Optimization Techniques for Multi Cooperative Systems MCTR 1021
Mechatronics Engineering
Faculty of Engineering and Materials Science
German University in Cairo

Optimization Based Merging for Connected Vehicles

By

Team 22
Marwa Lotfy Ismail
Ahmed Mohamed Sleem
Nada Tamer Abdelhay
Dina Mohamed Saad

Course Team

Dr. Eng. Omar Mohmoud Mohamed Shehata
Eng. Mohamed Salah
Eng. Jessica Magdy

December 25, 2023

This is to certify that:

(i) the report comprises only my original work toward the course project,

(ii) due acknowledgment has been made in the text to all other material used

Team 22
December 25, 2023

Contents

List of Abbreviations

CAVs	Connected and Automated Vehicles
MS	Merging Sequence
CACC	Cooperative Adaptive Cruise Control
AVS	Autonomous vehicles

List of Figures

Chapter 1

Introduction

1.1 Smart transportation and autonomous vehicles

A new era of urban development known as ”smart cities” is being ushered in by the use of cutting-edge technology to improve resident quality of life while also advancing efficiency and sustainability. The idea of smart transport, a dynamic and networked system intended to completely transform the movement of people and products inside and between these cutting-edge urban environments, is at the core of this urban evolution. In this sense, smart transport represents a major change in the conception, planning, and operation of cities, offering safer, more convenient, and environmentally friendly mobility options for all. It goes beyond simply streamlining routes and easing traffic congestion. Autonomous vehicles (AVS) have become a game-changer in smart cities, offering revolutionary answers to persistent urban transportation problems. Their exceptional coordination and communication skills have been important in helping to reduce traffic congestion. Autonomously navigating highways, AVS may merge and manoeuvre with efficiency, interacting with infrastructure and other cars to maximise traffic flow. By doing this, bottlenecks are lessened and safety is improved. Because AVS can anticipate and adjust to traffic patterns, intersection management—which is frequently a cause of congestion and accidents—has improved. Wait times are shortened and traffic flow is improved as a result. Overall, by drastically lowering traffic, enhancing intersection management, and expediting highway merging and ramp operations, autonomous cars have transformed transportation in smart cities and, in the process, raised the prospect of a safer, more effective, and sustainable urban lifestyle. The way we travel on the roads of the world is changing due to connected vehicles, which are sometimes referred to as the keystone of the transportation industry to come. These sophisticated cars and other forms of transportation are outfitted with state-of-the-art technology that enables real-time communication between them and the traffic infrastructure. The data that linked cars exchange opens up a world of amazing opportunities. They can enhance safety by alerting drivers of accidents and traffic bottlenecks, allowing for speedier route adjustments. They give traffic management systems vital information that helps to improve traffic flow and lessen congestion. Furthermore, because they can share data to manage challenging road conditions and enhance overall road safety, linked vehicles open the door for the development of autonomous driving.

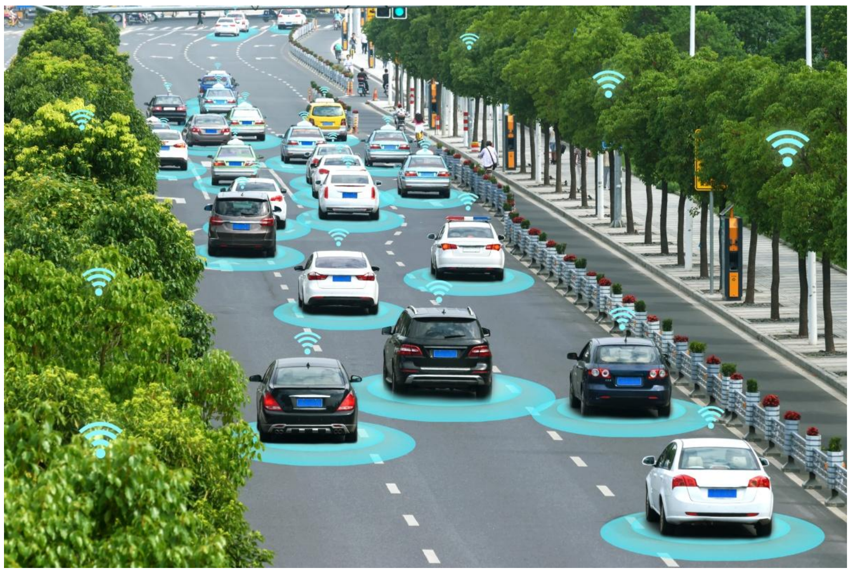

Figure 1.1: Autonomous vehicles

1.2 Merging vehicles in Highways on Ramps

Merging control of vehicles at highway on-ramps is a critical element in optimizing traffic flow and enhancing overall transportation efficiency. This process entails the seamless integration of vehicles entering the highway from an on-ramp with the existing traffic. Efficient merging is vital for reducing congestion, minimizing the potential for accidents, and ensuring the uninterrupted movement of vehicles on the highway. Advanced technologies, such as connected vehicles and autonomous driving systems, are poised to revolutionize this aspect of highway operations. Connected vehicles can communicate with each other and with the highway infrastructure in real-time, sharing information about their positions, speeds, and intentions. This enables a level of coordination and synchronization previously unattainable with traditional human drivers. AVS equipped with precise sensors, artificial intelligence, and instant communication capabilities, can anticipate traffic patterns and identify suitable gaps for merging onto the highway. These vehicles adjust their speed and trajectory in real-time, negotiating entry onto the highway with unparalleled precision. By removing human error and hesitation from the equation, these technologies not only make merging smoother but also enhance safety by reducing the risk of collisions and bottlenecks at on-ramps. Moreover, they enable a more efficient use of highway space and a more predictable traffic flow, ultimately improving the overall driving experience. As smart cities continue to evolve, the integration of merging control systems for vehicles at highway on-ramps holds great promise in reducing urban congestion and enhancing transportation sustainability. This technological advancement represents a critical step toward the realization of safer, more efficient, and more environmentally friendly mobility solutions in our increasingly interconnected and intelligent transportation networks.

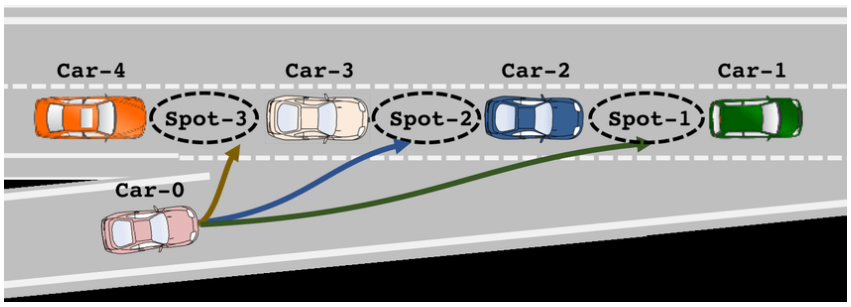

Figure 1.2: Merging vehicles

Chapter 2

Literature Review

There have been multiple research on the optimization of merging for connected vehicles. the study in [1] proposes a cooperative merging strategy for connected vehicles that uses a genetic algorithm to optimize traffic flow and minimize travel time. The strategy takes into account the inflow from on-ramps and uses wireless communication to improve traffic efficiency and fuel consumption. The paper presents simulation scenarios to verify the effectiveness of the proposed strategy and concludes with areas for future work.
There has been another study in [2] that proposes an optimization framework and analytical solution for coordinating connected and automated vehicles Connected and Automated Vehicles (CAVs) at merging roadways to achieve smooth traffic flow without stop-and-go driving. The effectiveness of the proposed solution is validated through simulation, showing significant reductions in fuel consumption and travel time. The majority of research efforts in this area have focused on intersections and merging highways.
Also the study in [3] presents an optimization framework for cooperative merging of platoons of connected and automated vehicles at highway on-ramps. The problem is modeled as a job-shop scheduling problem, and an optimal scheduling algorithm is presented to derive the sequence and time for each platoon to enter the highway safely. An optimal control problem is also formulated to derive the optimal control input (acceleration/deceleration) in terms of fuel consumption of the platoons at the highway. Simulation results show that the proposed framework significantly reduces crossing time and fuel consumption of platoons at highway on-ramps. The paper contributes to the literature on scheduling theory-based control algorithms for optimizing vehicle coordination in various transportation scenarios.
Another study in [4] discusses a rule-based cooperative merging strategy for connected and automated vehicles at highway on-ramps. The authors derive a near-optimal merging sequence using a rule-based adjusting algorithm and an energy-efficient motion planning method. Simulation-based case studies are conducted to compare the effectiveness and robustness of their method with other control strategies. The proposed strategy consistently outperforms the

other methods in terms of throughput and delay under unbalanced vehicle arrival rates. The paper provides a comprehensive review of related works on cooperative merging and identifies the differences of their work from the existing works. Overall, the paper presents an efficient and safe method for coordinating two strings of vehicles at highway on-ramps.
Finally, the paper in [5] presents a game-based approach to optimize merging behavior for connected and automated vehicles. The proposed approach improves traffic flow and fuel economy while reducing collision risks. The main contributions of the study are the mixed-strategy Nash equilibrium of the complete information static game for Merging Sequence (MS) allocation, the varying-scale grid search algorithm to numerically search for acceptable parameters in the cost function of the bi-objective optimization problem, and the adoption of a modified quick-sort algorithm to find the Pareto front. The paper also applies Pontryagin's minimum principle to calculate the optimal control law for the bi-objective optimization problem.
Ramp metering is an important concept for traffic management strategy used on highways and freeway systems to regulate the flow of vehicles entering the mainline traffic from on-ramps. It involves the use of traffic signals or control devices at the on-ramps to control the rate at which vehicles merge onto the mainline.
The metrics used to evaluate the performance of the proposed approach are average travel time, number of cars crossing the merging zone and average vehicle speed.

Chapter 3

Methodology

3.1 Problem Formulation

3.1.1 Objective Function

Given m number of vehicles traveling on the main road in a platoon formation and r number of vehicles traveling on the ramp towards the merging point, The aim is to discover the optimal merging sequence between vehicles from the on-ramp and those already on the main highway that maximizes the total number of vehicles successfully merging from the ramp 3.1, while minimizing the overall delay for vehicles already on the main road eq:3.2, where $r1$ is the number of vehicles that are left for the next merging decision.

$$f_1 = \frac{r - r_1}{r} \tag{3.1}$$

$$f_2 = \frac{-\sum_{i=1}^{m} t_i + \sum_{i=1}^{m} \frac{l_i}{v_{\min}}}{-\sum_{i=1}^{m} \frac{l_i}{v_{\max}} + \sum_{i=1}^{m} \frac{l_i}{v_{\min}}} \tag{3.2}$$

The final objective function is:

$$\max F = w_1 * f_1 + w_2 * f_2 \tag{3.3}$$

3.1.2 Constraints

The objective function in 3.3 is subject to the following constraints:

$$\begin{gathered}
0 \leq w_1 \leq 1 \\
0 \leq w_2 \leq 1 \\
w_1 + w_2 = 1 \\
a_{\min} \leq a_i \leq a_{\max} \quad \text{for } i = 1, 2, \ldots, m \\
a_{\min} \leq a_j \leq a_{\max} \quad \text{for } j = 1, 2, \ldots, r \\
v_{\min} \leq v_i \leq v_{\max} \quad \text{for } i = 1, 2, \ldots, m \\
v_{\min} \leq v_j \leq v_{\max} \quad \text{for } j = 1, 2, \ldots, r
\end{gathered}$$

When a vehicle enters the control zone, all the data regarding the vehicle motion is transmitted to a central unit having all the data for all the vehicles inside the control zone. For the main road vehicles. once a vehicle enters the control zone, its initial acceleration is maintained constant until it reaches the decision point. The optimization problem is solved before the decision point is reached. After passing the decision point and the optimal sequence is obtained, Cooperative Adaptive Cruise Control (CACC) law 3.4 is applied to generate the trajectory for all vehicles in the solution on the array, when an on-ramp is scheduled to merge, the following vehicles in the sequence create a space for the vehicle to be merged in the future as the on-ramp vehicle is virtually moving with the main line vehicle until it merges at the merging point.

$$a_f = \alpha \cdot a_l + \beta \cdot (v_f - v_l) + \gamma \cdot (s - s_d) \tag{3.4}$$

where s_d is the distance between the following vehicle and the preceding vehicle.

3.1.3 Decision variable

The decision variable of this problem is the merging sequence between the on-ramp and main road vehicles, the sequence matrix represented in binary, where the main road cars are represented as 1, and the ramp cars will be represented as 0, an example of the solution array is shown in Fig.3.2.

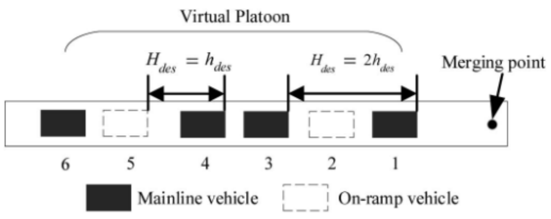

Figure 3.1: virtual platooning

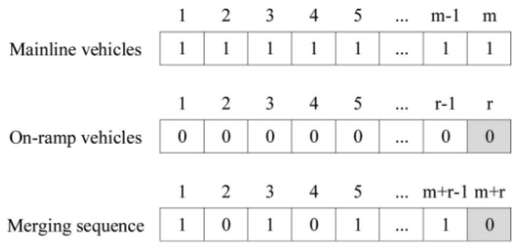

Figure 3.2: Solution Array

3.1.4 Assumptions

Several key assumptions are made to streamline the complexities of vehicle merging in a controlled environment. Firstly, the merging point of vehicles is assumed to be fixed upstream of the exit, ensuring a consistent location for merging. Lane-changing maneuvers are prohibited within the control zone, simplifying the scenario to include only the rightmost lane of the main road and a single lane on the on-ramp. Additionally, the focus is placed on longitudinal control of vehicles, disregarding lateral control, such as lane changes. The assumption of Cooperative Adaptive Cruise Control (CACC) vehicles, equipped with Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication, enables the exchange of state information and real-time merging instructions. The merging process is divided into short time intervals, with constant acceleration assumed within each interval, facilitating the modeling and analysis of vehicle behavior. These assumptions collectively provide a controlled and predictable merging environment, essential for the development and optimization of cooperative merging strategies.

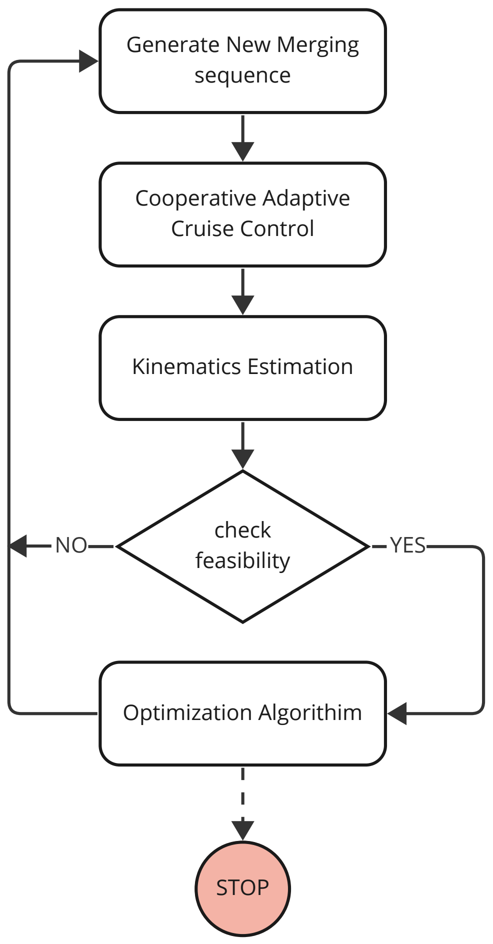

Figure 3.3: Proposed merging framework

3.2 Simulated annealing

Simulated Annealing (SA) is a heuristic optimization algorithm inspired by the annealing process in metallurgy, designed to find near-optimal solutions in complex search spaces. Starting with an initial solution, SA iteratively explores neighboring solutions, with its key feature being the ability to accept worse solutions with a decreasing probability as it cools down according to a predefined temperature schedule. This stochastic acceptance criterion allows the algorithm to escape local optima and converge towards global or near-optimal solutions. SA has been widely applied to various optimization problems, offering a versatile approach to challenges with intricate solution landscapes, making it a valuable tool for finding high-quality solutions in diverse domains, such as the traveling salesman problem, job scheduling, and parameter tuning in machine learning.

3.2.1 Procedure

1. Initialization: Begin with an initial solution to the optimization problem.
2. Temperature Initialization: Set an initial temperature parameter, which dictates the probability of accepting solutions with higher objective function values.
3. Iteration: Iterate through the following steps until a stopping criterion is met:
4. Generate Neighbor Solution: Generate a neighboring solution by perturbing the current solution, introducing randomness to the search process.
5. Objective Function Comparison: Compare the objective function values of the current and neighboring solutions.
6. Acceptance Probability: Calculate the probability of accepting the new solution based on the temperature and the difference in objective function values.
7. Update Solution: Accept the new solution with the calculated probability or, in some cases, with certainty if it improves the objective function value.
8. Temperature Update: Update the temperature according to a cooling schedule, gradually reducing it over iterations.
9. Stopping Criterion Check: Check whether a stopping criterion, such as a maximum number of iterations or a target temperature, has been reached.
10. Convergence: Conclude the algorithm when the stopping criterion is met, and the final solution is considered the optimized or near-optimal solution.

The process of SA is also illustrated in Fig.3.4.

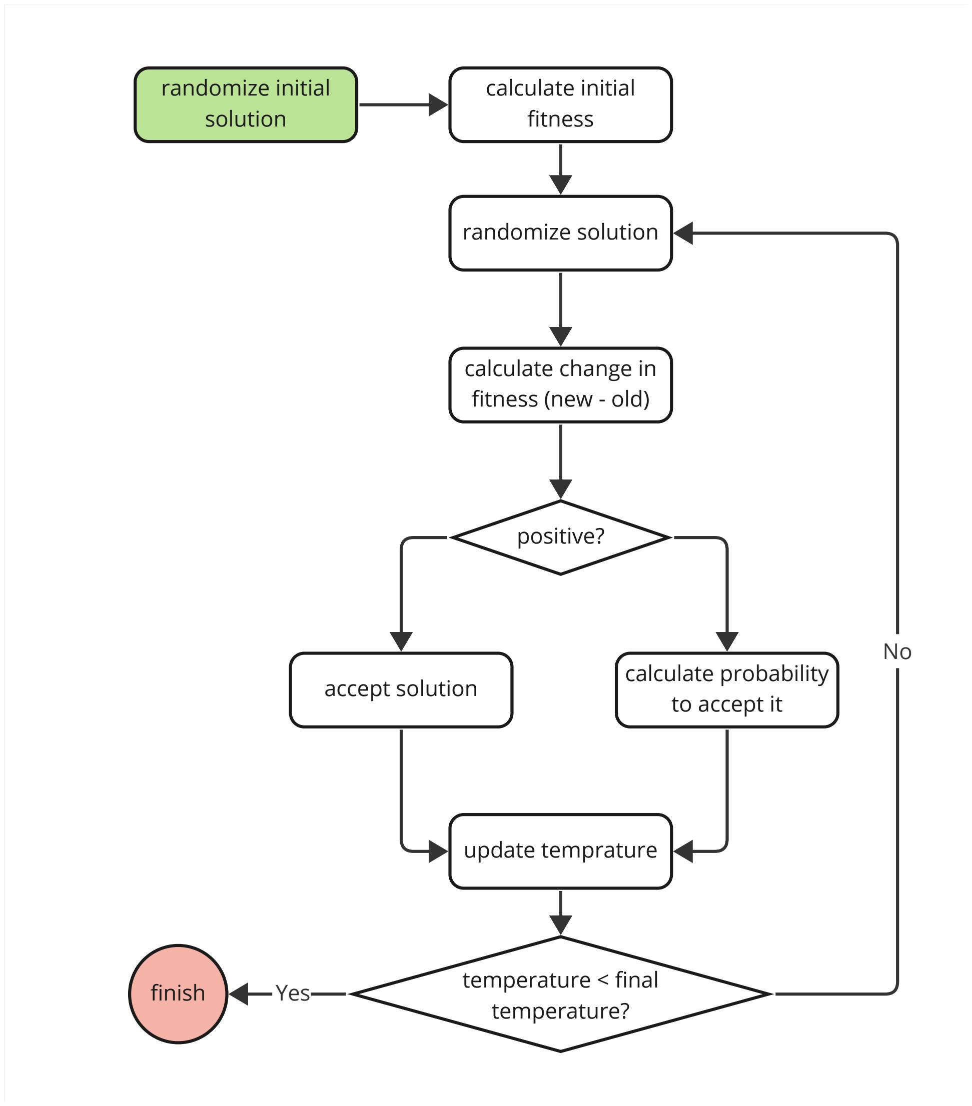
randomize initial solution
calculate initial fitness
randomize solution
calculate change in fitness (new - old)
positive?
accept solution
calculate probability to accept it
update temprature
temperature < final temperature?
No
Yes
finish

Figure 3.4: Flow of the Simulated Annealing Algorithm

3.2.2 Pros and Cons

Simulated annealing (SA) possesses notable strengths in optimization. It excels in global searches, making it effective for problems with multiple solutions. SA is adaptable and can handle various problem types without relying on function differentiability. Furthermore, its ability to escape local optima enhances its efficacy. In terms of cons, SA doesn't guarantee optimality as it is a heuristic method that explores the solution space probabilistically. Parameter tuning is crucial, and suboptimal choices may impact performance. While SA is computationally less expensive than some alternatives, it may still face challenges with large-scale problems. Additionally, handling constraints in SA can be intricate, and its convergence rate might be slower compared to other optimization techniques. Despite these drawbacks, SA remains a versatile optimization tool with applications across diverse problem domains.

3.3 Genetic Algorithm (GA)

A genetic algorithm (GA) is a search heuristic inspired by the process of natural selection and genetics. It belongs to the broader class of evolutionary algorithms, which are optimization algorithms inspired by the principles of biological evolution. GAs are commonly used to find approximate solutions to optimization and search problems.

3.3.1 Procedure

Here's an overview of the key components and procedures of a genetic algorithm:

1. Initialization: Generate a population of potential solutions (individuals or chromosomes). The solutions are typically represented as strings of parameters, where each parameter represents a potential solution to the problem.
2. Fitness Evaluation: Evaluate the fitness of each individual in the population. The fitness function quantifies how well an individual solves the problem at hand. The fitness function is problem-specific and defines the objective or goal of the optimization.
3. Selection: Select individuals from the current population to be parents based on their fitness. Individuals with higher fitness are more likely to be selected, mimicking the natural selection process.
4. Crossover (Recombination): Pair selected individuals and create new offspring by combining their genetic information. This is typically done by exchanging portions of the parents' chromosomes.
5. Mutation: Introduce random changes in the offspring's genetic information to maintain genetic diversity. Mutation helps explore new regions of the solution space.
6. Replacement: Replace the old population with the new population of parents and offspring.
7. Termination: Repeat the process for a certain number of generations.

The process of (GA) is also illustrated in Fig.3.5.

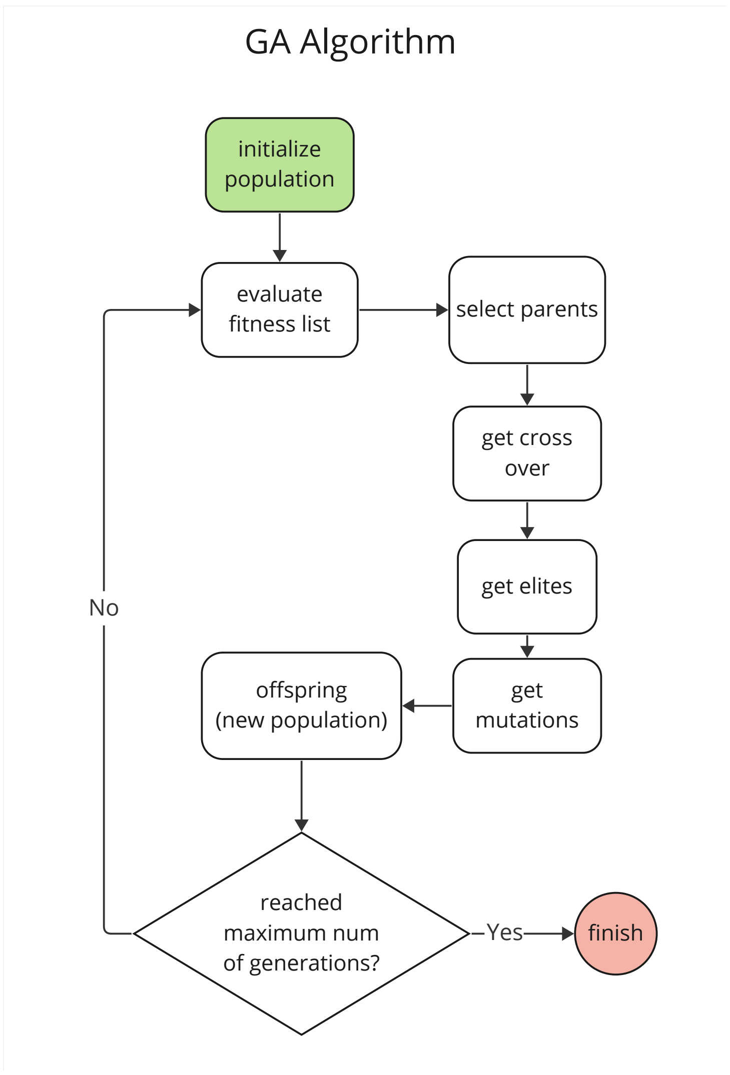

Figure 3.5: Flow of the Genetic algorithm

3.3.2 Pros and Cons

Genetic algorithms (GAs) offer several advantages, including their proficiency in conducting global searches across entire solution spaces, rendering them suitable for problems with multiple solutions. Their adaptability is evident in their capability to handle diverse problem types without relying on the differentiability of the objective function. Furthermore, GAs can exploit parallelism, allowing for the efficient utilization of parallel computing resources. However, GAs come with inherent limitations. They do not guarantee optimality, functioning as heuristic methods that may not always find the optimal solution. The performance of GAs is sensitive to parameter choices, necessitating careful tuning for different problem instances. Additionally, GAs can incur computational costs, especially for intricate problems and large populations. Another challenge lies in their limited ability to handle constraints, with some genetic algorithms struggling in problems involving constraints, often requiring supplemental techniques for improved performance.

3.4 Discrete Particle Swarm algorithm

Particle Swarm Optimization (PSO) is a nature-inspired optimization algorithm that draws inspiration from the social behavior of bird flocks and fish schools. Introduced by Kennedy and Eberhart in 1995, PSO is a population-based algorithm that simulates the movement of particles in a multidimensional search space. Each particle represents a potential solution to the optimization problem, and their positions are adjusted over iterations based on their own experience and the collective knowledge of the swarm. PSO is particularly well-suited for continuous optimization problems but can also be adapted for discrete and combinatorial problems.

For the discretization of the particle swarm algorithm in the context of a binary optimization problem, such as optimizing the merging of ramp cars on highways, certain modifications are required. In a binary problem, the decision variables take only binary values (0 or 1), representing the absence or presence of a certain feature or action. In the case of ramp car merging, each particle in the swarm corresponds to a potential solution, and the position of the particle is encoded as a binary string that represents the decision variables related to the merging actions. The updating mechanism of the particles is then adjusted to accommodate the binary nature of the problem, ensuring that the particles explore the solution space effectively and converge to optimal or near-optimal solutions.

3.4.1 Discrete Particle Swarm Optimization Procedure

1. Initialization: Initialize a swarm of particles, each representing a potential solution to the binary optimization problem. Assign random binary values to the decision variables of each particle. Initialize the particle's velocity, which represents the change in the binary positions.

2. Initial fitness Evaluation: Evaluate the fitness of each particle based on the objective function

3. Personal and Global Best Update: Update the personal best position for each particle based on its current fitness. Update the global best position by considering the fitness of all particles in the swarm.

4. Velocity and Position Update: Update the binary position of each particle based on its velocity, the equation is presented in the implementation section. Aditionally, update the fitness.

5. Convergence Check: Check if reached maximum number of iterations.
6. Repeat: If convergence criteria are not met, repeat steps 3-5 until convergence.
7. Solution Extraction: Extract the best solution found by the swarm, which corresponds to the global best position.

The process of DPSO is also illustrated in Fig.3.6.

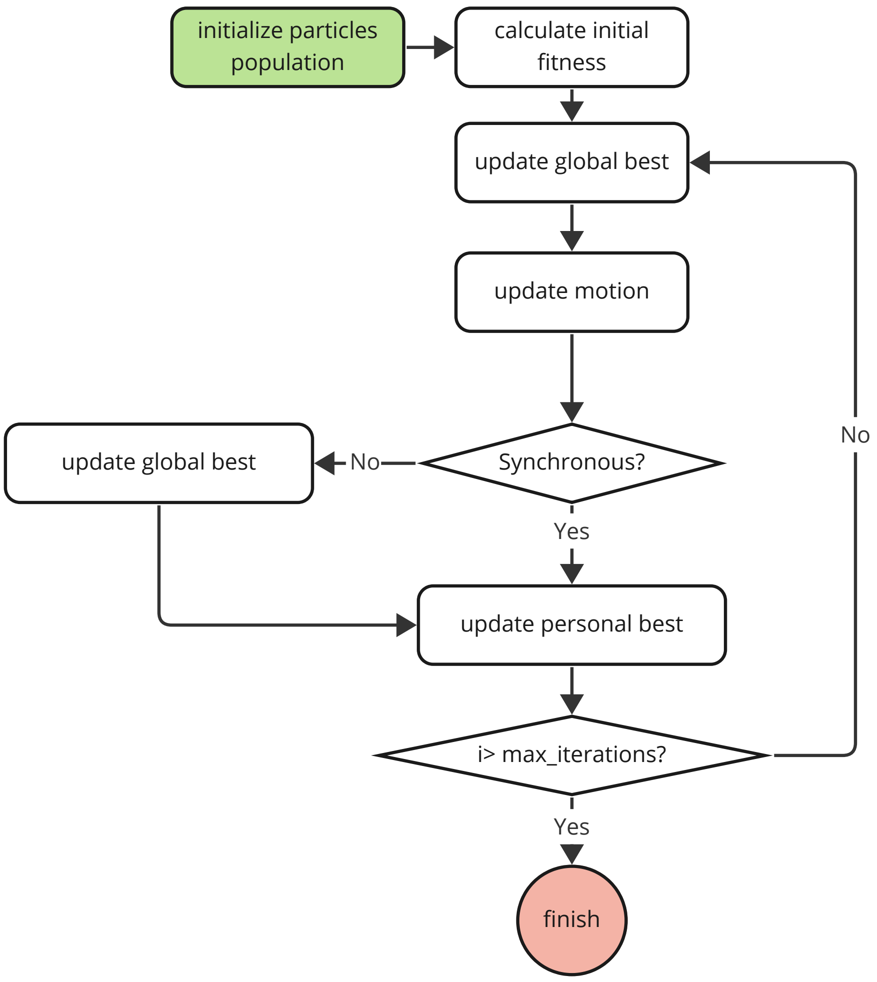

Figure 3.6: Flow of the Discrete particle swarm algorithm

3.4.2 Pros and Cons

Pros of using the discretized particle swarm algorithm for binary optimization problems such as highway ramp car merging include its simplicity, ease of implementation, and ability to handle high-dimensional search spaces. The algorithm's exploration-exploitation balance, inspired by social interactions, allows for efficient global search and local refinement. Additionally, PSO is known for its adaptability and scalability, making it suitable for a variety of applications. Furthermore, the algorithm is less prone to getting stuck in local optima compared to traditional optimization methods.

However, there are also cons associated with the discretized particle swarm algorithm. One limitation is its sensitivity to parameter settings, and the performance of the algorithm may heavily depend on the tuning of these parameters. Additionally, PSO may struggle with highly complex and dynamic optimization problems, as its search strategy is based on predefined rules and may not adapt well to changing environments. Another consideration is the risk of premature convergence, where the algorithm settles for suboptimal solutions before exploring the entire search space thoroughly. Despite these drawbacks, the discretized particle swarm algorithm remains a popular choice for binary optimization tasks, and ongoing research aims to address its limitations and enhance its performance in various practical scenarios.

3.5 Binary Fire Fly Algorithm

The Firefly Algorithm (FA) is a nature-inspired optimization algorithm developed by Xin-She Yang in 2008, drawing inspiration from the flashing behavior of fireflies. This metaheuristic optimization algorithm is particularly adept at solving complex optimization problems, especially those with multiple objectives and non-linear constraints.

The fundamental components of the Firefly Algorithm include the concept of firefly attraction, where fireflies are drawn to each other based on their brightness. Brighter fireflies are more attractive, so other fireflies are attracted to them and move towards them. The movement of fireflies is determined by their attractiveness and the distance between them, with the algorithm adjusting positions accordingly. The light intensity of a firefly correlates with its objective function value, and this value is updated iteratively until convergence.

In the binary version of the Firefly Algorithm (BFFA), the solution is encoded in binary form, with each bit representing a decision variable. The distance metric between fireflies is calculated based on the Hamming distance, measuring the number of differing bits between two binary strings. Attraction and movement are adapted for binary variables, ensuring the generation of valid binary solutions.

The return-based cost version of the Firefly Algorithm (Rc-BFFA) presented by Zahng and his colleagues in [6], introduces the concept of return-based cost to enhance exploration and exploitation. This extension considers not only the current objective function value but also the previous performance of a firefly. Fireflies with a history of contributing to better solutions are assigned a lower return-based cost, encouraging them to explore new regions. The algorithm dynamically adapts firefly movement based on this return-based cost, striking a balance between exploring new areas and exploiting known promising regions.

In summary, the Firefly Algorithm, stands out as a flexible and effective optimization approach. Its adaptability to binary decision variables and incorporation of return-based cost make it applicable to a diverse range of optimization problems.

3.5.1 Procedure

1. Initialization: Initialize the population of fireflies with binary solutions. Assign initial fitness values to each firefly based on the objective function.
2. Determine Return-Based Cost: Calculate the return-based cost for each firefly considering historical performance.
3. Attraction and Movement: Determine attractiveness based on current fitness and return-based cost. Adjust movement towards brighter neighbors, ensuring valid binary solutions.
4. Update Fitness Values: Evaluate fitness based on updated positions after movement. Update return-based cost for each firefly to consider past contributions. update the jump probability.
5. Convergence Check: Check if reached maximum number of iterations.
6. Repeat: If convergence criteria are not met, repeat steps 2-4 until convergence.
7. Output: Return the best solution found by the fireflies based on the fitness.

The process of RC-BFFA is also illustrated in Fig.3.7.

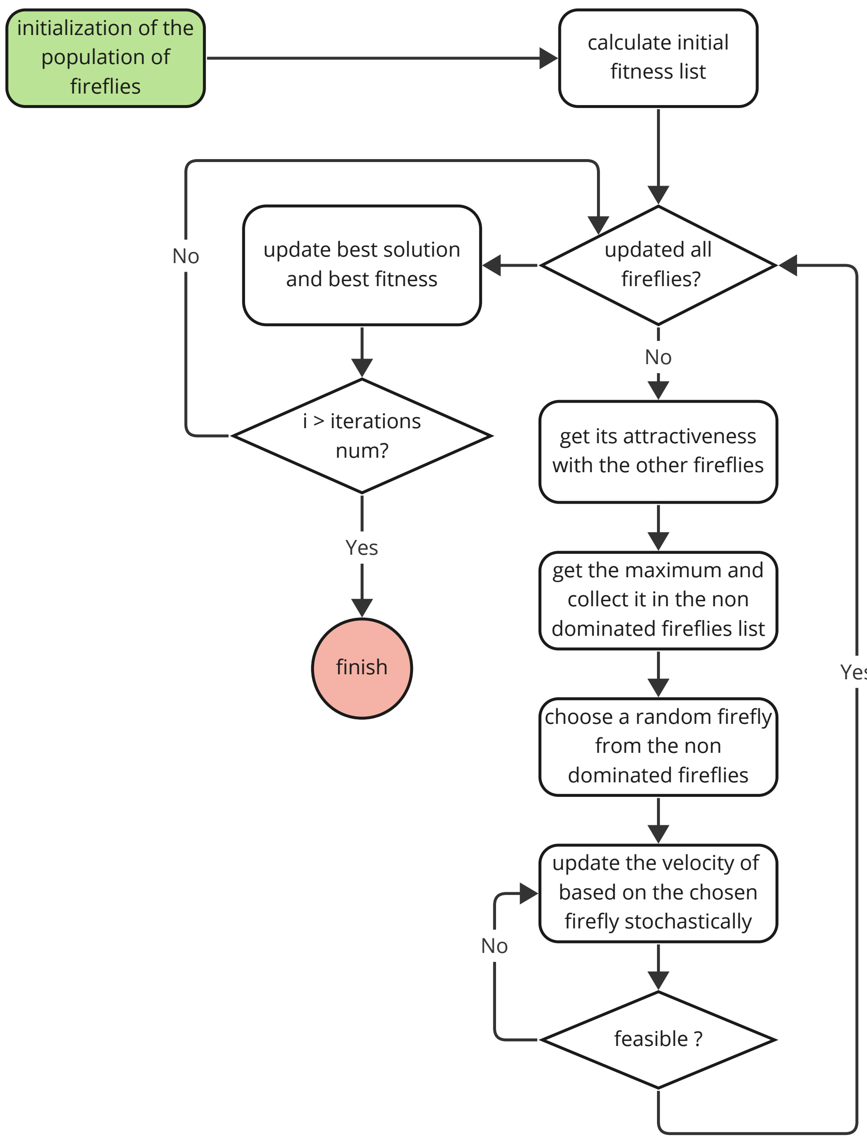
initialization of the population of fireflies
calculate initial fitness list
updated all fireflies?
update best solution and best fitness
No
i > iterations num?
Yes
finish
No
get its attractiveness with the other fireflies
get the maximum and collect it in the non dominated fireflies list
choose a random firefly from the non dominated fireflies
update the velocity of based on the chosen firefly stochastically
No
feasible ?
Yes

Figure 3.7: Flow of the Binary Firefly Algorithm

3.5.2 Pros and Cons

The Return Cost Binary Firefly Algorithm possesses several strengths and weaknesses that can impact its performance across various optimization problems.

The algorithm is adept at global optimization, efficiently exploring solution spaces to identify optimal or near-optimal solutions. Its ability to handle binary decision variables makes it versatile, allowing it to be applied to a broad spectrum of optimization tasks where variables are binary in nature.

A significant advantage of the algorithm lies in its ability to strike a balance between exploration and exploitation. The introduction of return-based cost encourages fireflies to explore new regions while leveraging past experiences to exploit known promising areas. This balanced approach can contribute to the algorithm's effectiveness in navigating complex solution spaces.

The Firefly Algorithm, demonstrates robustness in handling noisy environments. It can perform reasonably well in scenarios where the objective function exhibits stochastic behavior or is subject to random variations.

One notable drawback of the algorithm is its sensitivity to parameter choices. Achieving optimal performance often requires fine-tuning of parameters, which can be a challenging and time-consuming task. The return based cost version of the (BFFA) algorithm solves this problem.

In terms of convergence speed, the Firefly Algorithm might exhibit slower convergence rates compared to some other optimization algorithms, particularly in high-dimensional solution spaces. This characteristic can impact its practicality for certain time-sensitive applications.

While the algorithm excels in global optimization, it may face challenges in thoroughly exploring complex solution spaces, particularly when hindered by the presence of local optima. The algorithm's effectiveness is problem-specific, and its performance can vary based on the nature of the optimization task.

Additionally, the algorithm's memory requirements can be significant, especially for large populations or high-dimensional problems. This memory intensity could pose limitations in resource-constrained environments, affecting its scalability to certain applications.

Chapter 4

Results

4.1 Initial Scenario

We implemented a code that initializes two lists of car objects one for the main road and one for the ramp cars, in order to use in the simulation of our algorithm, the car class takes values like seen in Tab.4.1.

Parameter	Value
random_pos_lower	$5m$
random_pos_upper	$8m$
cars_main_num	$rand(5, 10)$
cars_ramp_num	$rand(5, 20)$
solution_size	10
initial_main_p	$decision\_position$
initial_ramp_p	$decision\_position\_random.randint(10, 30)$
initial_main_v	$16.67m/s$
initial_ramp_v	$13.888m/s$
initial_main_a	$3m/s^2$
initial_ramp_a	$0m/s^2$

Table 4.1: Parameters for Initializing Cars Info

Furthermore, we implemented a code that can generate different solutions, for each solution we need to check if it is feasible and follows the constraints or not, this check should be done at each time step the car spends in the acceleration zone, where our algorithm will be implemented in real life. We achieved that by implementing a cruise control algorithm for the generated car sequence, it evaluates the position and velocity and acceleration at each time step, the sampling period is referred as 'delta_time' in Tab.4.3. the constraints used is shown in 4.2, and the parameters for cruse control is shown in Tab.4.3.

Constraint	Minimum Value	Maximum Value
v_main	$2.777m/s$	$33.33m/s$
v_ramp	$1.388m/s$	$33.33m/s$
a_main	$-20m/s^2$	$20m/s$
a_ramp	$-20m/s^2$	$20m/s^2$

Table 4.2: Constraints

Parameter	Value
alpha	1
beta	1.2
gamma	0.5
desired_distance_bet_cars	6 m
decision_position	40 m
merging_position	140 m
delta_time	0.01 seconds

Table 4.3: Parameters for Cruise Control

4.1.1 Scalability and dynamic code

In our code, we can change the number of cars on the main road and the number of ramp cars, and we also can change the number of cars in the solution array, in addition, the initial positions of all cars are semi-random, where we generate a random number for the position added to the previous position of the previous cars.

You can change a lot of the parameters mentioned before and get new results.

4.2 Implementation of Simulated Annealing

We implemented this Python script to address the optimization of vehicle merging behavior on a highway ramp using a Simulated Annealing (SA) approach. The script begins by importing the necessary modules and setting critical constraints that define the problem, including speed and acceleration limits for both main and ramp vehicles.

To generate a suitable sequence of vehicles for merging, we implemented the 'generate_solution' function, which not only returns the sequence itself but also calculates distances to the merging point, aiding in the evaluation process. The 'check_feasibility' function is crucial for verifying that the generated solution complies with the predefined constraints.

The core of the optimization process lies in the Simulated Annealing (SA) loop, where we applied the SA algorithm iteratively. We fine-tuned parameters such as the initial temperature, cooling rate, number of iterations, and the final temperature. Each iteration involves initializing car configurations, generating new solutions, and conducting feasibility checks. The SA algorithm determines whether the new solution is accepted based on objective function values, temperature, and a probabilistic criterion. The best solution is consistently updated, and the optimization process concludes when the temperature reaches the specified final value.

In this implementation, the algorithms keeps randomizing unitill it finds a feasible new sequence.

4.2.1 Testing different cases

Case 1: Linear change in temperature

After running the simulated Annealing algorithm, with the parameters in 4.1 we get the following:
last solution: [0, 1, 1, 1, 1, 1, 0, 0, 0, 0]
last Objective function value (rounded): 0.949
Best solution : [0, 1, 1, 1, 1, 1, 0, 0, 0, 0]
Best Objective function value (rounded): 0.949.

The development of best solution of the SA algorithm versus temperature is shown in Fig.4.2.

The Objective function value is relatively high, Note that in our problem we want to maximize the total objective function and we have normalized the objective function so its highest value is 1.

Parameter	Value
initial_temperature	1000.0
cooling_rate	5
num_iterations	1000
final_temperature	0.05
w_1	0.5
linear	True

Figure 4.1: Simulated Annealing Parameters

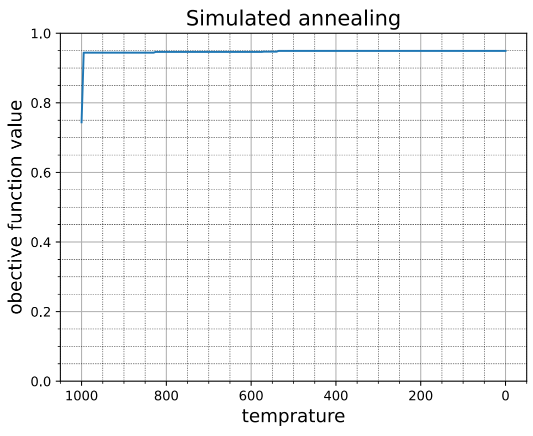

Figure 4.2: Objective function value vs temperature.

Case 2: Geometric change in temperature

After running the simulated Annealing algorithm, with the parameters in 4.3 we get the following:
last solution: [0, 0, 1, 1, 0, 1, 1, 0, 0, 1]
last Objective function value (rounded): 0.942
Best solution : [1, 0, 0, 1, 0, 1, 0, 1, 0, 1]
Best Objective function value: 0.944.

The development of best solution of the SA algorithm versus temperature is shown in Fig.4.4.

Parameter	Value
initial_temperature	1000.0
cooling_rate	0.8
num_iterations	1000
final_temperature	0.05
w_1	0.2
linear	False

Figure 4.3: Simulated Annealing Parameters

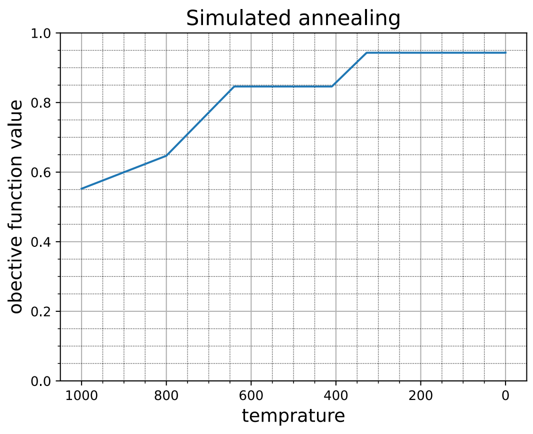

Figure 4.4: Objective function value vs temperature with simulated annealing.

Case 3:

After running the simulated Annealing algorithm, with the parameters in 4.5 we get the following:
last solution: [0, 0, 1, 1, 0, 1, 1, 1, 0, 0]
last Objective function value (rounded): 0.936
Best solution : [1, 1, 1, 0, 1, 1, 0, 0, 0, 0]
Best Objective function value (rounded): 0.954.

The development of best solution of the SA algorithm versus temperature is shown in Fig.4.6.

Parameter	Value
initial_temperature	500.0
cooling_rate	0.5
num_iterations	1000
final_temperature	0.05
w_1	0.5
linear	True

Figure 4.5: Simulated Annealing Parameters

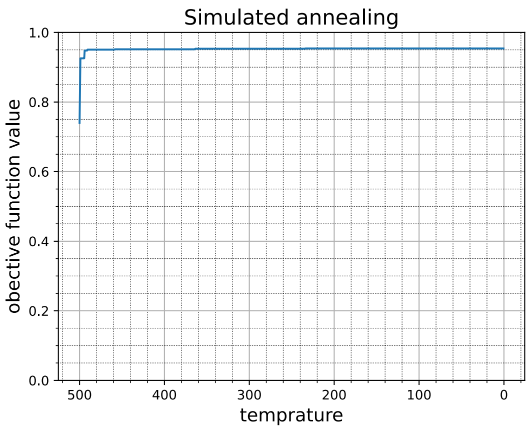

Figure 4.6: Objective function value vs temperature with simulated annealing.

4.3 Implementation of Genetic algorithm

The implemented genetic algorithm functions are designed to address a combinatorial optimization problem associated with car sequencing. The primary genetic algorithm function serves as the orchestrator, guiding the evolution of potential solutions through a series of steps. This includes the initialization of a population with feasible solutions, iterative application of selection, crossover, and mutation operations, and the continual tracking of the best solution throughout generations. The incorporation of elitism ensures that the most promising individuals persist across iterations, contributing to the overall efficiency of the algorithm.

Supporting functions play crucial roles in this optimization process. The initialization function generates an initial population, emphasizing the importance of feasibility by incorporating checks to ensure that only valid solutions are considered. The fitness calculation function assesses the performance of each individual in the population based on a specified fitness function. Crossover and mutation functions handle the creation of new offspring and the introduction of random changes, respectively. Notably, the code demonstrates a keen awareness of the feasibility of solutions, underscoring the significance of valid solutions in the evolutionary process.

Overall, the success of the genetic algorithm hinges on the effective collaboration of these functions, each tailored to address specific facets of the car sequencing problem. The algorithm's strength lies in its ability to balance exploration and exploitation through selection mechanisms, while the supporting functions contribute to the diversity and quality of the evolving population, ultimately working towards finding optimal or near-optimal solutions.

In the implementation, if any new individual has a non feasible solution then the algorithm keeps looping till it is feasible.

4.3.1 Values to Change for Different Cases

1. Population Size: The number of individuals in each generation. Larger populations may enhance exploration but also increase computation time.
2. Crossover Rate: The probability that crossover will be applied to a pair of individuals. Controls the balance between exploration and exploitation.
3. Mutation Rate: The probability that a gene in an individual will be mutated. Regulates the exploration of the solution space.

4.3.2 Testing different cases

Case 1: High elites ratio

The algorithm is tested with the parameters in Tab.4.7, and the following were obtained:
Best GA solution: $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$
Best GA fitness (rounded): 0.888
The best solution over generations is shown in Fig.4.8.

Parameter	Value
population_size	10
generation_size	100
elites_ratio	0.7
crossover_ratio	0.2
mutation_ratio	0.1
w_1	0.2

Figure 4.7: Genetic Algorithm parameters

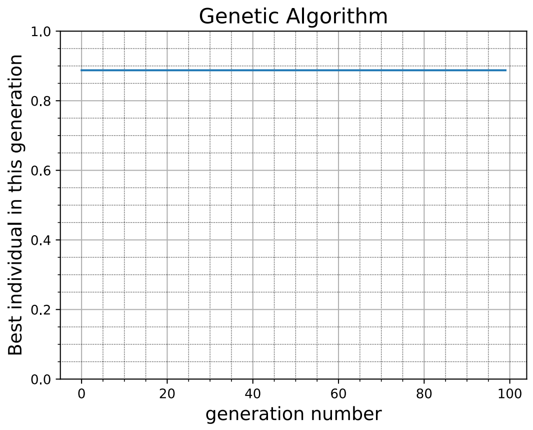

Figure 4.8: Best chromosome vs generation.

Case 2: High cross-over ratio The algorithm is tested with the parameters in Tab.4.9, and the following were obtained:
Best GA solution: $[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$
Best GA fitness (rounded): 0.908

The best solution over generations is shown in Fig.4.10.

Parameter	Value
population_size	10
generation_size	100
elites_ratio	0.1
crossover_ratio	0.8
mutation_ratio	0.1
w_1	0.2

Figure 4.9: Genetic Algorithm parameters

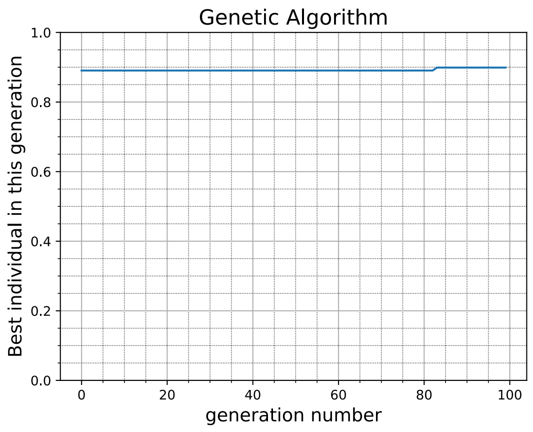

Figure 4.10: Best chromosome vs generation.

Case 3: High mutation ratio

The algorithm is tested with the parameters in Tab.4.11, and the following were obtained:
Best GA solution: $[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
Best GA fitness (rounded): 0.910

The best solution over generations is shown in Fig.4.12.

Parameter	Value
population_size	10
generation_size	100
elites_ratio	0.1
crossover_ratio	0.4
mutation_ratio	0.5
w_1	0.2

Figure 4.11: Genetic Algorithm parameters

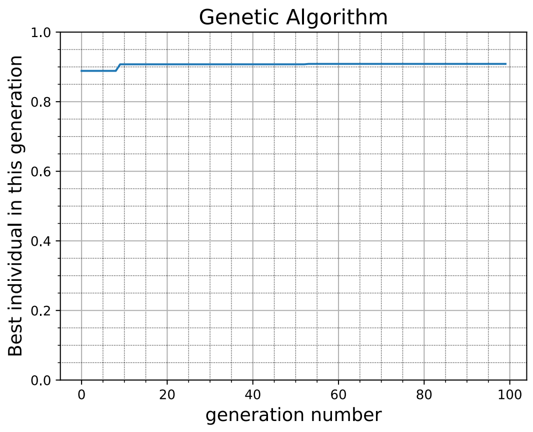

Figure 4.12: Best chromosome vs generation.

4.4 Implementation of Discrete particle swarm

Particle Swarm Optimization (PSO) is an optimization algorithm where particles in a swarm navigate through an n-dimensional search space to find optimal solutions. Each particle has a position ($x_{i,j}$) and velocity ($v_{i,j}$), and the swarm is guided towards promising regions based on past performance. Each particle retains its best position ($x_{i^*,j}$), and neighborhoods of particles share information to enhance optimization. PSO can have local or global neighborhoods, and the algorithm involves equations to update particle positions and velocities at each step. The inertia coefficient (w) controls velocity decay, while weights (pw and nw) influence attraction to individual and neighborhood best positions, respectively. A modification, Discrete PSO (DPSO), was introduced for binary-valued problems by James Kennedy1 and Russell C. Eberhart[7]. In DPSO, the velocity variable indicates the probability of a solution element being 0 or 1. The algorithm adjusts particle values based on a sigmoid function, and a velocity limit (V_{max}) prevents extreme probabilities, typically close to 6.0 according to Jim Pugh and Alcherio Martinoli [8]. DPSO has demonstrated effectiveness in optimizing discrete problems with binary-valued elements.

The equations used for DPSO are: For updating velocity:

$$v_{i,j} = v_{i,j} + c_1 \cdot r_1 \cdot (x_{i^*,j} - x_{i,j}) +_2 \cdot r_2 \cdot (x_{i'^*,j} - x_{i,j}) \tag{4.1}$$

For updating position:

$$x_{i,j} = \begin{cases} 1 & \text{if } r_3 < S(v_{i,j}) \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Here, $v_{i,j}$ represents the velocity of particle i in the j-th dimension, $x_{i,j}$ is the position of particle i in the j-th dimension, $x_{i^*,j}$ is the best position of particle i in the j-th dimension, $x_{i'^*,j}$ is the best position in the neighborhood of particle i, c_1 is the weight for attraction to the previous best location of the current particle, c_2 is the weight for attraction to the previous best location of the particle neighborhood, r_1, r_2 , and r_3 are a uniformly-distributed random variable in $[0, 1]$, and $S(x)$ is the sigmoid function given by $S(x) = \frac{1}{1+e^{-x}}$. The velocity term is constrained to $|v_{i,j}| < V_{\text{max}}$ to limit extreme probabilities, where V_{max} is typically close to 6.0.

In this implementation if a new particle position is not feasible it equates it to the old position.

4.4.1 Testing different Cases

Case 1: Star Tobology and Asynchronous

The algorithm is tested with the parameters in Tab.4.13, and the following were obtained:
Best DPSO solution: $[1, 1, 1, 1, 0, 0, 0, 0, 1, 0]$
Best DPSO fitness (rounded): 0.971

Parameter	Value
Neighborhood Size	30
Max Iterations	100
Synchronous	False
Star Topology	True
c_1 (Cognitive Parameter)	1.49
c_2 (Social Parameter)	1.49
Max Iterations	100
Max Velocity	6.0
w_1	0.7

Figure 4.13: Discrete PSP Parameters

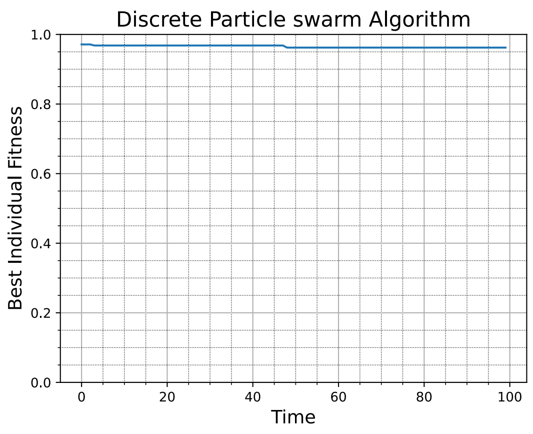

Figure 4.14: Fitness value vs time.

Case 2: Star Tobology and Synchronous

The algorithm is tested with the parameters in Tab.4.15, and the following were obtained:
Best DPSO solution: $[1, 1, 1, 0, 0, 0, 0, 1, 0, 1]$
Best DPSO fitness (rounded): 0.970

Parameter	Value
Neighborhood Size	30
Max Iterations	100
Synchronous	True
Star Topology	True
c_1 (Cognitive Parameter)	1.49
c_2 (Social Parameter)	1.49
Max Iterations	100
Max Velocity	6.0
w_1	0.7

Figure 4.15: Discrete PSP Parameters

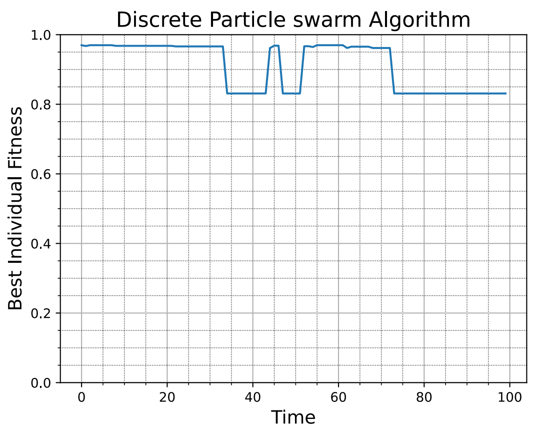

Figure 4.16: Fitness value vs time.

Case 3: Ring Tobology and Asynchronous

The algorithm is tested with the parameters in Tab.4.17, and the following were obtained:
Best DPSO solution: $[1, 0, 1, 1, 0, 0, 0, 0, 1, 1]$
Best DPSO fitness (rounded): 0.968

Parameter	Value
Neighborhood Size	30
Max Iterations	100
Synchronous	False
Star Topology	False
c_1 (Cognitive Parameter)	1.49
c_2 (Social Parameter)	1.49
Max Iterations	100
Max Velocity	6.0
w_1	0.7

Figure 4.17: Discrete PSP Parameters

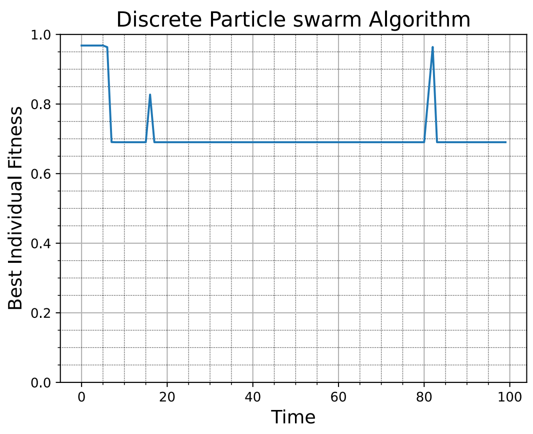

Figure 4.18: Fitness value vs time.

Case 4: Ring Tobology and synchronous

The algorithm is tested with the parameters in Tab.4.19, and the following were obtained:
Best DPSO solution: $[1, 0, 1, 1, 0, 1, 0, 0, 1, 0]$
Best DPSO fitness (rounded): 0.969

Parameter	Value
Neighborhood Size	30
Max Iterations	100
Synchronous	True
Star Topology	False
c_1 (Cognitive Parameter)	1.49
c_2 (Social Parameter)	1.49
Max Iterations	100
Max Velocity	6.0
w_1	0.7

Figure 4.19: Discrete PSP Parameters

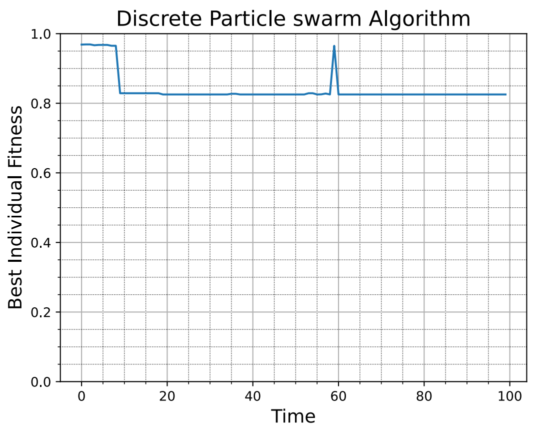

Figure 4.20: Fitness value vs time.

Case 5: High Weight for personal best

The algorithm is tested with the parameters in Tab.4.13, and the following were obtained:
Best DPSO solution: $[1, 1, 0, 1, 1, 0, 0, 0, 0, 1]$
Best DPSO fitness (rounded): 0.970

Parameter	Value
Neighborhood Size	30
Max Iterations	100
Synchronous	False
Star Topology	False
c_1 (Cognitive Parameter)	1.49
c_2 (Social Parameter)	0.7
Max Iterations	100
Max Velocity	6.0
w_1	0.7

Figure 4.21: Discrete PSP Parameters

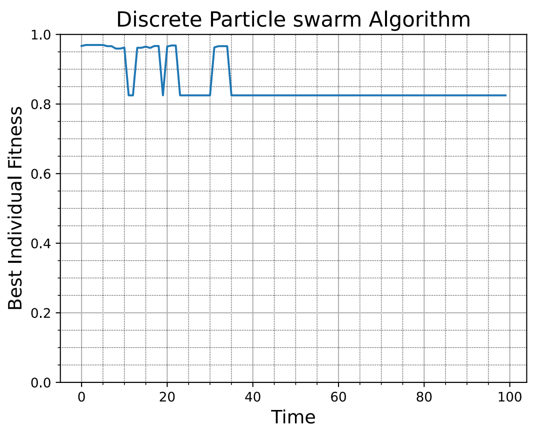

Figure 4.22: Fitness value vs time.

Case 6: High Weight for Global best

The algorithm is tested with the parameters in Tab.4.23, and the following were obtained:
Best DPSO solution: $[0, 1, 1, 1, 1, 0, 1, 0, 0, 0]$
Best DPSO fitness (rounded): 0.972

Parameter	Value
Neighborhood Size	30
Max Iterations	100
Synchronous	False
Star Topology	False
c_1 (Cognitive Parameter)	1.49
c_2 (Social Parameter)	0.7
Minimum Inertia Weight (w_{min})	0.2
Max Iterations	100
Max Velocity	6.0
w_1	0.7

Figure 4.23: Discrete PSP Parameters

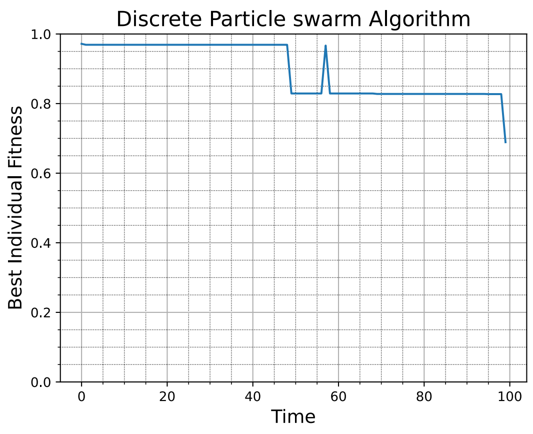

Figure 4.24: Fitness value vs time.

4.5 Implementation of Binary Fire Fly Algorithm

Recognizing limitations in the original FFA's measurement of attractiveness, the proposed Rc-BBFA introduces a novel indicator called return-cost attractiveness. This considers both return and cost values between fireflies, refining the measure of attractiveness. Return and cost functions are defined, contributing to the calculation of attractiveness in Rc-BBFA.

The return function is given by:

$$\text{return}(k, l) = \frac{f(X_l) - f(X_k)}{f_{\text{max}} - f(X_k)}$$

where f_{max} is the maximal objective function value among all fireflies.

The cost function is defined as:

$$\text{cost}(k, l) = \sum_{i=1}^{D} |x_{k,i} - x_{l,i}|$$

The attractiveness calculation in Rc-BBFA is redefined as:

$$\beta(k, l) = 0.5 \times \left(1 + \frac{1}{\text{cost}(k, l) + 1} + \text{return}(k, l)\right)$$

Where X_l represents an attractive one of X_k, which could be any firefly with a better brightness. $\beta(k, l)$ refers to the attractiveness of X_k from X_l, and rand represents a random value in the range of [0, 1].

Taking X_k as an example, we update its position using the proposed operator in [6] as follows:

$$x_{k,j} = \begin{cases} 1 - v_{k,j}, & \sigma > \text{rand}_1 \\ v_{k,j}, & \text{otherwise} \end{cases} \tag{12}$$

$$v_{k,j} = \begin{cases} x_{l,j}, & \beta(k, l) > \text{rand}_2 \\ x_{k,j}, & \text{otherwise} \end{cases} \tag{13}$$

where σ is the jump probability. rand_1 and rand_2 represent two random values in the range of [0, 1]. $V_k = (v_{k,1}, v_{k,2}, \ldots, v_{k,D})$ refers to a temporary vector. $X_l = (x_{l,1}, x_{l,2}, \ldots, x_{l,D})$ represents the attractive firefly with a good attractiveness value of $\beta(k, l)$.

This revised approach to attractiveness calculation reflects a preference for learning from fireflies that offer significant return at a minimal cost, enhancing the algorithm's ability in feature selection within complex solution spaces.

In this implementation, if a non feasible solution is reached, the algorithm keeps repeating until a feasible solution is reached.

4.5.1 Testing different Cases

One the benefits of the Rc-BBFA is the low number of parameters needed to tune the algorithm, so we have only two parameters to tune : the number of fireflies, the number of iterations before convergence as well as the weight of the first objective w_1 .

Case 1:

The algorithm is tested with the parameters in Tab.4.25, and the following were obtained:
Best BFFA solution: $[0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0]$
Best BFFA fitness (rounded): 0.934

Parameter	Value
Population Size	6
Max Iterations	100
w_1	0.3

Figure 4.25: Binary BFFA Parameters

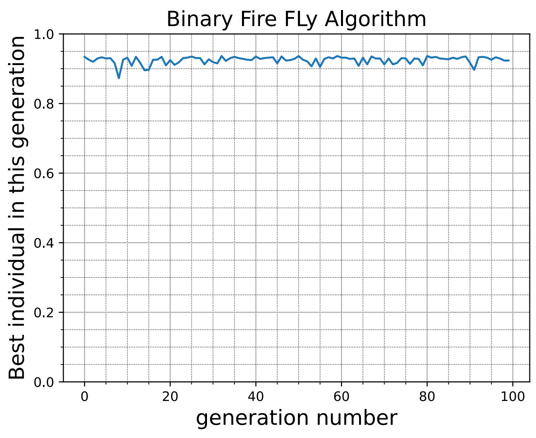

Figure 4.26: Fitness value vs generation.

Case 2:

The algorithm is tested with the parameters in Tab.4.27, and the following were obtained:
Best BFFA solution: $[0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1]$
Best BFFA fitness (rounded): 0.935

Parameter	Value
Population Size	20
Max Iterations	100
w_1	0.3

Figure 4.27: Binary BFFA Parameters

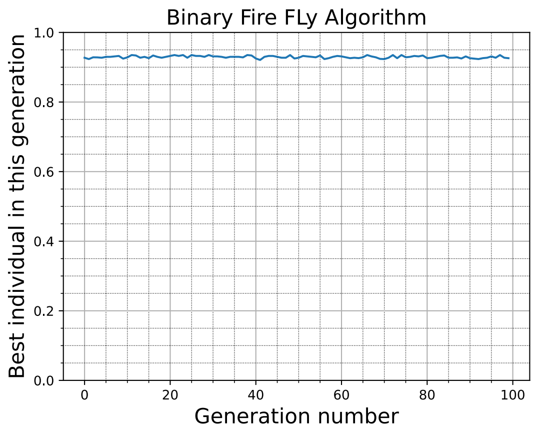

Figure 4.28: Fitness value vs generation.

Case 3:

The algorithm is tested with the parameters in Tab.4.29, and the following were obtained:
Best BFFA solution: $[0, 1, 0, 1, 0, 1, 1, 0, 1, 1]$
Best BFFA fitness (rounded): 0.971

Parameter	Value
Population Size	6
Max Iterations	100
w_1	0.7

Figure 4.29: Binary BFFA Parameters

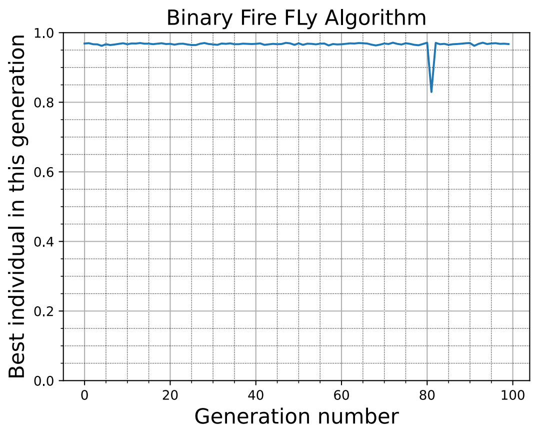

Figure 4.30: Fitness value vs generation.

4.6 Discussion

Comparing two optimization algorithms involves assessing their performance based on various criteria. it's often a good idea to perform multiple experiments on a diverse set of test problems to get a comprehensive understanding of their strengths and weaknesses.

4.6.1 Performance Evaluation

The simulated annealing and genetic algorithm are compared with the same initial scenario which has the same parameters as mentioned in 4.1 but with $w_1 = 0.2$ and parameters in 4.2 and in 4.3 as well. According to the chosen performance evaluation criteria, the values obtained in Tab.4.4 are averaged over 10 runs.

Criteria / Algorithm	SA	GA	DPSO	Rc-BFFA
Best fitness	0.972	0.972	0.973	0.973
Worst fitness	0.968	0.971	0.971	0.972
Avg. running time (s)	2.636	66.537	5.654	180.837
Avg Fitness	0.971	0.972	0.971	0.972
Standard deviation	0.000243	0.000218	0.000297	0.000183
coefficient of variation	0.000251	0.000225	0.000306	0.000188

Table 4.4: Comparison between SA and GA, DPSO and Rc-BFFA

Best and Worst Fitness:

SA and GA exhibit identical best fitness, highlighting their performance in achieving the same optimal solution. DPSO and Rc-BFFA also share the same best fitness, indicating their comparable effectiveness in finding optimal or near-optimal solutions. Rc-BFFA demonstrates the highest worst fitness, suggesting that it may avoid poor solutions more effectively than the other algorithms.

Average Running Time:

SA has the lowest average running time, making it the fastest among the considered algorithms. DPSO is more computationally efficient than Rc-BFFA, which has the highest average running time, indicating that Rc-BFFA may be more time-consuming.

Average Fitness:

SA, GA, and Rc-BFFA exhibit similar average fitness values, suggesting comparable overall performance in reaching solutions close to the optimum. DPSO has a slightly lower average fitness, indicating a potential trade-off between speed and solution quality.

Standard Deviation and Coefficient of Variation:

SA has the smallest standard deviation and coefficient of variation, indicating a high level of consistency and reliability in its performance. GA closely follows SA in terms of stability. DPSO and Rc-BFFA have slightly higher standard deviations and coefficients of variation, suggesting more variability in their performance.

Exploration and Exploitation:

Simulated Annealing (SA): SA, known for its ability to escape local optima, likely emphasizes exploration, and its performance consistency supports its reliability in both exploration and exploitation.

Genetic Algorithm (GA): GA's exploration and exploitation depend on its parameters and crossover/mutation operators. Its competitive results suggest a balanced approach.

Discrete Particle Swarm Optimization (DPSO): DPSO, being a nature-inspired algorithm, is generally good at exploration. However, its slightly lower average fitness might indicate a compromise in exploitation for faster exploration.

Return Cost Binary Firefly Algorithm (Rc-BFFA): Rc-BFFA, with the highest worst fitness, may focus more on exploration to avoid suboptimal solutions. Its higher average running time could be indicative of thorough exploration.

4.6.2 Convergence

The following figure compares the fitness of the (SA), (GA), (DPSO) and the (Rc-BFFA) algorithms over progress (generation) respectively.

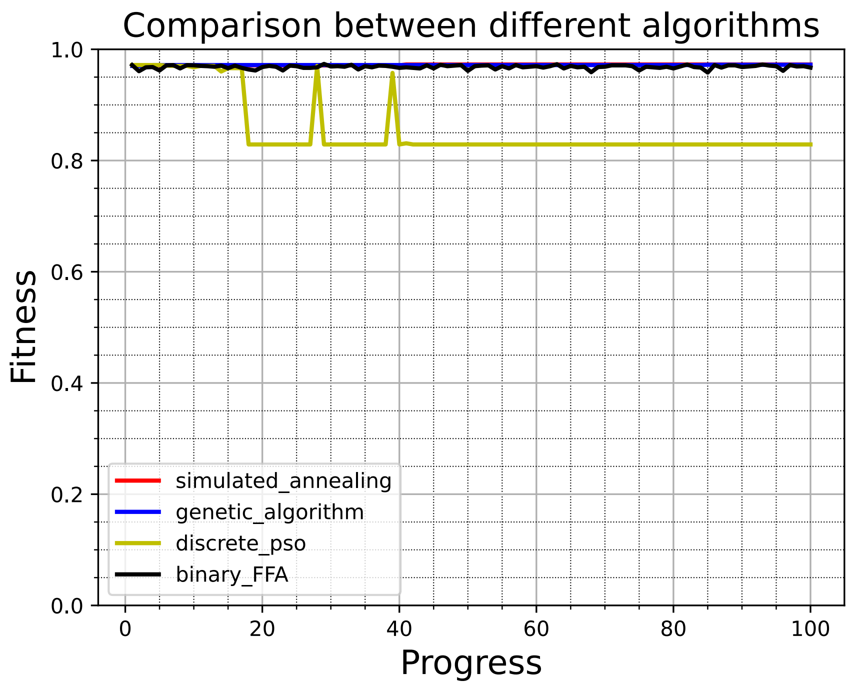

Figure 4.31: Comparison of the fitness between SA, GA, PSO and BFFA.

It can be concluded from Fig.4.31 that (SA) converges faster than the other algorithms so it is better in exploiting, and the (DPSO) shows more exploration even towards the end so it is better in exploration than the other algorithms. While on the other hand the (Rc-BFFA) does not converge and keeps oscillating around local maximum, this is contributed to the fact that the algorithm is heavily stochastic.

4.7 Conclusion

It is concluded that for the presented problem, the performance of the (DPSO) algorithm is better overall in getting better fitness value, and it does not take a lot of running time as well. On the other hand, the fastest result can be obtained using the simulated annealing algorithm since it is trajectory based and not population based. The binary firefly neither has the best running time nor the best fitness value so it is not the best algorithm to choose to to optimize the problem of merging of vehicles on highways.

In summary, each algorithm has its strengths and trade-offs in terms of exploration and exploitation, and the choice depends on the specific requirements of the optimization problem. For the case of vehicle merging having a fast running time is desired. so the best algorithms to choose are the (SA) and the (DPSO).

4.7.1 Future recommendations

With a sufficient computational power, the mentioned algorithms should be tested with higher number of cars as well as test with real data, and conduct experiments on actual vehicles to validate the obtained results, which should be the end goal after a lot of researching. The problem can also be tested with other optimization algorithms like whale optimization or gray wolf after discritization.

Bibliography

[1] Linghui Xu, Jia Lu, Bin Ran, Fang Yang, and Jian Zhang. Cooperative merging strategy for connected vehicles at highway on-ramps. *Journal of Transportation Engineering, Part A: Systems*, 145(6):04019022, 2019.

[2] Jackeline Rios-Torres and Andreas A Malikopoulos. Automated and cooperative vehicle merging at highway on-ramps. *IEEE Transactions on Intelligent Transportation Systems*, 18(4):780–789, 2016.

[3] Sharmila Devi Kumaravel, Andreas A Malikopoulos, and Ramakalyan Ayyagari. Decentralized cooperative merging of platoons of connected and automated vehicles at highway on-ramps. In *2021 American Control Conference (ACC)*, pages 2055–2060. IEEE, 2021.

[4] Jishiyu Ding, Li Li, Huei Peng, and Yi Zhang. A rule-based cooperative merging strategy for connected and automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(8):3436–3446, 2019.

[5] Haigen Min, Yukun Fang, Xia Wu, Guoyuan Wu, and Xiangmo Zhao. On-ramp merging strategy for connected and automated vehicles based on complete information static game. *Journal of Traffic and Transportation Engineering (English Edition)*, 8(4):582–595, 2021.

[6] Yong Zhang, Xian-fang Song, and Dun-wei Gong. A return-cost-based binary firefly algorithm for feature selection. 418-419:561–574.

[7] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 5, pages 4104–4108. IEEE.

[8] Jim Pugh and Alcherio Martinoli. Discrete multi-valued particle swarm optimization.