Parallel Computing for Iterative Hill Climbing Algorithm to solve TSP

Pramod Yelmewad, Param Hanji, Amogha Udupa, Parth shah, Basavaraj Talawar
System Parallelization & Architecture Research @ nitK (SPARK)
National Institute of Technology Karnataka, Surathkal, Mangalore, India
Email: pramod.cs16f04, 14IT129.param, 14IT204.amogha{@nitk.edu.in}, parths1229@gmail.com, basavaraj@nitk.edu.in

***Abstract*—Traveling Salesman Problem (TSP) is an NP-hard combinatorial optimization problem. Approximation algorithms have been used to reduce the TSP factorial time complexity to non-deterministic polynomial time successfully. However, approximation methods result in a suboptimal solution as they do not cover the entire search space adequately. Further, approximation methods are too time consuming for large input instances. GPUs have been shown to be effective in exploiting data and memory level parallelism in large, complex problems.**

Our novel Parallel Iterative Hill Climbing (PIHC) algorithm using the Variable-Nearest Neighborhood initial route gives near-optimal solution of large instance TSP problem on the GPU in a reasonable amount of time. We demonstrate improved cost quality on symmetric TSPLIB instances ranging from 200 to 85,900 cities. Our GPU implementation achieves a speedup of up to 279× over its sequential counterpart and 373× over the state-of-the-art GPU based TSP solver. The implementation gives a quality cost with error rate 0.71% in best case and 8.06% in worst case which is better than the state-of-the-art GPU based TSP solvers.

***Index Terms*—Traveling Salesman Problem, Variable Nearest Neighborhood, Approximation methods, GPU, Parallel Iterative Hill Climbing, TSPLIB.**

I. Introduction

A. The Traveling Salesman Problem

Traveling Salesman Problem (TSP)[1] is an NP-hard[2], $O(n!)$ combinatorial optimization problem. The time complexity of TSP is factorial when solved using brute-force method and exponential when solved using dynamic programing. Owing to its large number of applications in science and engineering, time efficient TSP solutions are of great importance.

The objective of TSP is to find the minimum route that passes through each city exactly once, returning to the originating city in the end. In short, in the map of n cities, distance of a route π has to be minimized,

$$d(\pi) = \sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \quad (1)$$

where $d_{i,j}$ is distance between city i and city j and π is combination from $(1, 2, ..., n)$[3].

B. Exact and Approximation Approaches

Optimization problems are solved using two broad types; viz., exact methods and approximation methods. In the worst-case, exact methods have to explore the entire search space to determine the optimal solution. Approximation methods give near-optimal solutions by exploring a limited search space in a reasonable amount of time[4]. In the worst case, finding the optimal solution for n city TSP involves exploring $\frac{(n-1)!}{2}$ feasible solutions in case of brute-force approach and $n^2 \times 2^n$ feasible solutions in case of dynamic programing. While the exact method always assures an optimal solution, the CPU time is prohibitive for large n. Some examples of exact methods are brute-force algorithm, dynamic programing, branch and bound, branch and cut approaches. Due to the factorial time complexity, exact methods are time-efficient approaches to solve large TSP instances.

Approximation algorithms explore a subset of feasible solutions by constraining the search space, and derive satisfactory solutions in a reasonable amount of time. Approximation methods start with a random initial solution and iteratively improve on it to converge to a near-optimal solution until some termination criteria is met[4]. The termination criteria may include the number of iterations, an optimal cost value, no improvement of solutions between iterations, etc. Popular approximation approaches are iterative local search, hill climbing, tabu search, simulated annealing, ant colony optimization, and genetic algorithm. Selection of subset of feasible solutions is dependent on the initial solution. Once the initial solution is determined, a subset of feasible solutions is generated based on it. Some feasible solution generation methods are 2-opt, 3-opt, k-opt, crossover and mutation.

In this paper, we have used the hill climbing approach as the base algorithm to implement a GPU based solution for the TSP problem. The initial route construction is performed based on Nearest-Neighborhood Method, rather than setting it up randomly or in-sequence. In the paper, we also reveal the benefit of constructing an initial route in this manner. Since Graphics Processing Unit (GPU) exploits data, memory and task level parallelism in applications to improve their performance significantly[5][6], we present the effectiveness of the proposed parallel strategy by comparing it with state-of-the-art iterative hill climbing based TSP solvers. Section II presents work-flow of Iterative Hill Climbing(IHC) algorithm and Section III presents the parallel strategy for IHC to solve TSP.

II. Iterative Hill Climbing Algorithm

Hill climbing is a popular approach used in approximation methods to solve TSP. It is a searching strategy used to find the best solution by performing iterative improvements over an initial solution. The steps involved in the hill climbing approach are:

- First, the initial solution, also called the candidate solution, s is determined.
- Once the initial solution is determined, its associating cost, $f(s)$, is calculated.
- Next, the neighborhood solutions, $N(s)$ of initial solution are generated using swapping techniques like 2-opt exchange.
- $\forall s^{'} \in N(s)$, $f(s^{'})$ is calculated using distance calculation method, where $s^{'}$ is neighborhood solution and $f(s^{'})$ is corresponding cost of $s^{'}$.
- Cost of each $f(s^{'})$ is compared with the initial cost $f(s)$. The best improved $f(s^{'})$ on the $f(s)$ is considered the locally optimal cost.
- To move towards the globally optimal solution, this algorithm must be called repeatedly. For each call, the locally found optimal solution $s^{'}$ acts as the initial solution s in the next iteration.
- This process continues until a termination criteria is met. Termination criteria could be the number of iterations, further improvement not possible or optimal solution reached. For each call of the hill climbing procedure, cost quality of the solution will potentially improve and move towards the global optimal.

Important components of the iterative hill climbing approach are initial solution construction and its neighborhood solutions generation. These steps are explained in the section below.

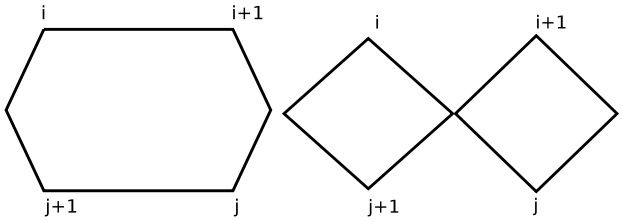
A. Initial Solution Construction

Construction of an initial solution is the first step in an optimization problem using approximation methods. Initial route is traditionally chosen sequentially or generated randomly. We present an intelligent initial route generation method using the Variable-Nearest Neighborhood (VNN). Initial route construction methods are also discussed below.

1) Sequenced Route: This is a fundamental way to construct the initial route where the route is made up of cities in sequence Eg. 1,2,3,. . .,n,1. In the worst-case, the sequenced route gives a maximal weighted Hamiltonian cycle. This is because, route formation takes place without considering distance between any pair of cities.

2) Random Route: Starting from the source node, the initial route is constructed by choosing a random neighbor for each node. The route computation complexity is $O(n)$. Similar to sequenced route, in the worst-case, the random route also gives a maximal weighted Hamiltonian cycle.

Most approximation-based TSP implementations [7][8][9][10][11][12] use randomly constructed TSP routes. Both the above methods do not consider the distances while identifying the neighbors.



(a) Before 2-opt move (b) After 2-opt move

Fig. 1: The 2-opt neighborhood generation mechanism

3) Variable-Nearest Neighborhood Route (VNN): The VNN initial route construction is motivated by the k-NN classification[13]. Instead of fixed k neighbors, we consider variable length neighbors in order to set an effective initial route. In this route construction technique, a random city is chosen as the source city (we use first city). The next city is the least cost neighbor among the $n-1$ unvisited neighbors of the current city. The neighbor selection procedure continues until all cities are visited. The last hop returns to the source city. This route construction mechanism is called Variable-Nearest Neighborhood (VNN) route construction. Since we use symmetric TSPLIB instances, for n city problem, VNN needs $O(n^2)$ time to construct initial route. To find an upper bound on the cost of this initial route, assume $cost_{opt}$ is optimal cost and $cost_{route}$ is cost of constructed route. The upper bound on cost of route is $\frac{cost_{route}}{cost_{opt}} \leq 0.5 \times (log_2 n + 1)$[14].

B. Neighborhood Solution Generation

Once the initial route construction is determined, the next task is to generate neighborhood solutions of the initial route.

1) 2-opt Move: 2-opt move is a fundamental technique to generate neighborhood solutions. The n-city TSP problem has $(n-1)!$ different feasible solutions. 2-opt method reduces the search space to be explored to an order of $O(n^2)$. In the 2-opt method, neighborhood solution is generated by removing two edges from the initial route, and then reconnecting the two newly created sub-routes so that a different and unique Hamiltonian Cycle is generated.

The 2-opt method includes the following steps to generate neighbor:

- Consider a pair of any two cities as i and j. For every pair, perform the bellow steps.
- Remove the edge between city i and city $i+1$, and the edge between city j and city $j+1$.
- Add a new edge between city i and city j, and another new edge between city $i+1$ to city $j+1$.

Figure 1 illustrates the 2-opt move mechanism between city i and j. Figure 1(a) is the original graph before applying the 2-opt move and Figure 1(b) is the newly generated neighborhood after applying the 2-opt move.

The 2-opt neighborhood generation mechanism is performed on all possible pairs of the cities in the initial route. Therefore, number of neighborhood solutions generated with 2-opt move is $\frac{n \times (n-1)}{2}$, where n is the total number of cities,

and the time complexity of 2-opt neighborhood generation mechanism is $O(n^2)$. The 3-opt move is variation of 2-opt move. It significantly slows down the searching process because it removes three edges and reconnects them in a such way that a unique Hamiltonian cycle is formed. Since three edges are removed and added, 3-opt mechanism explores $\frac{n\times(n-1)\times(n-2)}{6}$ feasible solutions and the time complexity of seeking the best neighborhood solution becomes $O(n^3)$.

III. Parallel Iterative Hill Climbing (PIHC) Algorithm

A. GPU Implementation

The Parallel Iterative Hill Climbing algorithm is presented in this section. Figure 2 presents the flowchart of the complete PIHC algorithm. Since GPU computing is heterogeneous in nature, the task of solving TSP is distributed between CPU and GPU. The details of the work carried out by CPU are given as follows:

- Determine the initial route using one of the route construction techniques described and calculate its cost using the distance calculation formula.
- Allocate space on the GPU global memory for the initial route, initial cost, distance matrix, and coordinates of TSPLIB instances.
- Transfer required data for computation on GPU like initial route and its cost, distance matrix, and coordinates of TSPLIB instances.
- Determine optimal number of threads, blocks to be used for computation and launch the kernel.
- After the kernel completion, checks fitness of the best solution determined by GPU at each iteration to decide whether to iterate again or not.
- When termination criteria is met, the search space exploration is stopped. CPU returns the best improved route and its cost.

The GPU is responsible for following tasks:

- Generate neighborhood solutions on initial route in parallel. Each thread is involved in neighborhood generation.
- Each thread computes cost of its corresponding neighborhood solution using the distance calculation method.
- Maintain synchronization among threads of each block to get accurate results.
- Based on cost of each solution, choose minimal cost, identify the corresponding thread and write these details to global memory enabling the CPU to read these details and decide whether next improvement is possible or not.

1) Neighborhood - Thread Mapping Strategy: This is an important step that enables parallel execution of the specified task. Therefore, the process of mapping the neighborhood generation to threads is a crucial step to ensure performance acceleration.

Each thread starts with a unique i and j obtained using its $thread\_id$. The threads then concurrently generate all neighborhood solutions and compute the respective costs.

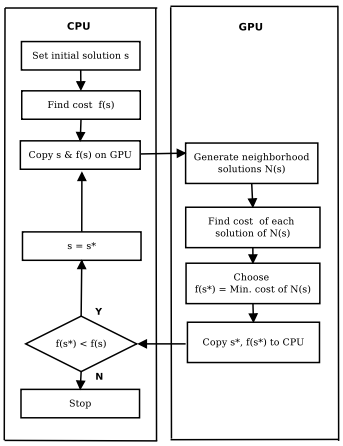


Fig. 2: Parallel Iterative Hill Climbing (PIHC) algorithm flowchart.

The total number of threads involved in this strategy is $\frac{n\times(n-1)}{2}$ using 2-opt move approach. In this case, the grid dimension will be $\left\lceil \frac{\frac{n\times(n-1)}{2}}{block\_dimension} \right\rceil$.

To optimize efficiency of PIHC algorithm, distances of cities are calculated on-the-fly instead of keeping pre-calculated distance matrix. The cities are arranged in order, according to the route generated by 2-opt approach.

B. Xeon Phi Implementation

we are planning to solve the TSP problem on the Intel Xeon Phi Knights Landing processor. This parallel architecture should theoretically give more boost up to 2-opt with VNN method. The plan here is to run the proposed method in heterogeneous cluster of Intel Xeon Phis with use of OpenMP+MPI. The proposed method can be optimized for TSP by the efficient use of scaler tuning, vectorization, multi threading and memory. The NVIDIA K40 GPU has 12GB of RAM which is lower than 16GB high bandwidth memory provided by Intel Xeon Phi KNL architecture which allows us to load more cities than GPUs. The clock speed of KNL is 1.4GHz which is double than K40 GPU, this allows more instructions per cycle to be executed. Combined optimization of Vectors AVX-512 and OpenMP parallel architecture should theoretically give more boost up to 2-opt with VNN method.

IV. Result Analysis

The proposed Parallel Iterative Hill Climbing algorithm has been evaluated over Nvidia's Tesla K40m GPU. The GPU has compute capability 3.5, 15 streaming multiprocessors with each multiprocessor containing 192 cores and running at 745 MHz frequency, global memory of 12 GB, per block shared memory of 48 KB with 65536 registers available at each block. The time analysis of sequential counterpart has been carried out on a 64 bit system which has 8 cores running at 3.6 GHz frequency. Evaluation has been performed using CUDA version 8 and also on OpenCL version 1.2. The

optimal cost of TSPLIB instances are known, therefore we have considered the symmetric TSPLIB[15] instances ranging from 200 to 85900 cities to determine the effectiveness of PIHC TSP solver in terms of speedup as well as cost. We present a time analysis of intial route construction methods, and time as well as cost analyses of the PIHC algorithm.

A. Initial Route Construction Result Analysis

Initial route construction is a crucial step in the approximation method. The time analysis of the three variations of initial route construction are presented.

1) Time Analysis: Figure 3 presents total execution time of PIHC TSP implementations. Each input instance is executed for the three initial route construction methods. Results for the TSPLIB instances from 200 to 18512 cities are shown. For moderately sized TSP instances ranging from 200 to 5934 cities, all three route construction mechanisms take similar time to find the near-optimal solution. As the number of cities increases (TSPLIB instances rl15934 and above), the total execution time taken by both sequenced and random route also increase drastically.

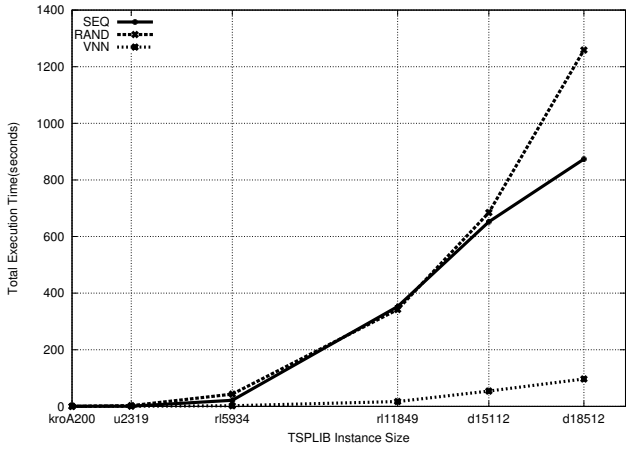


Fig. 3: Total PIHC execution time of sequenced, random, VVN initial route construction mechanism for TSPLIB instances

For the d18512 instance, the total execution times required are 873.67 s, 1258.91 s and 99.94 s respectively. Using Variable-Nearest Neighborhood route as the initial solution makes PIHC GPU implementation 8.78× faster than GPU implementations that use sequenced route, and 12.59× faster than random route construction. This happens because VNN route gives a good quality solution at the initial stage itself. Hence, PIHC needs lesser number of iterations to reach near-optimal solution compared to approaches using other initial route configuration methods.

The Variable-Nearest Neighborhood initial route construction mechanism takes care to choose minimal weighted neighboring city while constructing the initial route, unlike sequenced and random mechanisms. From the presented analysis, we conclude that the VNN route construction mechanism creates an initial solution closest to the optimal solution leading to faster PIHC TSP solvers.

B. Performance Analysis of PIHC

In this section we compare our PIHC implementation against two well-known state-of-the-art GPU-based TSP solvers using approximation methods namely TSP2.1[16] and LOGO[17]. We consider TSPLIB[15] instances ranging from 200 to 85900 cities, whose optimal costs are known in the TSPLIB testbed. The cost quality of LOGO TSP is reported in[17]. To obtain the cost quality of TSP2.1 (http://cs.txstate.edu/~burtscher/research/TSP_GPU/index.html) solver, each instance is run once using the flags $-O3 - arch = sm\_35 - use\_fast\_math$, 100 restart. The cost and time taken are recorded.

1) Cost Analysis: Table I presents the cost quality analysis of the state-of-the-art TSP solvers along with our implementation based on the PIHC algorithm. The PIHC algorithm gives the best quality solutions (nearest to the optimal) for every input instance compared to LOGO and TSP2.1. The error ranges (best case - worst case) for PIHC , LOGO, TSP2.1 are : 0.71% - 5.44%, 4.71% - 11.66%, 4.76% - 12.12% respectively.

The primary reason for the better quality results of the PIHC compared to the LOGO and TSP2.1 is the Variable-Nearest Neighborhood initial route construction technique. VNN provides an initial route whose cost is within $0.5\times(log_2 n+1)$ times the optimal cost for each input instance. Starting from such an initial route, the search space for the hill climbing algorithm is reduced and a higher quality route is obtained within a reasonable time. TSP2.1 uses a random tour as its initial route, whereas LOGO uses Multiple Fragment[18] heuristic to determine initial route.

2) Time Analysis: Since TSP2.1 source code is available, it was run on the same GPU as the PIHC, and the execution times were recorded. We report the full execution times from the start to the end of the respective implementation (including both the CPU and GPU time). Table I shows the execution time comparison of TSP2.1 with our PIHC approach. Compared to the TSP2.1, PIHC$_{global}$ gives a speedup in the range of 0.23× - 206× for the instances of the sizes 200 - 33810 cities. The TSP2.1 implementation for pla85900 instance did not finish on our GPU. The PIHC$_{vector}$ implementation actually outperformed even the PIHC$_{global}$ implementation, giving a speedup of 0.12× - 373.22×. The increased speedup is primarily due to the fact that optimizations are performed using the fast built-in vector operations for representing coordinates.

V. Conclusion

We present a GPU-based Parallel Iterative Hill Climbing (PIHC) algorithm using Variable-Nearest Neighborhood (VNN) initial route construction to solve symmetric TSPLIB instances ranging from 200 to 85900 cities in a reasonable amount of time with good quality cost. We reveal the benefit

TABLE I: Cost quality and execution time comparison of the proposed PIHC algorithm with the state-of-the-art TSP solvers using TSPLIB instances. Values in the parentheses are deviations from the optimal cost.

Instance	Cost (error rate)				Total Execution Time (seconds)			
	LOGO	TSP2.1	PIHC	Optimal	TSP 2.1	PIHC$_{global}$	PIHC$_{vector}$	Speedup
kroA200	31685 (7.78%)	30768 (4.76%)	29579 (0.71%)	29368	0.0412	0.1	0.34	0.12
rat783	9658 (9.67%)	9649 (9.57%)	9002 (2.22%)	8806	0.7974	0.37	0.35	2.28
vm1084	267210 (11.66%)	257729 (7.70%)	250992 (4.88%)	239297	2.3	0.39	0.36	6.39
pcb3038	147690 (7.25%)	153150 (11.22%)	145190 (5.44%)	137694	48.327	0.83	0.66	73.22
fnl4461	194746 (6.67%)	201764 (10.51%)	189881 (4.0%)	182566	154.02	1.8	1.33	115.8
rl5934	582958 (4.84%)	623463 (12.12%)	581802 (4.63%)	556045	397.65	2.27	1.97	201.85
d15112	1652806 (5.06%)	1738705 (10.52%)	1650340 (4.91%)	1573084	6214.77	55.81	32.67	190.23
d18512	675638 (4.71%)	716925 (11.11%)	671000 (3.99%)	645238	11500.96	99.94	58.15	197.78
pla33810	69763154 (5.21%)	73840715 (11.79%)	69494989 (5.21%)	66048945	70792.96	342.17	189.68	373.22
pla85900	149708033 (5.14%)	-	148470854 (4.27%)	142382641	-	5024.64	-	-

of using VNN approach to construct the initial route rather than setting up randomly or in-sequence.

Our proposed PIHC is a two pass approach. In the first pass, we construct the initial route using VNN. In the second pass, improvement is done on the initial route by applying our Parallel Hill Climbing Algorithm iteratively until the termination condition is met. The PIHC implementation using VNN is 8.78× than the sequenced route approach and 12.59× faster than the random initial route construction mechanisms. The worst-case cost of the initial route using VNN is within 1.08× the optimal cost of corresponding TSPLIB instance. For the sequenced and random initial route construction mechanism, the worst-case cost initial routes are 92.83× and 94.12× TSPLIB optimal cost. Hence, the VNN route construction approach significantly reduces the search space and results in a good cost quality solution in a reasonable amount of time.

Proposed approach is implemented using CUDA and OpenCL, and then evaluated and compared with LOGO and TSP2.1 GPU based state-of-the-art TSP solvers. It is observed that our implementations provide very good results with error rate 0.7% in best case and 8.05% in worst case. The proposed PIHC$_{global}$ implementation achieves 206× speedup over GPU based TSP2.1 implementation and 154× speedup over the sequential implementation. The corresponding speedups achieved by the PIHC$_{vector}$ implementation are 373× and 279× respectively. The faster execution of the PIHC$_{vector}$ implementations is a direct consequence of the built-in vector.

REFERENCES

[1] M. Johnson, D.S. and L.A., "The traveling salesman problem: a case study in local optimization," in *Local Search in Combinatorial Optimization*, 1997.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[3] P. Merz and B. Freisleben, "Memetic algorithms for the traveling salesman problem," in *Complex Systems*, 1997.

[4] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.

[5] S. Che, S. Che, M. Boyer, M. Boyer, J. Meng, J. Meng, J. W. Sheaffer, J. W. Sheaffer, K. Skadron, and K. Skadron, "A performance study of general- purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, 2008.

[6] S. Ryoo, C. I. Rodrigues, S. S. Stone, J. A. Stratton, S. Z. Ueng, S. S. Baghsorkhi, and W. m. W. Hwu, "Program optimization carving for GPU computing," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1389–1401, 2008.

[7] N. Fujimoto and S. Tsutsui, "A highly-parallel tsp solver for a gpu computing platform," in *Proceedings of the 7th International Conference on Numerical Methods and Applications*, ser. NMA'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 264–271.

[8] J. Zhao, Q. Liu, W. Wang, Z. Wei, and P. Shi, "A parallel immune algorithm for traveling salesman problem and its application on cold rolling scheduling," *Information Sciences*, vol. 181, no. 7, pp. 1212–1223, 2011.

[9] "Enhancing data parallelism for ant colony optimization on gpus," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 42 – 51, 2013, metaheuristics on GPUs.

[10] "Parallel ant colony optimization on graphics processing units," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 52 – 61, 2013, metaheuristics on GPUs.

[11] A. Delévacq, P. Delisle, and M. Krajecki, *Parallel GPU Implementation of Iterated Local Search for the Travelling Salesman Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 372–377.

[12] Y. Zhou, F. He, and Y. Qiu, "Optimization of parallel iterated local search algorithms on graphics processing unit," *The Journal of Supercomputing*, vol. 72, no. 6, pp. 2394–2416, 2016.

[13] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[14] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "Approximate algorithms for the traveling salesperson problem," in *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, Oct 1974, pp. 33–42.

[15] G. Reinelt, "Tsplib a traveling salesman problem library," in *ORSA JOURNAL ON COMPUTING*, 1991.

[16] M. A. O. Neil and M. Burtscher, "Rethinking the Parallelization of Random-Restart Hill Climbing A Case Study in Optimizing a 2-Opt TSP Solver for GPU Execution," *20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 99–108, 2015.

[17] K. Rocki and R. Suda, "High performance GPU accelerated local optimization in TSP," *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013*, pp. 1788–1796, 2013.

[18] J. L. Bentley, "Experiments on traveling salesman heuristics," *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 91–99, 1990.