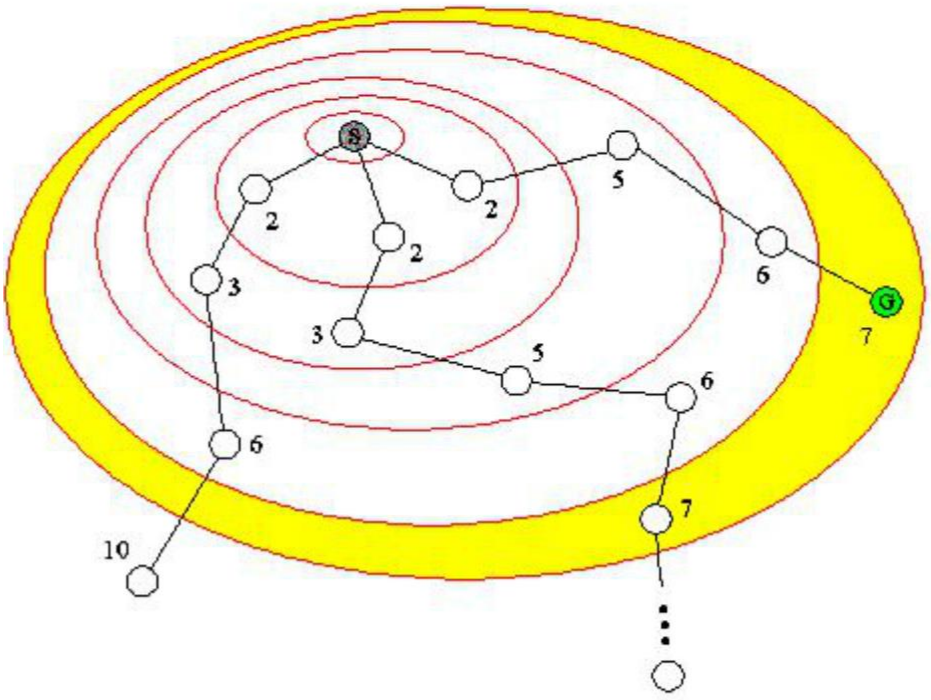
If a heuristic h is consistent, the f values along any path will be nondecreasing:

$f(n')$ = estimated distance from start to goal through n'
= actual distance from start to n + step cost from n to n' + estimated distance from n' to goal
= $g(n) + cost(n, n') + h(n')$
$\geq g(n) + h(n)$ because $cost(n, n') + h(n') \geq h(n)$ by consistency
= $f(n)$

Therefore $f(n') \geq f(n)$, so f never decreases along a path.

If a heuristic h is inconsistent, we can tweak the f values so that they behave as if h were consistent, using the **pathmax equation**:
$f(n') = \max(f(n), g(n') + h(n'))$

This ensures that the f values never decrease along a path from the start to a goal. Given nondecreasing values of f, we can think of A* as searching outward from the start node through successive **contours** of nodes, where all of the nodes in a contour have the same f value:



For any contour, A* examines all of the nodes in the contour before looking at any contours further out. If a solution exists, the goal node in the closest contour to the start node will be found first.

We will now prove the admissibility of A*.

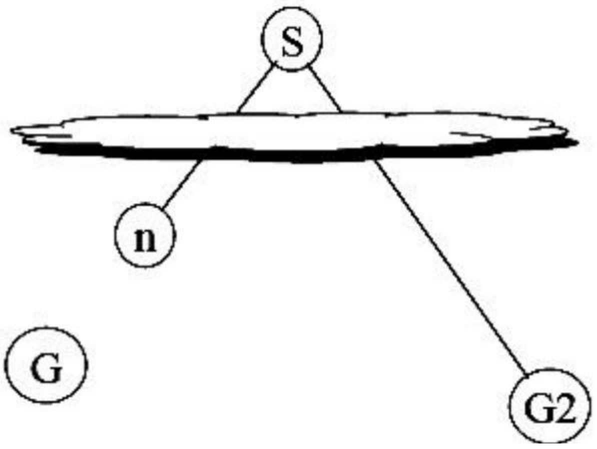
3.2.3 Proof of Admissibility of A*

We will show that A* is admissible if it uses a monotone heuristic.

A monotone heuristic is such that along any path the f-cost never decreases.
But if this property does not hold for a given heuristic function, we can make the f value monotone by making use of the following trick (m is a child of n)
$f(m) = max\ (f(n), g(m) + h(m))$

- Let G be an optimal goal state
- C* is the optimal path cost.
- G2 is a suboptimal goal state: g(G2) > C*

Suppose A* has selected G2 from OPEN for expansion.



Consider a node n on OPEN on an optimal path to G. Thus $C^* \geq f(n)$
Since n is not chosen for expansion over G2, $f(n) \geq f(G2)$
G2 is a goal state. $f(G2) = g(G2)$
Hence $C^* \geq g(G2)$.
This is a contradiction. Thus A* could not have selected G2 for expansion before reaching the goal by an optimal path.

3.2.4 Proof of Completeness of A*

Let G be an optimal goal state.
A* cannot reach a goal state only if there are infinitely many nodes where $f(n) \leq C^*$.
This can only happen if either happens:

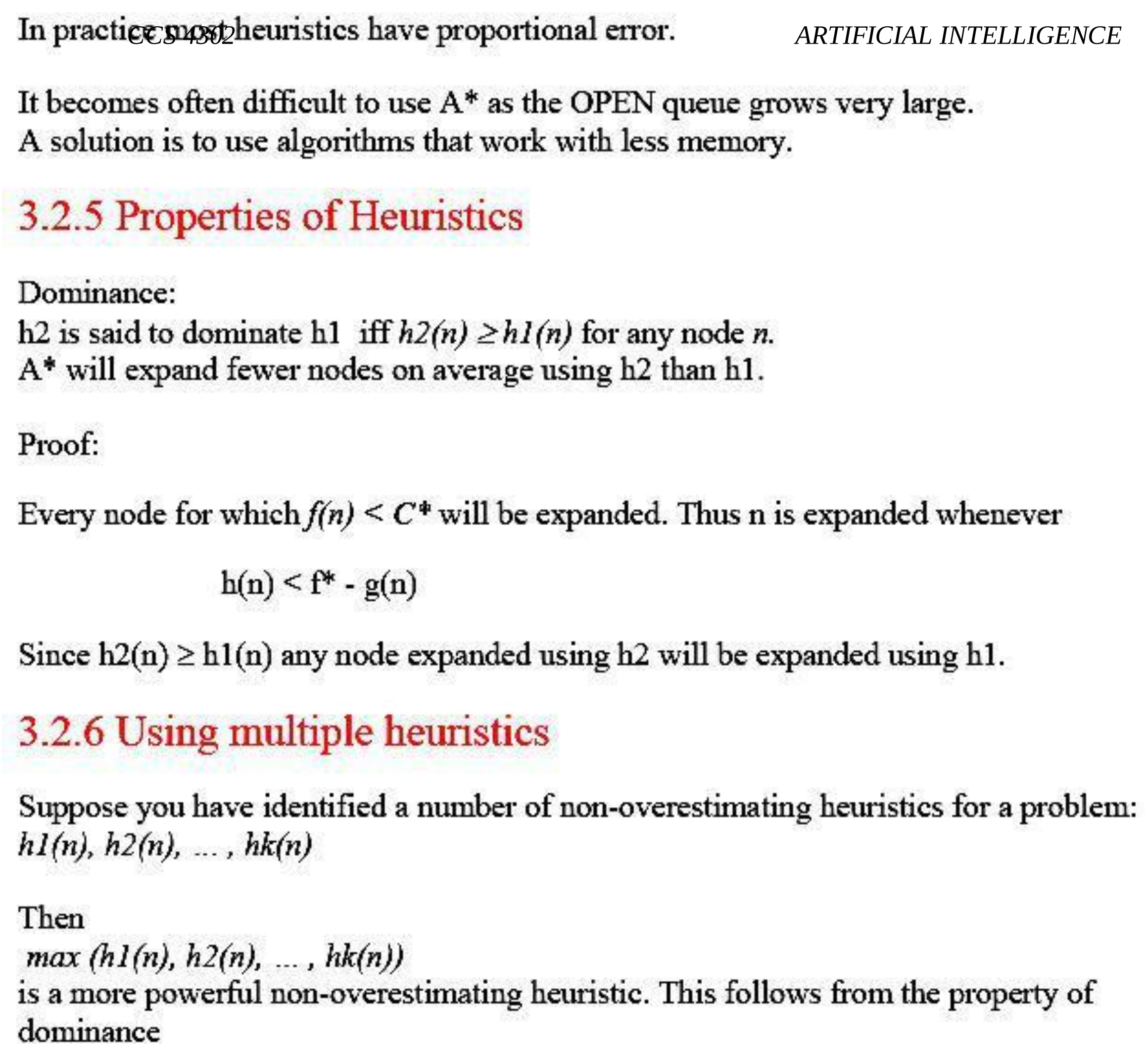
- There is a node with infinite branching factor. The first condition takes care of this.
- There is a path with finite cost but infinitely many nodes. But we assumed that Every arc in the graph has a cost greater than some $\varepsilon > 0$. Thus if there are infinitely many nodes on a path $g(n) > f^*$, the cost of that path will be infinite.

Lemma: A* expands nodes in increasing order of their f values.

A* is thus **complete** and **optimal**, assuming an admissible and consistent heuristic function (or using the pathmax equation to simulate consistency).
A* is also **optimally efficient**, meaning that it expands only the minimal number of nodes needed to ensure optimality and completeness.

3.2.4 Performance Analysis of A*

Model the search space by a uniform b-ary tree with a unique start state s, and a goal state, g at a distance N from s.
The number of nodes expanded by A* is exponential in N unless the heuristic estimate is logarithmically accurate
$|h(n) - h^*(n)| \leq O(\log h^*(n))$



In practice most heuristics have proportional error.

It becomes often difficult to use A* as the OPEN queue grows very large.
A solution is to use algorithms that work with less memory.

3.2.5 Properties of Heuristics

Dominance:
h2 is said to dominate h1 iff $h2(n) \geq h1(n)$ for any node *n*.
A* will expand fewer nodes on average using h2 than h1.

Proof:

Every node for which $f(n) < C^*$ will be expanded. Thus n is expanded whenever

$$h(n) < f^* - g(n)$$

Since $h2(n) \geq h1(n)$ any node expanded using h2 will be expanded using h1.

3.2.6 Using multiple heuristics

Suppose you have identified a number of non-overestimating heuristics for a problem:
$h1(n), h2(n), \ldots, hk(n)$

Then
$max(h1(n), h2(n), \ldots, hk(n))$
is a more powerful non-overestimating heuristic. This follows from the property of dominance

3.3 Beam Search

In computer science, beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set. Beam search is an optimization of best-first search that reduces its memory requirements. Best-first search is a graph search which orders all partial solutions (states) according to some heuristic which attempts to predict how close a partial solution is to a complete solution (goal state). In beam search, only a predetermined number of best partial solutions are kept as candidates.

Beam search uses breadth-first search to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost.http://en.wikipedia.org/wiki/Beam_search - cite_note-1 However, it only stores a predetermined number of states at each level (called the beam width). The greater the beam width, the fewer states are pruned. With an infinite beam width, no states are pruned and beam search is identical to breadth-first search. The beam width bounds the memory required to perform the search. Since a goal state could potentially be pruned, beam search sacrifices completeness (the guarantee that an algorithm will terminate with a solution, if one exists) and optimality (the guarantee that it will find the best solution).

The beam width can either be fixed or variable. One approach that uses a variable beam width starts with the width at a minimum. If no solution is found, the beam is widened and the procedure is repeated.

3.3.1 Name and Uses

The term "beam search" was coined by Raj Reddy, Carnegie Mellon University, 1976.

A beam search is most often used to maintain tractability in large systems with insufficient amount of memory to store the entire search tree. For example, it is used in many machine translation systems. To select the best translation, each part is processed, and many different ways of translating the words appear. The top best translations according to their sentence structures are kept and the rest are discarded. The translator then evaluates the translations according to a given criteria, choosing the translation which best keeps the goals. The first use of a beam search was in the Harpy Speech Recognition System, CMU 1976.

3.3.2 Extensions

Beam search has been made complete by combining it with depth-first search, resulting in Beam Stack Search and Depth-First Beam Search, and limited discrepancy search, resulting in Beam Search Using Limited Discrepancy Backtrackinghttp://en.wikipedia.org/wiki/Beam_search - cite_note-furcy-3 (BULB). The resulting search algorithms are anytime algorithms that find good but likely sub-optimal solutions quickly, like beam search, then backtrack and continue to find improved solutions until convergence to an optimal solution.

3.4 Hill climbing

In computer science, hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

For example, hill climbing can be applied to the travelling salesman problem. It is easy to find an initial solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained.

Hill climbing is good for finding a local optimum (a solution that cannot be improved by considering a neighbouring configuration) but it is not guaranteed to find the best possible solution (the global optimum) out of all possible solutions (the search space). The characteristic that only local optima are guaranteed can be cured by using restarts (repeated local search), or more complex schemes based on iterations, like iterated local search, on memory, like reactive search optimization and tabu search, on memory-less stochastic modifications, like simulated annealing.

The relative simplicity of the algorithm makes it a popular first choice amongst optimizing algorithms. It is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms. Although more advanced algorithms such as simulated annealing or tabu search may give better results, in some situations hill climbing works just as well. Hill climbing can often produce a better result than other algorithms when the amount of time available to perform a search is limited, such as with real-time systems. It is an anytime algorithm: it can return a valid solution even if it's interrupted at any time before it ends.

3.4.1 Mathematical description

Hill climbing attempts to maximize (or minimize) a target function $f(\mathbf{x})$, where $\mathbf{x}$ is a vector of continuous and/or discrete values. At each iteration, hill climbing will adjust a single element in $\mathbf{x}$ and determine whether the change improves the value of $f(\mathbf{x})$. (Note that this differs from gradient descent methods, which adjust all of the values in $\mathbf{x}$ at each iteration according to the gradient of the hill.) With hill climbing, any change that improves $f(\mathbf{x})$ is accepted, and the process continues until no change can be found to improve the value of $f(\mathbf{x})$. $\mathbf{x}$ is then said to be "locally optimal".

In discrete vector spaces, each possible value for $\mathbf{x}$ may be visualized as a vertex in a graph. Hill climbing will follow the graph from vertex to vertex, always locally increasing (or decreasing) the value of $f(\mathbf{x})$, until a local maximum (or local minimum) x_m is reached.

3.4.2 Variants

In simple hill climbing, the first closer node is chosen, whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one.

Stochastic hill climbing does not examine all neighbours before deciding how to move. Rather, it selects a neighbour at random, and decides (based on the amount of improvement in that neighbour) whether to move to that neighbour or to examine another.

Random-restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. It is also known as Shotgun hill climbing. It iteratively does hill-climbing, each time with a random initial condition x_0. The best x_m is kept: if a new run of hill climbing produces a better x_m than the stored state, it replaces the stored state.

Random-restart hill climbing is a surprisingly effective algorithm in many cases. It turns out that it is often better to spend CPU time exploring the space, than carefully optimizing from an initial condition.

4.0 CONCLUSION

Informed search strategies -Also known as "heuristic search," informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)

-Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*

5.0 SUMMARY

In this unit, you learnt that:

Heuristic search is an AI search technique that employs heuristic for its moves.

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.

A greedy algorithm is any algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding the global optimum

Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set

Hill climbing is a mathematical optimization technique which belongs to the family of local search.

6.0 TUTOR-MARKED ASSIGNMENT

1. What is A* Search?
2. Consider the following table

	A	B	C	D	E	F	G	H	I	J	K	L	M
A		36	61										
B				31									
C				32		31						80	
D					52								
E							43						
F										122	112		
G								20					
H									40				
I										45			
J											36		
K													32
L													102
M													0

Using the A* algorithm work out a route from town A to town M. Use the following cost functions.

1. G (n) = the cost of each move as the distance between each town (shown on map).
2. H(n) = The Straight Line Distance between any town and town M. These distances are given in the table below.
   Provide the search tree for your solution and indicate the order in which you expanded the nodes. Finally, state the route you would take and the cost of that route.

7.0 REFERENCES/FURTHER READING

Lowerre, B. (1976). "The Harpy Speech Recognition System", Ph.D. thesis, Carnegie Mellon University.

Russell, S. J. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. (2nd ed.). Upper Saddle River, New Jersey: Prentice Hall, pp. 111–114, ISBN 0-13-790395-2.

Zhou, R. & Hansen, E. (2005). "Beam-Stack Search: Integrating Backtracking with Beam Search". http://www.aaai.org/Library/ICAPS/2005/icaps05-010.php