Intro to Local Search

There are many problems in this world which are computationally intractable if we wish to get the deterministic answer to it. These types are much more common than we think in the real world, and we cannot afford to solve these problems slowly, and we must find a workaround to this issue.

Fortunately, for a many of these computationally intractable problems, finding a *good enough* answer suffices as well, instead of finding the perfect answer. Therefore, we can come up with algorithms which might not always give the *perfect answer*, but can give a *good answer,* with much better time complexity.

Now, we'll attempt to define Local Search more formally, and give a generalized working of the algorithm.

Definition of Local Search

Define $X =$ set of candidate solutions to the problem

Neighborhood

Since we have a set of solutions X, we need to define the concept of neighborhoods, and define for every $x \in X$, what are its neighbourhoods $y \in X$.

As we saw in the Maximum Cut problem, we had defined a neighbor as

$$x, y \text{ are neighbors} \equiv \text{ their configuration differs by position of 1 vertex}$$

Therefore, a **neighbor** is essentially a solution that coincides with the current solution in the most ways possible while being distinct.

Defining a Generalized Local Search algorithm

1. We first pick an arbitrary initial solution, x.
2. We iterate through all the neighborhoods of the current solution, and if we find a solution y such that y is superior to the current solution, we assign y to x.
3. We terminate the search once we arrive at a solution which has no superior neighbors, and return the solution x as our answer. This solution is known as a local optimum.

Picking the initial solution for the Local Search algorithm

There are mainly two methods of starting off with a local search algorithm

1. **Starting off with local search -** This strategy is used when we have no clue on how to start off with a solution, and are fully dependent on the

local search algorithm to give some solution. There is a good probability of picking a starting point which leads to giving a very poor answer, which is why we run several independent trials with different starting points to reduce the possibility of such an issue occurring.

2. **Using a greedy strategy or heuristic -** There are often many times where we have a heuristic algorithm that could help us start off with a decent initial solution. An example of this is the nearest neighbor heuristic in the Travelling Salesman Problem. After running the heuristic, we apply local search to improve upon the solution we get.

Choosing a neighbor among several superior neighbors

There will be many times in the execution of a local search algorithm that we have multiple superior neighbors to switch to. There are mainly 3 options on choosing what to do.

1. **Choosing the neighbor at random -** One may feel that if they pick a determined neighbor, their algorithm could be susceptible to input cases which are specially designed to break the algorithm designed. By introducing randomness, the probability of this is reduced.
2. **Biggest Improvement -** ****Another popular strategy is choosing to be greedy by going to the neighbor which gives the biggest improvement.
3. **Complex Heuristic -** One may feel that the biggest improvement principle may not be ideal, and that some other greedy intuitive strategy could give a better answer. Usually, complex heuristics to decide the next neighbor are application specific, based on the use case, and the nature of the input and data you have to work with.

Running Time

Say that we have a problem, with a finite optimum answer. We can be sure that the local search algorithm will terminate eventually, if we assume that every time we make a switch, we make an improvement in the score.

However, we cannot say that local search algorithms terminate quickly, although it might in some cases. Local search algorithms are "anytime" algorithms, hence in case you don't want to wait for too long, or you want an answer in a fixed amount of time, you can instead put a bound on the number of local search iterations that can be made, or a bound on the running time, after which you can just ask for the current solution that is held, even though it might not be the local optimum.