Optimization of Vehicle Merging For Connected Vehicles

Dina Mohamed[1], Marwa Lotfy[1], Ahmed Mohamed Sleem[1], Nada Tamer[1]

***Abstract*—This paper investigates advanced optimization techniques, namely Simulated Annealing (SA), Genetic Algorithm (GA), Discrete Particle Swarm Optimization (DPSO), and Return Cost Binary Firefly Algorithm (Rc-BFFA), to enhance highway on-ramp merging control. The study concludes that DPSO performs well in terms of fitness and efficiency, while SA stands out for its speed. The Binary Firefly Algorithm (BFFA) is found to be less optimal. The recommended choices for swift execution in vehicle merging are Simulated Annealing (SA) and Discrete Particle Swarm Optimization (DPSO). Future research should involve real-world testing with a higher number of vehicles and explore alternative optimization algorithms like whale optimization or gray wolf, particularly after discretization, to broaden the scope of potential solutions.**

***Index Terms*—highway on-ramp merging, optimization techniques, Simulated Annealing (SA), Genetic Algorithm (GA), Discrete Particle Swarm Optimization (DPSO), Return Cost Binary Firefly Algorithm (Rc-BFFA),**

I. Literature Review

Several studies focus on optimizing merging for connected vehicles. Xu et al. [1] propose a genetic algorithm-based strategy, considering on-ramp inflow and utilizing wireless communication to enhance traffic efficiency and reduce fuel consumption. Simulation results validate the strategy's effectiveness, suggesting potential areas for future research. Rios et al. [2] introduce an optimization framework for coordinating Connected and Automated Vehicles (CAVs) at merging roadways, demonstrating significant reductions in fuel consumption and travel time through simulations. Research in this domain primarily concentrates on intersections and merging highways.

II. Problem Formulation

A. 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 1, while minimizing the overall delay for vehicles already on the main road eq:2, where $r1$ is the number of vehicles that are left for the next merging decision.

$$f_1 = \frac{r - r_1}{r} \tag{1}$$

$$f_2 = \frac{-\sum_{i=1}^{m} t_i + \sum_{i=1}^{m} \frac{l_i}{v_{\min}}}{-\sum_{i=1}^{m} \frac{l_i}{v_{\max}} + \sum_{i=1}^{m} \frac{l_i}{v_{\min}}} \tag{2}$$

The final objective function is:

$$\max F = w_1 * f_1 + w_2 * f_2 \tag{3}$$

B. Constraints

The objective function in 3 is subject to the following constraints:

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

a) : 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 law 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.

$$a_f = \alpha \cdot a_l + \beta \cdot (v_f - v_l) + \gamma \cdot (s - s_d) \tag{4}$$

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

C. 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.1.

*This work was not supported by any organization

[1]Mechatronics Department - Faculty of Engineering and Materials Science - German University in Cairo, Egypt
dina67724@gmail.com, lotfymarwa410@gmail.com, ahmedsleem.mail@gmail.com, nadatameer@outlook.com

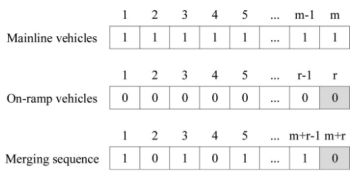

Fig. 1: Solution Array. Source: Xu (2019) in [1].

D. Assumptions

The streamlined approach to controlled vehicle merging involves key assumptions. The merging point is fixed upstream of the exit, maintaining consistency. Prohibiting lane changes within the control zone simplifies the scenario to the rightmost main road lane and a single on-ramp lane. Longitudinal control takes precedence over lateral control, focusing solely on vehicle acceleration. Assumptions include Cooperative Adaptive Cruise Control (CACC) vehicles with Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication, enabling real-time merging instructions. Merging is segmented into intervals with assumed constant acceleration, facilitating modeling and analysis. These assumptions establish a controlled and predictable merging environment, crucial for developing and optimizing cooperative merging strategies.

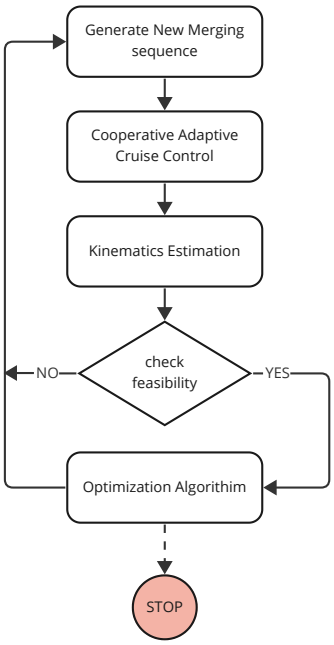

Fig. 2: Proposed merging framework

III. Implementation

A. 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.I.

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.**??**. the constraints used is shown in II. The cruise control system is defined by specific parameters, each playing a crucial role in its modeling and optimization. The values assigned to these parameters are as follows: alpha is set to 1, beta to 1.2, and gamma to 0.5. The desired distance between cars is established at 6 meters, with the decision position positioned at 40 meters and the merging position at 140 meters. The time interval, delta time, is defined as 0.01 seconds. These parameter values collectively govern the behavior of the cruise control system, influencing factors such as acceleration, decision-making, and merging positions.

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 I: Parameters for Initializing Cars Info

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 II: Constraints

IV. Trajectory based optimization technique

A. Simulated Annealing (SA)

Simulated Annealing (SA) is a heuristic optimization algorithm inspired by metallurgical annealing. Operating on an initial solution, it explores neighboring solutions probabilistically, accepting worse ones as it cools based on a temperature schedule. SA's strength lies in effective global searches and adaptability to diverse problems, making it versatile for tasks like the traveling salesman problem. While it doesn't guarantee optimality and requires careful parameter tuning, SA is valuable and versatile across various domains, offering an efficient approach for finding near-optimal solutions in complex search spaces.

The process of SA is also illustrated in Fig.3.

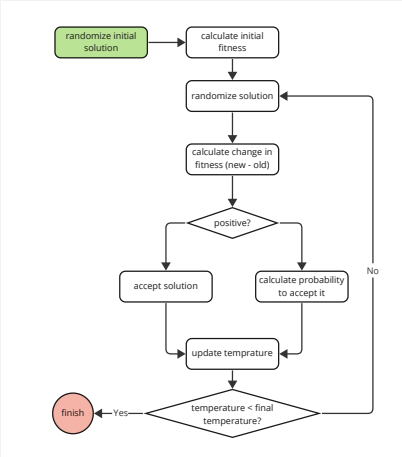

Fig. 3: Flow of the Simulated Annealing Algorithm

1) Testing one case:

a) Linear change in temperature: After running the simulated Annealing algorithm, with the parameters in III 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. 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

TABLE III: Simulated Annealing Parameters

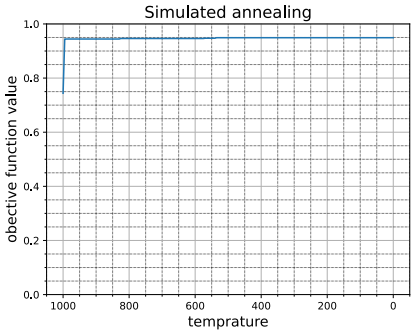

Fig. 4: Objective function value vs temperature.

V. Population based optimization technique

A. GA

A genetic algorithm (GA), part of evolutionary algorithms, mimics natural selection to find approximate solutions for optimization and search problems. It starts by initializing a population, evaluating fitness, selecting parents, recombining genetic information, introducing mutations for diversity, and replacing the old population. The process iterates for specified generations. GAs excel in global searches, adaptability, and parallelism but lack optimality guarantees, are sensitive to parameters, may incur computational costs for complex problems, and struggle with constraints, often necessitating additional techniques for enhanced performance. The process of (GA) is also illustrated in Fig.5.

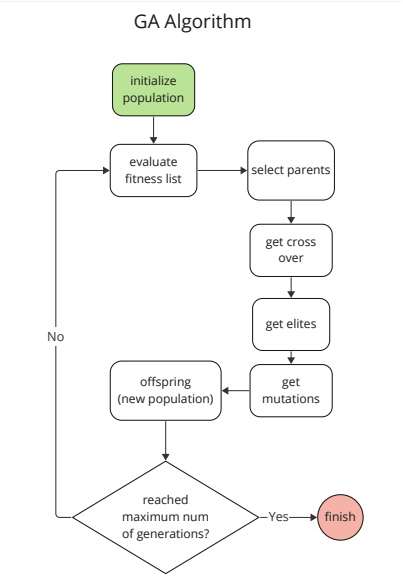

Fig. 5: Flow of the Genetic algorithm

1) Testing one cases:

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

The best solution over generations is shown in Fig.6.

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

TABLE IV: Genetic Algorithm parameters

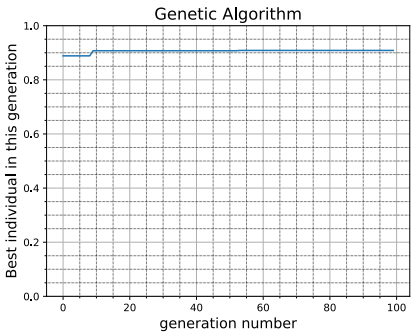

Fig. 6: Best chromosome vs generation.

B. DPSO

The Discrete Particle Swarm Optimization (DPSO) is a nature-inspired optimization algorithm based on the social behavior of bird flocks and fish schools. Originally designed for continuous optimization, it can be adapted for discrete problems such as binary optimization, as seen in the context of optimizing highway ramp car merging. The algorithm initializes a swarm of particles, each representing a potential solution encoded as a binary string. The updating mechanism is adjusted for the binary nature of the problem, ensuring effective exploration of the solution space. The DPSO procedure involves initialization, fitness evaluation, personal and global best updates, velocity and position updates, convergence checks, and solution extraction. Pros include simplicity, ease of implementation, and adaptability, while cons involve sensitivity to parameter settings and potential challenges with dynamic problems. Despite limitations, DPSO is favored for binary optimization tasks, with ongoing research aimed at addressing its drawbacks and enhancing performance.

The process of DPSO is also illustrated in Fig.7.

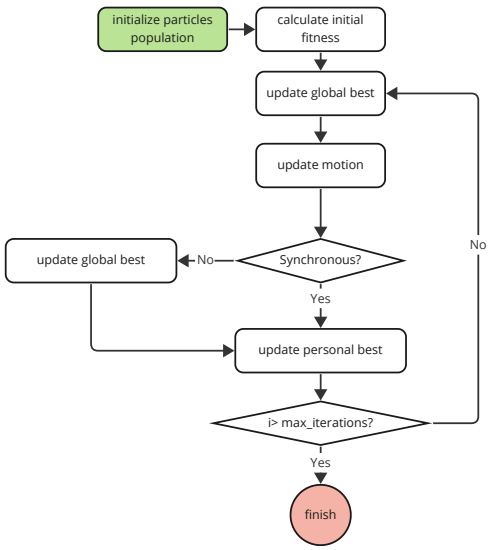

Fig. 7: Flow of the Discrete particle swarm algorithm

1) 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 [3]. 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 [4]. DPSO has demonstrated effectiveness in optimizing discrete problems with binary-valued elements.

The equations used for DPSO are: For updating velocity:

$$v_{i,j} = v_{i,j} + c_1 \cdot r_1 \cdot (x_{i^*,j} - x_{i,j}) + _2 \cdot r_2 \cdot (x_{i'^*,j} - x_{i,j}) \quad (5)$$

For updating position:

$$x_{i,j} = \begin{cases} 1 & \text{if } r_3 < S(v_{i,j}) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

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 $|v_{i,j}| < V_{\max}$ 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.

C. Rc-BFFA

The Binary Firefly Algorithm (BFFA) is a nature-inspired optimization method derived from the Firefly Algorithm (FA), which mimics the flashing behavior of fireflies. Developed for complex optimization problems, FA is adapted for binary decision variables in BFFA. The algorithm involves initializing a population, calculating return-based cost, determining attractiveness, adjusting movement, updating fitness values, and checking convergence. Notably, the return-based cost enhances exploration-exploitation balance. BFFA excels in global optimization, particularly with binary variables, and demonstrates robustness in noisy environments. However, it is sensitive to parameter choices, may have slower convergence, and could face challenges in thoroughly exploring complex solution spaces. Memory requirements, especially for large populations, might limit scalability in resource-constrained environments. Despite these considerations, BFFA remains

a flexible and effective optimization approach with diverse applications.

The process of RC-BFFA is also illustrated in Fig.8.

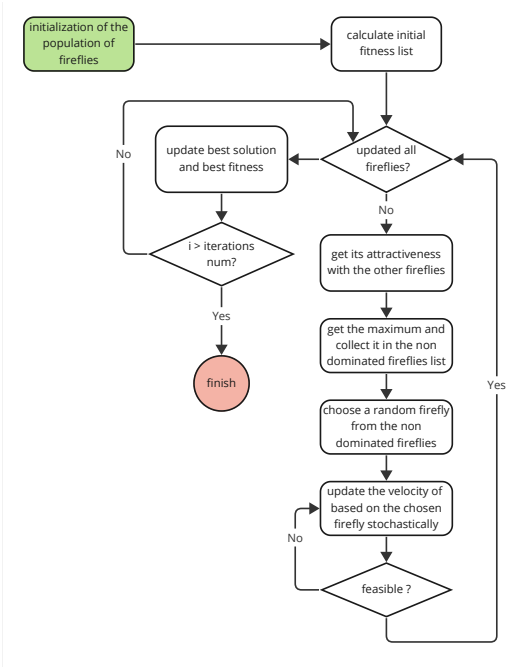

Fig. 8: Flow of the Binary Firefly Algorithm

VI. 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:

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

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 [5] as follows:

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

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

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.

VII. Results and 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.

A. Performance Evaluation

The simulated annealing and genetic algorithm are compared with the same initial scenario which has the same parameters as mentioned in I but with $w_1 = 0.2$ and parameters in II and in **??** as well. According to the chosen performance evaluation criteria, the values obtained in Tab.V 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 V: Comparison between SA and GA, DPSO and Rc-BFFA

1) 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.

2) 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.

3) 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.

4) 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.

5) Exploration and Exploitation::

a) 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.

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

c) 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.

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

B. Convergence

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

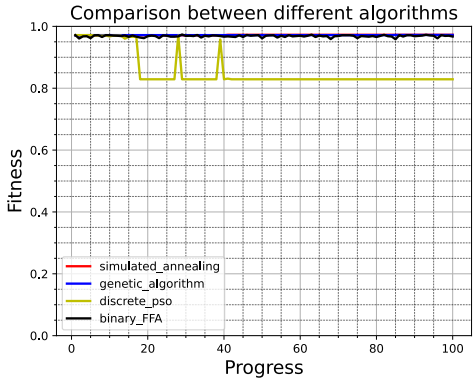

Fig. 9: Comparison of the fitness between SA, GA, PSO and BFFA.

It can be concluded from Fig.9 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.

VIII. Conclusion and Future Recommendations

A. 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).

B. 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.

References

[1] L. Xu, J. Lu, B. Ran, F. Yang, and J. Zhang, "Cooperative merging strategy for connected vehicles at highway on-ramps," *Journal of Transportation Engineering, Part A: Systems*, vol. 145, no. 6, p. 04019022, 2019.

[2] J. Rios-Torres and A. A. Malikopoulos, "Automated and cooperative vehicle merging at highway on-ramps," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 4, pp. 780–789, 2016.

[3] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5. IEEE, pp. 4104–4108. [Online]. Available: http://ieeexplore.ieee.org/document/637339/

[4] J. Pugh and A. Martinoli, "Discrete multi-valued particle swarm optimization."

[5] Y. Zhang, X.-f. Song, and D.-w. Gong, "A return-cost-based binary firefly algorithm for feature selection," vol. 418-419, pp. 561–574. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025516314098