Evolutionary Algorithms: Final report

Georgios-Vissarion Foroglou (r1024617)

December 31, 2024

1 Metadata

- **Group members during group phase:** Nikolaos Papageorgiou and Suat Mert Altug
- **Time spent on group phase:** Approximately 10 hours
- **Time spent on final code:** More than 40 hours
- **Time spent on final report:** Approximately 10 hours

2 Changes since the group phase

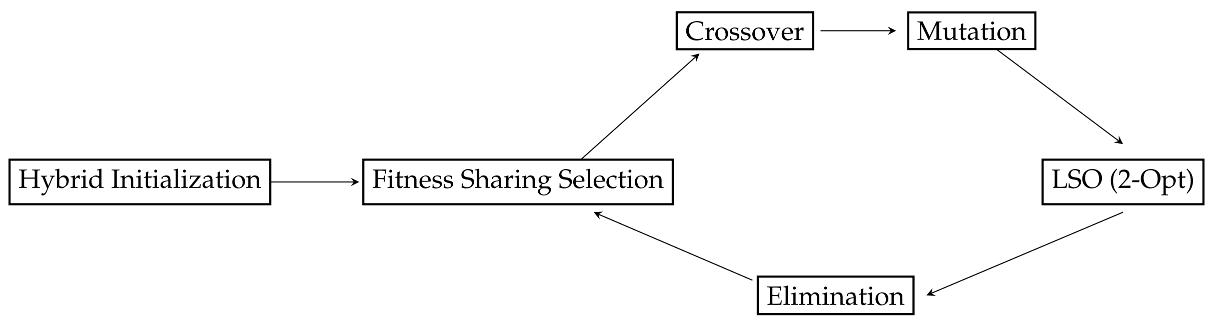
1. **Initialization Mechanism:** A heuristic-based initialization replaced random permutations to handle the complexity of larger datasets. Random permutations were kept for a limited amount of candidates to keep some randomness. Details of the Nearest-Neighbor approach are in Chapter 3.4.
2. **Selection Mechanism:** Ranking-based selection selection was induced with fitness-sharing, significantly improving the evolutionary algorithm. Chapters 3.5 and 3.10 elaborate on this improvement.
3. **Mutation Operator:** Inverse mutation was enhanced with 2-opt as a Local Search Operator (LSO) to address complex data. Chapters 3.6 and 3.9 provide details on 2-opt and its integration into mutation.

3 Final design of the evolutionary algorithm

3.1 The three main features

1. **Fitness Sharing** for Diversity Promotion before the Selection mechanism.
2. **Local Search Optimization (2-opt)** to refine candidates after Nearest Neighbor Initialization and after Inverse Mutation.
3. **Hybrid Initialization**. Combines heuristic and random initialization.

3.2 The main loop



3.3 Representation

After thorough examination and discussion in the group phase, we determined that a permutation-based representation is the most suitable for the Traveling Salesperson Problem (TSP). This representation aligns naturally with the problems requirements: each city must be visited exactly once in a specific order, forming a closed loop. Permutations provide a direct and constraint-free way to represent this sequence. While other representations like binary encoding or adjacency matrices were briefly considered, they introduced unnecessary complexity, such as requiring additional decoding steps or constraints, without offering significant advantages for solving the TSP.

In our approach, each candidate solution is implemented as a 1D numpy array of integers. Each element in this array represents the index of a city in the sequence of the tour, where the ith element corresponds to the city visited at the ith step. To explicitly account for the cyclic nature of the TSP, the last element of the array duplicates the first, ensuring the tour completes as a closed loop. The entire population is represented as a 2D numpy array, where each row corresponds to an individual candidate solution. This design choice simplifies genetic operations such as crossover, mutation, and fitness evaluation, as it provides a structured yet flexible format for manipulating individuals. Moreover, by leveraging numpys computational efficiency, the implementation can handle large populations and complex operations effectively, making it well-suited for tackling larger TSP instances

3.4 Initialization

Efficiently generating feasible solutions is a significant challenge, especially given the variability in datasets. Some graphs are fully connected, while others have cities that are not directly linked, increasing the risk of infeasible solutions. This issue becomes more pronounced with larger problem sizes, making random initialization alone insufficient. To address these challenges, I adopted a hybrid strategy aimed at balancing solution quality and diversity

3.4.1 Nearest Neighbor:

Approximately 80% of the population is initialized using this **Nearest Neighbor heuristic**. Starting from a random city, the tour is constructed by iteratively selecting the nearest unvisited city. This method produces candidates with relatively low initial objective values, offering a strong foundation for further optimization. The quality of heuristic-generated solutions is enhanced using the **2-opt** operator, which improves the tour by iteratively reversing segments to reduce cost. Chapter 3.9 provides detailed insights into the 2-opt implementation and its impact.

3.4.2 Random Permutations:

The remaining 20% of the population is generated randomly. These candidates are created as feasible permutations of city indices, ensuring that they adhere to the problem constraints. The inclusion of random initialization promotes diversity within the population, enabling the evolutionary algorithm to explore the search space more effectively.

3.5 Selection operators

The selection process in our evolutionary algorithm was designed to balance diversity and quality, leveraging a combination of **fitness sharing** and a **ranking-based** selection method. The combination of fitness sharing and ranking-based selection ensures a diverse yet high-quality pool of parents for crossover. Fitness sharing prevents premature convergence by maintaining diversity, while the ranking mechanism ensures steady progress by emphasizing higher-performing individuals.

3.5.1 Fitness Sharing

The first step in the selection process involves computing the **shared fitness** of each candidate in the population. This step ensures that individuals in less crowded regions are given higher effective fitness, encouraging exploration of diverse areas of the search space. Details about the implementation of fitness sharing can be found in Chapter 3.9.

3.5.2 Ranking-Based Selection

Once the shared fitness values are computed, the selection operator ranks the individuals and selects candidates probabilistically for reproduction. I decided to work with **quadratic function** to avoid linearity in later stages. This ensures that higher-ranked candidates are more likely to be selected, but lower-ranked individuals also have a non-zero chance, preserving diversity.

3.6 Mutation operators

For the mutation operation, we chose the **inversion mutation**, a method specifically suited for permutation-based problems such as TSP. According to Chapter 4.5 of the recommended book, permutation problems can be divided into two categories: ”order” and ”adjacency” problems. Since TSP is classified as an adjacency problem, the inversion operator is well-suited due to its respect for the connectedness of the cities, which minimizes the chances of producing infeasible candidates.

The inversion operator works by randomly selecting a sub-tour within a candidate solution and reversing its order. This operator introduces localized changes to the solution, making it a low-randomness mutation method that preserves much of the structure of the original candidate. In our algorithm, if a candidate is selected for mutation (based on a fixed mutation probability), the inversion process is repeated until a feasible solution is obtained. To further refine the quality of mutated candidates, we apply a **2-opt local search** operator after the inversion. This refinement iteratively optimizes the solution by reversing segments of the tour to reduce the total cost.

3.6.1 Tunable parameters

The inversion mutation inherently avoids the need for extensive parameter tuning, with the exception of the **mutation probability**, which determines the likelihood of applying the operation to a candidate solution. This parameter indirectly controls the level of randomness introduced during the evolutionary process. While developing the algorithm during the individual phase of the project, I experimented with a self-adaptive mechanism for dynamically adjusting the mutation rate in two different ways (1. increasing the mutation rate by a small amount each generation, and 2. increasing the mutation rate based on the diversity of the population on each generation). However, the additional computational overhead required for these approaches, especially for high-dimensional problems in the later generations, outweighed its marginal benefits in improving the algorithm's performance. Consequently, I opted for a simpler and more computationally efficient fixed-rate approach.

3.7 Recombination operators

Recombination is a critical operation in evolutionary algorithms, allowing offspring to inherit traits from their parents. For the Traveling Salesperson Problem (TSP), we focused on permutation-based recombination methods, implementing the **Partially Mapped Crossover (PMX)** technique. PMX is widely regarded as suitable for adjacency-based problems like TSP due to its ability to preserve the relative order of cities and maintain feasibility. The resulting offspring inherit key features from both parents while maintaining a valid TSP solution. The algorithm is computationally efficient and does not introduce any infeasible solutions unless the parents themselves are infeasible.

3.7.1 Tunable parameters

The PMX operator does not require parameter tuning or adaptive mechanisms, as its operation is entirely deterministic given the crossover points. However, This approach ensures robust offspring generation while minimizing unnecessary computations.

3.7.2 Comparison with Other Techniques

Initially, we explored another classic crossover operator for TSP: the **Edge Crossover**. Although theoretically appealing for preserving adjacency information, our implementation of the Edge Crossover proved to be significantly more computationally expensive compared to PMX. Additionally, it failed to provide better results in terms of solution quality or population diversity. The challenges stemmed from handling cases where no unvisited neighbors were available during the recombination process, which required complex workarounds that further increased the computational cost.

Given these observations, we decided to proceed with PMX as the primary crossover operator. PMX consistently produced feasible offspring with significantly less computational overhead, making it a more practical and reliable choice for this phase of the project. The simplicity and efficiency of PMX outweighed the potential advantages of Edge Crossover, aligning well with the projects constraints and objectives.

3.8 Elimination operators

For elimination we implemented **k-tournament selection** with **elitism**, balancing the preservation of high-quality solutions with the need for diversity.

More thoroughly, the elimination process begins by identifying the best candidate in the population using an elitism strategy. The objective values of all individuals in the augmented population are computed, and the individual with the lowest objective value is retained as the **elite**. This ensures that the best solution is never lost during the evolutionary process. The remaining individuals are selected using the **k-tournament selection** method.

3.8.1 Tunable parameters

The primary parameter for the elimination process is the **tournament size**, k. Larger values of k make the process stricter, favoring individuals with better objective values, while smaller values introduce more randomness, promoting diversity. In our implementation, k is chosen as a fixed value, but it could be adapted dynamically based on the diversity or convergence rate of the population in future iterations.

Elitism provides a safety net to preserve the best solution, while k-tournament selection introduces a degree of randomness, preventing premature convergence and ensuring exploration of the search space. This combination strikes a balance between exploitation and exploration.

3.9 Local search operators

Local search operators play a pivotal role in refining candidate solutions by exploring the neighborhood of a given solution. For our evolutionary algorithm, we implemented the **2-opt** local search operator, a widely used method for the Traveling Salesperson Problem (TSP). This operator focuses on improving a candidate solution by reversing segments of the tour to reduce the total cost.

The 2-opt operator significantly improved the quality of the candidate solutions. By iteratively optimizing the local structure of the tours, the operator reduced the overall cost of solutions and enhanced convergence. Its ability to produce refined solutions made it a valuable addition to both the initialization phase (for heuristic candidates) and as part of the mutation refinement process.

However, the computational cost of 2-opt increases with the problem size. To mitigate this, we limited the number of iterations and the scope of edge pairs considered. Despite this limitation, the improvements in solution quality justified the computational expense.

3.9.1 Implementation Details

The 2-opt operator works as follows:

1. A candidate solution is taken as input, represented as a permutation of cities with the first and last elements identical to form a closed cycle.
2. For a maximum of 10 iterations (a parameter to control computational cost), the algorithm iterates through pairs of edges in the tour.
3. For each pair of edges, the segment of the tour between them is reversed to create a new candidate.
4. The cost of the new candidate is calculated using the objective function. If the new candidate has a lower cost than the current best solution, it replaces the current best solution.
5. The process terminates early if no improvement is found during an iteration.

To further optimize computational efficiency, we restricted the search space by only considering pairs of edges within the nearest 10 nodes. This trade-off between exploration and computational cost proved effective for our algorithm.

3.9.2 Tunable parameters

The 2-opt operator has one primary parameter: the **maximum number of iterations**, which controls the computational budget for each candidate. In our implementation, this value was fixed at 10 based on heuristic testing.

3.10 Diversity promotion mechanisms

Maintaining diversity within the population is crucial for avoiding premature convergence and ensuring robust exploration of the search space. To achieve this, our evolutionary algorithm employs a **fitness sharing** mechanism, which penalizes individuals that are too similar to others, thereby encouraging diversity. This mechanism significantly enhanced the algorithm's ability to explore diverse areas of the search space. By preventing overly similar individuals from dominating, it reduced the likelihood of the population converging prematurely to local optima. This diversity was particularly beneficial during the early generations, where broad exploration is critical.

3.10.1 Implementation Details

The fitness sharing mechanism operates as follows:

- The objective values of all candidates are calculated using the `obj_fun` function. These values serve as the baseline fitness scores.
- For each individual, a penalty is applied based on its similarity to other candidates. Similarity is measured using the **Hamming distance**, which counts the number of differing positions between two solutions.
- If the Hamming distance between two individuals is less than a predefined threshold, σ_{share}, a sharing penalty proportional to the distance is added. This reduces the effective fitness of individuals in densely populated regions of the search space.

- The adjusted fitness is calculated as:

$$\text{shared\_fitness}_i = \frac{\text{objective\_value}_i}{1 + \text{sharing\_sum}_i}$$

  where the sharing sum aggregates the penalties from all similar individuals.

This approach ensures that candidates in sparse regions of the search space are favored, promoting a more diverse population.

3.10.2 Tunable parameters

The primary parameter for the diversity promotion mechanism is σ_{share}, which determines the threshold for penalizing similarity. A smaller σ_{share} value enforces stricter diversity, while a larger value allows for more overlap between individuals. In my implementation, σ_{share} was fixed after experimentation (value of 15) . I also attempted to incorporate an adaptive mechanism that adjusted σ_{share} based on the population's diversity in each generation. However, this approach proved to be computationally expensive, particularly in later generations for larger datasets. Consequently, I opted for a fixed value, which provided a better balance between efficiency and maintaining sufficient diversity.

3.11 Stopping criterion

There are various stopping criteria that could be employed in our EA, such as detecting convergence through the similarity of the mean and best objective values or analyzing changes across generations. However, given the strict 5-minute runtime limit, I opted for a fixed number of generations, allowing the algorithm to refine solutions within the available time.

3.12 Parameter selection

As I already mentioned self adaptation was considered for a couple of the EAs parameters. However, my implementation proved to be computationally inefficient and thus I opted for fixed values after some trial and error. Table 1 illustrates the final values of the main parameters.

Name	**Value**	**Why this choice**	**Effects of different values**
Pop_size	600	Examples, Trial and error	Larger value increases computation, improves solution quality, makes convergence gradual.
Num_parents	2 * Pop_size	Examples, Trial and error	Increasing causes increase in computation time and faster convergence.
k	3	Lectures, Examples	Larger value increases selection pressure
mutation_probability	0.1	Lectures, Trial and error	Increases diversity but slows convergence
σ_{share}	15	Trial and error	Bigger values allow more diversity in the population
num_generations	1000	Examples, Trial and Error	Increases computation time but smaller population might not suffice for convergence.

Table 1: Table of parameters

3.13 Other considerations

Some other notable mentions of my implementation consider a data preparation mechanism and an additional small diversity promotion mechanism. More thoroughly, the data went through a quick preprocessing in order to replace infinite values with large finite values. This process allowed the EA to learn more efficiently. Regarding the extra diversity promotion mechanism, I implemented a population restarting mechanism that restarted a small fraction of the population every 200 generations. This mechanism was an effort of avoid stagnation and induce some more diversity in the candidate solutions.

4 Numerical experiments

4.1 Metadata

Due to time issues with the deadline as well as the inability of implementing self-adaptation correctly I opted for fixed parameter values for all of the experiments listed below. The values of the parameters are listed in Table 1 and were chosen through trial and error and study of the material. Regarding my computer system, all of the

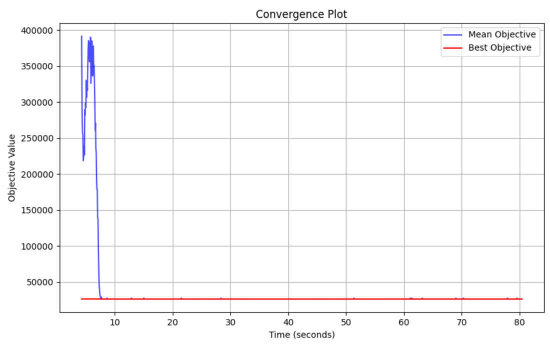


Figure 1: Experiment Results for Tour 50

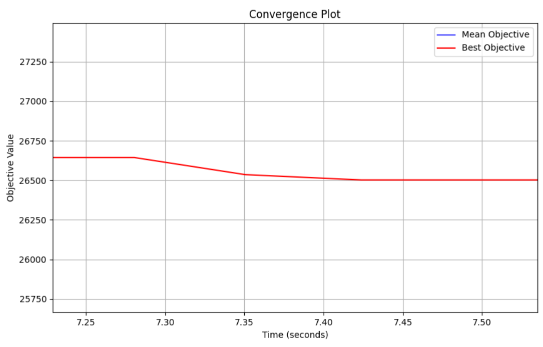


Figure 2: Zoom in at time of convergence

experiments ran in an Apple M1 Pro laptop with 16 GB of RAM, 8 cores for performance and 3.2 GHz clock speed. Regarding python, I'm using Python 3.12.4 and Numba 0.60.0.

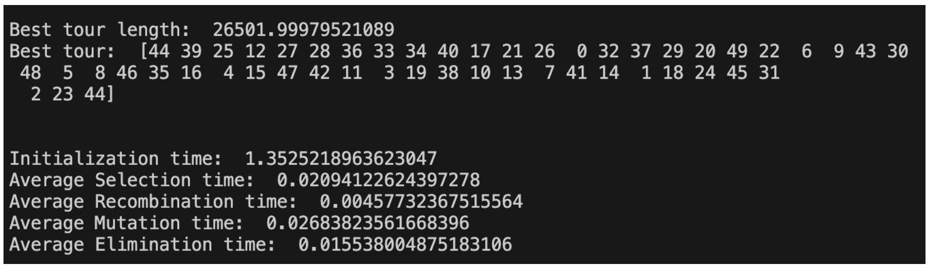
4.2 tour50.csv

4.2.1 Experiment 1 - Single run

This initial experiment involved running the EA on the smallest dataset of 50 cities. Figure 1 illustrates the convergence graph, while Figure 2 highlights the time of convergence to the optimal value. The final best tour length achieved by the EA, **26501.99**, is presented in Figure 3, alongside the sequence of cities comprising the optimal tour. Additionally, Figure 3 displays the time required for population initialization and the average duration of each major EA operation over the 1000 generations.

The heuristic value set as the baseline to beat for this problem was 27723, which the EA successfully surpassed. However, Figure 2 provides additional insights into the algorithm's behavior. The early convergence depicted indicates that the diversity of the population was lost prematurely. Furthermore, the limited improvement from the initial best objective value to the final one suggests that the EA's operations after the heuristic-based initialization were suboptimal.

In terms of speed, the EA performed well on this dataset, completing all 1000 generations within the 5-minute time limit, as expected given the dataset's limited complexity.



```
Best tour length:  26501.99979521089
Best tour:  [44 39 25 12 27 28 36 33 34 40 17 21 26  0 32 37 29 20 49 22  6  9 43 30
 48  5  8 46 35 16  4 15 47 42 11  3 19 38 10 13  7 41 14  1 18 24 45 31
  2 23 44]


Initialization time:  1.3525218963623047
Average Selection time:  0.02094122624397278
Average Recombination time:  0.004577323675155564
Average Mutation time:  0.02683823561668396
Average Elimination time:  0.015538004875183106
```

Figure 3: Experiment Results for Tour 50

4.2.2 Experiment 2 - 500 runs

In this second experiment I ran a similar experiment to that in section 4.3.1, for 500 distinct times. The goal of the experiment was to observe any patterns in the histogram of the 500 runs (Figure 4). Given the early convergence that was detected in Experiment 1 I decided to run the EA for 200 generations in each distinct experiment, for time efficiency purposes.

Regarding the two histograms, the left histogram shows a very narrow concentration of mean fitness values, suggesting that the variability in the population's average performance across runs is minimal. This would be reflected in a very low standard deviation, indicating consistent performance in terms of mean fitness. On the other hand, the right histogram shows more variability, with the best fitness values spread over a broader range. While many runs converged to values close to the optimal, some runs produced slightly worse solutions. The mean and standard deviation for this metric would likely indicate a higher level of variability compared to the mean fitness values.

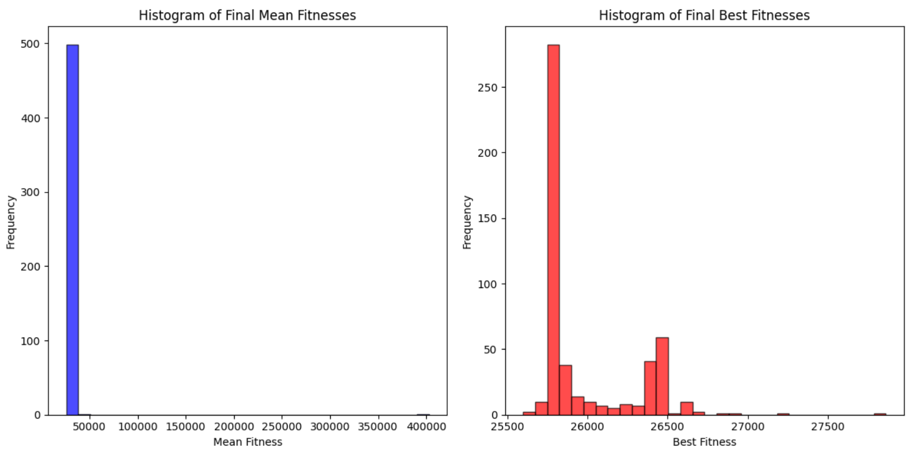
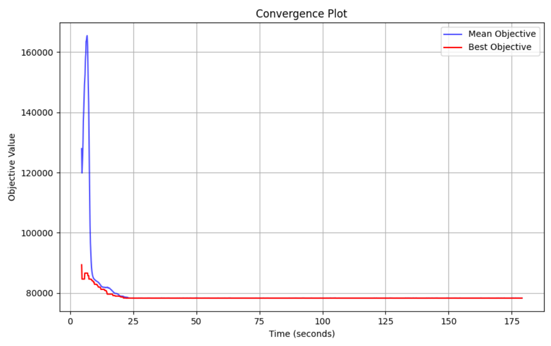


Figure 4: Histogram of Mean and Best Fitness for 500 runs



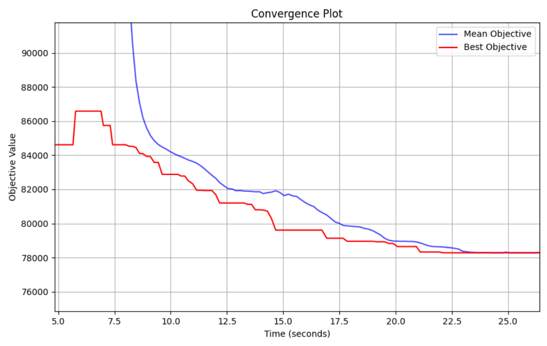


Figure 5: Experiment Results for Tour 100

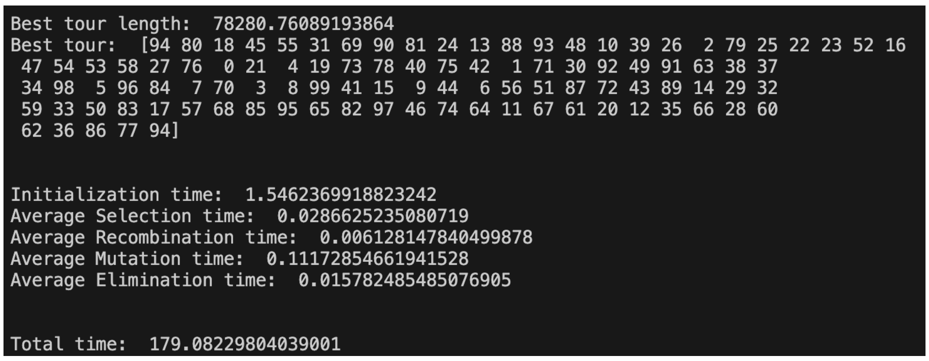
Figure 6: Zoom in at time of convergence

4.3 tour100.csv

This experiment involved running the EA on a dataset of 100 cities. Figures 5 and 6 illustrate the convergence behavior of the EA, while Figure 7 summarizes the best tour length and corresponding tour sequence. Additionally, it highlights the initialization time and the average time spent on key operations across 1000 generations.

The EA successfully achieved a best tour length of **78280.76**, as shown in Figure 7, improving upon the heuristic baseline of 90851. The convergence plots in Figures 5 and 6 reveal that the mean and best objective values stabilized relatively quickly, which indicates again a rapid loss of diversity in the population. This suggests that diversity promotion mechanisms required further refinement to prevent stagnation and provide more exploration capabilities to the EA.

In terms of speed, the EA performed well on this dataset, completing all 1000 generations within the 5-minute time limit, as expected given the dataset's limited complexity.



```
Best tour length:  78280.76089193864
Best tour:  [94 80 18 45 55 31 69 90 81 24 13 88 93 48 10 39 26  2 79 25 22 23 52 16
 47 54 53 58 27 76  0 21  4 19 73 78 40 75 42  1 71 30 92 49 91 63 38 37
 34 98  5 96 84  7 70  3  8 99 41 15  9 44  6 56 51 87 72 43 89 14 29 32
 59 33 50 83 17 57 68 85 95 65 82 97 46 74 64 11 67 61 20 12 35 66 28 60
 62 36 86 77 94]


Initialization time:  1.5462369918823242
Average Selection time:  0.0286625235080719
Average Recombination time:  0.006128147840499878
Average Mutation time:  0.11172854661941528
Average Elimination time:  0.015782485485076905


Total time:  179.08229804039001
```

Figure 7: Experiment Results for Tour 100

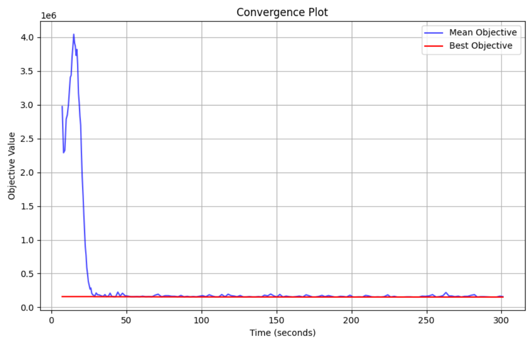


Figure 8: Experiment Results for Tour 500

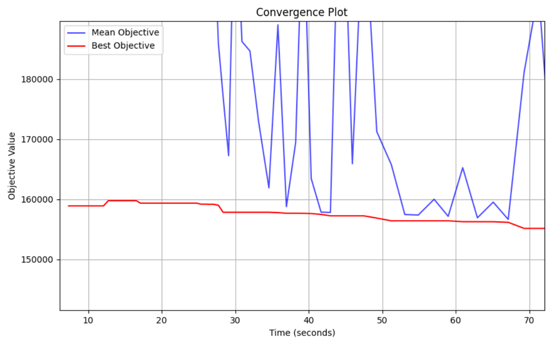


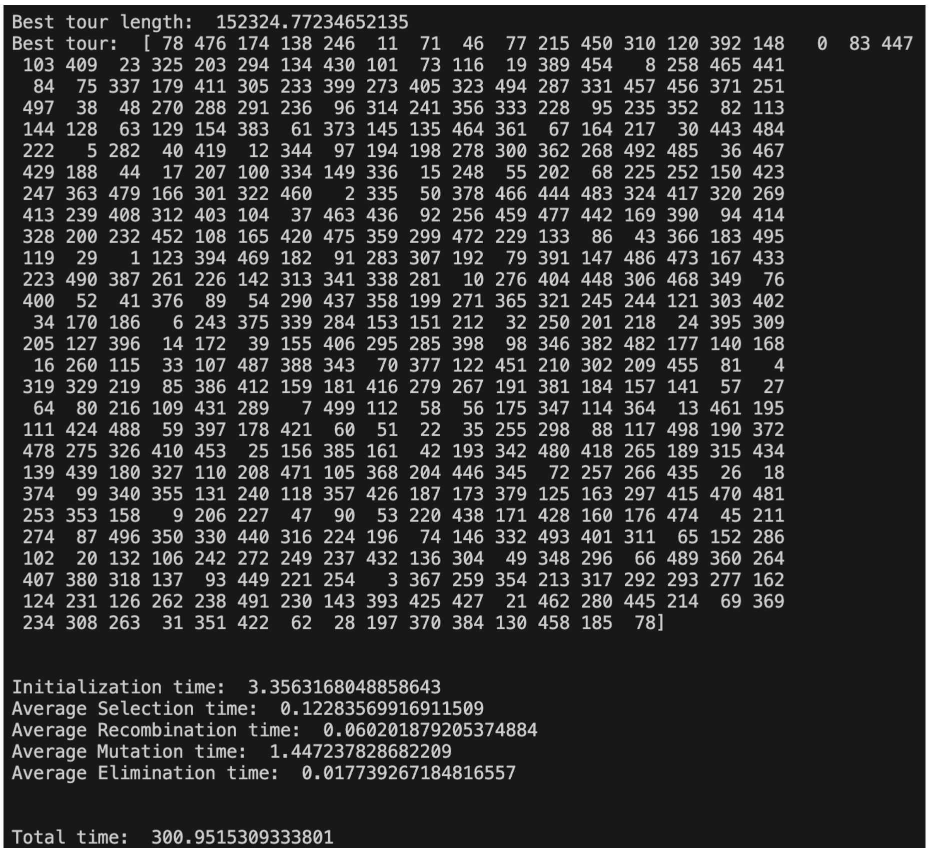
Figure 9: Zoom in at time of convergence

4.4 tour500.csv

This experiment evaluated the EA's performance on a dataset of 500 cities. Figures 8 and 9 provide convergence plots for the mean and best objective values, while Figure 10 summarizes the final results, including the best tour length, sequence, and computational time statistics.

The EA achieved a best tour length of **152324.77**, as detailed in Figure 10, which is better than the target of 157034. While this demonstrates the EA's capacity to handle significantly larger datasets, the convergence behavior revealed once again challenges in maintaining diversity and achieving substantial improvements over generations. More thoroughly, figures 8 and 9 highlight rapid convergence during the initial iterations, followed by stagnation. This behavior suggests a loss of diversity, limiting the algorithm's ability to explore new regions of the solution space.

Regarding speed, the EA didn't manage to completed all 1000 generations within the 5-minute time limit. However, this didn't prevent it from convering in a solution. Notable is the increased initialization time of approximately 3.36 seconds, as well as the average mutation time, which was notably higher. The increased times in these two operations was likely due to the increased complexity of handling larger candidate solutions.



```
Best tour length:  152324.77234652135
Best tour:  [ 78 476 174 138 246  11  71  46  77 215 450 310 120 392 148   0  83 447
 103 409  23 325 203 294 134 430 101  73 116  19 389 454   8 258 465 441
  84  75 337 179 411 305 233 399 273 405 323 494 287 331 457 456 371 251
 497  38  48 270 288 291 236  96 314 241 356 333 228  95 235 352  82 113
 144 128  63 129 154 383  61 373 145 135 464 361  67 164 217  30 443 484
 222   5 282  40 419  12 344  97 194 198 278 300 362 268 492 485  36 467
 429 188  44  17 207 100 334 149 336  15 248  55 202  68 225 252 150 423
 247 363 479 166 301 322 460   2 335  50 378 466 444 483 324 417 320 269
 413 239 408 312 403 104  37 463 436  92 256 459 477 442 169 390  94 414
 328 200 232 452 108 165 420 475 359 299 472 229 133  86  43 366 183 495
 119  29   1 123 394 469 182  91 283 307 192  79 391 147 486 473 167 433
 223 490 387 261 226 142 313 341 338 281  10 276 404 448 306 468 349  76
 400  52  41 376  89  54 290 437 358 199 271 365 321 245 244 121 303 402
  34 170 186   6 243 375 339 284 153 151 212  32 250 201 218  24 395 309
 205 127 396  14 172  39 155 406 295 285 398  98 346 382 482 177 140 168
  16 260 115  33 107 487 388 343  70 377 122 451 210 302 209 455  81   4
 319 329 219  85 386 412 159 181 416 279 267 191 381 184 157 141  57  27
  64  80 216 109 431 289   7 499 112  58  56 175 347 114 364  13 461 195
 111 424 488  59 397 178 421  60  51  22  35 255 298  88 117 498 190 372
 478 275 326 410 453  25 156 385 161  42 193 342 480 418 265 189 315 434
 139 439 180 327 110 208 471 105 368 204 446 345  72 257 266 435  26  18
 374  99 340 355 131 240 118 357 426 187 173 379 125 163 297 415 470 481
 253 353 158   9 206 227  47  90  53 220 438 171 428 160 176 474  45 211
 274  87 496 350 330 440 316 224 196  74 146 332 493 401 311  65 152 286
 102  20 132 106 242 272 249 237 432 136 304  49 348 296  66 489 360 264
 407 380 318 137  93 449 221 254   3 367 259 354 213 317 292 293 277 162
 124 231 126 262 238 491 230 143 393 425 427  21 462 280 445 214  69 369
 234 308 263  31 351 422  62  28 197 370 384 130 458 185  78]


Initialization time:  3.3563168048858643
Average Selection time:  0.12283569916911509
Average Recombination time:  0.060201879205374884
Average Mutation time:  1.447237828682209
Average Elimination time:  0.017739267184816557


Total time:  300.9515309333801
```

Figure 10: Experiment Results for Tour 500

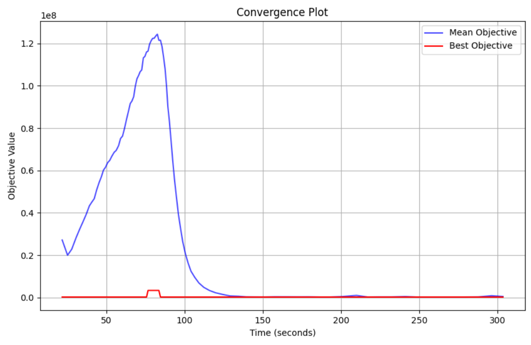


Figure 11: Experiment Results for Tour 1000

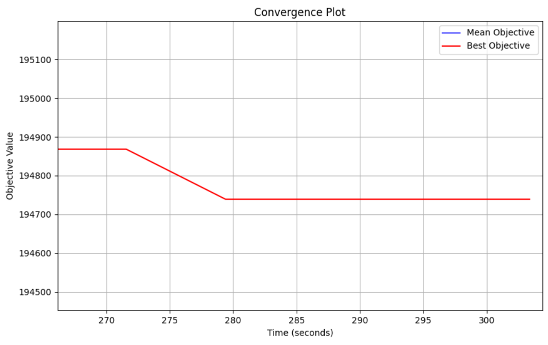


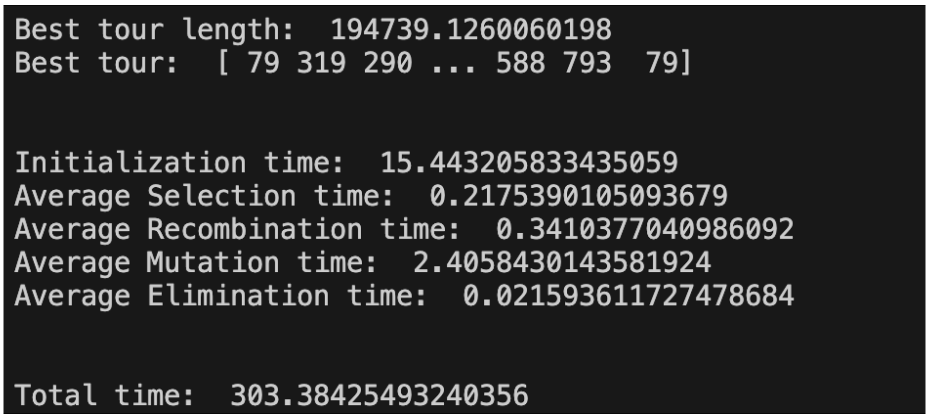
Figure 12: Zoom in at time of convergence

4.5 tour1000.csv

The final experiment evaluated the EA's performance on the largest dataset of 1000 cities. Figures 11 and 12 illustrate the convergence plots for mean and best objective values, while Figure 13 provides the best tour length, sequence, and computational time statistics.

The EA achieved a best tour length of **194739.12**, as shown in Figure 13. While this demonstrates the algorithm's ability to handle extremely large and sparse datasets, the convergence behavior indicates significant limitations in maintaining diversity and efficiency at this scale. First and foremost, the EA identified a valid tour with a reasonable length given the dataset's size. However, the improvements over the generations were minimal, suggesting suboptimal exploration of the solution space. Figures 11 and 12 highlight a rapid drop in mean objective value during the initial iterations, followed by stagnation for the remainder of the run. The limited improvements in the best objective value indicate early convergence and insufficient diversity in the population.

The computational requirements were significantly higher for this dataset. As shown in Figure 13, initialization alone took over 15 seconds, and the mutation operation accounted for the bulk of the computation time, averaging over 2.4 seconds per generation. Given these challenges, the EA completed only 92 iterations within the allotted 5-minute time limit.


```
Best tour length:  194739.1260060198
Best tour:  [ 79 319 290 ... 588 793  79]


Initialization time:  15.443205833435059
Average Selection time:  0.2175390105093679
Average Recombination time:  0.3410377040986092
Average Mutation time:  2.4058430143581924
Average Elimination time:  0.021593611727478684


Total time:  303.38425493240356
```

Figure 13: Experiment Results for Tour 1000

5 Critical reflection

Strengths of EAs:

1. **Flexibility**: EAs are extremely flexible since they can be applied to almost every optimization problem. Additionally, by their nature they are able to be transformed by changing any of their components (mutation, recombination, diversity promotion etc..) in order to fit many different types of problems. Thus, EAs have good generalization.
2. **Ability to adapt to complex spaces**: EAs excel at exploring complex, multimodal solution landscapes. They can explore any solution space while also keeping high levels of exploitation for prominent solutions. Additionally, EAs can avoid easily local minima with no need of fancy mathematical tricks.
3. **Parallelism**: EAs are capable to be parallelized in order to run more efficient. Each of their components

can be parallelized since it is completely independent. The effect of numba in our project is a really good example of this attribute.

Weaknesses of EAs:

1. **Computational Expensive**: EAs are computationally expensive, particularly for large problem sizes. This was evident in the mutation operation for larger datasets, where computational costs significantly increased. Even with the use of Numba some operations were really demanding.
2. **Premature Convergence**: A common weakness of EAs, that I observed, is their tendency to converge prematurely, especially when diversity mechanisms are insufficient. As observed in the experiments, the population often lost diversity early, leading to stagnation in the solution space. Thus, in order to have good and continuous optimization you need to put a lot of effort in keeping the population's diversity high without discarding good individuals.
3. **Multiple Parameters**: The performance of EAs is highly sensitive to parameter tuning, such as population size, mutation rate, and diversity thresholds etc.. Determining optimal values for these parameters requires extensive experimentation and can be problem-specific. All the above, can be probably simplified with sufficient self-adaptation.

This project underscored the immense potential of Evolutionary Algorithms (EAs) in tackling complex optimization problems. Despite the suboptimality of my implementation, it became clear that designing an efficient EA requires significant effort and is highly time-intensive. One key advantage of EAs is their ability to adapt to and converge on acceptable solutions across diverse data types, whether sparse or dense, and even in the presence of missing edges.

For the Traveling Salesperson Problem (TSP), EAs demonstrate a clear advantage over many traditional solvers by finding superior solutions. However, achieving this level of performance requires a substantial investment of time and effort potentially double what might be needed for simpler heuristics, such as the greedy algorithm, which can yield decent results with far less complexity.

In summary, EAs are a powerful and versatile tool for optimization, capable of delivering high-quality solutions to even the most challenging problems. While their implementation demands significant effort, their adaptability and effectiveness make them an invaluable resource for solving a wide range of optimization challenges.

6 Generative AI compliance form

Instances where GenAI was useful:

1. Code skeleton generation for advanced operations and acceleration of debugging (especially for Numba).
2. Ideas generation as inspiration to improve diversity for TSP.
3. Report refinement to fix grammar and vocabulary issues.

Instances where GenAI wasn't useful:

1. Optimization of certain functions and components to avoid bottlenecks. Detecting bottlenecks demands a close inspection of each loop and line of code, something that ChatGPT wasn't able to do.
2. Generation of parameter values, ChatGPT can't understand the effect of each parameter, it can only estimate, thus it could kind of guide you in a good direction but wasn't that efficient.
3. Ideas generation! This is written both in the useful section and here because it was really interesting to see that while the initial ideas generation was good, ChatGPT found it difficult to keep up with the conversation and tune its ideas based on my feedback.

6.1 Code skeleton generation and debugging

Generating the backbone for most of the advanced features like 2-opt and fitness sharing. Debugging and modification of functions in order to comply with numba's strict conditions.

Generative AI model: ChatGPT

Motivation: Acceleration of the whole process. I found it time consuming implemented each individual. While I have kept the most important components and functions in the report, I have tried to implement more than 3 different components for each of the initialization, selection, recombination and mutation mechanism. After spending many days without having any substantial results, not even being able to beat the heuristic values, I decided to ask the help of ChatGPT to see where my implementations were lacking. Regarding debugging, the motivation was again to accelerate the whole process since I was trying to have at least decent results for the report.

Methodology: Most of the times for code generation I copied my code and ask it to write the code for a specific operator. For example, i would attach my PMX function and then I would ask to generate a similar imporved PMX function. Later on, if I wanted to implement another recombination operator I asked it to generate Edge for example based on my implementation of PMX etc..

Postprocessing: Some times the functions were really good and functioned as expected with high efficiency. However, there would be times that I would have to go through some debugging because of bottlenecks or go through some trial and error to determine certain parameters since the model wasn't able to predict what parameters could help (the latter was really comon).

Reflection: To be honest, my interaction with ChatGPT was really useful. While my main motivation was to save up time, I realized that our discussions for ideas generations were extremely fruitful. It was like searching for information in a really sophisticated search engine. I was able to get good ideas then test their effect and play with their parameters on my own. Due to the amount of work we have for all of my 6 courses this semester I wouldn't be able to dive deeper in all these components and parameters that I tested if I didn't have ChatGPT to guide me through the endless possibilities.

6.2 Ideas generation as inspiration to improve diversity for TSP.

Generative AI model: ChatGPT

Motivation: Accelerating of the process and interactive discussion for the expected effect of certain parameters compared to a traditional search engine.

Methodology: Example: "What are 5 good diversity promotion mechanisms I could implement to my EA to improve diversity in later generations and avoid early stagnation.

Postprocessing: I didn't trust blindly ChatGPT since I know about the problem of hallucinations so I had to do some sort of crosschecking with the courses material or a traditional search engine (Google).

Reflection: As I mentioned before, I am happy with ChatGPT's performance since the ideas I got most of the time led me to some decent progress, especially early in the project that I didn't have any progress at all after multiple tries.