(CS454) Assignment #2:
Hybrid Optimisation for TSP

Junghyun Lee* (20170500)

Dept. of Mathematical Sciences & School of Computing, KAIST

October 5, 2020

Contents

*jh_lee00@kaist.ac.kr

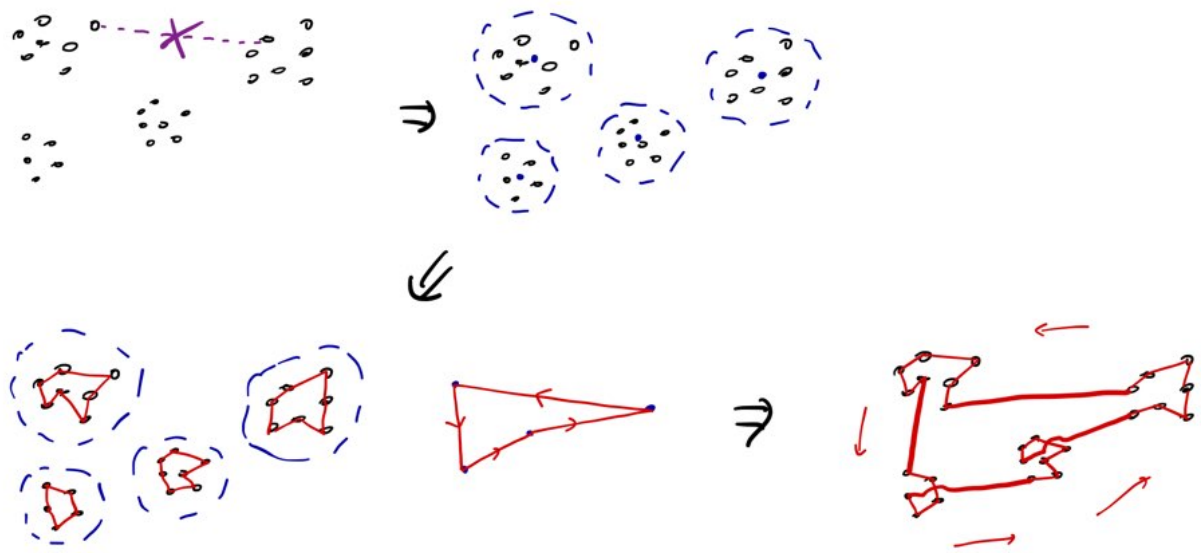


Figure 1: Rough picture of the algorithm

1 Background

This algorithm deals with the symmetric Euclidean travelling salesperson problem(TSP from here on forth) The precise mathematical formulation is as follows:

Input: A set of N coordinates $\{(x_i, y_i)\}_{i=0}^{N-1}$.

Question: Find the Hamiltonian cycle of K_N with the least weight, where K_N is a undirected complete graph on vertices $\{(x_i, y_i)\}_{i=0}^{N-1}$ whose edge weight is the usual Euclidean distance.

This problem is known to be NP-hard, and thus we need to resort to approximation algorithms.

Such algorithms can be divided into two types: stochastic and non-stochastic. There are a vast amount of literature on both types of algorithms (see [1, 2, 3] for an extensive survey of this) This implementation makes uses of both types of algorithms, as explained in the next section.

2 Algorithm

Figure 2 shows a very rough picture of this algorithm: This section provides a step-by-step description of the algorithm, along with suitable justifications/motivations of each step. Also, implementation details such as packages used are also provided.

2.1 k-Means Clustering

One of the most promising features of many global search meta-heuristics (such as GA, ACO, PSO...etc) is that they are applicable to almost any kind of optimization problems with no prior knowledge whatsoever: the objective function can even be blackbox. However, this is why not scalable; in other words, as the problem size grows, the trade-off between amount of required resources and quality of the solution worsens. This is paradoxically partly due to the characteristic of such algorithms that they do not exploit any specific structures of the problems. Thus I propose to use clustering as a "preprocessing" before applying the stochastic algorithms.

This allows for the algorithm to exploit the geometric feature of TSP: if points are clustered together, then they should be nearby in the optimal Hamiltonian cycle, and if points are far away, then they should be far apart. My algorithm uses k-means clustering, implemented by `sklearn.cluster.KMeans`. This creates local TSP problem instances, corresponding to each cluster.

2.2 Intracluster ACOs

Obviously, the next step is to solve each local TSPs. I've used ACO (ant colony optimization), which was implemented by myself in `TSP_ACO.py`. Reason for selecting ACO is that along with GA, it was shown empirically to outperform other stochastic metaheuristics[4]. Also, (although not implemented), it is easier and more intuitive to apply hyperparameter tuning for the ACO instance. For more details, see Section 4.2 Hyperparameters used are as follows: $T = 100, k = 10, a = 1, b = 1, \rho = 0.1$[1]. (Same formulation of the ACO as discussed in the class was used) This creates a partition of the complete graph(induced by the input coordinates) into n cycles, where n is the number of clusters. One obvious trick to accelerate this process is described:

2.2.1 Parallelizing these ACOs

Note that each cluster's TSP is independent of another, which means that this can be parallelized. However, in current implementation, this was not possible. For the detailed reason, refer to the Section 4.1.3.

2.3 Intercluster ACO

In order to combine the local solutions, produced by previous step, the algorithm solves a TSP instance, created by the medians of the clusters. Weaker setting for ACO was used: $T = 50, k = 5, a = 1, b = 1, \rho = 0.1$.

2.4 Combine the local solutions into a global solutions

Based on the order of the clusters as found previously, the algorithm combines all local solutions into a global solution in a greedy manner. Figure 2 and 3 illustrates how this is implemented.

2.5 (*) 3-Opt Algorithm

Many of the meta-heuristics available (including used ACO) may fall into some kind of a local minimum. To escape from this, the algorithm performs 3-Opt heuristic. Refer to Figure 4 for the algorithmic details. The locations (in the algorithm) of where the 3-Opt heuristic was performed are determined by the option `-topt`, of which is explained in Section 5.

Although not reported in this report, extensive experiments show that using 3-Opt in any ways improves the qualities of intermediate solutions produced (and thus improving the final solution's quality, overall) significantly.

Remark. *The Python implementation of the 3-Opt algorithm used is based upon the Wikipedia article*[2] *of the same name.*

2.6 (*) Number of clusters

As one could guess, the number of clusters probably has a huge effect on the quality of the solution. Indeed, as reported in Section 3.1, in the current implementation, the solution quality seems to have no discernible relationship; in other words, searching through all possible numbers of clusters is the best option that we have.

[1] T: maximum iterations, k: number of ants, a, b: exponents determining the dependency of the probability distribution on edge distance and pheromone deposit, ρ: pheromone evaporation ratio

[2] https://en.wikipedia.org/wiki/3-opt

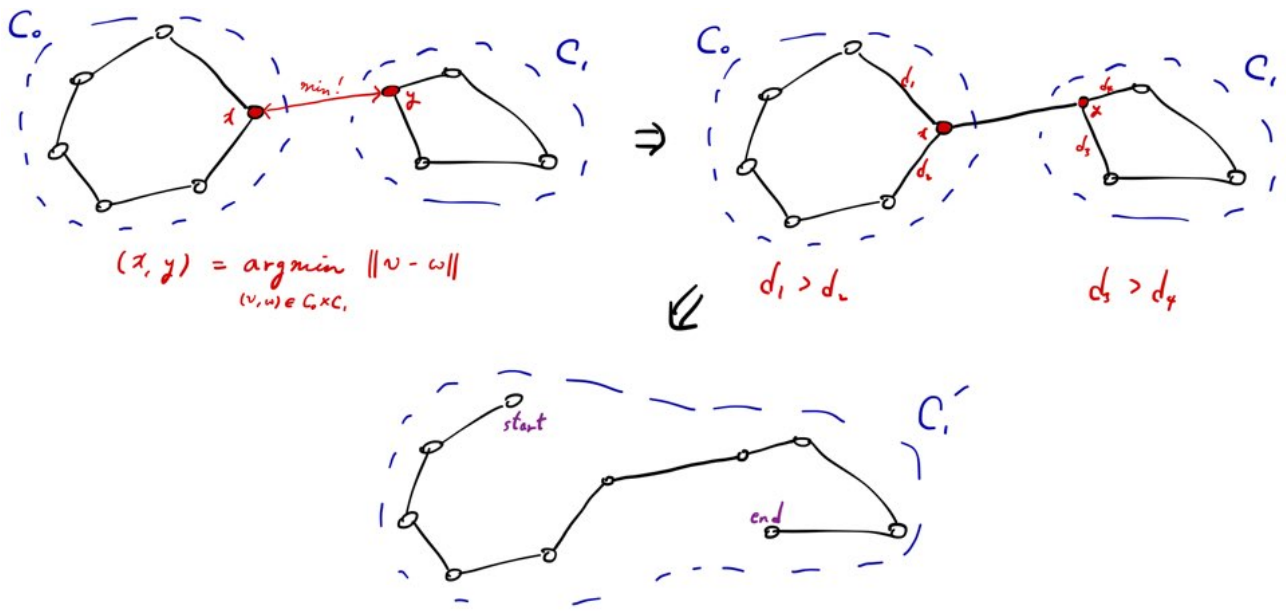


Figure 2: Initialization

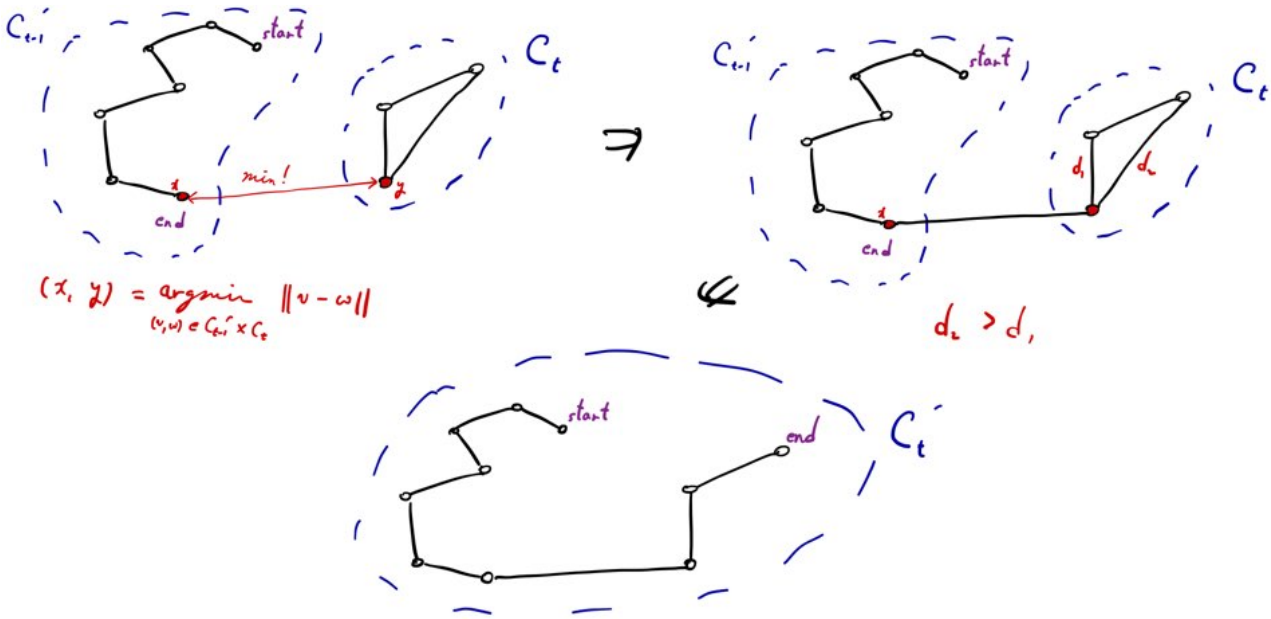


Figure 3: Rest of the iterations

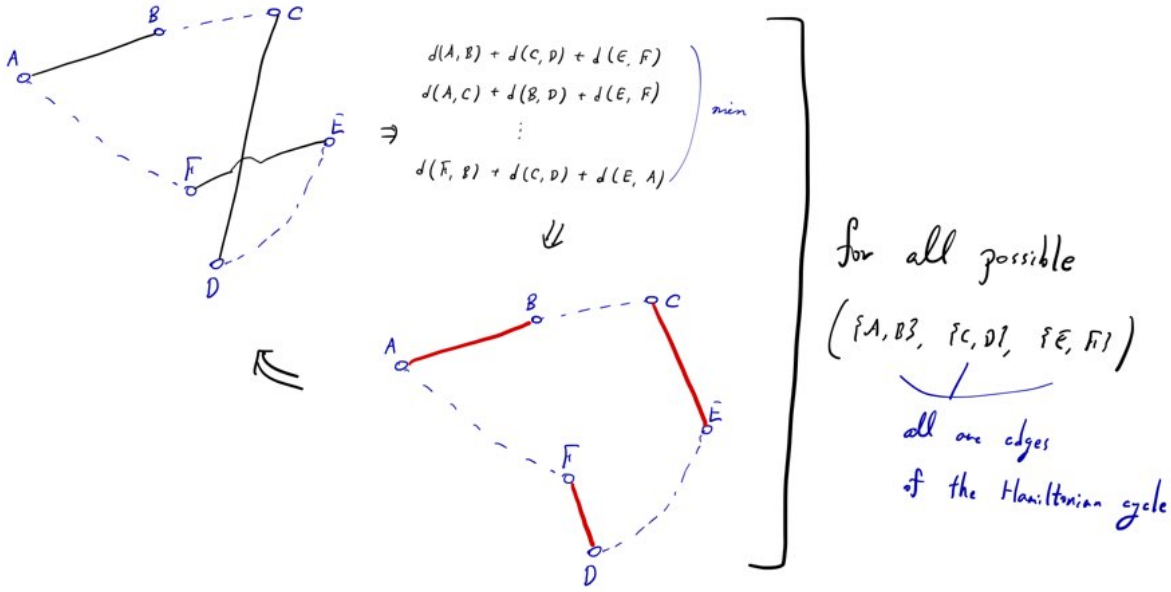


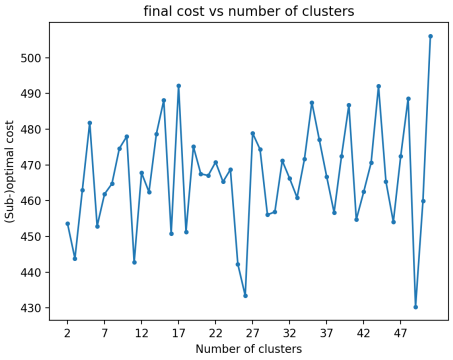
Figure 4: Overview of 3-Opt Heuristic

3 Experiments

To see the effectiveness of this algorithm, I've done several experiments. The computing environment used is my laptop, which is 2.3 GHz 8-Core Intel Core i9 with 32GB RAM. 11 TSP instances from TSPLIB[5] were used, all of which has known optimal solution.

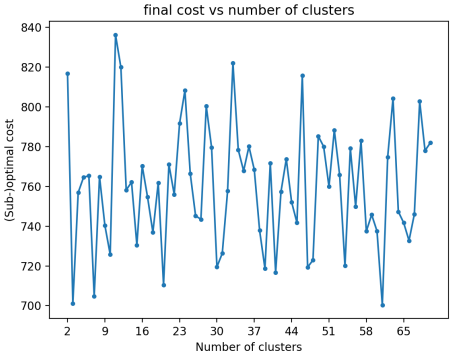
3.1 RQ1: Effect of number of clusters

Figure 6 and 5 shows the effect of number of clusters on the quality of the solution produced, for each dataset considered. Note that the overall behaviours are all random, and depending on the number of clusters, the quality of the solution changes greatly. This is one of the disadvantages of the current implementation.

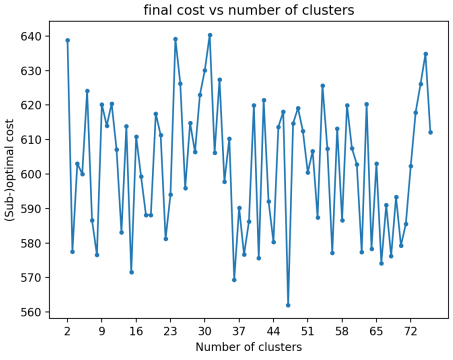


(a)

(b)

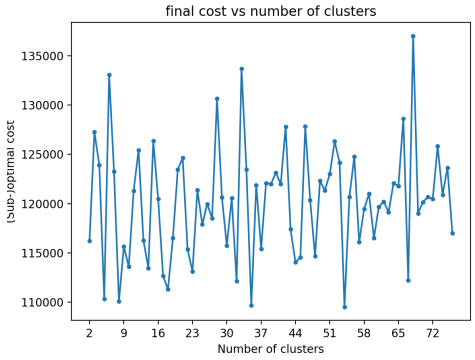


(c)

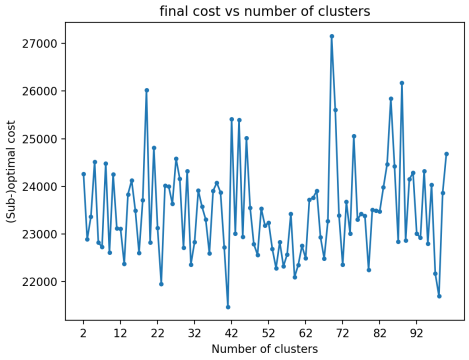


(d)

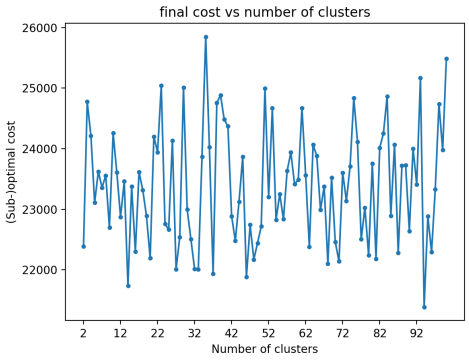
Figure 5: Datasets: (a) eil51.tsp (b) berlin52.tsp (c) st70.tsp (d) eil76.tsp



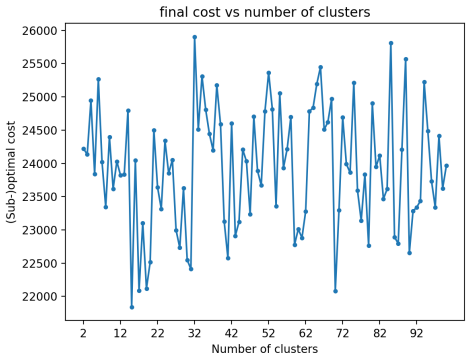
(a)



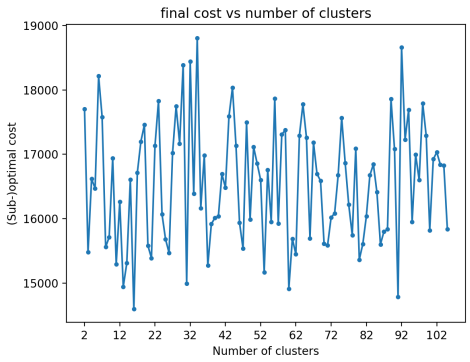
(b)



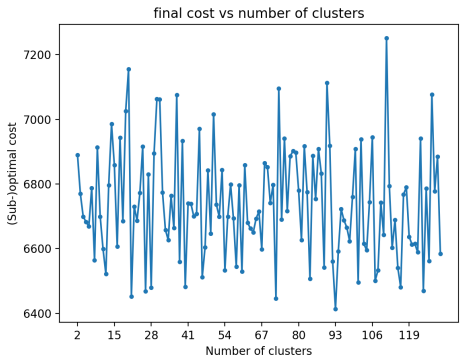
(c)



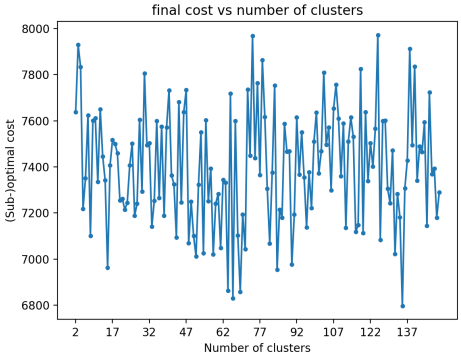
(d)



(e)



(f)



(g)

Figure 6: Datasets: (a) pr76.tsp (b) kroa100.tsp (c) kroc100.tsp (d) krod100.tsp (e) lin105.tsp (f) ch130.tsp (g) ch150.tsp

Table 1: All results (Datasets in increasing order of the instance size)

Dataset	Rel error(%)	Runtime(sec)	Optimal #clusters
eil51.tsp	0.073571	21.168900	49
berlin52.tsp	4.057245	23.480566	7
st70.tsp	3.203759	70.912842	61
eil76.tsp	3.050332	90.077797	47
pr76.tsp	1.256251	99.081978	54
kroa100.tsp	0.877605	292.646285	41
kroc100.tsp	3.058841	297.441060	94
krod100.tsp	2.567050	294.111822	15
lin105.tsp	1.493876	341.559732	16
ch130.tsp	4.958434	787.219632	93
ch150.tsp	4.056542	1421.423156	135

3.2 RQ2: Performance of this algorithm

Metric used to test the performance is the following:

$$\left(100 * \frac{resulting\_cost - optimal\_cost}{optimal\_cost}\right)\%$$

Table 1 shows the relative error(%) and running time for each datasets. Note how current implementation can achieve relative errors are within 5% of the given optimal solution. Considering how the ACOs used only had $T = 100$ iterations (taking parallelization into account), such result is pretty impressive. 3-Opt seems to have very strong positive effect on the quality of the produced solution

4 Future Works

Due to lack of time, I could not implement all the things that I've planned at the beginning. Let me write the basic details of each improvement that will be made in the future here. (Improvements will be done in next version, but only god knows when it comes out...)

4.1 Time Complexity

This is the biggest problem of the current implementation: it is too time inefficient. It cannot handle too large TSP instances(ex. `rl11849.tsp`), and as the above table shows, it may struggle for even (relatively) small instances. Here are some improvements that can be made:

4.1.1 Algorithm as a whole

Depending on the clustering, time complexity and the quality of solution may differ. Also, the greedy intermediate step may yield even worse result.

One direction of which this algorithm can be improved is to note that this algorithm does not make use of the triangle inequality, one of the most important properties of a metric space. Somehow changing the algorithm such that 1. it is less dependent on the clustering and 2. More suitable for metric TSP is, in my opinion, a promising future direction.

4.1.2 High Time Complexity of 3-Opt

Single application of 3-Opt takes $O(n^3)$ time, where n is the problem size. Especially when topt = 2(refer to Section 5 for this argument's definition), the overall complexity increases significantly. Somehow decreasing the number of times 3-Opt is used (or maybe switching to some other heuristic such as 2-Opt) will help with the time complexity. Also, note that current implementation utilizes CPUs only. [6, 7] suggests using GPU can significantly accelerate 3-Opt procedure.

4.1.3 Incomplete Parallelization

In my implementation, parallelization was done using `multiprocessing.Pool`. This package considers each `Pool` as a "daemonic" process, preventing the user from creating another `Pool` inside that `Pool`. In other words, although this algorithm can be parallelized in two ways at the same time, I was forced to choose only one way, which was finding the optimal number of clusters. By changing the package used, implementing both parallelizations will greatly help with the time complexity.

4.1.4 Better Computing Environment

Current computing environment is a bit weak in the sense that only 8 cores are available. Obtaining a cutting-edge computing environment and running the experiments will certainly effect the running time.

4.2 Hyperparameter Tuning

The hyperparameters for the current implementation were chosen without any tuning (partly because I didn't have enough time/resourcse) There have been many works on finding an optimal set of hyperparameters for ACO[8, 9, 10]. It has been shown that the optimal set of hyperparameters for ACO is very problem-specific, which motivates the need for an automated hyperparameter searching framework. Particularly inspired by [10], I suggest using PSO for tuning the hyperparameters of ACO, as a meta-optimization. It is not clear how GA can be applied to this, but by considering the hyperparameters of the ACO as a vector in some space, it is very clear how PSO can be applied. Also, since each local TSP is different, I expect that such change can make great improvements in the quality of solutions produced.

4.3 Better Experiment Settings

This relates to all the above points. After finalizing the algorithm, a much more intricate experimental setting will be used to perform new experiments:

1. Because I thought that the algorithm was yet premature, I've measured the performance of this algorithm only once for each dataset to first see the potential in this methodology. But since the algorithm itself is very stochastic, it is necessary to run the experiment k times ($k = 10$ following [10]) and report the mean/standard deviation.

2. By finalizing the algorithm, I'm hoping to see a significant improvement in the running time.

4.4 Better report/documentation

It would be good if the diagrams are a bit more polished, and the report itself is a bit more organized...

5 Usage

(The latest version is released in the Github repository[3]) **Make sure to check the `requirements.txt` for the required packages! (Those are standard packages: numpy, matplotlib, scikit-learn)** This section provides some details on each options:

5.1 filename (Required)

Name of the .tsp file to solve

5.2 sol_filename (Optional)

Name of the .opt.tour file (optimal solution) for the solver to compare its resulting solution to.

5.3 p [P] (Optional)

Number of ants for ACOs of intracluster TSPs. Default is 10.

5.4 -topt [TOPT] (Optional)

1. topt = 0: 3-Opt is not used at all
2. topt = 1(Default): 3-Opt is used only at the very end (i.e. 3-Opt is not used during the parallelization)
3. topt = 2: 3-Opt is used everywhere (almost as a filter)

5.5 -cratio [CRATIO] (Optional)

Specify the maximum ratio of the problem size that the searching for optimal number of clusters is done. Default is 1.

5.6 -plot (Optional)

Show plot of number of clusters vs cost of the solution

5.7 -v (Optional)

Verbose

5.8 -par (Optional)

Enable parallelization.

5.9 -cpus [CPUS] (Optional)

Specify number of cpus to be used for the parallelization. Default is `os.cpu_count()`.

[3] https://github.com/nick-jhlee/CS454-TSP-Solver

6 References

[1] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231 – 247, 1992.

[2] Y. Marinakis, *Heuristic and metaheuristic algorithms for the traveling salesman problemHeuristic and Metaheuristic Algorithms for the Traveling Salesman Problem*, pp. 1498–1506. Boston, MA: Springer US, 2009.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. USA: Princeton University Press, 2007.

[4] K. Chaudhari and A. Thakkar, "Travelling salesman problem: An empirical comparison between aco, pso, abc, fa and ga," in *Emerging Research in Computing, Information, Communication and Applications* (N. R. Shetty, L. M. Patnaik, H. C. Nagaraj, P. N. Hamsavath, and N. Nalini, eds.), (Singapore), pp. 397–405, Springer Singapore, 2019.

[5] G. Reinelt, "Tsplib—a traveling salesman problem library," *INFORMS Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[6] K. Rocki and R. Suda, "Accelerating 2-opt and 3-opt local search using gpu in the travelling salesman problem," in *2012 International Conference on High Performance Computing Simulation (HPCS)*, pp. 489–495, 2012.

[7] W.-B. Qiao and J.-C. Créput, "Massive 2-opt and 3-opt moves with high performance gpu local search to large-scale traveling salesman problem," in *Learning and Intelligent Optimization* (R. Battiti, M. Brunato, I. Kotsireas, and P. M. Pardalos, eds.), (Cham), pp. 82–97, Springer International Publishing, 2019.

[8] D. Gaertner and K. Clark, "On optimal parameters for ant colony optimization algorithms," in *Proceedings of the International Conference on Artificial Intelligence 2005*, pp. 83–89, CSREA Press, 2005.

[9] Z. Dan, H. Hongyan, and H. Yu, "The optimal selection of the parameters for the ant colony algorithm with small-perturbation," in *2010 International Conference on Computing, Control and Industrial Engineering*, vol. 2, pp. 16–19, 2010.

[10] M. Mahi, Ömer Kaan Baykan, and H. Kodaz, "A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem," *Applied Soft Computing*, vol. 30, pp. 484 – 490, 2015.