Artificial Intelligence

Local Search

Paper Code: CS-401

Anupam Pattanayak[1]

Assistant Professor,

Department of Computer Science,

Raja N. L. Khan Women's College (Autonomous),

Midnapore, West Bengal

May 5, 2020

[1]anupam.pk@gmail.com

Contents

1 Local Search **1**
1.1 Background 1
1.2 Introduction 2
1.3 Local Search & Optimization 3
1.4 Local Search: Example of 8-Queen Problem 4
1.5 Hill Climbing Algorithm 5
1.6 Tabu Search 6
1.7 Local Beam Search 8

1

Local Search

So far, we have seen *informed search* and *uninformed search* techniques. In this study material, we will discuss local search algorithms.

This study material has been prepared by consulting multiple books and video lectures. These references include the book by Russel[1], book by Coppin[2], and NPTEL Course lectures on *An Introduction to Artificial Intelligence* by Prof. Mausam[3] in SWAYAM platform of MHRD, Govt. of India.

1.1 Background

Let us briefly revisit whatever have been covered so far. Many computational problems can be mapped as AI search problems. Search is a fundamental approach in solving a new problem. In AI search techniques what we have seen so far is that, there are huge number of states and actions. There is a *start state* and one or more *goal state(s)*. Our target was to find a path from start state to goal state. We have seen search techniques which are *systematic*. The uninformed search techniques such as DFS, BFS, and IDS *algorithms search blindly at all directions*. Informed search techniques such as A^*, IDA^*, Depth First B&B are guided search. Here, next node is chosen for expansion on the basis of an *evaluation function* that estimates cost to reach a goal state. Based on our present knowledge, to solve a new problem, we model the given problem as a search problem. Then, we apply a search algorithm either from uninformed search techniques or from informed search techniques to explore the search space systematically to reach from start state to a goal state and obtain the path from start state to goal state as solution

[1] Artificial Intelligence A Modern Approach by Russel and Norvig, PHI.
[2] Artificial Intelligence Illuminated by Ben Coppin, Jones and Bartlett Publishers
[3] https://swayam.gov.in/nd1_noc20_cs42/preview

of the given problem. Local search technique is altogether a different way to search.

1.2 Introduction

Till now, a solution of a given problem was a path from start state to a goal state. For some problems, the path is not relevant. A solution is only relevant for some class of problems such as *8-queen problem* that you have probably come across in algorithm paper. In local search, we search in the solution space. Here, not the path, but state itself is a solution.

In 8-queen problem, we need to place 8 queens in an 8×8 board such that no two queens attack each other: that is, there should not be two queens in the same row or same column or same diagonal. The figure 1.1 shows a solution to the 8-queen problem.

						Q	
				Q			
	Q						
			Q				
					Q		
							Q
		Q					
Q							

Figure 1.1: A Solution to 8-Queen Problem

In the previously seen approach, where we were searching in state space, we would have formulated the 8-queen problem as shown below:

```
Our start state is empty.
Each action is to add one queen.
Goal test is that all queens placed and no two queens attack
    each other.
```

This approach is like finding shortest path to a goal. All goals are at depth length n. Note that, for 8-queen problem solution, it does not matter whether queen 1 is placed in first column or second column or any other

column. We are only interested in obtaining the final state. This type of problems are suitable for modelling as local search. In practice, solution is often found faster using local search than informed search.

1.3 Local Search & Optimization

There is a close relation between local search and optimization. First, let us revisit the concept of optimization problem that you have studied in under graduate level in *Operation Research* topics such as LPP. Here, we will look at the optimization problems in terms of search. Suppose, we are interested to find the best path to the goal in search space. When mapped as optimization problem, we can re-cast the problem as find the path which optimizes some *objective function*. In optimization problem, we are given with an objective function and a set of constraints. For local searches, we add a constraint as given below:

optimization function $\leq C$ in minimization problem, or

optimization function $\geq C$ in maximization problem.

Here, we work with the version of optimization problem where variables are discrete or integer. So, two conditions are to be met for applying local search:

I. The given problem is an optimization problem,

II. The sloution is a state.

In local search, we only keep track of present state. That is, current state is the single state we bother about, and we ignore keeping track of paths. That is why local search is very memory efficient. It is capable of finding reasonable solution in very large or even infinte state space. While we are in a current state, we can move only to a neighboring state - that is, we search in *local neighborhood*. So, given states and neighborhoods of the states, and given an objective function that evaluates a state, local search algorithms find a state that has optimum (maximum or minimum) value of objective function.

In local search, every state is a solution - bad or good. A solution is *good* where higher number of constraints are statisifed, and *bad* where lower number of constraints are satisfied.

1.4 Local Search: Example of 8-Queen Problem

Let us again visit the 8-queen problem from the local search view. The figure 1.2 shows a solution to the 8-queen problem.

						Q	
				Q			
	Q						
			Q				
					Q		
							Q
		Q					
Q							

Figure 1.2: A Solution to 8-Queen Problem

We have to formulate this 8-queen problem as an optimization problem. Here, a state in search space will be a solution. Can you guess, what will be the objective function? It is the *number of attacking pair of queens*. We have 8 queens. If we take any two queens at a time, there are total ${}^8C_2 = 28$ possibility of choosing a pair of queens. So, what will be minimum number of attacking pair of queens? This is 0 - which is required for our solution. So, our objective function h : number of queens attacking each other. Now, what will be a possible state space in 8-queen problem? It will be all possible positions of queens in 8×8 board. Suppose So, we have 8^8 states in the state space. Now, we need to define *neighborhood function* or *successor function*. Here, a neighborhood state could be a state where positions of 7 queens will be same as the current state, only one queen will be moved. A good successor function keeps good balance between immediate neighborhood of the current state and the legth of path to the solution. Defining a good successor function is a skill that has to be learnt by AI people. Then we need to define the heuristic function h ( ), or objective function that we want to optimize. This we have already discussed: number of queens that are attacking each other. We want to minimize this heuristic function.

1.5 Hill Climbing Algorithm

As the name *Hill Climbing* suggests, it's concept is associated with climbing to the hill top. Following figure 1.3 shows a sample hill. When monuntaineers

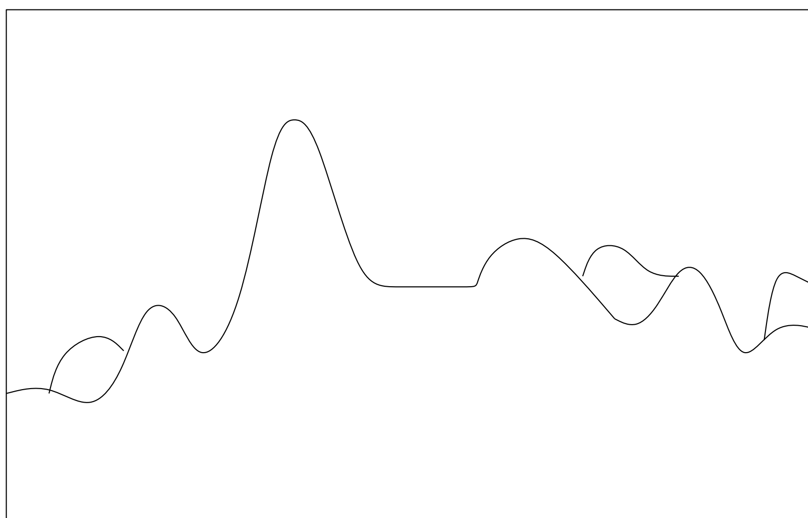

Figure 1.3: A Hill

go for expeditions such as climbing the Mount Everest, how does one realize if she has rached at the summit or peak of the Mount Everest? They keep a GPS tracker or altitude meter that guides them in this regard. However, there can be many other peaks with little bit smaller height in the surroundings of summit which can mislead monuntaineers to believe that they have reached at the peak in the absense of such GPS tracker.

Such things can also happen when we search for solution in the search space, where there are many local optimums and one global maximum. While doing the local search using hill climbing we may reach a local maximum, and we believe we have reached at the global maximum.

In hill climbin algorithm, since an objective function is associated with every state, so given a current state, all it's neighborhoods are evaluated with the objective function. If a neighborhood state is found with value higher than the current state and it's the maximum value amongst the neighborhood, then this neighbor becomes the new current state. This is repeated. If none of the neighbor has higher value than the current state then terminating condition has been reached and the hill climbing algorithm returns the local maximum and then terminates.

So, we can say that hill climbing algo. is the greedy local search. Current

state may have many alternative for next state. Choose the next state which seems the best. Following algorithm 1 gives the maximum version of Hill Climbing algorithm.

Algorithm 1 Hill Climbing Algorithm: Max. Verion

```
function HillClimbing(problem)
  Input: problem
  Output: returns a state that is local maximum

  local variables: current_node, neighbor_node

  1.  current_node ← MakeNode(InitialState(problem))
  2.  while(true)
  3.    neighbor_node ← SuccessorHighest(current)    /* best neighbor */
  4.    if(Value(neighbor_node) ≤ Value(current_node))
  5.      return State(current_node)
  6.    current_node ← neighbor_node
  7.  end of while loop
      end HillClimbing
```

In case multiple neighbors have the best value, then hill climbing algorithm randomly chooses any succesor among those neighbors. It does not look beyond the immediate neighbors. Hill climbing often gets stuck in local maxima (or minima). Concept of local maximum and global maximum is illustrated in the following figure 1.4 shows a sample hill.

In the graph whenever there is any *plateau*, or *ridge*, or *foothill* like those are found in mountains or hills, the local search algorithm like hill climbing will get stuck. However, if the initial state was near the global maximum, then hill climbing can return the global maximum.

1.6 Tabu Search

In hill climbing if the search gets suck in a local optimum, there is no way to come out of that. One solution is to take steps back from local optimum and

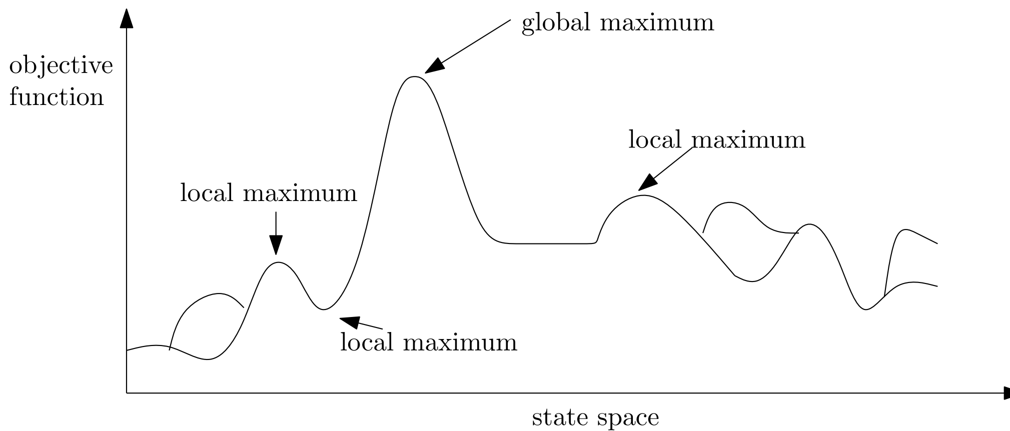

Figure 1.4: A Hill

go down to reach at the bottom. Once the bottom is reached then the search is resumed afresh with the hope that better solution will be visited. This is continued. However, a limit on possible number of sideway moves to be placed to prevent infinte looping. This is the concept of *tabu search*. In tabu search we keep track of last node and this node is not repeated. Following is the main features of the tabu search:

I. maintains a fixed length queue known as *tabu list*

II. add most recent current state to the queue and drop the oldest

III. prevents returning quickly to the same state

IV. never make it to the state that is currently tabu'ed

Following algorithm 2 gives the maximum version of Hill Climbing algorithm.

If the size of tabu list increases, tabu search asymptotically becomes *non-redundant*. That is, it would not visit the same state twice. In practice, tabu list queue size of 100 or such improves the performance of tabu search over hill climbing in many problems. If the tabu list size is extremely large or ∞ then tabu search essentially becomes a systematic search. In practice, a time bound is also used to terminate a local search.

Algorithm 2 Tabu Search Algorithm

function *TabuSearch*(problem)
Input: problem
Output: returns a state that is possibly global maximum

local variables: current_node, best_node

1. current_node ← Random(InitialState(problem))
2. best_node ← current_node
3. Tabu_List ← current_node
4. while(!Empty(Tabu_List))
5. current_node ← Non_Tabu_SuccessorHighest(current)
6. Tabu_List ← current_node
7. if(Value(current_node) $\leq$ Value(best_node))
8. best_node ← current_node
9. end of while loop
10. return State(current_node)

end *TabuSearch*

1.7 Local Beam Search

In local search techniques that we have seen up until now, the search maintains only one current state at any point of time. Can we think of an alternative approach where more than one current state is maintained? Suppose, the search starts with k number of states. Look at the neighbors of these k states. Then look at the k best states in next round. This is the idea of *beam search*. Beam search keeps k successors out of many successors.

The idea behind beam search is that maintaining only one state in memory is an extreme reaction to the problem of excessive memory requirement of uninformed or informed search. So, keep track of k states instead of just one state in the search space. Next, find all the successors of these k states. Accept the best successor as solution after this process is iterated over several

times until a time bound is reached.

Here, search starts in parallel. But soon , the search converges to one or two hills. So, the beam search loses diversity after quite a few rounds.