Computational Intelligence For Optimization Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Vehicle Routing Problem

Flix Data

Andreia Rio, number: 20230542

Catarina Oliveira, number: 20230556

Marta Soeiro, number: 20230580

Rodrigo Fazendeiro, number: 20230756

Link to the git repository: https://github.com/andreiario/FlixData

Members contribution: All members contributed equally to the development of this project and were involved in all decisions taken.

June 2024

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1. Introduction

To promote development and advancement, a robust public transportation system is crucial, linking cities for better collaboration, work, and social interaction. In our problem, we will try to use an uncommon, but promisor mean of transport, Hydrogen buses. These buses are a future sustainable alternative to diesel, although nowadays they have a limited range of about 600 kilometers. For this reason, our project leverages genetic algorithms to design an example of an efficient network of routes that, by optimizing routes to accommodate this reduced autonomy, enhance connectivity and reliability. This approach ensures a cost-effective, environmentally friendly transportation network, supporting urban mobility and economic growth.

To tackle this challenge, we have compiled a dataset comprising 24 of the 26 Portuguese NUTS III (Nomenclature of Territorial Units for Statistics), excluding the ones belonging to the islands, as we cannot travel there by bus. For calculating the distances between NUTS, we used Google Maps, and we gathered the smallest distance from all the paths available. Also, in some cases we could not use the name of the NUT to search for distance as they do not exist as an actual place, so we used a city/location in the middle of the region. From PORDATA[1], we got the population and petrol consumption. Petrol consumption allowed us to calculate an estimative of the hydrogen consumption to create a variable "has_petrol", where if the median of all NUTS was smaller than the hydrogen consumption, we could not fill the tank, attributing a value of 0, and 1 otherwise.

2. Problem Description

2.1. Vehicle Routing Problem (VRP)

The *Vehicle Routing Problem* (VRP) is a variation of the *Travelling Salesman Problem* (TSP), where we have a set of routes for a fleet of vehicles, with one or more depots, with the objective to form the routes with the lowest cost to serve all costumers. As the number of cities increases, finding the best solution with TSP becomes impractical, therefore, VRP uses combinatorial integer optimization to approximate the best route, still being a NP-hard problem, for the usage of heuristic methods used to solve a NP-difficult problem.

2.2. Application of VRP in our Problem

The primary modification we made to a standard Vehicle Routing Problem (VRP) resolution was that, in our problem scenario, the route does not necessarily end at the same location as the starting point. While finding the shortest route possible was one of our objectives, it was not the most critical task. However, we imposed several additional constraints to ensure optimal performance, including:

- **Hard constrains:**
  - Cities can only be visited once per route, with no repetitions, and every city must be visited.
  - The bus cannot take more than 100 people at a time.
  - A fuel restriction for the planned routes, with a maximum distance a bus can travel without refueling set to 600km.
- **Soft constrains:**
  - Minimize the distance traveled on every route.

[1] Pordata is the Contemporary Portugal Database, with official and certified statistics about Portugal and Europe.

- Even the number of people traveling in each route.
- Include the pair Lisbon and Algarve on the same route, for their popularity as summer destinations, as well as the pair Trás-os-Montes and Região de Coimbra, together, given that Região de Coimbra is a popular destination for students of Trás-os-Montes, due to their prestigious universities.

3. Representation and Fitness

The representation is a crucial step in genetic algorithms, which significantly impacts the encoding of solutions, compatibility with genetic operators, constraint handling, fitness evaluation, exploration of the search space, scalability, and the ability to incorporate problem-specific knowledge. A well-designed representation facilitates the efficient and effective application of genetic algorithms to solve complex optimization problems.

Our representation is comprised of a list of routes, where each route contains a list of cities. We can define the route size and number of routes as long as it contains exactly 24 cities. Hence, we decided to create 4 routes, each containing 6 cities. Also, for each city we have the distances and the availability of refueling. When our representation is defined, we ensure that each city appears only once while also ensuring the fuel constraint. To achieve the latter, it was necessary to define the fuel limit slightly below the actual limit to avoid not refueling and no longer being able to reach a city where it can refuel, thus creating impossible solutions.

The fitness of an individual represents the quality of the solution, specifically, how short the total travel distance is while adhering to constraints. To enforce hard constraints, we monitored the distance traveled without refueling, resetting this variable to zero upon reaching a city with refueling capabilities. If any route exceeds the maximum distance allowed without refueling, it is automatically assigned a very poor fitness score (10.000).

Regarding population constraints, several assumptions were made. For larger cities (with populations over 400), only 5% of the population is assumed to need the bus service. For smaller cities, this percentage is set at 20%. This distinction assumes that there are significantly more people traveling to larger cities than leaving them. Additionally, it is assumed that half of the population in each city wants to travel in one direction, while the other half wants to travel in the opposite direction. The number of people exiting the bus at each city is proportional to the number of people boarding the bus, based on the number of remaining stops on the route. In other words, we calculate the number of cities left on the route and divide the number of people boarding the bus by the number of remaining stops. This value is updated at each subsequent city until the destination is reached. Considering the number of people boarding and exiting the bus, the bus cannot carry more than 100 passengers at any time. If this limit is exceeded, a very poor fitness score is assigned. This constraint was verified for both directions, as considering only one would result in not considering the destination city's population.

To ensure our soft constraints are met, we developed a comprehensive fitness function. For distance, we simply calculated the total distance of each route. Since our goal is to minimize the fitness function, it inherently aims to reduce the total travel distance. Regarding the even distribution of passengers across routes, we calculated the total number of people taking the bus for each city on each route. We then found the difference between the highest and lowest passenger counts and added this difference to the total distance. If all routes carry the same number of passengers, fulfilling our soft constraint, the difference is zero, thus minimizing the fitness score.

Finally, for each specified pair of cities found on the same route, the value 300 was subtracted from the fitness score. This value was fine-tuned through multiple runs of the algorithm to ensure it provided a sufficient incentive for the model to consider this constraint without making it a priority. If the reward was set too low, the model would ignore this constraint due to the greater impact of the travel distance on the fitness score. Conversely, if the reward was set too high, the model would prioritize these city combinations even when it was not optimal.

Our fitness function can also be calculated through the following equation:

$$F = \sum_{i=1}^{N} di + \left(\max_{i=1}^{N}(ni) - \min_{i=1}^{N}(ni)\right) - B$$

Equation 1: Fitness Function *(**N*** is the number of routes; ***di*** is the distance traveled by route ***i; ni*** is the number of people served by route ***i; max(ni)*** is the maximum number of people served by a single route; ***min(ni)*** is the minimum number of people served by a single route; ***B*** is the bonus for having the specified cities together (either 0 if no pair of cities is in the same route, 300 if only one combination is made and 600 if both combinations are in the representation)

4. Selection

Selection is a step where the population part that will be reproduced in the next step is chosen, for which the only formal requirement that must be met is the fact that the fittest individuals should have a higher probability of being chosen. The selection methods we employed are:

- **Roulette Wheel Selection (RWS)**, where the chances of an individual being chosen are proportional to its fitness value.
- **Rank Selection (RS)**, like RWS, but each individual solution's chance of being selected is not proportional to its fitness, but to its rank in the list of all individuals, ordered by fitness.
- **Tournament Selection (TS)** choice of the most fit individual to be chosen, between two individuals selected randomly.
- **Elitism Selection (ES)**, on which a certain percentage of the population, ordered by fitness, is always transferred to the next population. The algorithm makes sure that the solutions that are the best known so far would not be lost in the process of selection.

5. Crossover

For our problem concerning a VRP (complex combinatorial optimization problem where the goal is to find the most efficient routes for a set of buses). Among the crossover techniques, the *Partially Mapped Crossover* (PMX), *Order Crossover* (OX) and *Cycle Crossover* (CX) are particularly suitable for VRP because they are designed to work with permutations, preserving relative order and uniqueness of elements (Puljic et al. (2013)).

- **Partially Mapped Crossover (PMX)** ensures that the offspring inherit elements from both parents while maintaining their positions and uniqueness as much as possible, making it suitable for VRP where these properties are critical. Two crossover points are selected at the same position from each parent, and afterwards the genes are copied from the second parent between the two cut points to the offspring. Then, the remaining genes in the offspring before and after the cut point are copied from the first parent in the order that they appear. Finally, it fills the remaining positions by mapping the elements from the other parent, ensuring no duplicates and maintaining the relative order.

- **Order Crossover (OX)** maintains the order and uniqueness of elements by selecting a subsequence from one parent and preserving the relative order of elements from the other parent. This technique ensures that the resulting offspring are feasible routes that respect the original sequence. It begins with the selection of two random crossover points. After, copies the segment between these points from one parent to the offspring. It preserves the order by filling the remaining positions with cities from the other parent in the order that they appear, with no duplicates.
- **Cycle Crossover (CX)** works by creating cycles between two parents and ensuring that each element appears exactly once in the offspring. This preservation of uniqueness and relative order makes CX another appropriate method for VRP. Starts by identifying several cycles between two parents. Then these cycles are copied from the respective parent to form the child. It starts with the first gene of the first parent, visits the gene at the same position of the second parent, and then goes to the position with the same gene in the first parent, effectively forming a cycle. Process repeated until all cycles are formed.

6. Mutation

After crossover, the use of mutation is essential to prevent the algorithm from being trapped in a local minimum. Mutation recovers the lost genetic materials as well as for randomly disturbing genetic information. It is an insurance policy against the irreversible loss of genetic material (O. Abdoun et al, 2012). Mutation techniques implemented were:

- **Swap Mutation** involves swapping two elements within a sub list.
- **Scramble Mutation** selects a subset of elements within a sub list and randomly scrambles them.
- **Shuffle Mutation** randomly shuffles the order of cities within each route.
- **Insertion Mutation** works by selecting an element and inserting it into a different position within the same sub list.
- **Route Swap Mutation** where two random routes are selected, and two random cities, one from each route, are swapped.

7. Fitness sharing

By running our algorithm, we ended up with solutions very similar or even equal. To solve this and to avoid premature convergence to suboptimal solutions we used fitness sharing. Used to maintain diversity in the population, fitness sharing adjusts the fitness of the individuals, so that those who are more diverse get a “reward” while the most similar ones get a “penalty”. This showed no relevant results, with some solutions converging to values smaller than 1.

8. Comparison and Conclusions

To compare and take conclusions about our algorithm we performed graphs to visually identify what selection and crossover methods worked better and the effects of the use of elitism and the use of fitness sharing (Figures 1 – 24).

The plots showed us that our best solution includes the usage of roulette wheel combined with PMX crossover (Figure 7). Our best individual is in gen 100 with the following representation, obtaining a fitness of 1777:

- **Route 1:** ['Tâmega e Sousa', 'Douro', 'Viseu Dão Lafões', 'Beiras e Serra da Estrela', 'Beira Baixa', 'Alto Alentejo']
- **Route 2:** ['Alto Tâmega e Barroso', 'Terras de Trás-os-Montes', 'Região de Leiria', 'Médio Tejo', 'Região de Coimbra', 'Alentejo Central']

- **Route 3:** ['Alto Minho', 'Cávado', 'Ave', 'Área Metropolitana do Porto', 'Região de Aveiro', 'Oeste']
- **Route 4:** ['Lezíria do Tejo', 'Península de Setúbal', 'Grande Lisboa', 'Alentejo Litoral', 'Baixo Alentejo', 'Algarve']

For a better understanding of our final solution, we have represented it on a map (Figure 25).

To validate the effectiveness of our optimal solution, we calculated the total distance covered by our final solution, which amounted to 2238km. Regarding the distribution of people per route, our solution allocated 186, 189, 325, and 256 people on each route respectively, demonstrating a reasonably balanced distribution. We reason that the observed difference between the distribution of people for each route results of a tradeoff between the small distance between cities. On the route with the highest number of people, the distance between cities is the lowest and, given that the distance has a higher impact on fitness, the optimal solution prefers to optimize the distance rather than the population constraint. Finally, the constraint of putting 2 pairs of cities together in the same route was also followed as seen above on our best individual (Route 2: "Terras de Trás-os-Montes"/ "Região de Coimbra") and (Route 4: "Grande Lisboa"/ "Algarve").

In our analysis, it's challenging to determine if the order of these routes is optimized to minimize the distance traveled on each route. We cannot definitively conclude whether the route order is influenced by the constraints on the number of people. To potentially enhance the distance, increasing the mutation rate of swap mutation could have been employed. This mutation technique involves changing the order of two elements within the same route, which may lead to improvements in overall distance traveled.

It is also important to highlight the role of elitism in obtaining the best response. It can be confirmed that, in our optimization problem, elitism helps the algorithm evolve and achieve better results within the same number of generations, by retaining the elite through every generation.

Roulette Wheel with Partially Matched Crossover (PMX) crossover is a good combination for our problem. This is probably due to the fact that Roulette Wheel offers a balanced exploration of the search space, by favoring better fit individuals while still considering lower-fit ones. Additionally, it ensures diversity in the population by giving all individuals an opportunity to crossover, irrespective of their fitness. This diversity is critical for avoiding premature convergence and facilitating exploration of a wide range of potential solutions in VRP.

The PMX crossover effectively maintains good solutions within the population. By exchanging subsequences between parents and fixing any conflicts to ensure valid permutations, PMX retains desirable traits from both parents, potentially yielding offspring with similar or improved fitness. In VRP, where efficiency in route finding is paramount, preserving good solutions through crossover is indispensable. Furthermore, PMX crossover inherently respects the constraints implemented, such as vehicle capacity, maximum fuel distance and each city being visited only once. Through the exchange of subsequences and resolution of conflicts, PMX ensures that offspring solutions adhere to these constraints, rendering them feasible solutions to our VRP.

Overall, our results seemed to significantly improve with the constraints we defined. Unfortunately, the *Order Crossover* (OX) did not work well for our problem. One of the causes could be since, when specific constraints are not well-handled by OX, the resulting offspring often violate these constraints, leading to poor performance, and it was exactly what we observed.

9. References

PORDDATA: Venda de combustíveis para consumo.

https://www.pordata.pt/municipios/venda+de+combustiveis+para+consumo-34-175.

PORDATA: População residente: total e por grandes grupos etários.

https://www.pordata.pt/municipios/populacao+residente+total+e+por+grandes+grupos+etarios-390.

Google Maps, https://www.google.com/maps.

Ochelska-Mierzejewska, J., Poniszewska-Marańda, A., & Marańda, W. (2021). Selected genetic algorithms for vehicle routing problem solving. Electronics, 10(24), 3147.

Kurnia, H., Wahyuni, E. G., Pembrani, E. C., Gardini, S. T., & Aditya, S. K. (2018). Vehicle routing problem using genetic algorithm with multi compartment on vegetable distribution. IOP Conference Series: Materials Science and Engineering, 325(1), 012012.

Puljić, K., & Manger, R. (2013). Comparison of eight evolutionary crossover operators for the vehicle routing problem. Mathematical Communications, 18(2).

Abdoun, O., Abouchabaka, J., & Tajani, C. (2012). Analyzing the performance of mutation operators to solve the travelling salesman problem. 1-18.

10. Appendix

Figure 1 – Median Fitness Values Across 100 Generations from 30 iterations (crossover = pmx_crossover, selection = selection_methods, mutate = random_swap_mutation)

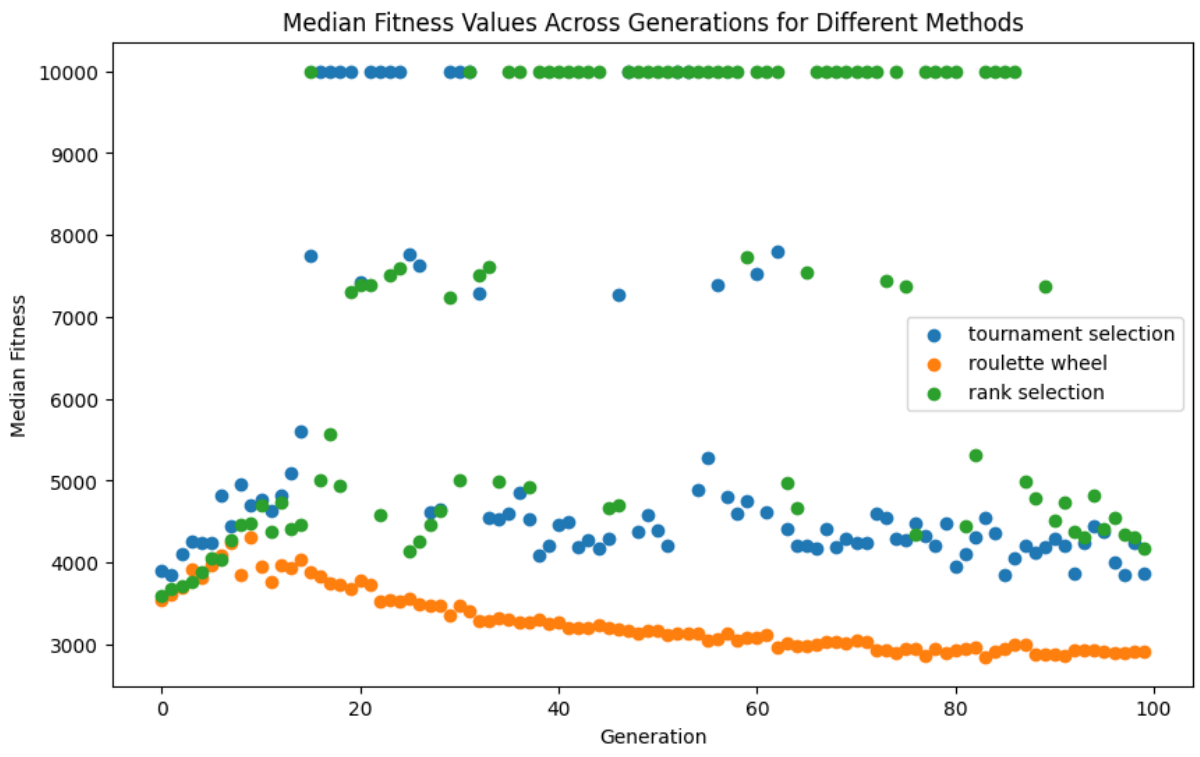


Figure 2 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = cycle_crossover, selection = selection_methods, mutate = shuffle_mutation)

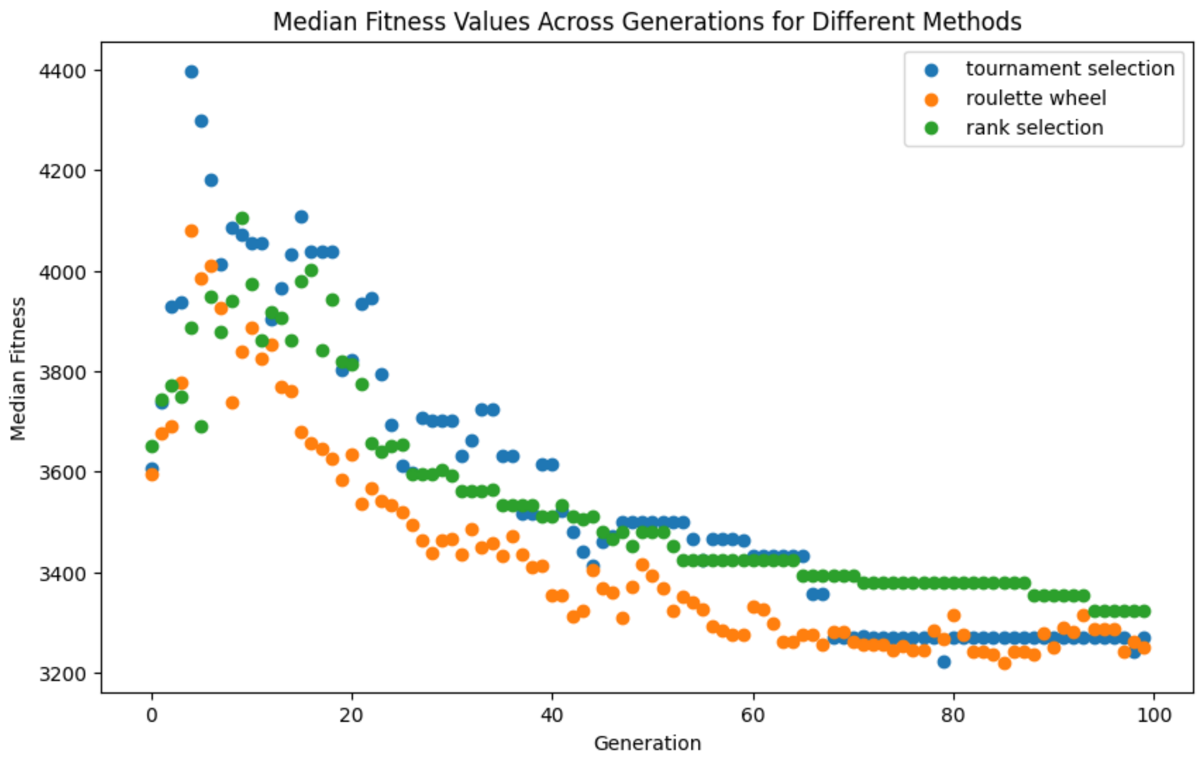


Figure 3 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = order_crossover, selection = selection_methods, mutate = route_swap_mutation)

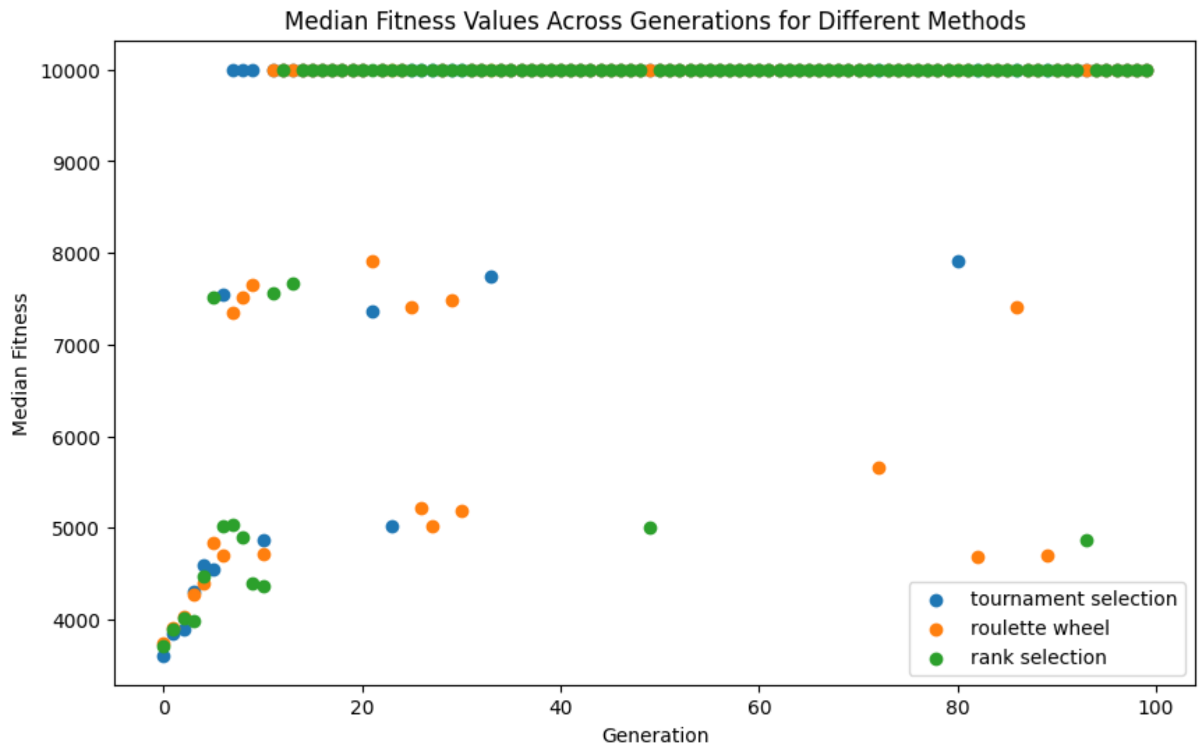


Figure 4 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = fps, mutate = random_swap_mutation)

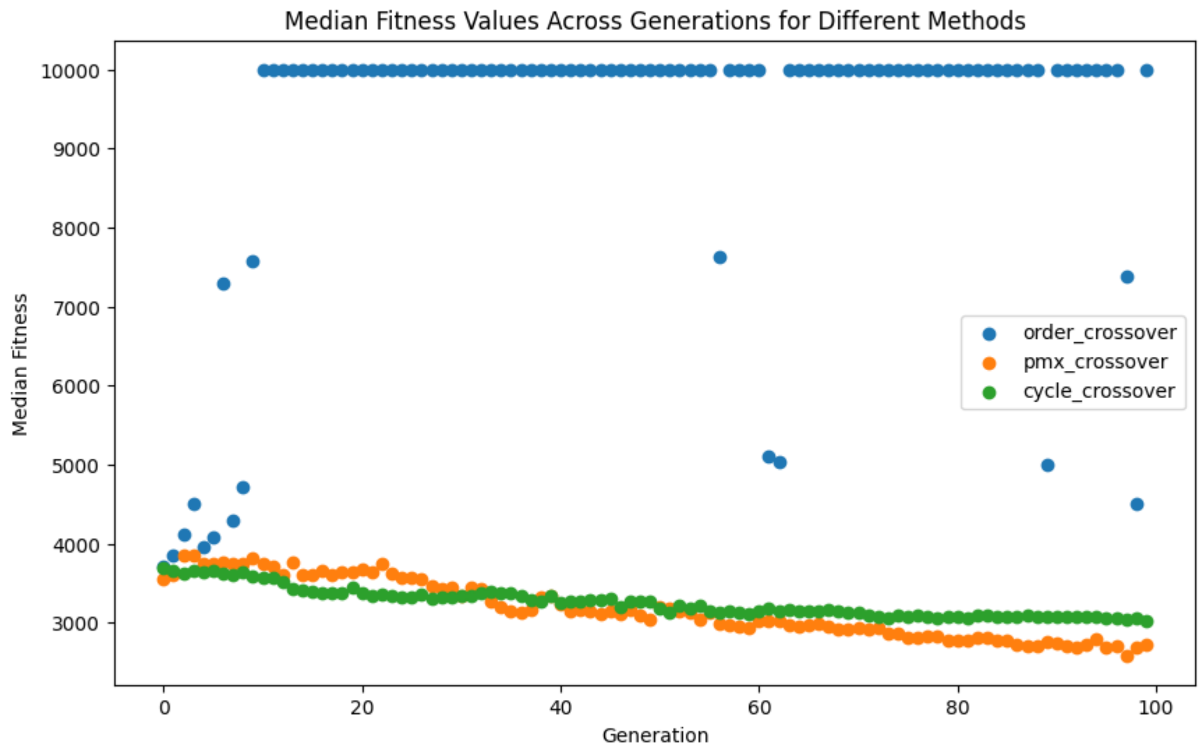


Figure 5 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = rank_selection, mutate = shuffle_mutation)

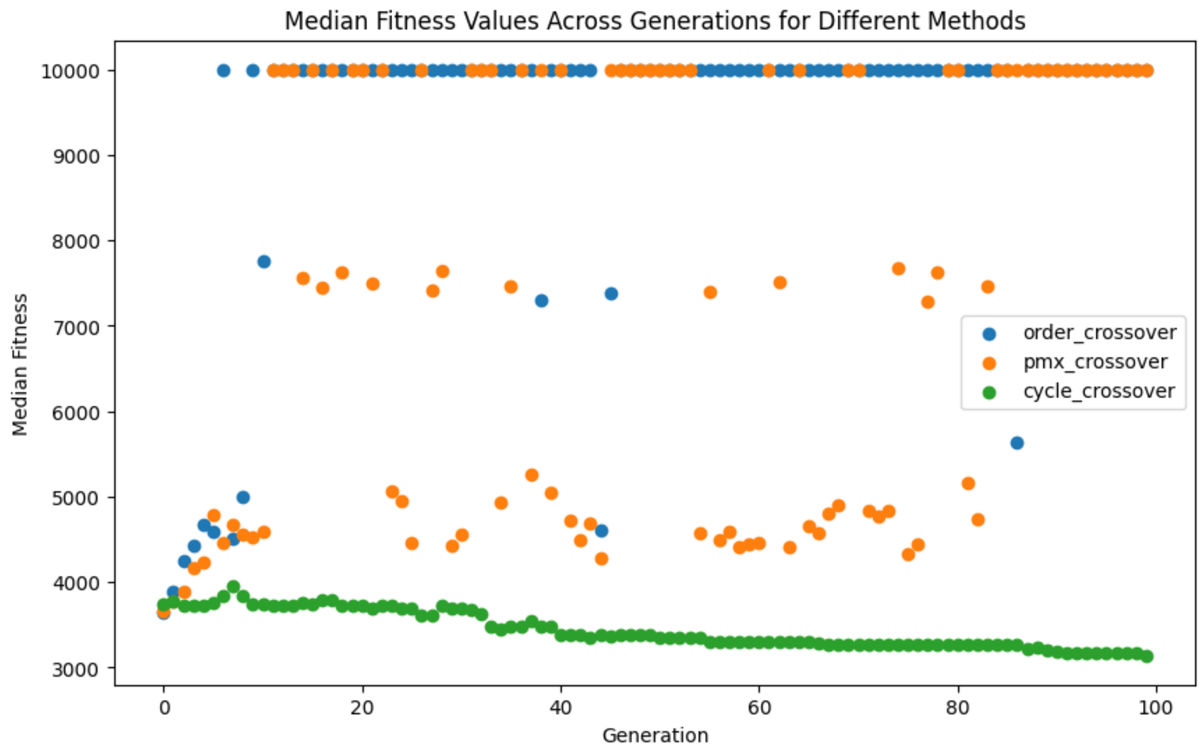


Figure 6 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = tournament_sel, mutate = route_swap_mutation)

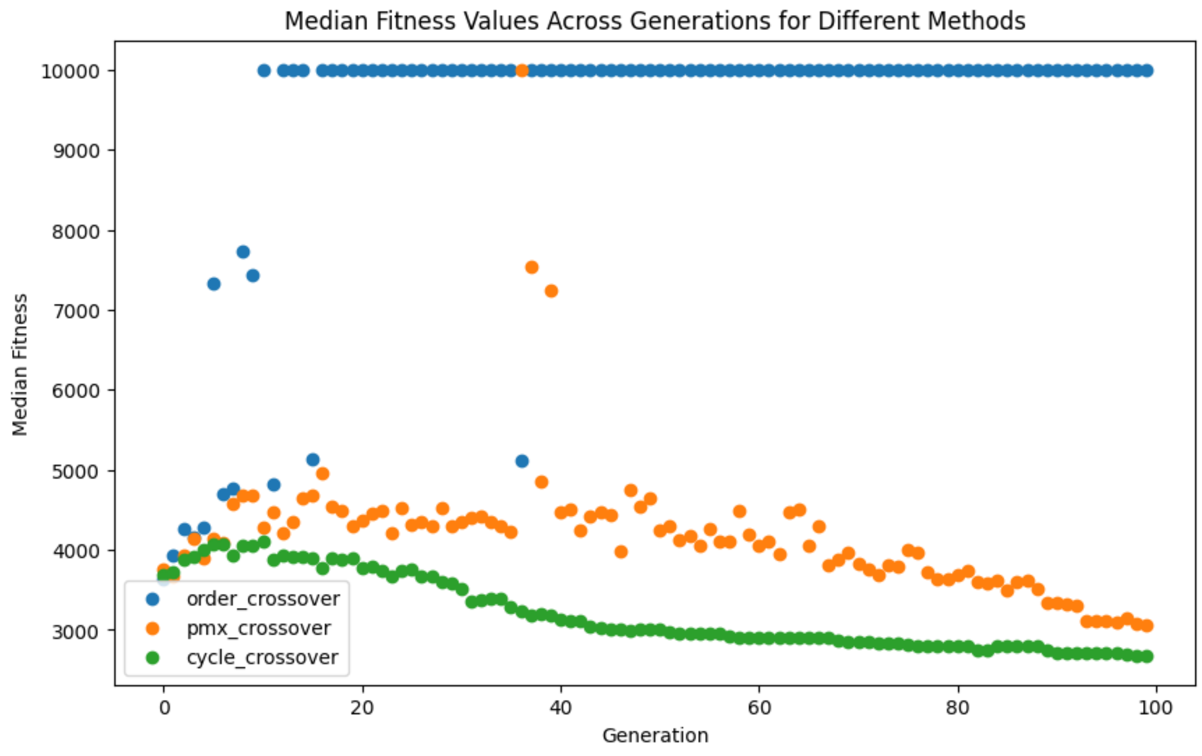


Figure 7 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = pmx_crossover, selection = selection_methods, mutate = route_swap_mutation, elitism = True)

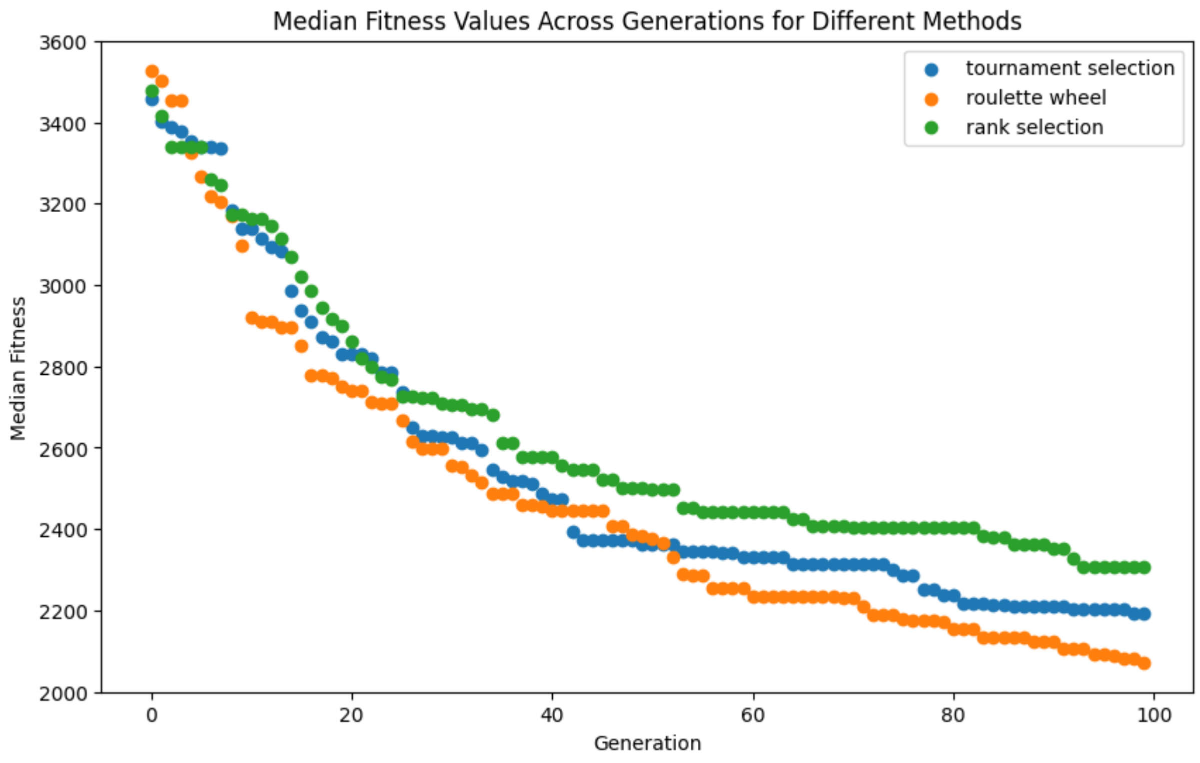


Figure 8 - Median Fitness Values Across 100 Generations from 30 iterations crossover = cycle_crossover, selection = selection_methods, mutate = random_swap_mutation, elitism = True)

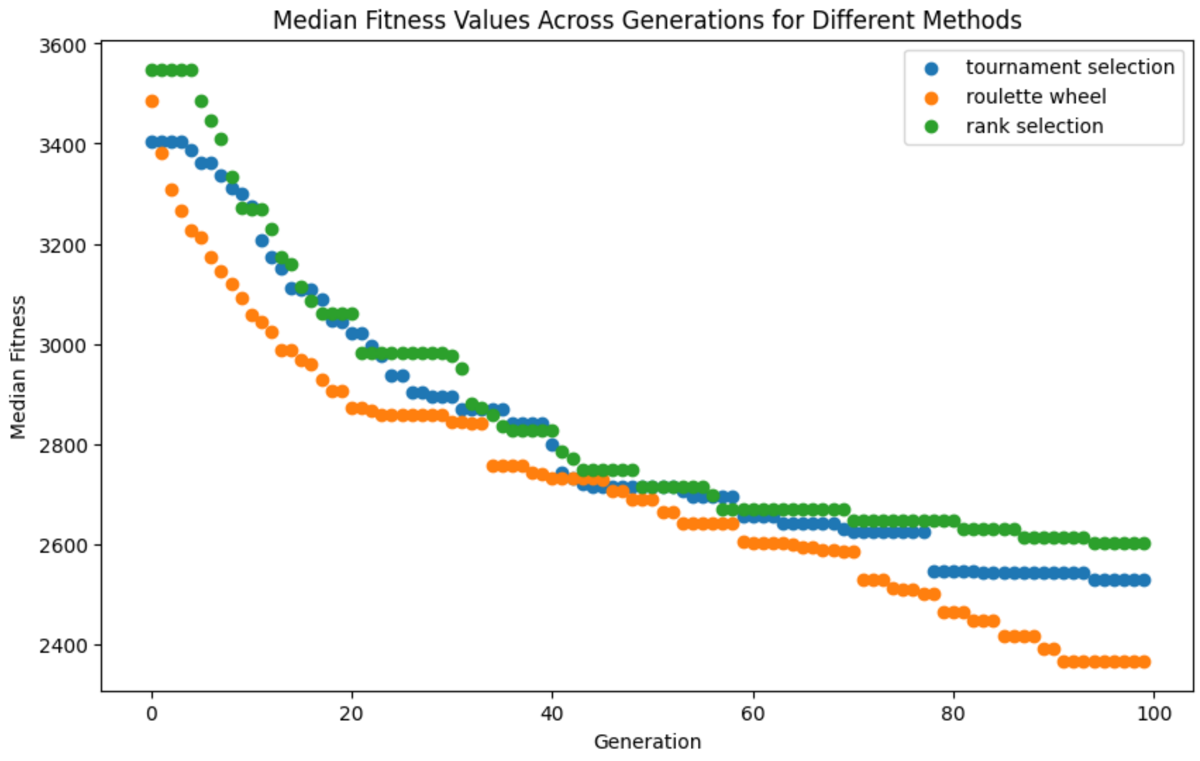


Figure 9 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = order_crossover, selection = selection_methods, mutate = shuffle_mutation, elitism = True)

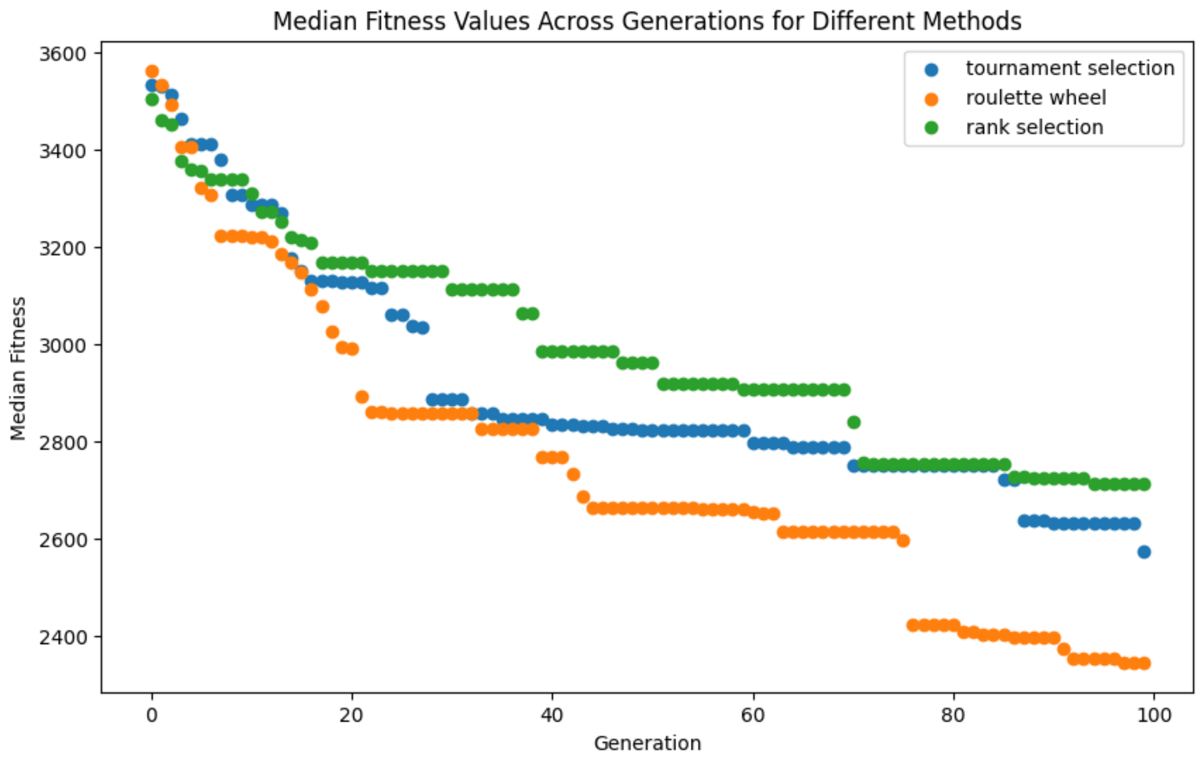


Figure 10 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = fps, mutate = shuffle_mutation, elitism = True)

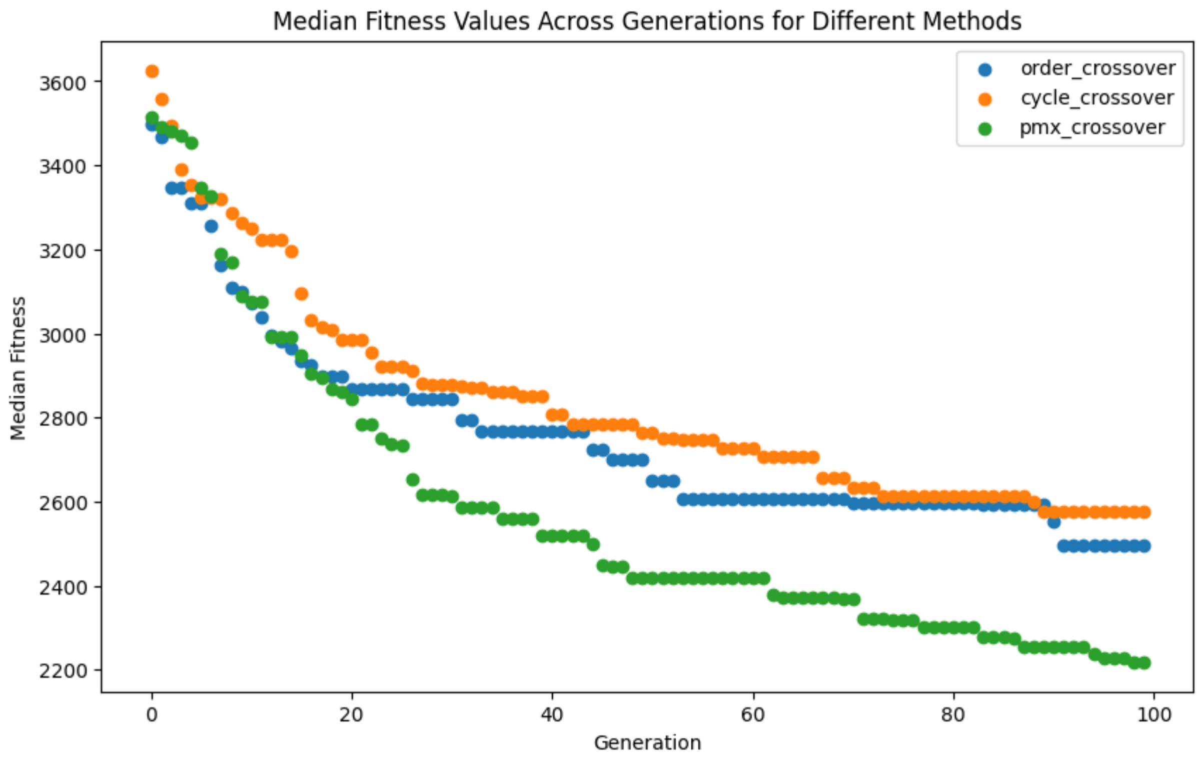


Figure 11 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = tournament_sel, mutate = random_swap_mutation, elitism = True)

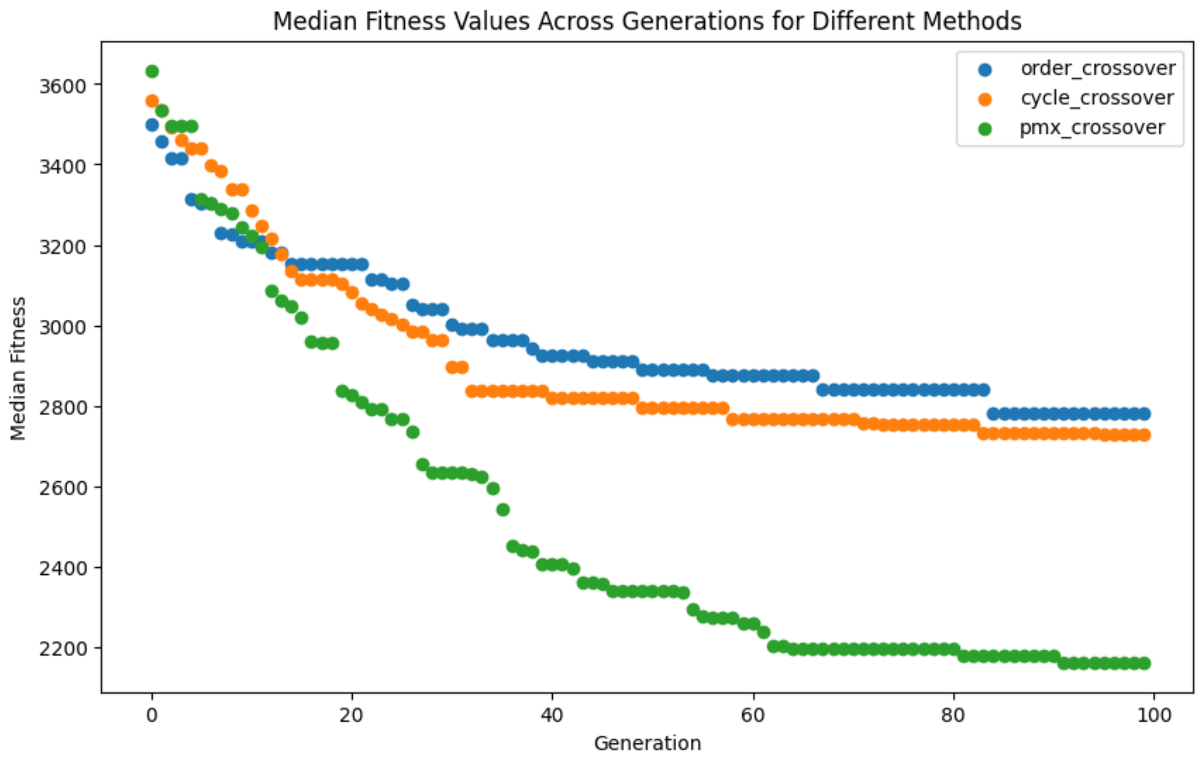


Figure 12 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = rank_selection, mutate = route_swap_mutation, elitism = True)

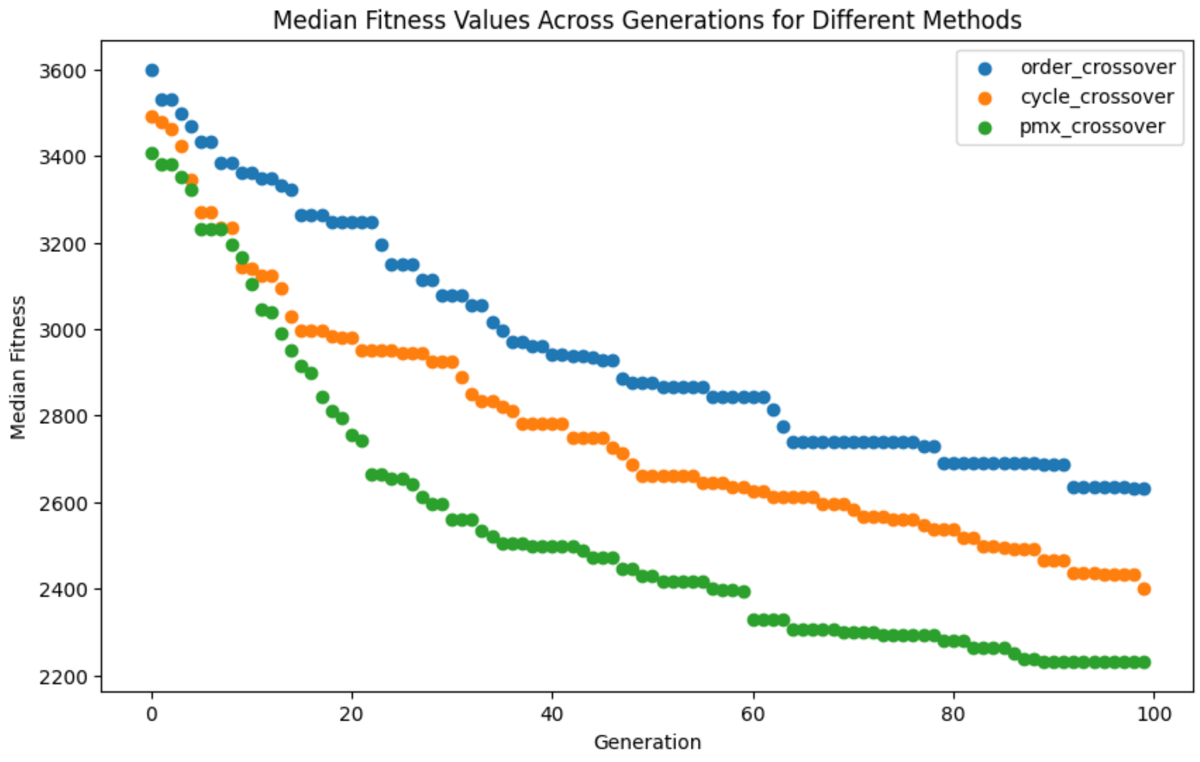


Figure 13 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = cycle_crossover, selection = selection_methods, mutate = shuffle_mutation, fit_share= True)

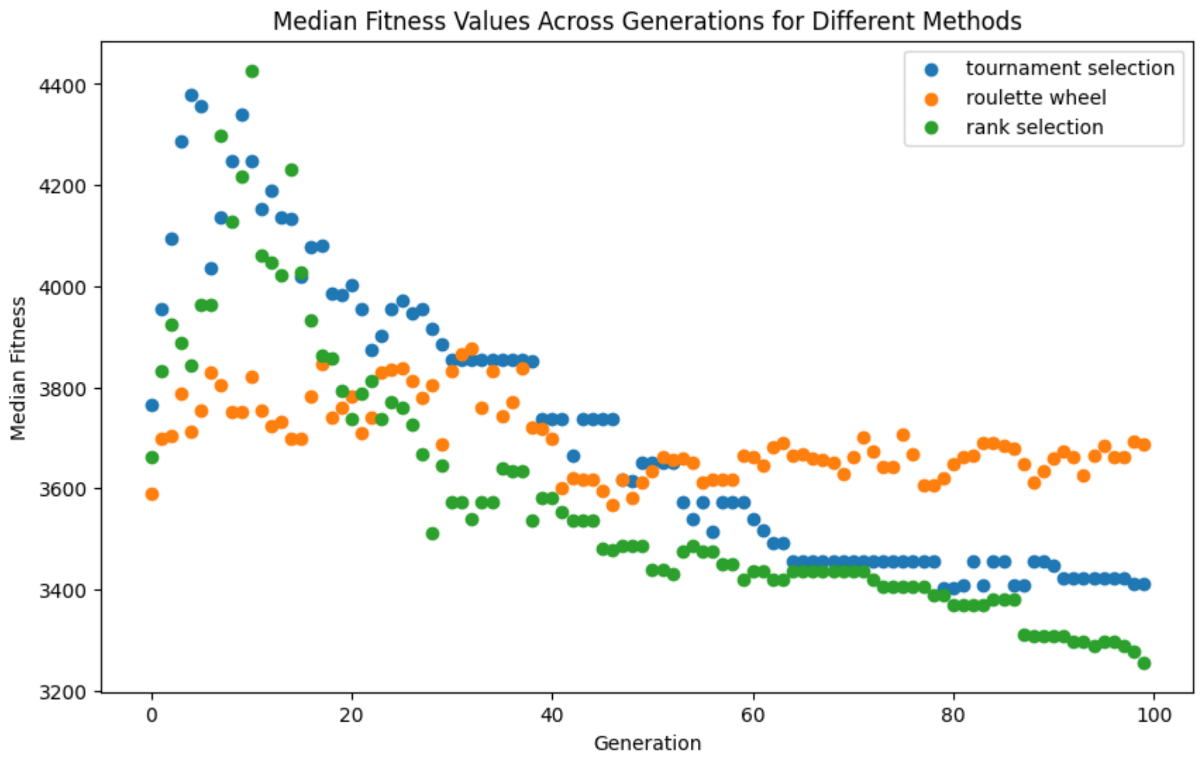


Figure 14 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = pmx_crossover, selection = selection_methods, mutate = route_swap_mutation, fit_share= True)

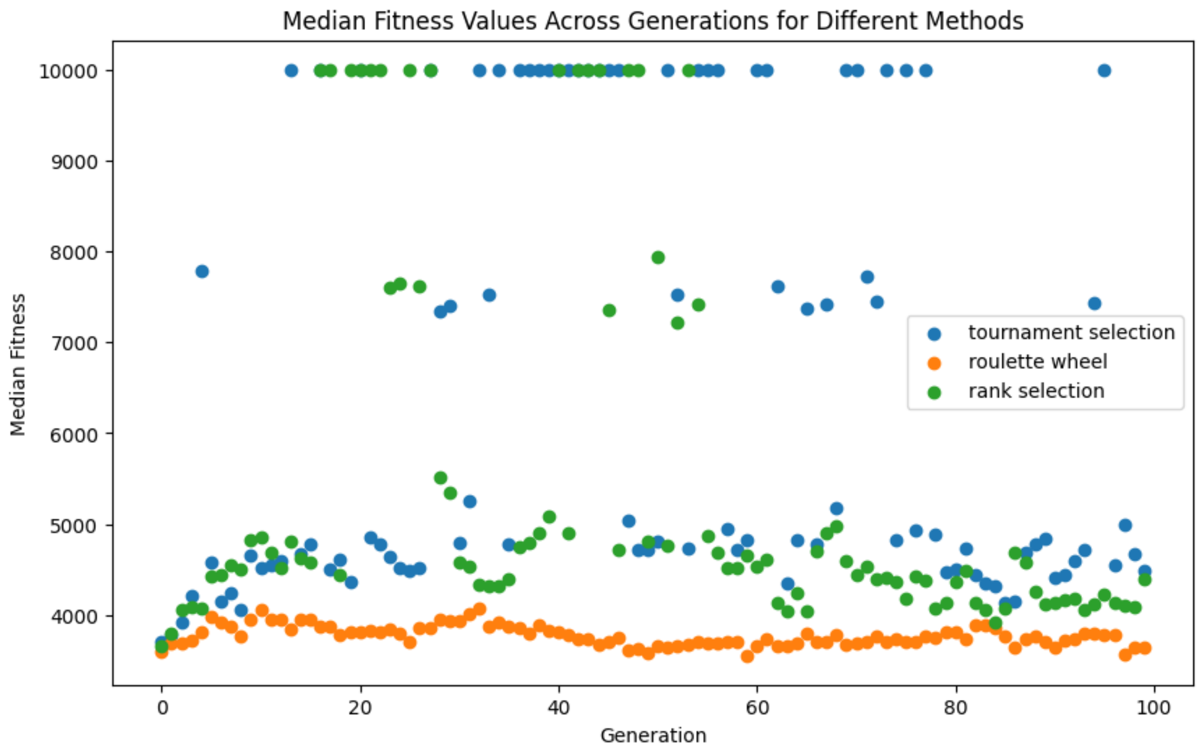


Figure 15 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = order_crossover, selection = selection_methods, mutate = random_swap_mutation, fit_share= True)

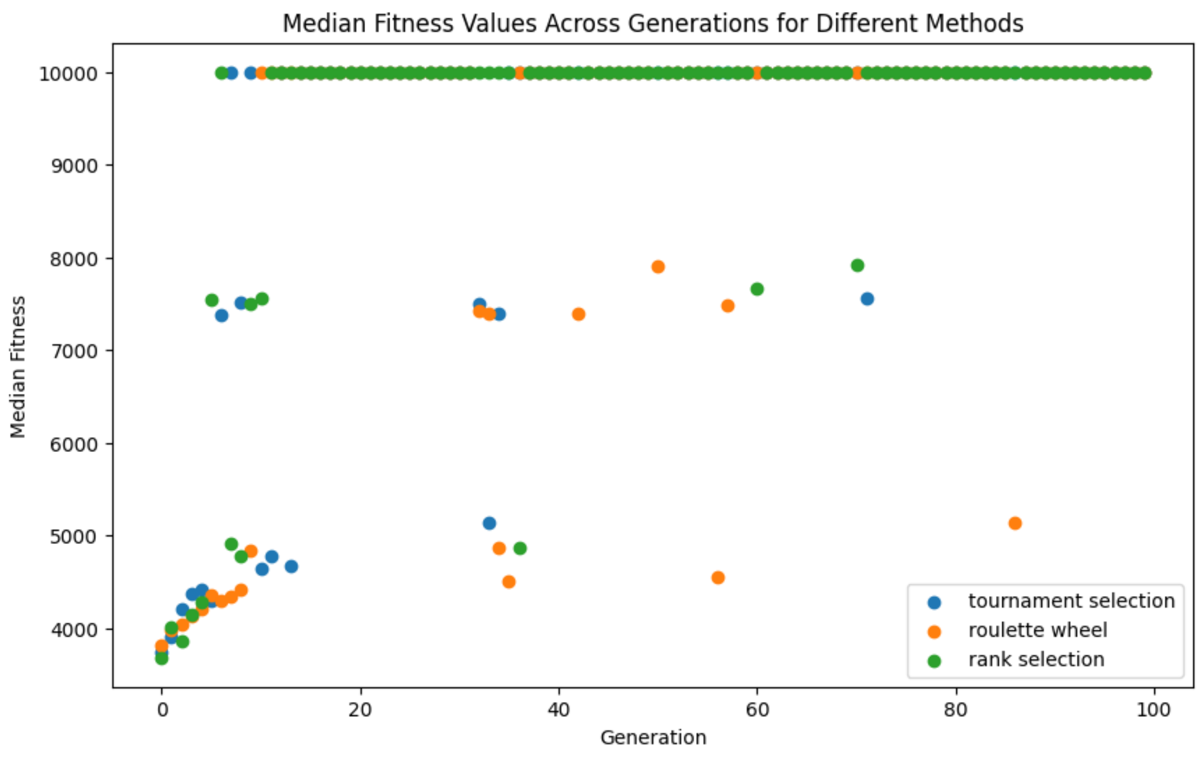


Figure 16 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = tournament_sel, mutate = shuffle_mutation, fit_share=True)

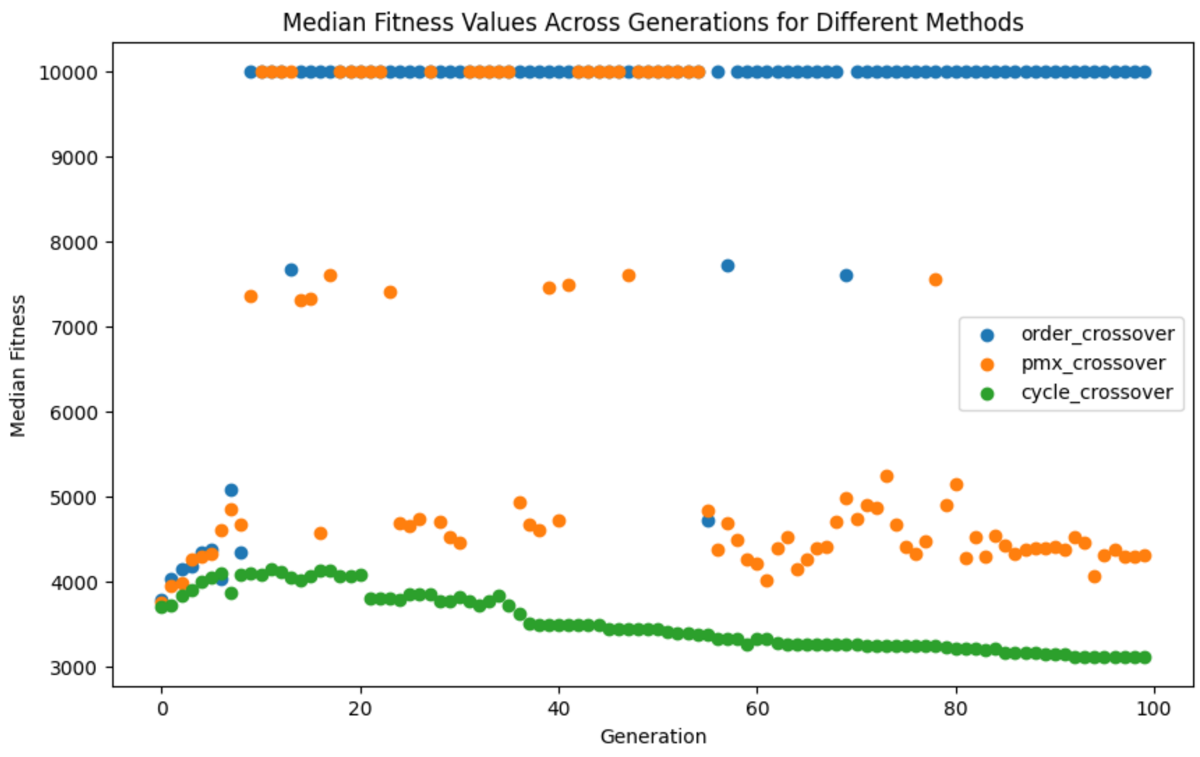


Figure 17 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = rank_selection, mutate = route_swap_mutation, fit_share=True)

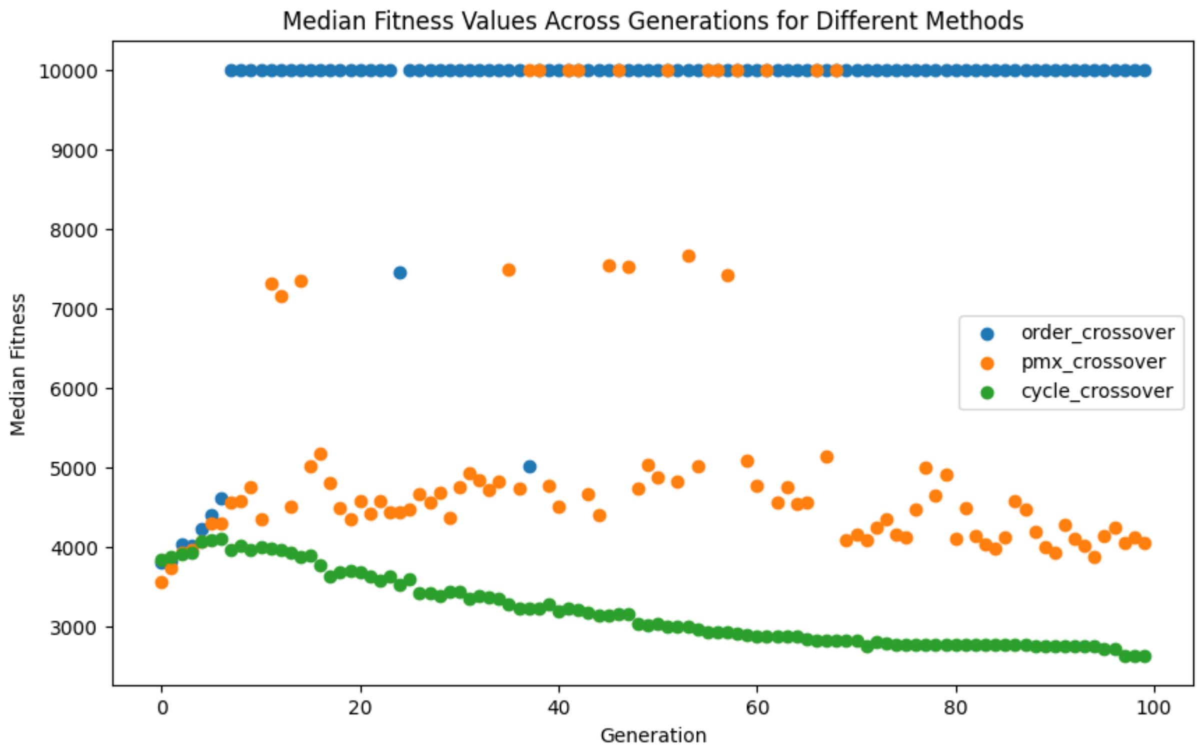


Figure 18 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = fps, mutate = random_swap_mutation, fit_share=True)

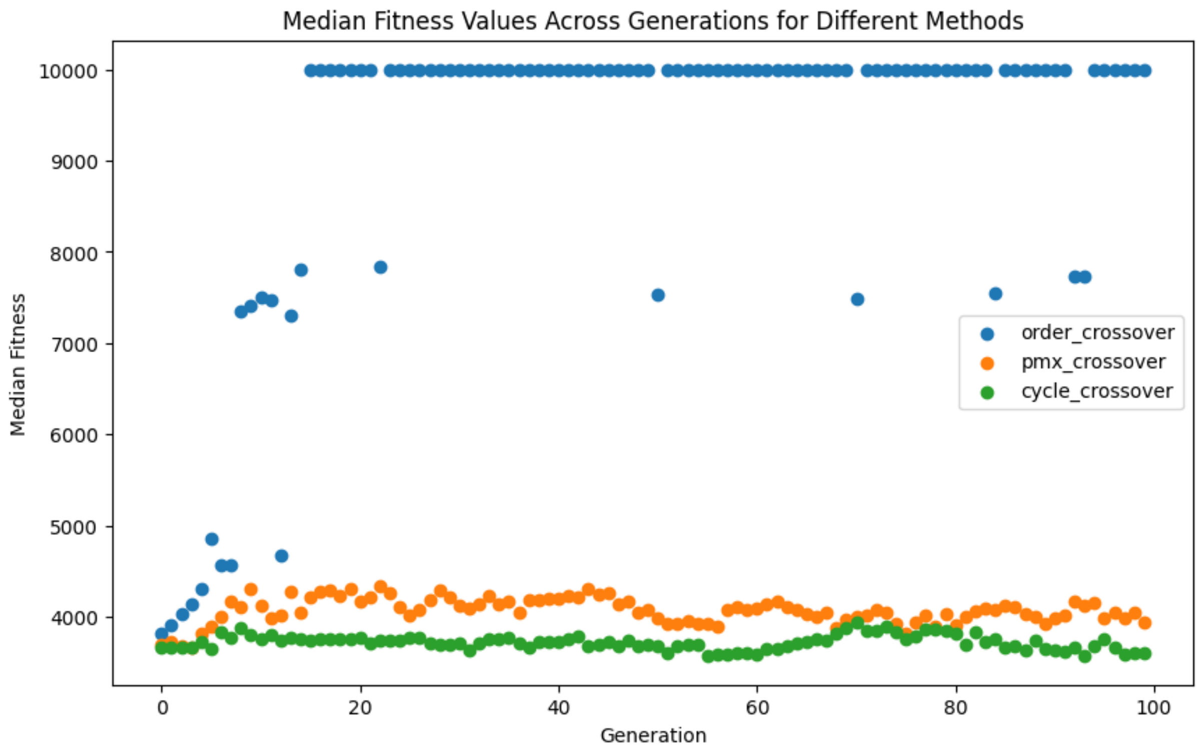


Figure 19 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = cycle_crossover, selection = selection_methods, mutate = shuffle_mutation, elitism = True, fit_share=True)

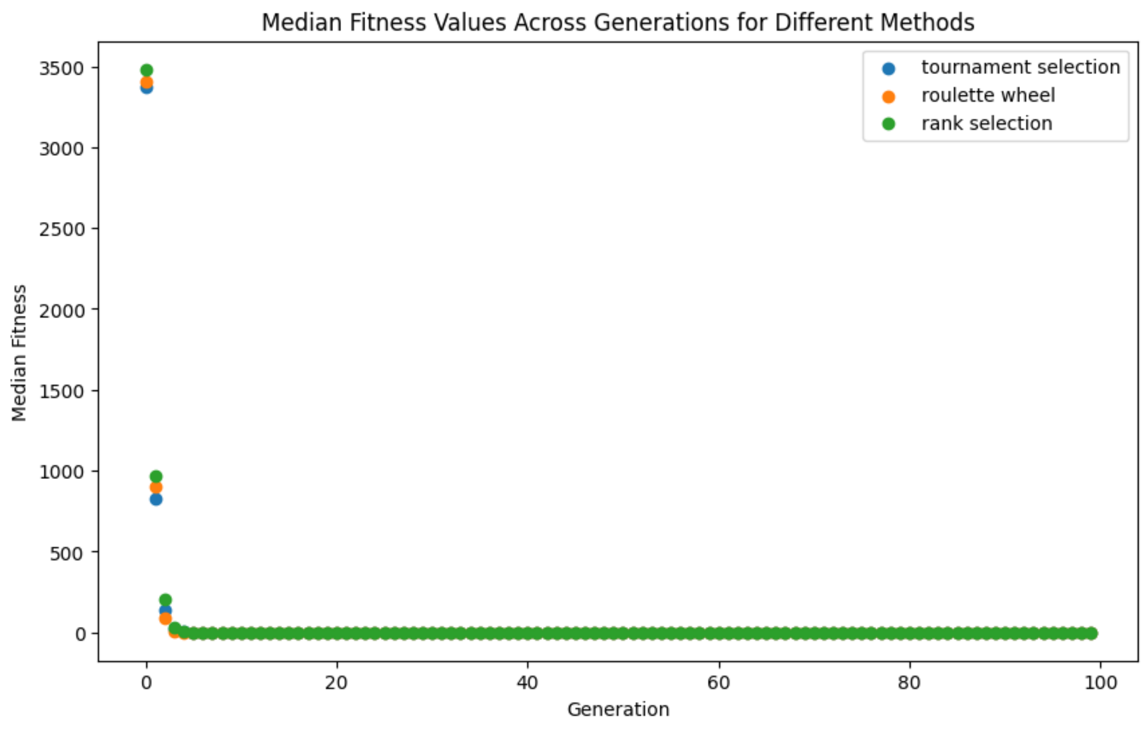


Figure 20 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = pmx_crossover, selection = selection_methods, mutate = shuffle_mutation, elitism = True, fit_share=True)

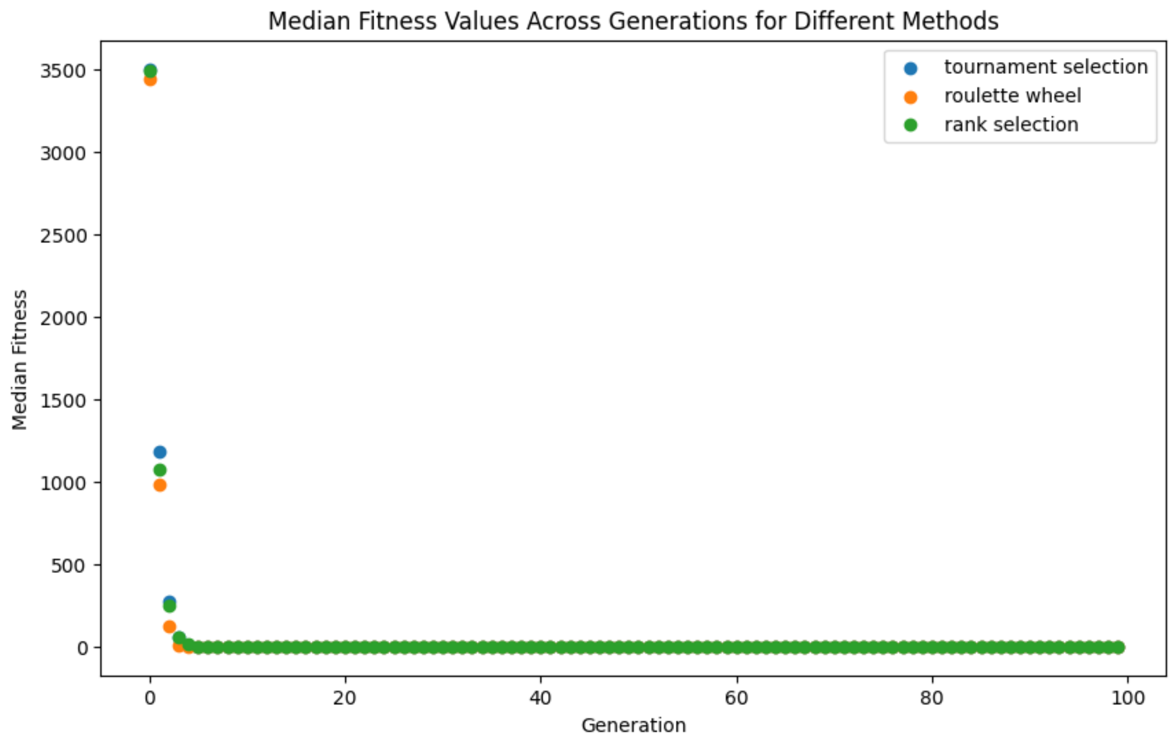


Figure 21 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = order_crossover, selection = selection_methods, mutate = shuffle_mutation, elitism = True, fit_share=True)

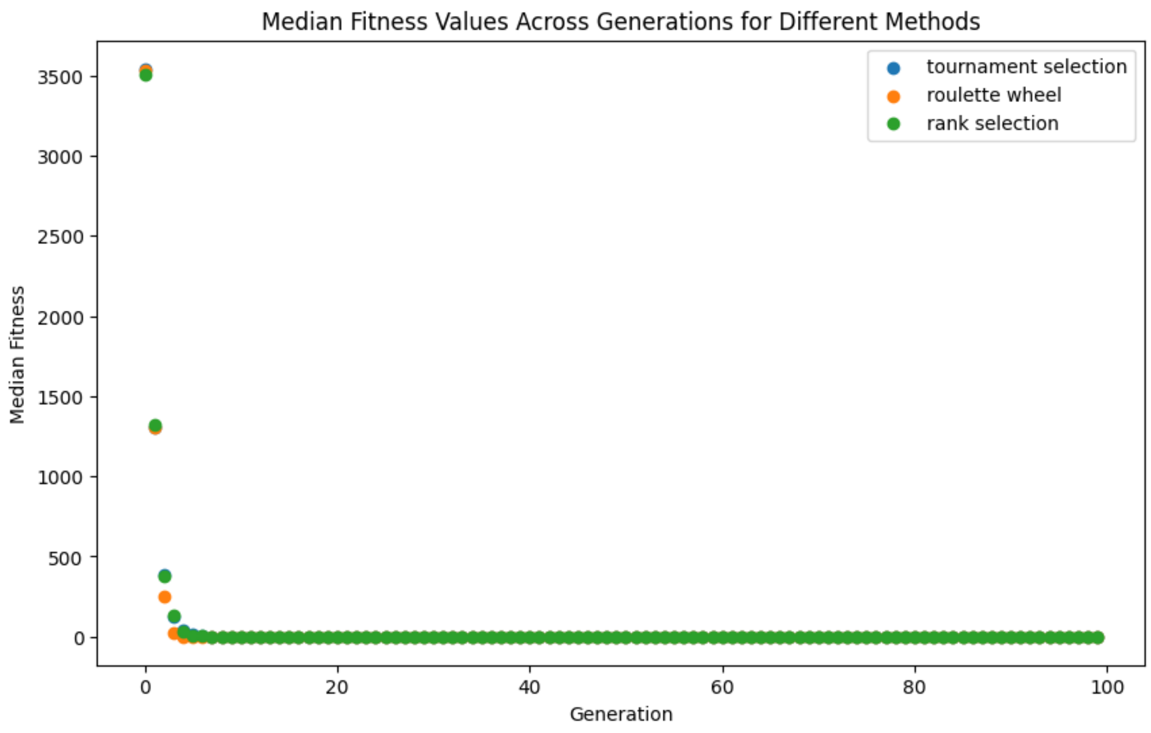


Figure 22 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = tournament_sel, mutate = shuffle_mutation, elitism = True, fit_share=True)

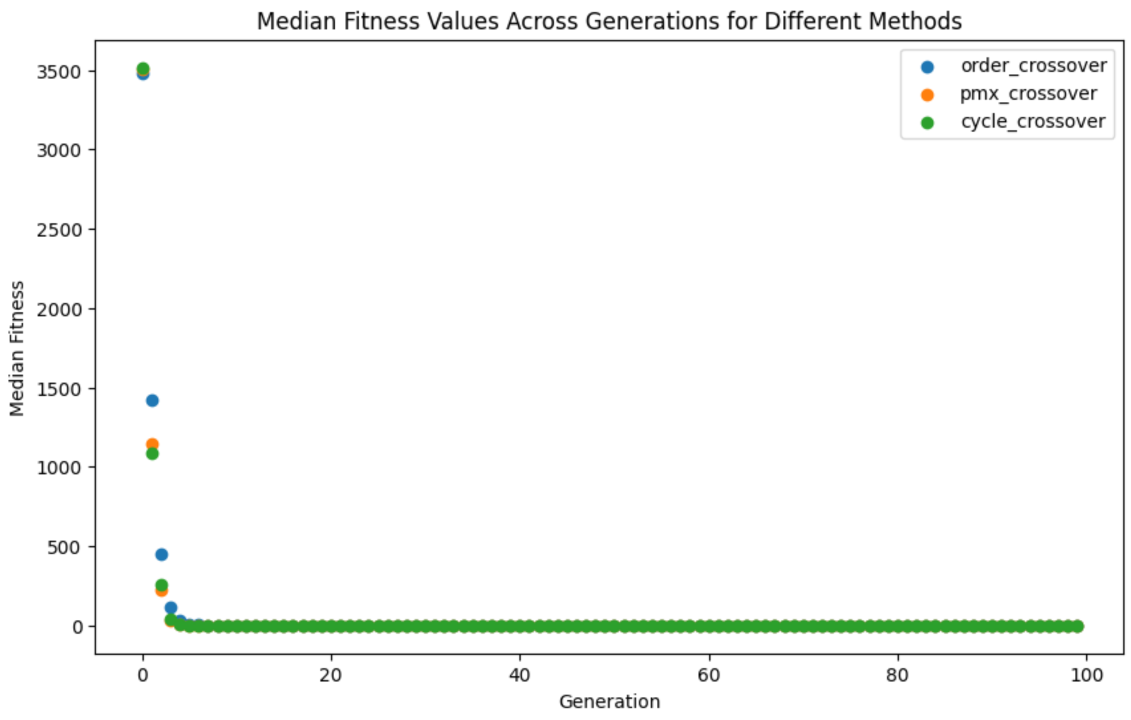


Figure 23 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = rank_selection, mutate = shuffle_mutation, elitism = True, fit_share=True)

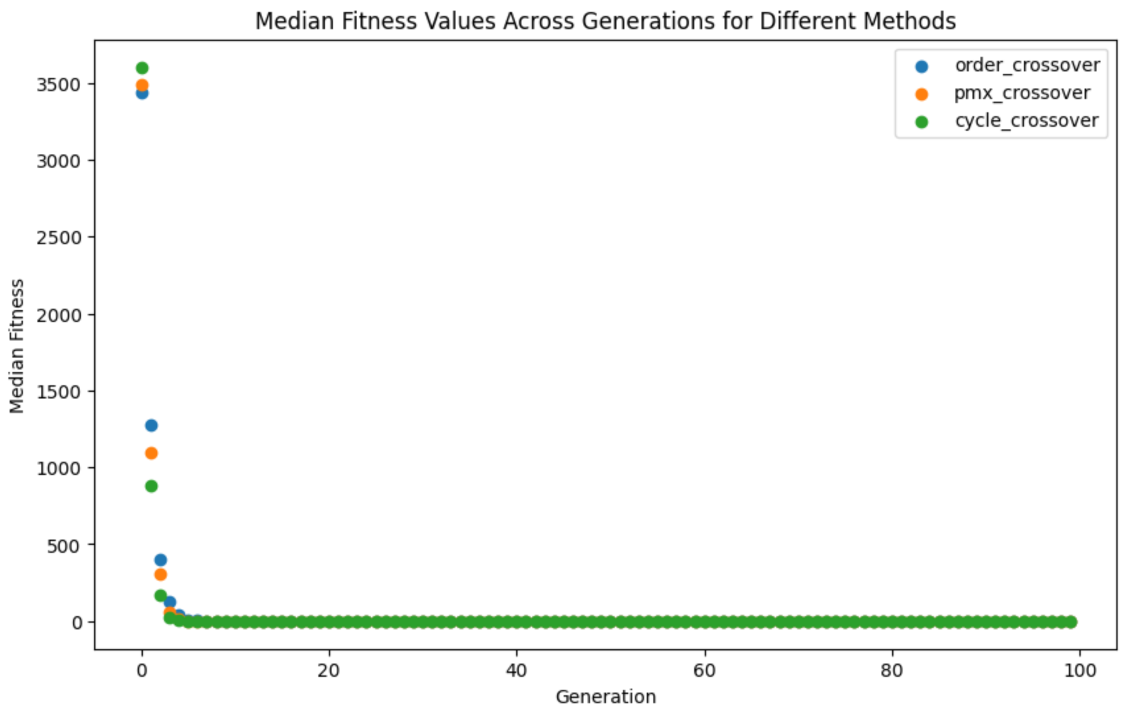


Figure 24 - Median Fitness Values Across 100 Generations from 30 iterations (crossover = crossover_method, selection = fps, mutate = shuffle_mutation, elitism = True, fit_share=True)

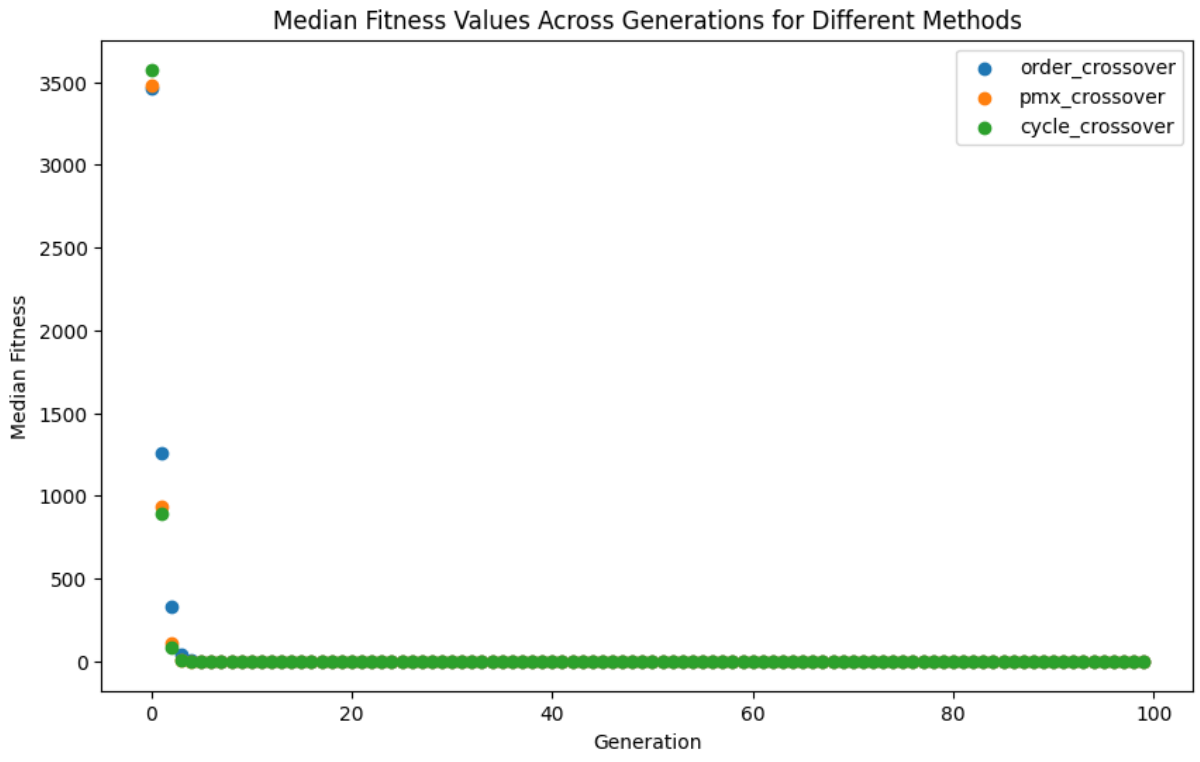
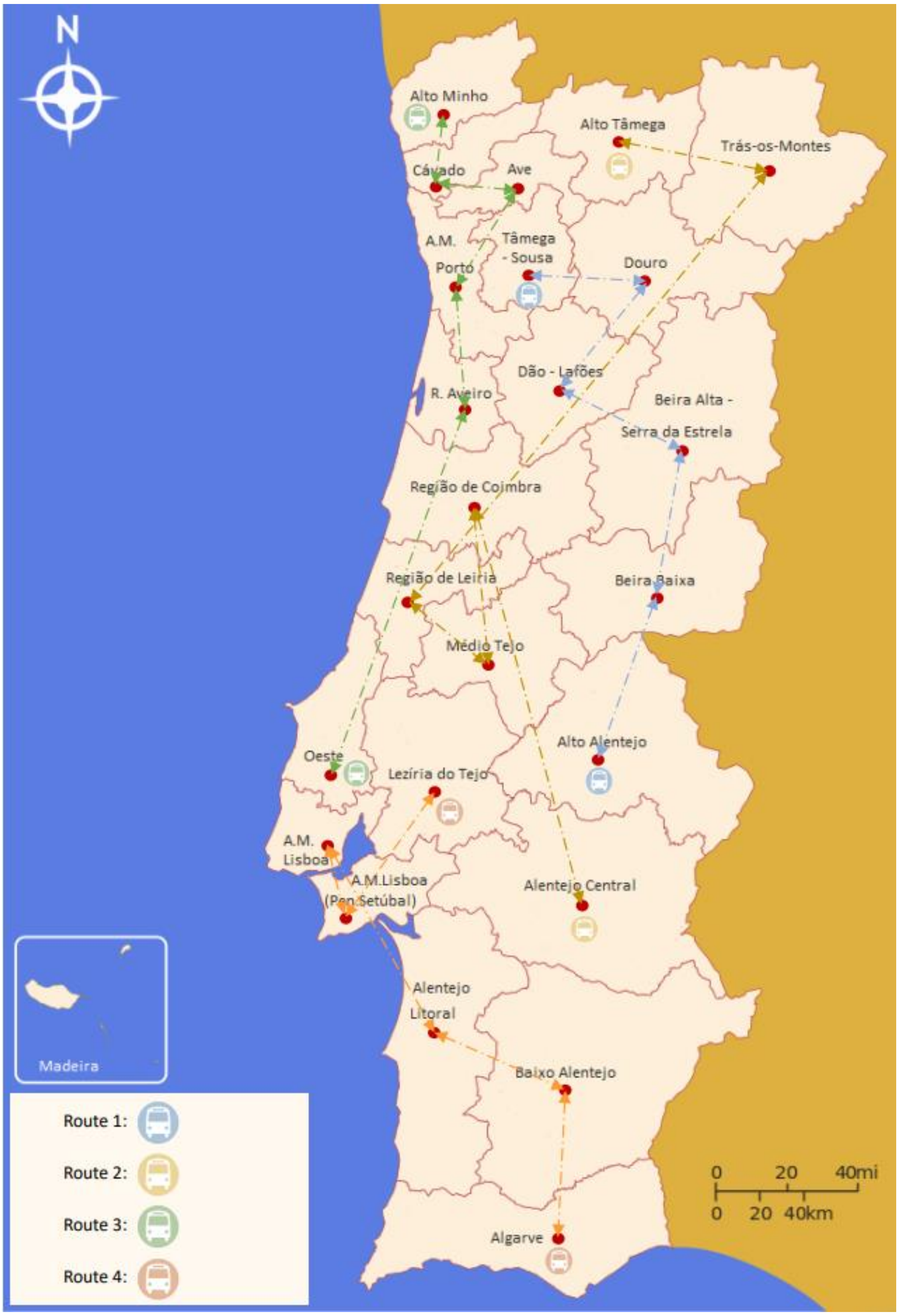


Figure 25 - Final Solution


N
Alto Minho
Alto Tâmega
Trás-os-Montes
Cávado
Ave
A.M.
Porto
Tâmega
- Sousa
Douro
Dão - Lafões
R. Aveiro
Beira Alta -
Serra da Estrela
Região de Coimbra
Região de Leiria
Beira Baixa
Médio Tejo
Alto Alentejo
Oeste
Lezíria do Tejo
A.M.
Lisboa
A.M.Lisboa
(Pen.Setúbal)
Alentejo Central
Alentejo
Litoral
Baixo Alentejo
Algarve
Madeira
Route 1:
Route 2:
Route 3:
Route 4:
0 20 40mi
0 20 40km