A parallel algorithm to find the exact solution of the travelling salesman problem

Mohammed W. Al-Neama[1], Iman Abdulwahab Ahmed[2], Salwa M. Ali[3,4]
[1]Education College for Girls, University of Mosul, Mosul, Iraq
[2]College of Islamic Science, University of Mosul, Mosul, Iraq
[3]Department of Computer Science, Unaizah College of Sciences and Arts, Qassim University, Qassim, Saudi Arabia
[4]Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt

Article Info

Article history:

Received Feb 4, 2023
Revised Mar 21, 2023
Accepted Apr 2, 2023

Keywords:

Branch-and-bound algorithm
Message passing interface
OpenMP
Parallel algorithm
Traveling salesman problem

ABSTRACT

The traveling salesman problem (TSP) is a problem in computer science that has been extensively studied and has a wide variety of real-world applications. It is considered an NP-hard issue since the only way to get a precise solution to it is to wait an exponentially long amount of time unless P=NP. Both accurate and heuristic algorithms exist to tackle this problem. The Branch-and-Bound algorithm (BnB) is often regarded as the most significant precise approach, despite the fact that it traverses, in the worst scenario, all potential tours in order to calculate the tour efficiently. This study proposed an efficient parallel algorithm to solve TSP by using a cluster multicore system. An exact algorithm will be used to get the optimal solution, and to expedite the search for the best path, we have employed a multi-threaded approach that capitalizes on the processing power of multiple CPU cores to concurrently process sub-problems.

Corresponding Author:

Mohammed W. Al-Neama
Education College for Girls, University of Mosul
Mosul, Iraq
Email: mweama@uomosul.edu.iq

1. INTRODUCTION

The traveling salesman problem (TSP) is a classic problem in combinatorial optimization. The goal of the TSP is to find the shortest tour of a given set of cities, visiting each city only once and returning to the starting point [1]. The TSP is a problem of finding a minimum Hamiltonian cycle on a completely directed graph with non-negative edge costs. A Hamiltonian cycle is a graph cycle that visits each node exactly once. In other words, the salesman must determine the cheapest path that he takes to make one stop in each city on a list of n cities, with the cost of travel from $city_i$ to $city_j$ being c_{ij}, before returning home [2]. The solution to the TSP can be used to optimize many real-world problems, such as route planning for delivery services and planning of sales routes [3], [4].

The TSP is well-known NP-complete mathematics problems, meaning there are no definite polynomial-time solutions. A graph illustrates the cities' locations and distances. Many studies, such as [1], [5]-[7] are working to solve the TSP and reduce its complexity to obtain an ideal solution. This paper proposes a parallel method for solving TSP using BnB.

However, a multi-core cluster computer environment [8] has gained popularity in recent years due to the performance limitations of single-node which has single-core settings. Complex calculation processing is shared by multiple processes, and parallel search should reduce calculation time. Scientific computing, data processing, engineering, military and government applications, and big data analytics employ cluster multi-core computers. It employed in applications that demand a lot of processing power or memory, or that need to be spread. Businesses and organizations employ multi-core cluster computers for high-performance computing.

The aim of this paper is to implement an exact method called branch-and-bound (BnB) algorithm for finding the minimum number of iterations (computing time) required to solve the standard TSP, and thus to determine the optimal route for the standard TSP that covers all the nodes by implementing on a cluster multicore system. This system provides the great computational capability and high-speed memory access to multi-terabyte data structures. This makes high performance computing (HPC) the preferred method for solving TSP. We conduct an experiment to verify the performance of this approach using TSP as the benchmark, and we explain how well it works.

2. LITERATURE REVIEW

In this section, previous and most relevant works to the research topic are reviewed. Many methods have been presented in previous studies to find the solution for TSP. Some of them are exact methods to find the optimal solution, and some of them are to find heuristic solutions.

This subsection presents the previous studies which are solve TSP using BnB. Land and Doig [9] proposed BnB in 1960 for mixed and pure integer linear programming. This method is also a good partial enumeration strategy for combinatorial data analysis. A BnB algorithm is an approach for finding the optimal solution to an optimization issue. The aim of BnB is to start with the associated TSP. The Hungarian Method will solve TSP. TSP's optimal solution provides a tour.

Balasubramanian and Grossmann [10] approach is a breakthrough. It solves an auxiliary TSP using the Hungarian algorithm. The studies in [11], [12] discuss a strategy for solving a TSP auxiliary problem. The researchers parallelized the TSP [13] to solve a random 80-vertex network on a multi-core system. The authors in [14] describe TSP applications that add online store warehouse delivery. In addition, genome assembly is a high-dimensional traveling salesman problem that was discussed in [15].

Parallel BnB tree traversal using a high-performance cluster and distributed memory is described in articles [16], [17]. Each node in the tree has offspring that are added to the task pool and distributed to nodes in the cluster. Cheang *et al.* [18], focused on parallelized heuristics and incomplete tree traversals.

Ebadinezhad [19] proposed a self-adaptive accountable care organization (ACO) with novel methods to enhance unclear convergence time and random algorithm choices. DEACO constantly adjusts ACO settings. This method uses grouping to choose the first city (start point) for the quickest trip. DEACO finds each cluster's lowest cost/shortest path. This exercise used TSPLIB data from MATLAB simulation with 10 TSP cases. The proposed method outperforms the traditional ACO in closure speed and search accuracy.

2.1. Standard TSP

The standard TSP is represented mathematically by the entire graph $G = (V, E)$, where set $V = 1, 2, \dots n$ denotes the set of cities correlating to the node, and each edge is weighted. As such, the d_{ij} is a representation of the separation between the vertices it links. To take a tour is to travel in a circle along a network of roads that link several different cities. A tour's total duration is simply the sum of its individual legs. To solve the problem in an n-city setting, Bodin *et al.* [20] provide the formula for a typical TSP. For the convenience of notation, let's assume that:

$$a_{ij} = \begin{cases} 1, & if\ the\ tour\ includes\ (i,j) \\ 0, & otherwise \end{cases}$$

where $d_{ij} = Distance\ between\ two\ nodes\ (i,j)$. The TSP model is given as (1).

$$Minimize\ f = \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} a_{ij}\ ; d_{ij} = \infty, \forall\ i = j \quad (1)$$

Subject to: $\sum_{i=0}^{n} a_{ij} = 1; j = 0,1,\dots,n\ ; \sum_{j=0}^{n} a_{ij} = 1; i = 0,1,\dots,n$ (2)

$$a_{ij} = (0,1);\ \forall\ i,j \in \{0,1,\dots,n\}\ ; A = (a_{ij}) \in X \quad (3)$$

The main objective (1) is to reduce the amount of travel time a salesman spends on the road. In other words, conditions (2) and (3) guarantee that each node is visited (entered and left) exactly once. Whether or not the arc i, j is included in the route is indicated by the variables a_{ij}, which are guaranteed to be integers by constraint (4). Sub-tour solutions that meet the assignment constraints are prohibited by the set S in condition (5). Figure 1 shows an example of a set (X) that contains $n = 15$ cities, and (W) is a set that represents the road between every two cities, implies that $d: W \to R$ be the cost of each road in W.

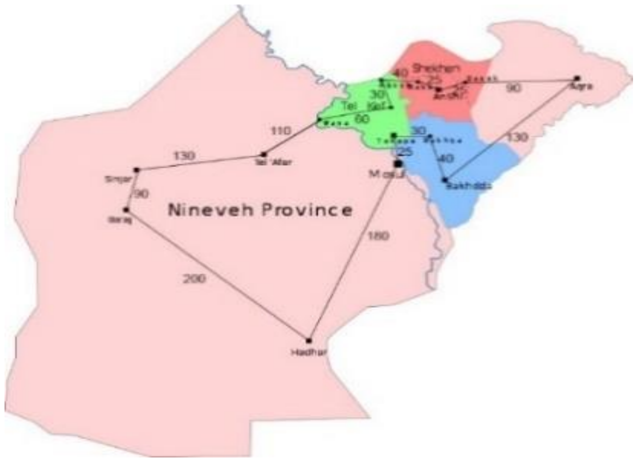

Figure 1. A tour of traveling salesman in 15 cities in Nineveh province including the cost of their roads

2.2. The complexity of the TSP

The complexity of the TSP lies in the fact that the number of possible routes grows exponentially as the number of cities increases. This means that even with sophisticated algorithms, a solution to the TSP is not guaranteed to be found in a reasonable amount of time. As a result, computer scientists have developed a variety of heuristic approaches to the problem, which are effective at finding approximate solutions in a reasonable amount of time [21].

The TSP is a hard problem, so, its complexity has motivated researchers to come up with efficient and effective algorithms to solve it. This has enabled organizations to use the TSP in their logistics and operations planning, making it a key tool in the world of industry. There are $(n-1)!$ possible Hamiltonian cycles if the seller is located in a graph with n cities and he needs to make at least $(n-1)$ visits to those cities [22].

3. PROPOSED METHOD

3.1. The parallel BnB algorithm to solve TSP

The parallel algorithm for the BnB method to solve the TSP is an effective way to reduce the complexity of the problem by using multiple processors to solve the problem in parallel. This algorithm is based on the concept of dividing the problem into smaller sub-problems that can be solved independently. The algorithm works by splitting the TSP into multiple sub-problems and assigning each sub-problem to a different processor. The sub-problems are then solved independently, and the results are combined to get the optimal solution. This approach allows the algorithm to work more efficiently and quickly than sequential algorithms and provides a much more efficient solution to the TSP.

Since the existing best value could not be improved upon to eliminate the unnecessary branches, several threads of a parallel program might explore them more thoroughly than a single thread of a sequential program. Similarly, we can achieve this result for clusters [23]. Finding a good cost function value for the initial best value for the exact algorithm could be done with the use of heuristic algorithms. The efficiency of parallelization is improved by using this method.

The search results are distributed among the threads using the suggested technique, which does not need the threads to communicate with one another. Instead, the results of the search are distributed by a hash table that is stored in shared memory. It is not appropriate to grant access to computing resources to threads that haven't yet found what they are looking for.

Each thread can run a separate branching and bounding procedure on the entire collection of potential solutions at once. The maximum lower limit value l_{max} and the global minimum upper limit u (best solution) are also kept in the shared memory along with the hash table. The last step is calculated using u and l_{max}. Information from the hash table can be read concurrently by all threads, but only one can write to it at a time. It is hoped that by doing so, the time spent on thread-to-thread communication and synchronization can be minimized. Furthermore, the search is stopped for all threads when one of the threads meets (1). Algorithm 1 will clarify the parallel implementation of the BnB algorithm for TSP using OpenMP involves dividing the search space among multiple processors and assigning each processor a subset of the partial solutions to explore. This parallelization can lead to significant speedup in the algorithm, especially for large TSP instances.

Algorithm 1. Parallel BnB TSP using OpenMP
```
    Input   : TSP instance
    Output  : Complete tour
1-  Initialize the lower bound on the cost of the optimal solution.
2-  Divide the search space among multiple processors using OpenMP parallelization
    directives.
3-  Assign each processor a subset of partial solutions to explore.
4-  Explore the partial solutions in parallel by branching on them and pruning those
    that are not optimal.
5-  Keep track of the best solution found so far and update the lower bound on the cost
    of the optimal solution.
6-  Repeat steps 2-5 until all partial solutions have been explored.
7-  Merge the results obtained by each processor to obtain the final solution.
8-  Load balancing: Load balancing is an important aspect of parallel branch-and- bound
    algorithms, since the size and complexity of each subtree may differ significantly.
    To achieve load balancing, you may need to dynamically adjust the number of threads
    assigned to each subtree based on its size and/or complexity.
```

3.2. Branch and bound for solving TSP

The BnB approach is a technique for finding a finite tree made up of possible solutions to the combinatorial optimization problem [24]. There are two parts to this strategy: branching and bounding. A solution set S that contains an admissible solution and a candidate solution can be partitioned into more than two subsets S_n by the branching operation. Recursively performing this step reduces the search space, getting it simpler to identify a solution that make the objective function $f(x)$ will be maximizes or minimizes. The branching action is meant to create a disconnected subset, while the bounding operation is used to truncate that subset. The process of pruning involves removing elements from a set to make it smaller. The size of the reduction is determined by the difference between the set's upper limit (u) and its lower limit $l(S_n)$.

If set S_n fulfills u_l for the minimization issue, then S_n may include candidates for solutions that could update u. Subsets that do not fulfill l_u are cut out. This means the algorithm will only continue with a subset of answers that may have superior options. Despite the branch and bound method being renowned for its accuracy, it can also be employed to obtain a reasonably good solution for practical purposes by halting the branching operation when certain constraints are satisfied.

The proposed approach conducts the branching operation by linking city i (randomly chosen as the branching city in the initial case) to city j from an unvisited city group ($j \in E$) to be selected subsequently. $l(S_i)$ can be determined through a single-tree relaxation [25], [26]. The upper limit u is obtained by applying the greedy algorithm to each subset and selecting the best value among all threads. In this study, the Lin-Kernighan method (LK method) solution [27] is utilized as the initial upper limit value. Since the aim of this study is to compute a satisfactory solution for practical use, a termination condition for the branch is established. These conditions are expressed as (4).

$$(u - l_{max})/(u + l_{max}) \leq \varepsilon \tag{4}$$

The left-hand side of this inequality denotes the discrepancy between the minimum upper limit value u and the maximum lower limit value l_{max}. In the proposed approach, every thread conducts an expensive branch and bound operation, and the resulting search outcome is stored in a shared hash table. The cost S_1 and the total cost M_1 of the chosen branches will serve as the keys to the shared hash table. So, we have:

$$M = \sum_{n=0}^{m} N_n(i,j) \tag{5}$$

here, S_1 is the cost being sought and M_n is the city chosen at a cost that is n times S_1. The hash table's value must be the smallest possible value found in the associated subset. The number of subsets grows exponentially with the number of cities, making it impractical to keep track of them all, as illustrated in Figure 2; consequently, results from the first location examined are given more priority Figure 2. Each thread, following a branch operation, verifies whether or not the chosen branch has been searched by looking it up in a hash table. If a hash table search has already been performed, the lower and higher bounds can be disregarded. If the range hasn't been explored yet, you can compute the minimum and maximum values and save the resulting hash in a common database. In Figure 2, the costs of the subsets' existence and the total cost S of the branches are combined with the lower limit values of the subsets S_1, S_3, and S_4 searched in P0 of Figure 2 to generate the hash table. Second, an effort is made to find a subset that is in a cost position. Since the minimum values for S_1 and S_3 are already stored in the hash table, you can just examine the result to complete the search without resorting to any further mathematics. That way, finding the most cost-effective subgroups is a quick process.

The process continues until either one of the threads satisfies the approximation requirement or all of the subsets have been explored, whichever occurs first, at a certain cost. To speed up the search, we adjusted the cost limit per thread. In (6) is the calculation for the upper cost bound.

$$d_{limit} = \lceil 2 \log N + 1 \rceil \tag{6}$$

Furthermore, the search process is expedited by avoiding the selection of a branch with a length $N(i,j)$ that surpasses the value D_{avg}, which is obtained by averaging the total distance D of the previously selected branches up to the depth d. Following is the criterion by which branches are chosen according to D_{avg}.

$$D_{avg} = D/d \tag{7}$$

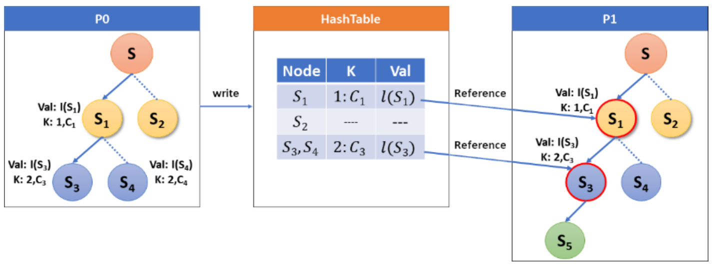

Figure 2. Scheduling computations on processing nodes, shows that P0 begin writing the hash table, while P1 is using the hash table as a reference

4. EVALUATION OF PERFORMANCE

The experiment setup for the implementation of the program traveling salesman problem would involve creating a specific set of cities and distances between them. The experiment would involve running the program with this data set and measuring the accuracy of the results. The accuracy would be measured by comparing the results of the program with the optimal solution for the given data set. Other factors that could be measured include the time taken for the program to run and the amount of memory it takes up. Additionally, the experiment could be repeated with different data sets to further test the accuracy of the program.

4.1. Setup of experiment

This subsection details the procedure and outcomes of the computational experiments used to assess the efficacy of the proposed approach. The program is written in C++, and OpenMP is used as a parallel implementation framework. The specifications of the platform are called Sun Microsystems cluster, provided by LinkSCEEM-2 systems at Bibliotheca Alexandrina, Egypt, with 130 nodes and 64 GB memory (each user is permitted 32 nodes), 8 GB RAM, 80 GB hard drive, and a twin port in fin band (10 Gbps), and Giga Ethernet Network interface; 64-bit Linux operating system. Programming language: C++; Compiler: GCC v. 6.3.0, compiler option, Parallel programming library: GNU OpenMP v. 6.3.0. There are various datasets available for the TSP, ranging from small-scale problems with a few cities to large-scale problems with thousands of cities. Table 1 lists the test's five different datasets from (TSPLIB) [28].

Table 1. the dataset used in the experiments

No.	Dataset name	Country name	No. of Cities
1	mu1979	Oman	1,979
2	eg7146	Egypt	7,146
3	ar9152	Argentina	9,152
4	fi10639	Finland	10,639
5	ho14473	Honduras	14,473

4.2. The run-time, speed-up, and efficiency

In this subsection, three common performance measurements, the run-time, the speed-up, and the efficiency, are used. The run-time of the implementation of the parallel algorithm is significantly faster than the run-time of the sequential algorithm. This is because the parallel algorithm is able to process multiple tasks simultaneously, thus reducing the total time it takes to complete a task. The speed-up of the implementation of the parallel algorithm is also impressive. The speed-up of the parallel algorithm is usually measured by the ratio of the execution time of the sequential algorithm to the execution time of the parallel algorithm. The speed-up of the parallel algorithm is usually much higher than the speed-up of the sequential algorithm, thus enabling a faster completion of tasks. The efficiency of the implementation of the parallel algorithm is also high. Efficiency is usually measured by the number of tasks that can be processed in a unit of time. The efficiency of the parallel algorithm is usually much higher than the efficiency of the sequential algorithm, thus making it more efficient in terms of resource utilization [29].

4.3. The performance evaluation of PA-TSP

The proposed algorithm of PA-TSP uses one core per thread and has been tested in five different environments with 8, 16, and 32 cores. The method's starting point, the proposed method's optimal endpoint, and the proposed method's calculation time are all evaluated. The method's computation time is not included in the proposed method's total computation time. When applied to (1). Each problem and each core number are tested five times, and the average is recorded. The average run-time by the number of cores is a crucial metric for evaluating the performance of a parallel TSP algorithm. It is affected by various factors, such as the size of the dataset, the number of cores used, and the efficiency of the algorithm. The outcomes are listed in the Table 2.

Table 2. Average run-time by number of cores in the PA-TSP algorithm for each dataset

No.	Dataset name	Final solution	Sequential	8 cores	16 cores	32 cores
1	mu1979	86,891	26.63	10.96	8.99	6.74
2	eg7146	172,386	121.37	31.28	25.65	19.24
3	ar9152	837,479	745.76	151.27	114.04	99.03
4	fi10639	520,527	860.10	167.66	137.48	103.11
5	ho14473	177,092	2,110.74	325.23	296.69	220.02

It is clear that the majority of the PA-TSP's issues are alleviated once more cores are added to the computation (cores 16–64 is faster than core 8). This demonstrates that, up to a certain number of cores, the proposed method searches for solutions at a faster rate. In light of this, the result of a core count of 32 is assumed to take less time to calculate than the other results. It is clear that all the dataset's core 8 have a much longer run-time than other core counts, but all results take less than several tens second to complete. It is thought that increasing the number of cores will prevent such long calculation times from happening since other experiments with the number of cores did not find such a big difference.

The system was able to calculate the optimal routes for the salesperson in a fraction of the time it would normally take on a single-core system. Additionally, the results were accurate and consistent across multiple simulations. The cluster multi-core system was able to utilize its resources efficiently and effectively, and was able to calculate the optimal routes for the salesperson in a fraction of the time it would normally take. The results of the parallel traveling salesman problem on a cluster multi-core system are a testament to the power of distributed computing, as shown in Figure 3.

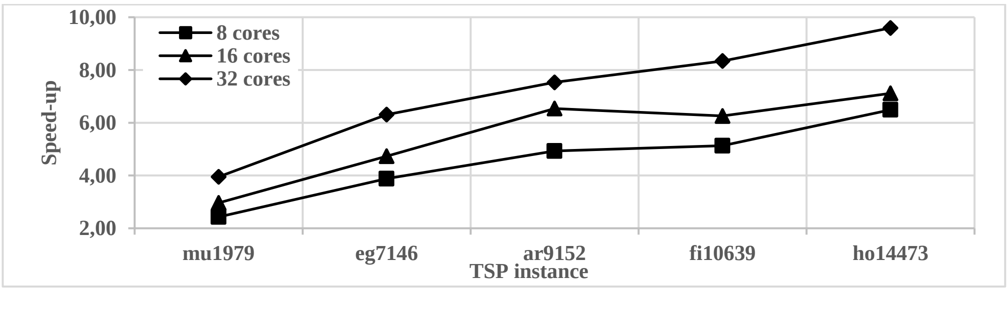

Figure 3. Speed-up comparisons between the sequential TSP algorithm and the proposed parallel TSP method using (8,16, and 32) cores

Furthermore, the use of multiple processors allowed for the system to make more efficient use of its resources, as any idle processor could be used to help in the processing of other tasks. This optimization in computing power meant that the system could solve complex problems more quickly and accurately, saving time and money. Overall, the results of the parallel traveling salesman problem implemented on a cluster multi-core system were a success and showed the potential of this powerful computing tool.

5. CONCLUSION

In this work, a parallel algorithm to solve the traveling salesman problem by using a branch and the bound was proposed. This technique efficiently utilizes the multi-core by performing a deepening iteration of the BnB algorithm in each thread and then sharing the search results among them via a hash table in shared memory. Experimental results corroborated the theory that increasing the core speeded-up the search process. Additionally, we showcased the potential effectiveness of the proposed method for large-scale problem instances, as the rate of speed improvement varied based on the problem instance size. To accurately assess the performance of the method for large-scale instances, future research should focus on verifying its performance with such instances, rather than solely relying on the problem instances examined in this study. Branch and bound techniques can enhance the precision of the best solution and accelerate the iteration process. Furthermore, they can optimize the algorithms utilized for determining the upper and lower bounds, as well as the method for selecting branches. Additionally, while the shared hash table's results are not modified once written, we posit that updating the hash table, such as overwriting its contents, could potentially yield performance improvements in cases where new results with the same key and superior outcomes are obtained. As we believe that the proposed method has applicability beyond just the TSP and can be utilized for other discrete optimization problems, we will also explore its performance on various combinatorial optimization problems. In conclusion, the implementation of the parallel algorithm results in faster run-time, higher speed-up, and higher efficiency than the implementation of the sequential algorithm. This makes it a great choice for processing multiple tasks at once, thus saving time and resources.

ACKNOWLEDGMENTS

The authors thank Bibliotheca Alexandria and University of Mosul.

REFERENCES

[1] W. B. Yahia, M. W. Al-Neama, and G. E. Arif, "PNACO: Parallel algorithm for neighbour joining hybridized with ant colony optimization on multi-core system," *Bulletin of the South Ural State University, Series: Mathematical Modelling, Programming and Computer Software*, vol. 13, no. 4, pp. 107–118, 2020, doi: 10.14529/mmp200409.
[2] W. B. Yahia, M. W. Al-Neama, G. E. Arif, and W. Yahia, "A Hybrid optimization algorithm of ant colony search and neighbour-joining method to solve the travelling salesman problem," *Advanced Mathematical Models & Applications*, vol. 5, no. 1, pp. 95–110, 2020.
[3] W. B. Yahia, G. E. Arif, M. W. Al-neama, and A. H. Ali, "Traveling salesman problem methods of solution survey," *International Journal of Psychosocial Rehabilitation*, vol. 24, no. 5, pp. 8565–8581, 2020, doi: 10.37200/IJPR/V24I5/PR2023807.
[4] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: applications, approaches and taxonomy," *Computer Science Review*, vol. 40, 2021, doi: 10.1016/j.cosrev.2021.100369.
[5] K. A. F. A. Samah, N. Sabri, R. Hamzah, R. Roslan, N. A. Mangshor, and A. A. M. Asri, "Brute force algorithm implementation for traveljoy travelling recommendation system," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 2, pp. 1042–1049, 2019, doi: 10.11591/ijeecs.v16.i2.pp1042-1049.
[6] Z. A. Ali, S. A. Rasheed, and N. No'man Ali, "An enhanced hybrid genetic algorithm for solving traveling salesman problem," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 18, no. 2, pp. 1035–1039, May 2020, doi: 10.11591/ijeecs.v18.i2.pp1035-1039.
[7] F. Chebihi, M. E. Riffi, A. Agharghor, S. C. B. Semlali, and A. Haily, "Improved chicken swarm optimization algorithm to solve the travelling salesman problem," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 12, no. 3, pp. 1054–1062, Dec. 2018, doi: 10.11591/ijeecs.v12.i3.pp1054-1062.
[8] F. Montagna, G. Tagliavini, D. Rossi, A. Garofalo, and L. Benini, "Streamlining the OpenMP Programming Model on Ultra-Low-Power Multi-core MCUs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12800 LNCS, 2021, pp. 167–182.
[9] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, vol. 28, no. 3, pp. 105–132, 2010, doi: 10.1007/978-3-540-68279-0_5.
[10] J. Balasubramanian and I. E. Grossmann, "A novel branch and bound algorithm for scheduling flowshop plants with uncertain processing times," *Computers and Chemical Engineering*, vol. 26, no. 1, pp. 41–57, 2002, doi: 10.1016/S0098-1354(01)00735-9.
[11] Z. Zhang, Z. Xu, S. Luan, X. Li, and Y. Sun, "Opposition-based ant colony optimization algorithm for the traveling salesman problem," *Mathematics*, vol. 8, no. 10, 2020, doi: 10.3390/MATH8101650.
[12] I. M. Ali, D. Essam, and K. Kasmarik, "A novel design of differential evolution for solving discrete traveling salesman problems," *Swarm and Evolutionary Computation*, vol. 52, 2020, doi: 10.1016/j.swevo.2019.100607.
[13] J. Lauri, S. Dutta, M. Grassia, and D. Ajwani, "Learning fine-grained search space pruning and heuristics for combinatorial optimization," *arXiv preprints*, 2020, [Online]. Available: http://arxiv.org/abs/2001.01230.
[14] Y. Chen, Z. Jia, X. Ai, D. Yang, and J. Yu, "A modified two-part wolf pack search algorithm for the multiple traveling salesmen problem," *Applied Soft Computing Journal*, vol. 61, pp. 714–725, 2017, doi: 10.1016/j.asoc.2017.08.041.
[15] H. Chandran, M. Meena, and K. Sharma, "Microbial biodiversity and bioremediation assessment through omics approaches," *Frontiers in Environmental Chemistry*, vol. 1, Sep. 2020, doi: 10.3389/fenvc.2020.570326.
[16] A. Ignatov and A. Gorchakov, "Tool for simulating branch and bound computations," *Open Computer Science*, vol. 10, no. 1, pp. 112–116, May 2020, doi: 10.1515/comp-2020-0115.
[17] B. Steinberg, A. Baglij, V. Petrenko, V. Burkhovetskiy, O. Steinberg, and E. Metelica, "An analyzer for program parallelization and optimization," in *Proceedings of the 3rd International Conference on Applications in Information Technology*, Nov. 2018, pp. 90–95, doi: 10.1145/3274856.3274875.

[18] B. Cheang, X. Gao, A. Lim, H. Qin, and W. Zhu, “Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints,” *European Journal of Operational Research*, vol. 223, no. 1, pp. 60–75, 2012, doi: 10.1016/j.ejor.2012.06.019.

[19] S. Ebadinezhad, “DEACO: Adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem,” *Engineering Applications of Artificial Intelligence*, vol. 92, 2020, doi: 10.1016/j.engappai.2020.103649.

[20] L. Bodin, B. Golden, A. Assad, and M. Ball, *the State of the Art in the Routing and Scheduling of Vehicles and Crews*. 1981.

[21] M. Khachay and K. Neznakhina, “Complexity and approximability of the Euclidean generalized traveling salesman problem in grid clusters,” *Annals of Mathematics and Artificial Intelligence*, vol. 88, no. 1–3, pp. 53–69, 2020, doi: 10.1007/s10472-019-09626-w.

[22] M. De Berg, K. Buchin, B. M. P. Jansen, and G. Woeginger, “Fine-grained complexity analysis of two classic TSP variants,” *ACM Transactions on Algorithms*, vol. 17, no. 1, pp. 1–29, Jan. 2021, doi: 10.1145/3414845.

[23] Q. Shu, T. Rötzer, A. Detter, and F. Ludwig, “Tree information modeling: a data exchange platform for tree design and management,” *Forests*, vol. 13, no. 11, 2022, doi: 10.3390/f13111955.

[24] H. Tahami and H. Fakhravar, “Literature review on combining heuristics and exact algorithms in combinatorial optimization,” *European Journal of Information Technologies and Computer Science*, vol. 2, no. 2, pp. 6–12, 2022, doi: 10.24018/compute.2022.2.2.50.

[25] A. Gharehgozli, C. Xu, and W. Zhang, “High multiplicity asymmetric traveling salesman problem with feedback vertex set and its application to storage/retrieval system,” *European Journal of Operational Research*, vol. 289, no. 2, pp. 495–507, 2021, doi: 10.1016/j.ejor.2020.07.038.

[26] V. Traub and J. Vygen, “An improved upper bound on the integrality ratio for the s–t-path TSP,” *Operations Research Letters*, vol. 47, no. 3, pp. 225–228, 2019, doi: 10.1016/j.orl.2019.02.005.

[27] S. Lin and B.W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973, [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/opre.21.2.498.

[28] University Waterloo, “National traveling salesman problems,” *TSP Test Data*, 2017. http://www.math.uwaterloo.ca/tsp/world/countries.html (Accessed Mar. 13, 2023).

[29] A. A. AbdulRazzaq, Q. S. Hamad, and A. M. Taha, “Parallel implementation of maximum-shift algorithm using OpenMp,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 3, pp. 1529–1539, Jun. 2021, doi: 10.11591/ijeecs.v22.i3.pp1529-1539.

BIOGRAPHIES OF AUTHORS

Mohammed Wajid Al-Neama he has a Ph.D. in computing mathematics from Al-Azhar University, Cairo (Egypt) in 2014. He currently works at the Education College for Girls, Mosul University. His research area includes bioinformatics and parallel computing. He can be contacted at email: mwneama@uomosul.edu.iq.

Iman Abdulwahab Ahmed she has obtained an MSc. in Mathematical from the University of Mosul. She is currently working at the Islamic Science College, University of Mosul. Her research area includes computing mathematics and numerical analysis. She can be contacted at email: ahmediman1963@gmail.com, i.a.a@uomosul.edu.iq.

Salwa M. Ali she has obtained the Ph.D. in computer science from Ain Shams University, Cairo (Egypt). She is currently worked at Unaizah College of Sciences and Arts, Qassim University, (K.S.A.). Her research area includes information science foundation and lambda calculus. She can be contacted at email: s.mussa@qu.edu.sa.