In Solving the Travelling Salesman Problem Using a Genetic Algorithm, How Does Order Crossover Compare to Partially Mapped Crossover in Terms of Improving the Efficiency of Convergence and Optimality of the Solution?

Computer Science Extended Essay

Candidate Code: HZS728

Session: November 2022

3999 words

Table of Contents

Introduction

The ‘Travelling Salesman Problem’ (TSP) refers to the challenge of finding the shortest path between multiple destinations. While this problem is solvable through mathematical methods when the number of destinations is low, the problem quickly becomes intractable as the number of destinations increase (Kitjacharoenchai et al., 2019). To overcome this limitation, modern heuristics and/or algorithmic approaches have made discerning practical solutions for the TSP practical.

A range of algorithms to solve the Travelling Salesman Problem have found use in fields such as determining optimal routing for lines in computer circuitry (Maimon & Braha, 1998), climatology (Stanislawska et al., 2012) and are used in mapping software to find optimum travel routes (Vidal et al., 2012).

The development of genetic algorithms in the 1970s (D Fraser, 1970) has provided the opportunity to efficiently find sufficiently optimal solutions to non-deterministic problems in a reasonable amount of time. Genetic algorithms imitate evolution in the sense of a Darwinist system where individuals compete for survival. This imitation manifests as the gradual improvement of populations as individuals are chosen by merit to carry on their genes to the next generation through the ‘crossover’ of genetics with another individual. By crossing over the ‘genes’ that make up the ‘chromosome’ of individuals of a population producing offspring with characteristics of the parents to explore directions for improvement.

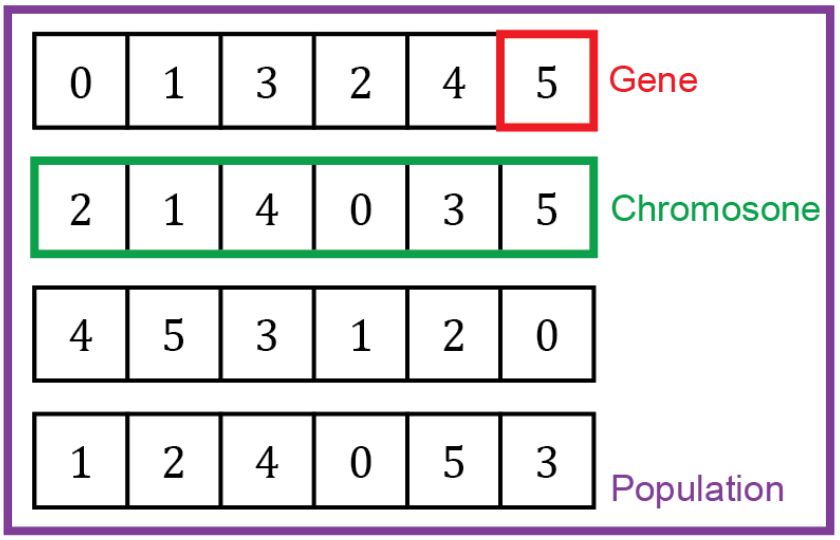

Figure 1. Visualisation of genes, chromosomes, and population

The aim of this investigation is to determine how Order Crossover compares to Partially Mapped Crossover in terms of improving the efficiency of convergence and optimality of the solution. This efficiency and optimality were evaluated through experimentation on the impact of exploration and exploitation through applying the crossover operator in the travelling salesman.

Background Information

The Travelling Salesman Problem is an example of a *non-deterministic* problem. Non-deterministic problems are problems where achieving a definitive result is either impractical or impossible (Bierwirth & Mattfeld, 1999).

Problems can be categorised under P, NP, NP-complete and NP-hard where P stands for 'polynomial time'. P can be categorised as any algorithm where the time complexity increases at a rate polynomial to the number of data points. NP stands for 'non-deterministic polynomial time' meaning that it has a time complexity greater than polynomial time and is the category that non-deterministic problems are in. 'NP-complete' is a subset of NP where a solution can be verified in polynomial time. 'NP-hard' is a superset of NP-complete where there is no known algorithm which can solve it in

polynomial time. (Cormen et al., 2009) These types of problems require the use of heuristics / meta-heuristics due to their impractical computational demands under deterministic approaches.

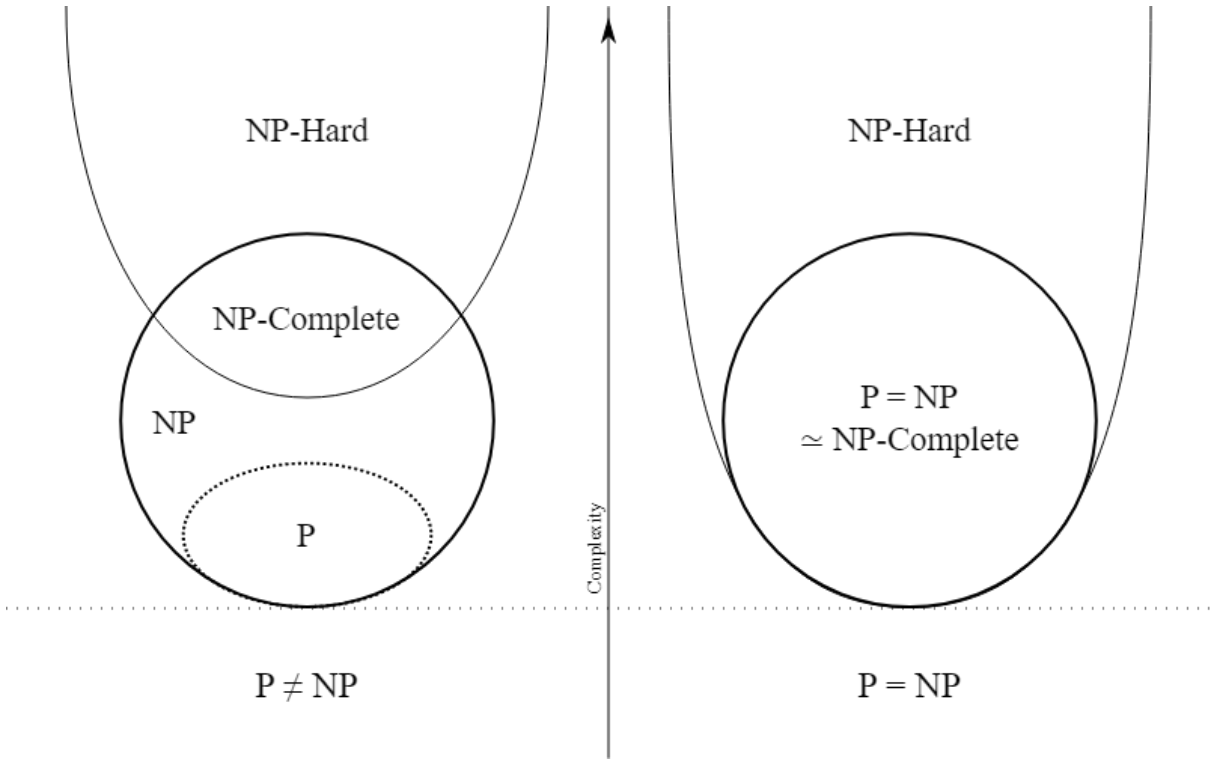

Figure 2. Euler diagram for P, NP, NP complete and NP-hard problems (*P Np Np-Complete Np-Hard.Svg*, n.d.)

Heuristics are an approach or technique used to reach an approximate solution to a problem for which determining the absolutely optimal solution would be impractical or impossible. Heuristics do not guarantee a sufficiently optimal solution but nevertheless, provide an approximate solution in polynomial time and use a reasonable amount of resources. Heuristics are iterative or recursive approaches to a problem that use empirical means (trial and observation) to approach a solution (Cormen et al., 2009). Basic examples of heuristics include 'trial and error', 'rule of thumb' and 'educated guesses'. The term 'meta-heuristic' is used to describe a situation where multiple sub-heuristics are applied by a greater heuristic (Cormen et al., 2009).

Genetic algorithms

One important type of meta-heuristic is a Genetic Algorithm (GA), a class of evolutionary algorithms (Storn & Price, 1997) which draws inspiration from evolution and the concept of survival of the fittest. GAs use a meta-heuristic approach to solve optimisation and search problems by evolving candidate solutions through a series of 'genetic operators' in order to arrive at a potentially optimal solution. These operators are inspired by natural processes of selection (based on fitness), crossover and mutation.

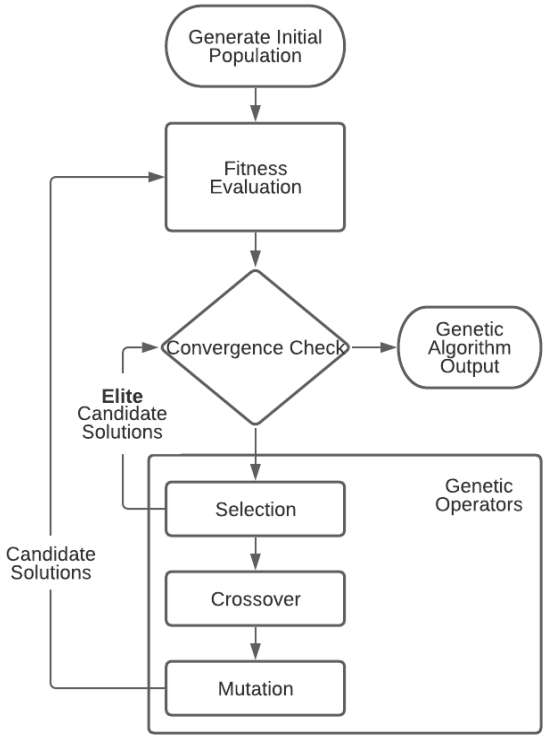

Figure 3. Flow diagram showing the major processes involved in a GA.

The first process in a GA is to generate an 'initial population'. The initial population is a subset of the 'search/problem/solution space' which comprises all valid 'candidate solutions' (Naimi et al., 2003). The initial population can be specifically tailored but it is normally randomly generated. GA initial populations should consist of all unique or almost all unique candidate solutions to ensure 'diversity' in the initial population (Naimi et al., 2003). Initial population diversity is essential in order to avoid

the population too quickly becoming homogeneous and thereby stunting exploration of the solution space.

Each member of a population is assigned a 'fitness value' which measures the optimality of the solution to the problem. This fitness value is calculated as a function of the performance/'objective value' achieved by the candidate solution in empirical testing against the problem. Traditionally the fitness value should be greater for more 'fit'/performant candidates and the problem should be treated as a 'maximisation problem' with the goal of finding the fittest possible candidate.

A GA judges its completeness based on the 'convergence' of the population. The level of convergence may be seen as the homogeneity of the population. In practice, convergence in a GA can be determined when a large proportion of the population shares the same genes over many generations. Convergence will almost always occur at an extremum in the fitness landscape, meaning that any change to the genes of the fittest candidates will only reduce their fitness.

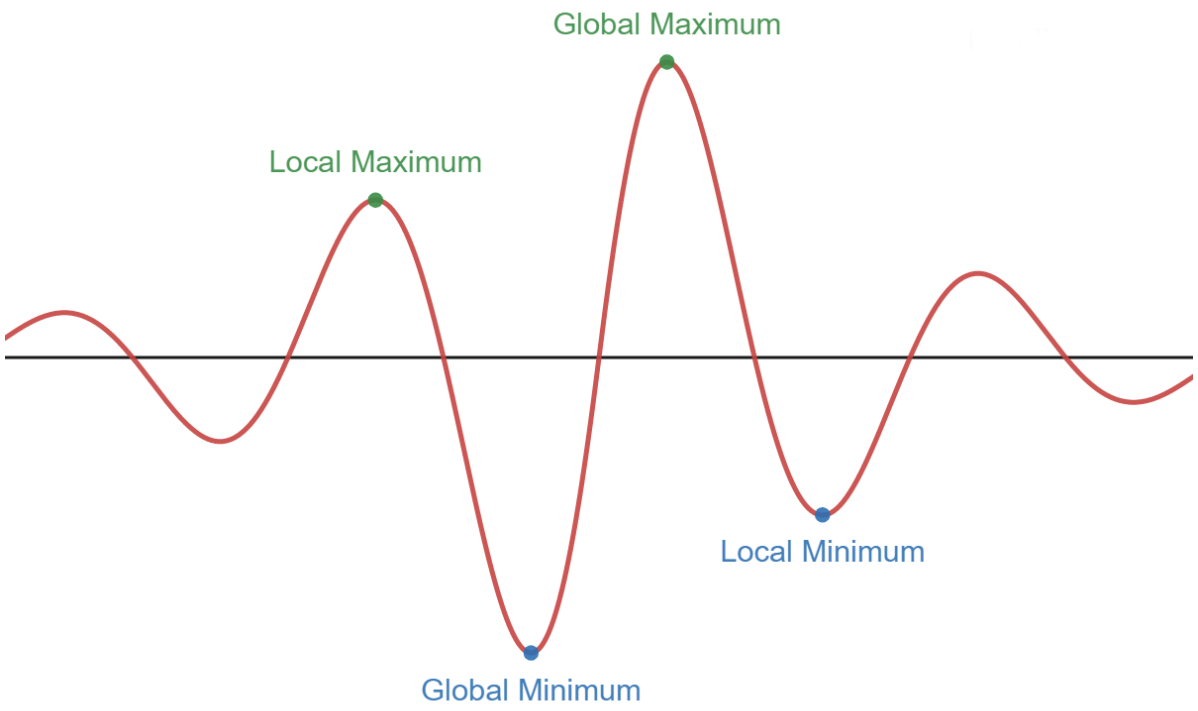

Figure 4. Visualization of local and global extrema of $f(x) = sin(x) \times e^{|\frac{x}{5}|}$

Genetic operators (selection, crossover, and mutation) play a crucial role in ensuring that population does not converge prematurely. Premature convergence is a decline in diversity within the population before the GA has had time to assess enough of the search space to sufficiently increase the likelihood of arriving at a global fitness extremum. To combat this phenomenon algorithm designers look to balance ‘exploitation’ and ‘exploration’.

GA exploitation relates to how readily the algorithm will exploit/converge upon the fittest candidates of the population. Exploitation accelerates convergence and thereby reduces the amount of resources required to find a solution. Additionally, exploitation increases the stability of the population as greater homogeneity decreases the genetic fluctuations across generations increasing confidence that the GA has settled upon an extremum. This speed and confidence comes at the expense of diversity/exploration of the search space and consequently hinders the likelihood of converging on fitter extrema.

GA exploration relates to how extensively the GA is geared towards exploring the search space. An increased focus on exploration results in a greater probability that the genes relating to global/optimal extremum will enter the population. This can greatly increase the effectiveness of a GA however, in excess, exploration can inhibit the GA’s ability to converge. Unrestrained diversity and too little ability to exploit the fittest candidates can cause the fitness of the population to become unstable. This inhibits the GAs ability to achieve its termination condition and reduces confidence in found solutions.

Crossover is the process of combining the characteristics of two or sometimes more (Herath & Wilkins, 2018) candidate solutions from the mating pool. Through crossover the ‘progeny’ will hopefully preserve the key characteristics of the ‘parents’ and through this the GA will explore combinations similar to the parents. Because the parents are selected with preference to fitter

candidates, the progeny will express possibly advantageous combinations of the parents' characteristics.

Different crossover operators tend to preserve different parental characteristics in each offspring and therefore, in GA design, should be strategically considered to ensure an effective balance of exploration and exploitation.

After crossover, each offspring is potentially altered by a 'mutation' operator. Each offspring has a chance to be altered randomly according to a defined 'mutation rate' established in the initial GA configuration. Mutation directly affects offspring characteristics which may result in new potentially advantageous characteristics. Furthermore, a higher mutation rate increases the exploration and thereby population diversity of the GA. Mutation is one of the most direct ways to avoid premature convergence as a sufficient rate ensures diversity. However, if the mutation rate is too high, then the GA may never converge on a solution

'Elitism' is a method for increasing the prevalence of fitter characteristics within the population. It does this by ensuring that the fittest candidate/s are passed directly into the next generation without being affected by mutation. It also ensures that even if the fittest candidate is not selected by the selection operator, that those advantageous characteristics are not lost. Elitism increases the exploitation of a GA therefore increasing the rate of convergence.

Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is the problem of finding the shortest tour (route) between a set of destinations. The scenario proposed by the TSP is that of finding the most efficient route for a travelling salesman to visit a number of cities (n) exactly once and then return to the home city. (Robinson, 1949)

The cities and their connectivity may be modelled as a complete graph (see figure 5) where each city is a vertex, meaning that every vertex is connected directly to every other vertex.

The tour may be modelled as a 'Hamiltonian cycle' whereby every vertex is visited only once, before returning to the starting vertex. This tour can be modelled as a permutation of the set of cities in order of when the city was visited in the cycle. The shortest possible tour is the solution to the TSP.

The worst-case time complexity of the TSP increases factorially as the number of cities increases. This is greater than polynomial time complexity and is therefore classified NP-hard. One can easily verify in polynomial time that a found tour belongs to the search space, therefore the TSP is also NP-complete (Mingozzi et al., 1997).

Due to the TSP being NP-hard, deterministic means such as a 'brute force' (trying every possible solution and returning the best found) approach would be infeasible as, in the worst case, this approach would have to check n! solutions. For example, a graph of 20 cities would involve checking $20! = 2.43 \times 10^{18}$ potential solutions which would require an impractical amount of time and resources to compute. Even though a deterministic dynamic programming algorithm such as the 'Held-Karp algorithm' is able to solve TSPs for a low number of cities, the algorithm quickly becomes infeasible once $n > 23$. Therefore, an approximation algorithm or heuristic is required to make this problem tractable for large values of n.

There are many ways to approximate a solution for the TSP, including the 'MST-DFS' heuristic or 'Christofides Algorithm'(Mingozzi et al., 1997). These are guaranteed to give a result close to the best possible solution however, a GA typically finds a sufficiently accurate approximation in less time.

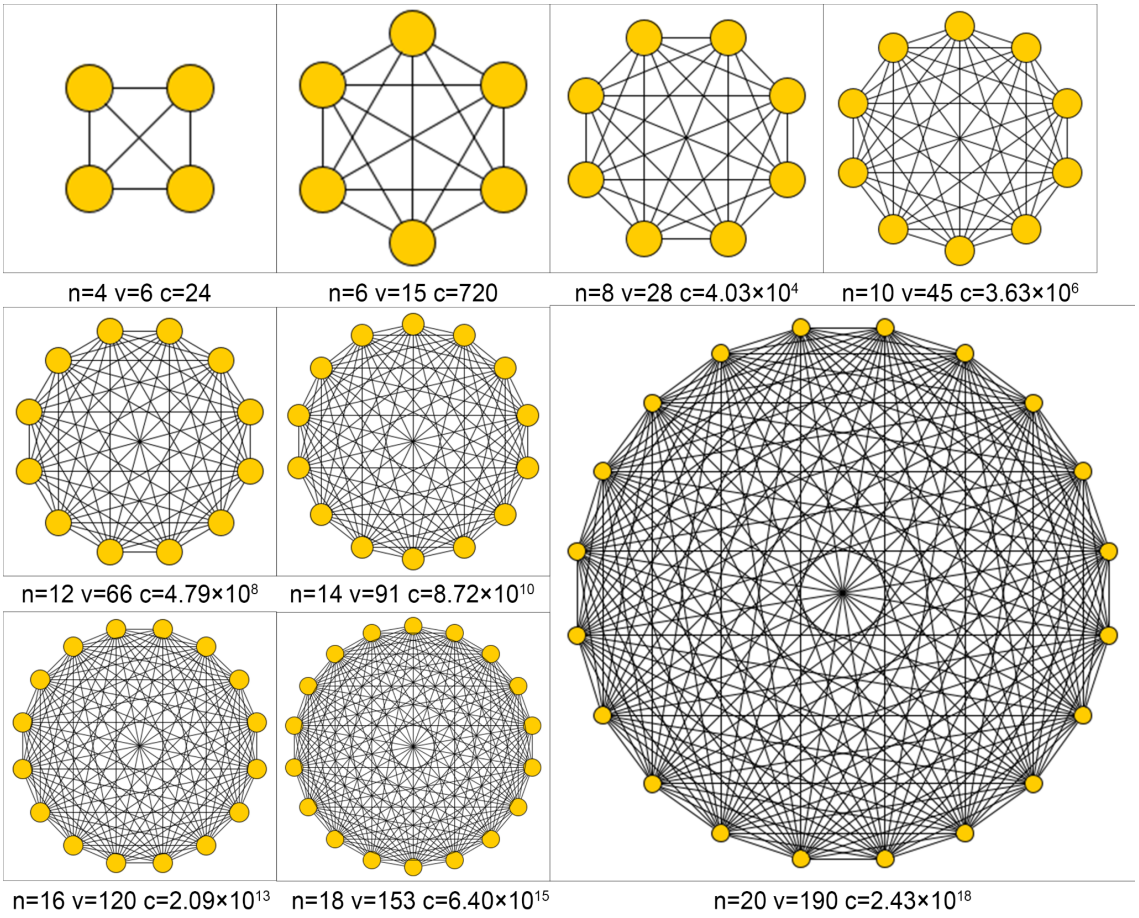

Figure 5. Complete graphs demonstrating how complexity of Hamiltonian Circuits would increase as the number of cites/points/nodes 'n' increases from 4 to 20. The number of vertices/edges is 'v'($\frac{n(n-1)}{2}$), and the number of possible tours 'c' (n!)

Methodology

The aim of this experiment was to gather a large amount of quantitative data comparing the crossover operators, Order Crossover (OX) to Partially Mapped Crossover (PMX) in otherwise identical GAs.

For this experiment a modified version of the 'scikit-opt' GitHub repository by user 'guofei9987', a software for optimisation problems, was used. It was chosen as it had in-built support for GAs and specifically the TSP (郭飞, 2017/2022).

Modelling Tours

An array of integers was used to model each tour of the TSP. Each city in the network was assigned an arbitrary unique integer identifier. The order of cities corresponds to the order of identifiers in the array. It was assumed that after the last city was visited then the salesman returned to the first city. It was therefore unnecessary to have the returning city at both the start and end of the array so it was only placed at the start. For example, a tour modelled as [1, 0, 2] would start at 1, then go to 0 then 2 before returning to city 1. Due to the order of identifiers being central to this model and the requirement that each city be visited only once, the array may be treated as a permutation without repetition.

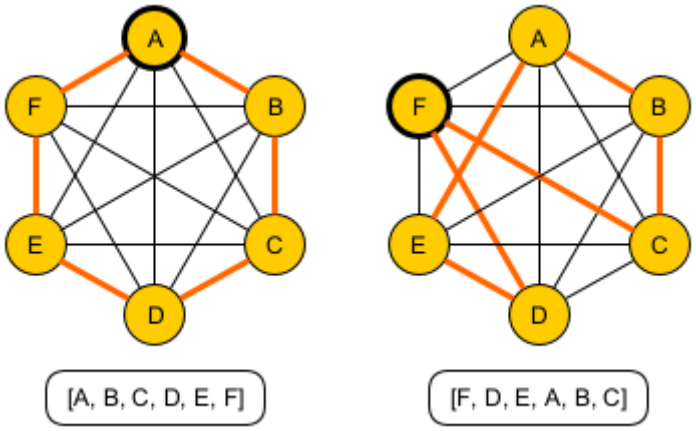

Figure 6. Two examples of Hamiltonian cycles on a complete graph, the two tours are also shown in array form - permutations without repetition.

Initial Parameters

The initial parameters for the GA were largely determined relative to the number of cities to be visited. Due to the factorial increase of potential solutions in relation to the number of cities, a high number of cities would be infeasible to repeat enough times for statistical reliability on the hardware available. However, too few cities and the effect of the crossover would be harder to measure with mutation taking a disproportionately large role. Given hardware limitations, 20 cities was chosen to maximise the measurable influence of crossover. While for this number of cities, deterministic algorithms such as the Held-Karp algorithm may be more appropriate than a GA (Mingozzi et al., 1997), the results may be extrapolated to higher numbers of cities where such deterministic algorithms are infeasible.

Greater population size improves the stability and diversity of a population. Additionally, the convergence of a large population does not fluctuate as much as a small population since mutations have less impact on the general fitness of the population. This makes it easier to measure the convergence of a generation and gives confidence in the provided solution. A population size of 100

was chosen for this experiment as it provided a good balance of convergence and diversity and was concordant with similar experiments (Larrañaga et al., 1999).

Test Cities

To evaluate the accuracy of a GA, the found solution would need to be tested against a deterministic solution requiring a group of cities with a known shortest route. For this, a circle with 20 evenly spaced cities along the circumference seemed fitting as the shortest route would always be to follow the next city along the circumference. It was concluded that the results derived from using this regular shape would not be influenced by its regularity. This is due to the separation between the string representation of the problem and the problem itself.

The initial population for the experiment was modelled as an array of 100 randomly generated candidate solutions, each a shuffled tour / random permutation of the cities.

Selection

Each candidate in each generation was assigned a fitness value as the reciprocal of the sum of distances in the tour.

In this experiment, a very exploitative method of elitism was used. This involved pooling both the parents and offspring of a generation into one array such that the array was equal parts parents and offspring. This array was ranked (sorted in descending order). The first half of this array became the new generation, so the population size was replenished.

This method of elitism was chosen as it simplified the implementation of the GA.

OX and PMX were both implemented according to their standard algorithms (Larrañaga et al., 1999) to ensure that the results collected are fair.

Crossover

Order Crossover

The algorithm for OX was as follows:

1. Two integers, '*a*' and '*b*', between 0 and *n=20* are randomly generated.
2. A slice/segment of parent 1's chromosome between indices *a* and *b* is directly copied to the child maintaining the slices indices.
3. The algorithm iterates through parent 2's chromosome from index *b* and checks to see if each gene is present in the child. If a gene is not present, the algorithm inserts this gene into the next empty index of the child after the slice.
    a. If the loop reaches the end of parent 2's chromosome the loop returns to index 0 and continues until the start of the slice.
4. This process was repeated with swapped parents to produce a second child.

Partially Mapped Crossover

The algorithm for PMX was as follows:

1. Two integers, '*a*' and '*b*', between 0 and *n=20* are randomly generated.
2. A slice/segment of parent 1's chromosome between indices *a* and *b* is directly copied to the child maintaining the slices indices.
3. Looking in parent 2 at the same indices as the segment, select each gene that was not copied to the child.
    a. For each of these values:
        i. Note the value, *v*, in parent 1 with the same index as the selected gene
        ii. Locate the index of *v* in parent 2
        iii. If this index was part of the segment copied to child 1 go to step i. using *v* and this index

iv. Else if the position isn't part of the segment copied to child 1, insert the selected gene into the child in this position.

4. Copy any remaining positions from parent 2 to the child.
5. This process was repeated with swapped parents to produce a second child.

While OX and PMX share a similar process for passing the characteristics of the first parent, the process for passing the characteristics of the second parent differ considerably. PMX gives preference to the index of the parents as shown in 'step 3a' while still preserving some, but not necessarily all, order of parent 2 in step 4. In contrast, OX will necessarily preserve the characteristics of their parents' order.

Mutation

A 'swap' mutation operator was used for this algorithm. This operator worked by first iterating through the genes of a candidate solution where on each iteration there is a chance the position of a gene will be swapped with a random other gene within the chromosome. This method of mutation was chosen as it inherently avoids repetitions of genes which is not allowed within the rules of the TSP.

Termination Condition

It was important to manage the termination of the GA to ensure that it only returned a result once the algorithm had had sufficient time to achieve a result. There are many 'termination conditions' used in GAs which generally fall into one of two categories: termination upon reaching convergence within the population and termination upon using a certain number of resources.

Often termination upon convergence refers to the homogeneity of a population (Ochoa, 2006). However, absolute homogeneity is extremely unlikely to occur as mutation will alter population and lead to a "cloud" of less fit candidates even if the vast majority of the population is homogenous. It is therefore pertinent to allow for some diversity in the population when determining convergence.

Additionally, the stability of the characteristics of the population between generations is an indicator of convergence as when there is little diversity, crossovers do little to change candidates. The identicality of fitness values may be used instead of comparing candidates' characteristics to aid in runtime execution speed and reduce complexity. If the majority of the population's fitness values stay consistent for a set number of generations, then one may consider that the population has converged.

The experiment was set up so that if the median fitness value remained consistent for 500 generations, a number consistent with similar experiments (Larrañaga et al., 1999), then the population was deemed to have converged, and the GA was terminated.

Additionally, a resource limited convergence was used. If the GA iterated through 10,000 generations, then the algorithm would be terminated to prevent an infinite loop from occurring.

Each GA run returned the total distance of the distance of the fittest candidate (float), number of generations (integer) produced, and the accuracy of the result as compared to the deterministic solution (Boolean).

Method

1. Run the GA with OX crossover and parameters as discussed in methodology
2. Record the tour, time, distance, generations, and accuracy of the result against a precalculated deterministic solution of length 3.1286 units. (see appendix 1)
3. Repeat steps 1-2 500 times
4. Repeat steps 1-3 with PMX instead of OX

Results

After carrying out the experiment the following mean-averages were calculated for the distance, generations, accuracy (boolean measure of whether the GA returned the deterministic solution or not), and the average percentage error ((result distance - solution distance)/solution distance).

	Averages			
	Distance	Generations	Accuracy	%Error
OX	3.2042	1089.5	88.80%	2.42%
PMX	3.1935	1007.8	91.42%	2.07%

Table 1. Calculated averages for the distance, generation count, accuracy and error for GA results.

The results showed that PMX was more effective than OX in every measurement.

PMX was faster, completing each test 8.44 seconds faster on average than OX. PMX on average resulted in a shorter distance however this is closely tied to the percentage error of the algorithms where PMX was on average 0.35% closer to the solution. PMX achieved the accurate result 2.62% more often than OX. PMX averaged 81.7 generations fewer than OX.

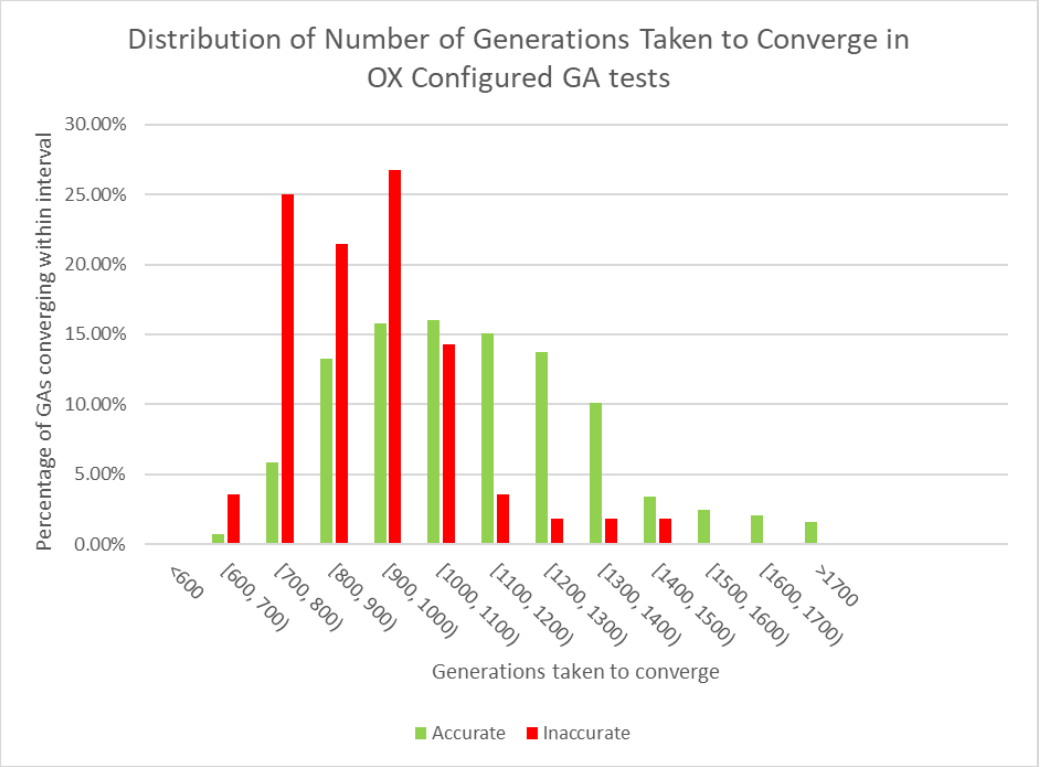

Figure 7. Distribution of number generations taken to converge in OX configured tests

Figure 7 shows the distribution of the number of generations that OX took to complete each test. It shows the inaccurate solutions being much more heavily skewed towards lower numbers of generations.

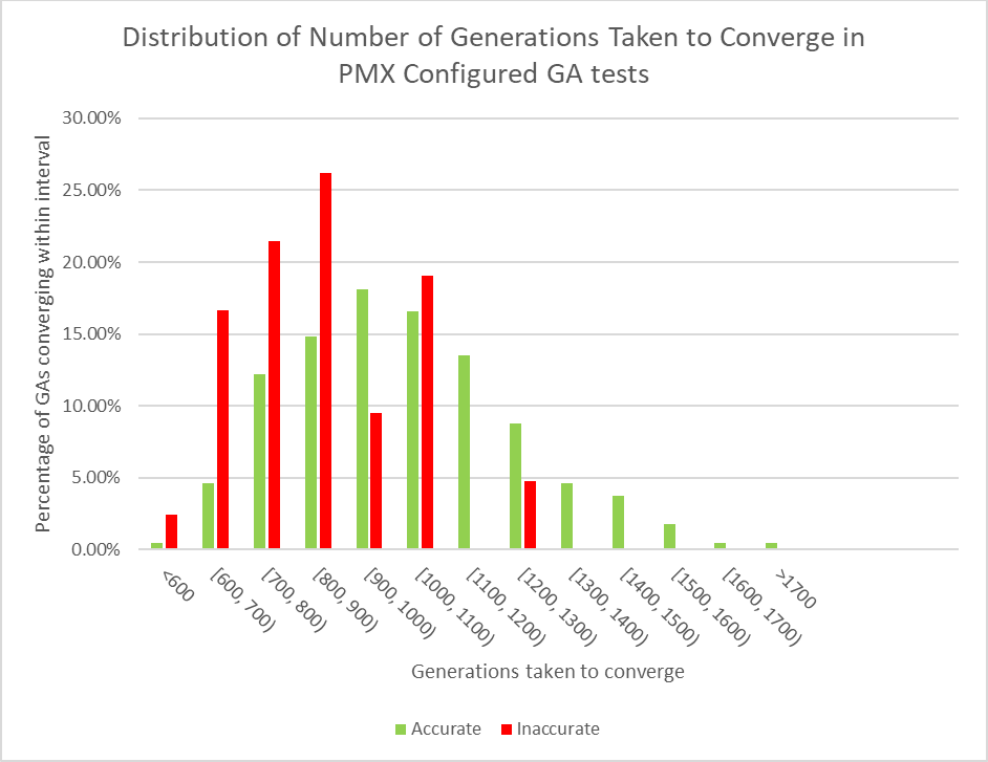

Figure 8. Distribution of number of generations taken to converge in PMX configured GA tests

Figure 8 shows the distribution of the number of generations that PMX took to complete each test. The inaccurate solutions are heavily skewed towards lower numbers of generations, with few failing after 1100 generations. There are also less earlier inaccurate convergences for OX.

Discussion

These results strongly suggest that PMX is far better suited towards GA approaches to the TSP than OX in these experimental conditions. PMX was found to be better suited than OX for resource limited applications with an average of 81.7 fewer generations per test, and for high precision applications with 2.62% higher accuracy and 0.35% less average error.

These results were surprising since OX is widely seen as the standard crossover for approaching the TSP (CernerEng, 2016). Additionally, this combination of both a higher degree of accuracy and fewer generations to reach convergence seems to contradict the general understanding of the effects of exploration and exploitation. This understanding being that an exploitative GA would require less

generations to achieve a result at the cost of accuracy due to premature convergence. Conversely, an exploratory GA is said to converge slowly but have the advantage of being able to explore more of the search space and consequently be more accurate.

Evaluation

The convergence rate (change towards candidate similarity over time) was expected to be proportional to the exploitation of the GA where an increase of exploitation would cause a faster convergence. An abstraction of this relationship is plotted in figure 9.

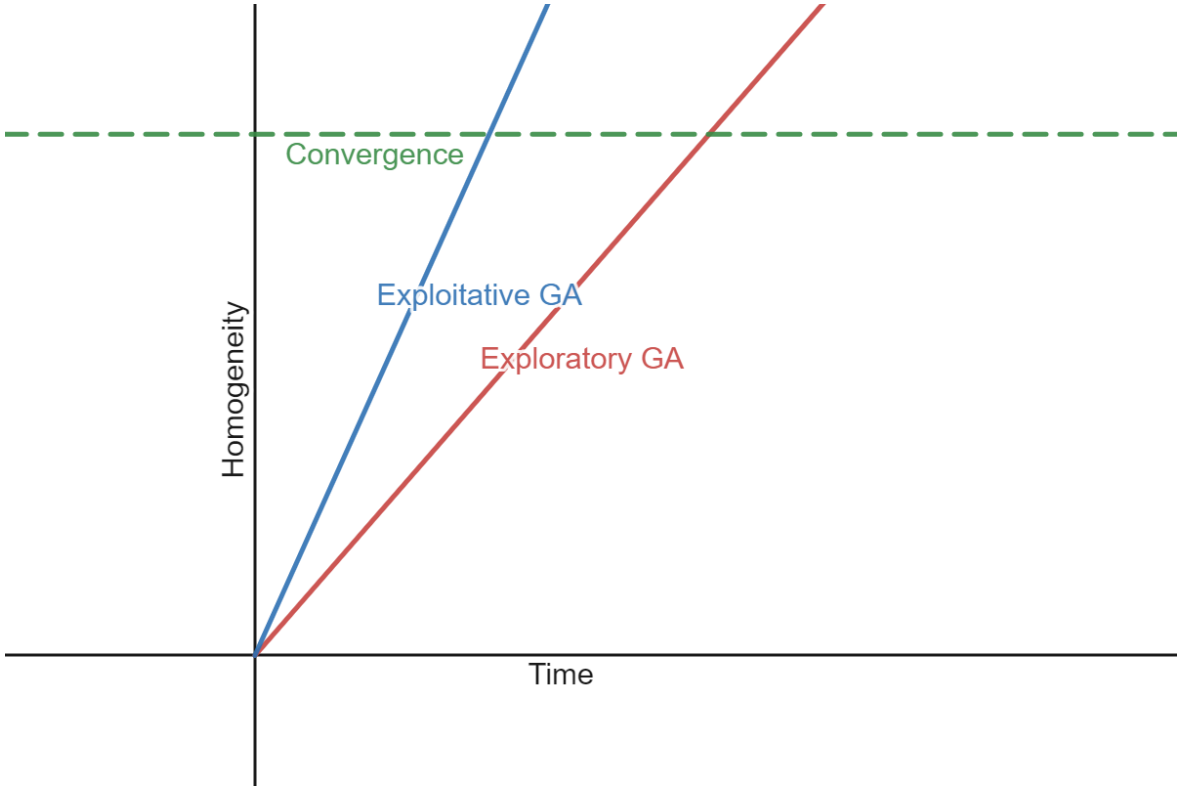

Figure 9. Linear relationships between Homogeneity and Time.

However, it was shown through experimentation that this relationship does not hold up as the OX configured GA (the more exploitative GA) was demonstrated to be slower to converge.

It could be argued that perhaps the presumption that PMX is more exploratory than OX was in fact false. However, this does not seem to be the case as PMX was also more accurate, a trait belonging to exploratory GAs.

Additionally, OX necessarily preserves the order of parent chromosomes. Since the characteristics of a candidate solution for the TSP are defined solely by order, the only relevant characteristic of a parent chromosome is their order. PMX does not necessarily preserve this order therefore carries fewer characteristics to offspring meaning it is more exploratory.

Another rationale was needed to reconcile these results with the understanding that exploitation allows for faster convergence. It was hypothesised that the change in diversity between each generation over time could follow nonlinear paths. The proposed relationships of homogeneity over time for exploratory and exploitative GAs are shown in figure 10.

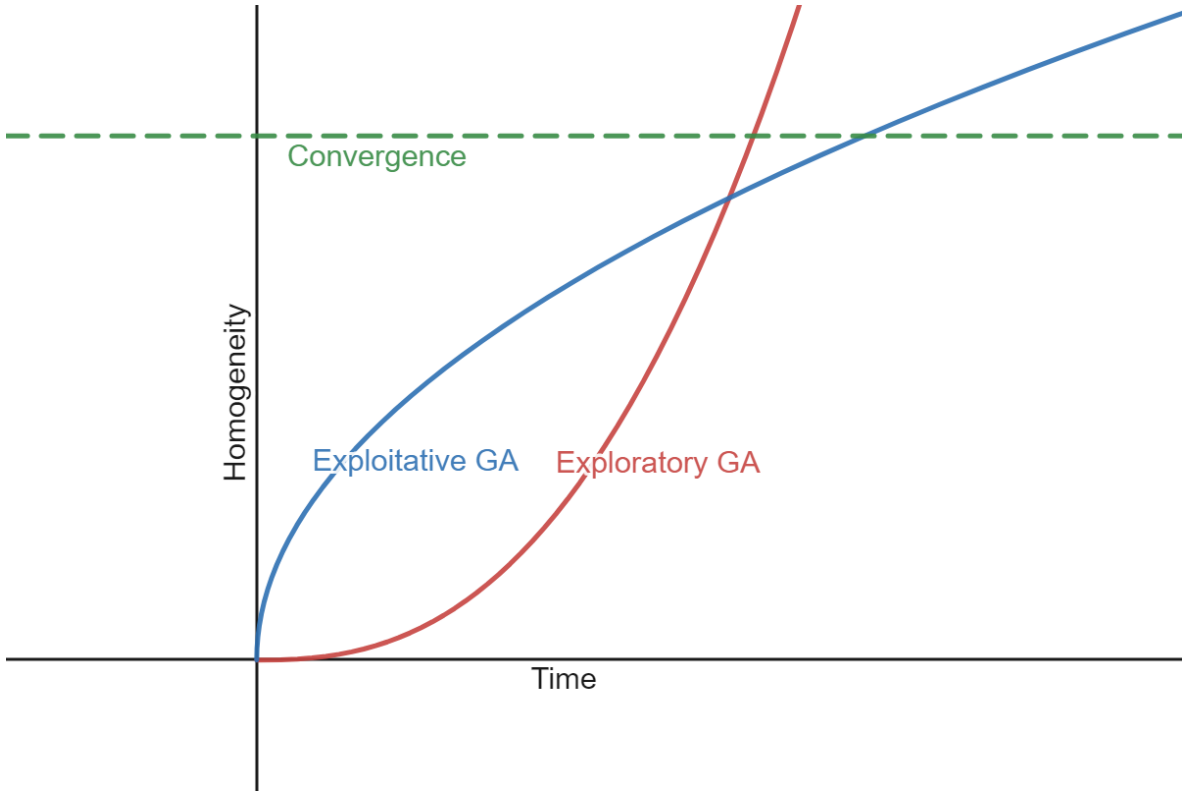

Figure 10. Non-linear relationships between homogeneity and time.

The exploitative GA, which may be achieved through use of OX, has an initially higher convergence rate than the exploratory GA, which may be achieved through use of PMX, however its convergence rate levels off as time goes on while the exploratory GA increases its convergence rate.

The theory for the curve of the exploitative GA is that it will still heavily exploit its candidates. However, this initial exploitation is conducted on an unfit population leading to a similar population of unfit candidates. Without sufficiently exploring the solution space the population does not contain any significantly fitter candidates. This results in a population full of the many similarly unfit candidates which significantly outnumber fitter candidates. This may lead to a decrease in the convergence rate where many similar but not identically unfit candidates compete for genetic supremacy in the population.

Conversely, the exploratory GA would not necessarily have this initial boom of convergence rate. It could then explore more of the search space, increasing the likelihood that a range of significantly fitter candidates could be found as diversity is encouraged rather than suppressed. It is believed that once these significantly fitter candidates have been found, the selection operator can greatly prefer these candidates. This preference would lead to this gradual increase in convergence rate that eventually leads to the exploratory GA overtaking the exploitative GA (see figure 10).

This hypothesis could be tested through further experimentation and plotting the real change in diversity over time in many simulations of the TSP. One could compare the curves observed in tests run on exploitative versus exploratory GAs. This would require a quantification of diversity to be plotted against time, therefore requiring a more concrete and mathematically defined definition of the phenomena.

Conclusion

GAs and the non-deterministic problems they make tractable permeate modern life. It was found that in solving the TSP using a GA in experimental conditions which may be generalised to similar scenarios, OX performs substantially better than PMX in terms of improving both the efficiency of convergence and optimality of solutions. Results indicated that OX is a more effective crossover

operator than PMX for both high precision and resource limited applications. Through further studies and experimentation genetic operators will only become more effective as these studies examine the delicate balance between exploration and exploitation.

Bibliography

Appendix A - Genetic Algorithm Internals and Advanced Topics > Crossover of Enumerated Chromosomes. (n.d.). Retrieved June 9, 2022, from http://www.wardsystems.com/manuals/genehunter/crossover_of_enumerated_chromosomes.htm

Bierwirth, C., & Mattfeld, D. C. (1999). Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, *7*(1), 1–17. Scopus. https://doi.org/10.1162/evco.1999.7.1.1

Calculus/Extrema and Points of Inflection—Wikibooks, open books for an open world. (n.d.). Retrieved March 12, 2022, from https://en.wikibooks.org/wiki/Calculus/Extrema_and_Points_of_Inflection

CernerEng (Director). (2016, July 12). *Genetic Algorithms—Jeremy Fisher*. https://www.youtube.com/watch?v=7J-DfS52bnI

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

D Fraser, A. B. (1970). *Computer Models in Genetics*. McGraw-Hill.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. Scopus. https://doi.org/10.1109/4235.996017

Fig. 2. Roulette wheel selection example. (n.d.). ResearchGate. Retrieved March 16, 2022, from https://www.researchgate.net/figure/Roulette-wheel-selection-example_fig2_251238305

Herath, A. K., & Wilkins, D. E. (2018, June). *Timetabling with Three-Parent Genetic Algorithm: A Preliminary Study*. International Conference of Control, Dynamic Systems, and Robotics. https://doi.org/10.11159/cdsr18.127

Kitjacharoenchai, P., Ventresca, M., Moshref-Javadi, M., Lee, S., Tanchoco, J. M. A., & Brunese, P. A. (2019). Multiple traveling salesman problem with drones: Mathematical model and heuristic approach. *Computers & Industrial Engineering*, *129*, 14–30. https://doi.org/10.1016/j.cie.2019.01.020

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, *13*(2), 129–170. Scopus. https://doi.org/10.1023/A:1006529012972

Maimon, O. Z., & Braha, D. (1998). A genetic algorithm approach to scheduling PCBs on a single machine. *International Journal of Production Research*, *36*(3), 761–784. https://doi.org/10.1080/002075498193688

Mingozzi, A., Bianco, L., & Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, *45*(3), 365–377. Scopus. https://doi.org/10.1287/opre.45.3.365

Naimi, H. M., Shahhoseini, H. S., & Naderi, M. (2003). *Uniformly distributed sampling: An exact algorithm for GA's initial population in a tree graph*. 185–189. Scopus.

Ochoa, G. (2006). Error thresholds in genetic algorithms. *Evolutionary Computation*, *14*(2), 157–182. Scopus. https://doi.org/10.1162/evco.2006.14.2.157

P np np-complete np-hard.svg. (n.d.). Wikepedia. Retrieved September 2, 2022, from https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, *31*(12), 1985–2002. Scopus. https://doi.org/10.1016/S0305-0548(03)00158-8

Robinson, J. (1949). *On the Hamiltonian Game (A Traveling Salesman Problem)* (https://apps.dtic.mil/dtic/tr/fulltext/u2/204961.pdf). The RAND Corportation; Wayback Machine. https://web.archive.org/web/20200629071813/https://apps.dtic.mil/dtic/tr/fulltext/u2/204961.pdf

Stanislawska, K., Krawiec, K., & Kundzewicz, Z. W. (2012). Modeling global temperature changes with genetic programming. *Computers & Mathematics with Applications*, *64*(12), 3717–3728. https://doi.org/10.1016/j.camwa.2012.02.049

Storn, R., & Price, K. (1997). Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, *11*(4), 341–359. Scopus. https://doi.org/10.1023/A:1008202821328

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems. *Operations Research*, *60*(3), 611–624. https://doi.org/10.1287/opre.1120.1048

郭飞. (2022). *Scikit-opt* [Python]. https://github.com/guofei9987/scikit-opt (Original work published 2017)

Appendix

```
    ax[0].set_ylabel('distance', fontsize='smaller')

    # ax[1].hist2d(ga_tsp.scatterX, ga_tsp.scatterY, bins=(50, 20), cmap=plt.cm.Blues)
    # ax[1].scatter(ga_tsp.scatterX, ga_tsp.scatterY, 0.1)
    ax[1].plot(range(len(ga_tsp.medianHistory)), ga_tsp.medianHistory)
    ax[1].set_xlabel('generation/iteration', fontsize='smaller')
    ax[1].set_ylabel('tour distance', fontsize='smaller')
    ax[1].set_title('distance of each tour over generations ({0}/gen)'.format(sizeOfPop), fontsize='smaller')

    # ax[2].plot(ga_tsp.standardDeviationHistory)
    # ax[2].set_xlabel('generation/iteration', fontsize='smaller')
    # ax[2].set_ylabel('standard deviation of generation', fontsize='smaller')

    plt.show()
iter = 10000
sizeOfPop = 200
num_points = 20
# points_coordinate = np.random.rand(num_points, 2)  # generate random coordinate of points
points_coordinate = np.array(
        generatePointsCoordinates_Circle(num_points)
    )

def isTourCircle (tour):
    circleCollection = collections.deque(list(range(0, num_points)))

    for i in range(num_points):
        circleCollection.rotate(1)
        if np.array_equal(tour, np.array(circleCollection)) or np.array_equal(tour, np.flipud(np.array(circleCollection))):
            return True
    return False

def timeFunction (func) :
    preTime = time.time()
    funcReturn = func()
    rnTm = time.time() - preTime
    return (funcReturn, rnTm)

def RunIterativeExperiment (n):
    tours = []
    times = []
    distances = []
    generations = []
    accuracies = []
    for _ in range(n):
        tour, rnTime = timeFunction(runAlgo)
        tours.append(tour.best_x)
        times.append(rnTime)
        distances.append(*tour.best_y)
        generations.append(tour.generation)
        accuracies.append(isTourCircle(tour.best_x))
        print(_, rnTime)
    return zip(tours, times, distances, generations, accuracies)

# p = pandas.DataFrame(
#     data=RunIterativeExperiment(200),
#     columns=['tours', 'times', 'distance', 'generations', 'accuracy'],
# )
# p.to_excel('results.xlsx', sheet_name='ox1')

# runAlgo()

tour = runAlgo()
plotTSP(tour)
```

Appendix 1. High level code written in Python interacting with GA library and recording results