# CS 451 - Computational Intelligence

---

# Assignment 2 - Optimization and Swarm Intelligence

---

Instructor - Dr. Saleha Raza
Authors - Abdullah Siddique (ms03586) and Kuldeep Dileep (kl04008)

# Contents

# 1 Coloring Graphs using ACO

## 1.1 Problem Formulation

Ant Colony Optimization is a technique inspired from the social behaviour of ants that can be used to solve NP-hard problems like the graph coloring problem.

To solve this problem, we went through several research papers, the most significant of which are quoted in the references section, and took inspiration from pre-existing algorithms to solve our problem.

There are various techniques to approach the problem of graph traversal using ants e.g. the greedy approach, DSATUR method, RLF method etc. We chose to go with the DSATUR method because of low time complexity ($O(n^2)$), easy implementation, and reasonably accurate results.

DSATUR stands for degree of saturation. In this algorithm, when an ant is deciding which node to travel to next, one major factor is the degree of saturation of each node that is not yet colored. The degree of saturation for a node is maximum when it has the maximum amount of colored neighbors. There are different ways to employ a DSATUR method. In the one we have utilized, the node with the highest DSATUR is prioritized i.e. assigned a higher probability of selection.

We will now state the flow of our algorithm as it is implemented in order to obtain the minimum number of colors required to color the graph.

We have two classes in our code that are governing the optimization process. One is labelled ACO and is responsible for managing ant colonies and their optimization in each iteration. The other is labelled Ant and represents each ant traversing a graph.

First of all we load the graph from the text file. We form a list of edges by reading the file and then pass it to our ACO class. The ACO class initializes an instance with specified parameter values of $\alpha$, $\beta$, $\gamma$ etc. Two global matrices are also created of size (vertices × vertices) which store the indices of the adjacent nodes and the pheromone values respectively. Then we run a loop for the amount of iterations. In each iteration, our ACO creates a new colony of ants consisting of specified number of Ant class instances. Each ant of a new colony is initialized at a random node. Then we update the status of each ant in that colony. Updating in this context signifies that the ant proceeds to traverse the entire graph.

As mentioned earlier, an ant starts its journey from a random node. Lists are maintained of the unvisited and visited nodes (tabu list). The probabilities defining which node should be visited next are given by the following formula:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_k (\tau_{ik})^{\alpha}(\eta_{ik})^{\beta}} \tag{1.1}$$

Here $\tau$ is called the trail intensity and $\eta$ is called the visibility. $\alpha$, $Q$ and $\beta$ are factors that influence their weightages. In our context, $\tau$ is taken to be the pheromone concentration between two nodes i.e. the current node and the node that is being considered for visitation. $\eta$ is taken to be the degree of saturation of the node that is being considered for visitation. Once the probabilities are generated that define the importance of each unvisited node, a random number between 0 to 1 is generated which determines which node will the ant visit. $\gamma$ gives the pheromone retention rate and $(1-\gamma)$ gives us the pheromone evaporation rate. Similarly $Q$ is used as a weight to the pheromone values of the best ant in each colony when updating the global pheromone matrix as stored in $\tau$.

After the next node is selected, the ant will traverse to it. It will then check the colors of all the neighbors of this new node and store these in a tabu list. Accordingly, a color will be assigned to this new node. This way, an ant iterates through the entire graph.

After all the ants in each colony have traversed the entire graph, we apply the evaporation factor to reduce the intensity of pheromones stored in the pheromone matrix. We also update the global pheromone matrix at the end of each iteration. This is done in a way somewhat different from the normal technique we studied in class. We are using a variant of the algorithm that we found in multiple papers to be an improved version of classical ACO, in which only the best ant at the end of an iteration is used to update the pheromone matrix. This means that we are using the best solution i.e. the one which has produced the minimum number of colors so far. If there are multiple such ants, they all may be considered. This strategy generally allows for better convergence towards the optimal solution.

For plotting purposes, we also maintain a record of values to be plotted for the BFS and AFS graphs. For BFS, we simply take the value of the best ant of the best colony so far. If the value of the best ant of the current colony is somehow worse than that of the previous colony, it is ignored and our BFS stays a monotonically decreasing curve. For AFS, we take the average of the number of colors predicted by all the ants in a colony in one iteration, and then take the average of this value across different iterations. Note that since we are minimizing the number of colors to be used to color the graph, and our fitness values are the number of colors required, we want our fitness to be decreasing. Utilizing this strategy, we are able to obtain a **minimum value of 16 colors** after experimenting with different parameters and approaches.

## 1.2 Results

### 1.2.1 Testing Algorithm on Smaller Sample Graphs

While formulating the algorithm, we utilized some smaller graphs as our test datasets in order to visualize the results and make debugging easier. The results of two of these graphs, colorized after the algorithm has been applied to them, are shown below.

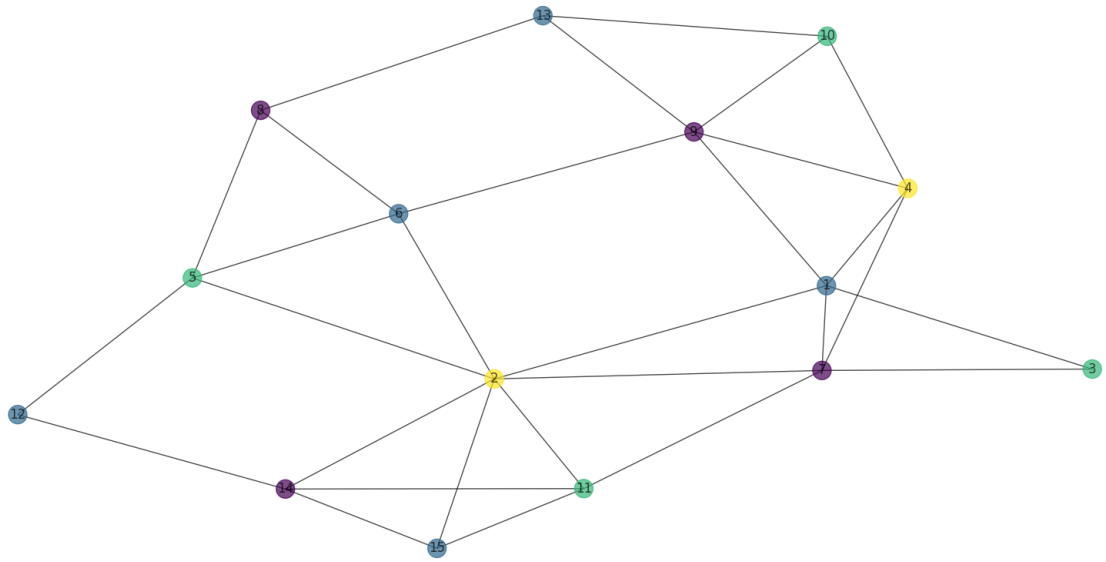


Figure 1.1: Test Sample Graph 1

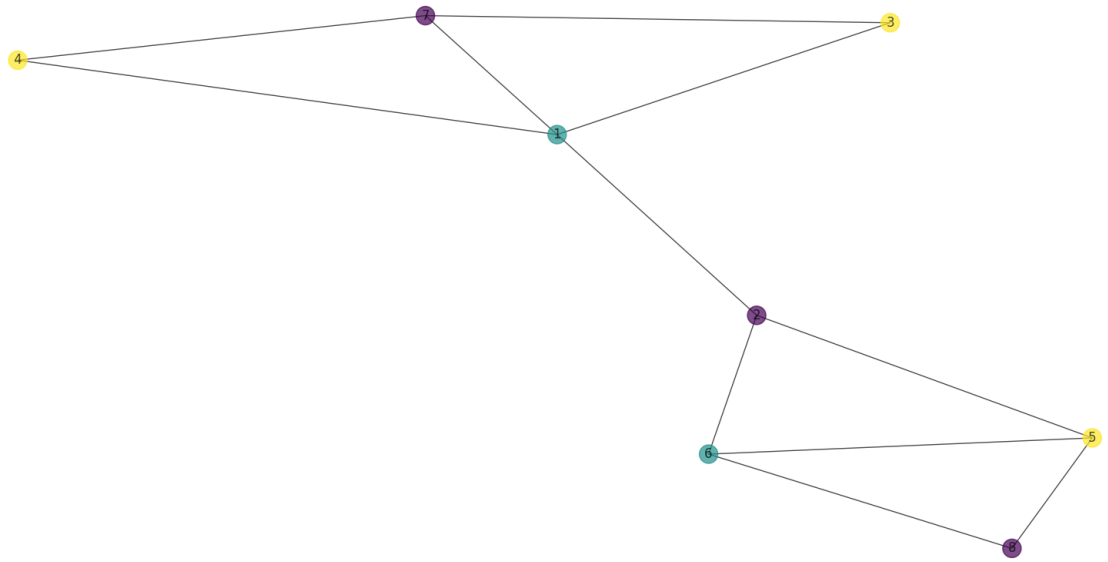


Figure 1.2: Test Sample Graph 2

We have not shown the graphical output for the dataset being tested i.e. 'gcol1.txt' because on account of the high number of edges, it doesn't convey much meaningful information visually.

### 1.2.2 Number of Ants = 20, Iterations = 100, Q = 1, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.8$

This graph was generated using the standard values as given in the assignment instructions to act as a reference point. This graph has helped us determine how changing different parameters changes the behaviour of the algorithm.

The results for this particular combination are sufficiently good, with the algorithm's BFS curve converging to 18 colors in about 5 iterations, while the AFS curve hovers around 21.5 colors.

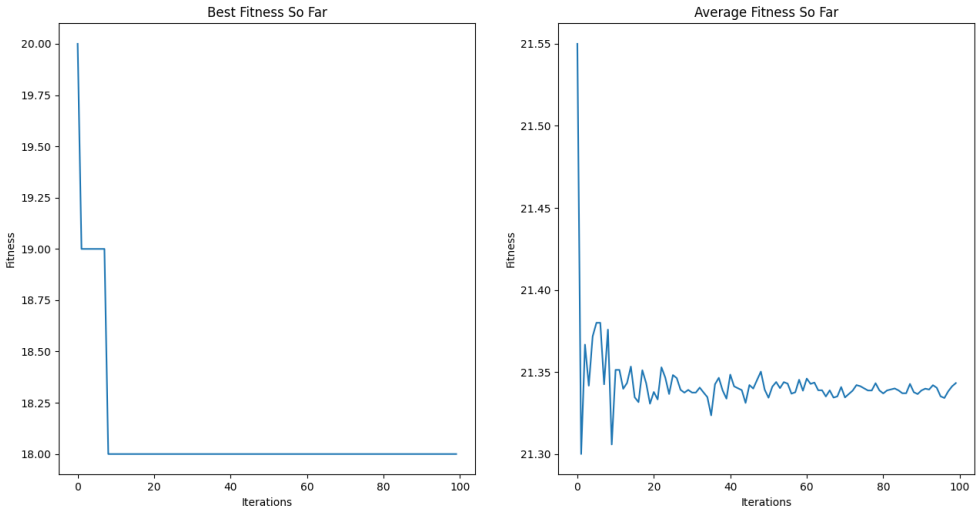


Figure 1.3: BFS (left) and AFS (right)

### 1.2.3 Number of Ants = 200, Iterations = 100, Q = 1, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.8$

We can observe from the results below that if we increase the number of ants while keeping the other parameters constant, our ACO computes 17 to be the minimum number of colors required to color the graph i.e. better accuracy and faster convergence. Whereas, AFS converges to 21 colors.

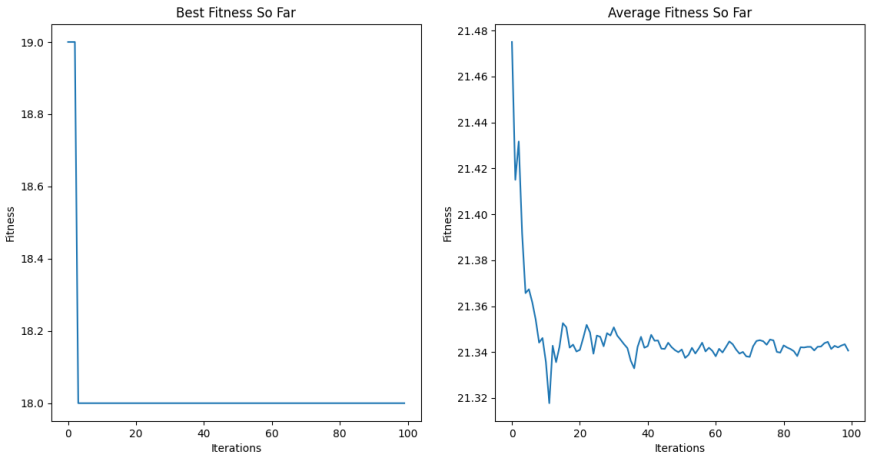


Figure 1.4: BFS (left) and AFS (right)

### 1.2.4 Number of Ants = 200, Iterations = 50, Q = 1, $\alpha = 0.4$, $\beta = 0.8$, $\gamma = 0.8$

We can observe from the results of the last graph and the graph below, that as we decrease $\alpha$, the BFS curve still gives 18 as the minimum colors required but the algorithm converges to 18 after more number of iterations. There is no significant difference in the AFS curve.

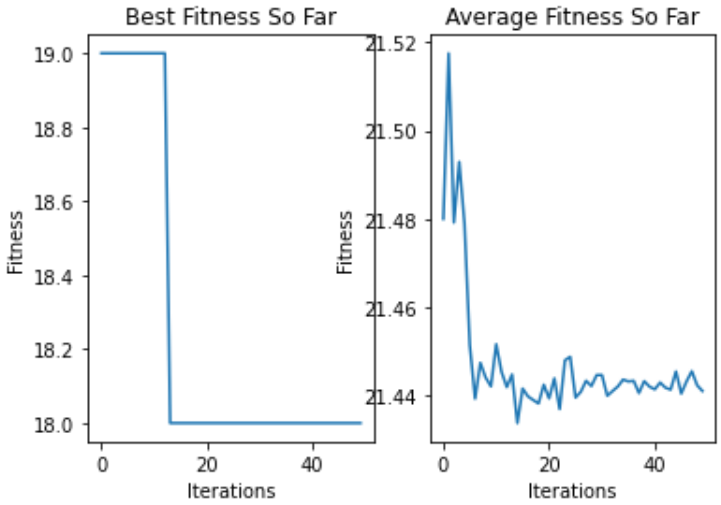


Figure 1.5: BFS (left) and AFS (right)

### 1.2.5 Number of Ants = 50, Iterations = 50, Q = 1, $\alpha = 2$, $\beta = 0.8$, $\gamma = 0.8$

We can observe from the results below that as we increase $\alpha$ to 2, the BFS curve converges to 18 colors in about 5 iterations, which is still more than the number of iterations required when $\alpha$ was 0.8. The AFS curve also converges to a lower value at about 20.9 colors, which is an improvement.

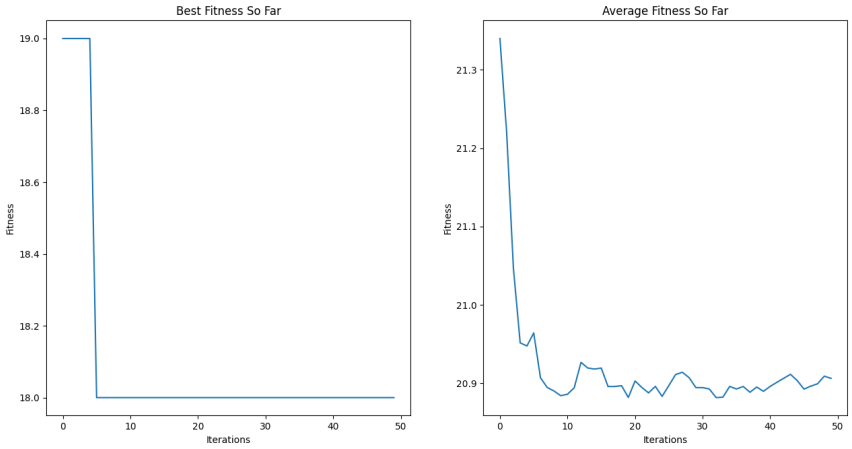


Figure 1.6: BFS (left) and AFS (right)

### 1.2.6 Number of Ants = 1000, Iterations = 50, Q = 1, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.8$

We can observe from the results below that as we increase the number of ants, our ACO algorithm outputs 17 as the minimum number of colors required in quite a few iterations. Which shows that increasing the number of ants improves the performance of ACO in terms of both accuracy and speed. However, the AFS curve hovers around 21.5 colors and does not show much improvement.

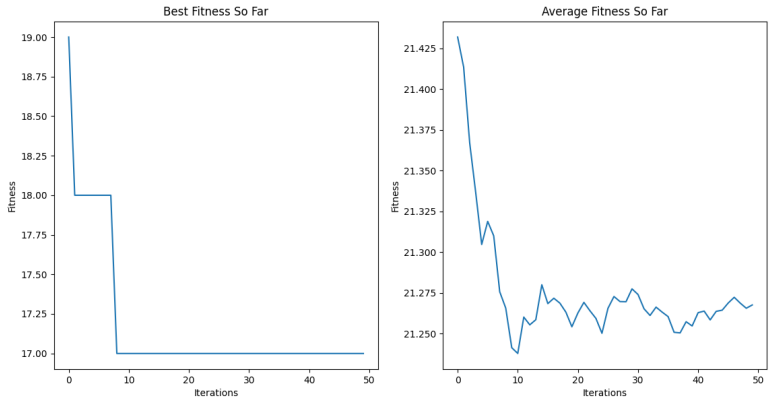


Figure 1.7: BFS (left) and AFS (right)

### 1.2.7 Number of Ants = 200, Iterations = 50, Q = 1, $\alpha = 0.8$, $\beta = 0.4$, $\gamma = 0.8$

From the results below and comparing to previous graphs, we observe that reducing beta doesn't have a major effect on the accuracy of ACO as it still gives 18 as the minimum number of colors required, but more iterations are required to get this result. The AFS curve does not show any significant change.

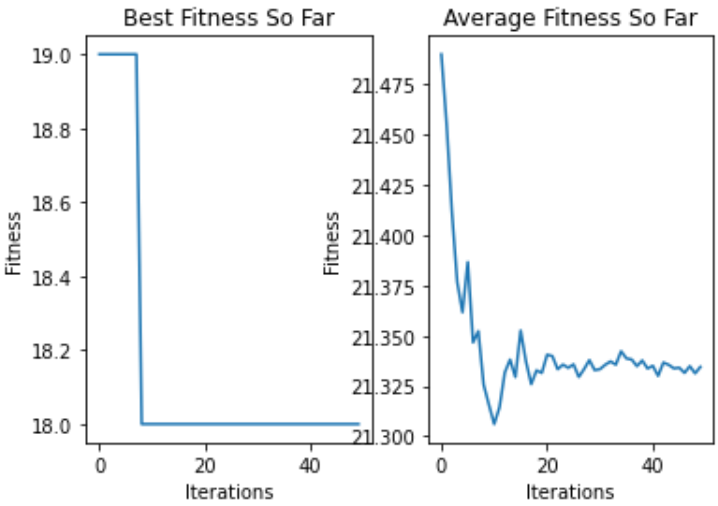


Figure 1.8: BFS (left) and AFS (right)

### 1.2.8 Number of Ants = 200, Iterations = 50, Q = 1, $\alpha = 0.8$, $\beta = 1.5$, $\gamma = 0.8$

From the results below, we note that increasing $\beta$ results in 17 number of colors which is more optimal than previous results. This curve also reaches the value of 18 colors much faster than it's counterpart with $\beta = 0.5$. Overall this shows better performance. The AFS curve remains around 21.5 colors.

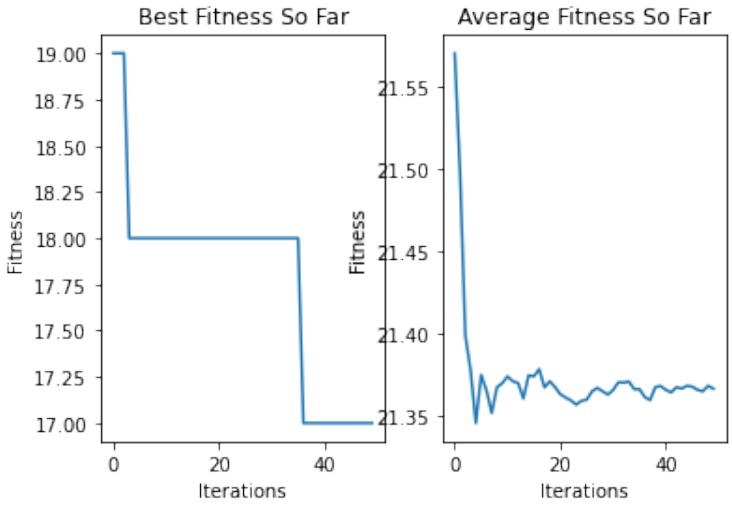


Figure 1.9: BFS (left) and AFS (right)

### 1.2.9 Number of Ants = 200, Iterations = 50, Q = 1, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.4$

From the results below we can conclude that decreasing retention rate still gives us the same minimum number of colors that is 18 but ACO needs much more iterations to converge as compared to previous cases. This makes since since more pheromones will be evaporating and algorithm will have poorer memory. The AFS curve shows a trend of gradually being smoothed out and decreasing which is notably absent in most other variants.

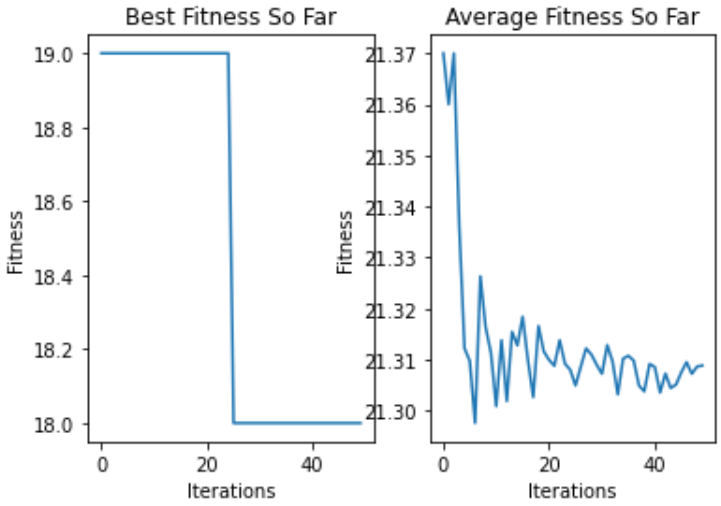


Figure 1.10: BFS (left) and AFS (right)

### 1.2.10 Number of Ants = 200, Iterations = 50, Q = 1, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 1.5$

From the results below, we observe that BFS converges to 18 at higher number of iterations when $\gamma$ increases too. The lack of accuracy makes intuitive since since higher retention rate means that the algorithm may converge to a local optima. The AFS curve seems to be more stable and smooth when retention rate is increased.

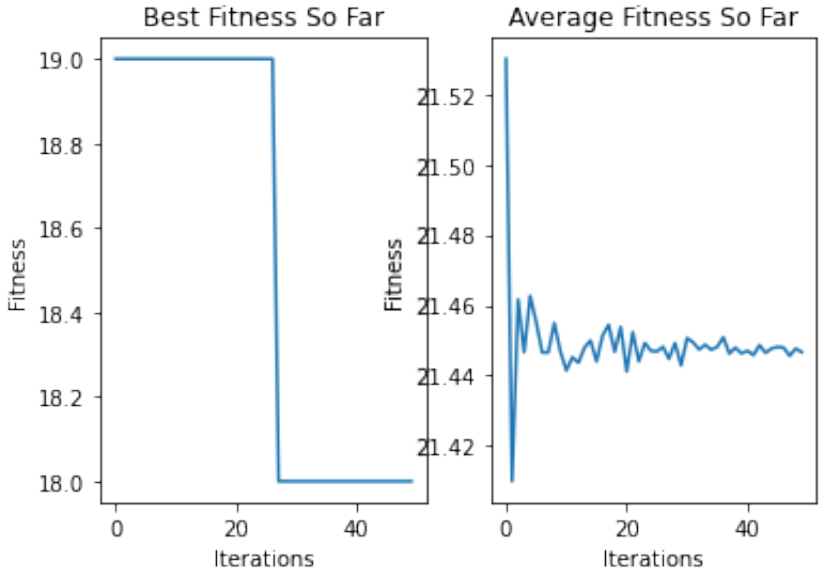


Figure 1.11: BFS (left) and AFS (right)

### 1.2.11 Number of Ants = 1000, Iterations = 50, Q = 0.5, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.8$

From the results below and our previous graphs, we observe that reducing $Q$ still results in the same final accuracy but from BFS we note an interesting behaviour that ACO immediately reaches the optimal solution as $Q$ reduces to half. No real optimization occurs for the specified number of iterations.

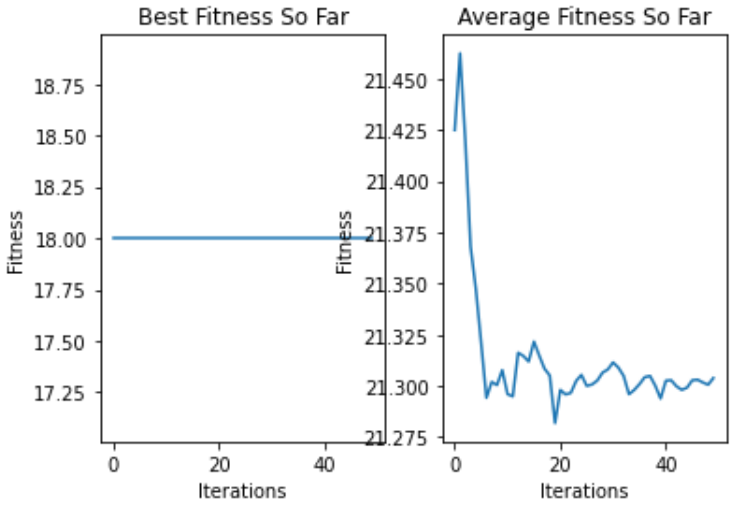


Figure 1.12: BFS (left) and AFS (right)

### 1.2.12 Number of Ants = 200, Iterations = 50, Q = 2, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.8$

From the results below and our previous graphs, we observe that increasing $Q$ makes the BFS curve converge to 18 colors more slowly. There is no significant change in the AFS curve.

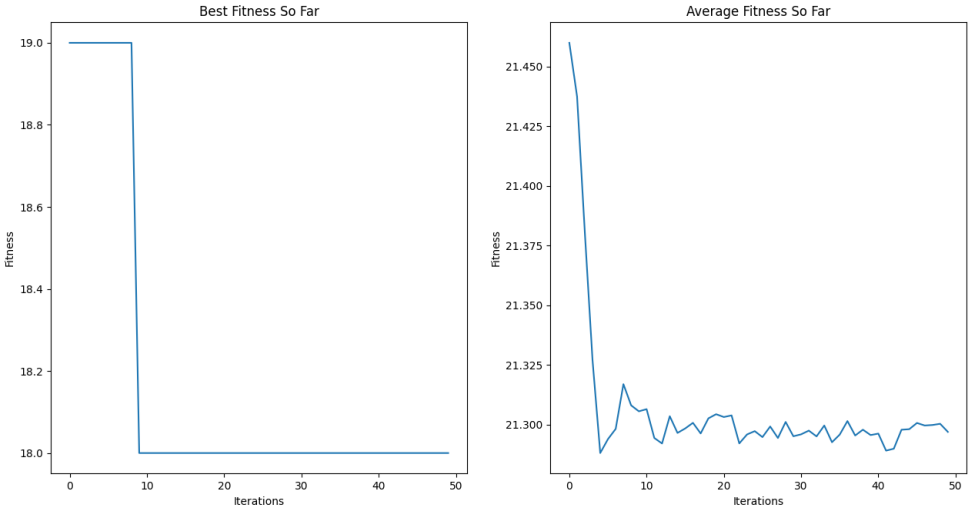


Figure 1.13: BFS (left) and AFS (right)

### 1.2.13 Number of Ants = 100, Iterations = 100, Q = 1, $\alpha = 2$, $\beta = 4$, $\gamma = 0.5$

We try a combination of varying multiple parameters i.e. increasing $\alpha$ & $\beta$ while decreasing $\gamma$ and we observe that ACO achieves a better solution i.e. 17 colors than before at lower number of iterations which wasn't observed in previous cases.

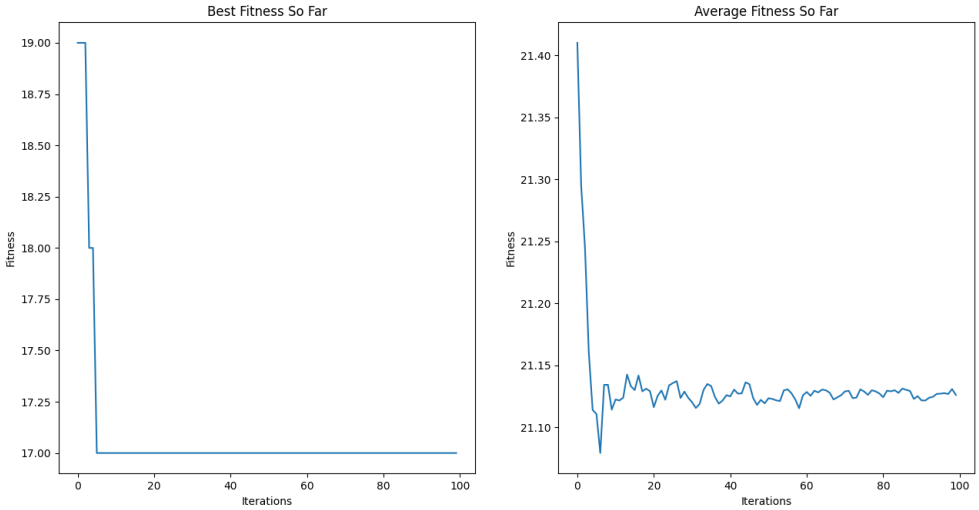


Figure 1.14: BFS (left) and AFS (right)

### 1.2.14 Best Result: Number of Ants = 100, Iterations = 100, Q = 1, $\alpha = 0.8$, $\beta = 0.8$, $\gamma = 0.8$

We are currently getting our optimal result at the aforementioned parameter combinations. It is likely that using 500 or 1000 ants or increasing the number of iterations further may give us better accuracy but that is impractical w.r.t time constraints. This appears to be a good compromise giving us the optimum value of **16 colors**. We also note a trend of AFS curve decreasing and reaching about 20 colors which is noticeably absent in most other combinations.

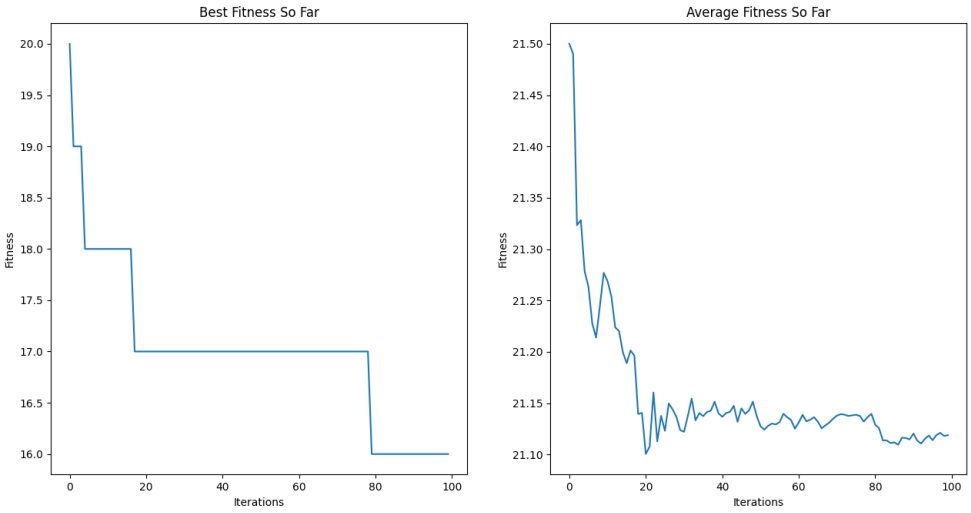


Figure 1.15: BFS (left) and AFS (right)

## 1.3 Analysis and Discussion

Before proceeding to discuss parameter tuning, we would like to make a comment regarding our curves for average fitness. No significant difference was observed across many different parameter combinations, and the curves usually stabilized at around 20/21 colors, even when the minimum colors obtained were 16 or 17. We think this might be because we are using only the best ant to update the pheromone matrix, so all ants in a given colony do not converge to the best solution immediately. Since the average is calculated from all ants in every colony across iterations, the outliers must play a role in preventing the average fitness values to match the best fitness ants. If we were to run the algorithm for a long enough number of iterations, we believe that we would notice a more pronounced downward trend in the AFS curves as better ants dominate the colonies as a whole.

### When Number of Ants is tuned while other parameters are kept constant

During our fine tuning we observed that when the number of ants is increased while other parameters are kept constant, then the ACO performance improves in terms of the minimum value obtained and convergence also occurs at a faster rate.

This makes sense intuitively since more ants traversing the graph make it more likely that at least one of the ants will come up with the most optimal coloring.

### When $Q$ is tuned while other parameters are kept constant

We note that when $Q$ is decreased, we do not have any significant optimization that takes place. The algorithm reaches an optimal in the very first iteration and does not show any improvement for the specified number of iterations. When $Q$ is increased, we reach the optimal value of 18 colors more slowly.

We know that $Q$ defines the weight of the best ant pheromone matrix as it is added to the global pheromone matrix. Small value of $Q$ therefore means that there are minimal updates to the value of the global pheromone matrix hence minimal optimization. On the other hand, larger value of $Q$ means that algorithm is more biased towards the path taken by a specified best ant, and may take longer in order to converge to a better solution value.

### When $\alpha$ is tuned while other parameters are kept constant

We observe that when $\alpha$ is increased, the BFS reaches convergence faster. On the other hand, when $\alpha$ is decreased, BFS takes more iterations to reach the minimum value.

We know that $\alpha$ controls the impact of the trail intensity i.e. pheromones on the probabilities of node selection. It makes sense that when the probabilities are strongly influenced by the pheromones of previous ants, it will converge to a solution faster (Whether it is local or global optima). However, when $\alpha$ has a lower value and pheromones have smaller impact on the probability, there is more stochasticity involved and it takes longer to reach the optimal value.

### When $\beta$ is tuned while other parameters are kept constant

We observe that as $\beta$ is reduced ACO converges more slowly. When $\beta$ is increased, the performance is greatly improved, we get a better solution which also seems to be converging faster.

Intuitively, this can be understood as we know that $\beta$ weighs the visibility which effects the probability of selecting next node for traversal. Here our visibility is defined by our degree of saturation. Higher value of $\beta$ means that we tend to choose the node with higher degree of saturation, which is known to increase the performance of an algorithm compared to the greedy approach or other alternatives.

### When retention rate is tuned while other parameters are kept constant

We observe that decreasing retention rate slows down our convergence while increasing the retention rate has a similar impact.

We learn from this behaviour that there needs to be an optimum value of retention rate. If the retention rate is too high, we have very little evaporation which means that we will converge to a local optima i.e. our algorithm is too exploitative. If the retention rate is too low, we have too much evaporation of pheromones and our algorithm will not perform optimally since it is too explorative.

# 2 Know More About Optimization

## 2.1 Meta-heuristic Optimization

Optimization is the action of making the best or most effective use of a resource or situation. These are often minimization or maximization problems. Optimization techniques can be categorized based on what derives them. One important factor that is a useful metric to determine the nature of optimization techniques is randomness. If an algorithm doesn't employ randomness to reach the final state, that is to say that it always reaches the same final state given the same input, then it's called Deterministic Optimization Algorithm. Whereas if an algorithm employs randomness such that at every execution a unique final state may be found, then it is categorized as Stochastic Optimization Algorithm.

Meta-heuristic optimization techniques fall under the aforementioned stochastic category because they employ randomization to achieve global optima by maintaining a balance between diversification and intensification (where both global and local search spaces are explored respectively). In other words, they have to maintain a balance between exploration and exploitation to perform optimally. Meta-heuristic algorithms incorporate a certain tradeoff of randomization and local search. For these reasons, almost all such algorithms are highly suited for obtaining global optima. All modern nature inspired algorithms are meta-heuristic in nature. Meta-heuristic techniques are one step above simple heuristic techniques in terms of complexity and efficacy.

One popular example of a meta-heuristic problem is the ant colony optimization problem as explored in part 1 of this assignment.

## 2.2 Gradient Based Optimization

Optimization techniques are often categorized using the gradient of a function as a metric. Algorithms which employ the gradient of a function for optimization are known as gradient based optimization algorithms, whereas those which require only the value of the function and not its gradient are known as gradient-free optimization algorithms.

Following are the situations where gradient based optimization techniques do not work:

1. When the function is non-smooth or noisy since the derivative is difficult to calculate accurately and the algorithm becomes unreliable.
2. When the function is discontinuous in nature because then its derivative does not exist for the discontinuous regions.
3. When it is far too computationally expensive to accurately compute gradient, so the technique becomes impractical.

4. When the function has a local optima since this algorithm can easily get stuck there and fail to converge to a global optima.

We can resolve the aforementioned problems by utilizing derivative free optimization techniques such as the genetic algorithms.

## 2.3 Artificial Bee Colony (ABC) Optimization

ABC is a bio-inspired meta-heuristic optimization technique which adopts the behavior of bees. In one popular variant of ABC, there are three types of bees namely scout bees, onlooker bees, and employed bees. Scout bees are the ones which search for new food sources and when they find a food source they communicate through pheromones and/or a 'waggle dance' to share the location of food source with onlooker bees. The onlooker bees determine the importance of the food source based on nectar quantity. More nectar in a food source means it's more optimal. The food sources stored in the memory of the hive are exploited by the employed bees until the sources are exhausted which is when they transform into scout bees and start searching for a new food source, and the process repeats itself.

Each employed bee is associated to a unique food source (possible solution) and hence number of employed bees is equal to the number of solutions at a given time. In ABC, as in any other meta-heuristic algorithm, scout bees perform exploration by randomly searching for food throughout the search space and employed bees perform exploitation to exploit food sources e.g. flower patches in the locality. Nectar and/or pheromones are used by onlooker bees in order to prioritize between different food sources (solutions) over iterations.

Just like ACO, ABC is also highly suited for discrete and combinatorial optimization, and can be used in a wide variety of applications.

# 3 References


- Costa, Daniele, and Alain Hertz. "Ants can colour graphs." Journal of the operational research society 48.3 (1997): 295-305.
- Bessedik, Malika, et al. "Ant colony system for graph coloring problem." International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). Vol. 1. IEEE, 2005.
- Hertz, Alain, and Nicolas Zufferey. "A new ant algorithm for graph coloring." Workshop on Nature Inspired Cooperative Strategies for Optimization NICSO. 2006.
- Bui, Thang N., et al. "An ant-based algorithm for coloring graphs." Discrete Applied Mathematics 156.2 (2008): 190-200.