# NatAlgReport

Name: Damian Elkins User-name: frcg69

Two-letter code for your chosen NatAlgReal algorithm: AB

Once I coded the algorithm I created a copy where I added simple capabilities to write all parameters to an excel file while testing. My aim was to maximize the performance within 10 seconds to get a good idea of how the program functions before making changes. I found that increasing lambda simultaneously increased rate of convergence and lowered the time taken to a much more consistent level. I also found that my minimum value with high cycles was consistently being found around 100-700 cycles in so I could lower the num_cyc to decrease running time. This allowed me to increase number of sources while maintaining similar times. This introduced more variance in the time. I found a good balance was to assign M and N to 170 with 800 cycles and a lambda of 5. After 100 runs with these parameters the mean fitness was 4.7 with a standard deviation of 1.38 and the mean time was 7.14. My peak run had the fitness 1.564726716067474 with a time of 7.8 at [4.168428197638733, -3.003706626993946, 3.973114181332673, 5.056398925800774] I found that my poorest performing runs found their minimum value much later indicating to me that the biggest factor was poor initialization of the food sources. I also found that my poorest performers were getting stuck on the borders, my current solution to the bees flying out of the search space being assigning the value that went out to the relevant max or min value. I felt there was room to have a smarter initialization process. To deal with the border problem I had the idea to "bounce" the bee away from the borders. For the initialization I decided on Latin Hypercube Sampling which creates a more even distribution. My original idea for the "bouncing" was to generate an n sized vector and send the bee off in that direction however, this could require multiple bounces and complicate things, so I settled on just bouncing back in the movement dimension with some random bounciness scaling. With my enhancements across 100 runs the average value was 3.62 with a standard deviation of 0.99 and took 7.43 second on average. My peak run got 0.603256549 in 7.1s at [1.514743169991506, -2.4946402634763585, -3.3123991045544035, -2.520636954357836]

Two-letter code for your chosen NatAlgDiscrete algorithm: FF

My initial discretization started by generating some random cliques. This was done by choosing some random starting points and then progressively adding neighbours of the starting point if they did not break the clique. From this I calculated all the distances to be the number of vertices where two cliques differ. So, cliques [0,1,0,0,1,1] and [1,0,0,0,1,1] would have a distance of 2. Fitness is then calculated as the size of the cliques generated. I now had to decide between maintaining clique structure and allowing non cliques to be created to better explore the search space. I had initially decided to maintain clique structure as this would simplify the fitness as I wouldn't have to constantly check for clique structure. In practice I found that the results were entirely based on the initial population search and the time saved was negligable. I would generate my best at the start and never improve. For example, when testing on graph A, I would generate 85 as the max clique about 50% of the time even though other attempts would generate maximum clique of size 86. This is due to the movement being so restrictive and the starting generation being prone to finding local maxima. I allowed for non-cliques to be generated by altering my fitness to penalize for the number of edges that were missing from the clique. I also limited the size of the cliques generated at the start to 5% of the total number of vertices avoid finding local maxima in generation. I settled on a movement method of removing the attraction # of points with the fewest edges in the clique and then taking the union of the two cliques to be the new clique assignment. Attractiveness is calculated in the same was as the base algorithm My random movement removes the number of vertices in the clique * alpha * a random value vertices with the lowest degree and then greedily generates a new clique by selecting a random vertex and adding all neighbours to the clique. For my random movement, I gradually scaled the alpha so that more random exploration occurred in later cycles. I added this because I found that exploration was plateauing quite quickly as fireflies converged on a value.
Once implemented I began experimenting with parameters. I initially had 100 fireflies and 100 cycles but found that there

(continued over)

Graphs A and B would settle on a value in the first generation. From this position I aimed to decrease N as the time increased exponentially with N. On graph A I found that I could consistently find a clique of size 86 within 10 cycles with 50 fireflies. I ran runs with 50 fireflies for 12 cycles 10 times and found a clique of size 86 every time in 4.2 seconds. I continued to push the number of fireflies down and 25 was the first I found that converged consistently in a faster time. I was able to get 86 from 5 cycles roughly 90% of the time in 0.5 seconds. I pushed it even further and was able to find 86 with 10 fireflies in 5 cycles about 20% of the time. This brought my best time down to 0.1 seconds. I decided not to use this as my general parameters, however, as the accuracy was much too low. I settled on 25 in 5 cycles. My experimentation with graph B was very surprising. I was able to consistently find 346 with 5 fireflies in 5 cycles roughly 95% of the time. I decided to alter the size of the cliques generated from the initial population to be 1% of the total number of vertices in the graph to lower the initialization time. When trying to find the best parameters for C I noticed that it tended to move to large, failed cliques quite quickly after creation. To account for this, I added some additional randomness to the movement. I added a probability that whenever moving, there is a 50% chance that you do not merge or add any new vertices, just remove vertices from the clique. This meant that occasionally when moving, a firefly would have nothing in its clique. In these cases, I generated a new one with the same process as the initial population was generated. This also ensured that there were occasionally complete cliques added for the fireflies to search around if they got too far away from a solution. With these changes I was able to get 58 consistently on graph C with 30 fireflies in 5 cycles in 14.3 seconds. This change also did not have any impact on the other 2 graphs.

My final discretization is as follows:
Initialize a population of size N by greedily creating cliques of a max size specified as a percent of total number of vertices. Calculate distances as the number of places where two cliques differ. Calculate fitnesses as the size of clique-number of missing edges between vertices. Loop for num_cyc comparing all fireflies. For each comparison move each firefly towards the one with higher fitness or randomly move if equal. Attractiveness is calculated the same. Movement involves removing the value of attraction of vertices with the least connections and then merging the two graphs 50% of the time. The other 50% of the time the clique remains shrunken or is regenerated if it is empty. Random movement removes the number of vertices in the clique * alpha * a random value it then has a 50% chance to greedily add new vertices to the clique by looking at the neighbours of a random point in the clique, otherwise it is left shrunken. All the fitnesses get recalculated and the loop continues until the total number of cycles is reached.