A Novel Clustering and Routing Algorithm for the Multiple Traveling Salesmen Problem: Sweep-Loop-Mutate

Jared McKlemurry and Youssef Harrath
Beacom College of Computer and Cyber Sciences
Dakota State University
Madison, SD, USA
email: youssef.harrath@dsu.edu

***Abstract*—The Multiple Traveling Salesmen Problem (MTSP) is a variation of the Traveling Salesman Problem used in many real-world applications involving travel distance optimization. Many existing solutions further divide this problem into two parts: clustering and routing. I propose a new solution in this paper for a radial clustering and routing algorithm that takes advantage of geometric properties of datasets to efficiently select subsets of points, create partial routes, and optimize them to form a complete solution. Each of three phases of the algorithm are detailed and results of testing, analysis, and comparison are explained.**

Keywords-Multiple Traveling Salesmen Problem, clustering, sweep, loop, mutate

I. Introduction

The traveling salesman problem (TSP) is an NP-hard optimization problem, popular in computer science research due to it's simple premise, applicability to real-world cases, and it presents a challenging opportunity for unique solutions. There are many variations of this problem, and we'll dive deeper into one such form of the closely-related multiple traveling salesman problem (MTSP). Our purpose here is to look at possible real-world applications of solutions to this problem and present a novel algorithm that solves it efficiently and flexibly while keeping certain realistic constraints in mind.

Real-world cases that might utilize such a solution are many. In the field of logistics, manufacturers and distributors try to move goods in a way that minimizes cost of business. Disaster relief workers need to quickly distribute aid to those in need. Military leaders manage complex supply lines where cost of movement can be great and timeliness is essential. In area-coverage challenges, such as aerial surveying for geographical data and crop management, or the placement of fiber internet lines in modern telecommunications, optimal pathing across large maps of points is critical. A similar solution could even be abstracted for use in scientific analysis for DNA sequencing, for finding relative similarities in genetic fragments.

Our objectives here are as follows:

- outline the exact problem to solve, as well as rules and assumptions
- propose and develop a new algorithm as a solution that fits the criteria
- test the algorithm and explore results in order to gain insight into its effectiveness

II. Problem Definition

The general MTSP can be given as a simple question: "What is the set of the most efficient routes that a number of salesmen can take in order to collectively visit all cities on a map?" The MTSP variant that will be the focus of our experiment stems from that question, but we will enforce certain constraints and make some assumptions for the sake of research and applicability to realistic cases.

The map has a single depot at which all salesmen start and end their routes. Every city must be visited exactly once and by only one salesman. All routes will be a complete circuit, one for each salesman, and they each visit a unique non-empty set of cities.

We assume here that a map of cities is given as a dataset of 2-dimensional points representing the cities. The cost of travel is equal to the Euclidean distance between pairs of cities (including the depot as a sort of special city). The depot is chosen from this set of cities nearest the centroid of all cities. The number of salesmen is at least 1 and at most equal to the number of cities, excluding the depot; this number is chosen before executing the algorithm. The cost of all routes is weighted equally and totaled together, equaling total distance traveled by all salesmen; this cost is minimized by the algorithm.

III. Research on Existing Solutions

There are existing solutions to the MTSP that should be mentioned and reviewed as they may help to understand previous approaches to the problem and spark ideas about how to design a new algorithm. Although some of the constraints, assumptions, and focuses may be different than those in our problem definition, they may still provide research value.

"A New Method to Solve the Multi Traveling Salesman Problem with the Combination of Genetic Algorithm and

Clustering Technique" [1] presents an algorithm called CGA-MTSP, which is a two-step approach using clustering and genetic algorithm. Two fitness functions are used, one for each step, for dividing cities into clusters and finding optimal routes respectively. Clustering first appears to be very effective at breaking the problem into smaller parts to reduce computation overhead. The genetic algorithm then iterates through populations of routes represented by chromosomes, using crossover and mutation operations at each stage, then testing for fitness. Some important differences between this setup and ours however, are that the number of salesmen (and therefore the number of clusters) is chosen ahead of time, and that each salesman has a different starting city, or 'depot'. Nevertheless, lessons learned from dividing the problem into smaller parts by clustering can certainly be carried forward.

"UAV Cluster Mission Planning Strategy for Area Coverage Tasks" [2] includes a series of steps, also including a clustering algorithm, and a particle swarm + ant colony optimization algorithm for path planning. The problem in this second study is aimed at optimizing flight paths of UAVs to cover large areas. To abstract and compare the study for research with our own, points along the UAVs' flight paths could be considered cities in the broader problem, offering an MTSP solution through focus on a real-world application. Fuzzy C-means clustering is used in the first step, which allows partial membership of a point to multiple clusters. The ant colony part of their algorithm uses dynamic pheromones and works in tandem with the particle swarm. This takes useful aspects of both systems and effectively utilizes them together to improve efficiency. While the study's focus on physical aspects of flight, including energy restrictions, 3D space, motion, and area coverage are not considered directly in our paper, the demonstration of powerful clustering and combination routing techniques helps build onto a foundation of our algorithmic design process.

Some other papers I researched and drew ideas from are "Finding Hamilton cycles in robustly expanding digraphs" [3], "A New Approach to Solve the Classical Symmetric Traveling Salesman Problem by Zero Suffix Method" [4], "When the greedy algorithm fails" [5], "Density-Based Clustering Heuristics for the Traveling Salesman Problem" [6], and "A Modern Application of the Traveling Salesman Problem: Understanding Wal-Mart" [7].

Finally, "A Comprehensive Survey on the Multiple Travelling Salesman Problem: Applications, Approaches and Taxonomy" [8] analyzes many variations of the problem and existing solutions as a whole. An expansive list of approaches here help to expand our background knowledge of solutions: deterministic/exact, (partheno-) genetic, particle swarm, ant colony, artificial bee colony, market-based, probability, game theory, fuzzy logic, analytical hierarchy process, etc. There is no novel implementation or experiment shown in the study however, only an exploration of existing approaches, problem variants, and research statistics. A broader understanding of these approaches will aid our design process and provide some direction through development as well.

IV. Proposed Solution

Now I'll present my own novel solution to the MTSP called Sweep-Loop-Mutate (SLM). This algorithm attempts to take advantage of some geometrical properties of a relatively centralized depot on the map in order to quickly converge on a near-optimal route cost. Throughout my research on this topic, I have yet to see another algorithm that functions similarly to mine.

The first property SLM utilizes is that cities in significantly different directions from the depot are likely to be far apart, and those in similar directions are likely closer. That is, by tracking the heading of each city relative to the depot, we can intuitively cluster cities that are in the same radial sector and dispatch a salesman for that cluster. Once clusters are initialized, the second property to consider is that routes often follow elliptical loops coming from and returning to the depot. We can divide these loops into two subroutes, left and right. By splitting the cluster sector along its center line, we can greatly simplify initial route construction and create more linear subroutes before joining them at their farthest points to complete the loop. These properties are generalizations however, and are dependent on the placement of the depot and cities on the map.

The first phase of SLM is the sweep. Cities are converted to polar coordinates using the depot as the origin. The list of cities C is then sorted by their heading angle θ. Clusters C_k are formed by divisive hierarchical clustering using the n greatest angular differences between adjacent cities in C, where n is the number of salesmen. Here, sectors and clusters are considered synonymous. The heading angle boundary, where the headings of 0 and 2π radians are equal, is carefully taken into account.

The second phase is the loop. Each sector's bounds, given by the edge cities in C_k, are used to calculate center lines. Cities in C_k are sorted by distance from the depot, then placed into subroutes L and R based on their angles relative to the center line. L and R together form the initial route for each salesman.

The third and final phase of SLM is the mutation. Two operations are performed on the routes and subroutes to attempt to smooth the path of travel and optimize cost. The crossover mutation creates a new sorting of cities by their proximity to the center line. Cities closer to the line are more likely to reduce the total cost when switched between subroutes. Potential costs are are calculated for moving each city individually, and are then swapped from L to R, or vice versa, if the potential cost is lower. Subroutes are then combined into a single ordered path P, forming a complete Hamiltonian cycle. The reorder mutation is allows adjacent cities along P to swap order positions. Again, potential costs are calculated and swaps are executed to optimize the cost. The reorder mutation is performed in a loop until no more swaps occur.

After all phases are complete, routes are finalized and the total cost is calculated using euclidean distances for each edge along every route.

V. Experimental Study and Analysis

To understand the effectiveness of the above methods, I'll be testing my algorithm on several input datasets with different numbers of salesmen, as well as two other algorithms for comparison. The comparison algorithms are Python implementations of Sedighpour, Yousefikhoshbakht, and Darani's genetic algorithm [9] (GA) and Pan and Wang's ant colony optimization algorithm [10] (ACO) (sourced from https://github.com/N0vel/Multiple-travelling-salesman-mTSP-GA-approach and https://github.com/ganyariya/MTSP_ACO respectively). This will serve as a way to benchmark how well SLM performs by measuring both optimized costs and execution times. I designed a testing framework to facilitate running all three algorithms on identical input data and settings. This framework reads in TSP files (sourced from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/), selects the node closest to the centroid of data as the depot for each dataset, executes and records test results from each execution, and optionally generates route plots for easy visualization. All tests are performed on my personal PC with the following specifications: Windows 10 (v22H2 b19045.5247), Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz, x64, 16 GB RAM. Tests are executed using Python version 3.12.3.

Below are selected results of the experiment. Each algorithm was tested on the same set of 41 datasets, using the same depots, and numbers of salesmen in 3, 5, 10. The expanded table with all collected data can be seen in Appendix 1. Column headers are abbreviated to represent the algorithm and number of salesmen (see expanded table), as well as the average values for each respective algorithm. Total cost (distance traveled by all salesmen) and execution time in seconds is recorded in each row. Each row also shows the corresponding dataset name with number of nodes.

Dataset	**Algorithm average (*cost*, *time*)**		
	aco-avg	**ga-avg**	**slm-avg**
eil51	(624.7 3.105s)	(805.6 11.687s)	(836.8 0.001s)
st70	(1042.7 7.280s)	(1673.6 14.271s)	(1397.9 0.001s)
eil76	(767.7 9.088s)	(1273.8 16.086s)	(957.6 0.001s)
rat99	(1857.7 18.927s)	(4122.6 20.504s)	(2613.3 0.001s)
ch130	(9129.7 40.806s)	(25242.7 28.228s)	(12036.9 0.002s)
ch150	(9558.0 62.632s)	(27913.5 34.060s)	(15482.7 0.003s)
rat195	(3392.3 145.439s)	(12154.6 49.152s)	(5622.3 0.003s)
gil262	(3518.0 346.180s)	(16756.8 77.426s)	(6362.4 0.003s)
a280	(3991.0 416.443s)	(20774.9 89.466s)	(14803.4 0.007s)
rd400	(22307.7 1351.855s)	(143771.2 1071.837s)	(47435.2 0.009s)

See Appendix 1 for full data results.

Figure 1. Experiment data for selected datasets

The table above shows results for 10 of the datasets tested. More datasets, up to 52, were available for testing, but execution times were simply too long to continue after running for approximately 12 hours. Therefore results shown here and in the comparison section below are based on the final 41 datasets.

Here are some example output plots to help explain some of the interesting facets of SLM as well as a side-by-side look at another algorithm's routes. Routes taken by different salesmen are differentiated by color.

Figure 2 demonstrates aspects of how each phase of SLM works to create the final set of routes. In both plots, routes never overlap each other because of the sweep phase's radial clustering method. The loop phase tends to generate clean-looking loops for sparse clusters as seen in the top 4 routes for both figures, but creates a much looser zig-zag pattern in figure 2.a due to the distance sorting method used. This is attempted to be remedied by the mutate phase in order to smooth out the loop paths, but is often not robust enough to make intuitive optimizations where multiple nodes should be reordered instead of one. We can see a great improvement however, when a more appropriate number of salesmen is chosen for a given dataset. The lower route is split in two in figure 2.b where just one more salesmen enables better division of clusters.

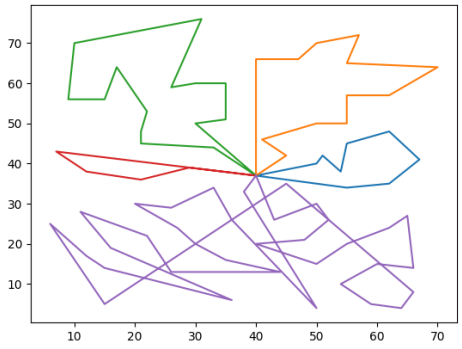
Figure 3 shows example routes from both SLM and ACO on the same dataset. ACO balances its route sizes and often meanders and overlaps. SLM creates loops in opposing directions and has self-overlapping routes where clusters are dense.

VI. Comparison to Existing Solutions

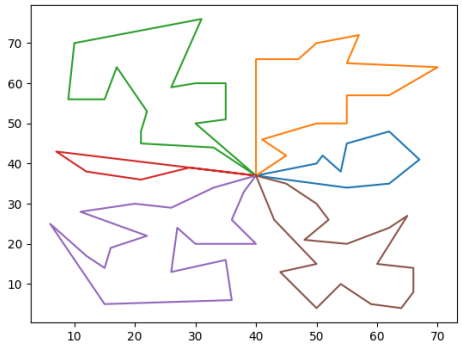
Now I'll compare results from all 3 algorithms tested. This should give a better understanding of how efficient each one is at performing the task of optimizing for lowest cost for MTSP. The first measure of comparison is cost. Lower total costs of all routes describe how close results are to an optimal solution.

Figure 4 shows cost values for 90 different execution results using 3 algorithms, 3 different values for n, and the 10 selected datasets shown in figure 1. We can see a similar overall trend that costs increase as the number of cities increase, but keep in mind that the scale of distances vary greatly between datasets and they are ordered here only for consistency. SLM appears to remain in the middle of the pack for most costs, with GA generally performing worse and ACO generally better. Interestingly though, greater values of n negatively affect GA's and ACO's costs, while SLM's costs decrease as n increases. I believe this is due to computational resources being divided among more agents in the comparative algorithms. SLM is able to refine routes more effectively when clusters are smaller. Based on data from all 41 datasets, SLM's mean cost is 1.623 times as high as ACO and 0.511 times as high as GA. SLM's median cost is 1.451 times as high as ACO and 0.475 times as high as GA.

Figure 5 shows execution times in seconds for the same 90 results as above. Here we see a more strict relationship between execution time and number of cities. Times for both ACO and GA are several orders of magnitude higher than those for SLM. The longest execution times were just shy of 35 minutes, while SLM completed the same tasks in 11 milliseconds or less. This vast difference is due to the comparative algorithms' iterative nature, which repeatedly processes the full datasets, while SLM subdivides data into clusters before

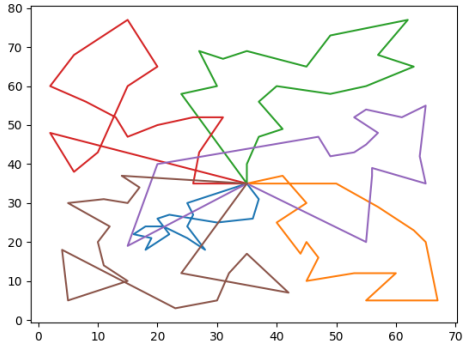


(a) SLM, n = 5

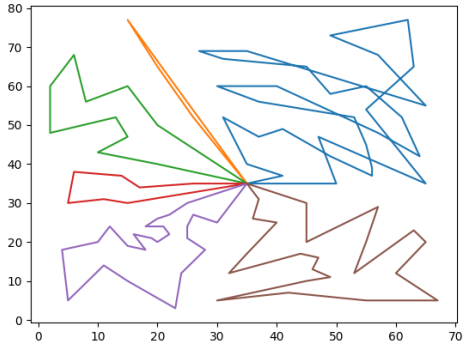


(b) SLM, n = 6

Figure 2. Routes for dataset eil76 showing different numbers of salesmen



(a) ACO, n = 6



(b) SLM, n = 6

Figure 3. Routes for dataset eil101 showing different algorithms

operating on each cluster a much smaller number times. In fact, I was able to independently execute SLM with $n = 10$ on a set of 18512 cities in just under 6 seconds. Based on data from all 41 datasets, SLM's mean time is 5.137e-05 times as long as ACO and 5.653e-05 times as long as GA. SLM's median time is 3.225e-05 times as long as ACO and 5.830e-05 times as long as GA.

Based on this analysis, SLM performs somewhat average in terms of optimal costs, but very well in terms of execution time. It it more effective with a higher number of salesmen, especially where cities are sparse enough to form effective loop routes.

VII. Conclusion

I believe my algorithm has achieved some strong success in some areas and still needs fine-tuning in others, but perhaps the methodology used here can take research on the MTSP in an exciting new direction. My goal when designing SLM was to use properties of routes on a 2-dimensional graph of points in effective and novel ways, while keeping computational efficiency in mind. With more development, SLM has the potential to rival more robust algorithms in finding near-optimal solutions while retaining its quick execution speed.

Listed here are some future works that may present new opportunities for improvement and discovery: testing on randomized depot locations, density-based clustering, self-selection of number of salesmen, reordering multiple cities at a time through mutation, mutation operation based on change in direction between edges, moving cities between clusters, and developing a hybrid algorithm using SLM for quick convergence before switching to another method. I plan to make my algorithm implementation as well as my testing framework available as open source code so that others may build on this work.

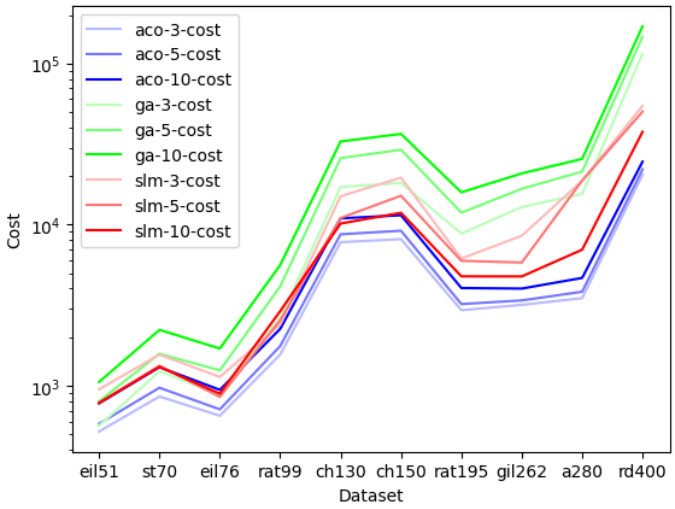


Figure 4. Subset of test results showing cost metrics for selected datasets

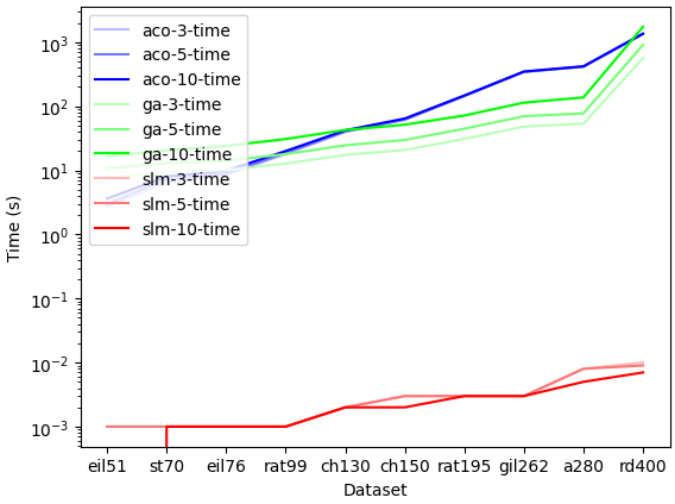


Figure 5. Subset of test results showing time metrics for selected datasets

REFERENCES

[1] J. Li, Q. Sun, M. Zhou, and X. Dai, "A new multiple traveling salesman problem and its genetic algorithm-based solution," pp. 627–632, 2013.

[2] X. Yan, R. Chen, and Z. Jiang, "Uav cluster mission planning strategy for area coverage tasks," *Sensors*, vol. 23, no. 22, 2023.

[3] D. Christofides, P. Keevash, D. Kühn, and D. Osthus, "Finding hamilton cycles in robustly expanding digraphs," *Journal of Graph Algorithms and Applications*, vol. 16, no. 2, p. 335–358, Jan. 2012. [Online]. Available: https://jgaa.info/index.php/jgaa/article/view/paper261

[4] V. Sudhakar and V. Kumar, "A new approach to solve the classical symmetric traveling salesman problem by zero suffix method," 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:6949858

[5] G. Gutin, J. Bang-Jensen, and A. Yeo, "When the greedy algorithm fails," *Discrete Optimization*, vol. 1, 11 2004.

[6] M. Agostinelli, "Density-based clustering heuristics for the traveling salesman problem," 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:55442081

[7] B. M. Jalynne Figueroa and M. Okimoto, "A modern application of the traveling salesman problem: Understanding wal-mart," 2012.

[8] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy," *Computer Science Review*, vol. 40, p. 100369, 2021.

[9] N. M. D. Mohammad Sedighpour, Majid Yousefikhoshbakhtb, "An effective genetic algorithm for solving the multiple traveling salesman problem," 2011.

[10] P. Junjie and D. Wang, "An ant colony optimization algorithm for multiple travelling salesman problem." pp. 210–213, 01 2006.

Appendix 1

Dataset	Algorithm-*n* (*cost*, *time*)											
	aco-3	**aco-5**	**aco-10**	**aco-avg**	**ga-3**	**ga-5**	**ga-10**	**ga-avg**	**slm-3**	**slm-5**	**slm-10**	**slm-avg**
eil51	(519.0 2.728s)	(580.0 2.978s)	(775.0 3.610s)	(624.7 3.105s)	(564.9 7.582s)	(799.2 10.763s)	(1052.9 16.716s)	(805.6 11.687s)	(947.3 0.001s)	(781.3 0.001s)	(781.8 0.000s)	(836.8 0.001s)
berlin52	(9331.0 3.023s)	(9749.0 3.396s)	(13193.0 3.997s)	(10757.7 3.472s)	(11201.5 7.539s)	(15426.6 10.704s)	(21881.6 16.313s)	(16169.9 11.519s)	(10126.4 0.001s)	(10261.6 0.001s)	(14220.2 0.001s)	(11536.1 0.001s)
st70	(856.0 6.539s)	(970.0 7.283s)	(1302.0 8.019s)	(1042.7 7.280s)	(1225.1 9.370s)	(1577.8 12.995s)	(2217.9 20.449s)	(1673.6 14.271s)	(1557.5 0.001s)	(1328.3 0.001s)	(1307.8 0.001s)	(1397.9 0.001s)
eil76	(650.0 8.507s)	(712.0 8.898s)	(941.0 9.859s)	(767.7 9.088s)	(880.0 10.106s)	(1243.1 14.063s)	(1698.2 24.089s)	(1273.8 16.086s)	(1131.4 0.001s)	(850.7 0.001s)	(890.7 0.001s)	(957.6 0.001s)
pr76	(132180.0 8.564s)	(150038.0 8.959s)	(195870.0 9.981s)	(159362.7 9.168s)	(192480.9 10.009s)	(277337.9 14.143s)	(380181.0 23.902s)	(283333.3 16.018s)	(220532.2 0.001s)	(191869.9 0.001s)	(194739.3 0.001s)	(202380.5 0.001s)
rat99	(1561.0 18.029s)	(1761.0 18.712s)	(2251.0 20.039s)	(1857.7 18.927s)	(2630.5 12.742s)	(4131.1 17.963s)	(5606.1 30.807s)	(4122.6 20.504s)	(2396.1 0.001s)	(2544.5 0.001s)	(2899.2 0.001s)	(2613.3 0.001s)
kroA100	(28891.0 18.834s)	(33495.0 19.541s)	(42437.0 21.037s)	(34941.0 19.804s)	(52180.0 12.992s)	(76207.5 18.040s)	(110222.2 28.541s)	(79536.5 19.858s)	(45925.8 0.001s)	(46640.3 0.001s)	(45254.3 0.001s)	(45940.1 0.001s)
kroB100	(28728.0 19.038s)	(33143.0 19.627s)	(44469.0 21.227s)	(35446.7 19.964s)	(49054.0 12.999s)	(77752.9 18.310s)	(102235.6 30.880s)	(76347.5 20.730s)	(50388.1 0.001s)	(51321.9 0.001s)	(51763.6 0.001s)	(51157.9 0.001s)
kroC100	(25603.0 18.863s)	(30847.0 19.400s)	(44005.0 20.953s)	(33485.0 19.739s)	(55860.7 13.055s)	(73890.8 18.169s)	(97259.7 28.584s)	(75670.4 19.936s)	(78783.1 0.002s)	(47949.4 0.001s)	(50021.1 0.001s)	(58917.8 0.002s)
kroD100	(27407.0 19.012s)	(31034.0 19.495s)	(41769.0 20.731s)	(33403.3 19.746s)	(44632.1 13.174s)	(66956.1 18.245s)	(106199.1 31.515s)	(72595.8 20.978s)	(44954.8 0.001s)	(40203.6 0.001s)	(41130.9 0.001s)	(42096.4 0.001s)
kroE100	(28088.0 19.017s)	(32495.0 19.548s)	(44981.0 20.997s)	(35188.0 19.854s)	(50220.2 12.974s)	(68525.1 18.247s)	(98937.2 30.807s)	(72560.8 20.676s)	(51379.8 0.001s)	(53446.9 0.001s)	(51327.5 0.001s)	(52051.4 0.001s)
rd100	(10253.0 18.856s)	(11597.0 19.547s)	(14830.0 20.956s)	(12226.7 19.786s)	(19420.5 13.026s)	(28157.1 18.059s)	(39823.7 31.108s)	(29133.7 20.731s)	(24048.9 0.002s)	(14361.6 0.001s)	(16276.1 0.001s)	(18228.9 0.001s)
eil101	(789.0 19.302s)	(838.0 19.760s)	(1047.0 20.946s)	(891.3 20.003s)	(1235.3 13.023s)	(1693.7 18.342s)	(2267.1 28.506s)	(1732.0 19.957s)	(1165.2 0.002s)	(947.4 0.001s)	(1044.8 0.001s)	(1052.5 0.001s)
lin105	(19250.0 21.857s)	(20412.0 22.431s)	(28562.0 23.935s)	(22741.3 22.741s)	(34104.9 13.745s)	(56402.5 19.173s)	(72118.8 33.007s)	(54208.7 21.975s)	(33540.5 0.001s)	(32617.2 0.001s)	(32825.2 0.001s)	(32994.3 0.001s)
pr107	(69145.0 22.679s)	(88279.0 23.143s)	(136427.0 24.702s)	(97950.3 23.508s)	(137882.5 13.933s)	(224797.8 19.527s)	(282426.7 30.599s)	(215035.7 21.353s)	(101131.4 0.001s)	(102078.7 0.001s)	(85482.8 0.001s)	(96231.0 0.001s)
pr124	(77649.0 35.229s)	(91702.0 35.887s)	(140046.0 37.854s)	(103132.3 36.323s)	(194808.2 16.680s)	(312227.7 23.448s)	(409664.8 37.336s)	(305566.9 25.821s)	(104915.7 0.002s)	(93290.3 0.002s)	(111793.2 0.001s)	(103333.0 0.002s)
bier127	(140070.0 38.708s)	(153262.0 39.084s)	(176385.0 41.075s)	(156572.3 39.622s)	(244318.1 17.316s)	(373677.8 24.583s)	(484294.8 41.743s)	(367430.2 27.881s)	(251704.0 0.002s)	(202038.1 0.001s)	(181996.3 0.001s)	(211912.8 0.001s)
ch130	(7770.0 39.755s)	(8700.0 40.495s)	(10919.0 42.168s)	(9129.7 40.806s)	(17103.1 17.416s)	(25840.6 24.571s)	(32784.5 42.698s)	(25242.7 28.228s)	(14993.2 0.002s)	(10979.3 0.002s)	(10138.1 0.002s)	(12036.9 0.002s)
pr136	(130274.0 44.784s)	(147552.0 45.360s)	(186994.0 47.045s)	(154940.0 45.730s)	(275123.0 18.222s)	(407647.3 25.966s)	(556670.1 41.058s)	(413146.8 28.415s)	(203756.9 0.002s)	(193533.9 0.002s)	(163708.2 0.001s)	(186999.7 0.002s)
pr144	(77091.0 52.946s)	(94410.0 53.620s)	(134531.0 55.647s)	(102010.7 54.071s)	(258319.9 20.138s)	(385540.5 28.079s)	(535948.2 48.958s)	(393269.6 32.392s)	(113967.7 0.002s)	(105176.7 0.002s)	(120493.2 0.001s)	(113212.5 0.002s)
ch150	(8115.0 61.506s)	(9147.0 62.003s)	(11412.0 64.389s)	(9558.0 62.632s)	(18123.3 20.816s)	(29111.0 29.798s)	(36506.2 51.567s)	(27913.5 34.060s)	(19532.9 0.003s)	(15090.7 0.003s)	(11824.3 0.002s)	(15482.7 0.003s)
kroA150	(35305.0 60.675s)	(38542.0 61.781s)	(49496.0 63.613s)	(41114.3 62.023s)	(84037.8 21.049s)	(128612.1 29.833s)	(177522.3 51.589s)	(130057.4 34.157s)	(69177.1 0.002s)	(60467.6 0.002s)	(55837.2 0.002s)	(61827.3 0.002s)
kroB150	(34813.0 61.266s)	(38509.0 61.889s)	(51041.0 63.899s)	(41454.3 62.351s)	(80483.9 20.801s)	(125211.4 29.530s)	(179729.6 51.297s)	(128475.0 33.876s)	(76421.5 0.002s)	(72128.6 0.002s)	(79144.0 0.002s)	(75898.0 0.002s)
pr152	(97708.0 62.965s)	(123144.0 63.882s)	(162761.0 66.056s)	(127871.0 64.301s)	(325461.4 21.058s)	(527450.8 29.983s)	(696195.2 51.875s)	(516369.1 34.305s)	(202377.3 0.002s)	(153413.0 0.002s)	(166285.3 0.001s)	(174025.2 0.002s)
u159	(54694.0 74.103s)	(63768.0 76.129s)	(83056.0 81.491s)	(67172.7 77.241s)	(150474.5 23.337s)	(222214.5 33.533s)	(296380.5 53.593s)	(223023.2 36.821s)	(91413.8 0.003s)	(82428.1 0.002s)	(81318.2 0.002s)	(85053.4 0.002s)
rat195	(2936.0 144.054s)	(3207.0 144.802s)	(4034.0 147.461s)	(3392.3 145.439s)	(8780.8 30.973s)	(11846.5 44.666s)	(15836.5 71.818s)	(12154.6 49.152s)	(6147.5 0.003s)	(5951.5 0.003s)	(4767.9 0.003s)	(5622.3 0.003s)
d198	(20528.0 152.619s)	(21876.0 154.032s)	(29772.0 157.909s)	(24058.7 154.853s)	(63655.5 31.572s)	(93005.6 45.516s)	(129112.6 73.467s)	(95257.9 50.185s)	(37426.1 0.004s)	(27233.2 0.003s)	(26215.3 0.003s)	(30291.5 0.004s)
kroA200	(39251.0 163.189s)	(43228.0 163.847s)	(54471.0 164.369s)	(45650.0 163.802s)	(135320.9 31.401s)	(175022.2 45.025s)	(236324.0 79.537s)	(182222.4 51.988s)	(89445.2 0.004s)	(91496.9 0.003s)	(88285.7 0.003s)	(89742.6 0.003s)
kroB200	(39032.0 152.898s)	(43358.0 152.494s)	(54202.0 154.087s)	(45530.7 153.160s)	(140828.2 31.440s)	(170831.1 45.423s)	(227327.7 73.248s)	(179662.3 50.037s)	(86924.1 0.003s)	(87158.1 0.003s)	(75972.4 0.002s)	(83351.5 0.003s)
ts225	(154361.0 212.565s)	(167394.0 214.163s)	(213754.0 216.916s)	(178503.0 214.548s)	(684748.4 37.237s)	(946869.5 53.811s)	(1152041.9 86.925s)	(927886.6 59.324s)	(721708.1 0.005s)	(321841.5 0.004s)	(200527.2 0.002s)	(414692.2 0.004s)
tsp225	(5200.0 210.026s)	(5603.0 211.019s)	(6773.0 214.116s)	(5858.7 211.721s)	(17030.8 37.122s)	(23374.4 53.630s)	(30112.8 86.465s)	(23506.0 59.072s)	(9430.7 0.004s)	(7101.8 0.003s)	(7948.5 0.003s)	(8160.3 0.003s)
pr226	(116714.0 218.387s)	(139514.0 220.037s)	(191528.0 223.502s)	(149252.0 220.642s)	(616364.5 37.483s)	(877951.2 53.988s)	(1167879.6 87.068s)	(887398.4 59.513s)	(406777.6 0.005s)	(216335.7 0.004s)	(187109.9 0.002s)	(270074.4 0.004s)
gil262	(3173.0 345.912s)	(3379.0 344.028s)	(4002.0 348.599s)	(3518.0 346.180s)	(12847.2 48.231s)	(16664.2 69.968s)	(20758.9 114.080s)	(16756.8 77.426s)	(8517.0 0.003s)	(5809.4 0.003s)	(4760.8 0.003s)	(6362.4 0.003s)
pr264	(71946.0 336.200s)	(85311.0 336.709s)	(124907.0 342.892s)	(94054.7 338.600s)	(375237.8 48.834s)	(558786.1 70.782s)	(722315.0 126.159s)	(552112.9 81.925s)	(183545.4 0.004s)	(177029.8 0.004s)	(147504.6 0.003s)	(169359.9 0.004s)
a280	(3488.0 413.254s)	(3825.0 415.514s)	(4660.0 420.562s)	(3991.0 416.443s)	(15465.8 53.525s)	(21315.7 77.479s)	(25543.0 137.395s)	(20774.9 89.466s)	(18698.6 0.008s)	(18735.5 0.008s)	(6976.0 0.005s)	(14803.4 0.007s)
pr299	(66520.0 526.809s)	(71635.0 529.683s)	(92141.0 539.293s)	(76765.3 531.929s)	(325573.8 61.188s)	(427121.7 88.457s)	(501750.7 143.163s)	(418148.7 97.603s)	(163822.3 0.006s)	(173236.1 0.006s)	(171572.9 0.005s)	(169543.7 0.006s)
lin318	(56883.0 630.306s)	(60094.0 643.760s)	(72938.0 640.963s)	(63305.0 638.343s)	(298827.7 67.810s)	(389390.8 97.812s)	(462548.2 175.407s)	(383588.9 113.676s)	(194733.6 0.006s)	(123624.6 0.005s)	(114119.9 0.005s)	(144159.4 0.005s)
linhp318	(56583.0 632.879s)	(60711.0 633.889s)	(73038.0 640.868s)	(63444.0 635.879s)	(298827.7 68.287s)	(389390.8 99.218s)	(462548.2 175.647s)	(383588.9 114.384s)	(194733.6 0.006s)	(123624.6 0.005s)	(114119.9 0.005s)	(144159.4 0.005s)
rd400	(20355.0 1339.321s)	(22015.0 1356.546s)	(24553.0 1359.699s)	(22307.7 1351.855s)	(114807.8 565.480s)	(146405.2 911.886s)	(170100.7 1738.146s)	(143771.2 1071.837s)	(54481.2 0.010s)	(50202.5 0.009s)	(37621.8 0.007s)	(47435.2 0.009s)
fl417	(18374.0 1429.938s)	(21414.0 1440.160s)	(30236.0 1432.916s)	(23341.3 1434.338s)	(214026.7 617.966s)	(302423.6 1014.566s)	(353782.5 1891.546s)	(290077.6 1174.692s)	(49233.2 0.010s)	(50264.1 0.011s)	(26814.5 0.006s)	(42103.9 0.009s)
pr439	(148790.0 1719.546s)	(155355.0 1751.453s)	(183989.0 1754.998s)	(162711.3 1741.999s)	(1067355.4 677.530s)	(1244208.6 1092.045s)	(1661667.1 2094.853s)	(1324410.4 1288.143s)	(499466.5 0.011s)	(482057.3 0.011s)	(336997.8 0.007s)	(439507.2 0.009s)

TABLE I. Experiment data for all tested datasets