Evolutionary Algorithms: Final report

Triet Ngo (r0869104)

December 28, 2021

1 Metadata

- **Group members during group phase:** Triet Ngo, Erick Gomez, Ioannis Mouratidis, and Zsolt Marko
- **Time spent on group phase:** 8 hours
- **Time spent on final code:** 52 hours
- **Time spent on final report:** 9 hours

2 Peer review reports (target: 1 page)

2.1 The weak points

1. Too selective elimination: it was pointed out that we put too much selective pressure in our implementation by using $\lambda + \mu$-elimination.
2. Special treatment of the first city: in our implementation, we fixed and hide city 1. Therefore it could not be used in mutation and combination operators.
3. Redundant use of iteration limit: on top of the time limit in Reporter, we also used an iteration limit in the early phase.
4. Use a more complex initialization scheme than complete randomization.
5. It was pointed out that we should only apply the recombination to a certain group of the population and Order Crossover should be used instead of Partially Mapped Crossover for efficiency purposes.
6. The result of consecutive runs varies dramatically.

2.2 The solutions

1. We indeed put too much pressure by using $\lambda + \mu$-elimination with a big population (roughly 500). But in my final implementation, I chose to keep using $\lambda + \mu$-elimination due to the small population size.
2. It is not special to treat city 1 differently. Since the problem is a closed-loop, having a fixed city reduces complexity and since city 1 is fixed, it is not necessary to change its order. Furthermore, the problem is a cycle, changing other cities also brought change to the order of city 1.
3. I implemented a heuristic initialization scheme called nearest neighbor. Furthermore, due to the small population size, I initialize 10 times the population size and reduce it using diversity promotion to maintain a sparser population. This also helps with the greatly varying result in our group phase.
4. I implemented Order Crossover for more efficiency after evaluating both recombination methods.

2.3 The best suggestion

The suggestion I found most helpful was addressing the initialization of our group phase algorithm. It was suggested that we perform clustering to promote our initial population based on distances between tours. This helps avoid having a non-existing edge (infinite distance) and gives the algorithm a greater start.

While I did not implement the suggested algorithm, the insight regarding the non-existing edge was extremely helpful. It also indeed gave my algorithm a greater head start after I implemented the heuristic initialization scheme. In the group phase, we could not even find a valid tour due to the large city size, but bringing invalid cities right from the start gave the algorithm more opportunity to explore. This also matters greatly since, in the large city sample, our algorithm can only run a few iterations.

3 Changes since the group phase (target: 0.5 pages)

1. Changing Partially Mapped Crossover to Order Crossover.

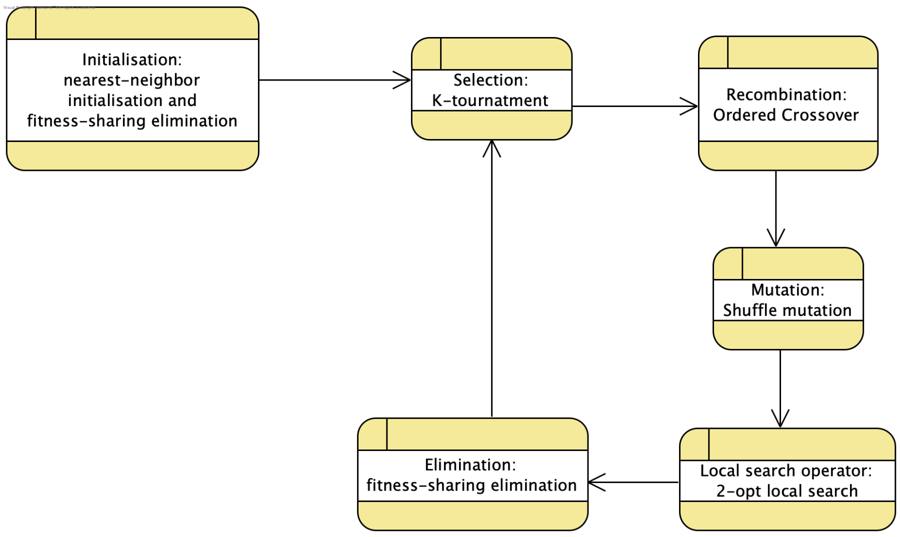
2. Changing Swap mutation with Shuffle mutation.
3. Implementing 2-opt (and its variations) local search operator.
4. Implementing diversity promotion: fitness-sharing in the elimination step.
5. Implementing heuristic initialization: nearest neighbor and fitness-sharing elimination.
6. Double the running time for large samples using multi-processes.

4 Final design of the evolutionary algorithm (target: 3.5 pages)

4.1 The three main features

1. Heuristic initialization with nearest neighbor and fitness-sharing.
2. 2-opt local search operator and its variants boosted with parallel computing for large instances.
3. Diversity promotion scheme such as fitness-sharing in initialization and elimination.

4.2 The main loop


Initialisation: nearest-neighbor initialisation and fitness-sharing elimination
Selection: K-tournatment
Recombination: Ordered Crossover
Mutation: Shuffle mutation
Local search operator: 2-opt local search
Elimination: fitness-sharing elimination

4.3 Representation

There are two main options for representing the order of visiting cities in the Traveling Salesman Problem. The one I chose for my implementation is path representation. In path representation, the cities are listed in the order of visiting. The advantages of using path representation are:

- It is compatible with the mutation operators (swap, shuffle) and recombination operators (PMX, ordered crossover).
- It is easier to fix and hide the first cities in the representation without affecting the algorithm.
- Equivalent cycles are easier to detect in path representation.

Another popular choice for representing the Traveling Salesman Problem is cycle representation. In cycle representation, each position in the representation contains the next city that will be visited. There are a few downsides of this representation:

- Equivalent cycles require additional checks. For example (2 3 4 1) and (3 4 1 2) represent the same path but require additional checks in order to confirm.
- It is not easy to fix and hide the one city in the cycle representation.
- Recombination and mutation operators often break the cycle into sub-cycles which require additional work to re-attach the sub-cycles together.

These make the cycle representation more error-prone and have a higher complexity than the path representation.

In my implementation of path representation, I used a Numpy array with length N - 1 to represent the order of visiting cities from 2 to N. By fixing the first city, it will reduce the search space and complexity of the algorithm.

4.4 Initialization

Initialization is one of the defining factors of a good evolutionary algorithm. In our group phase, we used random initialization which was easy to implement but could not solve large instances due to randomness can choose invalid edges (infinity edges). Therefore in the individual phase, I tried out a few advanced initialization mechanisms such as nearest-neighbor [2], validation of randomness path.

I started to improve on the group phase complete randomness path generation by validating the path afterward. If the path is valid then it is chosen into the initial population. The result was more promising than complete randomness but was shown to be significantly inferior to heuristic algorithms such as **nearest neighbor**.

The reasons for using a heuristic algorithm of nearest neighbor are:

- Among heuristic algorithms such as Nearest Neighbor, Convex hull or Minimum spanning tree (Prim Algorithm), etc [2]. Nearest Neighbor is the easiest one to implement and validate.
- While heuristic algorithms do not give the optimal solutions but considering the **simplicity**, **low complexity** of nearest neighbor, it is fitted to be chosen as a path generator for the initial population.

The nearest neighbor algorithm can be summarized as follow:

1. Initialize by choosing a random city as starting city.
2. Select from the remaining cities the city that is closest to the previously chosen city.
3. Insert the city to the tour
4. If the tour is not complete, go back to **step 2**

In my implementation, I chose a relatively low population size (N = 40). Therefore, the nearest neighbor alone cannot ensure the diversity of the initial population. In order to balance the time-complexity of the initialization step and the diversity of the initialization, the algorithm starts by generating **10 times** the number of the population then apply an elimination scheme using fitness-sharing.

4.5 Selection operators

In this implementation, I preserve our group phase **k-Tournament** selection. k-Tournament is the ideal choice regarding complexity and efficiency. It does not suffer from early convergence or vanishing fitness pressure of similar candidates such as Fitness Proportional Survival and Ranking selection.

In k-Tournament, we can determine the selective pressure by manipulating the k value. Due to the small population size, a fixed value of k is chosen which is **2**.

4.6 Mutation operators

In the group phase, we implement the swap operator but it is shown to be inefficient in introducing adequate randomness to the algorithm. Therefore, in the individual phase, **shuffle mutation** is implemented as a replacement for the swap operator.

Having a small population with a diversity promotion scheme is problematic since the candidates are supposed to be sparse. The optimal candidate is usually **unique (only one best solution)** compared to having the population converge to one or several local optima. Therefore, having a fixed mutation rate is problematic since it can change the best candidate to an unwanted one as well as does not apply enough mutation to the bad candidate so it can explore more. Therefore, **self-adaptive mutation rate** is used, in each iteration, every candidate with fitness values that are **smaller than the average fitness of the population will have its mutation increase and vice versa**.

4.7 Recombination operators

There are two popular recombination operators for permutation and the Traveling Salesman problem: **Partially Mapped Crossover (PMX) and Order Crossover (OX)** [1] [3]. According to Larranaga, a variation of OX is shown to have better results than PMX [3]. Both algorithms start by **perserving a sub-tour** from the parents but PMX preserves the absolute positions of elements in both parents, while OX focuses on transmitting the relative order of the parents. Therefore, OX introduces more exploration to the algorithm which is better than PMX since we also implement a local search operator which heavily improves the exploitation aspect of the algorithm.

4.8 Elimination operators

The elimination step preserves the population size by applying a function/filter to the population and its children combining them into the population for the next iteration. Two common schemes are: $(\lambda + \mu)$-elimination and (λ, μ)-elimination. It is recommended to use (λ, μ)-elimination, when the children size is significantly higher than the population size $\mu/\lambda = 5$ citeIECBook. Since this implementation consists of small population size, low children size are not much larger than the population size, and a diversity promotion scheme, it is not appropriate to use (λ, μ)-elimination.

The $(\lambda+\mu)$-elimination is included with **fitness-sharing** (details of fitness-sharing in section 4.10). This ensures the diversity of the population while perserving the "featured" candidates.

4.9 Local search operators

Local search operators are functions that aim to slightly improve the candidate by scouting in its neighborhood for a better solution. In this assignment, I implemented 1-opt, 2-opt, and its variants, 3-opt. While 1-opt is time-efficient, it is inferior to 2-opt*. 3-opt has a wider search range but its complexity is significantly high making it unfit for large instances. Therefore, 2-opt operators are implemented in the algorithm. In the 2-opt operator, we iterate through the cities and exchange a pair of cities to find a better solution than the candidate. Three variants of its implementation are:

- A full-scale 2-opt.
- 2-opt but halt at first improvement.
- 2-opt with smaller neighborhood (10% of the neighborhood for each city).

A full-scale 2-opt is the most optimal for TSP since it will most likely to **find the optimal** value given the neighborhood of the candidate but its **time complexity dramatically increase** for large instances with the number of cities higher than 500 which significantly reduce the iterations and the algorithm explorative aspect.

One solution for this problem is **halting the local search at the first improvement**. In other words, the local search will ensure to find a better candidate (if exists) but at the same time reduce the complexity when searching for one. This solution is significantly faster than the full-scale 2-opt but its improvement is inferior.

Another solution for the problem is to introduce both **randomness and a smaller search-space** of the 2-opt operator to the algorithm. For each city, only swap it with 10% of the remaining cities which reduces the complexity by 10 times but also allows the local search operator to explore more in the neighborhood instead of stopping at first improvement. This solution seems to be the balance of the above two variants above and is **used as a default local search operator for large instances (N $\geq$ 300) while its full-scale version is used for the smaller instances**.

4.10 Diversity promotion mechanisms

In our group phase, we faced two main problems: rapid convergence (low diversity in the population) and low quality of the candidates due to infinite edges. The later problem is solved by implementing heuristic initialization but the algorithm seems to only converge to a local optimum instead of keeping various local optimums which reduces the explorative aspect of the algorithm.

Therefore, I decided to implement **a diversity promotion scheme (fitness-sharing)** for my algorithm. Fitness-sharing is used in the **elimination step and initialization**. The purpose of fitness-sharing is to promote diversity by applying a penalty to neighboring candidates when choosing them into the final population.

The effect of fitness-sharing is shown in the figures in section 5, in all the figures, **the mean objective value remain relatively higher than the best object value**. Therefore, the population does not converge at a local optimum but various ones and in turn promote the explorative aspect of the algorithm, helping it find a better solution.

For fitness-sharing, determining the neighborhood or rather the **distance between two candidates** is important since it affects the complexity and quality of fitness-sharing. There are two candidates for this: **transposition distance and hamming distance**. While transposition distance is many representatives as the distance between permutation, it is an NP-hard problem. Therefore, a **safer choice is hamming distance**.

4.11 Stopping criterion

In an evolutionary algorithm, the algorithm should stop when it finds the global optimum. Since this is usually not the case with evolutionary algorithms due to time limits and the tendency of stepping in local optimum, stopping criteria are introduced to stop the algorithm to run more than it should for small instances and allow large instances to run in limited time.

To balance between enhancing the solution candidates and time constraints. **A combination of time-limit (default) and improvement of best solution candidate are chosen**. The algorithm will stop after 5 minutes or 50 iterations without improvement in the best solution candidate.

4.12 Parameter selection

While having a large number of parameters and hyper-parameters, I did not have enough time to perform a hyper-parameter search for all of them. But there are a few parameters that I focused on:

- Population size.
- Value of k-Tournament.
- Default mutation rate and mutation step in self-adaptive mutation rate.
- α and neighborhood size in fitness-sharing

In general, I selected a few value instances from the parameters and run the algorithm with all the instances a few times then observed: **the diversity of the population, quality of the best solution, the running time**. The one with the most balancing features will be chosen.

For population size, I tried to balance between running time, the diversity of the population as well as the quality of the solution. The values of **20, 30, 40, 50, 70, 100** were chosen as candidates for the population size. For values that are larger than 50, the number of iterations for large instances is very limited and for values that are smaller than 40, the diversity of the population and the quality of the algorithm decreased significantly. The same principles are used to determine the rest of the parameters.

4.13 Parallel Computing

While measuring parts of the algorithm, **nitialization and local search operator takes** the most time for large instances (nearly 10 to 20 times of the other parts). An observation for local search operator and generating paths with the nearest neighbor) is that it **can be performed in parallel (apply local search operator in a different segment of the population) or generating paths in parallel**. Therefore, multi-processes in Python are implemented to speed up the local search operator and initialization. Since the given CPUs will have 2 cores, only 2 instances of multi-processes will be initiated. Last but not least, **starting and merging multi-processes also take time**, and it is shown to be **slowing the small instances rather than reducing the running time by half in large instances**.

5 Numerical experiments (target: 1.5 pages)

5.1 Metadata

The parameters in this implementation are:

- Population size = 40
- Children size = 80
- k in k-Tournament = 2
- Initial mutation rate = 0.0
- Adaptive mutation step = [0.4-0.5]
- Alpha in fitness-sharing = 2.0
- Neighborhood limit in fitness-sharing = 10
- Number of maximum generated paths with nearest-neighbor = 400
- multi-threading and 2-opt with smaller neighborhood for instance with N $\leq$ 300

The characteristics of my computer:

- Apple M1 Pro (Only use 2-efficiency cores with clock speed at 600 - 2064 MHz)
- 16GB of unified memory
- Python 3.9.8

5.2 tour29.csv

For the tour29.csv instance, the best length I could find is **approximately 27200** in figure 2. But **the most common one is with length 27778** with the following city sequence 1, 11, 10, 12, 15, 19, 18, 17, 21, 23, 22, 29, 28, 26, 20, 27, 25, 24, 16, 14, 13, 9, 7, 3, 4, 8, 5, 6, 2.

While I think the common solution of the algorithm is not near to the optimal one, the best solution approximately 27200 is relatively close to the optimal solution.

Based on the convergence graph for tour29.csv (figure 1), it is shown that:

- The **diversity of population is perserve** in the run (relatively higher than the best objective)
- The best solution is halting after a first few iterations when it **steps into the local optimum**. But after several iterations, due to the **promoting exploration by self-adaptive mutation rate**, it can reach other regions of the search space to find a better solution.
- The running time for the algorithm is **approximately 30 seconds**. While I believe it can still be improved based on figure 2, it is **better stop earlier** than constantly relying on the mutation to find better value until the time limit ran out.

After running the solution 1000 times, it is recorded that:

- The mean of the mean objectives: 29618.98592731305
- The standard deviation of mean objectives: 518.9131454044993
- The mean of the best objectives: 28102.765502493625
- The standard deviation of best objectives: 312.90054786145845

This shows while having deviation in between runs, the fluctuation of solutions is much smaller than the ones in the group phase (standard deviation of 500 versus standard deviation several thousand).

5.3 tour100.csv

While having similar features to tour29.csv. Based on the convergence graph for tour100.csv (figure 3), it is shown that:

- The **diversity of the population is perserve** in the run (relatively higher than the best objective)
- The best solution is halting after a first few iterations when it **steps into the local optimum**. But after several iterations, due to the **promoting exploration by self-adaptive mutation rate**, it can reach other regions of the search space to find a better solution.

- While it is similar to tour29.csv in terms of behavior. The **number of a small improvement in best candidate solution** is more frequent due to larger search space.

The **best solution** for tour100.csv that I can find is 252061.

5.4 tour500.csv

The differences of this instance from the previous one is more noticeable. Based on the convergence graph for tour500.csv (figure 4), it is shown that:

- The **diversity of the population is perserve** in the run (relatively higher than the best objective).
- The number of iteration is dramatically decreased.
- The best solution takes a **longer to reach the local optimum**. This is due to the switching from a full-scale 2-opt to its 10% version and the instance has a much larger search space.
- The run is **stopped by time limit** instead of the deviation stopping criterion. While it continues to improve and explore, we can sometimes see sharp improvement due to mutation on top of gradual improvement with the local search operator.

The **best solution** for tour100.csv that I can find is 120710. While I do not think this is close to the global optimum, I believe that this is the best local optimum that the algorithm can find within the time limit (it halts for the last 5 iterations).

5.5 tour1000.csv

While having similar features to tour500.csv. Based on the convergence graph for tour1000.csv (figure 5), it is shown that:

- The **diversity of the population is perserve** in the run (relatively higher than the best objective)
- The number of iteration is slightly decreased.
- The run is **stopped by time limit** instead of the deviation stopping criterion. While it continues to improve and explore, we can sometimes see sharp improvement due to mutation on top of gradual improvement with the local search operator.
- The algorithm **could not find the best local optimum** within the time limit. Meaning if it is given more time, it can continue to improve its solution and we can we that by the end of the run, the best objective is not stabilized.

The **best solution** for tour100.csv that I can find is 223520 and I do not think it is the global optimum.

6 Critical reflection (target: 0.75 pages)

In my experience, the **three main strengths** of evolutionary algorithms are:

1. Evolutionary algorithms can find a relatively good solution (better than the heuristic ones) for a suitable optimization problem within a time limit.
2. The algorithms are straightforward, easy to adjust and modify to fit the optimization problem compared to some specialized algorithms that are designed and developed for a specific problem.
3. Unlike theory-based algorithms that are developed for a long time, a robust, black-box, without prior knowledge of the problem such as an evolutionary algorithm, can be used to solve various problems and give a relatively good solution.

The **weak points** of evolutionary algorithms are:

1. Early convergence to a local minimum can be a problem if the goal of the user is to find the global optimum. An evolutionary Algorithm cannot prove that its solution is a global optimum, unlike math-based algorithms.
2. The time complexity of the algorithm is directly coupled with the algorithms it used and the size of the problem. With large instances, sometimes the complexity can grow quadratic, cubic, but some other specialized methods can use advanced parallel computing and utilize that with nowadays powerful GPU and TPU.
3. While it is simple and robust, an evolutionary algorithm is not reliable when applied to problems like theory-based algorithms that are specifically designed for a given problem.

There are a few lessons that I learned in this course:

- Evolutionary Algorithm is a simple, beginner-friendly, robust, good approximation global optimization algorithm for problems such as The Travelling Salesman.
- While it is simple to implement an Evolutionary Algorithm, it is not sufficient to archive better solutions than the heuristic solutions. By constantly learning, observing the behaviors of the algorithms, I ended up with a much better solution than the one in the group phase.
- Initialisation is extremely important in sufficiently promoting the diversity as well as reducing the search space. It becomes more significant in large instances.
- While local search operators can take time, it is better to include its variants that reduce the complexity. A local search operator is a very good tool to explore the neighborhood of the candidate solutions that are almost guaranteed to yield a better solution in one iteration that sometimes can take several iterations to archive.
- Diversity promotion scheme is crucial in maintaining a diverse population which will have a better chance of finding a better solution instead of premature converge at a local minimum.
- I would like to explore more especially in parameter search but due to the large workload from other courses. I have to prematurely stop the improvement of my project.

7 Appendix

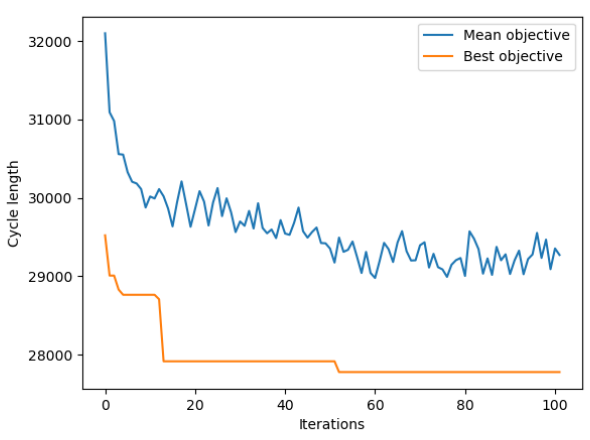


Figure 1: Convergence graph for tour29.csv

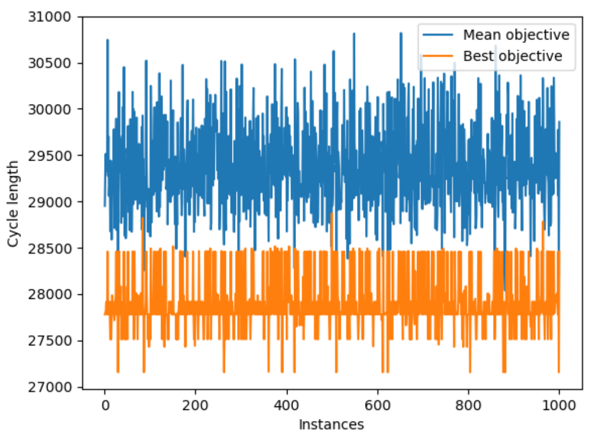


Figure 2: Histogram of 1000 runs for tour29.csv

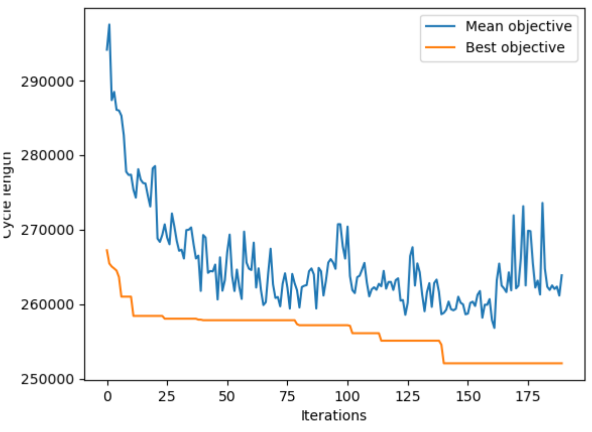


Figure 3: Convergence graph for tour100.csv

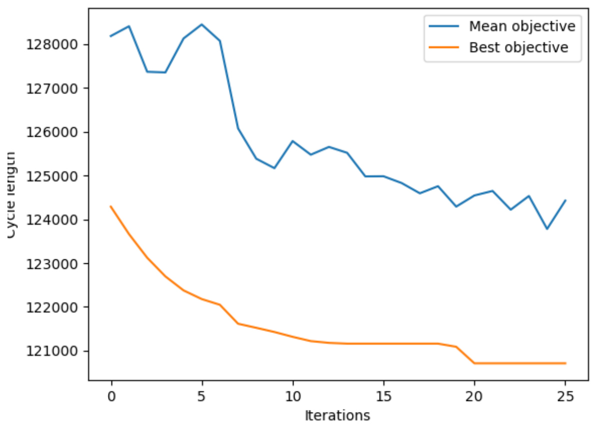


Figure 4: Convergence graph for tour500.csv

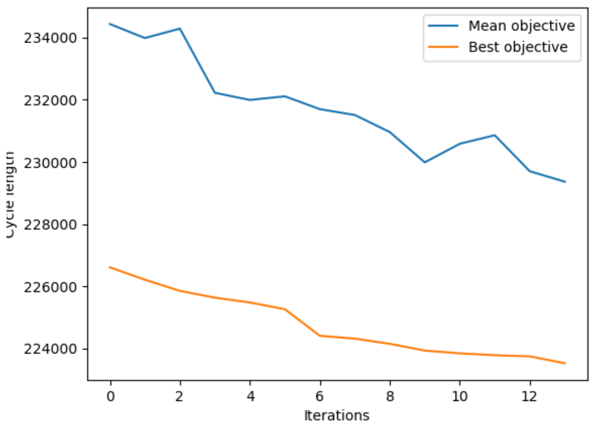


Figure 5: Convergence graph for tour1000.csv

References

[1] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003, pp. 1–300. ISBN: 978-3-662-05094-1.

[2] David S. Johnson and Lyle A. McGeoch. “The Traveling Salesman Problem: A Case Study in Local Optimization”. In: 2008.

[3] Pedro Larranaga et al. “Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators”. In: *Artificial Intelligence Review* 13 (Jan. 1999), pp. 129–170. DOI: 10.1023/A:1006529012972.