Optimising the travelling thief problem using an evolutionary algorithm

Group J

1 Research Undertaken

The Travelling Thief Problem (TTP) is a multi-objective optimization problem combining the Travelling Salesman Problem (TSP) and the Knapsack Problem (KNP) with the main objective of reducing the travel time and make decisions about which items to pick up in order to maximise the value of items in the knapsack. In the course of our research, we went through various approaches to solve TTP, such as the weighted sum approach [1], ant colony optimisation [2] and simulated annealing technique [3]. The weighted sum approach transforms the multi-objective problem into a single-objective problem. Ant colony optimisation focuses mainly on optimal TSP tours and secondary on packing plans. This approach prioritizes finding effective TTP tours. It emphasizes better overall solutions over just optimizing travel routes. The simulated annealing technique offers an effective solution for the TTP by modifying the knapsack steps. It begins with a random item selection strategy and develops a travel plan using the Lin-Kernighan heuristic. This plan is further refined by eliminating less profitable cities through a modified simulated annealing approach. This method demonstrates superior performance in small and medium-sized instances and remains competitive in larger scenarios. We also researched the Non-dominated Sorting Genetic Algorithm -II algorithm (NSGA-II) [4] and other hybrid genetic algorithms. Based on this, we decided to develop an evolutionary algorithm inspired by the NSGA-II algorithm [5] and incorporate heuristic functions from a hybrid genetic algorithm [6]. NSGA-II's uses a fast sorting, crowding distance mechanism, and elitism strategy, allowing the best solutions from the previous generation to be carried forward to the next generation and preventing premature convergence, ensuring quality outcomes. It was chosen because it was familiar to the team members and could be broken down into manageable functions. In order to implement NSGA-II and make it less complex, we decomposed the TTP problem into smaller, manageable sub-problems, allowing us to perform various experiments on different parameters to effectively explore the results of each set of parameters, making it the optimal choice for our project.

2 Details of the Developed Algorithm

Problem Representation

As discussed previously, the algorithm which was implemented through this project was an NSGA-II inspired Evolutionary Algorithm. The efficient implementation of this algorithm required efficient representations of the distances between the nodes provided in each instance and the item to city mappings. Due to the large size of some of the provided instances, using a distance matrix to represent the distance between cities and a matrix to represent the item to city mappings was deemed inefficient. Instead, the distances between cities were stored in a one-dimensional pairwise distances array where the distance between city i and city j, such that $i < j < m$ where m is the number of cities, can be found in the $\lfloor m * i + j - ((i+2) * (i+1))//2 \rfloor$ position of the 0-indexed pairwise distances vector. The item to city mappings were stored in a dictionary where the keys were the city IDs and the values were lists of the item IDs which can be found at that specific city. Furthermore, the item weights and values were also stored in one-dimensional vectors.

Solution Representation

Following the problem definition, the solutions must comprise two elements: a tour plan and a packing plan. Both the tour plan and the packing plans were represented as one-dimensional arrays. The one-dimensional tour plan array contained the ID of the cities in the order of which they were visited by the thief. On the other hand, the packing plan array was a one-dimensional binary array of the length of the number of items. For example, if the thief picked item 2 then the element at the second index of the packing plan array was set to 1, and 0 if they did not pick it.

The Algorithm

The implemented algorithm aims to solve the Travelling Thief Problem taking some inspiration from both Blank et al. (2017), who compared the performance of greedy algorithm and NSGA-II in solving the TTP, and Wachter (2015) who used a hybrid evolutionary algorithm to solve the TTP. The implemented algorithm can be broken down into the parts shown in Figure 1 and described below.

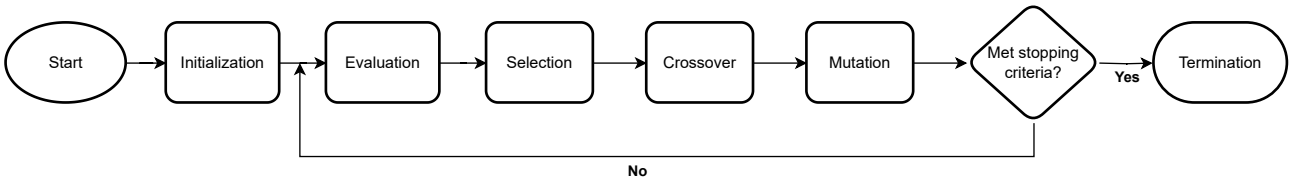

Figure 1: Flow chart of an evolutionary algorithm.

1. **Initialise the candidate solutions**:

   Tours: Initialise the tours randomly by creating a random permutation of the city IDs

   Packing Plan: Initialise the packing plan optimally using local heuristic function obtained via a combination of methods used in [7] and [6]. Compute the value to weight ratio for each item and sort the items descendingly according to this ratio. Then add each item to the packing plan in descending order with regards to the value to weight ratios whilst the total weight of the knapsack does not exceed uniformally sampled Q' on the interval $[0, Q]$ where Q is the maximum weight that the knapsack can handle.

2. **Evaluate the Initial Population**: Compute each candidate solution's tour cost and packing plan cost. Also compute each candidate solution's rank and crowding distance, as prescribed by NSGA-II [4].

3. **Perform NSGA-II Style Tournament Selection [4]**: Randomly sample, without replacement, two solutions from the population and return the solution with the best rank. If ranks are equal then return the solution with the greatest crowding distance. Perform this twice to obtain two parent solutions.

4. **Perform the Crossover Operation**:We used a methodology similar to [6] to perform the crossover on the parents selected by the tournament selection operator. Indeed, we randomly chose between crossing over the tour plan using single point ordered crossover or crossing over the packing plan using single point crossover to produce two children solutions.

   Deriving the Tour from the Packing Plan: If the packing plan was crossed over, we used a heuristic function, defined in [6], to derive the tour from the crossed over packing plan. The defined heuristic function ordered the tour plan such that the cities at which no items were picked up were visited first and the remaining cities were ordered in ascending order with regards to the weight picked at each city.

5. **Perform the Mutation Operations**: Following the crossover operation, both the tour plan and the packing plans of the children were mutated. The tour plan was mutated using single

swap mutation and the packing plan was mutated using bit flip mutation, where 25% of the mutations were flipped.

6. **Perform the Replacement Operation**: Finally, once a number of children equal to the population number have been generated through steps 3./4./5. a replacement operation is performed in the same way as for NSGA-II. The produced children are injected into the existing population. The costs of each candidate solution are computed, as well as their associated ranks and crowding distance. This population is then sorted first by rank ascendingly and then descendingly by crowding distance before being truncated back to the initial population size.

7. **Stopping the Algorithm**: Steps 3./4./5./6. are repeated until the algorithm has reached 100 generations.

3 Breaking the Problem Down

Once the representation of the evolutionary algorithm was determined, we thought about different initialisations, crossover, and mutation functions to perform experiments. This was broken down into 22 separate functions, which team members implemented. Table 1 shows the functions implemented and who wrote them.

Team Member	Functions
Jack	read_TTP_instance_data, random_tour, pack_one, repair_packing_plan, pack_random, pack_optimal,generate_initial_population, knapsack_cost, get_pdist_distance, tour_cost_pdist, evaluate_candidate_solution_cost_pdist, packing_plan_crossover,derive_tour_from_packing_plan, replacement
Ursula	get_ranks, get_crowding_distance, get_pareto_front, get_best_solutions
Nafees	packing_plan_mutation
Swaroop	tournament_selection
Kaiyuan	tour_mutation
Kanchan	OX1_tour_crossover

Table 1: Table showing the division of functions amongst the team members.

All of the functions were discussed during a team meeting, and a skeleton code, including the input, output and requirements, was created. This meant that all team members knew what they were implementing and had no unnecessary overlaps. Each member was assigned functions based on their abilities, knowledge of the problem, and available time. This led some team members to take on more functions than others. The code was reviewed in the next meeting, and any issues were identified and rectified quickly.

4 Results

Experiments were conducted on the smallest of the provided TTP instances **(a280-n279)** to identify the optimal parameters of the algorithm described in the previous section. The previously described algorithm was then ran on the larger TTP instances and Figure 1 displays the Pareto fronts obtained for all of the competition's specified TTP instances.
For the smaller instances, such as **a280-n279**, **a280-n1395**, **a280-n2790**, and **fnl4461-n4460**, it appears that the algorithm does not efficiently explore the entirety of the Pareto front, made evident by the "gaps" present in the Pareto front for each of the instances. This inevitably leads to suboptimal solutions due to the lack of diversity in the solutions.
On the other hand, for the medium-sized and large-sized instances, **fnl4461-n22300**, **fnl4461-n44600**, **pla33810-n33809**, **pla33810-n169045** and **pla33810-n338090**, Figure 1 indicates that the algorithm displays a greater exploration of the Pareto front.

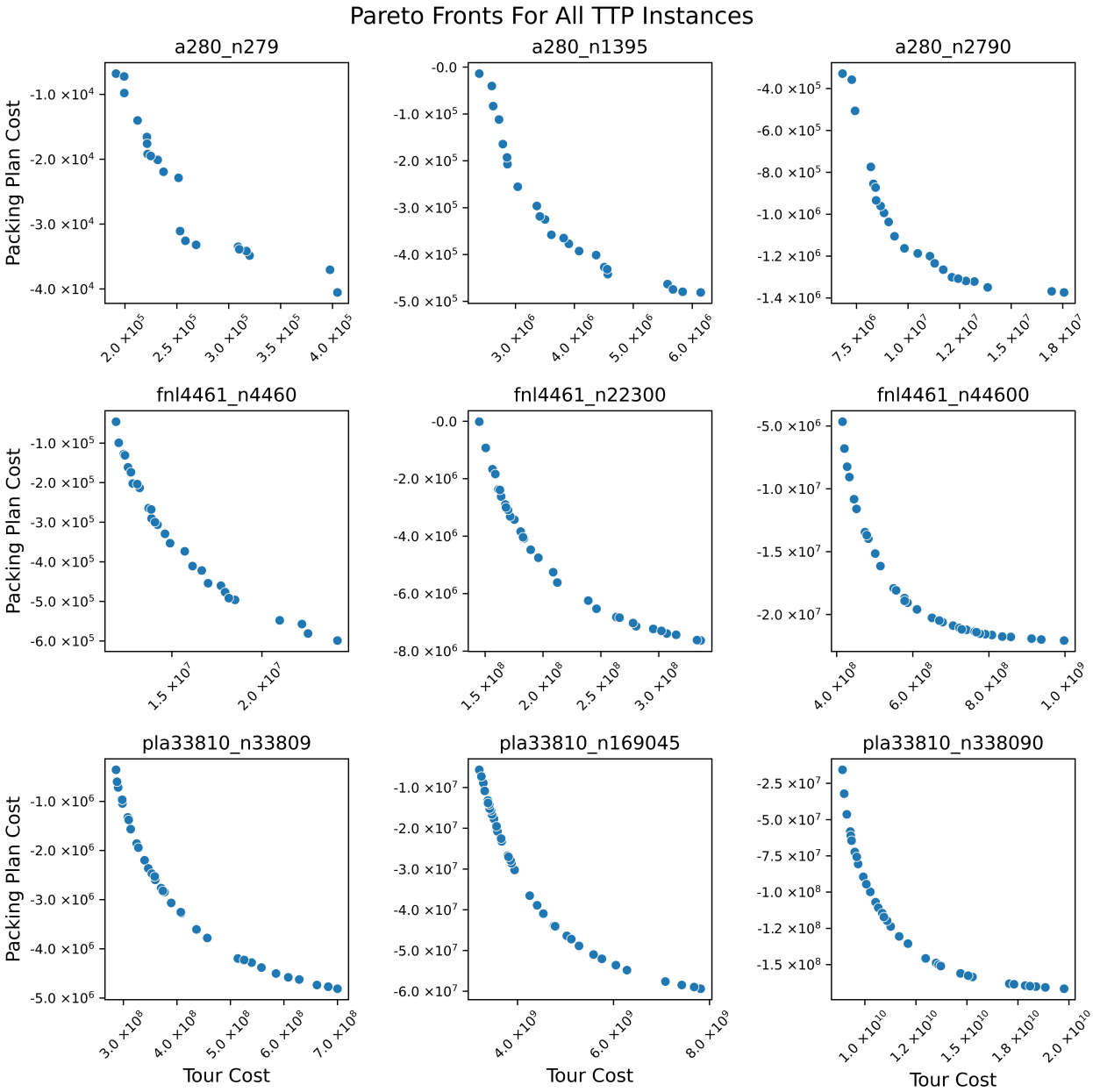

Figure 2: Pareto Fronts for all TTP instances

However, even though the Pareto fronts may look "nice", with a relatively diverse set of solutions, the obtained solutions are of poor quality regarding the tour costs and the packing plan costs. Indeed, for most of the TTP instances, the final solutions have a tour cost which is greater than the value accumulated in the knapsack of an order of 10. This means that the thief has ended the tour in debt for all the instances. These results may be due to the number of generations which was chosen to be the stopping criterion for the algorithm. It was decided that the algorithm would terminate after 100 generations due to the time complexity of the algorithm, especially on the larger-sized instances where running the algorithm could take multiple hours (40 hours to run 20 generations on the **pla33810-n338090** instance). Furthermore, the previous attempts in the literature to solve the TTP using evolutionary algorithm derived methods implemented heuristic functions to initialise "optimal" tour plans such as the nearest neighbour heuristic [6], the startnode search heuristic function [6] and the 2-opt optimal operator [6, 7]. Through this project, attempts were made to implement such heuristic functions however it was quickly determined that using such methods, with the computational power available to us, was not a realistic endeavour, especially on the large TTP instances. However, implementing such operators may have led to improved solutions.

References

[1] J. B. Chagas and M. Wagner, "A weighted-sum method for solving the bi-objective traveling thief problem," *Computers & Operations Research*, vol. 138, p. 105560, 2022.

[2] M. Wagner, "Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem," in *Swarm Intelligence: 10th International Conference, ANTS 2016, Brussels, Belgium, September 7-9, 2016, Proceedings 10*, pp. 273–281, Springer, 2016.

[3] H. Ali, M. Z. Rafique, M. S. Sarfraz, M. S. A. Malik, M. A. Alqahtani, and J. S. Alqurni, "A novel approach for solving travelling thief problem using enhanced simulated annealing," *PeerJ Computer Science*, vol. 7, p. e377, 2021.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[5] R. Kumari and K. Srivastava, "Nsgaii for travelling thief problem with dropping rate," in *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1647–1653, 2023.

[6] C. Wachter, *Solving the travelling thief problem with an evolutionary algorithm*. PhD thesis, Wien, 2015.

[7] J. Blank, K. Deb, and S. Mostaghim, "Solving the bi-objective traveling thief problem with multi-objective evolutionary algorithms," in *Evolutionary Multi-Criterion Optimization: 9th International Conference, EMO 2017, Münster, Germany, March 19-22, 2017, Proceedings 9*, pp. 46–60, Springer, 2017.