AI module 7
Problems with search methods and solutions

Introduction

We have seen quite a few search methods and also have seen some issues. We will discuss some other issues that demand to change or modify the search methods to adapt to the situation. We will also see how a decision of picking up a typical heuristic function help solving it in better way. Some search methods also require to backtrack to provide explanation to the user, some of them need to solve some part of the problem before solving other parts as that part is very critical to the overall solution. At the end we will see an extension to the hill climbing process which helps achieving a solution independent of the starting point known as iterative hill climbing.

The issues that we discuss here apply to many search algorithms if not the most.

Local and global heuristic functions

We have already seen that the local maxima or minima is a serious problem. That problem also has some relation to the heuristic function chosen. If the heuristic function chosen is local in the sense that it does not really generates the value using overall situation but local situation, it is possible for it to stuck in local maxima. Unlike that, if the function chosen takes into consideration the overall situation and not confined to local parameters it has more chances to reach to a final state.

To understand the difference between a global and a local function, let us take an example from a domain called blocks world. The blocks world problems include a plain surface and some blocks of same size. The problem is about rearranging blocks from a given position to some other position. For example we might have two blocks resting on the surface called A and B, we may like to have A on top of B or vice versa. The other constraint is that only one block may be moved at a time, may be atop the surface or some other block. A block which does not have any other block on top of it is called a free block. One cannot move a block directly if it is not free. He has to move all the blocks resting on top of that block first, one by one, and only then it can move that block. The blocks world problem sounds simple enough for KG kids to play around. Blocks world problems and solutions proved quite useful to teach robots to stack and unstack things to get desired result. For example if a robot is asked to pick up a box which is lying somewhere in the room, may be under some other box and place it at some other place, may be on top of some other box, the blocks world algorithmsare helpful.

Let us take an example of a blocks world problem shown in figure 7.1. We have blocks from A to F arranged as shown in the initial stage. We want them to be arranged in the form shown in the final state. As we are only able to move one block at a time we would like to find out the solution such that we should reach to final state ASAP. We will use some heuristic function for that.

The first part is to represent the initial and final state. We will use a form of predicate logic which we have used earlier to represent both the initial and final states as well as the state space.

Accordingly the initial state is defined as

On(-,A) // the – indicates the surface
On(-,E) // On (X,Y) means Y on top of X
On (-,F)^On(F,G)^On (G,D)^On(D,C)^On(C,B)

And the final state is defined as
On(-,G)^On(G,F)^On(F,E)^On(E,D)^On(D,C)^On(C,B)^On(B,A)

What will be the state space? We must state that all blocks are either has a block on top of them (can be any other block except the – which represents the surface) or are free (nothing on top of them).

∀□, □ X∈ □A..G, – □Y ∈□A..G□On(X,Y) V Free(X)

What will be the rules?
On(X,Y) and Free(Y) -> Free(X) and On(_,Y) //putting the block sitting on top of other block on table
Free(X) ^ Free (Y) -> On(X,Y) and ~Free(X) //when two blocks have nothing on top of them,
// putting one on top of other

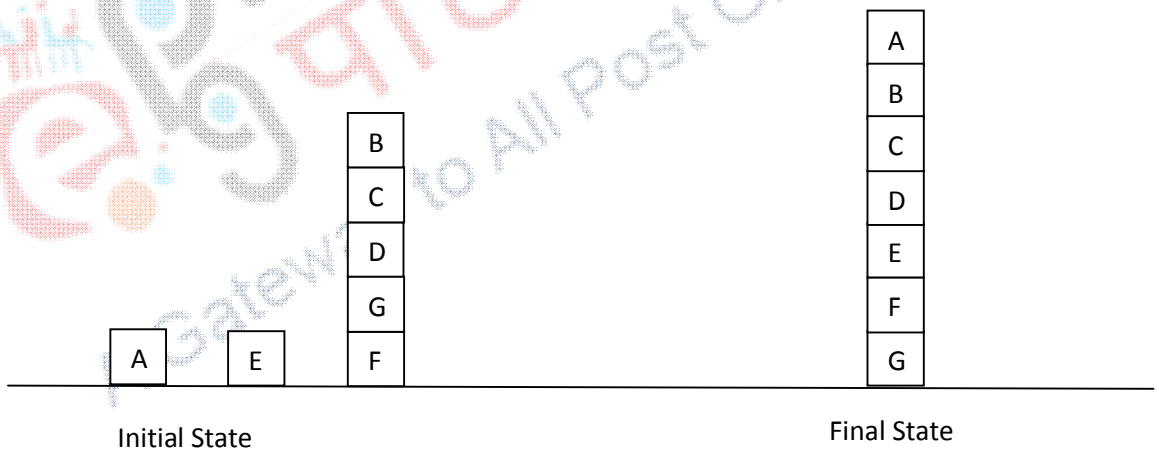

Figure 6.1 The blocks world problem

Using these two simple rules we can march forward from initial state. We can simplify the search using a heuristic function. Rich and knight has a similar problem with two different heuristic functions. We will

be using those functions here. The first heuristic function adds one point to a block resting on the block it should be resting on and -1 otherwise. Looking at which the initial state heuristic function values are calculated as follows.

-1 for A, -1 for E, -1 for F and -1 for G and -1 for D as all of them are resting on something they should not be resting on. Similarly Both B and C are resting on the block they should be resting on, so both of them fetch 1 each. The total comes out to be -3. The Final state every block is resting on what it should so the total comes out to be 7.

Now let us look at all possible moves from the initial state. There are total five possible moves. Each of the resultant state's heuristic values are also calculated. Figure 7.2 shows all possible resulting states and the heuristic values of each resulting states. One can see that only one of the five possible states is better with heuristic value -1. Let us pick up that state. What are possible moves from it? You can see that all the states which result from it are not better than this state and thus hill climbing terminates at this point[1]. Figure 7.3 shows all those states. One can see that none is better than previous.

[1]You may consider the state-3 to have equivalent value so consider it as a successor. Even if we do that, the only move possible is to place E back on the surface which again has the same heuristic value -1. Each of the moves from now on has lesser heuristic value and so even this refinement does not lead us much further.

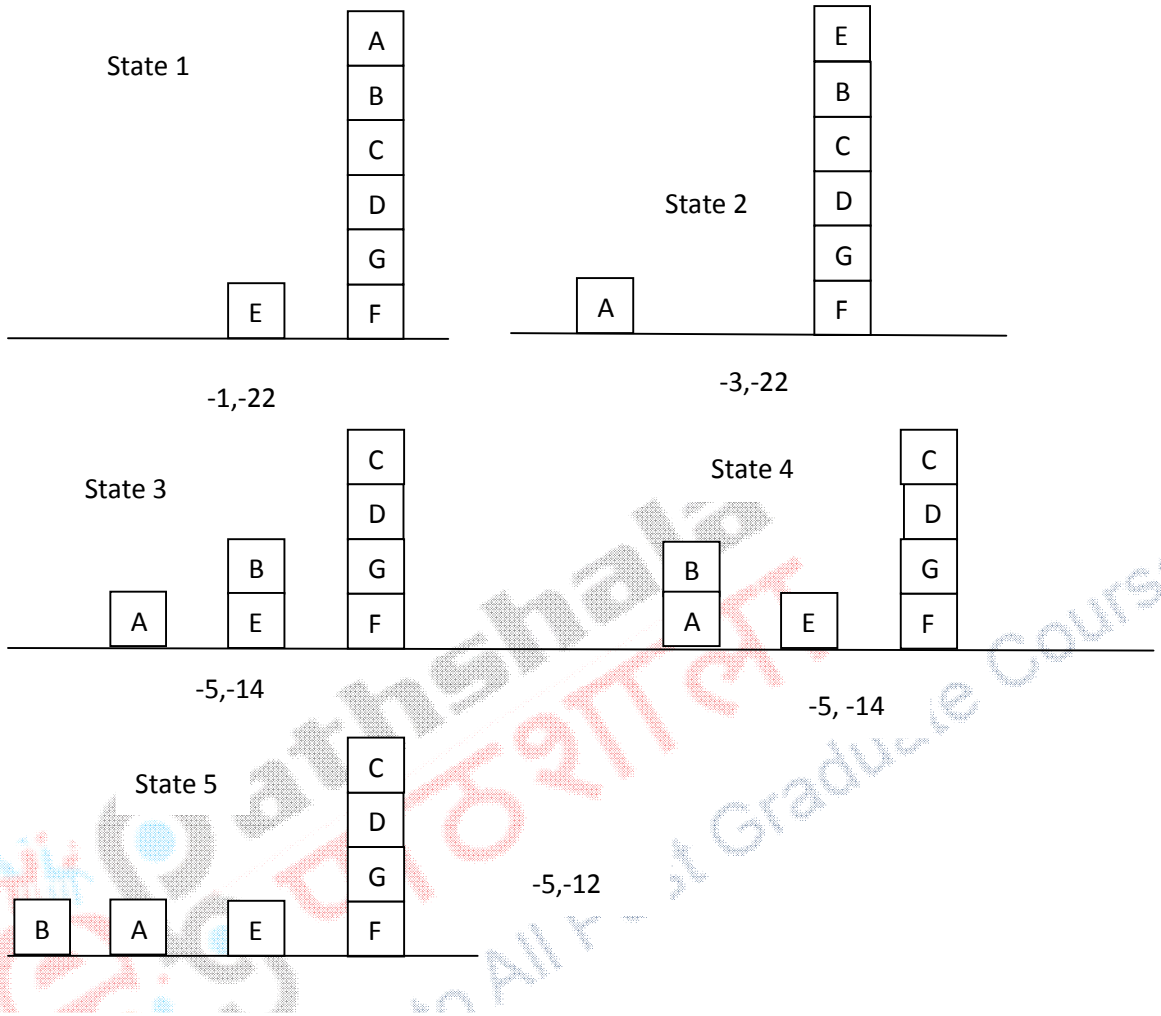

Figure 7.2 The heuristic function values, the local heuristic function values are written first and global heuristic function values are written next.

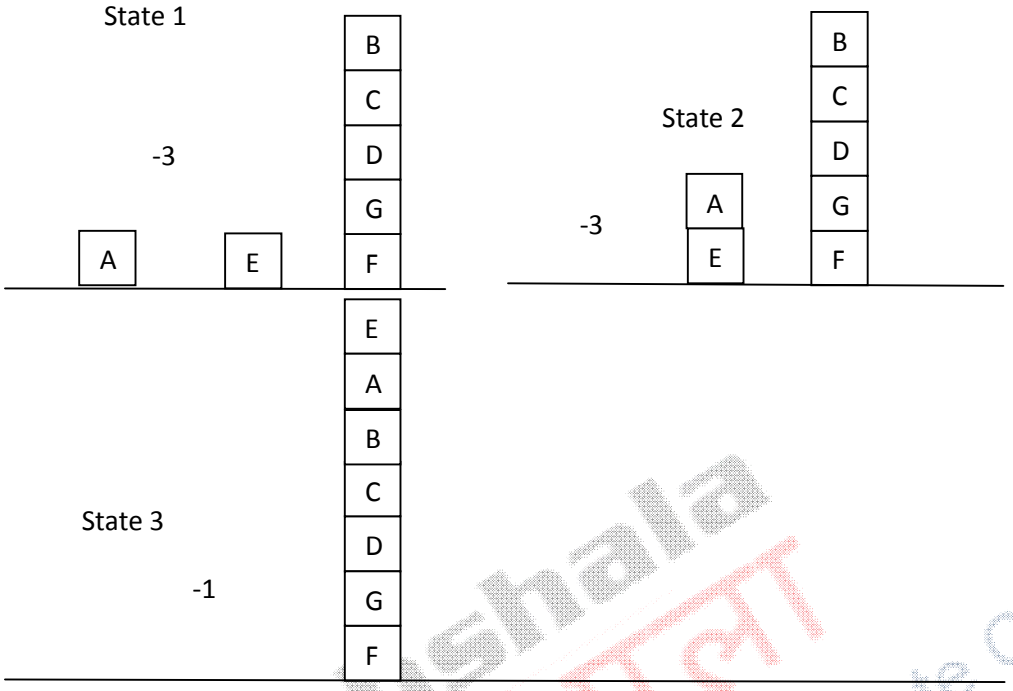

Figure 7.3 three states resulting from the best move

You can remember our last module discussion about adding randomness to the movement and escaping local minima. Anyway, we would like to change the heuristic function to see what we can do. Let us take another heuristic function to see if the situation can be improved.

The second heuristic function does not consider the block on which the block under consideration is resting on but entire stack of blocks. Thus we will award one point for each of the block of the entire correct stack. If the stack contains even one block out of place, we will give a negative point for each ofthe blocks of the entire stack. Using the new heuristic function, the initial state's value calculated as

-1 for A, -1 for E, -1 for F, -2 for G, -3 for D, -4 for C, -5 for B, making it = -17

For the final state, 1 for G, 2 for F, 3 for E, 4 for D, 5 for C, 6 for B and 7 for A makes it =28

Now let us look at all possible moves from the initial state and calculate heuristic values of each of them. The state 1

-1 for E, -1 for F, -2 for G, -3 for D, -4 for C, -5 for B, -6 for A making it -22

State 2 will have identical value -22

State 3 -1 for A, -1 for E, -2 for B, -1 for F, -2 for G, -3 for D, -4 for C making it -14

State 4 will have identical value -14

While state 5 -1 for A, -1 for B, -1 for E , -1 for F, -2 for G, -3 for D, -4 for C making the total = -12

Figure 7.2 also depicts the heuristic value for the new function immediately after the value calculated for the first heuristic function.

Now the next best state is state 5 and not state 1. You can clearly see that out of all other possible moves, placing C on surface would be the best and so on till all blocks are placed on the surface. From there on placing right blocks on top of the structure not only increases the value of heuristic function but also would lead towards solution. We will never encounter the problem of local maxima ever if we use this function. The function will always be increasing for correct moves.

Why it is so different searching using different heuristic functions? If you look carefully you can get the difference. The first heuristic function is a local heuristic function. It only looks if the block is resting on the block it should and award point for the same irrespective of the global position of that block, considering other blocks either resting on top of them or below them. This heuristic function value does not change if the block it is resting on, contains correct stack of blocks or not. One can easily understand that if a block is resting on a correct block but have otherwise incorrect structure, it must be broken down and restructured. As this heuristic function does not consider that, it falls for local maxima.

Unlike that, the second heuristic function is a global function. The second function only allows complete support stack to be present. Even if one of the block is not correctly placed in the stack, it disallows the entire stack. Not only that, it also preferred such structures to be broken down by allowing more negative points for the blocks resting on top of incorrect structures.

In fact the example clearly indicates that global heuristic functions are to be chosen for avoiding local minima or maxima. Unfortunately not all domains and heuristic functions are simple enough for a designer to decide about the same. Many complex domains including chess contains some good heuristic functions but it is really hard for anybody to decide if the function is really global or not.

Local maxima is not the only problem. Let us look at a few other problems.

Plateau and ridge

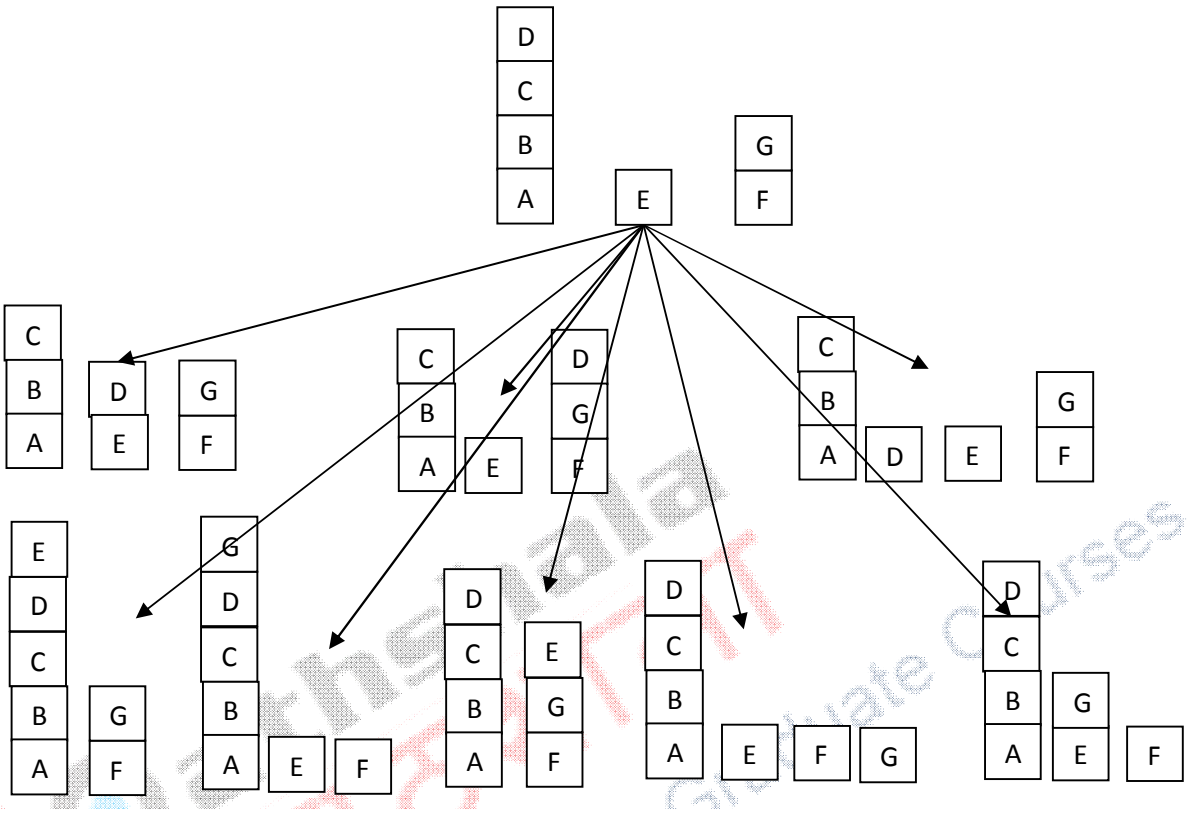

Figure 7.4 the plateau problem

We will begin our discussion with two important problems usually discussed with local maxima, plateau and ridge.

Look at figure 7.4. It takes a typical blocks world problem with the same final state as the problem shown in figure 7.1. We will also take the heuristic function which is local. If the heuristic value of all states is calculated using the local heuristic function, the value comes out to be the same as the parent, -7. Now we have no better move, no worse move, all moves are equal. If you try proceeding further, the same situation might continue to remain for a while. The search algorithm, which is designed to look at better moves stuck up here. Even when it is ready to accept worse moves, it cannot choose between moves as all of them look the same. If one draws a graph, all these points look at the same height and the region looks like a plateau, hence the name.

The plateau nullifies the power of heuristic function and thus the problem solver has to try either randomly in any one direction or use some other criteria to decide the next point to proceed further.

Ridge is a special case of local maxima where there is only one narrow path to the solution. Ridge can be countered only by trying multiple moves together.

Frame problem

One more interesting problem some search algorithm fails to counter is the problem of side effects. When we design a state space we have to be careful about what we are going to store in the description of the state space. Our trivial problems had not shown us but many complex problems have difficulty in deciding what we can keep in a state space.

Let us an example of a robot. If the robot is moving along the room, what should it store in a state space? His own location? Obviously. Location of other objects of the room? May be only the objects along the path to avoid them. Should it also remember the celling is above and floor is beneath? Not that obvious.

Once the state space does not have everything, we might have strange side effects of operations. Consider a case where there are three boxes in the room and nothing else. Let us name these boxes as A,B and C. The robot is asked to move B to a new location and it did so. Should it also check the locations of A and C? It should not as it has not touched them. However, if C is placed on top of B, then? How do the robot know? Also, should it also check if the celling is above and floor is still beneath?

In fact there are many variables in robot's domain, which of them to be included in the state space representation and which of them are to be tracked to represent current state is not an easy question. When the side effects of applying rules affect the current state of the problem in a way that we get the incorrect representation of the current state (we assume C to be at the same location as it was but it is not so actually), the problem is known as a frame problem. This problem is more related to the state space representation but it can hamper search algorithm if it fails to assess the correct space the process in.

One can also understand that storing everything in state space also is not a good solution as the robot might need to remember the complete path if ever required to backtrack. If the nodes along the path takes up lot of memory, both the time and space requirements go beyond manageable level.

Problem decomposability and dependency

Another problem is to decide how to decompose the problem and solve it. Humans usually decompose big problems to solve, for example if a conference is to be organized, it is usually decomposed into paper management, venue management, food management, VIP and invitee management etc. each of the parts may also be decomposed further, for example paper management may be decomposed into managing publication and responding to queries about the paper presentation part, managing to talk to reviewers, sending papers and get reviews from them, organize papers into categories and so on.

Such a process has two inherent issues, first, how to divide the problem into different components and second to determine the interplay between those components. For example if the problem is about finding the best route between two given points, it is quite possible that a typical link between those two points is most important and must be managed before others. Humans almost always do that. For example if we plan to go to Mumbai from Ahmedabad, we first book the train ticket from Ahmedabad Railway station to Mumbai Railway station even though the actual search process should begin from our

house to Ahmedabad railway station. We do not start our search by choosing the best method and time to go to railway station first and then find the train best suited after that. We decide about the train first and accordingly decide the best solution. It is quite possible that the train starts at 10:00 O clock in the morning where the traffic to the station is heaviest. We do not decide to have least traffic solution from home to railway station say at 5 O'clock in the morning and wait till the train arrives. We will go for suboptimal solution which land us at railway station just in time for the train. This solution (home to railway station) depends on the solution to the critical part of the problem (the railway problem). Such dependency is extremely important for a good search process to learn and utilize in finding a solution.

Similarly if we are going abroad, for example some place in Singapore, we manage flight from Mumbai to Singapore first and find other parts of the solution which suits that solution. Interestingly that might include finding train to reach to Mumbai.

In other words we may not always start searching from the start state.Depending on the importance of some critical part of the problem, we may solve that critical part first, come back to connect the starting point of that critical part with the initial state and may be end part of the critical part to the final state. We may even need to do it recursively. For example going to Singapore, we may solve the flight part first, may be train from Ahmedabad to Mumbai next, Singapore airport to the actual destination after that and so on. That means our search process moves to and fro during the search process unlike what we have seen so far.

In a case where such dependency does not exists and components can be independently solved, the search algorithm might run in parallel and solve independent components. The basic form of the algorithm that we have used so far need to be modified accordingly.

Independent or Assisted Search

One more problem that we have not yet seen about the search process is the interaction with user. If the search process is independent, it comes out with the answer on its own and does not require human intervention. The search processes that we discussed earlier were of that type. Sometimes though, it is better if some form of assistance is provided.

For example an expert system might suggest the doctor possible diagnosis and medication. The doctor might provide his own opinion for the expert system to change or modify its decision. This is quite like the assistants learning under the doctor. For example the medical expert system might suggest some medicine X and the doctor reminds the system that the patient has high BP so instead of X, Y is better.

Another example is about a robot used in thigh bone surgery. Robots can do a better surgery as their hands are more stable than human counterparts. They can drill a finer hole and insert the rod more precisely. They, unlike humans, cannot exactly pin point the place to operate. Humans help these robots by moving the robotic hands to the exact position manually. Once the robotic arm reaches the exact spot, they can continue from there.

Some systems which proves mathematics theorem also requires assistance in the beginning. Once they know the exact spot to start proving the theorem, they can complete the same.

In fact interacting with other humans and experts to learn or confirm is a very good idea for many expert systems. A search process must have some method to incorporate the human interaction part to decide the direction of search.

Search for Explanation

Sometimes finding the solution is not enough. The search process is also required to backtrack and respond to the user to explain how it solved the problem. For example if we provide the theorem to prove and the programs responds back "Yippee.... Proved!), none of us accepts that. We expect the program to provide us the proof with justification. The program must provide how it searched the solution using some reasonable step by step explanation.

Another domain where explanation is must is the expert systems dealing with humans. For example a medical diagnostic system informs the patient that he has cancer, no patient in the world would believe it immediately. The system must explain why it reached to that conclusion (for example it might say that your biopsy report is positive and it is confirmed by two reputed laboratories).

Another important point which sometimes drive the search algorithms to operate in a typical even if inefficient way is that the explanation must be in human understandable form. For example which one is a better explanation for you?

"The value of XOIT_YTHI_0033 is 2345, BNJY_UKNC_4756 is above 100 so you have malaria"

"It is already reported that, you have fever, you felt cold for no obvious reason, you had weakness and you have law level of red blood cell count. All these indicate that you have malaria"

Obviously the second one. The first method may be more efficient but it reasons unlike human to solve the problem and its explanation, thus is beyond understanding and thus not acceptable. Thus a search process, if required to reason, it must do so analogous to human way of reasoning.

After discussing some important problems with search process, we will embark on our last search method, iterative hill climbing, which shows how conventional hill climbing process can be extended to solve the problem in a better way.

Iterative Hill Climbing

Iterative hill climbing solves the problem of local maxima using multiple random starting points. One problem with hill climbing is that the end node depends on the start node. We always try to find a route to higher or lower (depending on whether we are climbing or descending) position from the start node and stop when we cannot proceed further. Thus the search process will find a point in the higher direction from the beginning. If the heuristic function chosen is truly global we will surely get the global maxima (the correct solution) but otherwise we may held up at some local maxima.

One simple solution to this problem is to decide multiple random starting points to start and get multiple answers. The iterative hill climbing method determines the best answer from the lot. The Iterative Hill Climbing can be described as follows. The hill climbing process is carried out in a loop for

each starting point picked up from the state space, the solution is entered in the array called solution array and continued till the user defined criteria for termination is reached.

When the user defined search criteria is over, the best solution from the array is picked up and returned.

1. Choose one random point from the state space, if the search process is over evaluate the solutions from the solution array and returns the best solution.
2. Apply hill climbing and get a solution. Use heuristic function provided.
3. Add that solution to a solution array
4. Go to 1

The iterative hill climbing process is a better solution to a state space which is convoluted, i.e. made up of many local maxima. It has more chances to pick up correct starting point to reach global maxima.

One interesting property of the Iterative Hill Climbing is that it is possible for it to get different answers every time it runs. It is because the answer depends on the choice of starting points which is random.

Summary

In this chapter we have tried to highlight some typical problems faced by searching methods used for real world problems. We have seen that the search process efficiency depends on the heuristic function and a heuristic function with global perspective is better. When the search process cannot maintain complete information about the state, it is sometimes not possible to detect side effects of actions taken which is known as frame problem. Humans solve problems by decomposing them and keeping them to manageable size. Correct decomposing of a problem demands learning about dependency and interplay between different components. The search process might require human assistance in proceeding further. Sometimes the search process needs to explain its finding. It must be able to backtrack as well as reason like human for that. Finally we have looked at iterative hill climbing method which is basically a hill climbing method called in loop with different random starting points and getting the best solution from them.