Single-Depot Multiple Travelling Salesman Problem optimisation using a Genetic algorithm and a Particle Swarm Optimisation algorithm

Mihai Bojescu
Master in Artificial Intelligence and optimisation
Faculty of Computer Science of University "Alexandru Ioan Cuza" of Iași
Iași, Romania
bojescu.mihai@gmail.com

***Abstract*—This document contains a study on optimising the Single-Depot, Multiple Travelling Salesman Problem using a combinatorial Genetic algorithm and a combinatorial Partile Swarm Optimisation algorithm. The studied problem is presented in detail, along with the algorithms, their inner workings, diagrams, hyperparameters, timings and results. The study was conducted on the eil51, berlin52, eil76, rat99 datasets with 2, 3 and 5 salesmen.**

***Index Terms*—Single-Depot Multiple Travelling Salesman Problem, Genetic algorithm, Particle Swarm Optimisation algorithm, SD-MTSP, GA, PSO, TSPLIB, optimisation, combinatorial function optimisation.**

I. Introduction

The Multiple Traveling Salesman Problem (MTSP) is a generalization of the well-known Traveling Salesman Problem (TSP), where multiple salesmen are involved to visit a given number of cities exactly once and return to the initial position with the minimum traveling cost. MTSP is highly related to other optimisation problems such as Vehicle Routing Problem (VRP) and Task Assignment problem [1]. In this study, we will optimise the Single-Depot, Multiple Travelling Salesman Problem, which is a variation of MTSP.

Genetic algorithms were proposed by John Holland in 1973 after many years of studying the idea of simulating evolution. These algorithms model genetic inheritance and the Darwinian struggle for survival. Together with two other directions: evolutionary strategies and evolutionary programming, they form the class of evolutionary algorithms [2]. In this study, we will be using a combinatorial GA.

Particle Swarm Optimisation (PSO) algorithm is one of the most well-regarded algorithm in the literature of stochastic optimisation approaches. It belongs to the family of swarm-based techniques and is a population-based algorithm. As GA and ACO, it is of the most well-regarded algorithm in the literature. There are different versions of this algorithm in the literature to solve constrained, dynamic, discrete, multi-objective, multi-modal, and many-objective problems [3]. In this study, we will be using a modified PSO which can work with combinatorial problems.

This paper will apply the algorithms stated above in order to optimise the problem stated above.

II. Single-Depot, Multiple Travelling Salesman Problem

MTSP is one of the most important optimisation problems, and it has been applied in several real-life scenarios. Depending on the application requirements, the salesmen in MTSP can be represented by ground vehicles such as robots or trucks, or by flying vehicles such as Unmanned Aerial Vehicles (UAVs) known also as drones. Whereas the cities to be visited by the salesmen can have different representations, such as customers in transportation and delivery services, sensor nodes for Wireless Sensor Networks data collection, targets in military applications, victims in emergency missions and critical sites in disaster management applications [1].

MTSP is a multi-goal problem. Given n salesmen and m cities, the problem seeks to optimise the following two goals:

1) To minimise the total tour cost of n salesman that have to visit m cities, at the end returning to the "home" city
2) To minimise the tour cost of n salesmen that have to visit m cities, at the end returning to the "home" city

Mathematically, MTSP can be expressed in the following way: Given $n \in \mathbb{N}$ salesmen and $m \in \mathbb{N}$ cities, the tour definition

$$tour_i = \{city | city \in \{1, ..., m\}\}, i \in \{1, ..., n\} \quad (1)$$

$$tour_i \cap tour_j = \emptyset, i \neq j, i \in \{1, ..., n\}, j \in \{1, ..., n\} \quad (2)$$

$$\bigcup_{i=1}^{n} tour_i = \{1, ..., m\} \quad (3)$$

$$|tour| \leq \lfloor \frac{m - 1 + n}{n} + 1 \rfloor \quad (4)$$

and the cost function

$$cost(tour) = (\sum_{i=1}^{k-1} distance^2(city_i, city_{i+1})) + distance^2(city_k, city_1), k = |tour| \quad (5)$$

find a combination of tours such that

$$minimise(\sum_{i=1}^{n} cost(tour_i)) \tag{6}$$

and

$$minimise(cost(tour_i)), \forall i \in \{1, ..., n\} \tag{7}$$

are satisfied.

The MTSP problem can be optimised using multiple heuristic algorithms: Artificial Neural Networks (ANNs), Simulated Annealing algorithms (SAs), Particle Swarm Optimisation algorithms (PSOs), Genetic algorithms (GAs), Ant Colony Optimisation algorithms (ACOs) etc. [4].

III. Genetic Algorithm

A. Introduction

In literature, Genetic Algorithms are regarded as highly effective for optimising the MTSP, thus they are extensively used (Xu et al. 2018) due to their convergence speed, their main drawback being their high dependence on a well-generated, diverse initial population.

Multiple Genetic Algorithms were proposed, ranging from their chromosome representation and used genetic operators.

For chromosome representations, Tang et al. (2000) suggested a one-chromosome representation for optimising MTSP and was used for tackling the hot rolling production scheduling problem. Carter and Ragsdale (2006) proposed a two-part chromosome representation and relevant operators. Brown et al. (2007) proposed a one-chromosome and two-chromosome representation, while Yuan et al. (2013) proposed another two-part chromosome representation algorithm.

As for genetic operators, many used simple swap, slide and reverse-swap operators for the mutation operator. For the crossover operator, Király and Abonyi (2011) proposed a two-point crossover operator, Lo et al (2018) proposed a edge-recombination crossover operator, Sedighpour et al. (2012) proposed an order crossover operator, and Singh et al. (2018) proposed a distance-preserving crossover operator for solving the MTSP.

B. Hyperparameters

In this paper, the following hyperparameters were chosen for the genetic algorithm:

1) population size = 100
2) generations = 2000
3) mutation probability = 0.1

C. Population generation

In this paper, a population of 100 individuals was generated randomly and was segmented in multiple tours with their maximum length according to formula 4, in order to match the tour sizes in the provided datasets [5].

D. Chromosome representation

In this paper, a one-chromosome representation was used, where each chromosome was wrapped in an class named "Individual" that handled its own fitness calculations using formula 5. Each "Individual" represents a full solution for the MTSP. The tours in the "Individual" are entities of type `t.List[t.List[int]]` with their home depot omitted, and the fitness value of the solution is represented as a `float` variable. An example of an individual can be found in the code listing below.

```
Individual(
    genes=[
        [39, 47,  4, 24, 25, 26, 27, 11,
         50, 32, 42,  9,  8,  2, 48, 34,
         43, 15, 28, 49, 19, 22, 30, 20],
        [46, 12, 13, 51, 10,  3,  5, 14,
         37, 23, 45, 36, 38, 35, 33, 21,
         31, 44,  7, 40, 18, 17, 16,  6,
          1, 41, 29]
    ],
    fitness=10361.20300592295
)
```

Listing 1: Example of individual

E. Algorithm

The base algorithm was not modified, as it did not need any modifications performed. Mainly, the only change needed was to perform dependency injection on the operators, in order to ensure that the algorithm respects the Open-Closed principle.

F. Selection operator

In this paper, a tournament selection operator was used, where $size(tournament) = 10$, thus from a random selection of 10 individuals, only the best one is picked. There were attempts to use the roulette wheel selection, but the results of using the tournament selection operator were better.

G. Crossover operator

For the crossover operator, the distance-preserving crossover operator was used as described by Singh et al. (2018) [7]. The operator, originally used on chromosomes with two-part representation, was *trivially* adapted to work using one-chromosome representations. In order for the operator to be used, an encoder was created that, given a segmentation, can encode the tours from a `t.List[t.List[int]]` to a `t.List[int]` and decode them back.

The operator, given parents $parent_1$ and $parent_2$ produces children $child_1$ and $child_2$ in the following manner:

1) Encode the tours of $parent_1$ into a list of ints, $parent_1^e$
2) Encode the tours of $parent_2$ into a list of ints, $parent_2^e$
3) Create child $child_1^e$ of length $|parent_1^e|$
4) Create child $child_2^e$ of length $|parent_2^e|$
5) Copy last gene of $parent_2^e$ into first gene of $child_1^e$
6) Copy last gene of $parent_1^e$ into first gene of $child_2^e$

7) Copy first gene of $parent_2^e$ into last gene of $child_1^e$
8) Copy first gene of $parent_1^e$ into last gene of $child_2^e$
9) For $i = \{1, ..., |parent_1^e|\}, j = \{2, ..., |parent_1^e| - 1\}$:
   a) If i^{th} gene of $parent_2^e$ is equal to j^{th} gene of $parent_1^e$, then copy j^{th} gene of $parent_2^e$ into j^{th} gene of $child_1^e$
   b) If i^{th} gene of $parent_1^e$ is equal to j^{th} gene of $parent_2^e$, then copy j^{th} gene of $parent_1^e$ into j^{th} gene of $child_2^e$
10) Decode $child_1^e$ into the tours for $child_1$
11) Decode $child_2^e$ into the tours for $child_2$

An example can be found in the figure below:

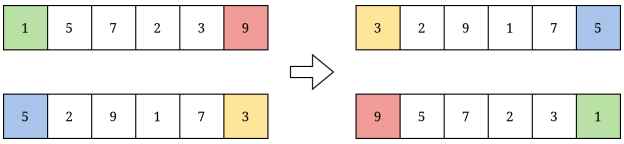

Fig. 1: Crossover operator

The justification behind this crossover strategy depends on the idea that the city in optimal/suboptimal tour takes place in the same location [6].

H. Mutation operator

The mutation operator in this paper is comprised of 3 operations, each executed according to the mutation probability hyperparameter:

1) Swap operation
2) Slide operation
3) Reverse-swap operation

The swap operation, randomly picking two position a and b with $a, b \leq |genes_{child}|$, performs a simple swap between gene a and gene b of the child. A sample can be found in figure 2 below.

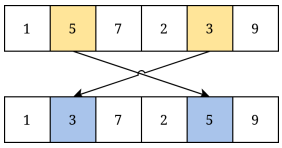

Fig. 2: Swap operation

The slide operation, randomly picking two position a and b with $a, b \leq |genes_{child}|$, moves gene a to gene b and slides each gene before gene b one step before. A sample can be found in figure 3 below.

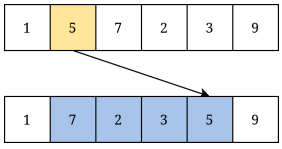

Fig. 3: Slide operation

The reverse-swap operation, randomly picking two position a and b with $a, b \leq |genes_{child}|$, reverses the genes array between gene a and gene b, including gene a and gene b. A sample can be found in figure 4 below.

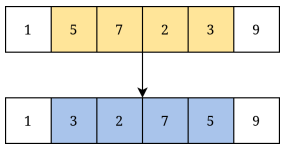

Fig. 4: Reverse-swap operation

IV. Particle Swarm Optimisation

A. Introduction

PSO is a wonderful optimisation technique, inspired from the flocking behaviour of birds. It was successfully applied in many fields that implied numerical function optimisations, fields such as industry engineering, civil engineering, energy systems engineering, electrical engineering and geology engineering [8].

In order to be able to apply PSO to MTSP, several changes needed to be performed. The motion aspect was transformed in order to be applied for combinatorial functions. This will be detailed in the following section, IV-B.

B. From motion to operators

PSO is well known for being an algorithm which incorporates motion. The particles "move" through the problem space, exploring it according to their hyperparameters:

1) inertia
2) their tendency towards their personal best
3) their tendency towards their global best

Along their some randomly parameters chosen at each step, in order to explore the space better.

The above statement works perfectly for numerical functions, but is troublesome for combinatorial functions. The values needing to be discrete, can cause a lot of aproximation errors, thus, in this paper, the concepts were replaced:

1) For velocity and inertia, a 2-opt algorithm was used instead
2) For tendency towards a particle's personal best, the path-relinker from the GRASP algorithm was used
3) For tendency towards a particle's global best, the path-relinker from the GRASP algorithm was used
4) For randomness, the same swap, slide and reverse-swap operator from the Genetic Algorithm (section III-H) was used

C. Hyperparameters

In this paper, the following hyperparameters were chosen for the Particle Swarm Optimisation algorithm:

1) population size = 100
2) iterations = 100
3) 2-opt probability = 0.2
4) path-relinker towards personal best probability = 0.4
5) path-relinker towards global best probability = 0.3
6) swap probability = 0.1

D. Population generation

Same as the Genetic algorithm in section III-C, the population was generated randomly, without any changes.

E. Algorithm

The algorithm performs the following steps, as described in the bibliography [8]:

1) $\forall individual \in population$:
   a) Select a random number $k, k \in \{1, 2, 3, 4\}$ respecting the input hyperparameters
   b) If $k = 1$, run the 2-opt algorithm operator on the $individual$
   c) If $k = 2$, run the path-relinker algorithm operator on the $individual$, with their $individual$'s personal best as the guiding solution
   d) If $k = 3$, run the path-relinker algorithm operator on the $individual$, with the global best as the guiding solution
   e) If $k = 4$, run the swap operator on $individual$
2) Update the individual best of each individual
3) Update the global best

A flow diagram of the algorithm can be found in figure 5 below:

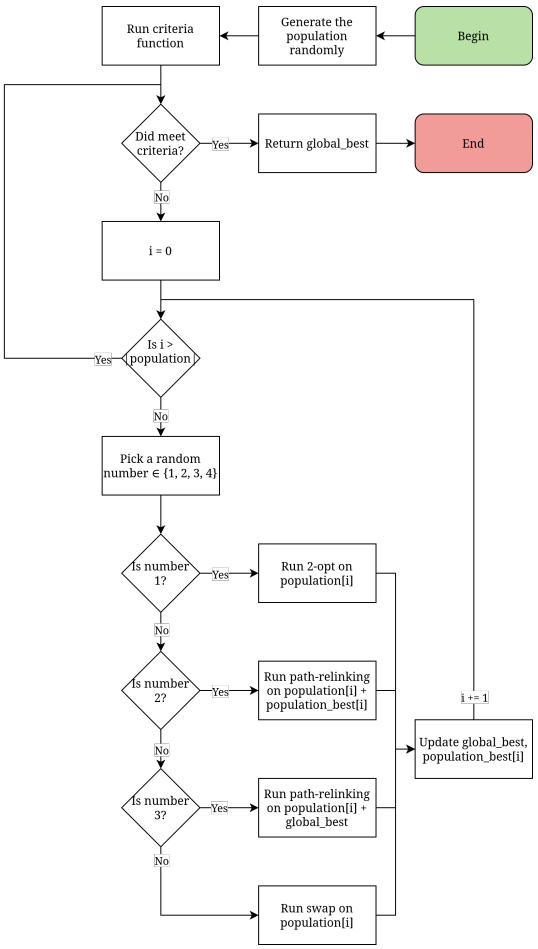

Fig. 5: Combinatorial PSO algorithm

F. 2-opt algorithm operator

The 2-opt algorithm is a local search algorithm first proposed by Croes in 1958, with the bassic moves suggested by Flood. It aims to recombine edges in the MTSP tour such that better alternatives are found, until no more updates can be performed. At each, the algorithm checks wether the objective function returned better result than the currently known one [8]. The main idea of the algorithm is to ensure that there are no edges in a route that cross over themselves [9].

A diagram showing how the algorithm works is provided in figure 6 below.

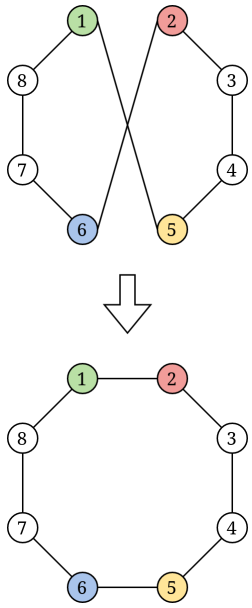

Fig. 6: 2-opt algorithm sample

This operator aims to replace a PSO's particle's velocity and independent movement.

G. Path-relinker algorithm operator

The relinking procedure (Glover et al. 2004) is used to generate new solutions by exploring trajectories confined in the neighbourhood space that connect high-quality solutions. The solution that begins the path is named the initiating solution, while the solution ending the path is named the guiding solution [10].

In literature, multiple variants can be found for this procedure, categorised by how the path is built between the initiating solution and the guiding solution: forward relinking, backward relinking, backward-and-forward relinking, relinking (Glover et al. 1996, implemented by Resende and Ribeiro 2010a), greedy randomised adaptive path relinking and evolutionary path-relinking [11]. Each variant has its own set of advantages and disadvantages. In the computational experiments performed by Aiex et al. (2005), backward relinking stood out as better performing than forward relinking, and had equivalent performance as backward-and-forward relinking, but with lower computational costs [11].

In this paper, the backward path relinking was used. Thus, the initial solution was the current individual's position and the guiding solution was - depending on the function call - the individual's best position or the global best position.

A sample can be found in figure 7 below. When step 4 is reached, no other better paths moving towards the given guiding solution could be found, thus the algorithm returns the solution built at step 4.

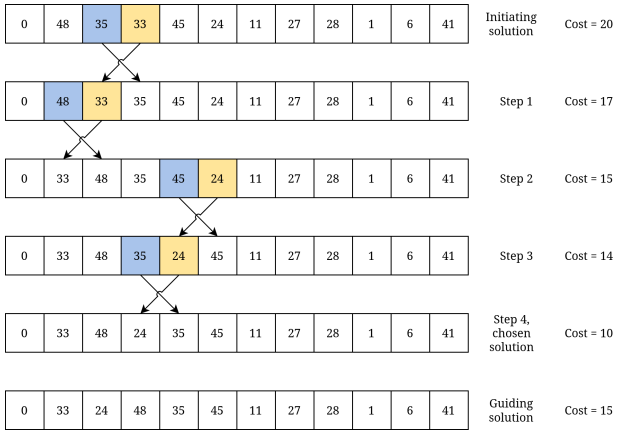

Fig. 7: Path-relinker algorithm sample

This operator aims to replace a PSO's particle's tendency towards their personal best and the global best.

H. Swap operator

For the PSO algorithm, the same swap + slide + reverse-swap operator as the genetic algorithm was used with remarkable success. During the first implementation of the algorithm, only a simple swap of two elements was used as in figure 2, but the results it produced were 10-20% worse overall. As an example, for the eil51 dataset with 2 salesmen, the simple swap operator produced a mean value of 569, while the mean value for the swap + slide + reverse-swap operator was 489.

This operator aims to enhance PSO's exploration of the problem space by introducing more randomness with which 2-opt and path-relinker can work.

V. Results

The algorithms were run using the following datasets:

1) `eil51`, a 51-city MTSP problem by Christofides/Eilon
2) `berlin51`, a 52-locations MTSP problem by Groetschel based on real-life Berlin, Germany
3) `eil76`, a 76-city MTSP problem by Christofides/Eilon
4) `rat99`, a 99-locations MTSP problem by Pulleyblank

Each of the datasets was run with $n \in \{2, 3, 5\}$ salesmen. The runs were executed on a machine with an Intel Core i7 1260p processor at a constant frequency of 2GHz, using 10 of the 12 physical cores available. The algorithms can run on as little as 1GB of RAM. In order to achieve good timings, the CPU should run on as a high frequency as it can achieve, as the algorithms are compute-intensive.

The results of the runs can be seen in tables 8, 9 for the genetic algorithm and tables 10, 11 for the PSO, and visualisations are available on figures 12, 13, 14, 15, 16, 17, 18 and 19 on the next pages.

VI. Discussions

The two algorithms detailed in this paper were successful in finding near-optimal results for the MTSP problem datasets used in this paper. Their runtimes - albeit high from a normal user perspective - exhibit a notable reduction in comparison when compared to those of the CPLEX algorithm (GA 1m36s, PSO 4m45s, CPLEX 10080m for `rat99`, $n = 5$ salesmen). In this paper, the Genetic algorithm provided better results, was faster overall and needed far fewer modifications to work with combinatorial data. As for the particle swarm optimisation algorithm, the runtime was higher, but the results were comparable to the genetic algorithm.

VII. Conclusions

Optimisation algorithms like Genetic Algorithms and Particle Swarm Optimisation - often used for numerical function optimisation - can be used for combinatorial function optimisation like MTSP with some modifications. The results of running the alforementioned algorithms on combinatorial functions are often near-optimal, all while showing superior time efficiency.

The algorithms can further be improved through stochastic hyperparameter tuning and through hybridisation. In literature, there were attempts to hybridise the Ant Colony Optimisation algorithm with the 2-opt operator [12] and to hybridise a particle swarm optimisation algorithm using the GRASP algorithm [8] with good results.

References

[1] O. Cheikhrouhou, I. Khoufi (2021). "A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy". Computer Science Review, 40, 100369. doi: 10.1016/j.cosrev.2021.1003.

[2] L. Henri, E. Croitoru (2022). "Genetic algorithms". Faculty of Computer Science of University "Alexandru Ioan Cuza" of Iași.

[3] S. Mirjalili (2018). "Particle Swarm Optimisation. Evolutionary Algorithms and Neural Networks". 15-31. doi: 10.1007/978-3-319-93025-1_2.

[4] M. A. Al-Furhud, Z. H. Ahmed (2020). "Experimental Study of a Hybrid Genetic Algorithm for the Multiple Travelling Salesman Problem". Mathematical Problems in Engineering, vol. 2020, p. 3431420. doi: 10.1155/2020/3431420.

[5] https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/index.html

[6] P. Singamsetty, J. K. Thenepalledoi (2021). "An efficient genetic algorithm for solving open multiple travelling salesman problem with load balancing constraint". Department of Mathematics, School of Advanced Sciences, VIT, Vellore-632014, Tamil Nadu, India. doi: 10.5267/j.dsl.2021.5.003

[7] D. R. Singh, M. K. Singh, T. Singh, R. Prasad (2018). "Genetic algorithm for solving multiple traveling salesmen problem using a new crossover and population generation". Computación y Sistemas, 22(2).

[8] S. D. Gulcu, H. K. Ornek (2019). "Solution of Multiple Travelling Salesman Problem using Particle Swarm Optimization based Algorithms". International Journal of Intelligent Systems and Applications in Engineering. ISSN: 2147-6799

[9] https://en.wikipedia.org/wiki/2-opt

[10] B. Penga, Zhipeng Lü, T.C.E. Cheng (2014). “A Tabu Search/Path Relinking Algorithm to Solve the Job Shop Scheduling Problem”. hool of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, P.R. China. doi: 10.48550/arXiv.1402.5613

[11] C. C. C. Ribeiro, M. G. C. Resend (2010). “Path-relinking intensification methods for stochastic Local Search Algorithms”. Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria, 156, Niteráoi, RJ 24210-240 Brazil. URL: https://mauricio.resende.info/doc/spr.pdf

[12] K. Saraswathi, A. Tamilarasi (2016). “Ant Colony Optimization Based Feature Selection for Opinion Mining Classification”. Journal of Medical Imaging and Health Informatics, Volume 6. doi: 10.1166/jmihi.2016.1856

Dataset	Dimensions	Best cost [min, max]	Median cost [min, max]	Worst cost [min, max]
eil51	2	452.4917, [205.7164, 246.7753]	489.3968, [242.2249, 247.1719]	515.7682, [221.6817, 294.0865]
	3	498.3084, [156.9638, 183.4012]	518.9448, [160.8375, 184.4372]	566.9566, [161.5549, 207.9621]
	5	572.3836, [77.3725, 131.3495]	661.8094, [101.8904, 150.5947]	733.4215, [68.8432, 183.4685]
berlin52	2	8201.6263, [2595.4587, 5606.1676]	8590.6848, [2697.2115, 5893.4733]	9307.7586, [2992.1265, 6315.6321]
	3	8996.8522, [2115.4115, 4001.3250]	9594.4198, [1223.1735, 4365.7421]	10863.9695, [1867.7822, 4966.9029]
	5	10892.5415, [578.9702, 3015.3396]	11556.2447, [931.1152, 2959.9547]	11912.2920, [700.1437, 2974.0190]
eil76	2	581.0691, [276.2812, 304.7880]	622.9683, [300.8131, 322.1552]	670.6645, [289.7983, 380.8662]
	3	638.9018, [195.2157, 222.1578]	687.3440, [189.9131, 252.9988]	737.6305, [180.3664, 292.0079]
	5	801.1751, [131.5410, 170.4222]	878.9306, [125.3454, 189.6171]	947.2019, [119.7203, 219.1424]
rat99	2	1478.4560, [726.2807, 752.1753]	1564.7694, [778.9769, 785.7925]	1641.2690, [799.5244, 841.7446]
	3	1738.7757, [561.4327, 610.7246]	1873.4486, [604.1806, 644.0537]	1936.9353, [581.5455, 680.7555]
	5	2419.9191, [404.0482, 520.0018]	2569.2387, [410.6488, 548.4963]	2757.0272, [456.9319, 604.7306]

Fig. 8: Fitness results for the genetic algorithm

Dataset	Dimensions	Best time (seconds)	Median time (seconds)	Worst time (seconds)
eil51	2	35.0730	39.3025	40.2845
	3	36.6657	40.6693	40.9740
	5	38.3636	40.4140	42.9961
berlin52	2	35.8471	40.5474	40.7477
	3	36.7146	41.5331	42.1390
	5	38.5821	43.4023	44.1265
eil76	2	59.2793	62.9946	63.8993
	3	61.3011	64.2504	65.2739
	5	62.8406	66.3060	67.2039
rat99	2	88.9989	91.9548	93.5630
	3	88.7005	93.2481	94.1409
	5	90.4842	94.3258	96.2429

Fig. 9: Runtime results for the genetic algorithm

Dataset	Dimensions	Best cost [min, max]	Median cost [min, max]	Worst cost [min, max]
eil51	2	449.7163 [214.1512, 235.5651]	475.5942, [192.9591, 282.6351]	499.3102, [244.9411, 254.3691]
	3	489.5325, [157.1746, 166.5948]	531.2806, [142.3028, 196.6333]	578.3335, [181.7417, 201.7480]
	5	640.2244, [88.7013, 146.2878]	697.6553, [102.2351, 153.4138]	784.5660, [117.1869, 173.1557]
berlin52	2	7976.8215, [2996.0896, 4980.7319]	8415.4363, [2969.9692, 5445.4672]	8647.5823, [4186.8835, 4460.6987]
	3	8771.5855, [2360.1379, 3811.5594]	9661.0789, [3082.9605, 3393.0226]	10190.7955, [2871.2697, 3782.7879]
	5	11253.1301, [577.7584, 3023.0627]	12362.8712, [1312.6386, 3029.0983]	13593.6078, [1099.0985, 3678.7885]
eil76	2	582.0823, [267.7988, 314.2835]	609.2940, [258.9784, 350.3156]	639.0766, [306.9239, 332.1527]
	3	650.3413, [188.5105, 235.7174]	682.4711, [188.3825, 247.9872]	741.4631, [234.9759, 256.2703]
	5	836.9620, [127.3422, 187.9968]	949.0180, [142.0018, 209.3440]	967.5711, [161.3675, 222.9497]
rat99	2	1404.4009, [677.7814, 726.6194]	1466.0078, [708.4931, 757.5147]	1530.4149, [761.7108, 768.7042]
	3	1705.9939, [436.4255, 666.6766]	1814.2634, [514.3615, 672.1688]	1968.9282, [620.0822, 677.6363]
	5	2460.4928, [327.3417, 574.1595]	2662.7770, [436.9524, 586.0860]	2779.0884, [436.0260, 609.4613]

Fig. 10: Fitness results for the particle swarm optimisation algorithm

Dataset	Dimensions	Best time (seconds)	Median time (seconds)	Worst time (seconds)
eil51	2	60.0396	86.3428	101.1629
	3	47.5369	61.4150	76.4317
	5	36.8183	48.7286	61.3814
berlin52	2	63.2038	91.6510	103.4645
	3	48.7222	59.6309	84.4854
	5	43.3237	54.1800	62.8043
eil76	2	187.6626	238.9693	287.5738
	3	145.2984	180.1258	215.3485
	5	110.0897	140.1826	165.7784
rat99	2	363.4923	511.2023	552.7536
	3	281.8676	348.8739	430.0076
	5	211.3089	288.1322	1411.9221

Fig. 11: Runtime results for the particle swarm optimisation algorithm

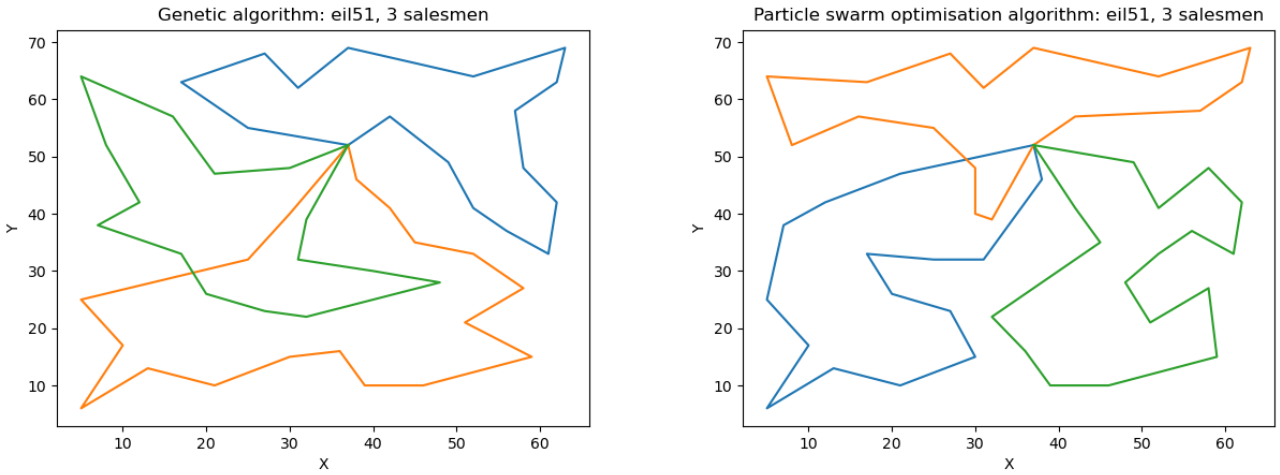

Fig. 12: `eil52`, 3 salesmen

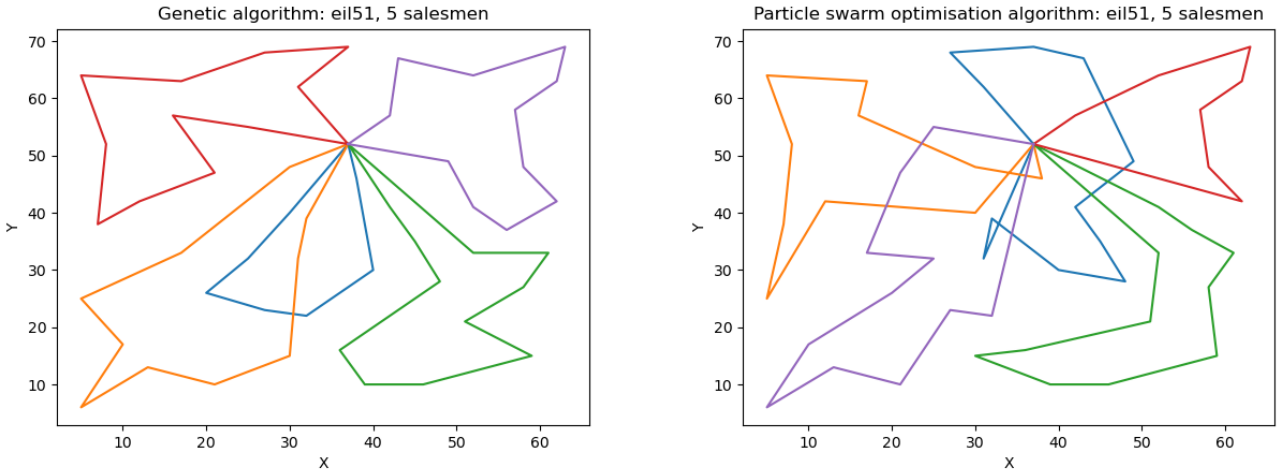

Fig. 13: `eil52`, 3 salesmen

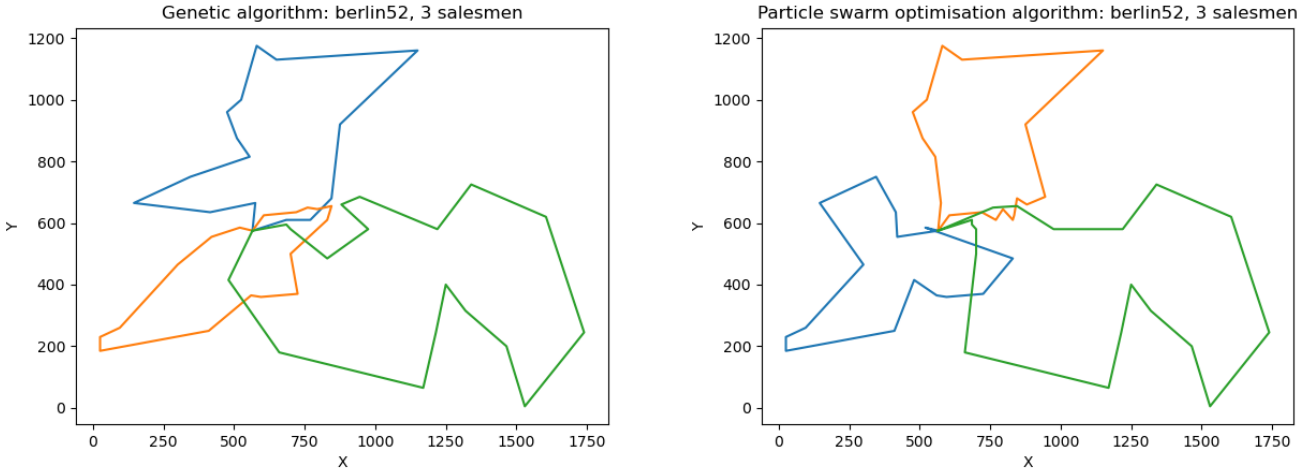

Fig. 14: `berlin52`, 3 salesmen

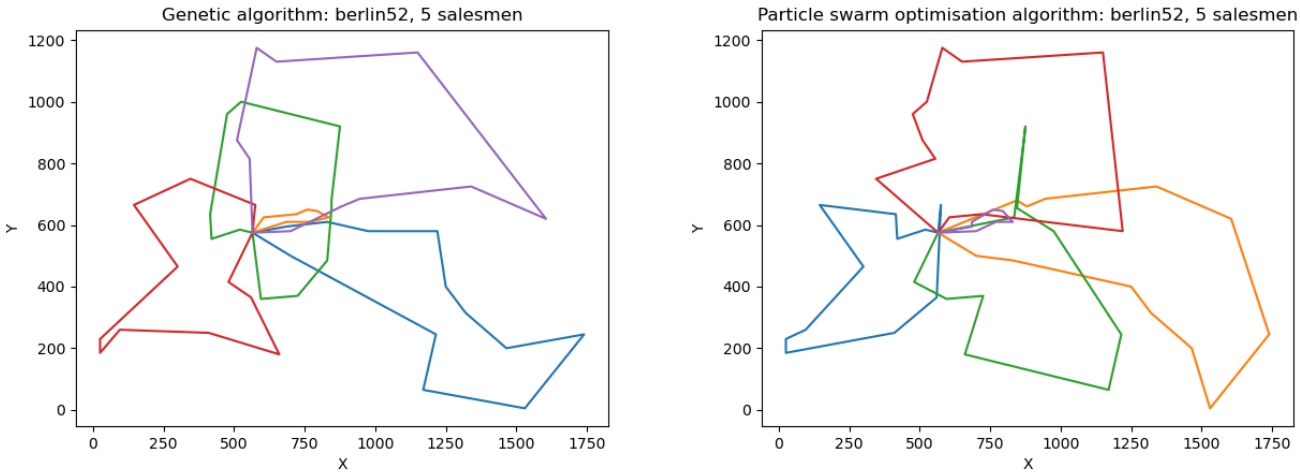

Fig. 15: `berlin52`, 5 salesmen

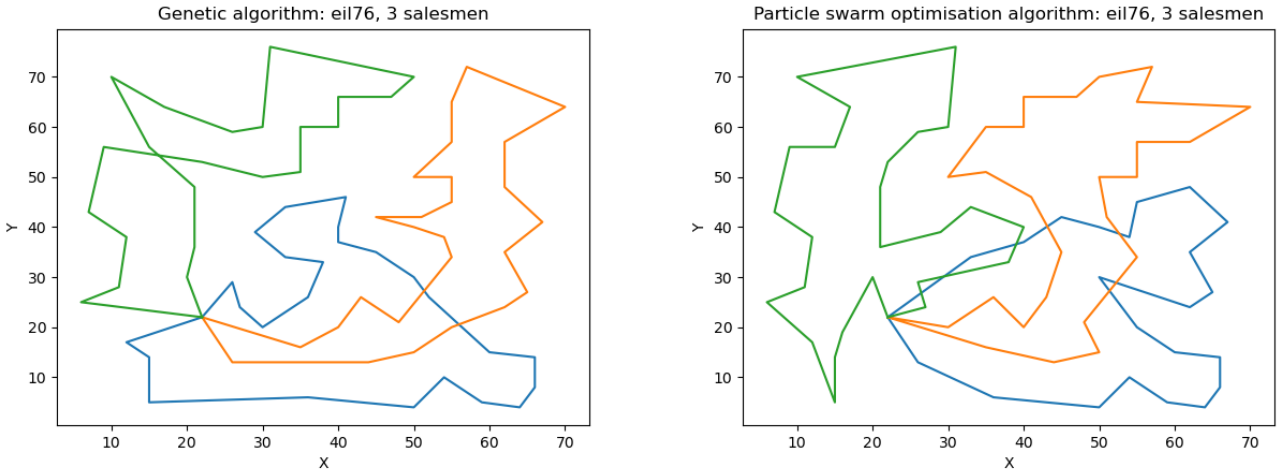

Fig. 16: `eil76`, 3 salesmen

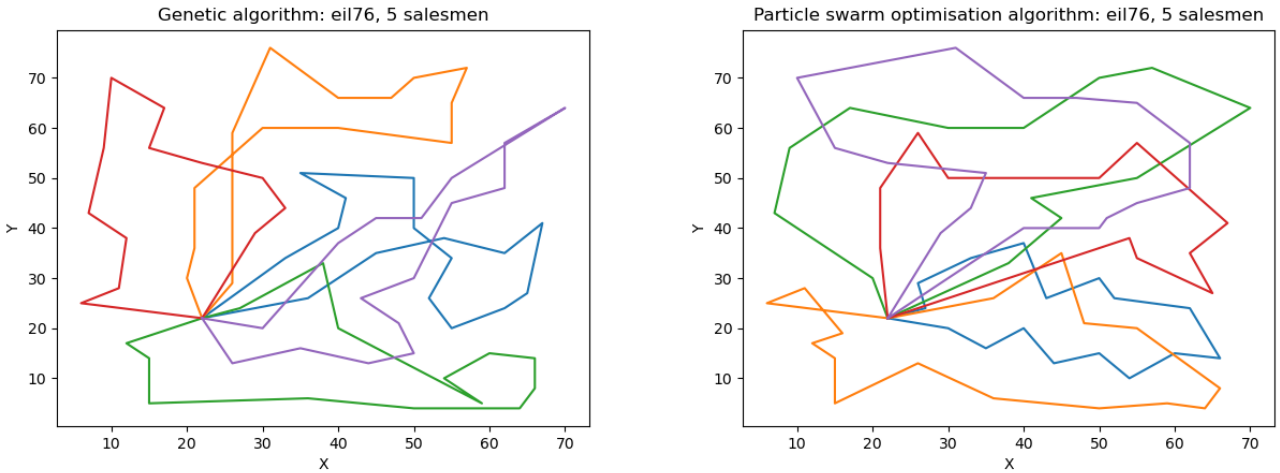

Fig. 17: `eil76`, 5 salesmen

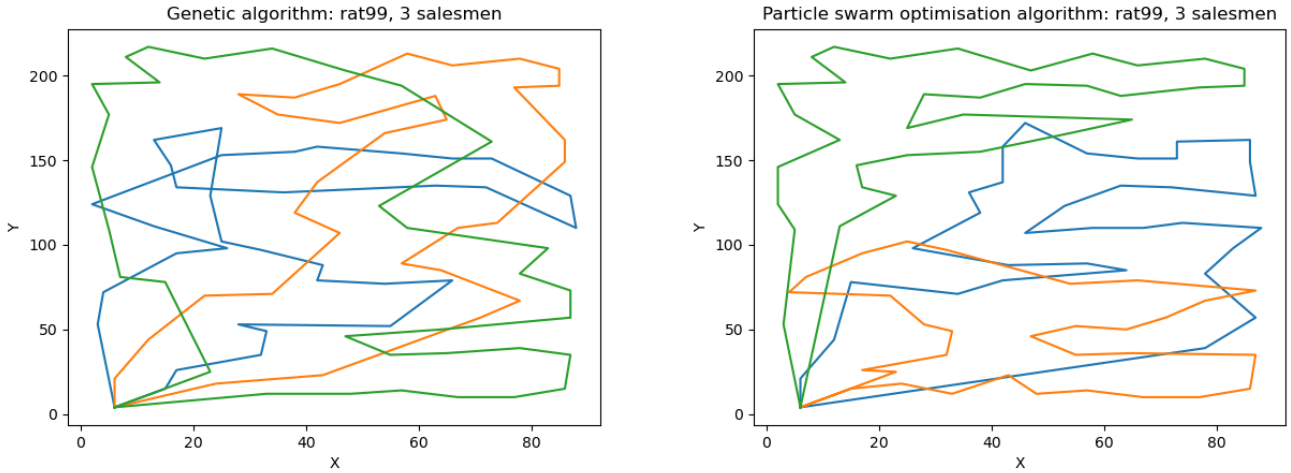

Fig. 18: `rat99`, 3 salesmen

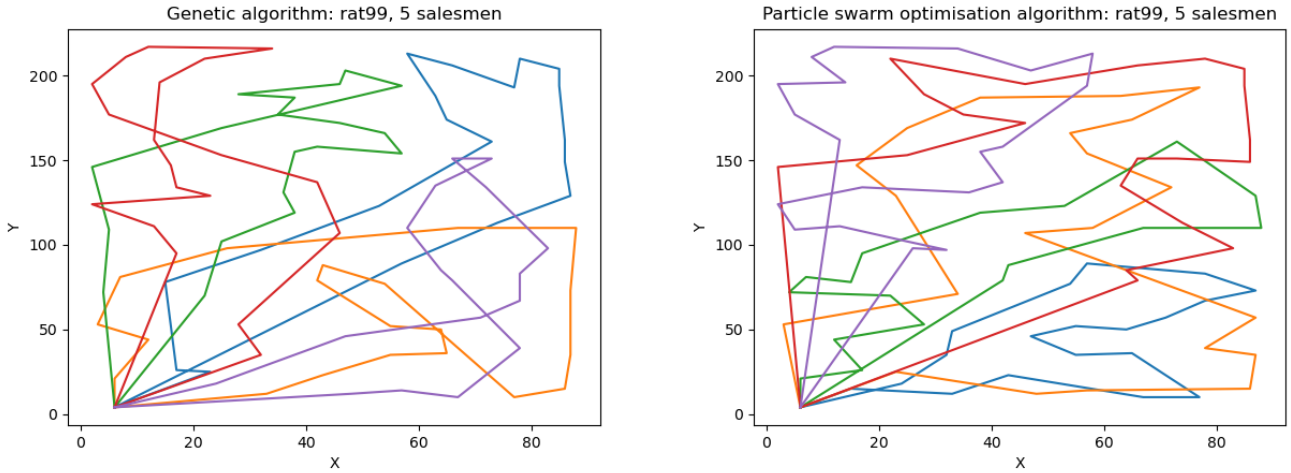

Fig. 19: `rat99`, 5 salesmen