Università degli studi di
Milano-Bicocca

Decision Models

Final Project

A Metaheuristic Approach to the Traveling Santa Problem

Authors:
Khaled Hechmi – 793085 – k.hechmi@campus.unimib.it
Gian Carlo Milanese – 848629 – g.milanese1@campus.unimib.it

July 17, 2019

Abstract

A variant of the Traveling Salesmen problem is studied with a purely metaheuristic approach. In this variant, which was proposed as a Kaggle Challenge in 2018, every 10th step in a route through all the cities is 10% more lengthy unless originating from a city with a prime CityId. The main algorithms chosen to find a solution are implementations of the Genetic Algorithm and a discrete version of Particle Swarm Optimization. After performing a clustering of the 197769 cities of the dataset into 1000 subsets, both algorithms were run in order to find an optimal route inside each cluster and an optimal ordering of clusters. Finally, the candidate routes found with each algorithm were the starting point for an application of the Simulated Annealing technique with a special mutation function that takes the penalties for non-prime CityId's into account.

1 Introduction

The *Traveling Salesman Problem* (TSP) is a fundamental problem in combinatorial optimization, with a wide range of practical applications. The standard formulation of this problem is the task of finding the shortest path that travels through every city from a given list, starting and ending at the same point and visiting each city only once.

The aim of this study is to investigate a variation of the TSP proposed as a *Kaggle* challenge, namely the *Traveling Santa 2018 - Prime Paths* challenge [1]. The difference from the standard TSP lies in the fact that every 10th step of a route is 10% more lengthy unless originating from a city whose *CityId* is a *prime number*, where the length of a step is computed as the *Euclidean distance* between the two cities. In the rest of the report the distance function that computes the total length of a route taking into account the discussed penalties will be referred to as *EDP*, short for *Euclidean Distance with Penalities*.

The code written for this project can be found at `https://github.com/hechmik/metaheuristics`.

2 Datasets

The dataset can be found on the dedicated web page for the Kaggle challenge [1] and consists of 197769 rows, each specifying the *CityId* of a city and its coordinates: Table 1 shows the first five rows of the dataset, while Figure 1 shows its distribution.

Table 1: The head of the cities dataset

CityId	X	Y
0	316.836739061509	2202.34070733524
1	4377.40597216624	336.602082171235
2	3454.15819771172	2820.05301124811
3	4688.09929763477	2935.89805580997
4	1010.69695174829	3236.75098902635

3 The Methodological Approach

The TSP and its variant considered in this study are prime examples of problems with a very simple formulation, but for which finding the optimal solution is not feasible. Indeed, the simplest strategy for solving this problem would be to generate every possible path in order to find which is the one that minimizes the distance function. The issue with this approach is that it requires to evaluate $(n-1)!$ paths, where n is the number of cities in the dataset (the city with *CityID* 0 should not be considered because of the problem constraints): if the dataset has more than very few rows this approach is not sustainable.

Therefore, a metaheuristic approach was adopted in the context of this project. In particular, the following main algorithms were chosen in order to find a solution:

- *Genetic Algorithm*
- *Particle Swarm Optimization*

The *Genetic Algorithm* (GA) is highly inspired by evolutionary traits found in biology. A population of candidate solutions – represented by permutations of a sequence of numbers – is initialized and is then evolved for a number of generations through the application of selection, crossover and mutation functions. At each generation every member of the population is scored according to a fitness function, which for TSP problems is usually the inverse of the tour length. The best elements from the population are selected as parents and a number of children is generated through a crossover function, which for the purposes of this project was a two-point crossover function (an example can be found in Table 2); the worst members of the population are then replaced by the children of the fittest parents. As it happens to biological individuals, a percentage of them may suffer from mutation: Table 3 shows a few examples of mutation functions. The mutation function chosen for this project is the *shift mutation* operation [2]. The main idea behind this algorithm is that the evolution and improvement of the solutions should be led by the fittest

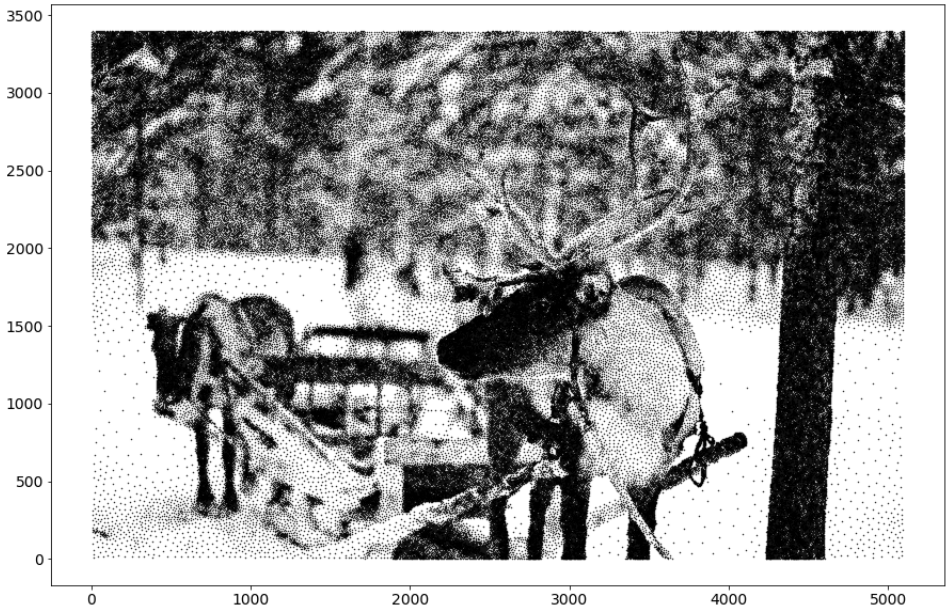

Figure 1: The distribution of the cities

individuals, even though slightly different routes may be explored thanks to the help of the mutation phase. Most of the functions used in this project for implementing GA are found in [3].

Table 2: Two-point crossover

Parent 1	1	3	2	5	4	7	6
Parent 2	7	6	5	4	3	2	1
Child	7	3	2	5	6	4	1

The *Particle Swarm Optimization* (PSO) is inspired by swarm movements. After the initial creation of particles and their velocities in a random way, at each step they are updated by taking into consideration the best solution ever found for each of them and the current best solution. The main idea is that the particles should move in the same direction of the best particles, because there is a higher probability of moving towards the optimal point, even though they move in slightly different ways in order to keep exploring new solutions. For this problem a particular PSO implementation [4] originally thought for the Flow Shop Scheduling problem was chosen: even though the context is different the algorithm is still suitable, since permutations have to be performed in order to find the best path. Its main characteristics are the following:

Table 3: Mutation functions

Original element	1	2	3	4	5	6	7
Reverse mutation	1	3	2	4	5	6	7
Shift mutation	5	6	7	3	4	2	1
Swap mutation	1	5	3	4	2	6	7

- The paths corresponding to a particle are generated with the *Smallest Position Value* (SPV) rule. This means that a path is computed for each particle as follows: the first city corresponds to the index of the lowest particle element, the second one corresponds to the index of the second lowest particle element and so on.
- The velocity decays at each iteration: this is done because at each iteration the algorithm is near to the local optimum, therefore it needs to move slower in order to not miss it.
- Swap mutation is applied to 10% of population particles randomly.

Along with this version, a second one was implemented that uses a hybrid approach [5]. Besides the classic PSO steps, it introduces an evolutionary element: at each iteration the particles and velocities of the worst half of the population are replaced with the ones of the best half of the population. In the rest of the paper this hybrid version of PSO will be referred to as HPSO.

The Genetic Algorithm and the two variants of Particle Swarm Optimization were run in parallel through the steps that will soon be described in order to compare their performance.

A third algorithm was finally chosen to be applied to the best solutions found by the previous algorithms to further improve the results, namely the *Simulated Annealing* (SA), a probabilistic technique for approximating the global optimum whose name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects [6]. The algorithm starts from an initial solution (which can be random, or given), which is mutated at each iteration in order to find a new solution. Depending on the fitness score of the new solution the algorithm can either keep the current solution, or move to the new one. In particular, the SA will always choose a new solution with a better score; on the other hand, if the new score is worse than the current one, the algorithm will move to the new solution with a probability proportional to the *temperature* parameter: the higher the temperature, the higher the probability of moving to a worse solution. As the iteration progresses

the temperature is lowered, so that the algorithm will eventually converge. The chance of temporarily moving to a worse solution makes it harder for the algorithm to immediately converge to a local optimum. A special mutation function that takes into consideration the penalties for cities with non-prime CityId's corresponding with tenth steps was defined for this phase.

The following subsections outline the steps undertaken in order to solve the problem, each of which was run using the main metaheuristic algorithms.

3.1 Clustering

Considering the large search space, a simple implementation of a metaheuristic algorithm (such as GA or PSO) run on the full set of cities cannot be expected to produce good results: indeed, the best solution achieved with such an approach showed a very small margin of improvement over a random permutation of cities.

To overcome this issue the dataset was partitioned in a number of clusters by using the implementation of the K-Means algorithm offered by the `scikit-learn` Python package. The dataset was thus partitioned into 1000 clusters, with the resulting largest cluster consisting of 407 cities, the smallest one of just 12, and a median value of 201. Figure 2 shows the partitioned dataset.

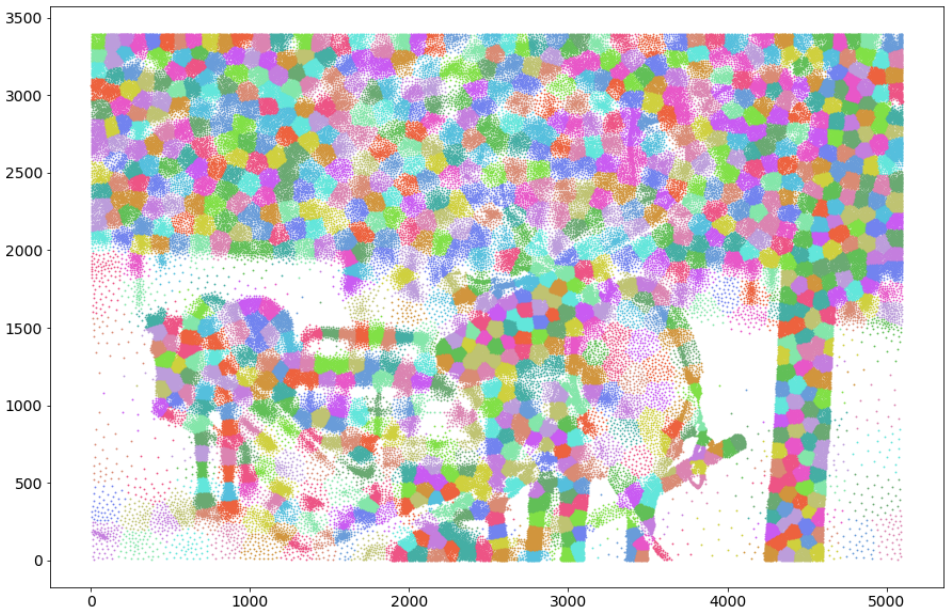

Figure 2: The cities partitioned in 1000 clusters

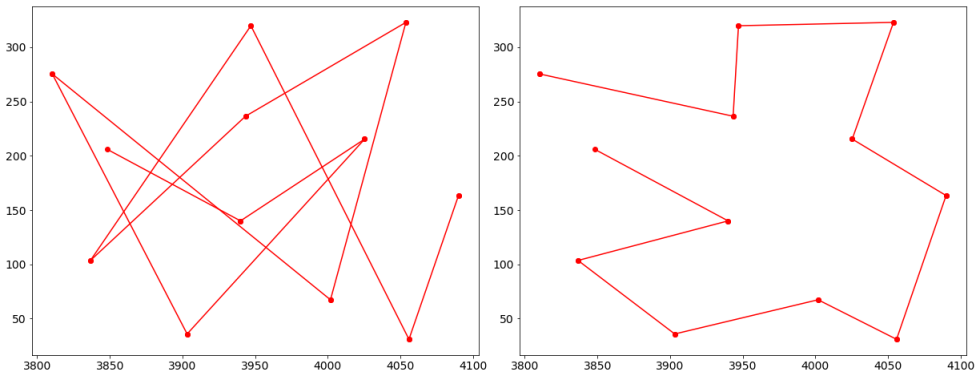

Figure 3: The path inside the smallest cluster before and after applying GA

3.2 Finding the optimal route inside each cluster

After identifying the clusters, both GA and PSO were run in order to find the best route among the cities inside each cluster. At this stage the problem of finding the best route inside each cluster is treated as a standard TSP problem, with the total length of a route inside a cluster being computed as the sum of the Euclidean distances between each step. Figure 3 shows the route inside the smallest cluster before and after running the GA inside it.

Concatenating all the routes in the order given by the labels of the clusters and rearranging the resulting route so that it starts and ends at 0 already showed a very significant improvement to the total EDP distance on the full set of cities.

3.3 Finding the optimal permutation of clusters

Each cluster could be considered as a province, or a region, made up of a number of cities in its territory, and it is intuitive to think that a good route through all the provinces can approximate a good route through all the cities. Indeed, another improvement to the route found in the previous steps was achieved by finding an optimal route through all the clusters. To this end, both GA and PSO were run in order to find a better permutation of the clusters. While an option would be to evaluate the fitness of a permutation of clusters by evaluating the route that goes through their centroids, in this case the fitness function to optimize was chosen as the length of the tour obtained by concatenating the routes inside each cluster found in the previous step. At this stage the total tour length was again computed by using the standard Euclidean distance.

After finding an optimal ordering of clusters, concatenating all the routes accordingly and rearranging it so that the route starts and ends at 0 showed another improvement to the total EDP on the full set of cities.

3.4 Fine-tuning the routes

In the previous steps the problem was treated as a regular TSP problem, without considering the penalization for 10th steps in the route originating from cities with a non-prime ID. This last subsection focuses on the approach adopted to improve the routes by taking this penalization into account. The improvement achieved with this final step is not expected to be very significant, as the penalization itself is marginal.

For this final stage the SA technique with a special mutation function was employed. In order to decrease the search space, the routes found in the previous step were partitioned into 1000 consecutive segments, each composed of 200 cities apart from the final one, and the SA was run for each of these segments. The size of the segments being a multiple of 10 was useful in order to keep track of cities corresponding to 10th steps in the route more easily. The mutation function implemented for this step performs the following steps:

1. given a route, find the cities corresponding to a 10th step such that their CityId is not prime, but the CityId of their successor is prime;
2. assign a high weight to each of these cities, and a lower weight to the remaining cities;
3. randomly choose a city from the route with probability proportional to its weight, and swap it with its successor.

4 Results and Evaluation

Table 4 compares the distances of the best routes found by the two main metaheuristic algorithms for each step described in the previous section. The EDP for a random route was computed by taking the average of the distances of 100 randomly generated routes, while that of the "clustering" step by concatenating the CityId's inside each cluster, sorted by ID, with the clusters sorted by their labels, and then ordering the resulting route so that it started and ended at 0. The "Clustering", "Finding Routes Inside Clusters", "Sorting Clusters" and "Fine-tuning" stages refer to sections 3.1, 3.2, 3.3 and 3.4, respectively. For the sake of clarity, the last row concerns the application of SA with the special mutation function to the best routes

found by the GA and HPSO algorithms, and does not actually involve running the GA or HPSO algorithms.

The execution times for the steps concerning the GA algorithm and the clustering are relative to an Intel i7 CPU @2.8 GHz, while those for the PSO algorithm to an Intel i5 CPU @2.6 GHz.

Table 4: Improvements on EDP at each stage

Stage	**Main Algorithm**	**EDP**	**Execution Time (Minutes)**
Random Route	\	447,106,962	\
No Clustering	GA	444,162,148	72
	HPSO	445,142,148	23
	PSO	445,191,308	5
Clustering	\	13,261,882	88
Finding Routes Inside Clusters	GA	7,878,150	109
	HPSO	9,726,464	43
	PSO	10,501,353	36
Sorting Clusters	GA	6,788,202	168
	HPSO	9,667,539	65
	PSO	10,456,245	35
Fine-tuning	GA	6,679,052	32
	HPSO	9,487,836	41
	PSO	\[1]	\

5 Discussion

It is evident from Table 4 that the most significant improvement was achieved by simply partitioning the cities dataset into 1000 clusters. However, it is evident from

[1]SA was not applied to the PSO route because the HPSO one had a better starting solution

Figure 1 that the patterns in the data are not classic geometrical objects (for example, the tree on the right and the reindeer muzzle). Most of the 1000 clusters that were found, displayed on Figure 2, are of the same dimension and shape and do not approximate the existing patterns: for this reason the use of more appropriates clustering algorithms is expected to lead towards even better results.

The use of the metaheuristic algorithms on each cluster instead of the entire dataset also led to a very significant improvement. There are nevertheless at least two improvements that could be implemented. The first one involves the distance and the mutation functions used inside each cluster: using EDP and a mutation function that takes into account the penalties for 10th steps originating from a city with a non-prime CityId at this earlier stage might result in an improvement on the final result. An approach of this kind might even be essential for problems where similar but more significant penalties are applied. Secondly, the best route in each cluster has been found without considering the cities in nearby clusters, which might lead to problems when concatenating the clusters. Therefore, a second improvement might result by dividing the cities into *overlapping subsets*, for instance with an approach similar to the one described in [7].

As was expected, fine-tuning the two best routes found by GA and HPSO by applying SA with the described mutation function only led to a marginal improvement. Better results might be obtained with different mutation functions, for instance a mutation function that not only considers the city that is next to one corresponding to a 10th step, but also reasonably close ones. On the other hand, a *swap mutation* function that operates by swapping non-prime cities on 10th steps with prime cities not on tenth-steps was implemented, but did not lead to better results.

Comparing the two algorithms, GA is the one that always led to the best results, even though PSO and HPSO had lower execution times. The reason why the two Particle Swarm Optimization implementations are faster is that the wait interval is lower (the algorithm would stop if for 100 consecutive iterations the solution didn't improve, while GA waited for 500 iterations) and has a finite number of iterations (1000 when the entire dataset is subject to PSO and HPSO, 800 when the two algorithms are working on the clusters, while GA has infinite iterations): this was done because using the same GA parameters would have led to considerable higher execution times. Further research might consist in implementing and comparing additional algorithms and techniques, such as *Ant Colony Optimization*.

Regarding Particle Swarm Optimization, HPSO led to an improvement both on the EDP value and on execution times when the two algorithms were evaluated on the entire dataset. It is important to state that the PSO version was improving when the algorithm stopped, therefore increasing the maximum number of iterations

will surely lead to a better path. In the cluster scenario, the hybrid approach led to a relevant improvement in the final solution: the solution improved by nearly 800 thousands units, even though the execution time is longer. Both algorithm converged before 800 iterations, therefore the most natural improvement would be to increase the wait interval.

6 Conclusions

To summarize, a metaheuristic approach was adopted to find a solution to a variant of the TSP problem in which 10th steps originating from non-prime cities are 10% more lengthy. The dataset consisting of 197769 cities was partitioned in 1000 clusters with a K-Means algorithm in order to reduce the search space for the two main metaheuristic algorithms chosen, the Genetic Algorithm and the two variants of the Particle Swarm Optimization. After the clustering, each algorithm was run inside each cluster in order to find an optimal sub-route through it, and later to find an optimal permutation of clusters. Finally, the Simulated Annealing technique was implemented and run on the best routes found in the previous steps with a special mutation operator that reverses with high probability two cities corresponding to a 10th step if the CityId of the first one is non-prime, while the CityId of the second one is prime.

The main conclusion of this project is that in order to get satisfactory results on large datasets with metaheuristics algorithms it is essential to focus on the initialization of starting solutions. In this case a relevant improvement was obtained thanks to the use of K-Means algorithm, because it exploits data patterns up to a certain point. The first improvement to our work would be the use of different clustering algorithms for better exploiting the available patterns. Because of the clear distinct elements that are in the picture, such as the trees, the reindeer and Santa's sled, the use of *Object Detection* algorithms may help: knowing the contours of the main figures will surely be useful for creating ad-hoc clusters.

One of the main limitations of our work was the time required by each algorithm in order to converge to a good solution: this is the main reason that led to use a small starting population. The operation that requires more time is the distance computation of the complete paths. This operation is executed in a subsequent way, therefore it is possible to parallelize it on other CPU cores or on GPU: it will be necessary to adapt the existing codebase in order to make it compatible with multi-thread computation. After this improvement it will be possible to increase the size of the starting populations and the wait interval before a result is returned without the need to use better hardware. Another possible parallelization is the evaluation of the

different algorithms: in this way it will be possible to evaluate different combinations of parameters in a fraction of the time that is currently required.

References

[1] (2018) Traveling Santa 2018 - Prime Paths. Kaggle. [Online]. Available: https://www.kaggle.com/c/traveling-santa-2018-prime-paths/overview

[2] A. Abbbb, A. Shariat, and M. Babaei, "An efficient crossover operator for traveling salesman problem," *International Journal of Optimization in Civil Engineering*, vol. 2, pp. 607–619, 2012. [Online]. Available: https://www.researchgate.net/publication/236026740_AN_EFFICIENT_CROSSOVER_OPERATOR_FOR_TRAVELING_SALESMAN_PROBLEM

[3] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 63, no. 3, pp. 337–370, 1996. [Online]. Available: https://doi.org/10.1007/BF02125403

[4] M. Tasgetiren, Y.-C. Liang, M. Şevkli, and G. Gencyilmaz, "Particle Swarm Optimization Algorithm for Makespan and Maximum Lateness Minimization in Permutation Flowshop Sequencing Problem," *Proceedings of the Fourth International Symposium on Intelligent Manufacturing Systems*, 2004. [Online]. Available: https://www.researchgate.net/publication/252625779_Particle_Swarm_Optimization_Algorithm_for_Makespan_and_Maximum_Lateness_Minimization_in_Permutation_Flowshop_Sequencing_Problem

[5] C. Ding, W. Peng, and W. Wang, "Hybrid metaheuristics and their implementations," *International Journal of Online and Biomedical Engineering*, 2015. [Online]. Available: https://online-journals.org/index.php/i-joe/article/view/4762/3574

[6] Wikipedia contributors, "Simulated annealing — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Simulated_annealing&oldid=901616990, 2019, [Online; accessed 7-July-2019].

[7] R. Bazylevych, B. Prasad, R. Kutelmakh, R. Dupas, and L. Bazylevych, "A Decomposition Algorithm for Uniform Traveling Salesman Problem," 2009, pp. 47–56.