Tailer Nguyen

1792311

Class of 22

COSC 4368

Lab Report

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p=30 & z=0.5 | Run1 | | | | Run2 | | | |
| #sol se#sol searched  S s | solution | f(sol) | #sol  searched | | solution | F f(sol) |
| (2.9, (404,504) | 100  23 | -0.26137034 -0.27677607 | -377.44 | 100  143 | | -0.3668371, 0.35060921 | -377.4209 |
| (-2.5( (0,0.23) | 39 | 0.42171928 -0.13452246 | -46.67 | 100  32 | | 0.45416454, -0.18387131 | -46.67 |
| ((4.2 (-200,+300) | 54 | -0.113652 -1.58540308 | -543.991286 | 51 | | 0.50557794 1.41295889 | -543.991286 |
| (+412, -99.9) | 11  112 | 0.50557794 1.41295889 | -556.624884 | 87 | | -0.36534181 1.03813613 | -556.624884 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P=30 & z = 3 | Run1 | | | Run2 | | | |
| #sol se#sol searched  S s | solution | f(sol) | #sol  searched | solution | F f(sol) |
| (2.9, (404,504) | 100  20 | 0.56955542, 2.21137907 | -377.44 | 100  15 | -0.44168775 -1.66159915 | 461.  -377.44 |
| (-2.5( (0,0.23) | 5 | 2.2506692 -2.69269162 | -46.67 | 100  61 | 1.27801459 -2.52241388 | -  -46.6743 |
| ((4.2 (-200,+300) | 55 | -0.113652 -1.58540308 | -543.991286 | 72 | [-243.0813067194887, 274.5730274634014] | -543.991286 |
| (+412, -99.9) | 96 | [3[ 62.19484550624776, -106.9183858637526] | -556.624884 | 34 | [362.02[0.12227242 1.11380723] | -556.624884 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P=250 & z = 3 | Run1 | | | Run2 | | | |
| #sol se#sol searched  S s | solution | f(sol) | #sol  searched | solution | F f(sol) |
| (2.9, (404,504) | 100  132 | 0.39108642, -1.48494474 | -377.44 | 100  2 | 0.56955542 2.21137907 | -377.444 |
| (-2.5( (0,0.23) | 60 | 2.51327654 -0.09964994 | -46.67 | 100  59 | 2.2506692 -2.69269162 | -46.67 |
| ((4.2 (-200,+300) | 55 | 2.2506692 -2.69269162 | -543.991286 | 23 | 2.51327654 -0.09964994 | -543.991286 |
| (+412, -99.9) | 80 | 0.12227242 1.11380723 | -556.624884 | 55 | [362.02[0.02511897 0.32667256] | -556.624884 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P=250 & z = 0.5 | Run1 | | | Run2 | | | |
| #sol se#sol searched  S s | solution | f(sol) | #sol  searched | solution | F f(sol) |
| (2.9, (404,504) | 100  20 | -0.3668371 0.35060921 | 377.44 | 100  123 | 0.42171928 -0.13452246 | 461.  -377.44 |
| (-2.5( (0,0.23) | 34 | 0.45416454 -0.18387131 | -46.67 | 100  55 | 0.42171928 -0.13452246 | -46.6743 |
| ((4.2 (-200,+300) | 109 | 0.42171928 -0.13452246 | --543.991286 | 91 | 0.45416454 -0.18387131 | -543.991286 |
| (+412, -99.9) | 101 | [2.81117, 0.321] | -556.624884 | 10 | [362.02[-1.69313386 2.50691213] | -556.624884 |

Problem Set 1 Task 1 Report:

For the 33rd run, I used the values p = 1000, z = -5, and sp = [-500,-500]. I ended up getting a value of -724.836770 and a solution of [-1.69313386 2.50691213]). This is a much lower minima than our other test runs, and we can say that the random seeding does affect the values of the solution, but it barely changes the value of the f(solution).

The algorithm that I implemented was a random restart hill climbing function. By using this random restart solution, we locate many local optima within the problem and this may prove closer to be next to the other minima of the function. We evaluated how accurate our function was by also using a candidate evaluation variable which checked the solution being better than the solution evaluation variable, and if so, we replaced it and returned it back. This algorithm is efficient because it only uses a nested for loop, but it is only checking if it is within bounds. Other than checking in bounds, the solution runs pretty fast.

In my solutions, I noticed that increasing Z (the step-size increments) incurs an increase in the solution size by a small amount (except for run p = 250 and z = 0.5 to p = 250 and z = 0.3 for some reason). This is because hill-climbing uses a greedy approach, and the program will not move to a lesser state and terminate itself.

For changing the sp, the starting point, I noticed that changing the starting point drastically changes the solution points. This is because that, because the sin function constantly changes from negative to positive and from positive to negative, catching a different starting point results in a local minima and maxima at different starting points.

For the p value, this is the number of neighbors randomly generated during our solution. The higher the number is, the more accurate our solution will be. However, our algorithm runs longer because our for loop runs p times.

Due to the random restart hill climbing solution, the RHC proved consistent and accurate in calculating the f(solution).