Evolutionary Algorithms: Final report

Yash Mahesh Mathradas (r0823209)

September 8, 2022

1 Metadata

- **Group members during group phase:** Cristian Frigolett and Sofie Landuyt
- **Time spent on group phase:** 10 hours
- **Time spent on final code:** 40 hours
- **Time spent on final report:** 10 hours

2 Changes since the group phase

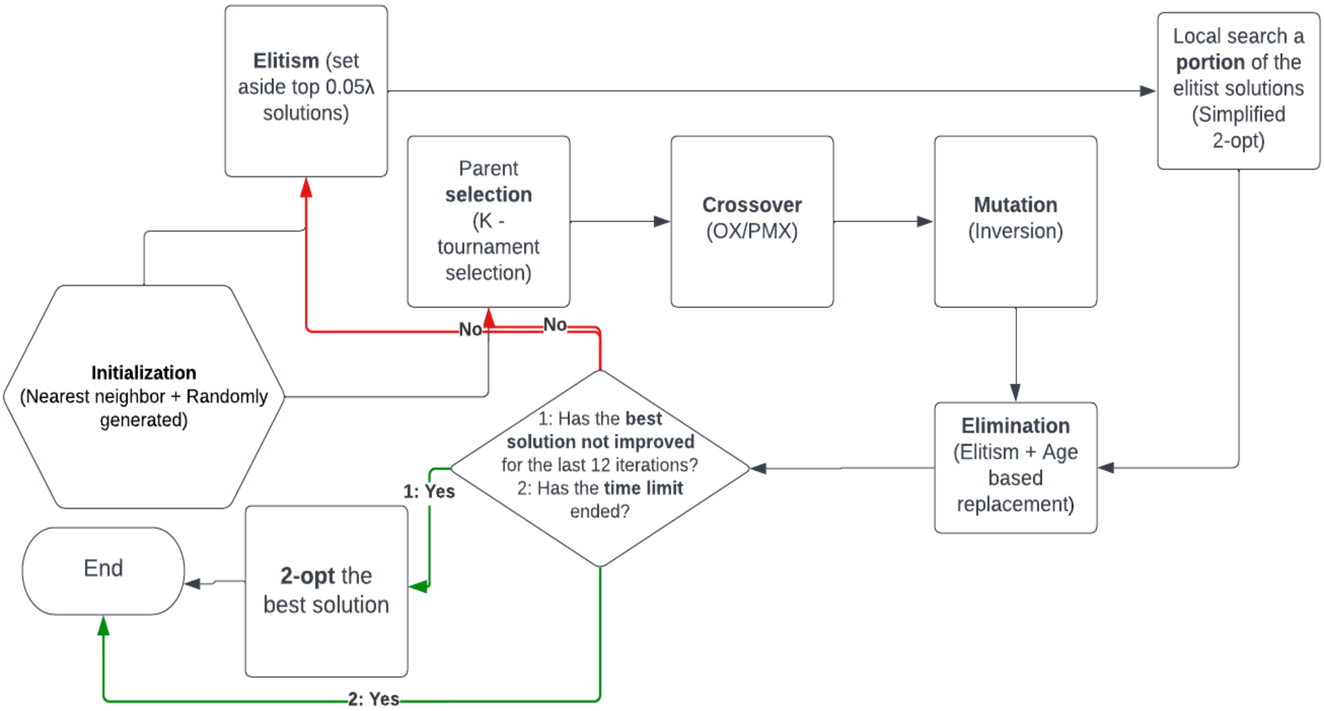
1. Added a mutation operator, which previously was an extremely simplified and preliminary version of a local search operator.
2. Replaced the swap ”mutation” with the inversion mutation operator.
3. Added the order crossover operator (OX) to work in combination with the existing PMX operator.
4. Added the nearest neighbor heuristic to initialize a portion of the starting population size.
5. Changed the hyperparameters and added parameter control mechanisms for some of them.
6. Added a simplified version of the *2-opt* local search operator. Also added the original *2-opt* to run on the best candidate, if time permits.

3 Final design of the evolutionary algorithm

3.1 The three main features

1. The *nearest neighbor* heuristic to initialize a portion of the starting candidates: This provides good solutions to improve the search space.
2. The simplified version of the *2-opt* local search operator: This is not as computationally intensive and effective as the original *2-opt*, but still is able to push solutions toward the optima in their vicinity whilst being fast.
3. The mix of dual crossover operators. I.e., order crossover (OX) for 80% of parent sets and partially mapped crossover (PMX) for 20% of the parent sets creates a diverse set of offspring solutions.

3.2 The main loop


Elitism (set aside top 0.05λ solutions)
Local search a portion of the elitist solutions (Simplified 2-opt)
Parent selection (K - tournament selection)
Crossover (OX/PMX)
Mutation (Inversion)
Initialization (Nearest neighbor + Randomly generated)
No
No
1: Has the best solution not improved for the last 12 iterations? 2: Has the time limit ended?
Elimination (Elitism + Age based replacement)
1: Yes
End
2-opt the best solution
2: Yes

3.3 Representation

I continued using the *path representation* from the group phase where the cities are listed in the order they are visited. If city i is the j^{th} element of the array, then city i is the j^{th} city to be visited in the tour. This representation seemed most intuitive when we designed the first draft of the evolutionary algorithm (EA) during the group phase and I was more comfortable in performing operations on this representation, hence I decided to continue with it. In Python, I represent each candidate solution as a *list*. From left to right determining the order of the tour. The end of the tour, which is the first element of the array, is implicit and not added to the end of the array.

3.4 Initialization

To initialize the population, the majority of the population was **randomly generated** using the *random.sample()* function. The remaining portion of the population was generated using the **nearest neighbor** heuristic, the size of which was determined by *minimum (length of tour (l), 10% of the population size λ)*. If the former is selected, a candidate using the heuristic is generated using each of the cities as starting point, if the latter is selected then the starting cities are randomly chosen. The nearest neighbor heuristic is describe as such: a random starting point is chosen, from where the nearest city is selected, and continued until the end of the tour and back to the starting point.

The **population size** λ was determined based on the number of cities in the tour and is inversely proportional to the number of cities to improve the computation for resource intensive tours (such as in the 1000 cities scenario) and add more candidates where computation is less of a concern. The formula for the population size is shown in equation 1, whose coefficients were determined by a polynomial regression model to fit the data points of what I determined to be computationally effective, l is the length of the tour and the function $floor$ truncates the decimal values of the result.

$$\lambda = floor(1790.408 - 2.58678l + 0.00141l^2);\ l = [29, 100, 250, 500, 750, 1000] \tag{1}$$

3.5 Selection operators

To select the parents from the candidates, I decided to continue with **k-tournament selection** as we had implemented during the group phase. I added a parameter control mechanism for the k parameter. I.e., k increases linearly as the number of iterations increase as shown in equation 2, where i is the iteration number. This is to promote exploration during the initial stages of the algorithm with a small k and lead to further exploitation of potential candidates in the later stages of the algorithm with a higher k value.

$$k = minimum(floor(3 + \frac{1}{2} * i),\ floor(2.5\%\lambda) \tag{2}$$

I also considered fitness proportion selection, however, decided not to go with it because of its downsides such as premature convergence when the high ranking candidates takeover or lack of selection pressure if the fitness levels are approximately uniformly distributed. The former may especially be the case as I used a heuristic to generate a portion of the starting population.

3.6 Mutation operators

The mutation operator I implemented is the **inversion mutation** operator. I also implement an parameter control mechanism that varies the mutation probability (where each offspring undergoes a mutation with probability $P_{Mutation}$) as a function of the iteration number (i) as shown in equation 3. This is to introduce more randomness in the later exploitative stages of the algorithm by increasing this $P_{Mutation}$, but keep it low in the earlier explorative stages of the algorithm where there are many solutions that are created randomly and sufficient diversity may exist.

Another mutation operator that I experimented with was the swap mutation operator whose version we had also implemented in the group phase. However, from the peer reviews it was reported that the swap mutation is fairly "invasive", hence I decided to change to the inversion mutator.

$$P_{Mutation} = 0.025(1 + log_{10}i) \tag{3}$$

3.7 Recombination operators

I implemented the **order crossover** recombination operator (OX) as the primary crossover operator (80% probability of being selected as the crossover operator for a set of parents) in the final version of my code. This is described as taking a random segment from a parent, and copying that portion into the child whilst adhering to the equivalent positions of the parent copied from. The beginning and ending position of the selected segment are called the first and second crossover points respectively. From the second crossover point in the second parent, the remaining unused numbers are added to the child in the "order they appear in the second parent, wrapping around at the end of the list." The second child is created in a similar manner with the roles of the parents reversed. [Dav91; ES03]

I also tested the Partially Mapped Crossover (PMX) that was implemented in the group phase. *PMX* can be described in the following steps (according to the definition of Whitley [BFM00][ES03]):

- Choose two crossover points at random, copy the segment from the first parent P1 into the first child.
- From the start of the crossover interval, look for elements in second parent P2 that aren't present in P1.
- For each of these uncopied elements, say x, see what element is occupied in the child at the same index, let's call this element y.
- Place x into the index of y of P2 into the child.
- If the position of y is already occupied in the child by an element z, put x in the index of z of P2 into the child.
- The remaining positions in the child can be filled from P2. Follow the same procedure with reversed parental roles to form the second child.

However, PMX does not necessarily hold the property of *respect*, that "that any information carried in both parents should also be present in the offspring", which is suggested as a desirable property [Rad91; ES03]. Additionally, in a study by Abdoun and Abouchabaka, it was shown that the ordered crossover yielded superior results compared to PMX. This was also validated in impromptu tests that I ran with both PMX and OX where the latter yielded generally better results (not included in this report). Hence, I decided to continue with PMX as the secondary crossover operator (20% probability of being selected as the crossover operator for a set of parents) in the final version of the code. The reason for this is that PMX may add more diversity in the search space (from the lack of *respect* property) as my final implementation lacks the diversity promotion schemes.

3.8 Elimination operators

For the elimination operator, I decided to continue with a combination of **elistism** and a simple **age-based replacement** as the primary driver where at the beginning of every iteration, the top 5%λ candidates and kept aside and protected, but are also included in the selection, crossover and mutation. From the resulting children (μ) at the end of an iteration, 95%λ are randomly selected and combined with the protected "elitist" candidates and the parents are discarded.

I decided the elitism rate to be 5% as I wanted good candidates to be protected but to not dominate the pool and still allow for sufficient randomness.

3.9 Local search operators

The local search operator I implemented is a simplified version of the *2-opt* algorithm. To describe it, the implemented version of the algorithm considers a swap of each of the neighboring cities and checks if the resulting tour is an improved version. If it is, the improved tour replaces the original; this is done for all neighboring cities in the tour except the first and second city. Which is then repeated for the resulting tour until the resulting improvement is less than 1% of the prior best tour. The local search is applied to the current best candidate and two random candidates from the pool of "elitist" candidates prior to combination with the children. This is done to push the good candidates to their corresponding optima.

I understand the drawbacks of this method, that can potentially result in a lack of diversity or premature convergence because of exploitation of solely the "best" candidates. For this reason, I chose to only randomly select a very small portion of the "best" candidates to undergo local search. With that being said, also applying local search to a subset of the offspring, parents or the worst solutions may also be viable alternatives that can combat the risks of diversity loss.

Initially, I also added the original *2-opt* local search algorithm, which in essence inverts all possible pairs of cities in the tour (excluding the first). However, despite being very effective, this resulted in a high computation time especially for the larger problems and I decided to instead implement the simplified version.

Finally, if the convergence criteria is met before the time limit has exceeded, the resulting best solution undergoes one iteration of the original *2-opt* local search to optimize to the neighboring minimum.

3.10 Diversity promotion mechanisms

I did not implement a diversity promotion scheme. The reason for this is that I believe there may be sufficient diversity added by the high proportion of randomized candidates in the initial population, increasing mutation rate as a result of iteration number, and the PMX operator (since it does not inhibit the *respect* property). Additionally, the computation time may also be impacted by adding such a mechanism.

Regardless, not having a diversity promotion mechanism is certainly a drawback as this increases the likelihood of pre-mature convergence, getting stuck in a local optima and under-exploitation of other candidate solutions.

3.11 Stopping criterion

The stopping criteria I considered is very basic. I.e., if the best solution has not improved in the past 12 iterations and the time limit of 5 minutes has not exceeded, we stop the algorithm, apply a *2-opt* iteration to the current best solution and report it. The 12 iterations was arbitrary and from the numerous times I ran the code, having the stopping iteration as 12 or another higher number generally didn't make a high difference to the final objective.

3.12 Parameter selection

Parameters in the algorithm, such as population size (λ), number of heuristic starting values and number of elitist candidates are determined as a function of the tour length to improve computation time for especially the larger tours.

I determined the initial values of the mutation rate, and k in tournament selection to be 2.5% and 3 respectively. This is to reduce the chance of ruining potential candidates in the initial phases of the algorithm as sufficient diversity may exist then, but also have a fast and explorative computation that a small k can enable. In the later phases, both these values increase to combat diversity loss and enable exploitation of good candidates respectively. The parameter k also reaches a maximum as a function of the population size, this is to ensure the value doesn't skyrocket in which case few candidates will start dominating the pool.

Other parameters such as elitism rate and the probability for which crossover operator to select for a set of parents was arbitrarily chosen myself and the former is expressed as a percentage of the population size.

3.13 Other considerations

As described in section 3.8, the top 5% of the candidates at the beginning of every algorithm iteration are set aside to be protected to combine **elitism** with the age-based elimination.

4 Numerical experiments

4.1 Metadata

The parameter that depend on the number of cities are the initial population size (λ), which in turn also determines the number of elitist candidates (5%λ).

The computer system that I ran the evolutionary algorithm is a 2020 MacBook Air (M1 processor; 3.2 GHz, 8 core CPU), 16 GB of main memory running Python 3.8.

Note: Comparison with the optimal solution in the following sections is not provided as the leaderboard was not implemented for the retake project. In the evolution graphs, the blue dashed line refers to the best fitness and the solid red line refers to the mean fitness.

4.2 tour29.csv

The population size that was obtained for this tour is $\lambda = 1716$.

The best tour length I found was 158679.11 with the sequence corresponding to: [4, 8, 5, 9, 15, 24, 2, 16, 3, 19, 14, 18, 12, 23, 10, 25, 28, 21, 22, 26, 17, 20, 11, 6, 1, 13, 27, 7, 0].

From the plot 1, we can see the best fitness starts at a value that is representative of the greedy heuristic solution, this is because of the nearest neighbor start. We do observe a quick convergence of the best solution in 0.5 seconds, this may also be accounted to the fast processing power of the local system used, but may also reveal a downside of the current EA that the local search exploitation is only done on the "best" candidates, hence under-exploiting other potential solutions.

The mean fitness on the other hand starts at a high value because of the randomly generated candidates but quickly converges toward the best fitness. This may either suggest many good candidates exist in the pool or suggest that there is a potential lack of diversity in the candidate solutions and the best solution obtained may be

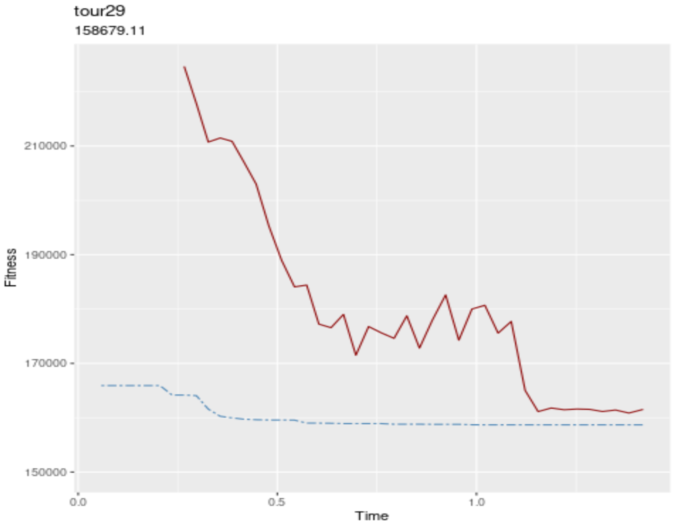


Figure 1: Evolution of tour29

a local optima.

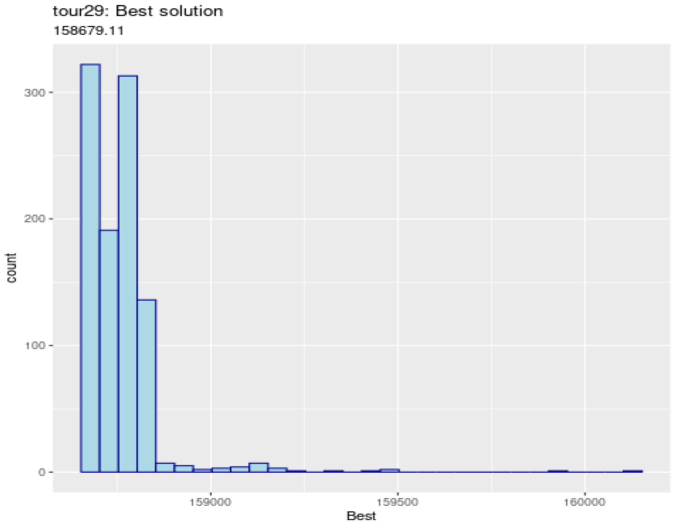


Figure 2: Tour29 Best Fitnesses. Mean: 160334.27. St. Dev.: 2643.96

From plot 2, we can observe that the distribution of the best fitnesses is right skewed and we can conclude that most of the optima are very close together. Additionally, in many of the cases we arrive at the best objective of 158679.11. We also notice there are some outliers but they are not too far off in relatively in terms of the optima we found, this is also represented by the low standard deviation of the best fitnesses.

The observed standard deviation is much higher in the case of the mean fitnesses shown in figure 3. The mean of the mean fitnesses is close to 220000 and far from the obtained minima's, suggesting there are many scenarios where there exist diverse candidates in the pool toward the end of the algorithm.

4.3 tour100.csv

The population size that was obtained for this tour is $\lambda = 1545$.

In the case of tour 100, we can observe in figure 4, where the algorithm converges fairly close to the optima we found in about 5 seconds. The speed can once again be accounted to the fast processor of the local machine, but also the design of the EA. The good starting candidates can be accounted to the heuristic candidates. We also observe a drop in the final iteration, this is because the best candidate is also run through an iteration of *2-opt*. However, the mean fitness is far from the best fitness lines, this may be because of a diverse candidate pool, many of whose fitness is infinity. It is possible that we get stuck at a local optimum, again accounted to the under-exploitation of potential solutions of the local search method and the lack of diversity promotion schemes.

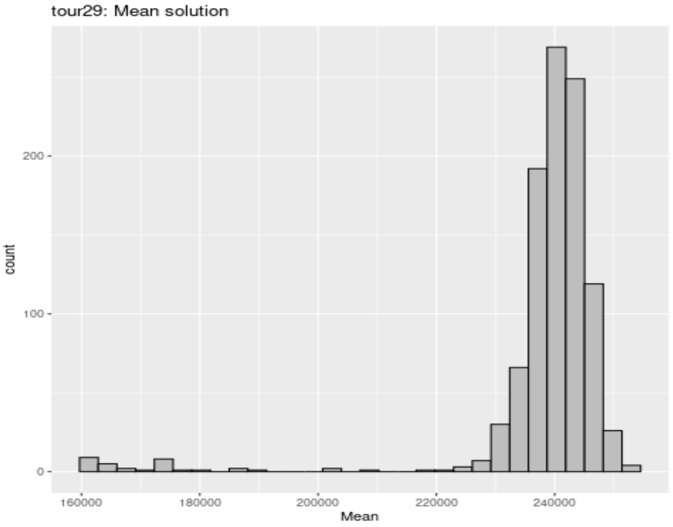


Figure 3: Tour29 Mean Fitnesses. Mean: 213964.05. St. Dev.: 100563.47

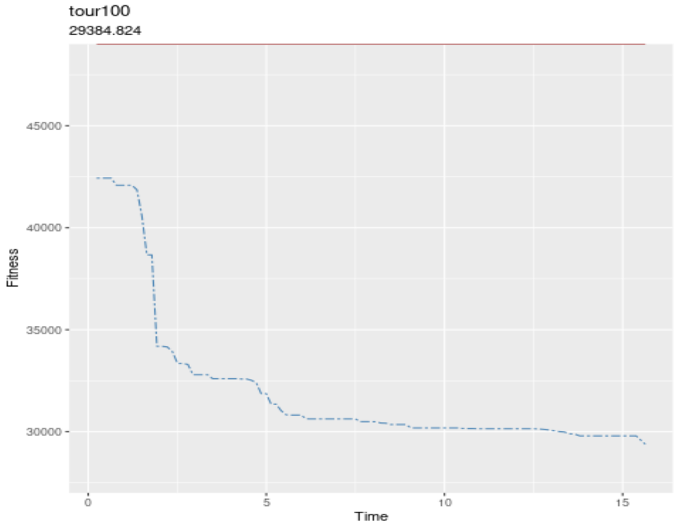


Figure 4: Evolution of tour100. Best fitness: 29384.824

4.4 tour500.csv

The population size that was obtained for this tour is $\lambda = 849$.

From the plot of figure 5, we again observe the good starting candidates at the beginning of the algorithm and the genetic algorithm converges fairly quickly to around the fitness value of 100000. However, it fails to break that barrier which may be again be accounted to the under exploitation of potential candidates, because the simplified local search is limited to the elitist pool.

However, when one iteration of the *2-opt* is applied to the best candidate, we obtain a value of 92412.6. We notice this takes a fair amount of time on just one candidate, as this is a resource intensive computation. Although this is a much better fitness, it is unlikely to be the global optimum as additional algorithm iterations or local searches of this candidate may also lead to a better fitness.

4.5 tour1000.csv

The population size that was obtained for this tour is $\lambda = 613$.

Similar to the previous case, from the figure 6, we can observe the genetic algorithm starts near the heuristic solutions, is able to converge fairly quickly, but gets stuck at a local optima with the current mechanisms in place. Once again, the final *2-opt* iteration is able to drastically improve the fitness, albeit taking a significant portion of the computation time. This time however, the mean fitness is closer to the best fitness curve, this could also in some cases suggest most candidates are similar and some diversity promotion scheme, increased population size or exploitation of ”worst” candidates or offspring may help offset this.

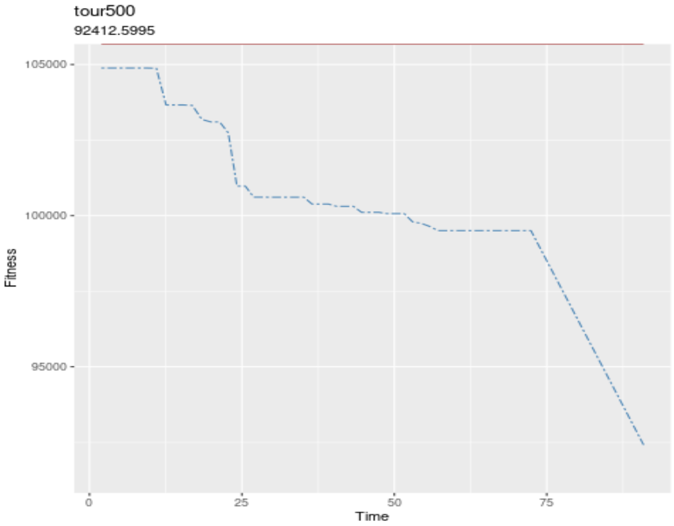


Figure 5: Evolution of tour500. Best fitness: 92412.600

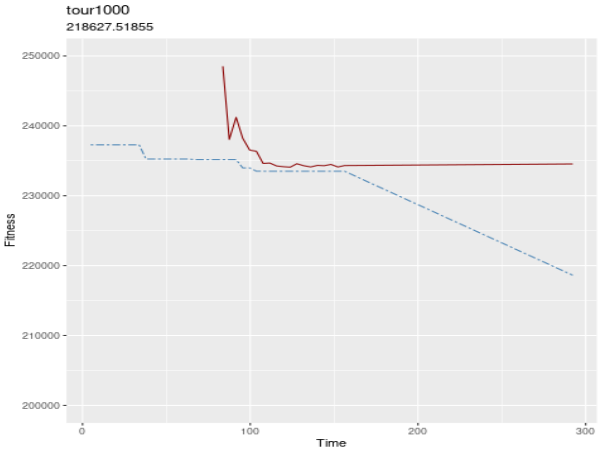


Figure 6: Evolution of tour1000. Best fitness: 218627.518

5 Critical reflection

The three main strengths of my EA are:

1. **Application of heuristics**: the local search and nearest neighbor heuristic either already provide good candidate solutions or are able to bring potential candidates toward their neighboring minima.
2. **Relatively quick convergence**: In the graphs of the tours in section 4, we observe the portion of the algorithm reliant on evolutionary mechanisms is able to converge fairly quickly from the starting values.
3. **Combination of crossover operators**: The combination of a primary (OX) and secondary (PMX) crossover operator allows the primary operator to exploit the good parents but allow the secondary operator to promote diversity in some cases.

The three main weaknesses of my EA are:

1. **Lack of diversity promotion schemes**: Although there are methods that add randomness in the later stages of the algorithm, there does exist a lack of diversity promotion schemes. The smaller population size for the larger tours may also be a culprit, albeit done to improve the computation at the cost of diversity.
2. **Potential premature convergence**: This may especially be the case for the larger tours, where the EA converges quickly, yet the final local search can improve the solution by a significant amount. This is premature converge may be related to the previous point of lack of diversity promotion schemes. Additionally, how the local search operator is used only on the ”best” candidates may lead to an under-exploitation of other potential candidates.
3. **Absence of automated hyperparameter tuning**: The tuning for the some parameters was performed arbitrarily, however, if sufficient values were benchmarked for different problems, a more robust EA could be

devised with an automated hyperparameter tuning mechanism with relatively little effort.

This project was challenging, yet quite the learning experience. I am pleasantly surprised on the applications and effectiveness of meta heuristics such as evolutionary algorithms. When I first heard the idea, I was quite skeptical about its potential, however, I've seen a number applications resulting from this course where genetic algorithms can be useful. Moreover, it is a fun task to construct one as well, especially for such a design focused project where the sky is the limit. Finally, I was encouraged to hone my coding skills as a result of this intensive project.

In my case, genetic algorithm worked pretty well for smaller number of cities, however when the number of cities grows larger, the local search methods are able to provide a large improvement since the genetic algorithm gets stuck at a local optimum. With that being said, there is no rule that one should rely on one of the other, the beauty is in the combining the best of both worlds and allowing the evolutionary algorithms to work quite hand-in-hand with heurisitcs for such NP-hard problems.

6 Other comments

I still believe numerous improvements can be made to my current implementation, if time was not a constraint. I have listed some of these ideas below. I understand that my algorithm lacks sufficient diversity generation mechanisms (such as crowding), but also mechanisms to improve other potential solutions that are not in the "best" candidate pool. The combination of the aforementioned can add diverse, yet good candidates to improve the results of the EA.

Several other mechanisms can be also be added such as randomized nearest neighbors whose essence is the nearest neighbor algorithm, but the neighbor is randomly determined from the best *n* neighbors (which is a parameter we can choose), which can allow for an increased number of good yet diverse starting candidates. Other forms of self-adaptivity could also have been added, for instance, based on recognition of being stuck in a local optimum, to increase the explorative portion of the EA.

Adapting parameter control mechanism of the local search to switch between exploiting a portion of the offspring, the "best" and the "worst" solutions could also be implemented to exploit potential candidates based on the current stage of the algorithm.

More computationally efficient code is also a possibility, especially for local search operators, which are quite intensive yet effective for large tours. However, my lack of a computer science and coding background was a slight limitation to such a possibility.

References

[Dav91] L Davis, ed. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[Rad91] Nicholas J. Radcliffe. "Forma Analysis and Random Respectful Recombination". In: *ICGA*. 1991.

[BFM00] Thomas Bäck, David Fogel, and Zbigniew Michalewicz. *Evolutionary Computation 1—Basic Algorithms and Operators.* Jan. 2000, pp. 274–284. DOI: 10.1887/0750306653.

[ES03] 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.

[AA12] Otman Abdoun and Jaafar Abouchabaka. "A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem". In: *CoRR* abs/1203.3097 (2012). arXiv: 1203.3097. URL: http://arxiv.org/abs/1203.3097.