Optimization Techniques
Minimization of a multivariable function – Genetic algorithms

Dimitrios Karatis 10775, *Electrical and Computer Engineering, AUTH*

***Abstract*—This document gives an overview of how optimization techniques can be used to minimize travel times in a traffic network. It focuses on using mathematical modeling and genetic algorithms to efficiently distribute vehicle flows across roads, aiming to balance traffic. The project tackles a key challenge in transportation systems, where managing congestion and optimizing travel times are crucial for urban planning and traffic control.**

***Index Terms*—optimization techniques, minimization, genetic algorithms, mathematical modeling, traffic, traffic control.**

I. Introduction

Consider a road network where nodes represent intersections and edges represent traffic routes with specified directions. Each edge is numbered for identification. When traffic is light, travel times between intersections are relatively constant. However, as the number of vehicles increases, travel times grow significantly due to congestion. For a given road i, let t_i (in minutes) represent the baseline travel time under negligible traffic conditions. Also, x_i is defined (vehicles per minute) as the flow rate of vehicles on road i, and c_i (vehicles per minute) as the maximum capacity of that road. Lastly, the travel time on road i as a function of x_i is given by:

$$T_i(x_i) = t_i + a_i \frac{x_i}{1 - \frac{x_i}{c_i}} \text{ [minutes].}$$

It is observed that $\lim_{x_i \to 0} T_i(x_i) = t_i$ and $\lim_{x_i \to c_i} T_i(x_i) = +\infty$. The objective is to minimize the total travel time across the network for a given input flow V (vehicles per minute), while ensuring that vehicles entering a node equal those exiting it, thus avoiding accumulation or deficits at any intersection.

The values of c_i are indicated in red on the edges of the network. For example, $c_1 = 54.13$. Additionally, we will consider that $a_i = 1.25$ for $i = 1, \ldots, 5$, $a_i = 1.5$ for $i = 6, \ldots, 10$, and $a_i = 1$ for $i = 11, \ldots, 17$, with the total input flow being $V = 100$. Below we can see an image of the network.

This document was prepared by Dimitrios Karatis as part of the final assignment for the Optimization Techniques course at Aristotle University of Thessaloniki.

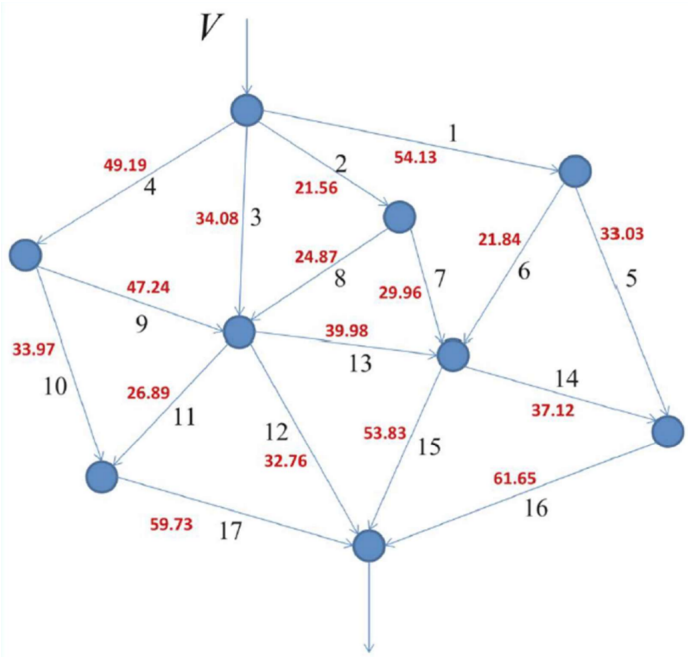

Fig. 1: The given traffic network

II. Mathematical Formulation of the Problem

The goal is to mathematically formulate the optimization problem for minimizing the total travel time across the given road network. Each road i has a base travel time t_i (in minutes) when traffic is light, a maximum vehicle flow capacity c_i (in vehicles per minute), and an additional delay proportional to the current vehicle flow x_i. The travel time on road i is expressed as:

$$T_i(x_i) = t_i + \frac{a_i x_i}{1 - \frac{x_i}{c_i}},$$

where a_i is a constant dependent on the road's characteristics. As traffic increases towards the maximum capacity ($x_i \to c_i$), the denominator approaches zero, causing the travel time to approach infinity. The total travel time across the network for all vehicles is then given by:

$$T_{\text{total}} = \sum_{i=1}^{n} T_i(x_i),$$

where n is the total number of roads in the network.

The problem requires minimizing T_{total} under the following constraints:

1) **Flow Conservation at Intersections:** For every intersection, the total incoming flow must equal the total outgoing flow. This ensures that no vehicles accumulate or disappear at intersections.
2) **Total Vehicle Flow:** The sum of the flows across all roads must equal the total incoming vehicle rate, $V = 100$ vehicles/min.
3) **Flow Capacity:** The flow on each road x_i must not exceed its maximum capacity c_i, i.e., $x_i \leq c_i$.

The optimization problem can be formulated as:

$$\textbf{Objective:} \quad \min_{x_1,x_2,\ldots,x_n} \sum_{i=1}^{n} \left(t_i + \frac{a_i x_i}{1 - \frac{x_i}{c_i}} \right)$$

Constraints:

1) $\sum_{\text{incoming}} x_i = \sum_{\text{outgoing}} x_i$, $\forall$ intersections.
2) $\sum_{i=1}^{n} x_i = V$.
3) $x_i \leq c_i$, $\forall i$.

or:

- $x_1 + x_2 + x_3 + x_4 = V$.
- $x_1 = x_5 + x_6$.
- $x_2 = x_7 + x_8$.
- $x_4 = x_9 + x_{10}$.
- $x_3 + x_8 + x_9 = x_{11} + x_{12} + x_{13}$.
- $x_{13} + x_7 + x_6 = x_{14} + x_{15}$.
- $x_{14} + x_5 = x_{16}$.
- $x_{11} + x_{10} = x_{17}$.
- $x_{17} + x_{12} + x_{15} + x_{16} = V$.
- $x_i \geq 0$, $\forall i$.
- $x_i - c_i \leq 0$, $\forall i$.

III. Implementing a genetic algorithm (GA)

The genetic algorithm is designed to *minimize* the total traversal time by *maximizing* the **fitness function**, defined as **- total_time** , within the network (graph). This process is carried out while ensuring compliance with the problem's constraints, which were previously analyzed in the Mathematical Formulation. The key stages of the algorithm are outlined below:

A. Creating the Initial Population

The initial population for the genetic algorithm is generated by randomly assigning weight values while ensuring they satisfy predefined constraints. The process begins by distributing an initial set of weights in a way that maintains a specified total value. These weights are then progressively subdivided into smaller components, maintaining proportional relationships and adhering to upper and lower limits. Additional constraints are applied to ensure all weight values remain within valid ranges, preventing negative values or exceeding predefined thresholds. This structured approach ensures that the generated population is diverse and meets the necessary conditions for optimization in the genetic algorithm.

B. Fitness Function

The fitness function evaluates each individual in the population based on the total traversal time, with the goal of minimizing it. It ensures that population values remain within valid limits and then calculates the time required for each individual using a predefined mathematical model. The total traversal time

$$T_{\text{total}} = \sum_{i=1}^{n} T_i(x_i),$$

$$T_i(x_i) = t_i + \frac{a_i x_i}{1 - \frac{x_i}{c_i}},$$

is summed across all individuals, and to align with the genetic algorithm's objective of maximizing fitness, the function returns the negative of this total time:

$$F_{\text{Fitness}} = -T_{\text{total}}$$

C. Roulette wheel selection

The roulette wheel selection function is responsible for choosing individuals for the next generation based on their fitness values. It assigns selection probabilities proportionally, favoring individuals with better fitness scores (less negative fitness, meaning smaller overall time) while still giving weaker ones a chance to be selected. To enhance differentiation, fitness values are scaled non-linearly, increasing the likelihood of selecting high-performing individuals. A cumulative probability distribution is then used to simulate a roulette wheel mechanism, where random values determine which individuals are chosen. This selection method ensures diversity while guiding the genetic algorithm toward minimizing total traversal time, as defined by the fitness function.

D. Crossover

The crossover mechanism is designed to generate new potential solutions by combining characteristics from two parent solutions. This process aims to introduce genetic diversity while ensuring that the offspring remain valid within the problem constraints.

First, two offspring are initialized as exact copies of their parents. Then, a random crossover point is selected, which determines where the genetic material will be exchanged between the parents. The first part of the offspring's genetic material is inherited from one parent, while the remaining part is taken from the other. This results in two new offspring that are combinations of their parents.

To ensure that the generated solutions remain valid, the algorithm checks whether the offsprings satisfy the predefined constraints. If both offsprings are valid, they are returned as the next generation. However, since a random crossover might produce infeasible solutions, the algorithm attempts the crossover multiple times (up to a maximum number of attempts). If after several tries no valid offsprings are found, the original parents are retained to prevent the introduction of invalid solutions.

E. Mutation

The mutation function introduces random changes to the weight values (meaning the x values) to promote diversity and explore the solution space. It applies a mutation rate to decide which weights will be mutated. The changes are drawn from a normal distribution with a mean of zero and a standard deviation scaled by a factor of the current weight, allowing for controlled perturbations.

$$\textit{mutation\_amount} \sim \mathcal{N}(0, \sigma \cdot w_i)$$

$$w_{\text{new}}(i) = w(i) + \textit{mutation\_amount}$$

This ensures that the changes are generally small, with occasional larger variations, which helps in searching for better solutions. After each mutation, the function checks if the new weights are valid, ensuring they remain within the problem's constraints. If the mutated solution is invalid, the function attempts to fix the weights through multiple iterations. If the solution is still invalid after several attempts, the weights are reverted to their original values. This process ensures that the algorithm explores new solutions while maintaining feasibility and respecting the problem constraints.

F. Stopping Criteria

The algorithm terminates when the improvement in fitness values between generations becomes negligible, meaning it stops when it gets smaller than a predefined threshold. Specifically, the change is calculated as:

$$\Delta f = \frac{||f_{\text{k+1}} - f_{\text{k}}||}{||f_{\text{k}}||}.$$

If $\Delta f < \epsilon$, where ϵ is a very small threshold, for example, 10^{-7}, the algorithm is considered to have converged.

G. Creating the Next Population

The *GenerateNextPopulation* function creates the next generation of individuals in the population based on their fitness levels. It uses the fitness proportions, which determine how many times each selected individual should be included in the new population. Individuals with higher fitness are more likely to be selected and reproduced, thus increasing their representation in the next generation. If the population is not fully filled by the selected individuals, the remaining spots are filled with randomly chosen individuals from the current population. This ensures that the next generation is a mix of well-performing individuals and some diversity, promoting exploration and exploitation in the genetic algorithm.

H. The main Genetic Algorithm

The process begins with an initial population, which is randomly generated and evaluated based on its fitness, which in this case corresponds to the total time calculated from the design variables multiplied by -1. The algorithm uses the fitness of individuals to guide the search for better solutions, employing selection, mutation, and crossover techniques.

In each generation, individuals are selected for reproduction based on their fitness scores, with a higher fitness increasing the likelihood of selection. The selected individuals then undergo crossover, exchanging genetic material to produce offspring with a combination of traits from both parents. This promotes the exploration of new solution spaces. To maintain diversity and avoid premature convergence, a mutation operation is applied to some individuals, slightly altering their genetic material. Also, the mutation rate decreases over generations, gradually fine-tuning the solutions as the algorithm converges.

The algorithm is designed to stop either when the maximum number of generations is reached or when the population's fitness converges, indicating that further generations would yield no significant improvement. Throughout the process, the algorithm also maintains feasibility by ensuring that the constraints are met in the generated solutions, ensuring that only valid individuals are considered in each generation.

Lastly, the fitness evolution and the changes in the best and average fitness over generations are plotted to provide insights into the algorithm's progress and overall effectiveness and efficiency.

IV. For a total incoming vehicle rate of 100 (V = 100 = const)

A. Results, Plots, Observations

When keeping the incoming vehicle rate V to 100 we get the following plots, for a specific run and a population size of 30:

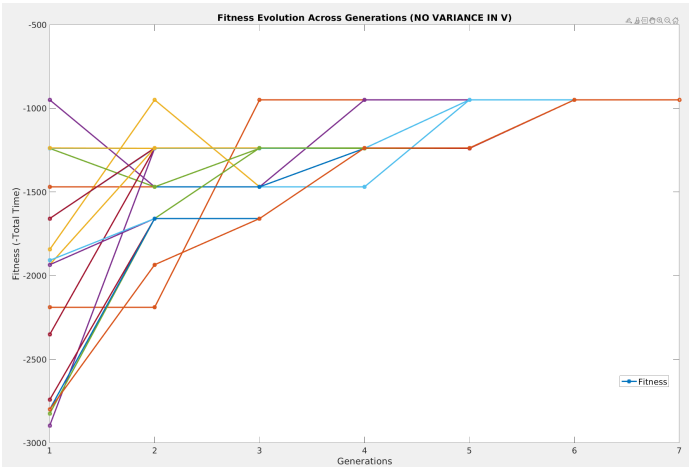

Fig. 2: Fitness Evolution Across Generations (**NO variance in V**)

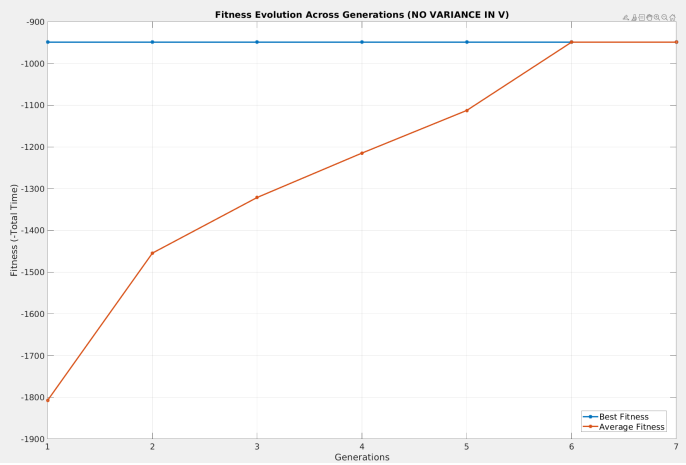

Fig. 3: Fitness Evolution Across Generations (**NO variance in V**)

The results of my genetic algorithm implementation for this specific run indicate that convergence was achieved within five generations, as shown in the fitness evolution plots. The first plot illustrates the progression of fitness values for multiple individuals, revealing a general trend of improvement as weaker solutions were phased out. The second plot, which tracks the best and average fitness values, provides deeper insight into the algorithm's performance. Notably, the best fitness remains flat after an initial improvement, suggesting that the algorithm has reached a plateau and is no longer discovering better solutions. This stagnation could indicate that the algorithm has become trapped in a local minimum, preventing further optimization. Meanwhile, the average fitness continues to improve, demonstrating that the population as a whole is still evolving, although without finding a more optimal solution beyond the current best individual. The final chromosome values represent the best solution found, with a total time of 949.04. The relatively fast convergence suggests that the selection, crossover, and mutation strategies effectively guided the population toward an optimal or near-optimal solution. However, to mitigate the risk of premature convergence, future iterations of the algorithm could incorporate mechanisms such as increased mutation rates, adaptive selection pressure etc.

```
Number of generations until convergence: 7
Final Chromosome:
34.22 11.65 24.57 29.56 21.02 13.21 5.96 5.69 8.87 20.69 1.31 23.56 14.26 21.17 12.25 42.19 21.99
Total time: 949.04
The solution is feasible.
```

Fig. 4: Results (**NO variance in V**)

The main code for this implementation can be found at the ***GeneticAlgorithm_Part1.m***

V. For a total incoming vehicle rate that can fluctuate by ±15% of it's initial value (V = ± 15% · 100)

A. Results, Plots, Observations

When introducing variance to the incoming vehicle rate by ± 15% of the initial V value, we get the following plots, for a specific run and a population size of 30:

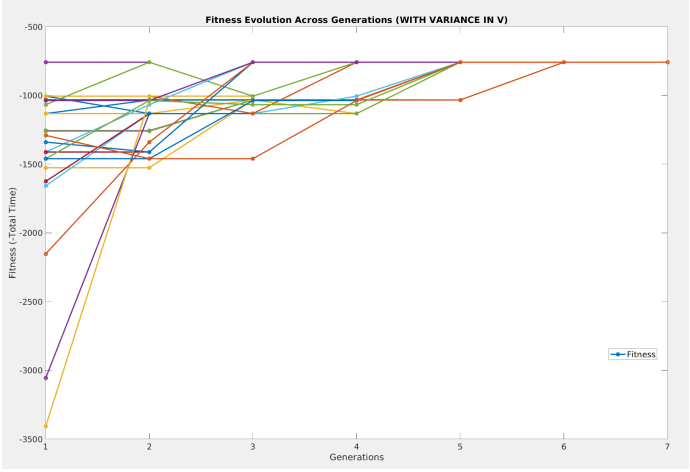

Fig. 5: Fitness Evolution Across Generations (**WITH variance in V**)

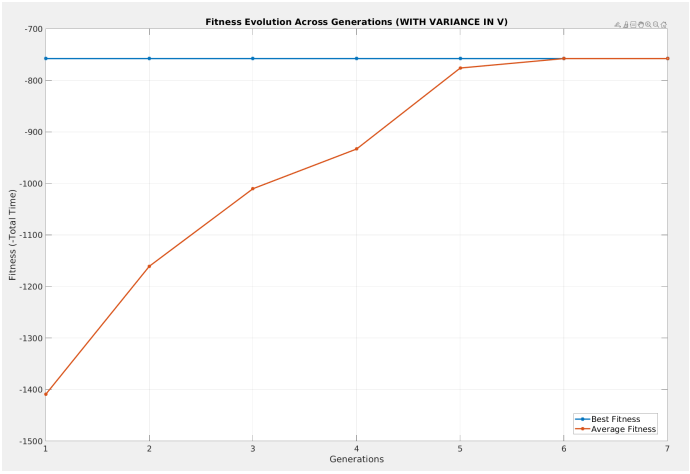

Fig. 6: Fitness Evolution Across Generations (**WITH variance in V**)

After introducing variability to the parameter V, the optimization process showed a reduction in the total time, reaching a value of 757.79. The fitness evolution graphs illustrate key observations about the performance of the genetic algorithm.

In the first plot, which shows the fitness of individual solutions across generations, there is an initial improvement in fitness values, with some solutions significantly increasing in performance. However, after a few generations, the improvement plateaus, suggesting that the algorithm has converged. This is further confirmed by the second plot, which tracks the best and average fitness over generations. The best fitness remains flat after a certain point, while the average fitness continues improving until it stabilizes. This indicates that the algorithm may have reached a local minimum, where further exploration does not yield significant improvements.

The algorithm converged after seven generations, meaning that no further enhancements in fitness were observed beyond this point. While the final solution is feasible, the stagnation of the best fitness suggests that the algorithm may be stuck in a local optimum rather than finding the global best solution. This could be due to insufficient genetic diversity in later

generations or the need for a different selection or mutation strategy to escape local minima.

Overall, the introduction of variance in V positively impacted the optimization, as it resulted in a lower total time. However, the observed stagnation in fitness values highlights potential areas for improvement in the genetic algorithm's design, such as adjusting mutation rates or introducing adaptive techniques to enhance exploration beyond local optima.

```
V: 86.01
Number of generations until convergence: 7
Final Chromosome:
26.28 1.81 22.11 35.81 16.37 9.92 1.24 0.57 18.06 17.75 12.96 17.21 10.57 18.68 3.05 35.05 30.71
Total time: 757.79
The solution is feasible.
```

Fig. 7: Results (**WITH variance in V**)

The main code for this implementation can be found at the ***GeneticAlgorithm_Part2.m***