Group Submission ECMM409, Group B

Short Introduction and Problem Statement

The Traveling Thief Problem (TTP) combines the well-known travelling salesman problem with the knapsack problem. It aims to mirror real world problems which often focus on the interaction of several NP-hard problems. In this project an evolutionary algorithm was implemented according to the rules of a TTP competition from GECCO19 to generate solutions for a set of testcases [14]. This report first reviews and evaluates different approaches to solve this problem, secondly focuses on the specific implementation of our EA algorithm and the teamwork process and finally presents the results obtained.

Algorithm Selection

We researched a wide range of potential algorithms, all of which have been used to solve multi-objective problems and specifically the Travelling Thief Problem. In the following we present the most promising approaches.

Simulated Annealing [6] is a probabilistic optimization approach inspired by the annealing process in metallurgy. A temperature parameter is used that controls the probability of accepting worse solutions during the search. In the beginning, the focus is on exploration, and the further the temperature cools down, the more the focus shifts to exploitation. The objective is to find an optimal or near-optimal solution to a problem by exploring the solution space in a way that allows for escaping local optima. The approach has been successfully used for solving multi-objective problems in general [7]. It has also been used to effectively and efficiently optimize the TTP [8, 9]. The advantages of this approach are its global search capability and its ability to escape local optima due to the probabilistic acceptance criterion. The main disadvantage is its parameter sensitivity. The right tuning of parameters like initial temperature, cooling schedule, and acceptance probability function is crucial for a fast and efficient convergence.

Ant Colony Optimisation belongs to the category of swarm intelligence approaches and is inspired by the real-life foraging behaviour of ants. As the ants move across a construction graph, they deposit a pheromone trail which gets reinforced as other ants detect the existing pheromones and move across the same path [12]. Ant colony optimisation has been used to solve the TSP and also the TTP [11]. The reseachers who have used this approach, however, differ in which part of the TTP problem they use ACO for. Some just use ACO to generate initial tours and to then use a packing heuristic and local search to further improve the solution [11]. Others use ACO in every iteration to generate a route and then use a heuristic to determine a corresponding packing. Each ant is then evaluated on the resulting combined fitness value [10]. While this nature inspired approach can easily achieve close to optimal tours, it is harder to adapt to solve the TTP. Often just packing heuristics are used. This would mean it is only partially a nature inspired approach. Furthermore, the algorithm is very sensitive to parameter settings.

NSGA-II [1] (Non-dominated Sorting Genetic Algorithm II) is a popular and widely used algorithm for multi-objective approaches. It is an evolutionary algorithm that uses non-dominated sorting to classify individuals into different fronts or layers based on their dominance relationships. To rank individuals inside one front, crowding distance is used. This is a measure for how crowded an individual's surrounding space is. This helps in preserving solutions that contribute to maintaining a well-distributed set of Pareto-optimal solutions. Furthermore, it is an elitist algorithm, which means that best individuals from the current population are selected to be included in the next generation. It has been shown [2] that elitist algorithms work especially well for many-objective optimization, whereas steady state algorithms work well for multi-objective optimization. NSGA-II has been used for a wide range of multi-objective combinatorial optimization problems [3]. It was also successfully used to optimize the TTP, outperforming other algorithms [4, 5]. A noteworthy approach featuring elements from NSGA-II is the Non-Dominated Sorting Based Customized Random-Key Genetic Algorithm [13]. This approach uses the biased random key encoding to deal with the mixed-variable nature of the problem and achieved outstanding results.

Based on the conducted research and after weighing the advantages and disadvantages of the different approaches we finally decided to implement an evolutionary algorithm. This is because it comes with numerous advantages: The implementation of an evolutionary algorithm is easily divisible into tasks, and it can be applied to a wide range of potential problems which helps us to gain invaluable experience for the future. Moreover, it

is very adaptable, and a wide range of different operators, heuristics and improvements can be explored. Furthermore, the approach has been successfully used by other teams to achieve very good results even for larger datasets [13].

General Team Structure

To organize this project, we held regular meetings to discuss the progress and distribute new tasks. Furthermore, we communicated urgent updates via WhatsApp and posted the shared documents and bigger updates in a Teams chat. We usually distributed tasks to every team member and set a deadline to finish them until the next meeting where we would talk about the work done, discuss feedback, clear up any questions, and assign new tasks. We made it a priority to try and make sure we could all work on the code at the same time. To that end we defined every function to be written as a kind of black box with given inputs and outputs. These shared variables and outputs were clearly defined for every task. This way the separate functions just has to be put together in a main function and it is easy to include new versions of the same functions and even work on them at the same time. We had four main sprints which will be discussed in the following:

- Sprint 1: In this round everyone focused on research and on exploring a suitable approach to solve the TTP.
- Sprint 2: After finally deciding on the implementation of an EA we distributed the programming tasks to everyone in the group as follows: Data Loading (Yinghui), Initializing Populations (Zhen), Fitness Function (Ashutosh), Multi-objective Functions (Nils), Tour and Packing Crossovers (Binnan), Solving Overweight Packings (Yinghui), Setting up the Main Function, Testing and Integrating the Code (Richard).
- Sprint 3: Further improvements to the individual functions were done and all Python lists were converted to numpy arrays
- Sprint 4: The code was put together and all the bugs were ironed out. Further, the efficiency and the quality of the final Pareto set was improved (Richard, Nils)
- Sprint 5: The parameter search was conducted (Yinghui, Binnan, Zhen, Ashutosh), the final runs were carried out and the output was formatted (Nils). Finally, everything was put together in the final report (Richard, Nils)

Implementation

For this task a steady-state evolutionary algorithm with tournament selection was developed. First the population is initialized, either randomly or according to heuristics. Secondly, for every iteration two individual are selected via tournament selection, from which two children are generated via a crossover operator. Afterwards, they are the mutated with a mutation probability. Finally, the two new individuals replace the worst individuals in the population if that would constitute an improvement. In the following section some of the finer details of the implementation will be explored.
The implementation is based on numpy arrays, and a strong focus was placed on the efficient implementation of functions. This is also why the most important functions in the code, such as the initialization and the fitness function underwent multiple iterations. To achieve this, loops over arrays are avoided wherever possible and numpy specific functions were used instead.

Initialization: The population of the evolutionary algorithm consists of n routes and n corresponding packings. They are stored in the separate arrays named routes and packings. The routes are encoded as a permutation vector starting with city 1. The permutation corresponds to the order in which the cities are visited. The packings are encoded as binary vectors with one bit for every item. For example, the vector [0 1 0 1 1] translates to items 2, 4, and 5 being picked up. For the fitness evaluations another array is required named assigned_city, which stores the city every item can be picked up at. The initialization in the code includes multiple options. The first is a random initialization. For this, a random route is chosen starting from city 1. The packings are initialized as a random bitstring of ones and zeros. Alternatively, the routes can be initialized using the nearest neighbour heuristic and the packings can be initialized based on a ranking where the value of each item is determined by its profit divided by weight, and the distance left to the end of the route. To further improve the routes a local search can be used. For this, a 2-opt search [15] was used which was efficiently implemented by looking only at the local improvement as a result of the inverted sequence instead of evaluating the whole length of the route.

Fix Overweight: At multiple points in our code a packing could emerge which would violate the capacity constraints of the knapsack. To fix these packings, a function was written to remove items according to their value-to-weight ratio until the weight is admissible again.

Fitness Function: The fitness function calculates the total profit of the stolen items and the time taken to complete the route. It returns these two values separately and does not weigh one against each other. It is important to know that optimizing both of these values separately happens because of the replacement function and the fitness function just provides an objective evaluation of both of our optimization objectives.

Crossovers: Three crossovers were implemented for the routes and three separate crossovers for the packings. For the routes a simple ordered crossover (OX1), a cyclic crossover (CX) and a partially mapped crossover (PMX) were implemented. Regarding the packings, a single point crossover, a two-point crossover, and a uniform crossover were implemented. The effects of the different combinations of these crossovers will be further discussed in the experiments section of this report.

Mutations: The mutations were added to the project later on because the local search that was implemented before this did not provide the expected results in terms of efficiency or quality. Because there were already numerous other parameters to optimize the decision was made to focus on just one mutation for the routes and one for the packings. For the routes a reverse sequence mutation was chosen and for the packings a bitflip operation was chosen.

Replacement: The replacement function is an integral part of the algorithm, and it balances the optimization of the two objectives. In every iteration the worst individual to be replaced is chosen by this function. It returns the individual from the last front in the current population with smallest crowding distance. The crowding distance in this case is the Euclidean distance to the two neighbouring points in the same front. The two points at the edge are assigned a distance of infinity and are preserved this way. This approach forces the points to spread out and generates a wide range of solutions including edge cases. It is what enabled the algorithm to find a high-quality Pareto front.

Experiments

We performed an extensive parameter search for our algorithm. All resulting hypervolumes were averaged over five different seeds. The parameters can be grouped into the following categories: (a) Initialization Parameters, (b) Population and Tournament Size, c) Crossovers d) Mutation Rate. Since the data sets feature three types of knapsacks (bounded strongly correlated, uncorrelated similar weights, and uncorrelated), we performed three different parameter searches to potentially detect differences for the knapsack types. However, as the following subsections will reveal, no significant difference for the best parameters could be detected for different knapsack types.

Initialization Parameters: All the parameters tested in this category relate to local heuristics for the initialization of the population. Therefore, an increase in local heuristics usually leads to a better outcome. It does, however, impact the runtime of the algorithm, especially for the larger data sets. We therefore tried to find a middle ground between good results and a reasonable runtime.

First, the effect of the percentage of routes initialized by nearest neighbour initialization (percentage_NN) was tested. We found out that the positive effect the heuristic has on the hypervolume tapers off after percentage_NN = 0.4, which is why we chose this value. The same was true for the packing heuristic. The hypervolume does not significantly improve for a packing heuristic for more than 20% of the population. That the positive effect does not steadily increase the more individuals are created with a heuristic is likely due to the fact that a fraction of the population with very good objective values is enough to create offspring that is Pareto optimal.

Population and Tournament Size: Because we performed the parameter search on the first three data sets that require 100 Pareto optimal points, our parameter search returned the best results for a population size of 300 and a tournament size of 20. Since it is quite computationally expensive to maintain such a high population size for the larger data sets that do not require that many Pareto optimal points, we reduced the population and tournament size accordingly. Thus, for the medium-sized data sets we used a population of 150 and a tournament size of 10. For the three largest dataset we used a population size of 60 and a tournament size of 5. In multi-objective problems a focus on exploration is necessary to find a good Pareto front. Both a rather high population size and a low tournament size favour exploration over exploitation and enable different families of solutions.

Crossovers We tested combinations of three types of TSP crossovers, OX (Ordered Crossover), CX (Cyclic Crossover), PMX (Partially Mapped Crossover), and three types of Knapsack crossovers, SPC (Single Point Crossover), DPC (Double Point Crossover), and UC (Uniform Crossover). Interestingly, there was no significant difference in hypervolume for the knapsack crossovers across the three knapsack types. Combinations of OX and SPC, and PMX and SPC performed best. Since the Pareto set was significantly smaller for OX, we chose PMX and SPC as our crossovers.

Mutation Rate Our experiments showed that a mutation rate of 0.4 works best, which is a good middle ground between exploration and exploitation. A mutation rate that is too high would probably introduce too much randomness and couldn't benefit from the existing knowledge base.

Results

We conducted our actual runs with the final parameter choice stated as above. After running our algorithm on file1 for 10,000 iterations, our convergence plot, which can be seen in Figure 1 on the left, revealed that the algorithm converged after a few thousand iterations, so we set the maximum number of iterations to 5,000. For the three largest data sets, we reduced the number of iterations to 1,000.
Figure 2 shows our final Pareto fronts for data set 1, 5, and 9 respectively. It can be seen that the points on fronts are well distributed and reach good objective values for the time and the profit.
The efficiency of our algorithm should also be highlighted. 5,000 iterations including a heuristic initialization can be computed in less than a minute for the smallest three data sets. Even for the largest data set with 33,810 cities and 338,090 items, our algorithm can perform 1,000 iterations in less than 20 minutes.

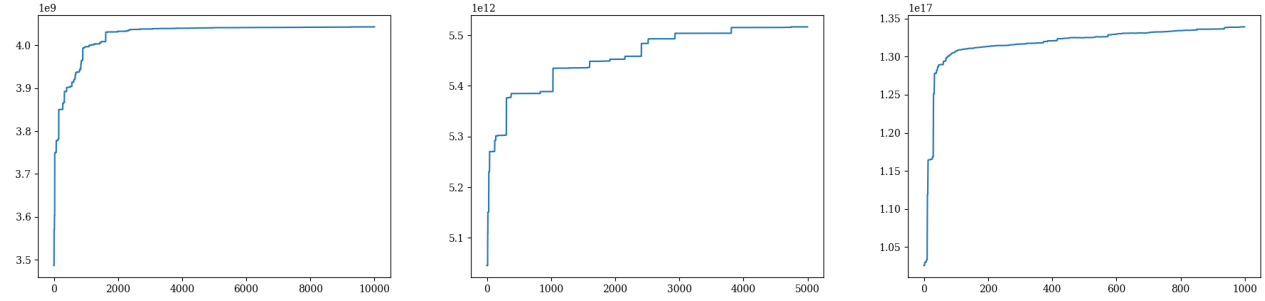

Figure 1: Convergence Plots with Hypervolume as Indicator for Data Set 1 (left), Data Set 4 (middle) and Data Set 7 (right)

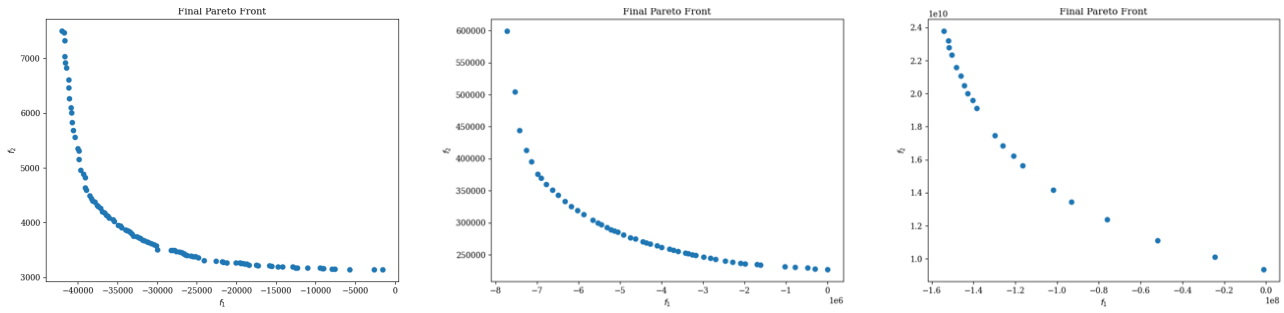

Figure 2: Pareto Front for Data Set 1 (left), Data Set 5 (middle), and Data Set 9 (right)

Conclusion

We would like to conclude this report by stating that our algorithm successfully optimizes the given problems in a very reasonable computation time. Working on this task as a group and being able to compare ourselves to other teams was very rewarding. In the future it will be interesting to see further algorithmic progress for this problem and the optimization of the Travelling Thief Problem can be expected to help solve real-world problems in fields like logistics, supply chain optimisation or data packet routing in networks.

Literature

[1] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

[2] ShiZhen and C. T. ZhouYang (2004). "Comparison of steady state and elitist selection genetic algorithms," 2004 International Conference on Intelligent Mechatronics and Automation, 2004. Proceedings., Chengdu, China, 2004, pp. 495-499, doi: 10.1109/ICIMA.2004.1384245.

[3] S. Verma, M. Pant and V. Snasel (2021). "A Comprehensive Review on NSGA-II for Multi-Objective Combinatorial Optimization Problems," in IEEE Access, vol. 9, pp. 57757-57791, 2021, doi: 10.1109/ACCESS.2021.3070634.

[4] J. Blank, K. Deb, S. Mostaghim (2017). Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms. In: Trautmann, H., et al. Evolutionary Multi-Criterion Optimization. EMO 2017. Lecture Notes in Computer Science(), vol 10173. Springer, Cham. https://doi.org/10.1007/978-3-319-54157-0_4

[5] K. Erdoğdu (2022). Comparing ALNS and NSGA-II on Bi-Objective Travelling Thief Problem. In: Proceedings of the International Conference on Industrial Engineering and Operations Management Istanbul, Turkey, March 7-10, 2022

[6] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi (1983). Optimization by Simulated Annealing. Science, New Series, Vol. 220, No. 4598. (May 13, 1983), pp. 671-680.

[7] S. Bandyopadhyay, S. Saha, U. Maulik and K. Deb (2008). "A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA," in IEEE Transactions on Evolutionary Computation, vol. 12, no. 3, pp. 269-283, June 2008, doi: 10.1109/TEVC.2007.900837.

[8] H. Ali, M. Zaid Rafique, M. Shahzad Sarfraz, MSA Malik, MA Alqahtani, JS. Alqurni. A novel approach for solving travelling thief problem using enhanced simulated annealing. PeerJ Comput Sci. 2021 Mar 16;7:e377. doi: 10.7717/peerj-cs.377. PMID: 33834093; PMCID: PMC8022508.

[9] Mohamed El Yafrani, Belaïd Ahiod. Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing. Information Sciences, Volume 432, 2018, Pages 231-244, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2017.12.011.

[10] M. Wagner (2016). Stealing things more efficiently with ants: A swarm intelligence approach to the travelling thieves problem. 10th International conference on swarm intelligence. Pages 273-281.

[11] M. Yafrani and B. Ahiod (2015). Cosolver2B: An efficient local search heuristic for the Travelling Thief Problem, 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), Marrakech, Morocco, 2015, pp. 1-5.

[12] M. Dorigo, M. Birattari and T. Stutzle (2006). Ant colony optimization. in IEEE Computational Intelligence Magazine, vol. 1, no. 4, pp. 28-39.

[13] J.Chagas, J. Blank, M. Wagner, M. Souza, K. Deb. (2020): A Non-Dominated Sorting Based Customized Random-Key Genetic Algorithm for the Bi-Objective Traveling Thief Problem.

[14] S. Polyakovskiy, M. Bonyadi, et al. (2014). A comprehensive benchmark set and heuristics for the traveling thief problem. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14). Association for Computing Machinery, New York, NY, USA, pp. 477–484.

[15] Karagül, Kenan & Aydemir, Erdal & Tokat, Sezai. (2016). Using 2-Opt based evolution strategy for travelling salesman problem. An International Journal of Optimization and Control: Theories & Applications (IJOCTA). 6.