NatAlgReport

Name: Joseph Crawley User-ID: tpkl28

Two-letter code for your chosen NatAlgReal algorithm: WO

In my research on parameter optimization, I experimented with different parameters by running the algorithm 50 times and seeing the percentage of iterations which reached the global minimum of 0 at (0,0,0) as well as keeping track of the average run time of each configuration. I discovered that the number of whales (N) and the number of cycles (num_cyc) have the most significant impact on performance. I found that N is most efficient when it is greater than 75 and less than 200. With N less than this range, the exploration of minimums is limited, resulting in a non-global minimum. However, values greater than 200 resulted in timing issues, as the global minimum had already been found. To determine the best value for num_cyc, I incrementally increased the value until a 100% success rate of finding the global minimum was achieved. My findings showed that a num_cyc value of 1100 resulted in a 100% success rate and had an average runtime of 2.1 seconds when using N=200. A num_cyc value of 600 achieved a global minimum with an 86% chance, but for optimal results, a num_cyc value of 1100 was used. Additionally, I found that values between 0.1 and 0.01 for the "b" parameter, which affects the bubble net attacking spiral, yielded the best results in realvalued optimization. However, this didn't greatly change timings and effectiveness. A b value of 0.01 was used in my implementation. It's worth noting that due to the precision of the Python float type, the function compute_f returns 0.0 on inputs very close to (0,0,0) which means that my implementation doesn't always converge exactly to 0,0,0 but values very close. The time complexity of my whale optimization implementation is O(N * num_cyc) this is because each whale gets ‘processed’ num_cyc times yielding us this time complexity

(continued over)

Name: Joseph Crawley User-ID:tpkl28

Two-letter code for your chosen NatAlgDiscrete algorithm: WO

When adapting the Whale Optimization Algorithm (WOA) for use in Graph Coloring, the key process that needed to be discretized was the movement of the whales towards one another. To achieve this, I implemented a system in which a whale would adopt the color of a vertex from the whale it was moving towards. The "A" vector, which controls the degree of movement towards a target whale, remained largely unchanged in my discrete version of the WOA. The normal of the A vector is used to decide whether a whale moves towards the most fit whale or a random whale, and each element of the A vector determines how many elements will be inherited from the target whale.

In order to discretize the bubble net attacking process, I employed a similar approach as used for the searching process. I utilized the logarithmic spiral function to work with a randomly generated value, creating a probability of X(t) to inherit from the most fit whale. On analyzing the logarithmic spiral function (e^bl * cos(2pil)), I found that it yielded values roughly between 1 and -1. Therefore, to create a probability from it, I used a random uniform variable between 1 and -1 to introduce "randomness" in the process.

I found that using a similar methodology to my implementation of WOA with real values, I was able to successfully select the parameters for the discrete setting as well. I observed that the 'b' parameter had a significant impact on performance, with values above 0.1 resulting in poor performance. Through experimentation, I determined that the optimal value for 'b' was 0.01 for all graph values of 'a', 'b', and 'c'. To ensure efficient execution and minimize conflicts, I incrementally increased the values for 'N' and 'num_cyc' while keeping the total runtime under 60 seconds. For example, for graph file A, the optimal parameters were (25, 800, 0.01) with 14 conflicts, for graph file B (15, 175, 0.01) with 143 conflicts, and for graph file C (5, 110, 0.01) with 691 conflicts.