

Parallelized solution to the asymmetric travelling salesman problem using central processing unit acceleration

Akschat Arya¹, Boominathan Perumal², Santhi Krishnan³

¹BTech in Computer Science and Engineering, VIT University, Vellore, India

²Department of Information Security, VIT University, Vellore, India

³Department of Analytics, VIT University, Vellore, India

Article Info

Article history:

Received Aug 18, 2021

Revised Jan 11, 2022

Accepted Jan 24, 2022

Keywords:

Asymmetric travelling salesman problem
Dynamic programming
Held-karp algorithm
Multithreading
Parallelization

ABSTRACT

Travelling salesman problem is a well researched problem in computer science and has many practical applications. It is classified as a NP-hard problem as its exact solution can only be obtained in exponential time unless $P = NP$. There are different variants of the travelling salesman problem (TSP) and in this paper, asymmetric travelling salesman problem is addressed since this variant is quite often observed in real world scenarios. There are a number of heuristic approaches to this problem which provides approximate solutions in polynomial time, however this paper proposes an exact optimal solution which is accelerated with the help of multi-threading-based parallelization. In order to find the exact optimal solution, we have used the held-karp algorithm involving dynamic programming and to reduce the time taken to find the optimal path, we have used a multi-threaded approach to parallelize the processing of sub-problems by leveraging the central processing unit cores (CPUs). This method is an extension of a well researched solution to the TSP; however, this method shows that solutions to computationally intensive problems involving sub-problems such as the asymmetric travelling salesman problem (ATSP) can be accelerated with the help of modern CPUs.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Akschat Arya
BTech in Computer Science and Engineering, VIT University
Vellore, India
Email: akschatarya1@gmail.com

1. INTRODUCTION

In the simple traveling salesperson problem (TSP), we are given an undirected graph $G = (V, E)$ and $cost\ c(e) > 0$ for each edge $e \in E$ and the objective is to find a hamiltonian cycle with the minimum cost. A hamiltonian cycle visits every vertex in V exactly once. In this paper we are addressing the asymmetric travelling salesman problem (ATSP) which frequently has to be dealt with in real world scenarios. Let $M = (V, A)$ be a given directed graph, with vertex set $V = \{1, \dots, n\}$ and arc set $A = \{(i, j) : i, j \in V\}$. Let c_{ij} be the cost for the arc $(i, j) \in A$ with $c_{ii} = +\infty$ ($i \in V$). A hamiltonian circuit (tour) of G is a circuit visiting each vertex of V exactly once. The objective of the ATSP is to find a Hamiltonian circuit $M^* = (V, A^*)$ of M with minimum cost $= \sum_{(i,j) \in A^*} c_{ij}$

There are different variants of the travelling salesman problem which have been addressed by researchers earlier and both approximate (faster) and exact (slower) solutions have been provided. Some possible solutions for some of the other variants as per earlier research are as follows: i) symmetric TSP: GPU accelerated solution provided by Kimura *et al.* in [1], ii) ATSP: approximation algorithms by

decomposing directed regular multigraphs provided by Kaplan *et al.* in [2], iii) ATSP with windows: exact solution through a graph transformation provided by Albiach *et al.* in [3], iv) ATSP with replenishment arcs: polyhedral results provided by Mak and Boland in [4].

Meet in the middle algorithm was used by Kazuro Kimura *et al.* to accelerate the execution time but this method can only be used on the symmetric TSP by leveraging the symmetric aspect of the problem and thus Kimura *et al.* in [1] achieved an acceleration by a factor of 1.5 and that of 1.7 using man-in-the-middle (MITM) when n (*number of vertices*) was odd and even, respectively. Since this paper aims to address the asymmetric travelling salesman problem, we have not used MITM, instead we make use of the following techniques to accelerate the processing time: i) multi-threaded program to utilize central processing unit (CPU) cores, ii) thread-safe hashmap to store results of the dynamic cost function.

CPU parallelization has also been achieved for other algorithms like the ant colony optimization for the TSP. Ling *et al.* in [5] have presented an adaptive parallel ant colony optimization (PACO) algorithm using massively parallel processors (MPPs). A method of adjusting the time interval adaptively for information exchange according to the diversity of the solutions is also proposed by Chen ling *et al.* to avoid early convergence and improve the quality of results [5]. Fejzagić *et al.* have shown that it is possible to efficiently parallelize metaheuristic algorithms like ACO using task parallel library [6].

Gizems Ermis *et al.* have investigated the acceleration from CUDA by using 2-opt and 3-opt local search heuristics and shared explained some parallelization strategies to utilize GPU resources effectively [7]. Haim Kaplan *et al.* has provided approximation algorithms for asymmetric TSP by the decomposition of directed regular multigraphs [2]. Experiments by Saxena *et al.* in [8] show that parallelization tools like OpenMP and CUDA can significantly reduce the execution time for genetic algorithms used in solving the TSP. Rashid in [9] presented a parallel heuristic integrating a greedy approach into a genetic algorithm with local-search using GPU acceleration.

Most of the previous work have presented an approximate algorithm for the general TSP or an exact algorithm without CPU parallelization for the ASTP. In this paper we present an exact algorithm for the asymmetric TSP utilizing CPU parallelization and thread-safe hashmap to accelerate the execution process. Alrashdan *et al.* have used enhanced crossover operation using genetic algorithm with their probabilities in order to create an efficient method to provide a near optimal solution for the ATSP [10]. A Two-way parallel slime mold algorithm by flow and distance (TPSMA) is proposed by Liu *et al.* in [11] in order to solve slime mold algorithm's problem of poor local optimization. Ascheuer *et al.* has provided a computational study which has indicated that most ATSP with time windows instances ranging till 50–70 nodes can be optimally solved using branch and cut [12]. Kang *et al.* propose an effective method of constructive crossover such that large number of genes can be effectively evolved by exploiting the GPUs parallel computing power and an effective parallel approach to genetic TSP where crossover methods cannot be easily implemented in parallel fashion [13]. Vasilchikov has shown that the little algorithm also has good potential for recursive-parallel computations and can be used with a combined approach [14]. Sample instances for the TSP (and related problems) from various sources and of various types are provided by TSPLib in [15]. We have also made use of the datasets provided by TSPLib. Svensson *et al.* have provided a constant factor approximation algorithm by the reduction to subtour partition cover (an easier problem obtained when the general connectivity requirements are relaxed significantly into local connectivity conditions) [16]. Azimi *et al.* have presented a new model using simulated annealing with multiple transporters for the TSP [17]. A new hybrid algorithm for the probabilistic traveling salesman problem (PTSP) is proposed by Marinakis based on greedy randomized adaptive search procedure (GRASP), particle swarm optimization (PSO) and expanding neighborhood search (ENS) strategy [18].

Han *et al.* Have solved the large-scale colored travelling salesman problem using an improved ant colony optimization (IACO) algorithm in [19]. Ereemeev *et al.* have verified in [20] the usefulness of a parallel adaptive ant colony communities for the dynamic travelling salesman problem (DTSP). Ereemeev *et al.* have proposed a new memetic algorithm for the asymmetric travelling salesman problem (ATSP) with optimal recombination in [20]. Rashid and Mosteiro have provided a novel solution in [21] that integrates local-search heuristics, a greedy algorithm and a genetic algorithm. Odili *et al.* in [22] present a comparative performance analysis of some of the metaheuristic algorithms like the improved extremal optimization (IEO), african buffalo optimization algorithm (ABO), max-min ant system (MMAS), the heuristic randomized insertion algorithm (RAI) and cooperative genetic ant system (CGAS) to solve the ATSP. Fosin *et al.* have presented a new parallel iterated local search approach in [23] with 2-opt and 3-opt operators for symmetric TSP, using GPU acceleration. Li *et al.* have provided an improved multicore based parallel branch and bound algorithm to solve classic TSP with its shortcomings in [24]. Rico-Garcia *et al.* have provided a parallel implementation of the discrete teaching learning-based optimization algorithm (DTLBO) by utilizing a multicore GPU environment in order to improve the performance of the algorithm and to obtain suboptimal or optimal solutions to the traveling salesman problem [25]. However, most of these methods mentioned in

the aforementioned papers provide only an approximate solution or do not consider the asymmetric version of the TSP with parallelization. Our method has shown that the optimal solution of ATSP and similar computationally intensive problems with sub-problems can be accelerated with parallelization using modern CPUs.

2. METHOD

2.1. Theoretical analysis

We have used the Held-Karp algorithm on a dataset of n nodes to find an exact solution to this node set. Before Parallelizing the algorithm, we need to perform the theoretical analysis of the standard algorithm. The time and space complexity can be calculated as follows.

Time complexity: Let the given set of nodes be $V \in \{v_1, v_2, \dots, v_n\}$ with v_1 as the initial node. For every other node v_i such that $i \neq 1$, the aim is to find the minimum cost path with v_1 as the starting node, v_i as the ending node such that all other nodes are visited exactly once. For a set of size k, we consider k-2 subsets each of size k-1 such that all subsets don't have k^{th} in them.

Thus, by evaluating the sum of minimum cost path for each subset of k-1 nodes starting with the initial node we get the time complexity. This is given by (1):

$$n - 1 + \sum_{k=2}^{n-1} k(k - 1) \times \binom{n-1}{k} \tag{1}$$

Also the occurrences of computations for the next phase is given by (2):

$$\sum_{k=2}^{n-1} k = \frac{n(n-1)}{2} - 1 \tag{2}$$

thus from (1) and (2), (1) reduces to (3):

$$(n - 1)(n - 2)2^{n-3} + (n - 1) \tag{3}$$

on further reduction we get the time complexity as $O(2^n n^2)$

Space complexity: The Held-Karp algorithm is executed in exponential time but still offers relatively faster execution compared to exhaustive enumeration. This is compensated by using a lot more space than exhaustive enumeration. The space complexity is given by (4):

$$\begin{aligned} n - 1 + \sum_{k=2}^{n-1} k \times \binom{n-1}{k} \\ = (n - 1)2^{n-2} \end{aligned} \tag{4}$$

on reduction we get the space complexity as $O(2^n n)$.

2.2. Processing architecture

We have utilized CPU parallelization to achieve faster execution time. The architecture of a CPU with multiple cores is represented by Figure 1. A multi-core processor is a type of processor that contains multiple cores or processing units on the same chip. This kind of processor is different from a superscalar processor, which can issue multiple instructions per clock cycle from one instruction stream (thread) and contains multiple execution units. However, multiple instructions per clock cycle from multiple instruction streams is issued by a multi-core processor. Every core in a multi-core processor potentially can be superscalar too, implying that on each clock cycle, multiple instructions can be issued from a single thread by each core. Simultaneous multithreading (Intel's hyperthreading technology is an example) was an early form of pseudo-multi-core architecture. A processor capable of concurrent multithreading includes multiple execution units in the same processing unit, thus it can be said that it has a superscalar architecture and can issue more than one instruction per clock cycle from multiple threads. However, temporal multithreading can issue one instruction at a time from multiple thread where a single execution unit in the same processing unit is included.

2.3. Parallelization

Parallelization is achieved by mapping sub-problems created by the first recursive call to threads which will be running in parallel. This is illustrated in Figure 2 where $cost(i, N)$ is the cost to optimally visit all vertices in a set with N nodes starting from node i and $adj(i, j)$ is the cost to travel from node i to node j .

If there are k sub-problems created by the first recursive call and t threads mapped to them then each thread i will run x_i sub-problems such that,

$$x_i = \begin{cases} \frac{k}{t} & \text{if } k \text{ is divisible by } t \\ \lfloor \frac{k}{t} \rfloor & \text{if } k \text{ is not divisible by } t \text{ and } i \in (0, t - 1) \\ k - t & \text{if } k \text{ is not divisible by } t \text{ and } i = t \end{cases}$$

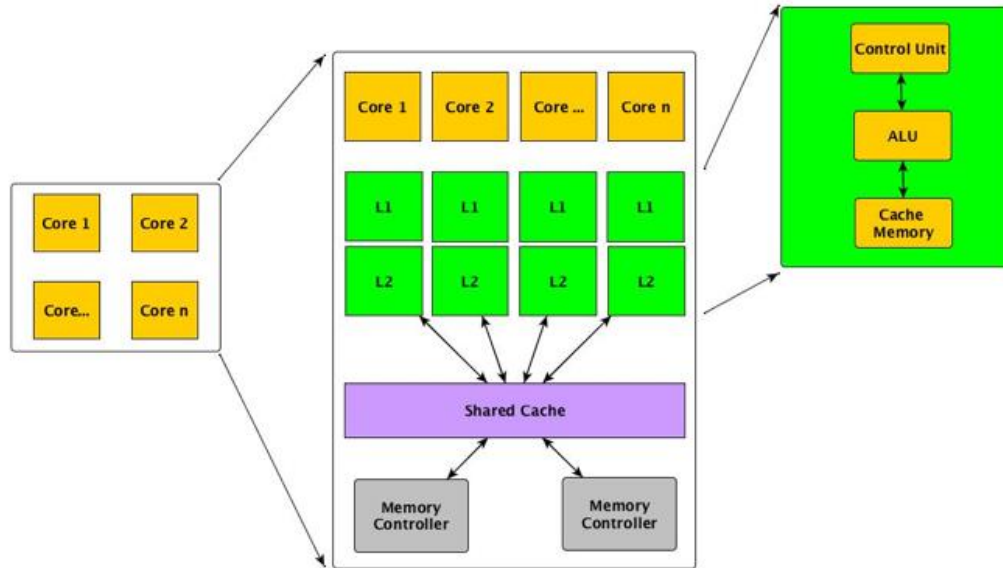


Figure 1. Multi core CPU architecture

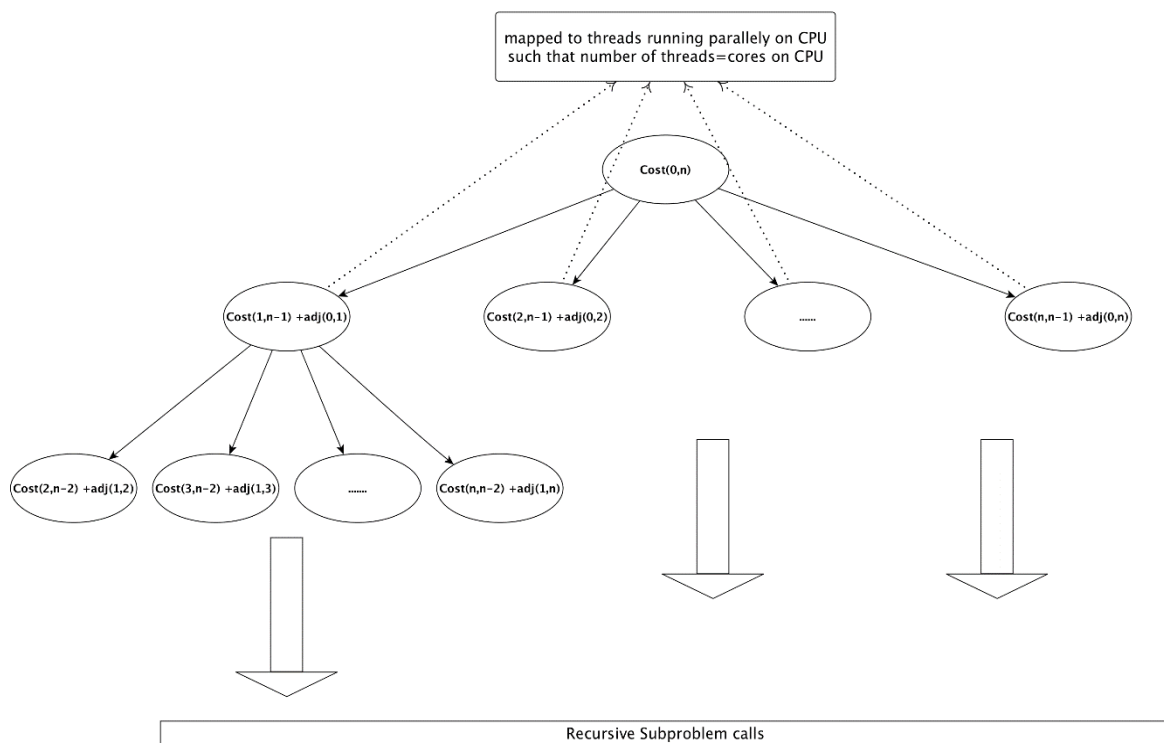


Figure 2. Thread mapping to recursive call

The challenge was to create a common thread-safe structure to cache the intermediary results from the recursive calls. This data structure should be shared with all the threads. For this purpose, we have used Java's ConcurrentHashMap with Java threads to achieve thread-safe parallelization. The hashmap creates an empty, new map with the specified initial capacity, concurrency and load factor level. The implementation of initial capacity performs internal sizing to accommodate these many elements whereas the implementation of concurrency tries to do the same. Initial concurrency level parameters and capacity parameter of ConcurrentHashMap constructor (or Object) in Java are set to 16 by default. As we are parallelizing the process with 9 threads we do not need to change the parameters. Thus, instead of using a map wide lock, ConcurrentHashMap maintains a list of locks by default such that the initial capacity is equal to the number of locks. Each lock is used to lock on a single bucket of the Map. This indicates that the number of threads which is set equal to the concurrency level specified in the parameter can modify the collection at the same time only if each thread works on different bucket. Hence, unlike hashtable, the operations like delete, create, update and read are done without locking on the entire map. Retrieval operations are usually not blocked, so they may overlap with operations involving updates. The entire architecture is illustrated by Figure 3.

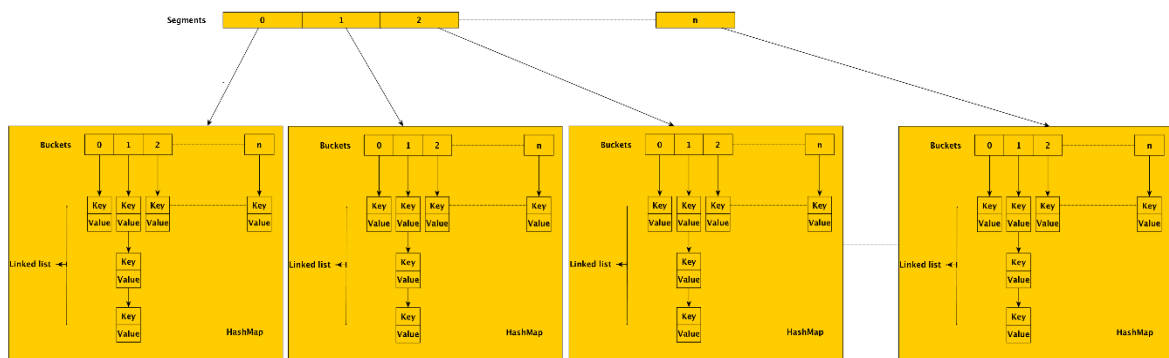


Figure 3. ConcurrentHashMap internal structure

Concurrency level constructor argument(optional) guides the allowed concurrency among operations involving updates, which is used as a hint for internal sizing. In order to permit the indicated number of concurrent updates without contention the table is partitioned internally. The actual concurrency will vary since hashtables in placements are random in nature. Algorithmically the process can be represented as the following Figure 4.

Algorithm 1: Parallelized Held-Karp Algorithm

```

0   Data: nodes set  $S$  with  $|S| = n$ , initial node  $v_0 \in S$ 
      weights from node  $v_i$  to  $v_j = W(v_i, v_j)$ 
      number of threads =  $t$ 
1   Result: A shortest tour that visits all locations in  $V$ 
2   Initialize threadsafe map  $T$  to store previous weights;
3   fun  $cost(S, v)$ :
4     if  $T$  contains key  $S, v$  then:
5       return  $T(S, v)$ ;
4     else if  $|S| = 1$  then:
5       return  $W(v, v_0)$ ;
6     else
7       Set  $v$  as a visited node;
8       for each node  $v_i$  in  $S$ :
9         if  $v_i$  is not visited:
10        SubCost  $\leftarrow \text{Min}(W(v, v_i) + \text{cost}(S - \{v\}, v_i), \text{SubCost})$ ;
11         $T(S, v) \leftarrow \text{SubCost}$ ;
12        Set  $v$  as a unvisited node;
13      return SubCost;
14  map  $cost(S, v_0) : cost(S, v_n)$  to thread 1  $\leftarrow C_1$ 
15  map  $cost(S, v_n) : cost(S, v_{2n/t})$  to thread 2  $\leftarrow C_2$ 
16  map till  $cost(S, v_{n(t-1)/t}) : cost(S, v_n)$  to thread  $t \leftarrow C_n$ 
17  return  $\text{Min}(C_1, C_2, \dots, C_n)$ 
    
```

Figure 4. Parallelized held-karp Algorithm

3. RESULTS AND DISCUSSION

Testing the algorithm with 17 nodes produces the shortest and the most optimal path with total distance = 39 as verified from TSPLib. Figure 5 captures the time of execution for solving the problem with the nodes n ranging from 18 to 22 from the 34-node dataset of TSPLib considering t threads(x-axis) running in parallel at a time.

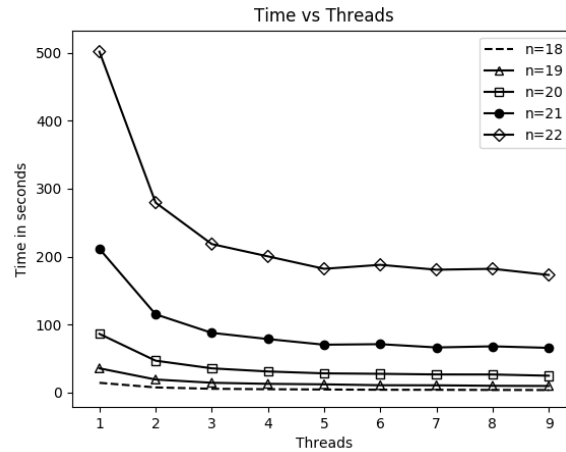


Figure 5. Time vs number of threads

From Figure 5 we can clearly see parallelization with more number of threads running in parallel has helped in reducing the execution time in each of the cases considering nodes n such that $18 \leq n \leq 22$. The same information from Figure 5 is represented in Table 1. For each node set of n nodes from the 34-node dataset the speed-up ratio such that $18 \leq n \leq 22$ is represented in Table 2.

Table 1. Performance in terms of time taken in seconds

Threads \ Nodes	Threads								
	1	2	3	4	5	6	7	8	9
18	14.151	7.375	5.392	4.72	4.327	3.929	3.844	3.643	3.526
19	35.468	18.996	14.222	12.629	11.912	10.594	10.496	9.76	9.516
20	86.207	46.732	35.457	30.829	28.06	27.388	26.471	26.391	24.601
21	211.379	115.009	87.717	78.567	70.265	71.04	66.224	67.849	65.61
22	501.693	279.428	218.44	200.299	182.02	187.85	180.67	182.016	172.911

Table 2. Speed-up ratio

Nodes	Speed-up Ratio
18	4.013
19	3.727
20	3.504
21	3.22
22	2.9





4. CONCLUSION

The experiment has successfully demonstrated that the proposed parallelized algorithm for solving the ATSP optimally helps in reducing the execution time compared to traditional Held karp algorithm and is a viable method to compute the optimal path for the ATSP. Although the computation time is higher than suboptimal methods, the proposed methodology gives the exact solution to the ATSP, which justifies the high computational time. Other optimal and suboptimal methods can incorporate CPU parallelization like the proposed methodology to produce even better results. In the future, hybrid algorithms can be used along with parallelization using GPU and CPU both to solve computationally intensive problems such as the ATSP.





REFERENCES

- [1] K. Kimura, S. Higa, M. Okita, and F. Ino, "Accelerating the held-KARP algorithm for the symmetric traveling salesman problem," *IEICE Transactions on Information and Systems*, vol. E102D, no. 12, pp. 2329–2340, Dec. 2019, doi: 10.1587/transinf.2019PAP0008.
- [2] H. Kaplan, M. Lewenstein, N. Shafirir, and M. Sviridenko, "Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs," in *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, vol. 2003-January, pp. 56–65, 2003, doi: 10.1109/SFCS.2003.1238181.
- [3] J. Albiach, J. M. Sanchis, and D. Soler, "An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation," *European Journal of Operational Research*, vol. 189, no. 3, pp. 789–802, Sep. 2008, doi: 10.1016/j.ejor.2006.09.099.
- [4] V. Mak and N. Boland, "Polyhedral results and exact algorithms for the asymmetric travelling salesman problem with replenishment arcs," *Discrete Applied Mathematics*, vol. 155, no. 16, pp. 2093–2110, Oct. 2007, doi: 10.1016/j.dam.2007.05.014.
- [5] L. Chen, H. Y. Sun, and S. Wang, "A parallel ant colony algorithm on massively parallel processors and its convergence analysis for the travelling salesman problem," *Information Sciences*, vol. 199, pp. 31–42, Sep. 2012, doi: 10.1016/j.ins.2012.02.055.
- [6] E. Fejzagic and A. Oputic, "Performance comparison of sequential and parallel execution of the Ant Colony Optimization algorithm for solving the traveling salesman problem," in *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2013 - Proceedings*, 2013, pp. 1301–1305.
- [7] G. Ermiş and B. Çatay, "Accelerating local search algorithms for the travelling salesman problem through the effective use of GPU," *Transportation Research Procedia*, vol. 22, pp. 409–418, 2017, doi: 10.1016/j.trpro.2017.03.012.
- [8] R. Saxena, M. Jain, S. Bhadri, and S. Khemka, "Parallelizing GA based heuristic approach for TSP over CUDA and OpenMP," in *2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017*, Sep. 2017, vol. 2017-January, pp. 1934–1939, doi: 10.1109/ICACCI.2017.8126128.
- [9] M. H. Rashid, "A GPU Accelerated parallel heuristic for travelling salesman problem," in *Proceedings - 2018 IEEE/ACIS 19th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2018*, Jun. 2018, pp. 82–86, doi: 10.1109/SNPD.2018.8441139.
- [10] W. K. Alrashdan, S. Abu Owida, and W. Alsharafat, "Solving Asymmetric Travelling Salesman Problem Using Group Constructive Crossover," *IJCSNS International Journal of Computer Science and Network Security*, vol. 17, no. 9, 2017.
- [11] M. Liu *et al.*, "A Slime Mold-Ant Colony Fusion Algorithm for Solving Traveling Salesman Problem," *IEEE Access*, vol. 8, pp. 202508–202521, 2020, doi: 10.1109/ACCESS.2020.3035584.
- [12] N. Ascheuer, M. Fischetti, and M. Grötschel, "Solving the Asymmetric Travelling Salesman Problem with Time Windows by branch-and-cut," *Mathematical Programming, Series B*, vol. 90, no. 3, pp. 475–506, May 2001, doi: 10.1007/PL00011432.
- [13] S. Kang, S. S. Kim, J. Won, and Y. M. Kang, "GPU-based parallel genetic approach to large-scale travelling salesman problem," *Journal of Supercomputing*, vol. 72, no. 11, pp. 4399–4414, May 2016, doi: 10.1007/s11227-016-1748-1.
- [14] V. V. Vasilchikov, "On Optimization and Parallelization of the Little Algorithm for Solving the Travelling Salesman Problem," *Automatic Control and Computer Sciences*, vol. 51, no. 7, pp. 551–557, Dec. 2017, doi: 10.3103/S0146411617070215.
- [15] G. Reinelt, "TSPLIB," *Universität Heidelberg Institut für Informatik*, 2013. <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>.
- [16] O. Svensson, J. Tarnawski, and L. A. Vêgh, "A Constant-factor Approximation Algorithm for the Asymmetric Traveling Salesman Problem," *Journal of the ACM*, vol. 67, no. 6, pp. 1–53, Nov. 2020, doi: 10.1145/3424306.
- [17] P. Azimi, R. Rooeinfar, and H. Pourvaziri, "A new hybrid parallel simulated annealing algorithm for travelling salesman problem with multiple transporters," *Journal of Optimization in Industrial Engineering*, vol. 15, pp. 1–13, 2014, Accessed: Jan. 24, 2022. [Online]. Available: www.SID.ir.
- [18] Y. Marinakis and M. Marinaki, "A Hybrid Multi-Swarm Particle Swarm Optimization algorithm for the Probabilistic Traveling Salesman Problem," *Computers and Operations Research*, vol. 37, no. 3, pp. 432–442, Mar. 2010, doi: 10.1016/j.cor.2009.03.004.
- [19] S. Han, M. Xu, Q. Lin, Q. Li, and Q. Guo, "An Improved Ant Colony Optimization for Large Scale Colored Traveling Salesman Problem," in *Proceedings - 2020 International Conference on Intelligent Computing, Automation and Systems, ICICAS 2020*, Dec. 2020, pp. 400–405, doi: 10.1109/ICICAS51530.2020.00090.
- [20] A. V. Eremeev and Y. V. Kovalenko, "A memetic algorithm with optimal recombination for the asymmetric travelling salesman problem," *Memetic Computing*, vol. 12, no. 1, pp. 23–36, Jul. 2019, doi: 10.1007/s12293-019-00291-4.
- [21] M. H. Rashid and M. A. Mosteiro, "A greedy-genetic local-search heuristic for the traveling salesman problem," in *Proceedings - 15th IEEE International Symposium on Parallel and Distributed Processing with Applications and 16th IEEE International Conference on Ubiquitous Computing and Communications, ISPA/IUCC 2017*, Dec. 2018, pp. 868–872, doi: 10.1109/ISPA/IUCC.2017.00132.
- [22] J. B. Odili, A. Noraziah, and M. Zarina, "A Comparative Performance Analysis of Computational Intelligence Techniques to Solve the Asymmetric Travelling Salesman Problem," *Computational Intelligence and Neuroscience*, vol. 2021, pp. 1–13, Apr. 2021, doi: 10.1155/2021/6625438.
- [23] J. Fosin, D. Davidović, and T. Carić, "GPU implementacija operatora lokalnog pretraživanja za simetričan Problem Trgovačkog Putnika," *Promet - Traffic - Traffico*, vol. 25, no. 3, pp. 225–234, Jun. 2013, doi: 10.7307/ptt.v25i3.300.
- [24] Y. Li, K. Ma, and J. Zhang, "An efficient multicore based parallel computing approach for TSP problems," in *Proceedings - 2013 9th International Conference on Semantics, Knowledge and Grids, SKG 2013*, Oct. 2013, pp. 98–104, doi: 10.1109/SKG.2013.41.
- [25] H. Rico-García, J. L. Sanchez-Romero, A. Jimeno-Morenilla, and H. Migallon-Gomis, "A parallel meta-heuristic approach to reduce vehicle travel time in smart cities," *Applied Sciences (Switzerland)*, vol. 11, no. 2, pp. 1–17, Jan. 2021, doi: 10.3390/app11020818.





BIOGRAPHIES OF AUTHORS

Mr. Akschat Arya     has obtained his BTech in Computer Science and Engineering from Vellore Institute of Technology, Vellore. He has received scholarship from VIT, Vellore for his performance in VITEEE exam. Currently, he is working as a Data Engineer and his research interest includes Machine learning and High-performance Computing. He can be contacted at email: akschatarya1@gmail.com.



Dr. Boominathan Perumal     has obtained his Ph.D. from Vellore Institute of Technology. He is currently designated as Associate Professor Grade 1 in School of Computer Science and Engineering, VIT Vellore, India. He is having a total of 16 years of teaching experience in computer science. His research interest includes cloud computing, optimization techniques and machine learning. Currently he holds a position of faculty coordinator for IEEE Computer society student chapter in VIT. He is a lifetime member of IEEE and Computer society of India. He can be contacted at email: boominathan.p@vit.ac.in.



Prof. Santhi Krishnan     has received her Ph.D. in Computer Science and Engineering from Pondicherry University, Puducherry, India. She has pursued her M.E in Computer Science and Engineering from Anna University, Chennai. She has received her M.Sc, in Computer Science from Bharathidasan University, Trichy, India. Currently, she is working as Associate Professor in the School of Computing Science and Engineering, VIT University, Vellore, India. She has authored many national and international journal papers and one book. Also, she has published many chapters in different books published by International publishers. She is a member of IEEE and she is holding membership in many professional bodies like CSI, ISTE, IEEE and IAENG. Her areas of research include Big data Analytics, Data Mining and Computational Intelligence. She can be contacted at email: santhikrishnan@vit.ac.in.