# Ant Algorithms applied in the Maximum Flow Problem

Miron Czech, Jakub Pryc, Ivan Zarzhitski

AGH University of Science and Technology

## 1 Introduction

The optimization of flow in networks is a significant challenge across various fields, including transportation, telecommunications, and computer networks. The maximum flow problem, a fundamental optimization problem in network theory, aims to determine the maximum amount of flow that can be sent through a network from a source to a sink. Solving this problem efficiently has practical implications for network design and resource allocation. Traditional algorithms have made considerable progress, but they often face limitations in terms of scalability and adaptability to dynamic network environments.

In recent years, nature-inspired algorithms have gained significant attention for their ability to solve complex optimization problems. Ant algorithms, inspired by the foraging behavior of ants, have emerged as a promising approach to tackle various combinatorial optimization problems. These algorithms use a decentralized and iterative process, where artificial ants simulate the real-world behavior of ants to find optimal solutions.

## 2 Technology

We used Python programming language for ease of development and good performance when combined with NumPy computational library. For data visualisation we used matplotlib library that provides flexible options for plotting our results.

## 3 Algorithms

We decided to create three different algorithms using different approaches and comparing their performance to one of Ford-Fulkerson algorithm. This allowed us to asses how close our results were to optimal solutions to the problem and how fast our algorithms were.

### 3.1 Algorithm 1

1. Preparation. For each arc produce virtual arcs according to capacity. The capacity of virtual arcs starts from 0 then adds 1 in turn until the original capacity of the arc.
2. Remember $\tau$ number for every arc created in 1. with the same initial value for each one.
3. Choose one arc with probability equal $p = \frac{\tau_j}{\sum_{i=1}^{n} \tau_i}$ for n arcs created at j index.
4. Create a new graph with chosen arcs and perform on it classical Ford-Fulkerson algorithm.
5. For each chosen arc modify its $\tau$ according to the formula: $\tau_i(k+1) = \tau_i(k) \cdot \rho + \frac{f}{Q}$, where $\rho \in (0,1)$, $Q \in \{2,3,4,...\}$ are constant for the simulation and $f$ is achieved flow (0 if couldn't reach chosen arcs sum of flow).

Repeat the algorithm m times.

### 3.2 Algorithm 2

1. Preparations. For each arc produce virtual arcs according to capacity. The capacity of virtual arcs starts from 0 then adds 1 in turn until the original capacity of the arc. In addition choose "restoration" subgraph with n - 2 arcs. Order the vertices in restoration order.

2. Remember $\tau$ number for every arc created in 1. with the same initial value for each one.
3. Choose one arc with probability equal $p = \frac{\tau_j}{\sum_{i=1}^{n} \tau_i}$ for n arcs created at j index.
4. Try to restore the graph using "restoration" subgraph and the flow rule. If not remake step 3.
5. For each chosen arc modify its $\tau$ according to the formula: $\tau_i(k+1) = \tau_i(k) \cdot \rho + \frac{f}{Q}$, where $\rho \in (0,1)$, $Q \in \{2,3,4,...\}$ are constant for the simulation and $f$ is achieved flow (0 if couldn't reach chosen arcs sum of flow).

Repeat the algorithm m times.

### 3.3 Algorithm 3

In each of n iterations m ants will traverse the graph. On each vertex they will choose an edge and go to the next vertex until they cannot go anywhere or have reached the sink. An iteration can be described by the following procedure:

1. define a graph called a *residual network* that is a copy of the graph given by the problem.
2. individually "release" m ants into the graph at the source vertex and for each of them continue as follows:
    - choose the next vertex *j* from the set *k* of neighbouring vertices with current one with probability:
    $$p_{i,j} = \frac{(\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}}{\sum_{s \in k}(\tau_{i,s})^{\alpha}(\eta_{i,s})^{\beta}} \tag{1}$$
    where $\tau$ is pheromones on edge *i,j*, $\eta$ is the capacity of the edge, $\alpha$ and $\beta$ are parameters of exploitation and exploration.
    - repeat this process of choosing each time assuming chosen vertex *j* as new vertex *i* until the ant reaches the sink or has nowhere to go.
    - if the ant reached the sink then find the capacity *c* of the lowest capacity edge on the traversed path.
    - subtract *c* from all traversed edges and add to all edges (creating new ones when necessary) opposite to the ones traversed.
    - add *c* divided by the size of the graph to the pheromones on this edge for the next iteration.
    - increase *flow* of the current iteration by *c*
3. after all ants finished update the pheromone values with this iteration changes and multiply it by $\rho \in (0,1)$, the factor of evaporation repeat for n times.

## 4 Measurements

We created a simple algorithm for creating randomized directed acyclic graphs of given size, density and capacity ranges that we later used to improve our testing. We took measurements for different graph sizes and iteration counts. Unfortunately we didn't have enough computing possibilities to get results from algorithm 2.

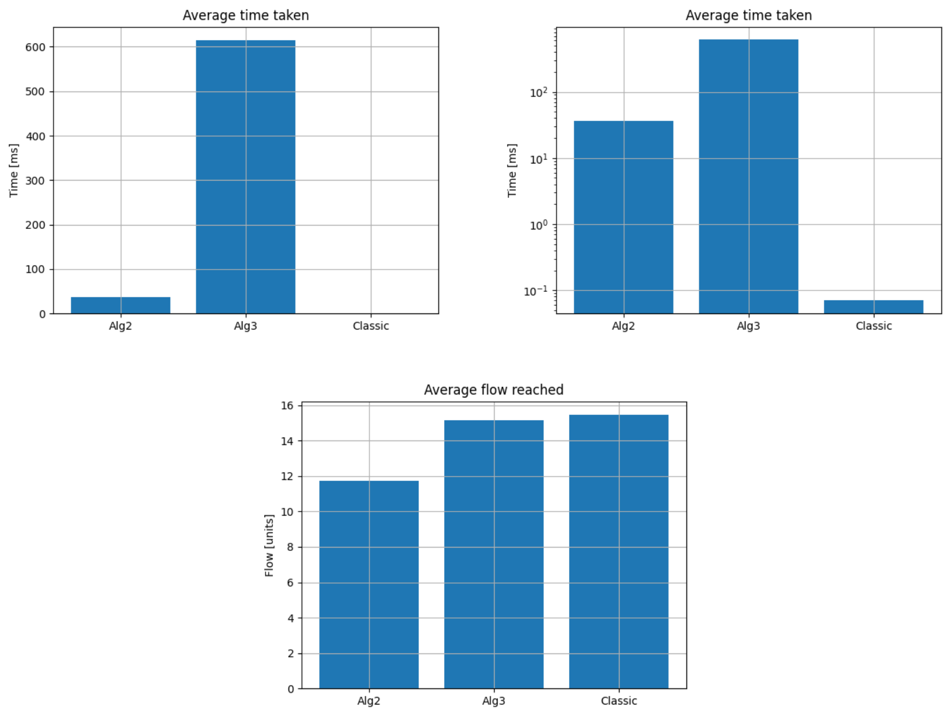

Figure 1: Average time and maximum flow measurements for graphs of size 10 and 100 iterations

Analyzing Figure 1 we can see that algorithm 3 is almost as accurate as Ford-Fulkerson algorithm in terms of accuracy, however the difference in speed is highly visible on the graph using logarithmic scale. The ant algorithms are slower by a few orders of magnitude.

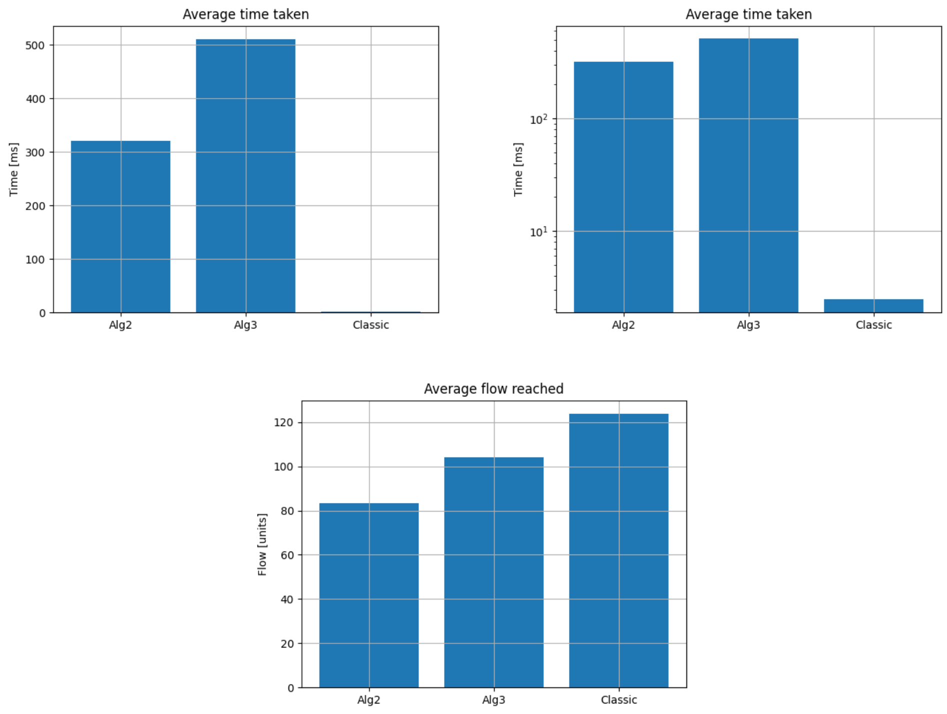

Figure 2: Average time and maximum flow measurements for graphs of size 50 and 100 iterations

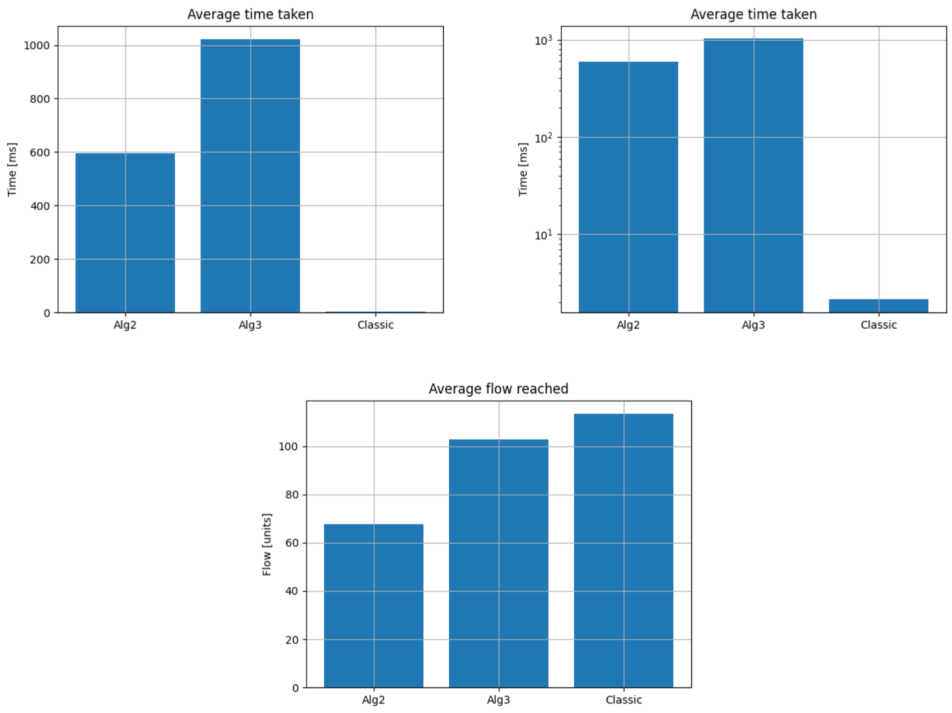

Figure 3: Average time and maximum flow measurements for graphs of size 50 and 200 iterations

On Figures 2. and 3. we can observe that by increasing the number of iterations we can significantly improve the results of ant algorithms. This effect can be seen on the following figure:

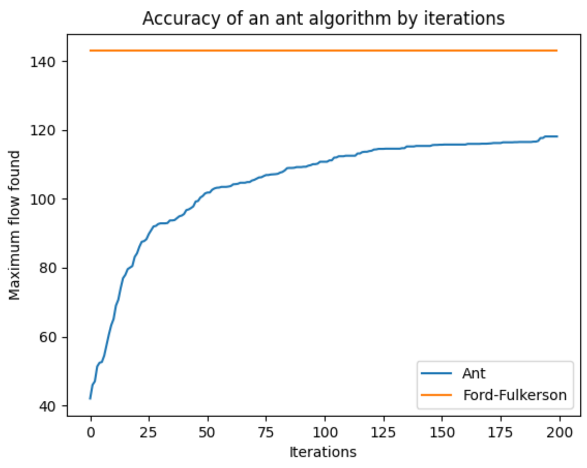

Figure 4: Accuracy of an ant algorithm by iterations

## 5 Summary

Ant algorithms proved to be useful in many combinatorial optimization problems, such as Travelling Salesman Problem because of their adaptability and flexibility. Based on the findings, we concluded that while ant algorithms have shown promise in solving various other problems of similar nature, their application to the maximum flow problem falls short of the performance achieved by classical algorithms. The Ford-Fulkerson algorithm is one of known algorithms that solves maximum flow problem in polynomial time complexity. The ant algorithms seem much more applicable to NP-complete problems, because their adaptability in approximating the solution can be sufficient for many applications and more desirable than computationally expensive optimal solutions.