Optimal Control for Connected and Autonomous Vehicles at Signal-Free Intersection

Amr Abdrabo[1] (43-15882), Youssef Abdelhady[1] (40-0893), Bishoy Essam[1] (43-15419), Paula Amir[1] (43-7652), Michel Magdy[1] (43-5944)

***Abstract*—In the last decade, there is a great effort made to tackle the problem of traffic due to an increase of the drivers. This problem is an optimization one. In Optimization problem is mainly focusing on finding optimal feasible solutions from all possible solutions. In this problem we assumed that all autonomous vehicles are connected together in one network and controlled by control unit so we can signalized them inside the controlled zone. This paper proposes four different metaheuristic algorithms which are Simulated Anneling (SA), Genetic Algorithm (GA), Particle Swarm optimization (PSO) and last but not least PathFinder algorithm (PFA) to help find these feasible solutions. These algorithms have much in common but also have some differences. We make comparison among these four algorithms from perspective of average execution time, standard deviation and most optimum results. After some trials and experiments, we find that the fastest algorithm is SA, while the best standard deviation score and most optimum results achieved by GA.**

***Index Terms*—Keywords—Connected and automated vehicles, multi-agent, cooperative, signal timing optimization, speed harmonization, smart cities, smart traffic**

I. Introduction

In the last two decades there are huge efforts made to solve the traffic problem because of increasing the number of drivers. Traffic issues is a serious challenge for governments because safety is and will remain the first priority. As people spend so much time driving, causing more chaos. The development in the smart technology plays an important role in controlling the traffic. The new technology which is the connected vehicle technology helps traffic control with methods not only to collect real-time vehicle information and make online decisions, but also to coordinate their action and make decision cooperatively. Traffic signals help to control and coordinate traffic in urban street networks and can make a significant in network performance. As on major roads, signalized intersection may cause about high vehicular delay. This technology needs to collect online information from the the connected vehicles which helps the signals controller to define vehicle arrivals more accurately then finding more efficient signal plans. And then sharing this signal plans with other signal controllers yields network-wide optimal operations. Providing signal intersection with the consequently speeds, time arrivals may helps to improve the performance of the network. We applied these algorithms on this problem for effective performance and results.

First algorithm used is SA. SA first introduced by Kirkpatrick et al. (1983) and Cerny (1985) independently. SA is a stochastic algorithm used to global extreme of object function that has local minimums. SA is based on the analogy the way the crystalline structure of a metal is being cooled with some statistical mechanics. For example, sharp decrease of temperature may lead to non-symmetric structure which make metals weak. SA helps us to get global optima solution to cool metals.

Second algorithm used is GA. GA is proposed first time by John Holland in 1975. GA is also stochastic search algorithm used to get global optimum feasible solution. GA is inspired from Darwin's theory of natural evolution. This algorithm compose in combining solutions (genetic crossover), using local moves (mutations) and reflects the process of natural selection where the fittest individuals are selected for reproduction population with the best solutions (natural selection).

Third algorithm used is PSO. PSO first introduced by Kennedy and Eberhart in 1995, is regarded as one of the evolution algorithms. PSO is also metaheurisitc stochastic search algorithm. PSO draws its inspiration from the behavior of the bird flocking that search for their food. The social hierarchy among the members of a group of animals typically determines how they move. These animals may be forced to choose between following a leader or not following a leader. However, leadership is transient, and just a few people may be aware of food sources, hunting areas, or routes.

Fourth algorithm used is PFA. PFA is also considered as a metaheurisitc search algorithm. This algorithm is based on animal swarms' collective movement and mimics the leadership hierarchy of swarms to select the best food or prey.

II. Literature review

In [1] They proposed a cooperative method for traffic signal optimization and vehicle speed control to improve the transportation efficiency and vehicle fuel economy both in macro traffic level and micro vehicle level. The simulation results showed that the cooperative method can effectively adjust the traffic signal timing according to the real time traffic condition and simultaneously optimal vehicle trajectory/speed profiles, which results in the improved transportation efficiency and vehicle fuel economy. In addition, the cycle length and communication range have significant effects on the performance of the algorithm.

This paper [2] addresses the problem of coordination between connected autonomous vehicles(CAVs) at signal-free intersections by a centralized methodology. A weighted sum of the aggregate energy consumption and traveling time of all CAVs is optimized. The OCP is solved using GPOPS-II

[1]Mechatronics Department - Faculty of Engineering and Materials Science - German University in Cairo, Egypt

which is a general-purpose MATLAB software for solving continuous optimal control problems. The paper provides numerical examples at the end. Also [3]This paper provides a similar study to the previous one.The difference is that this paper introduces multi-lane roads and allows the vehicles to change their direction inside the intersection region. This paper solves the OCP problem using Particle Swarm Optimization (PSO).

The coordination of CAVs travelling through an unsignalized intersection was examined in this research [4] in order to optimize the trade-off between travel duration and fuel consumption. The Intersection Controller (IC) is able to choose to minimize either the fuel consumption or the total travel time depending on the traffic conditions, or even use an intermediate state that combines the best of both worlds.At the end it optimises the trajectories of several vehicles that are projected to arrive at the Control Zone within a time-window using a centralised scheme.

In the context of autonomous vehicle applications, this research [5] has created a priority control algorithm for optimising vehicle movement at crossings. To reduce intersection delays while maintaining traffic safety, the proposed approach employs game theory and trajectory-based conflict point identification.they developed strategies for vehicles in different conditions. Two different mathematical models were developed; one with 100 percentage autonomous vehicles and the other with a mixed traffic of autonomous vehicles and ordinary vehicles.

III. Methodology

A. Problem formulation

There are four double-lane roads meeting at an intersection as shown in Figure 1. The area around the intersection is referred to as Control Zone (CZ) and extends L distance from the intersection boundary, while the center of the intersection is called Merging Zone (MZ). The length of the MZ is denoted by S and assumed to be $S < L$. For simplicity it is assumed that vehicles do not turn so that the only directions of movement allowed are D = {North-South (NS), South-North (SN), East-West (EW), West-East (WE)}. Each Connected Autonomous Vehicle (CAV), i, that is entering the CZ has an arrival time to the CZ, t_i^0, initial velocity, v_i^0 and direction of movement $d_i \in D.t_i^m$ and t_i^f denote the required travel time of CAV i to arrive and exit the MZ respectively. It is considered that when $t_i^0 < t_j^0$, it is also true that $i < j$. N is denoted as the total number of arrivals, i.e, $i \in N = \{1, \dots, N\}$.

1) Decision variables:

- Acceleration profile, $u(t)$ of each vehicle, i, from t_i^0 to $t_i^0 + t_i^m$

2) Objective functions:

$$\text{minimize} \quad \sum_{i=1}^{N} t_i^m \tag{1}$$

$$\text{minimize} \quad \int_{t_i^0}^{t_i^0+t_i^m} u_i^2(t)\,\mathrm{d}t \tag{2}$$

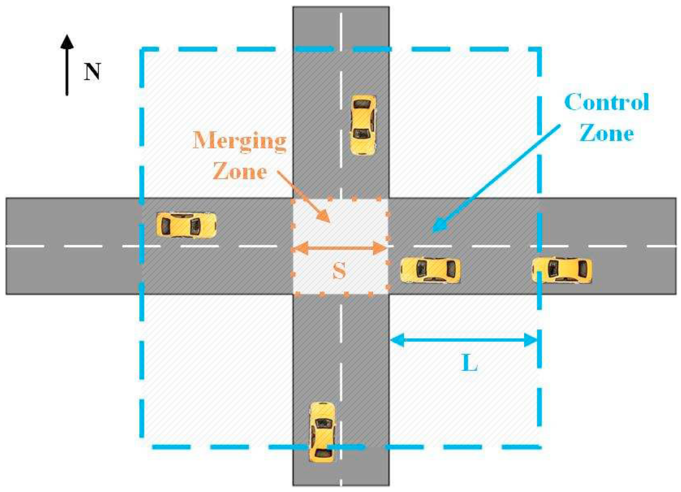


Fig. 1: Scheme of autonomous intersection crossing with connected and autonomous vehicles

where the first objective function minimizes the travel time and the second objective function minimizes the fuel consumption of all the vehicles. Both of the objective functions are subject to (4), (5), (6) and (7).

3) Constrains:

- CAVs follow second order kinematics such that:

$$\begin{aligned} \dot{x}_i(t) &= v_i(t), \quad x_i(t_i^0) = 0 \\ \dot{v}_i(t) &= u_i(t), \quad v_i(t_i^0) = v_i^0 \end{aligned} \tag{3}$$

  where $x_i(t)$, $\dot{v}_i(t)$ and $u_i(t)$ are the position, velocity and acceleration of CAV i at time t with $u_i(t)$ being the control input

- To ensure that the velocity and acceleration are within physical limits, the following constraints are imposed:

$$\begin{aligned} 0 \le v_i(t) \le v_i^{max} \quad & \forall t \in [t_i^0, t_i^0 + t_i^m] \\ u_i^{min} \le u_i(t) \le u^{max} \quad & \forall t \in [t_i^0, t_i^0 + t_i^m] \end{aligned} \tag{4}$$

- Each CAV $i \in N$ arrives at the MZ at a predefined velocity $v_i^m = v^m$ and moves with constant velocity inside the MZ:

$$v_i(t) = v^m, \quad \forall t \in [t_i^0 + t_i^m, t_i^0 + t_i^f] \tag{5}$$

- To ensure rear-end collision avoidance, all vehicles traveling in the same direction of movement are enforced to maintain a safe distance δ at all times such that

$$x_k(t) - x_i(t) \ge \delta. \quad \forall t \in [t_i^0, t_i^0 + t_i^f] \tag{6}$$

  where k denotes the index of the vehicle in front of CAV i in the same direction of movement.

- In order to avoid lateral collisions inside the MZ, the merging time of CAVs i and j, $i < j$ travelling in perpendicular directions (i.e. NS and EW) must adhere to the constraint

$$t_i^0 + t_i^m + \frac{S}{v^m} \leq t_j^0 + t_j^m \tag{7}$$

which ensures that vehicle j enters the MZ only after vehicle i has exited the MZ.

4) Implementation:

- We implemented our problem using Python programming language. In Figure 2, the acceleration trajectory (the solution) is discretized over the distance of the control zone by a constant ΔL. The trajectory would have a constant size of $n = \frac{L}{\Delta L}$, where L is the length of the control zone = 300 m and ΔL is chosen to be equal to 5 m.

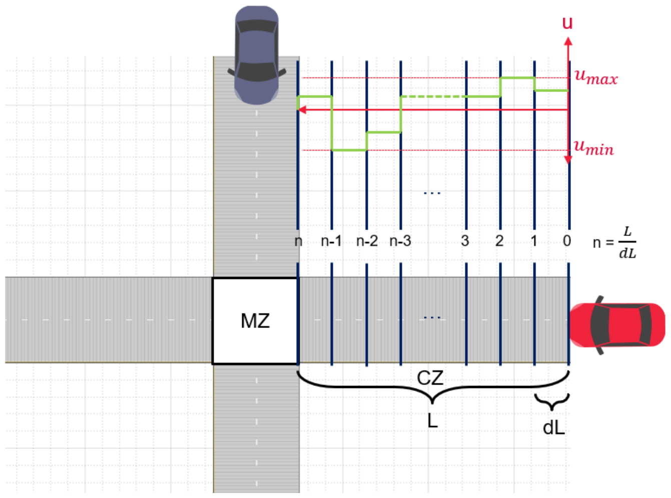


Fig. 2: Discretization of our problem

B. Optimization algorithms

These are the meta-heuristic optimization algorithms which implemented and tested for our problem to find the optimal solution. All of them are implemented using Python programming language.

1) Simulated Annealing (SA):

- SA is a trajectory based optimization algorithm emulating the physical annealing process. It's a simple algorithm where we first initialize a random solution at the initial temperature, then try to find a better while cooling down. We decide if this solution is good or not based on the energy calculated (objective function). In high temperatures, the probability to accept worse solutions is high (exploration), meanwhile, in low temperatures this probability should be low (exploitation). Figure 3 shows the pseudo code of the SA algorithm.

2) Genetic Algorithm (GA):

- GA is a population based optimization algorithm inspired by the genetic evolution. The chromosome here or the individual is considered as the solution and it consists of number of genes (decision variables). This individual

```
- Initialize random solution, best_sol, best_f, T0, Tf, imax
- While STOPPING CRITERIA (Tcur>Tf and i≤imax):
    - Repeat for nT (iterations per temperature):
        - Generate random new feasible solutions:
        sol_new=sol_cur+rand
        - Compute change in energy: ΔE=Δf=f(sol_new) −f(sol_cur)
        - If ΔE<0:
            - Accept better solution: f(sol_new)
        - else if r<p=e^(−ΔE/T) where r is a random number 0~1:
            - Accept solution: sol_cur=sol_new
        - If f(sol_cur) < best_f:
            - Update best reached solution & best_f
    - Update temperature according to cooling schedule.
    - Update iteration counter: i=i+1
```

Fig. 3: Pseudo code for SA algorithm

can be evaluated based on its fitness value (objective function). As we deal now with population algorithm, so we will have multiple of individuals equal to the population size.

- Our population should be updated after each generation (iteration). We select a number of individuals with the highest fitness value (elites) to be survived to the next generation. We also select some individuals as parents to generate new children. This process is called cross-over . We then apply mutation process by selecting some individuals with the lowest fitness value and change their genes. Figure 4 shows the pseudo code for GA algorithm.

```
- Generate initial random population
- Initialize num_elite, num_crossOver, num_mutation
- While STOPPING CRITERIA (i≤imax):
    - Elite members are survived from sol_prev
    - Select parents from sol_prev for crossover them
    - mutate worst members from sol_prev
    - Compute fitness value: fxnew
    - Update iteration counter: i=i+1
```

Fig. 4: Pseudo code for GA algorithm

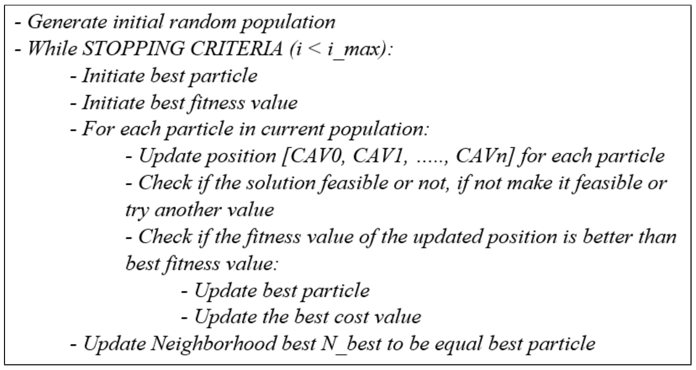
3) Particle Swarm Optimization (PSO):

- PSO is a population based optimization algorithm based on the social behavior exhibited by bird flocks when striving to reach a destination. We consider the position of the particle (the bird) is the solution and we update it through these two equations:

$$v_{i+1} = wv_i + c_1r_1 \left(\text{Pbest}_{\,i} - x_i \right) + c_2r_2 \left(\text{Nbest}_{\,i} - x_i \right) \tag{8}$$

$$x_{i+1} = x_i + v_{i+1} \tag{9}$$

in equation (8) we update the velocity and it has three components. the first one from the left is the inertia component. This component added as the particle cannot suddenly change its direction of motion due to its inertia. The second one is Cognitive Component. Each particle remembers its personal best position and try to adjust its velocity to move towards it. The last component is the social component. Each particle tries to keep up with the swarm leader which is swarm neighborhood best position and adjust its velocity to move towards it. Then in equation (9) we add this velocity to the previous position to calculate the new position. Figure 5 shows the pseudo code for this algorithm.



```
- Generate initial random population
- While STOPPING CRITERIA (i < i_max):
        - Initiate best particle
        - Initiate best fitness value
        - For each particle in current population:
                - Update position [CAV0, CAV1, ....., CAVn] for each particle
                - Check if the solution feasible or not, if not make it feasible or
                try another value
                - Check if the fitness value of the updated position is better than
                best fitness value:
                        - Update best particle
                        - Update the best cost value
        - Update Neighborhood best N_best to be equal best particle
```

Fig. 5: Pseudo code for PSO algorithm

4) *PathFinder Algorithm (PFA):*

- PFA is a population based optimization algorithm similar to the PSO but now we have a leader called the PathFinder and all other members try to follow this leader. The PathFinder is the member which has the best fitness value in the whole herd. The position is considered as the solution just like PSO but here we have two different equations to update the position: one for the PathFinder equation (10) and one for the other members equation (11).

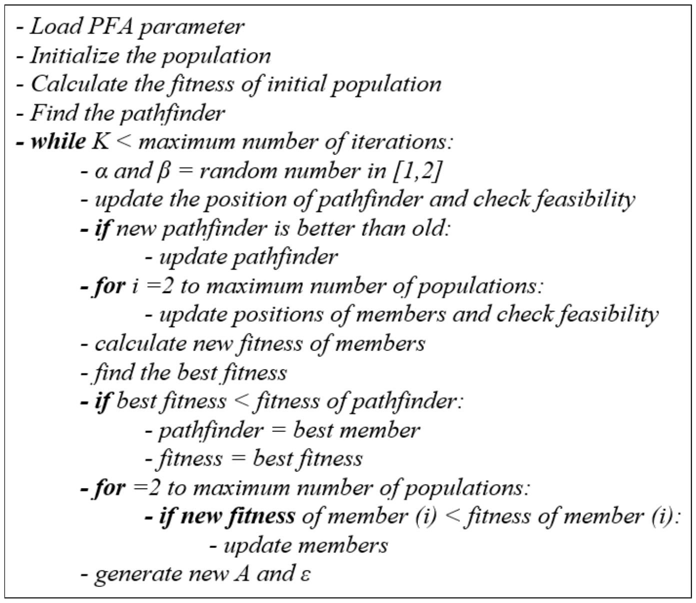
$$x_p^{K+1} = x_p^K + 2r_3 \cdot \left(x_p^K - x_p^{K-1}\right) + A \tag{10}$$

$$x_i^{K+1} = x_i^K + R_1 \cdot \left(x_j^K - x_i^K\right) + R_2 \cdot \left(x_p^K - x_i^K\right) + \varepsilon, \quad i \geq 2 \tag{11}$$

$$A = u_2 \cdot e^{\frac{-2K}{R_{\max}}} \tag{12}$$

$$\varepsilon = \left(1 - \frac{K}{K_{\max}}\right) .u_1.D_{ij}, \quad \mathrm{D}_{ij} = \|x_i - x_j\| \tag{13}$$

In equation (10) we update the position of the PathFinder x_p. K is the number of iteration and r_3 is a random number between [0,1]. A is defined in equation (12). In equation (11), the position of the other members x_i is updated for the member i. The member j is a neighbour to be followed. This component added to make sure that no member can move alone away from the herd. R1 is equal to $\alpha r1$ is and R2 is equal to $\beta r2$. r_1 and r_2 are random numbers between [0,1]. α is the coefficient for interaction and β is the coefficient of attraction. ϵ is defined in equation (13). Figure 6 shows the pseudo code for PFA.



```
- Load PFA parameter
- Initialize the population
- Calculate the fitness of initial population
- Find the pathfinder
- while K < maximum number of iterations:
        - α and β = random number in [1,2]
        - update the position of pathfinder and check feasibility
        - if new pathfinder is better than old:
                - update pathfinder
        - for i =2 to maximum number of populations:
                - update positions of members and check feasibility
        - calculate new fitness of members
        - find the best fitness
        - if best fitness < fitness of pathfinder:
                - pathfinder = best member
                - fitness = best fitness
        - for =2 to maximum number of populations:
                - if new fitness of member (i) < fitness of member (i):
                        - update members
        - generate new A and ε
```

Fig. 6: Pseudo code for PFA algorithm

IV. Results and Discussions

A. *Results of Each Algorithm*

In order to provide an objective comparison between all the algorithms, they were all run on the same parameters of our problem: $L = 300$m, $S = 50$m, $dL = 5$m, $V_{max} = 25$m/s, $V_{min} = 5$m/s, $u_{max} = 50$m/s^2, $u_{min} = -50$m/s^2, $V_m = 25$m/s and an estimated response time of the brakes $T_d = 0.5$s. Number of cars is 4. The stopping criteria of all the algorithms was based on the maximum number of iterations which was 100 for all the population-based algorithms and 1000 for the SA (being much faster). For populations-based algorithms, the population size is 50.

1) Simulated Annealing (SA): The SA was run for a total of 1000 iterations. As shown in Fig. 7, the cost function converges to a minimum value after approximately 900 iterations. Prior to that, when the temperature (energy) was high, the exploration feature was dominant after which the exploitation feature became the more dominant to drive the algorithm to converge to a minimum cost value.

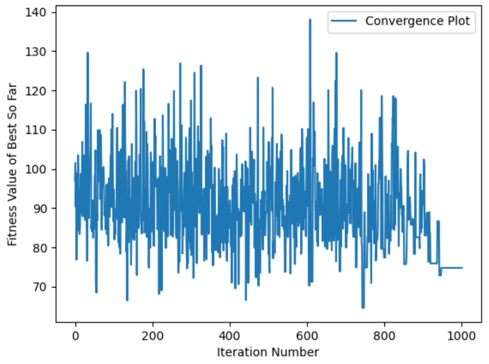
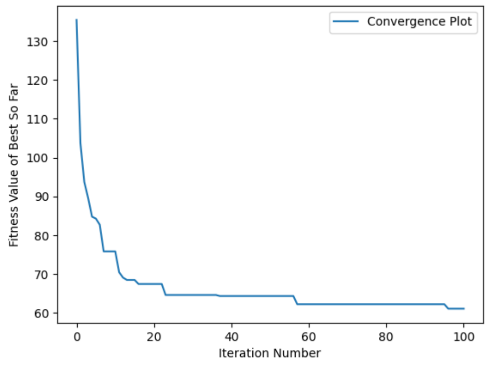


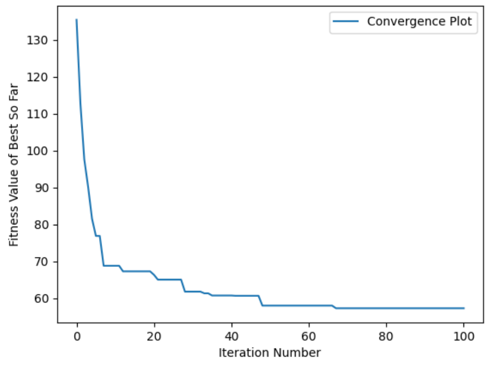
Fig. 7: Convergence plot of SA algorithm

It is also worth noting that when we tested the algorithm for an increased number of iterations, it allows more time for exploration which can lead to finding a more optimal solution that was undiscovered before,

2) Genetic Algorithm (GA): The GA was run under the following parameters: $\%$Elitism $= 0.1$, $\%$Crossover $= 0.8$, $\%$Mutation $= 0.1$, Mutation Selection $=$ worst. However, we tested it for different selection criteria for parents and survivors: (rank selection for parents, fitness selection for survivors) and (random selection for parents, age selection for survivors) as shown in Fig. 8 which shows that random selection for parents and age selection for survivor gives faster convergence at the 65th iteration compared to the another case which converged at the 98th iteration to a worse result.



(a) GA convergence for rank selection for parents and fitness selection for survivors

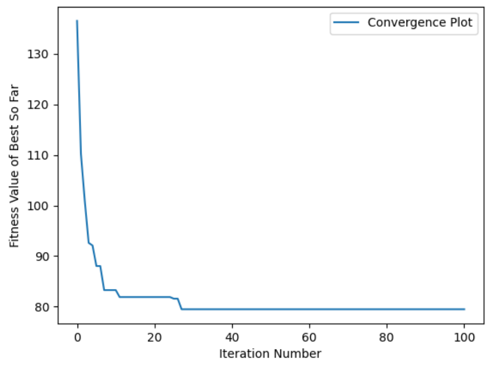


(b) GA convergence for random selection for parents and age selection for survivors

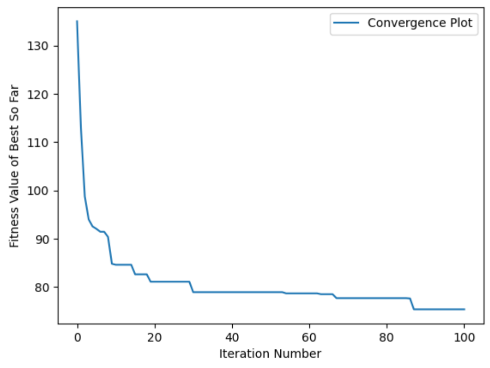
Fig. 8: Results of GA under different selection criteria for parents and survivors

3) Particle Swarm Optimization (PSO): The PSO algorithm was run under the following parameters: Neighbourhood Topology: Star, Fitness Update: asynchronous and for two different Inertia Weight schemes: constant (with value 0.792) and varying (linear decreasing) as function of the current iteration number, i, according to Eq. (14) with $w_0 = 0.8$ and $w_f = 0.2$. The results are shown in Fig. 9 which shows that varying the inertia weight gives better results as it allows for more exploration in the first iterations and decreases to give a more dominant exploitation behavior in the later iterations.

$$w_i = w_o - i\left(\frac{w_0 - w_f}{i_{max}}\right) \tag{14}$$



(a) PSO convergence for constant inertia weight



(b) PSO convergence for varying inertia weight

Fig. 9: Results of PSO under different inertia weight schemes

4) PathFinder Algorithm (PFA): The pathfinder algorithm was run using the parameters mentioned in the previous section. There are no parameters provided by the algorithm to tune it for different cases except only the maximum number of iterations which was kept at 100 for comparison consistency with the other algorithms. The algorithm convergence plot is shown in Fig. 10.

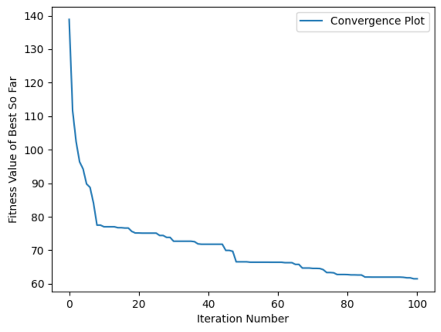


Fig. 10: Convergence plot of PFA algorithm

B. Discussions

The above results show that GA outperforms the other algorithms in terms of solution optimality followed by the PFA. The SA and PSO algorithms gives solutions with nearly the same fitness value. We also want to discuss how the fuel and minimization contributes to the convergence of the total cost function. As shown in Fig. 11, we can see that both fuel and time decreases with iterations. However, we can see that there are some iterations where fuel and time increases as in the 20th and 30th iterations respectively. This is due to the fact that decreasing the travel time requires burning more fuel and vice versa. However, there are other factors that cancel this factor such as the absence of the effect of friction in our model.

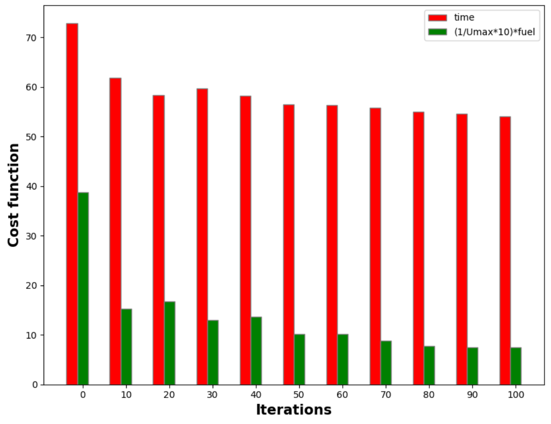


Fig. 11: Fuel vs Time

C. Key Performance Indices (KPIs)

To get a deeper comparison between the algorithms, each algorithm is run 10 times and the performance of each algorithm is evaluated based on 3 different KPIs: average of the best solution cost value, standard deviation of the best solution cost value and the average execution of each algorithm. The 3 KPIs are illustrated in TABLE I. The KPIs proves that the GA is best algorithm to give an optimal repeatable solution and it also has the best execution time among the population-based algorithms. However, SA outperforms it in terms of the execution time only.

Algorithm	Average	Standard deviation	Average execution time (sec)
SA	73.446	2.057	17.770
GA	57.984	1.302	131.497
PSO	74.3578	1.7123	193.7593
PFA	61.1193	2.2163	465.0918

TABLE I: Comparison between SA, GA, PSO and PFA based on 3 different KPIs.

D. Visualization & Validation: MATLAB Driving Scenario

In order to get a real physical visualization of the solution, the Driving Scenario Toolbox in MATLAB is used where four intersecting roads where constructed with the same parameters (control zone length, merging zone length, ...etc) as that of our problem (see Fig. 12). After the problem has been solved in python, the solution trajectories are then exported and loaded into MATLAB to be used in order to animate the vehicles motion that is planned by our optimization algorithm. The animation shows that the vehicles respects the minimum safe defined distance between each others in order to avoid collisions.

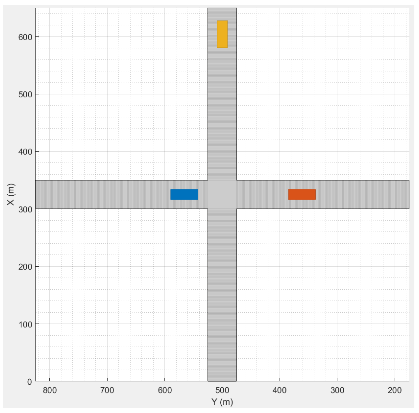


Fig. 12: Driving scenario of our solution for 4 cars

V. Conclusion and Future Recommendations

Traditional solutions to the traffic problem specially at intersection areas are not always the best fit when it comes to cost, autonomity and throughput. In this paper, a solution is proposed that copes with industry 4.0 revolution and the evolve of the smart cities. Exploiting the advantage of the autonomous vehicles being connected together, our proposed solution was based on optimizing the vehicles behavior in a defined control zone before entering the intersection area.

We discussed 4 different optimization algorithms that can be used to give an optimal solution to our problem: SA, GA, PSO and PFA. We evaluated the performance and optimality of each algorithm and performed comparison between them. In addition, a visualization aid is used in order to animate the solution for better understanding of the solution and whether it meets the requirements or not.

More future work can be done to test the algorithms on much vast cases of our problem. For example, the case where the cars can change their directions inside the merging zone in addition to the case of multiple driving lanes.

References

[1] B. Xu, X. J. Ban, Y. Bian, W. Li, J. Wang, S. E. Li, and K. Li, "Cooperative method of traffic signal optimization and speed control of connected vehicles at isolated intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 4, pp. 1390–1403, 2018.

[2] B. Chen, X. Pan, S. A. Evangelou, and S. Timotheou, "Optimal control for connected and autonomous vehicles at signal-free intersections," vol. 53, no. 2, pp. 15 306–15 311, 2020.

[3] M. A. Guney and I. A. Raptis, "Scheduling-based optimization for motion coordination of autonomous vehicles at multilane intersections," vol. 2020, pp. 1–22, Mar. 2020.

[4] A. Hadjigeorgiou and S. Timotheou, "Optimizing the trade-off between fuel consumption and travel time in an unsignalized autonomous intersection crossing," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 2443–2448.

[5] A. Baz, P. Yi, and A. Qurashi, "Intersection control and delay optimization for autonomous vehicles flows only as well as mixed flows with ordinary vehicles," *Vehicles*, vol. 2, no. 3, pp. 523–541, 2020.