Heuristic Search & Hill Climbing

Heuristic Search

The Traveling Salesman Problem (TSP), where a salesman makes a complete tour of the cities on his route, visiting each city exactly once, **while traveling the shortest possible distance, is an example of a problem which has a combinatorial explosion.** As such, it cannot be solved using *breadth-first or depth-first* search for problems of any realistic size. Unfortunately, there are many problems which have this form and which are essentially intractable (they can't be solved). In these cases, finding ***the best possible answer is not computationally feasible,*** and so we have to settle for a good answer. In this section we discuss several heuristic search methods which attempt to provide a practical means for approaching these kinds of search problems.

What is the Heuristic Search ?

Heuristic search methods are characterized by this sense that **we have limited time and space in which to find an answer to complex problems and so we are willing to accept a good solution.** As such, we apply ***heuristics or rules of thumb*** as we are searching the tree to try to determine the likelihood (احتمال) that following one path or another is more likely to lead to a solution. Note this is in **stark contrast** to brute-force methods which chug along merrily regardless of whether a solution is anywhere in sight.

Heuristic search methods use objective functions called ***heuristic functions*** to try to gauge the value of a particular node in the search tree and to estimate the value of following down any of the paths from the node.

Generate and Test Algorithm

The generate and test algorithm is the most basic heuristic search function. The steps are:

1. Generate a possible solution, either a new state or a path through the problem space.
2. Test to see if the new state or path is a solution by comparing it to a set of goal states.
3. If a solution has been found, return ***success***; else return to step 1.

Example:-

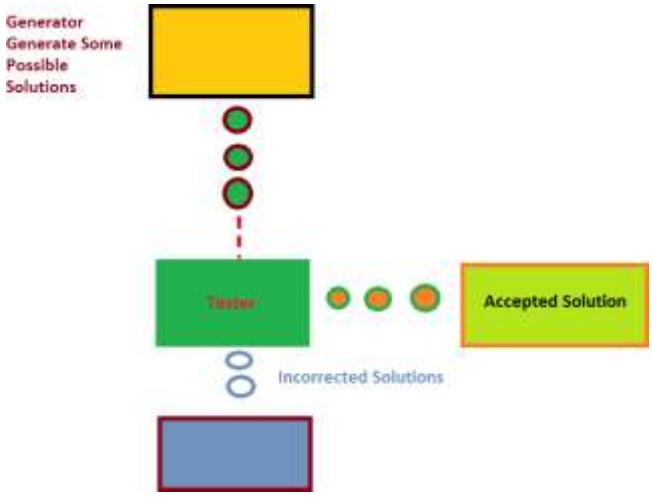


Figure (1): Example of Generate and Test Algorithm

Question for Students: what is the difference between this algorithm and depth first search algorithm?

Disadvantage of this algorithm

- It may take an extremely long time. For small problems, generate and test can be an effective algorithm, but for large problems, the undirected search strategy leads to lengthy run times and is impractical.
- The major weakness of generate and test is that *we get no feedback on which direction to search.* We can greatly improve this algorithm by providing feedback through the use of heuristic functions.

Hill climbing Algorithm

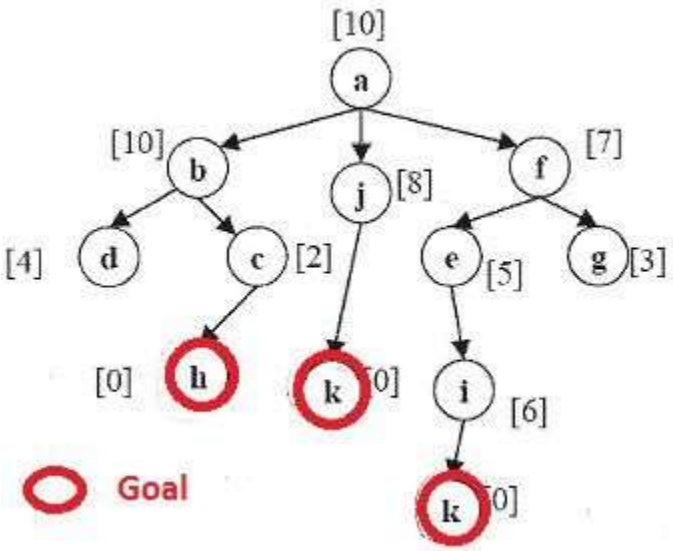
It is an improved *generate-and-test algorithm*, where feedback from the tests are used to help direct the generation (and evaluation) of new candidate states. When a node state is evaluated by the goal test function, a measure or estimate of the distance to the goal state is also computed.

Pseudo-Code Algorithm

```
function HILL-CLIMBING(problem) returns a solution state
```

```
inputs: problem, a problem
static: current, a node
        next, a node
current <— MAKE-NODE(INITIAL-STATE[problem])
loop do
        next— a highest-valued successor of current
        if VALUE[next] < VALUE[current] then return current
        current *—next
end
```

Example



In this example, (a) is initial State and (h) and (k) is final states and it says that the numbers near the states are the heuristic values.

What are disadvantages of this algorithm?

- One problem with hill climbing search in general is that the algorithm can get caught in **local minima or maxima**. Because we are always going in the direction of least cost, we can follow a path up to a locally good solution, while missing the globally excellent solution available just a few nodes away. Once at the top of the locally best solution, moving to any other node would lead to a node with lower goodness.

- Another possibility is that a **plateau** or flat spot exists in the problem space. Once the search algorithm gets up to this area all moves would have the same goodness and so progress would be halted.
- It isn't complete and can't guarantee to find the global maxima. The benefit, of course, is that it requires a fraction of the resources. In practice and applied to the right problems, it's a very effective solution.
- Hill Climbing don't generally backtrack, because it doesn't keeping track of state (it's local search) and you would be moving away from a maxima

In the example above :

- Hill climbing on the tree, we start a➔ f ➔ g and then what ?? finish(without result). This is called a local minima

Solutions

- A common way to avoid getting stuck in local maxima with Hill Climbing is to use random restarts. In the example in above if G is a local maxima, the algorithm would stop there **and then pick another random node to restart from**. So if J or C were picked (or possibly A, B, or D) you would find the global maxima in H or K. it will find the global maxima or something close; depending on **time/resource limitations and the problem space.**
- Another solution to avoid getting trapped in suboptimal states, variations on the hill climbing strategy have been proposed. One is to inject noise into the evaluation function, with the initial noise level high and slowly decreasing over time. This technique, called simulated annealing, allows the search algorithm to go in directions which are not "best" but allow more complete exploration of the search space.