Local Search Algorithm Analysis

Zach Robinson

May 20, 2016

1 Hill Climbing

The hill climbing algorithm that I wrote is very simple. The algorithm starts at the point on the provided function with (x, y) = (0, 0) and attempts to find a more optimal (minimal) solution by evaluating the function with an increment or decrement of either x or y. The algorithm checks, in order, whether or not increasing x, decreasing x, increasing y, or decreasing y by the provided step size results in a more optimal solution. The algorithm continues to reach for a minimal solution in this manner until it reaches a point at which none of the mentioned increments or decrements result in a more optimal solution, or in other words, when it reaches a local minimum.

The algorithm is well suited to quickly find the true minumum in a simple space without multiple valleys or plateaus, where the algorithm would not get stuck and return a sub-optimal solution.

2 Hill Climbing with Random Restarts

The algorithm that I wrote for hill climbing with random restarts builds off of the basic hill climbing algorithm that I already described. It takes an additional parameter that defines the number of 'restarts' to execute. For the number of restarts required, the algorithm generates random values (x, y) within the given range and evaluates the provided function at this point. It then executes in the manner of the basic hill climbing algorithm, searching for a more optimal solution. For each execution, once the search for an optimal solution plateaus, the point (x, y, z), is stored. Once all restarts have been executed, the algorithm loops through the solutions found and returns the most optimal.

The algorithm, like the basic hill climbing, is also well suited to find the true minimum in a simple space without multiple valleys or plateaus, however in this case would take N times longer to find a solution than the basic hill climbing algorithm, where N is the number of restarts executed. This algorithm is better suited, however, to handle spaces with multiple valleys and plateaus because it has an N times greater chance of starting its climb in an area in which it can find the true minimum location.

3 Simulated Annealing

The algorithm that I wrote for simulated annealing works by starting at a random position in the space (calculated with a random (x, y) pair) and proceeds to calculate additional random neighboring positions, moving at every opportunity to a more optimal solution and moving with some probability to a less optimal position. The neighboring solutions are chosen by moving x and y by some random amount, bounded by the step size parameter. As the algorithm progresses, the probability that a sub-optimal move will be accepted steadily decreases. This is achieved by 'cooling' the temperature parameter down to near zero over the course of the execution. Once cooled to a temperature of near zero, the algorithm terminates and returns the position at which it finished.

This algorithm, unlike the hill climbing algorithm, is not guaranteed to find the most optimal position in any space, even a space with only one minimum to fall into. It is, however, better suited than the basic hill climb to find a reasonably good solution in a complex space with many valleys. There is more similarity between this algorithm and the algorithm for hill climbing with restarts, in that both rely on an element of randomization. The hill climbing algorithm with restarts is an optimization over the basic hill climb in that it has a much higher chance of starting a climb in a position at which it can climb to the global optimal position. The simulated annealing strategy institutes an element of randomization that, instead of relying on randomly choosing to start in the 'correct' spot, hopes to find that correct spot by overcoming local minima with some probability.

4 Analysis

From multiple tests, there is no noticable speed advantage between the algorithms with the exception that the basic Hill Climbing is more likely to finish first as it is extremely prone to becoming stuck at a local minimum quickly. Both the Hill Climbing with Random Restarts and Simulated Annealing strategies complete execution quickly and I can discern no advantage between the two.

The search strategy that consistently finds the more optimal position is the Hill Climb with Random Restarts strategy. It is often able to achieve a more optimal position than the Simulated Annealing search, while sometimes both algorithms produce the same result. It is rare that the Simulated Annealing search beats the Random Restart algorithm. The basic Hill Climb fails to be relevant as it is designed to always start at position (0,0) which happens to be a local minimum for the function being tested.

Part of the observed optimality of the Random Restart algorithm as I've written it is in the application of the different function parameters over those of the Simulated Annealing algorithm. I have found more optimal parameters for the Random Restart algorithm, while I have yet to find such optimal parameters for the Simulated Annealing algorithm. Additionally contributing to its optimality, if the Hill Climb with Random Restarts search finds itself anywhere on the most optimal path, it will with 100

I chose to additionally combine the Simulated Annealing and Hill Climbing strategies,

executing the Simulated Annealing algorithm, and then executing a Hill Climb starting from the position determined by the Simulated Annealing search. I expected this strategy to be the most optimal, as it combines the advantages of both. This expectation was refuted, however, by the testing results as the Random Restart algorithm continued to outperform this variant as well.