**Concept of Heuristic Search Technique**

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed

**Heuristic Search Techniques in AI**

Other names for these are Informed Search, Heuristic Search, and Heuristic Control Strategy. These are effective if applied correctly to the right types of tasks and usually demand domain-specific information.

Before move on to describe certain techniques, let’s first take a look at the ones we generally observe. Below, we name a few.

* Best-First Search
* A\* Search
* Bidirectional Search
* Tabu Search
* Beam Search
* Simulated Annealing
* Hill Climbing
* Constraint Satisfaction Problems

## Constraint Satisfaction Problems (CSP)

Let’s talk of a magic square. This is a sequence of numbers- usually integers- arranged in a square grid. The numbers in each row, each column, and each diagonal all add up to a constant which we call the *Magic Constant*. Let’s implement this with Python.

**A\* Search Algorithm and Its Basic Concepts**

A\* algorithm works based on heuristic methods, and this helps achieve optimality. A\* is a different form of the best-first algorithm.

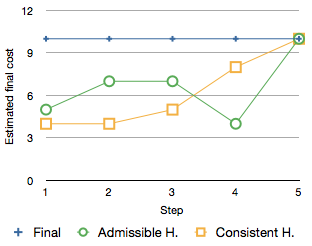
When A\* enters into a problem, firstly, it calculates the cost to travel to the neighboring nodes and chooses the node with the lowest cost. If The f(n) denotes the cost, A\* chooses the node with the lowest f(n) value. Here ‘n’ denotes the neighboring nodes. The calculation of the value can be done as shown below:

f(n)=g(n)+h(n)f(n)=g(n)+h(n)

g(n) = shows the shortest path’s value from the starting node to node n  
h(n) = The heuristic approximation of the value of the node

The heuristic value has an important role in the efficiency of the A\* algorithm. To find the best solution, you might have to use different heuristic functions according to the type of the problem. However, the creation of these functions is a difficult task, and this is the basic problem we face in AI

**What is a Heuristic Function?**



Essentially, a heuristic function helps algorithms to make the best decision faster and more efficiently. This ranking is based on the best available information and helps the algorithm decide the best possible branch to follow. Admissibility and consistency are the two fundamental properties of a heuristic function.

**Admissibility:**

A heuristic function is admissible if it can effectively estimate the real distance between a node ‘n’ and the end node. It never overestimates; if it ever does, it will be denoted by ‘d’, which also denotes the accuracy of the solution.

**Consistency:**

A heuristic function is consistent if the estimate of a given heuristic function turns out to be equal to or less than the distance between the goal (n) and a neighbor and the cost calculated to reach that neighbor.

A\* is indeed a very powerful algorithm used to increase the performance of artificial intelligence. It is one of the most popular search algorithms in AI. The sky is the limit when it comes to the potential of this algorithm. However, the efficiency of an A\* algorithm highly depends on the quality of its heuristic function.

**Implementation with Python**

In this section, we are going to find out how the A\* search algorithm can be used to find the most cost-effective path in a graph. Consider the following graph below.

Implementation with Python


The numbers written on edges represent the distance between the nodes, while the numbers written on nodes represent the heuristic values. Let us find the most cost-effective path to reach from start state A to final state G using the A\* Algorithm.

Let’s start with node A. Since A is a starting node, therefore, the value of g(x) for A is zero, and from the graph, we get the heuristic value of A is 11, therefore

g(x) + h(x) = f(x)

0+ 11 =11

Thus for A, we can write

A=11

Now from A, we can go to point B or point E, so we compute f(x) for each of them

A → B = 2 + 6 = 8

A → E = 3 + 6 = 9

Since the cost for A → B is less, we move forward with this path and compute the f(x) for the children nodes of B

Since there is no path between C and G, the heuristic cost is set to infinity or a very high value

A → B → C = (2 + 1) + 99= 102

A → B → G = (2 + 9) + 0 = 11

Here the path A → B → G has the least cost but it is still more than the cost of A → E, thus we explore this path further

A → E → D = (3 + 6) + 1 = 10

Comparing the cost of A → E → D with all the paths we got so far and as this cost is least of all we move forward with this path. And compute the f(x) for the children of D.

A → E → D → G = (3 + 6 + 1) +0 =10

Now comparing all the paths that lead us to the goal, we conclude that A → E → D → G is the most cost-effective path to get from A to G.

