Journal of Aeronautics and Space Technologies

15(2) (2022) 108-124

Journal Homepage: jast.hho.msu.edu.tr

RESEARCH ARTICLE

Data Collection from Wireless Sensor Networks: OpenMP Application on the Solution of Traveling Salesman Problem with Parallel Genetic Algorithm and Ant Colony Algorithm

Kablosuz Sensör Ağlarından Veri Toplama: Gezen Postacı Probleminin Paralel Genetik Algoritma ve Paralel Karınca Kolonisi ile Çözümü Üzerine OpenMP Uygulaması

Reşat Buğra Erkartal[1]*, Ömer Çetin[2], Atınç Yılmaz[1]

[1,] *Beykent University, Department of Computer Engineering, Faculty of Engineering and Architecture, 34485 İstanbul, Turkey,*

[2] *National Defence University , Hezârfen ASTIN, Department of Computer Engineering, 34149 Istanbul, Turkey*

Received: *April 13, 2022* ***Revised:*** *June 14, 2022* ***Accepted:*** *July 18, 2022*

Abstract

Parallelization of algorithms can reduce time in many cases while using multiple cores at the same time. Although Algorithms such as Genetic Algorithm (GA) and Ant Colony (AC) are widely used optimization algorithms to solve the non-linear problems it is usually time consuming. This study aims to solve a well-known NP-Hard problem, The Travelling Salesman Problem (TSP), by using both parallel and serial GA and AC. As an application, the data collected from the wireless sensors networks (WSNs) were used and the performance values of the running algorithms were compared. Reducing the travelling time in WSNs avoids losses in energy consumption caused by multi-tab transmission, but causes a long delay. Additionally, application was made with Open Multi-Processing (OpenMP) and its performance was compared with serial programming. According to the findings while both methods reduces the time in half when they run parallel, the performance of GA is much superior than AC.

Keywords: Genetic Algorithm, Travelling Salesman Problem, Parallel Programming

Özet

Algoritmaların paralelleştirilmesi, birçok durumda aynı anda birden fazla çekirdek kullanırken zamanı azaltabilir. Genetik Algoritma (Genetic Algorithm-GA) ve Karınca Kolonisi (Ant Colony- AC) gibi algoritmalar doğrusal olmayan problemleri çözmek için yaygın olarak kullanılan optimizasyon algoritmaları olmasına rağmen, genellikle zaman alıcıdır. Bu çalışma, iyi bilinen bir NP-Zor problemi olan gezgin satıcı problemini (The Travelling Salesman Problem-TSP) hem paralel hem de seri GA ve AC kullanarak çözmeyi amaçlamaktadır. Uygulama olarak kablosuz sensör ağlarından (Wireless sensor Networks-WSNs) toplanan veriler kullanılmış ve çalışan algoritmaların performans değerleri karşılaştırılmıştır. WSN'lerde seyahat süresinin azaltılması, çok sekmeli iletimin neden olduğu enerji tüketimindeki kayıpları önler, ancak uzun bir gecikmeye neden olur. Ayrıca OpenMP (Open Multi-Processing) ile uygulama yapılmış ve seri programlama ile performansı karşılaştırılmıştır. Elde edilen bulgulara göre her iki yöntem de paralel çalıştığında süreyi yarı yarıya azaltırken; GA'nın performansı AC'den çok daha üstündür.

Anahtar Kelimeler: Genetik Algoritma, Gezen Postacı Problemi, Paralel Programlama

*Corresponding Author
E-mail: bugraerkartal@beykent.edu.tr

1. INTRODUCTION

The number of sensor nodes in mobile networks can be expressed in hundreds, and the geographical point they are located can cover a very large area. In order to collect data from these nodes, it is necessary to visit all nodes one by one and obtain the required data. However, the problem of navigating the nodes in which order is one of the most difficult difficulties encountered. If the nodes are visited in a non-optimal order, the will be wasted and both opportunity and operational costs will increase. In this case, the data may be accessed delayed or not reached at all. In the literature, this type of problem is called "The Traveling Salesman Problem" (TSP) [1]. In this problem, when a postman is given a list of certain destinations, he is asked to return to the starting point using the shortest path. Such situations are a common and encountered problem in the fields of computer science and operations research [2]. At the beginning of the problems encountered here, the complexity of the problem increases as the number of locations to be visited increases.

TSP is a combinatorial optimization problem with decision variables known as NP-hard. In complexity theory, NP is the name given to any problem that can be solved by a non-deterministic polynomial time algorithm. If NP problems cannot be solved in deterministic polynomial time, the problem is considered "hard"[3]. There is no NP-hard "quick" solution in TSP, and the complexity of computations to find the best route will increase exponentially as more nodes are added to the problem. While the most direct solution seems to be to try all combinations and see which route is the shortest, the run time for this approach can be calculated by the factorial (n) of the number of nodes to be visited (O(n!)). Therefore, when the number of nodes to be visited exceeds 20, the solution of the problem becomes impossible [4].

In order to overcome this complexity, there are many heuristics and meta-heuristics in the literature [5, 6]. Methods such as genetic algorithm, ant colony, simulated annealing algorithm, Lin–Kernighan heuristics are the methods that are frequently used in solving TSP problem [7-9]. In this study, genetic algorithm and ant colony algorithm were used; Open Multi-Processing (OpenMP) supported parallel programming approach is used to increase real-time computation performance and efficiency of the method.

1.1. The Travelling Salesman Problem

TSP is a classical combinatorial optimization problem that is simple to specify but very difficult to solve [10]. There are points on a two-dimensional space that the postman must visit. The aim is to stop by these points only once and complete the tour in the shortest distance. The mathematical formulation of the problem is shown in equations (1), (2), (3) and (4).

Objective Function:

$$min \sum_{i} \sum_{j} d_{ij} x_{ij} \tag{1}$$

$$s.t. \sum_{j} x_{ij} = 1 \qquad \forall i = 1, \dots, N \tag{2}$$

$$\sum_{i} x_{ij} = 1 \qquad \forall j = 1, \dots, N \tag{3}$$

$$x_{ij} \in 0 \; veya \; 1 \tag{4}$$

Here N is the number of nodes, dij is the distance from node i to node j, and xij is the decision variables. The constraints are written to ensure that each node is visited only once.

1.2. Genetic Algorithm

Genetic Algorithm is an optimization algorithm that is revealed by simulating rational behavior in nature. The definition of optimization is to ensure the survival of the chromosome that produces the best result according to the cost function throughout the iterations to reach the best solution. The equivalents of the terms used in classical optimization in the Genetic Algorithm are given in Table 1.

Table 1. The equivalents of the terms used in classical optimization in the Genetic Algorithm[11].

Combinatorial Optimization	Genetic Algorithm
Feasible Solution	Chromosome
Optimal Solution	Proccessed Chromosome
Solution Set	Population
Objective Function	Fitness Function

Genetic Algorithms (GA) is a powerful technique inspired by biology. From the TSP perspective, each chromosome is the order of nodes to be visited. In general, GAs use selection, mutation, and crossover to generate new search points in a fixed-sized set [12]. The algorithm has five main stages. In first two stages after the initial population is generated, the fitness values of all individuals in the population is calculated. According to the fitness values of the individuals, the best individuals from the population are selected for the next step. At this point, how many individuals will be selected is completely under the control of the user. After the selection process, crossover and mutation are done between the selected individuals to produce the next generation. In the last step, if the new individuals obtained are optimal, the algorithm is terminated. Otherwise, all processes will be applied recursively to the next generation. In its simplest form, it can be summarized as follows.

Step 1: A random initial population of chromosomes is generated.

Step 2: Calculate the fitness function values to select the appropriate chromosomes.

Step 3: Apply the crossover and mutation operators sequentially.

Step 4: If the stopping criterion is met, it is stopped and the best answer is the chromosome with the best fitness function value. If not:
Step 5: Go to step 2.

While GAs are very effective and practical way for solving many problems, run times are the deciding factor in large scale problems since every fitness function values should be calculated for every member in the solution set. Fortunately, this process should be done independently for each individual this makes the problem a suitable candidate for parallel programming. The reason for this is that it is possible to evaluate each step separately as independent processes.

The advantages of GA can be listed as follows:

- No assumptions are required.
- It can optimize multiple objective functions simultaneously.
- Easy to parallelize.
- Since it is a probabilistic model, it is easier to apply to most problems than other models.

Parallel computing refers to the technique of dividing large troubles into smaller, independent, frequently comparable elements that may be accomplished concurrently through multiple processors interacting thru a shared memory. Process synchronization performs an crucial position in retaining the consistency of shared data.

A “race situation” can occur between jobs when dividing or allocating threads to cores [13]. A race condition occurs when multiple processes, at least one of which is performing a write operation, are running simultaneously in the shared memory space. This can result in deadly deadlock and starvation as well as inconsistent data readings. To overcome the problem, a critical zone should be created and only one of the competing jobs that can exist in this zone at a time should be allowed to enter. Both software and hardware solutions are available to solve the critical zone issue. The mutual exclusion approach, which is widely used in the literature, is the semaphore approach, which can be realized with both software and hardware support. The main difference that distinguishes the semaphore approach from the classical approaches is that processes perform uninterrupted operations wait() and signal() to indicate whether they have received or freed the resource, the mutex is a locking mechanism and the process must unlock if it wants to obtain the resource [14].

Because GAs perform probabilistic search on a set of solutions, it is possible to distribute the solution across multiple threads over multiple cores using a parallel programming approach to speed up runtime. The advantages of paralleling GA are not limited to increasing the computational speed. In addition, since the population is distributed, the quality of the solution also increases in this process [12]. This situation was evaluated by comparing the values taken by the fitness function in the application section.

There are three basic versions of Parallel Genetic Algorithm (PGA) in the sources. These; the master-slave model is the static migrating sub-populations model and the static intersecting sub-populations model [15]. The schematic representation of the three models is shown in Figure 1.

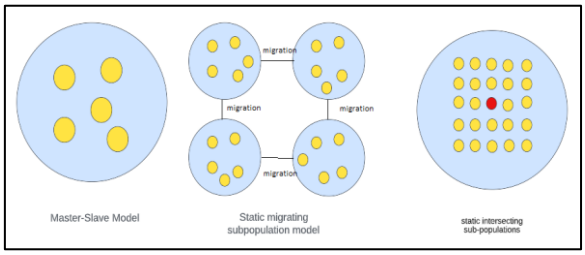

Figure 1. Schematic representation of PGA model.

The basic idea behind the algorithms above is the divide and conquer approach. In other words, it is to solve the task by breaking it down into parts and dividing those parts into multiple processors. In the master-slave version, there's a single population and the corresponding values of the fitness function are calculated individually among numerous processors. In the cellular version, the population is pre organized into smaller groups where individuals can migrate from one sub-population to the other only nearby boundary area, however the decision areas overlap and permit a few interplay among all people. This version is appropriate for extraordinarily parallel computers. In the island version, the populace is split into a couple of clusters, every of that is known as an island. On one island, people mate freely, at the same time as on different islands mating is prohibited. Then, a brand new operator, known as the migration operator, is delivered into the island version, a small part of the populace is periodically migrated among the islands in an effort to carry new genetic cloth to every island [15].

1.3. Ant Colony Algorithm

One of the important features of ants is their ability to go to the target point without using their eyesight. If the path they use to reach their destination changes for any reason, the ants leave a trail called pheromone on the path they have just discovered, and guide other ants that follow.

However, a decision has to be made at the crossroads. Although this decision is made randomly at first, the optimal route will be updated since the number of ants that will pass through the shorter alternative will be higher in the other alternative. Since the algorithm is inspired by the behavior of ants from their colonies, the algorithm is called the Ant Colonies algorithm (AC). Figure 2 shows the flow diagram of the algorithm.

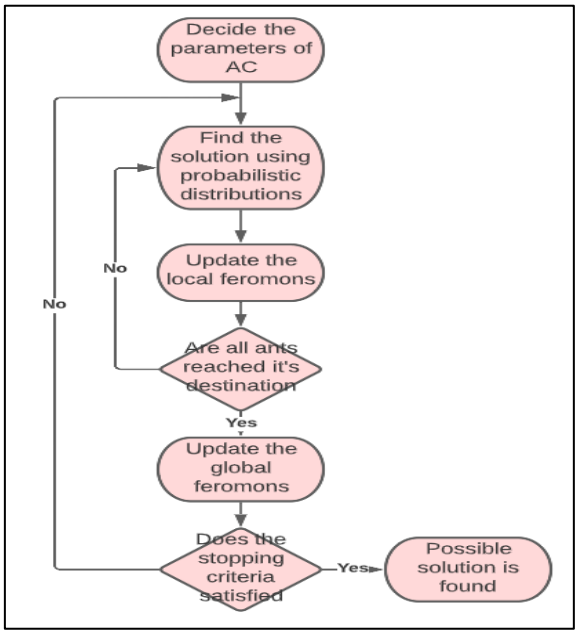

Figure 2. Flowchart of AC algorithm

2. LITERATURE SURVEY

Sallabi and EI-Haddad propose a genetic algorithm model with self-described crossover, population selection, multiple mutation, partial local optimal mutation operation and rearrangement operators to solve TSP [14]. The new crossover method in their proposed model uses Swapped Inverted Crossover (SIC) to produce better laps. The method used in population selection is used to find the maximum cost value of all neighboring cities in the tour. In the multiple mutation process, the selected individuals are replicated several times, and each of these selected copies is mutated using different neighbors.

Dwivedi et al. He proposes a new operator for the crossover operation, which is one of the important stages of TSP, which they call Sequential Cunstructive Crossover (SCX). SCX uses the best chromosomes of the parental structure. The method used in the study was compared with other crossover operators under the same conditions and it was proven that SCX gave a better solution [16].

Rani and Kumar compared two methods called roulette selection and Stochastic Universal Selection (SUS) used in population selection of genetic algorithm with each other. They found that the SUS selection method gives good results when the population size is small, while the roulette method works well in smaller populations [17].

Khan and Hanif's work is to propose a new binary chromosome representation method and fitness function. Their results show that chromosomes are closer to the optimal response when they are represented by binary numbers [18].

Pandhare et al. In their study, they parallelized the genetic algorithm using Hadoop Map/Reduce. In addition, they were compared with each other using three different crossover methods. It has been seen that parallelization has a significant contribution to

time saving, and no big difference has been detected between the crossover methods used [19].

Gupta and Panwar compared crossover methods in their studies. The methods compared are Sequential Constructive Crossover (SCX), Generalized N-point Crossover (GNX), and Edge Recombination Crossover (ERX). As a result of the study, it was determined that the SCX method was the most effective method in the solution of TSP [20].

Er and Erdogan used Parallel Genetic Algorithm (PGA) for the solution of TSP. Compared with MapReduce parallel genetic algorithm and sequential genetic algorithm for 171 cities, it was revealed that MapReduce GA found better solutions and took shorter time than Serial Genetic Algorithm (SGA) when the problem size increased [21].

Abbasi et.al. used both AC and GA in order to solve a vehicle routing problem. According to their findings, the highest speedup of the parallel algorithm on the GPU, the dual–core processor, and the quad-core processor is 13.73, 1.99, and 3.99, respectively [22].

Sieminski and Kopel proposed a method of parallel adaptive AC algorithm to solve dynamic TSP. In the paper they implement asynchronous and synchronous models using MapReduce process. Accoring to them the distance matrix can be improved to increase the efficiency of AC algorithm [23].

Yelmewad and Talawar used Parallel iterative hill climbing algorithm to solve TSP on Graphical Processing Unit (GPU). The number of cities at their database is 85900 and their GPU implementation gives up to 193 speed boost [24].

Sena et al. applied the genetic algorithm method as a traveling salesman problem on the workstation in 2022. In the study, tests were performed with clusters created with 1, 2, 4, 8 and 16 workstations using different population sizes. The performance of the method was optimized with the variation of the mutation rate and mutation range apart from the population size, and the results were compared [25].

Placido et al. applied the genetic algorithm method for the diagnostic discovery of solar panels in 2022. The study focuses on the effect of which metrics on the difficulty of solving the problem for the solution of the traveling salesman problem. In addition, a genetic algorithm model with two local searches is proposed to solve the problem. As a result, 32 new best solutions were obtained from 62 samples [26].

Kanna et al., in their study in 2021, proposed a hybrid optimization algorithm for modeling the large-scale traveling salesman problem. The proposed hybrid model Earthworm-based Deer Hunting Optimization Algorithm is formed by integrating two metaheuristic methods. As a result, the proposed model enabled the solution of TSP to be solved with less complexity [27].

Krishna et al. used rider optimization and spotted hyena optimization algorithms together to solve the TSP in 2021. The best solution aimed in the study is to minimize the distance to be covered by the vendor determined according to the proposed hybrid algorithm. As a result, the results obtained were compared with other algorithms and it was shown that the proposed method offered a better solution; the feasibility of the proposed hybrid algorithm has been demonstrated [28].

3. EXPERIMENTAL STUDIES

3.1. Genetic Algorithm

The steps and model parameters used in this study are explained in this section. The selection of the initial population, which is the first step of the genetic algorithm, was made randomly. Since the algorithm must return to the starting point, the length of the first generated chromosome is one more than the number of nodes selected.

In calculating the fitness value, which is the second step, briefly the minimum of the sum of the distance between the nodes was taken.

In the next step, in the selection of parents and children, in the selection of individuals to be included in the next population, the best individuals were included in the new population, while the worst individuals were removed.

One-point crossover was used for the crossover process, which is one of the most important steps of the genetic algorithm. In a single-point crossover, a point is determined among the selected parental chromosomes and exchanged. For example, if we take E1=[10101110] and E2=[11001100] and apply crossover after the 4th gen, the new individuals will be formed as T1=[10101100] and T2=[11001110].

Finally, a random gene was selected within the chromosomes for the mutation function.

In summary, the parameters of my model used in this study are as follows.

- Fitness Function: Shortest distance.
- Selection: Tournament Selection Method.
- Crossover: Single point crossover.
- Mutation: The replacement of two random genes.
- Crossover Ratio= 90%.
- Mutation Rate= 10%.

The flow chart of the model is shown in Figure 3. By using the flow chart in Figure 3, the serial and parallel structures of the program were created for 5, 10, 15, 20, 25 and 30 nodes, and the running times and fitness function values were compared.

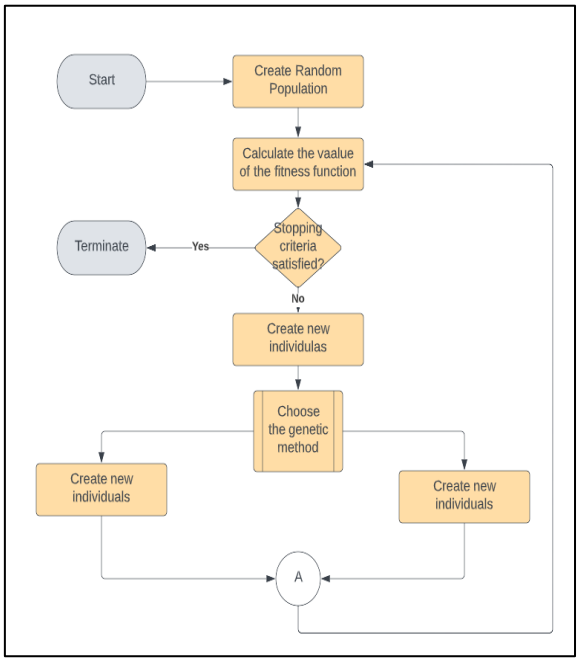

Figure 3. Genetic Algorithm's Flowchart.

3.2. Ant Colony Algorithm

The movement of the artificial ant depends on the amount of pheromone on the graphics edges. The probability of a virtual ant's transition from node i to node k is given in equation 5.

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha} + \eta_{ij}^{\beta}}{\sum(\tau_{iu}^{\alpha} + \eta_{iu}^{\beta})} \tag{5}$$

Here, p_{ij}^{k} (i,u) is the probability of choosing u alternative path options from i point of ant k. The amount of phenomenon found from point i to point j is represented as τ_{ij} and the inverse of the distance on the path from i to j is represented as η_{ij} (i,j). α and β are the parameters that determine the relative importance of the distance in updating the pheromone.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t+1) \tag{6}$$

$$\Delta\tau_{ij}^{k}(t+1) = \begin{cases} \frac{1}{L^{k}}(t+1), & (i,j) if\ used \\ 0, & otherwise \end{cases}$$

$$\Delta\tau_{ij}^{k}"(t+1) = \begin{cases} \frac{1}{L_{eniyi}}(t+1), & (i,j)\ best\ tour \\ 0, & otherwise \end{cases}$$

τ_{ij} given in Equation 6 is the initial pheromone level and ρ is the pheromone evaporation parameter. The local update rule dynamically changes the tours, making the routed routes attractive. While the ants take different paths, there is a high probability that one of them will improve the solution by taking a shorter path than the previous solutions. $\Delta\tau_{ijk}$ (t+1) It is calculated by different methods in different ant colony algorithms.

After all the created ants reach the end point, the pheromone amounts must be updated. The pheromone update is done in two different ways: local pheromone update and global pheromone update. Equation 6 contains the equations of the updates.

3.3. Parallelization

In traditional computing architecture proses can be called as Single Instruction Single Data (SISD). Besides in parallel computing architectures, depending on instructions and data structures it can be divided into different categories namely Single Instruction Multiple Data (SIMD) where it has one sequence of instructions applied to multiple data, Multiple Instructions Multiple Data (MIMD) where it has multiple sequence of instructions executing on multiple data and lastly Multiple Instructions Single Data (MISD). In SIMD systems there are two main frames to describe the architecture – lockup execution and vector processing. Where in the case of lockup processing, all Program Elements (PE) execute the same instructions on various data, in vector processing the same instruction set is repeatedly executed on different data. On the other hand, MIMD multiple programs executes on different data, however, if all PEs should have to cooperate to solve the problem also there should be interaction between the programs and/or the data.

From the memory point of view there are mainly two physically memory architectures for parallel computing – shared memory where all PE's are connected to the same physical memory unit throughout to interconnection and may be simultaneously accessed by multiple programs, and in distributed memory where the different physical memories are logically shared over a large address space. An illustration of parallel programming is given at the Figure 4 below.

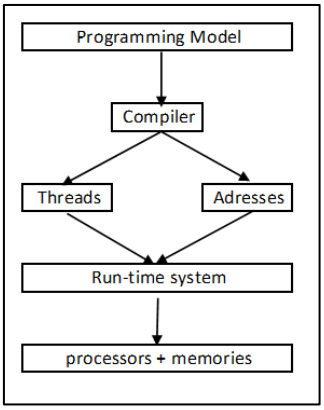

Figure 4. An illustration of parallel programming.

In shared memory based parallel programming architecture where the processes access the same data concurrently and the outcome of execution depends on the particular order in which the access takes place is called a race condition. In order to solve the race condition problem, a critical section should be determined meaning that at any time, only single process is allowed to access to same memory block. The main reason for parallel programming is to execute code efficiently because parallel programming saves computation time and allows applications to be executed in a shorter time. As a result of executing code efficiently, parallel programming often scales to problem size so it can solve larger problems. In general, parallel programming is a way of achieving concurrency, specifically performing multiple simultaneous actions at the same time. In the literature commonly used parallel programming models are presented for different parallelism architectures like Parallel Virtual Machine (PVM), Message Passing Interface (MPI) for distributed memory, Pthread and OpenMP for shared memory, MPI+OpenMP for shared memory distributed systems.

4. EXPERIMENTS

Within the scope of the study, the theoretically defined approach has been implemented as two applications, which can work in serial and in parallel computing. Comparisons were made to evaluate the working performances of serial and parallel applications by making computation time periods. Serial and parallel applications were performed on a computer with Intel Core i7-9750H CPU @ 2.60 GHz (with 12 cores), 16 GB RAM, Windows 10 Education 64-bit operating system.

"clock_t" function from C++ library is used for determining computation time. This function is located in the "ctime" library and the start is measured with the clock() command at the start stage of the program block whose performance will be measured, and the end is measured with the clock() command again at the end stage. Computation time can be obtained difference between start and end.

In order to make the comparison fair, the iteration numbers of the serial and parallel algorithms are taken as the same in the algorithm. Even if the number of places to visit is the same due to the nature of the genetic algorithm, it is impossible to predict when the convergence will occur, since there is randomness in the mutation and crossover processes.

In the following parts of this section serial and parallel computation of problem is presented orderly and computation performance of the algorithms are compared as serial case and parallel approach.

4.1. Serial Case

30 sensors are selected randomly in 2 dimensional space. Other parameters of the GA and AC algorithms are given at Table 2 and Table 3. It should be noted that the calculation performances will vary according to different number of sensors. The calculations will take longer when there are more sensors. Theoretically when more ants were selected possibility of finding the optimal solution will increase, but this will lead eventually to performance issues, which depend on the setup. Also there will be a stability point where adding more ants will not affect the performance of the model while reaching the optimal

solution. Finding the exact number of ants is an empirical problem based on fine tuning which is not discussed in this study. Furthermore tour sizes in AC algorithm is selected constant in order to compute the performance of parallelism. Besides this results a change in the optimal value.

Table 1. Results of Serial GA Application.

Tour Size	Generation Size	Best Value Found	Time(sn)
10	1000	62.6	12.504
10	2000	60.1	25.237
20	1000	59.2	45.896
20	2000	58.5	87.321

Table 2. Results of Serial AC Algorithm.

Number of Ants	Tour Size	Best Value Found	Time(sn)
20	100	60.3	21.4
40	100	59.6	22.2
60	100	58.5	24.2

As seen from Table 2 and Table 3, the convergence times of the algorithm increase as the number of rounds and generation size increase.

As it can be observed from the tables above both algorithms converges to 58.5. As can be seen from the tables above, when the number of tours and generations increase in GA, it is seen that the time taken for the calculation increases, but the found fitness function value improves. The reason for this is that the number of populations produced and the complexity of the operation increase while calculating. Computational complexity increases as the fitness function value of each individual in the population is recalculated at each step, and the number of crossover and mutation combinations used to calculate it increases. This also affects the elapsed time. On the other hand, it was observed that the obtained fitness function value improved when the number of rounds and generations increased.

In the AC algorithm, there is an inverse proportion between the time and the optimal value, as in GA.As the number of ants increases, the convergent value improves while the time lengthens. As a result of the experiments, it was observed that both algorithms converged to approximately the same value.

4.2. Parallel Application

GA and AC models are based on the well-known master/slave approach as its described in section 1.2 and 1.3 This approach is very suitable for multiple input, multiple data (MIMD) machine architectures. In considering different parallel techniques, it must be noted that parallel performance can be degraded when there is large communication overhead between processors. All of the methods assume a distributed rather than shared memory system, as these architectures are more common [29]. For this method, some of

sequential AC searches are run throughout available processors. Each colony is differentiated at the values of key parameters. While any of the parameters may be various throughout the processors, random seed might be the clean choice. The benefit of this technique is that no conversation is required among the processors. This is a naive method that may be run as some of sequential applications on an MIMD machines. In this research distributed memory is used.

In order to make a fair comparison of the parallel application with the serial application, the number of rounds and the number of generations are given as the same. The results of the parallel application are shown in Table 4.

Table 3. Parallel GA application time.

Tour Size	Generation Size	Best Value Found	Time(sn)
10	1000	62.5	7.305
10	2000	60.1	14.577
20	1000	59.1	21.007
20	2000	58.5	40.52

Due to system limitations 12 threads are used for parallel application in this research.

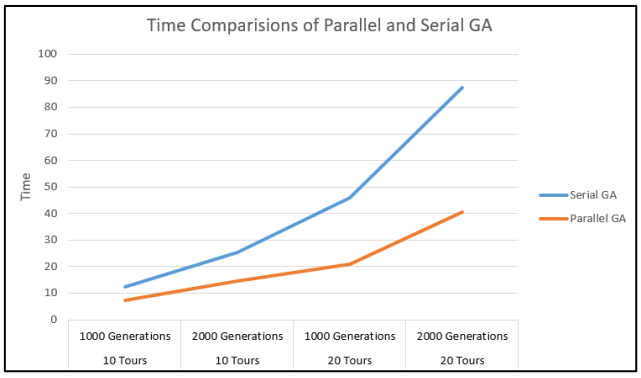

Figure 5. Comparison of serial and parallel GA algorithms.

Considering the parallel application computation performance, it is seen that its computation performance is better than the serial application. In Figure 5, the comparison of parallel application and serial application is shown on the graph.

Table 4. Results of parallel AC.

Number of ants	Tour Size	Best Value Found	Time(sn)
20	100	60.2	12.2
40	100	59.6	11.2
60	100	58.5	10.9

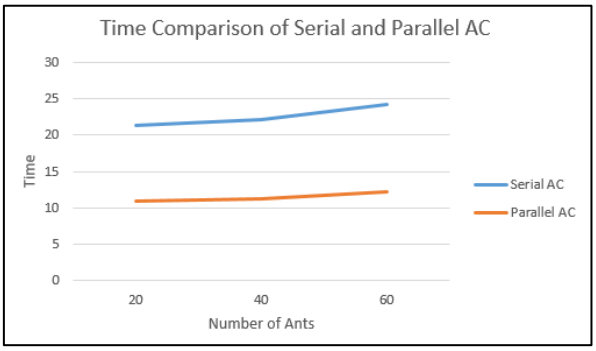

Figure 6. Comparison of serial and parallel AC.

As shown in Figure 5 and Figure 6, the parallel program ran much faster than the serial program. It was observed that the parallel structure gave better results when the number of turns and generations increased. When looked at for 10 tours and 1000 generations, the performance increase was 1.71 times, while this rate was calculated as 2.15 for 20 tours and 2000 generations. Likewise, it was observed that the optimal values for the ant colony algorithm were reduced to half the time as well. When best value performances of both serial and parallel are compared it is observed that they are very similar to each other. While the number of ant and generation size increases best value found converges to 58.5 for AC and GA.

5. CONCLUSION

In this study, the similarities of the nodes that need to be navigated in order to collect data from wireless sensor networks with the approach defined as the TSP in the literature were revealed, the solution was developed with Parallel Genetic Algorithm and Ant Colony Algorithm. Application has been developed. In order to solve TSP, both serial and parallel calculations were made using genetic algorithm and ant colony, which is one of the meta-heuristic methods.

It has been experimentally demonstrated that parallel computing has a much better computational performance compared to the serial one. Especially when the number of iterations is increased, it is seen that the performance is more than twice as fast.

In order to increase the performance of the model, a processor with more cores may be preferred According to Amdhal's law, if the proportion of execution time that the part benefiting from improved resources originally occupied can't be improved, the overall performance of the system cannot be improved, then speed cannot go beyond that, no matter how many processors are used. However, Brauer et al. in their study, they showed that the acceleration is not proportional to the number of processors and that acceleration does not occur when the number of processors is above thirty [30].

As a result, data collection from wireless sensor networks very important progress to improve network efficiency. Another point is data collection period affects network performance directly, efficient data collection period must be fast and effective. To achieve efficient data collection progress with minimum delay, computation of path must be defined fast, and results must be closest optimal solution. To achieve required

performance of computation by producing efficient results OpenMP application on the solution of TSP with parallel genetic algorithm and ant colony algorithm is designed and compared with traditional computation performances. Especially when the number of iterations is increased to achieve better solutions, it is seen that the computation performance is more than twice as fast, and delay is minimized.

REFERENCES

[1] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, *Genetic algorithms for the travelling salesman problem: A review of representations and operators,* Artif. Intell. Rev. **13**(2), 129–170 (1999).

[2] J. K. Lenstra and A. H. G. R. Kan, *Some simple applications of the travelling salesman problem*, J. Oper. Res. Soc. **26**(4), 717–733 (1975).

[3] J. H. Holland and others, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[4] A. Downey, *The little book of semaphores.* Green Tea Press, 2008.

[5] B. Gavish and S. C. Graves, *The travelling salesman problem and related problems,* 1978.

[6] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah, *Some applications of the generalized travelling salesman problem,* J. Oper. Res. Soc. **47**(12), 1461–1467 (1996).

[7] D. Karapetyan and G. Gutin, *Lin--Kernighan heuristic adaptations for the generalized traveling salesman problem,* Eur. J. Oper. Res. **208**(3), 221–232 (2011).

[8] K. Helsgaun, *An effective implementation of the Lin-Kernighan traveling salesman heuristic*, Eur. J. Oper. Res. **126**(1), 106–130 (2000).

[9] K. Helsgaun, *General k-opt submoves for the Lin-Kernighan TSP heuristic,* Math. Program. Comput. **1**(2), 119–163 (2009).

[10] M. Jünger, G. Reinelt, and G. Rinaldi, *The traveling salesman problem*, Handbooks Oper. Res. Manag. Sci. **7**, 225–330 (1995).

[11] G. G. Emel and Ç. Taşkin, *Genetik algoritmalar ve uygulama alanlari*, Uludağ} Üniversitesi İktisadi ve İdari Bilim. Fakültesi Derg. **21**(1), 129–152 (2002).

[12] J. R. Cheng and M. Gen, *Parallel Genetic Algorithms with GPU Computing,* in Industry 4.0, T. Bányai and A. P. F. De Felice, Eds. Rijeka: IntechOpen, 2020.

[13] T. Elmas and S. Taşkıran, "Paralel Yazılım Geliştirme Sürecinde Yarış Durumu Denetimi," Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, İstanbul, 2008.

[14] O. M. Sallabi and Y. Elhaddad, *An improved genetic algorithm to solve the traveling salesman problem,* 2009.

[15] M. Nowostawski and R. Poli, "Parallel genetic algorithm taxonomy," *in 1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No. 99TH8410),* 1999. pp. 88–92.

[16] V. Dwivedi, T. Chauhan, S. Saxena, and P. Agrawal, *Travelling salesman problem using genetic algorithm*, IJCA Proc. Dev. Reliab. Inf. Syst. Tech. Relat. Issues (DRISTI 2012), 2012.

[17] K. Rani and V. Kumar, *Solving travelling salesman problem using genetic algorithm based on heuristic crossover and mutation operator*, Int. J. Res. Eng. Technol. **2**(2), 27–34 (2014).
[18] F. H. Khan, N. Khan, S. Inayatullah, and S. T. Nizami, *Solving TSP problem by using genetic algorithm*, Int. J. Basic Appl. Sci. **9**(10), 79–88 (2009).
[19] M. L. Islam, D. Pandhare, A. Makhthedar, and N. Shaikh, A *Heuristic Approach for Optimizing Travel Planning Using Genetics Algorithm*, Int. J. Res. Eng. Technol. eISSN, 2014, pp. 1163–2319.
[20] S. Gupta and P. Panwar, *Solving travelling salesman problem using genetic algorithm,* Int. J. Adv. Res. Comput. Sci. Softw. Eng. **3**(6), 376–380 (2013).
[21] H. R. Er and N. Erdogan, "Parallel Genetic Algorithm to Solve Traveling Salesman Problem on MapReduce Framework using Hadoop Cluster," arXiv Prepr. arXiv1401.6267, 2014.
[22] M. Abbasi, M. Rafiee, M. R. Khosravi, A. Jolfaei, V. G. Menon, and J. M. Koushyar, *An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems*, J. cloud Comput. **9**(1), 1–14, 2020.
[23] A. Siemiński and M. Kopel, *Solving dynamic TSP by parallel and adaptive ant colony communities,* J. Intell. \& Fuzzy Syst. **37**(6), 7607–7618 (2019).
[24] P. Yelmewad and B. Talawar, *Parallel iterative hill climbing algorithm to solve TSP on GPU,* Concurr. Comput. Pract. Exp. **31**(7), 4974 (2019).
[25] G. A. Sena, D. Megherbi, and G. Isern, *Implementation of a parallel genetic algorithm on a cluster of workstations: traveling salesman problem, a case study,* Futur. Gener. Comput. Syst. **17**(4), 477–488 (2001).
[26] A. Di Placido, C. Archetti, and C. Cerrone, *A genetic algorithm for the close-enough traveling salesman problem with application to solar panels diagnostic reconnaissance,* Comput. \& Oper. Res. **145**, 105831 (2022).
[27] S. K. R. Kanna, K. Sivakumar, and N. Lingaraj, *Development of deer hunting linked earthworm optimization algorithm for solving large scale traveling salesman problem,* Knowledge-Based Syst. **227**, 107199 (2021).
[28] M. M. Krishna, N. Panda, and S. K. Majhi, *Solving traveling salesman problem using hybridization of rider optimization and spotted hyena optimization algorithm,* Expert Syst. Appl. **183**, 115353 (2021).
[29] W. P. Computing and I. Foster, *Designing and building parallel programs.* Addison-Wesley, 1995.
[30] M. J. Brauer, M. T. Holder, L. A. Dries, D. J. Zwickl, P. O. Lewis, and D. M. Hillis, *Genetic Algorithms and Parallel Processing in Maximum-Likelihood Phylogeny Inference,*Mol. Biol. Evol. **19**(10), 1717–1726 (2002).

To Cite This Article: B. Erkartal, Ö. Çetin, A. Yılmaz, *Data Collection from Wireless Sensor Networks: OpenMP Application on the Solution of Traveling Salesman Problem with Parallel Genetic Algorithm and Ant Colony Algorithm*, Journal of Aeronautics and Space Technologies **15**(2), 108-124 (2022).

VITAE

Buğra Erkartal received his B.Sc. degree in Systems Engineering from Yeditepe University, Turkey in 2003. He received his M.Sc. degree in Systems Engineering from Yeditepe University, Turkey in 2018. He is currently working at Beykent University as lecturer.

Ömer Çetin successfully completed his PhD in computer engineering in 2015 on autonomous systems and parallel programming applications. He is currently teaching courses on autonomous systems, image processing and parallel programming in undergraduate and graduate programs at different universities as assistant professor. He is currently the director of the autonomous systems laboratory and he has numerous international articles and papers on autonomous path planning, real-time programming applications and he continues to carry out research activities in these fields.

Atınç Yılmaz received his B.Sc. and M.Sc. degree in Computer Engineering. He received Phd degree in Computer an Informatics Engineering from Sakarya University, Turkey in 2015. His working areas are artificial intelligence, machine learning and big data technologies. He is currently teaching courses on these fields and base of computer sciences. Dr. YILMAZ is currently working at Beykent University as an assistant professor. He is also head of the computer engineering department in this university.