Received 6 December 2022, accepted 30 December 2022, date of publication 2 January 2023, date of current version 20 January 2023.

Digital Object Identifier 10.1109/ACCESS.2022.3233786

RESEARCH ARTICLE

Parallel Ant Colony Optimization Algorithm for Finding the Shortest Path for Mountain Climbing

ESRA'A ALHENAWI[1], RUBA ABU KHURMA[2], AHMAD A. SHARIEH[3], OMAR AL-ADWAN[2,3], AREEJ AL SHORMAN[3], AND FATIMA SHANNAQ[4]

[1]Department of Software Engineering, Faculty of Information Technology, Al-Ahliyya Amman University, Amman 19328, Jordan
[2]Department of Computer Science, Faculty of Information Technology, Al-Ahliyya Amman University, Amman 19328, Jordan
[3]Department of Computer Science, The University of Jordan, Amman 11942, Jordan
[4]Faculty of Computer Science and Informatics, Amman Arab University, Amman 11953, Jordan

Corresponding author: Ruba Abu Khurma (r.khurma@ammanu.edu.jo)

ABSTRACT The problem of finding the shortest path between two nodes is a common problem that requires a solution in many applications like games, robotics, and real-life problems. Since its deals with a large number of possibilities. Therefore, parallel algorithms are suitable to solve this optimization problem that has attracted a lot of researchers from both industry and academia to find the optimal path in terms of runtime, speedup, efficiency, and cost compared to sequential algorithms. In mountain climbing, finding the shortest path from the start node under the mountain to reach the destination node is a fundamental operator, and there are some interesting issues to be studied in mountain climbing that cannot be found in a traditional two-dimensional space search. We present a parallel Ant Colony Optimization (ACO) to find the shortest path in the mountain climbing problem using Apache Spark. The proposed algorithm guarantees the security of the selected path by applying some constraints that take into account the secure slope angle for the path. A generated dataset with variable sizes is used to evaluate the proposed algorithm in terms of runtime, speedup, efficiency, and cost. The experimental results show that the parallel ACO algorithm significantly ($p < 0.05$) outperformed the best sequential ACO. On the other hand, the parallel ACO algorithm is compared with one of the most recent research from the literature for finding the best path for mountain climbing problems using the parallel A* algorithm with Apache Spark. The parallel ACO algorithm with Spark significantly outperformed the parallel A* algorithm.

INDEX TERMS Apache spark, ant colony, parallel algorithm, path-finding problem, optimization.

I. INTRODUCTION

Finding the best path between two nodes (locations) in an area such as a mountain is an optimal search problem. The goal here is to find the shortest path or a path with the least cost, for example, between two nodes, between one node and other nodes, or between the set of nodes and another set of nodes [1].

The path planning issue under various constraints for massive data in geography distance is becoming more challenging [2], [3]. In the literature, there are many works focused on developing sequential or parallel solutions for optimal path problems. Parallel processing is an ultimate approach to solving applications of the shortest path with big data [4], [5], [6].

Mountain climbing is one of the most recent applications for path-finding problems in a three-dimensional environment. In this paper, a three-dimensional mountain curve is generated using a complex mathematical equation. Then, a set of points were generated randomly below the curve between the source node, and the destination node on the mountain surface. However, some of these nodes were considered obstacles, that would not be passed by ants as in the case in nature [7].

Classical search algorithms are ineffective in solving complex and nonlinear problems. Therefore, optimization algorithms based on meta-heuristics algorithms were utilized to establish search techniques for tackling a wide variety of

The associate editor coordinating the review of this manuscript and approving it for publication was Daniel Grosu.

complex problems like path-finding problems. Researchers in [8], [9], and [10] categorized meta-heuristic algorithms into ten classes: (1) Biology-based, (2) Physics-based, (3) Social-based, (4) Music-based, (5) Chemical-based, (6) Sport based, (7) Swarm-based, and (9) Plant-based, and (10) Water-based.

Genetic Algorithm(GA) [11] is biology-based; Electromagnetic field optimization (EFO) [12] is physics-based; Social group optimization (SGO) [13] is social-based; Harmony Search Algorithm (HSA) [14] is music-based; Artificial Chemical Reaction Optimization Algorithm (ACROA) [15] is chemical-based; Intelligent Water Drops algorithm (IWDs) [16] is water-based; Particle Swarm Optimization (PSO) [17] and Ant Colony Optimization (ACO) [18] algorithms are swarm-based. Compared with other optimization algorithms, ACO has advantages, including strong robustness, good global optimization ability, and inherent parallelism [19], [20].

The sequential ACO (SACO) may lead to poor convergence due to the randomness of the probabilistic transfer due to updating the pheromone while iterating. The convergence can be sped up by updating pheromones on the path of the optimal ant of each generation using a parallel version of the SACO algorithm [20].

The swarm particle optimization algorithm (SPO) is utilized with extremely significant academic and practical value [21]. SACO is one of the most representative SPO algorithms with self-organizational, distributive, operational, flexible, and robust properties. As a result, SACO is gradually being used to solve a wide range of problems in a variety of applications, including path-finding, scheduling, and other optimization problems. Based on Google Trends indicators of SPO algorithms from 2000 to 2020, research on deploying and on SACO has the second-highest rank after PSO, where the publication number, based on Google Scholar data, equals 35,000 research papers on SACO. The overall research shows an increasing growth trend from 2000 to 2020 [21].

SACO simulates ant behavior in nature by finding the best route to find food for solving any problems [22]. SACO can be utilized for finding the shortest path for mountain climbing from a specific point on the edge of a mountain to a specific destination on its surface. This is the point on which this paper focuses. An ant, along with the necessity to find food and bring it back to its nest, manages to explore a vast area and determine the food's location to its peers, so they can bring it back to the nest [23], [24], [25]. Ants detect where their destination is located, without having a global view of the ground [26], [27]. In an ant colony, a single individual has a very limited effect [24], but as a part of a well-organized colony, it can perform as an effective agent that works for the development of the colony [25]. Ant colonies appear to operate as a unified entity [26]. An ant walks from or to a food source and deposits a pheromone on the ground. Other ants can smell the deposited pheromone, and its presence guides other ants to choose their path; they tend to follow strong pheromone concentrations [23]. The deposited pheromone on the ground forms a pheromone trail. This allows the ants to find good sources of food that have been previously identified by other ants [26]. The ants will leave their one nest within a ground containing food source, and come back to the nest [28]. After a while, the path being used by the ants will converge on the best path [29].

In Wang and Han's study [30], they developed a hybrid symbiotic organism search (SOS) and ACO algorithm for solving the traveling salesman problem (TSP). Some parameters were optimized using SOS and used in the ACO. The results of this hybrid method showed good adaptive abilities for finding competitive solutions.

Li et al. [2] adjusted the ACO control parameters by adapting the greedy strategy to have a GSACO algorithm. The GSACO showed better performance in terms of convergence speed and run time than the ACO.

In parallel and distributed computing systems, a problem can be decomposed to have more than multiple processors or computers work to solve a problem. The main goal is to handle big data and speed up the process. There are two main methods of problem decomposition based on tasks or data decomposition [31], [32]. In data decomposition, the data is divided into parts and assigned to a specific core or computer. The cores or processors will run in parallel to complete the whole computation task that involved computation and communication processes [33] and [7], [34].

The ACO has a distributed nature where multiple ants can work as independent agents on part of the solution at the same time. Therefore, ACO can be implemented as a parallel ACO algorithm (PACO). [35], [36], [37], and [38] studies showed the possibilities of developing a parallel ACO algorithm to take advantage of multi-core hardware to find the best path. The need for an optimal path in applications such as in infrastructures, distributed nodes in networks, traveling salesman,...etc, and the nature of parallel behaviors of ants motivate us to develop a parallel ACO algorithm.

Apache Spark is an open-source framework that has been widely used to analyze big data for various applications [39]. It represents an optimization on Hadoop, where it supports batch processing only, but Spark is good for both batch processing and stream processing [40]. This means that Spark is suitable for analyzing data from social media sites and searching websites because it includes all the streaming data in the analysis process.

Spark is In-memory computation and provides real-time data processing capability [41]. It runs applications faster than Hadoop, easy to program, manage, and use. It includes Spark streaming, MLLib, GraphX, and Spark SQL for real-time data processing, machine learning, graph processing, and SQL querying, respectively. It also provides fault tolerance, and it can be scaled. The spark was designed for big data analysis [42], [43]. Spark has its read-only data item distributed in a space between clusters of machines that is named Resilient Distributed Dataset (RDD), and it works as a fault tolerance mechanism [44].

The main contributions of this paper are as follows:

- It documents the most recent path-searching applications in the literature and the way they are connected.
- It develops the best sequential ACO algorithm for finding the shortest path in one of the most recent path-finding problems called the 3D mountain climbing problem. There are some interesting issues to be studied that cannot be found in a traditional two-dimensional space search. In this paper, A complex mathematical equation is used to produce a 3D mountain curve.
- It develops a parallel version of the ACO algorithm using Apache Spark for finding an optimal path for the same problem.
- It compares the serial and parallel versions in terms of running time, speedup, efficiency, and cost, experimentally and theoretically, by analyzing each algorithm's complexity.
- It compares the proposed parallel ACO algorithm's performance with one of the most recent research results for solving the same optimization problem from the literature.

The rest of this paper is organized as follows. Section II displays the related works. Sequential and parallel algorithms are illustrated in Section IV. Setups for the experiments discussed in Section III. Section V presents the experimental results. Section VI is devoted to the obtained results discussion. Finally, the conclusion is presented in Section VII.

II. RELATED WORK

Many algorithms and techniques have been proposed in the literature to find the optimal path using either sequential or parallel methods. The following two subsections presented some of these works, which vary based on the nature of the problem and the type of path. Table 1 summarizes these efforts.

A. FINDING SHORTEST PATH PROBLEM

Finding the shortest path between two nodes is a common problem that requires a solution in many applications like games, robotics, and real-life problems. Numerous algorithms and techniques have been intended by several researchers to tackle this problem. However, finding the shortest path is not an easy task, where the best path varies from one application to another as it may be referring to the path length, the security [45], or smoothness [36]. Therefore, there is a growing demand for optimized solutions, and this section deliberates some related techniques based on meta-heuristics optimization to solve the issues in optimizing the shortest path.

The authors in [46] used a deep reinforcement learning algorithm and a local planning method for unmanned vehicles in harsh environments.

In addition, Sinodkin et al. [47] developed a hybrid A* algorithm to address the problem of path planning for self-driving vehicles based on a vehicle's kinematics using the Unity 3D game platform. Zhigalov et al. applied the A* algorithm for path-finding of vehicle movements in rough environments [48].

Furthermore, the authors in [49] proposed a sequential version of the hill-climbing algorithm and a parallel version using the Message Passing Interface (MPI) for finding the best path in the mountain environment. For comparing the two versions, they used three different sizes of datasets. Results showed that the parallel version outperformed the sequential one.

B. PARALLEL SHORTEST PATH ALGORITHMS

The use of parallel methods has been very common in several hot research areas. This is due to the increasing requirements for speedy and efficient parallel-based methods that can be concurrently applied. Hence, many researchers implement a parallel path-finding algorithm. For instance, the authors in [50] introduced a parallel evolutionary artificial potential field (PEAPF) concept for handling the dynamic obstacles in path planning for mobile robot navigation.

In addition, authors in [32] proposed a parallel algorithm to increase multi-core CPU utilization based on the same approach used for solving TSP.

The ACO algorithm is usually used to find the optimal path, and recently several studies have been presented to improve the performance of the ACO algorithm to find the optimal path by implementing a parallel ACO architecture. For instance, [38], the authors proposed an effective parallel ACO algorithmic method for CPU-based SIMD architecture. In this method, each ant is mapped to the CPU core, and each ant path is mapped into vector instructions. In addition, a new fitness function is proposed, which is called the Vector-based Roulette Wheel (VRW).

Moreover, The researcher in [37] applied a parallel, independent run where many ant colonies execute on processors and communicate depending on pheromone trials by sharing a memory or a message-passing interface (MPI). Meanwhile, Yu et al. [36] developed a parallel ACO algorithm for finding the best path in any environment in automated guided vehicles (AGVs) and intelligent warehouse planning. AGVs are required to search for a better path in a given work environment, according to warehouse path planning.

In addition, Islam et al. [35] deployed an ACO search algorithm for designing a source update for MANETs. The parallel ACO was used on a distributed memory machine to detect cycles using MPI. Results show that the proposed approach obtains a relative speedup of 7 with 10 processors.

Apache Spark has been used to improve system performance, especially when the amount of data becomes large or when computational operations increase dramatically. In the literature, Apache Spark has been used in various fields such as the medical field [51], machine learning field [52], [53], [54], [55], and path-finding, in which one of the recent work done by Alazzam et al., where the authors applied a parallel A* algorithm using a Hadoop Insight cluster provided by Azure with six worker nodes to find the optimal path using the Apache Spark in the mountain climbing environment. They evaluated the proposed algorithm in terms of runtime, cost, efficiency, and speedup on a generated dataset with different sizes. The proposed algorithm reached a speedup of 4.85.

Many works in the literature intensively investigate sequential or parallel strategies of ACO implementation in various applications ([2], [30], [36], [56]). However, no studies have applied the parallel ACO to finding the best path in mountain environments using Apache Spark. Therefore, this paper proposes sequential and parallel ACO algorithms using Apache Spark to find the shortest path between two nodes, one at the bottom of the mountain and the target on the top of the mountain surface, using the ACO algorithm. By analyzing each algorithm's complexity, the proposed parallel algorithm is compared with a sequential algorithm and a parallel A* algorithm in terms of running time, speedup, efficiency, and experimental and theoretical costs.

III. EXPERIMENTAL SETUPS

A. DATA GENERATION AND HYPOTHESES

The following are the problem's hypotheses:

- Point D is directly reachable from point S, if, and only if, there is a link (path) between those points with a slope that equals or is smaller than 22o.
- Some points are considered obstacles, such as an unsecured area, a lake, or a cliff. These obstacles will not be passed by ants, as is the case in nature [7]. So, they are not included in the path.
- A complex mathematical equation is used to produce a 3D mountain curve, and then a 2D segment is obtained when the third dimension is supposed to be 0.
- A set of points N has been randomly generated under the curve according to the previous hypotheses.
- Start and destination points are generated randomly within a specified range.

Fig. 1 shows the generated data on a mountain with random points under its curve. Given a source point s and a destination point d on the curve, the best path p is from s to d, denoted by $|pi|$, and it is defined to be a sequence of line segments on the curve that starts from s and ends at d. The length of a line segment pi between two points $p1(x1, y1)$, $p2(x2, y2)$, which is greater than 30, is defined to be the Euclidean distance in Equation 2.

The length of path p, which is denoted by $|p|$, is defined to be the sum of the length of all line segments of pi. In addition, the slop S of pi to be the acute angle ($<$ 22) is between pi and the shadow of pi computed using Equation 3. Moreover, for each line segment, pi computes the edge value EV represented by a combination of $|pi|$ *and* ($trail$) according to Equation 2 and Equation 4, respectively. More formally, we define the optimization problem to maximize F the previous constraints as shown in Equation 1.

$$F\,(EV) = \frac{1}{S} + trail^{\alpha} * \left(\frac{1}{|pi|}\right)^{\beta}, \quad where\ \alpha = 1,\ \beta = 5. \tag{1}$$

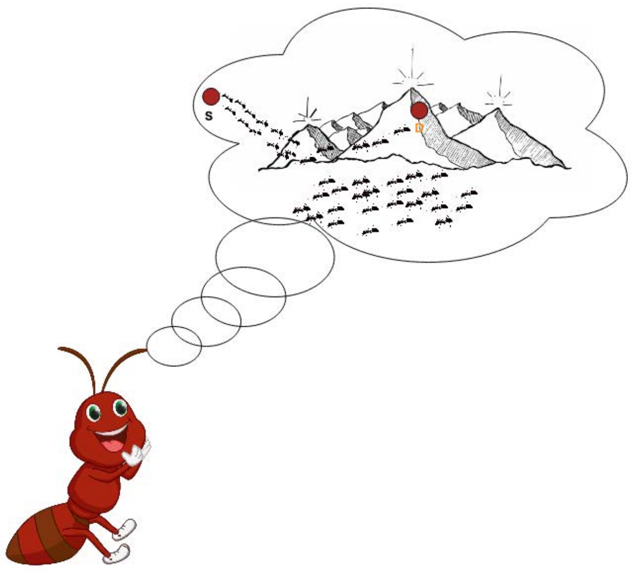

FIGURE 1. Mountain Climbing using ACO.

Alpha was set to 1 and Beta was set to 5 to prioritize cost over the trial at the beginning. The edge is preferred when it has more trails and less cost. The cost has a higher priority over the trail because we want the shortest path based on path cost, whereas the trail must have the maximum value. Sometimes, however, the path that has the highest pheromone value may not be the shortest; one ant may follow it randomly, and other ants may follow that road. Here, the pheromone effect is less than the cost in Equation 1.

$$|pi| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2}$$

$$S = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{3}$$

$$trail = \frac{500}{N}, \quad where\ N\ is\ a\ number\ of\ points. \tag{4}$$

B. PERFORMANCE METRICS

This subsection presents the definitions and equations for all metrics used to evaluate the proposed approach for finding the best path in the mountain environment [58]:

- *Runtime* represents the time consumed from the beginning of the execution process until it is finished.

$$Runtime = Termination\_time - Initialization\_time \tag{5}$$

- *Speedup* represents the ratios between sequential and parallel runtimes used for solving a specific problem.

$$Speedup = \frac{Sequential\ time}{Parallel\ time} = \frac{T_s}{T_p} \tag{6}$$

- *Efficiency* is represented in a ratio between speedup values and the number of processors.

$$Efficiency = \frac{Speedup}{P} = \frac{T_s}{p\,T_p} \tag{7}$$

TABLE 1. Summary of related work.

Ref.	Year	Algorithm	Scope
[57]	2021	A*	The optimal path using the Apache Spark in the mountain climbing environment
[48]	2021	A*	Path-finding for vehicle movement in rough environments
[47]	2021	A*	Path planning for self-driving vehicles
[46]	2020	deep reinforcement learning algorithm	Unmanned vehicle in harsh environments
[49]	2020	Hill Climbing	Finding the best Path in the mountain environment
[36]	2020	parallel ACO	Finding the best path in any environment in automated guided vehicles (AGVs) and intelligent warehouse planning
[50]	2015	parallel evolutionary artificial potential field (PEAPF)	Path planning for mobile robot navigation

- *Cost* is the runtime multiplied by the number of the used processors.

$$Cost = pT_P \tag{8}$$

- *TotalParallelOverhead*(*To*)
  It is the difference between parallel performance on one processor and the cost of the sequential algorithm used for solving specific problems.

$$\begin{aligned} To &= parallelperformanceononeprocessor \\ &\quad - sequentialtime = pT_p - T_s. \end{aligned} \tag{9}$$

C. EXPERIMENTAL ENVIRONMENT

Apache Spark is utilized to run the experiments of the parallel ACO algorithm. This algorithmic method is assessed based on a varying number of nodes and cores. Averages across multiple runs are then considered in recording the outcomes.

Experiments were carried out to enhance the performance of the parallel ACO which demonstrated certain encouraging outcomes. To enhance the performance of finding a path by implementing the parallel ACO across the sequential ACO, operations were run on 1-8 cores. Every experiment was carried out five times and the average duration was recorded as the final value for each test case. All these experiments were tested based on the MapReduce framework and Spark platform that deployed on Microsoft Azure virtual machine(VM). The VM has the following specifications:

- CPU Inter(R) Xeon(R) CPU E52673 v4 2.30GHz
- RAM 128GB
- All algorithms have been written using Java language and run on Eclipse IDE.

For these experiments, we utilized test cases to validate and confirm the robustness of our proposed version of the parallel ACO. This is implemented by a varying number of nodes from 250 up to 2000 with an increment step (current number of nodes $\times 2$) to evaluate various ranges of datasets' sizes, including small (represented by 250 points graph), medium (represented by 1000 points graph), and large (represented by 2000 points graph). All cases are implemented repeatedly with 1-8 cores running over 5 cycles, with the averages of all readings recorded as final.

Also, small(250 points), medium(1000 points), large (2000 points), and huge(50K points) datasets using Parallel ACO with Spark have been applied on a cluster environment, where Apache Spark runs on Hadoop Insight Cluster provided by Microsoft Azure with a Linux operating system, two master nodes, and six worker nodes with eight cores per node as a cluster setting. Table 2 displays the ACO parameter values that were utilized in the experiments. Tables 3 and Table 4 display the properties of the utilized system.

IV. IMPLEMENTATION OF THE PROPOSED SEQUENTIAL AND PARALLEL ACO VERSIONS FOR MOUNTAIN CLIMBING PROBLEM

This section displays the implementation of the sequential ACO algorithm, and the parallel ACO algorithm using the Spark Framework.

A. IMPLEMENTATION OF SEQUENTIAL ACO ALGORITHM

The ACO algorithm has two main stages:

- Route construction:
  Using ACO, an individual ant simulates a climber, and its path is constructed by incrementally selecting points in the neighborhood of the starting point until it reaches the destination point on the mountain's surface. Initially, each ant starts at the same point, where the set of points included in its tour is empty. The ant selects the next point to visit from the set of randomly selected nodes between the starting point and the destination point [7], [59], [60]. The list of randomly generated points and the storage capacity of this climber is updated before another point is selected. To the best of our knowledge, the proposed algorithm tries to find the shortest path, which computes the slope requirement (slop less than 0.1) based on Equation 2 and the distance

TABLE 2. ACO parameters values.

parameter	Value	Description
$Maxiterationsnumber(m)$	1000	Maximum number of iterations
N	based on number of graph generated nodes	Number of generated nodes
$have\_ant$	initially "False" for all points except the source one	start from points that have ants
$slope$	less than or equal to 0.4	slope value to guarantee safe climbing
$init\_phermone$	0	The initial pheromone on each edge
$InitVel$	200	The initial velocity of each ant
$Pheromone\_increament$	0.5	The pheromone updating parameter
$Pheromone\_decreament$	$500/N$	The pheromone updating parameter

TABLE 3. VM properties for the first part of experiments.

Component	Value
Processor	Inter(R) Xeon(R) CPU E52673 v4 2.30GHz
RAM	128 GB

TABLE 4. Cluster properties for the second part of experiments.

Component	Value
Number of master nodes	2
Number of worker nodes	6
Number of cores per node	8

requirement (Euclidian distance greater than 30) according to Equation 3.
In addition, the total distance D is computed as the objective function value for the complete route of the artificial ant. The ACO algorithm constructs a complete tour for the first ant before the second ant starts its tour. This continues until a predetermined number of ants m construct a feasible route. Using ACO, each ant must construct a route to reach the destination point from the starting point [38].

- Trail updating:
  To improve future solutions, the pheromone trails of the ants must be updated to reflect the ant's performance and the quality of the solutions found. This update is a key element of the adaptive learning technique of ACO, and it helps ensure the improvement of the subsequent solutions. Trail updating includes local updating of trails after generating the individual solutions and global updating of the best solution route after a predetermined number of solutions m has been accomplished. First, local updating is conducted by reducing the amount of pheromone on all visited arcs to simulate the natural evaporation of the pheromone and to ensure that no path becomes too dominant. After a predetermined number of ants $m$construct a feasible route, a global trail update is performed by adding a pheromone to all of the arcs included in the best route found by one of n ants. This method encourages the use of shorter routes and increases the probability that future routes will use the arcs contained in the best solutions.

  This process is repeated for a predetermined number of iterations, and the best solution from all of the iterations is presented as an output of the model and should represent a good approximation of the optimal solution for the problem [27], [28], [31]. Algorithm 1 presents the pseudo-code for the sequential ACO algorithm used for the mountain climbing problem.

Algorithm 1 Sequential Algorithm for Mountain Climbing Problem Using ACO

Input: Source (S), Destination (D), Randomly generated nodes N, Max_iterations_number(m), Max_ants_number (k), and constant quantity (Q).
Output: Best path from S to D.

```
1:  while ($Iterationnumber \neq m$) do
2:      while ($Iteration\_num \neq k$) do
3:          Check all nodes for constraints from S or previous node
4:          Add edges from s or the previous node to each node in node C that satisfies the constraints
5:          Generate random number $r \leftarrow 0 - 100$
6:          if ($r == 0.01$)
7:          Select a random edge from available ones for passing the ant
8:          else
9:          Select edge with maximum EV.
10:         $pheromone+ = (Q/N)$
11:         $pheromone- = .5$
12:     end while
13: end while
14: if (D.edges.size = = 0)
15: Path is not found
16: else
17: while not reach S do
18:     Store edge with the smallest cost
19: end while
20: returns the best path
```

B. IMPLEMENTATION OF PARALLEL ACO ALGORITHM

Implementation for parallel mountain climbing using ACO goes through the following steps on Spark:

1) PREPARING INPUT

In this step, the input point is prepared by specifying id, edges set, $have\_ant$, x, and y dimensions for each point. Initially, all nodes have (Id, $edges$, $have\_ant$ = "$False$", x, y), where Id is a unique identifier for the node. Edges ($cost$, $trail$) are a set of the available edges of nodes and have_ant is a Boolean variable. In addition, put variable have_ant is true for the source.

2) MAPPING TO RDD NODES

After pre-processing the data, the nodes are mapped into RDD nodes to be accessed by all processors or machines.

3) SPECIFYING THE NUMBER OF ITERATIONS

The number of iterations is set to 1,000 iterations. Each iteration has the following steps:

- Filtering: In this step, the nodes are filtered for retrieved nodes that have ants and distributed to processors.
- Exploring all possible nodes: In this step we build edges from the current node to all nodes that satisfy the constraints [57]:
  1) Not obstacle node
  2) Go toward the destination away from the source node
  3) Its $slop$ < 0.4 ($angle$ < 22) to guarantee safe climbing
  4) Its $cost$ > 30
- Selecting Edge:
  In this step, an ant selects one edge to follow from all possible nodes that were explored previously, either randomly with a small probability or based on the maximum EV in Equation 1.
- Updating the pheromone's value:
  This step aimed to increase the trail (pheromone) value for the selected edge and decrease the trail value for all explored edges.

4) RETRIEVING THE BEST PATH

After all, iterations are finished, the shortest path from the destination to the source that is based on the minimum accumulative cost for edges will be retrieved. Algorithm 2 shows the pseudo-code for parallel ACO using Spark.

V. RESULTS

This section is organized into two subsections. The first subsection presents the results of comparing the Parallel version of the ACO algorithm and the best result of the sequential ACO algorithm concerning finding the best path for the Mountain climbing problem.

Subsection two displays the comparison results of using the proposed parallel ACO using a cluster environment versus one of the most recent state-of-the-art research for solving the same optimization problem (Parallel A*).

Algorithm 2 Parallel Algorithm for the Mountain Climbing Problem Using ACO

Input: Source (S), Destination (D), Randomly generated nodes N, Max_iterations_num(m), number of processor(p), and Constant quantity(Q).
Output: Best path from S to D.

```
1:  Map N nodes to RDD file
2:  while ($Iteration\_num \neq m$) do
3:      Read RDD file
4:      Filter all nodes A in RDD where ($have\_ant$ = "$true$")
5:      Distribute $A$ into $p$ processor
6:      For each node in $A/p$
7:      Check all nodes for constraints
8:      Add edges from $A/p$ to each node in nodes $C$ that satisfy constraints
9:      Set ($have\_ant$ = "$true$") for all $C$ nodes
10:     Generate random number $r$ ←0 -100
11:     if (r = = 0.01)
12:     Select a random edge from available ones for passing the ant
13:     else
14:     Select the edge with maximum EV.
15:     $pheromone+ = (Q/N)$
16:     $pheromone- = .5$
17: end while
18: if (D.edges.size = = 0)
19: Path is not found
20: else {
21: while not reach S do
22:     add edge with the minimal cost to the best path
23: end while}
24: return best path
```

A. PARALLEL ACO VERSUS PARALLEL A* RESULTS

This subsection of results displays the findings that are generated from running an Apache Spark on Hadoop Insight Cluster provided by Microsoft Azure with a Linux operating system, two master nodes, and six worker nodes with eight cores per node as cluster settings.

Fig. 2 presents the results of parallel ACO versus parallel A* for a small graph with 250 points, while Fig. 3 illustrates the results of a medium graph size with 1000 points, and Fig. 4 demonstrates the results of a large graph with 2000 points.

Fig. 5 presents improvement and performance comparisons of parallel run times in 2, 3, and 4 cores, against sequential implementations on the single processor for various test cases.

Fig. 6 shows the increasing speedup using the parallel ACO algorithm corresponding to 2, 3, and 4 on various numbers of nodes.

Fig. 7 demonstrates the efficiency of the parallel ACO algorithm for different graph sizes using various numbers of cores each time (from 1 to 4 cores).

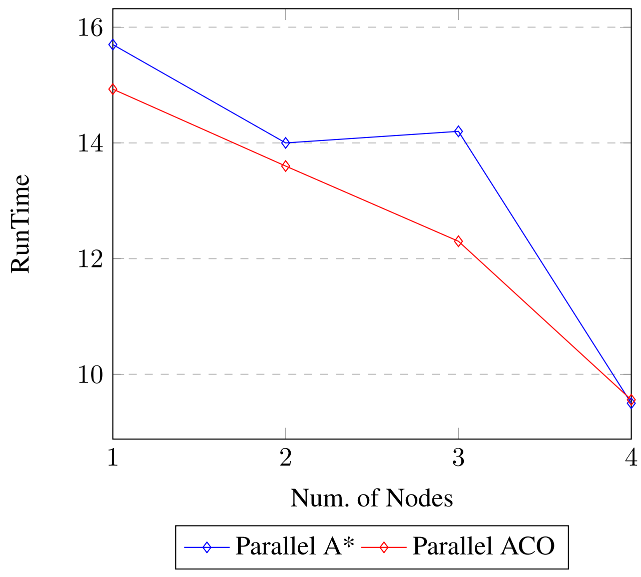

FIGURE 2. **Runtime results of Parallel ACO vs. Parallel A* for a Small graph (with 250 points).**

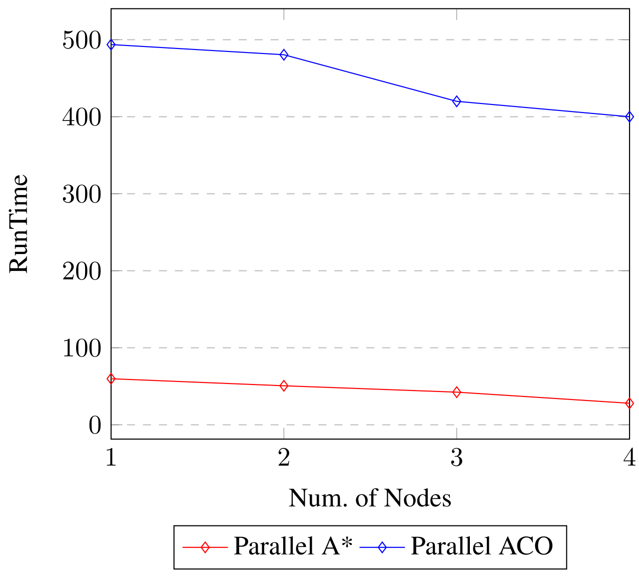

FIGURE 4. **Runtime results of Parallel ACO vs. Parallel A* for a Small graph (with 2000 points).**

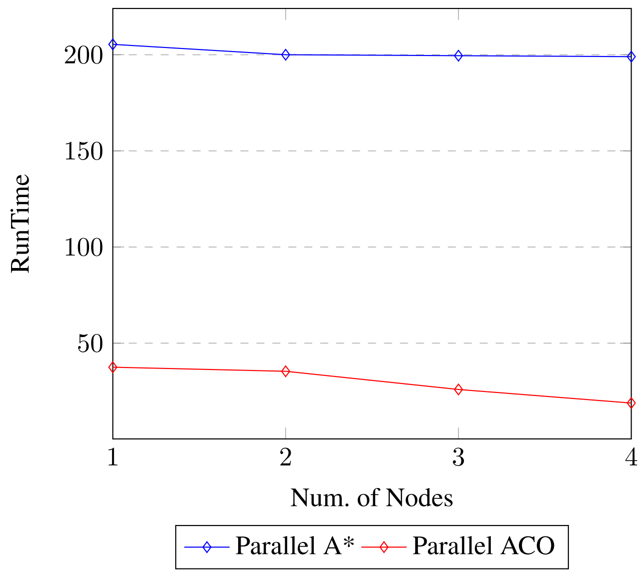

FIGURE 3. **Runtime results of Parallel ACO vs. Parallel A* for a Small graph (with 1000 points).**

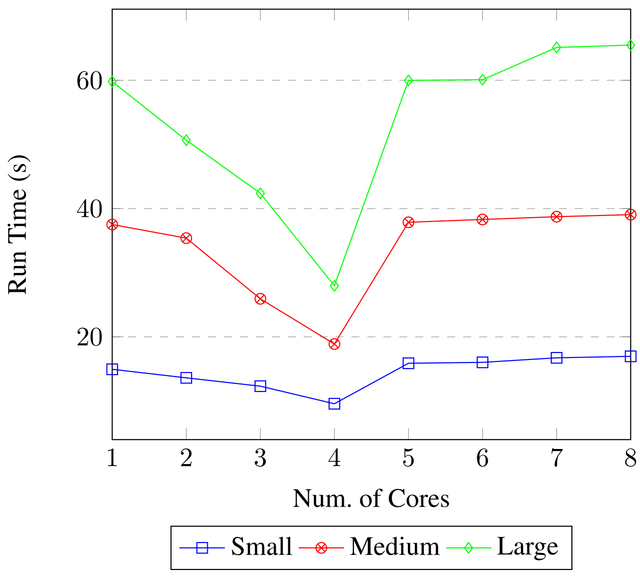

FIGURE 5. **Runtime for ACO with a different graph sized and 1-8 cores.**

Fig. 8 demonstrates the cost for ACO algorithms, based on various numbers of nodes and cores. As shown, the cost increased due to the increase in both graph's size and the number of cores.

Fig. 9 demonstrates the run time for small(250 points), medium(1000 points), large(2000 points), and huge(50K points) datasets using Parallel ACO with Spark on a cluster environment.

Fig. 10 presents the comparison results that have been obtained from using Parallel ACO versus using parallel A* based on run time.

Fig. 11 displays comparison results in terms of efficiency between the proposed parallel ACO versus parallel A*.

VI. DISCUSSION OF THE RESULTS

In this section, the findings are discussed and assessed based on runtimes, speedup, and efficiency. Runtime decreases when the number of cores increases with a varying number of nodes. However, the runtime after 4 cores will stop decreasing due to the additional Spark overhead caused by the communication time (where Apache Spark schedules the workload) as the problem becomes smaller than the number of cores as shown in Fig. 5. As a result, the proposed algorithm helps to determine the suitable number of cores that achieve the advantages of parallelism.

The parallel ACO achieves the best speedup values, up to 25%, particularly with large processor counts. The maximum speedup is satisfied at 2000 nodes when the number of

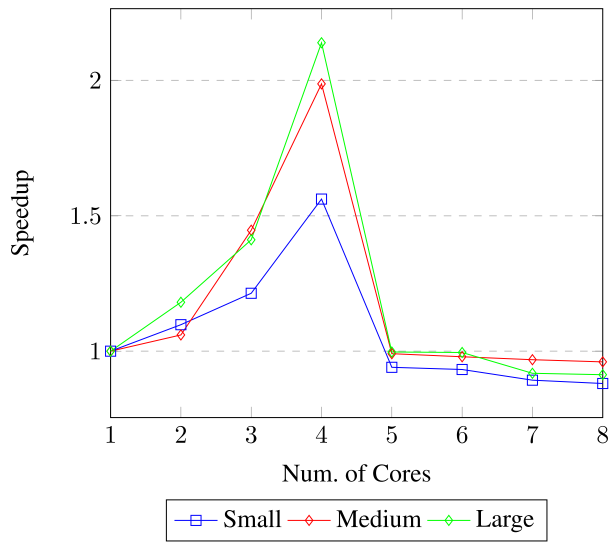

FIGURE 6. Speedup for ACO with a different graph size and 1-8 cores.

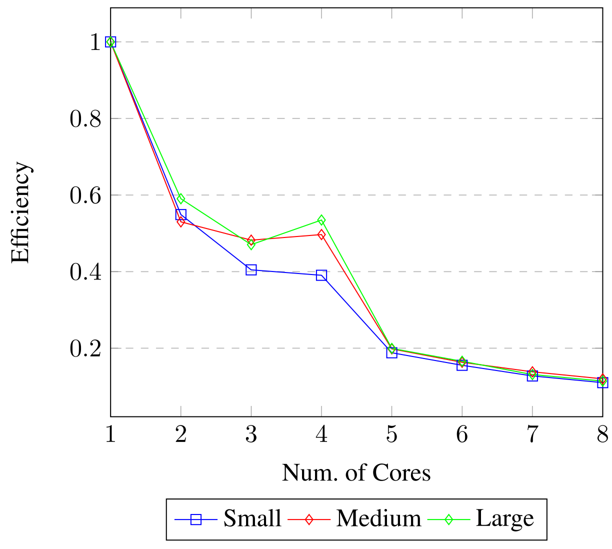

FIGURE 7. Efficiency for ACO with a different graph sized and 1-8 cores.

cores is 4 as shown in Fig. 6, which shows speedup versus the number of cores that are greater than 4. Speedup stops increasing due to the additional Spark overhead.

It is observed from Fig. 7 that efficiency decreases for the same graph size (the graph that has the same number of points) when the number of cores increases. Additionally, it is clear that when the number of points in the graph increases, the efficiency also increases using the same number of cores. On the other hand, the efficiency decreases using 5, 6, 7, and 8 cores because Apache Spark schedules the workload.

1) COMPARING THE PROPOSED APPROACH (USING THE PARALLEL ACO) VERSUS ONE OF THE MOST RECENT STATE-OF-THE-ART APPROACHES (USING PARALLEL A*) RESULTS IN DISCUSSION

Results illustrated that using the parallel ACO provides better results than the parallel A* for different graph size that ranges

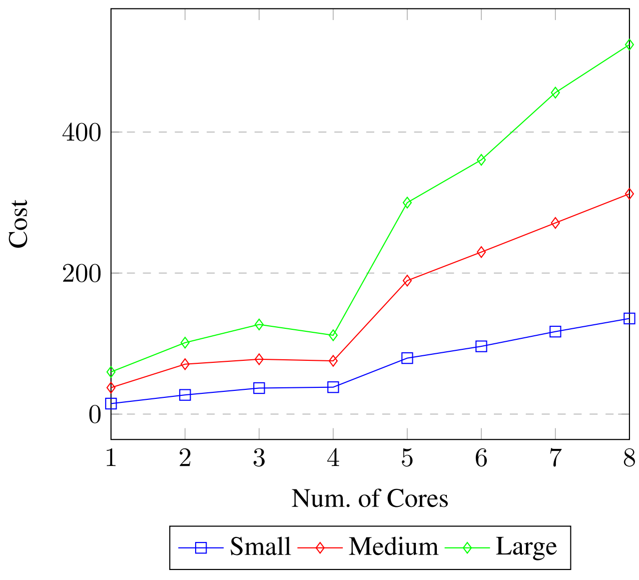

FIGURE 8. Cost for ACO with a different graph sized and 1-8 cores.

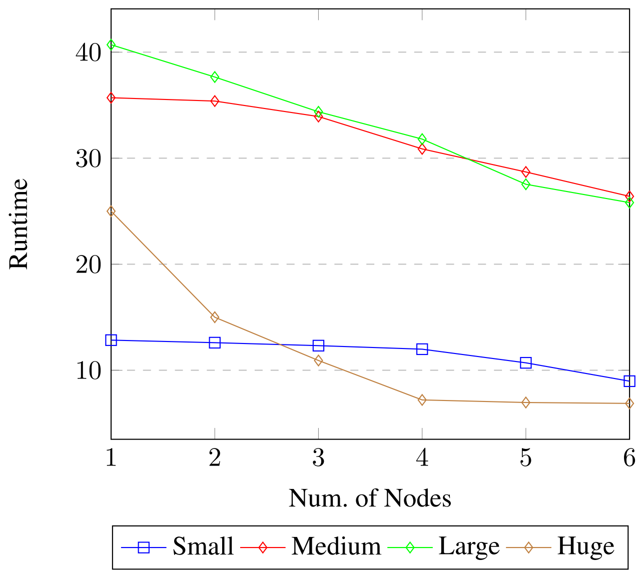

FIGURE 9. Runtime using ACO with a different number of nodes on different graph sizes.

between (250 to 2000) points using various number of cores (1 - 4 cores). Using Wilcoxon's test, it is clear that the parallel ACO is significant ($p < 0.05$) outperformed the parallel A* regardless of the graph size, but the improvement increases for larger graph sizes.

2) RESULTS OF USING THE PROPOSED APPROACH (USING THE PARALLEL ACO) ON A CLUSTER ENVIRONMENT WITH DIFFERENT GRAPH SIZES

Results show that Apache Spark is more efficient in huge datasets, where Spark doesn't achieve better runtime results using more nodes over datasets of the small, medium, and large sizes; while on a huge dataset the run time results decrease in an efficient way using the proposed method with

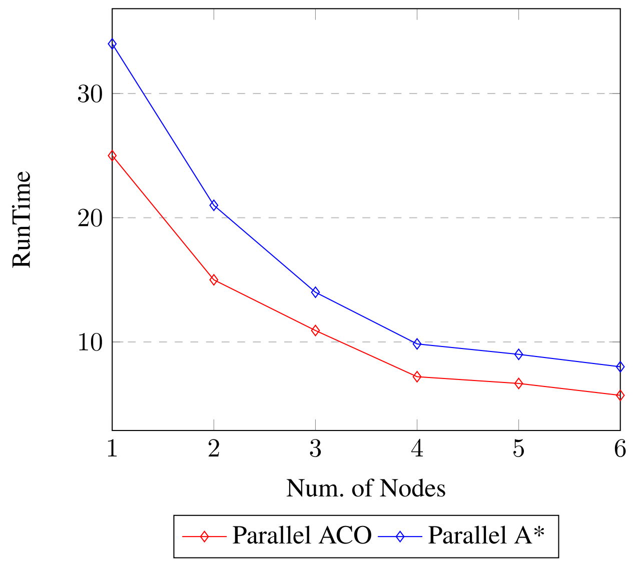

FIGURE 10. Runtime results of Parallel ACO vs. Parallel A* with a different number of nodes over a huge graph on a cluster environment.

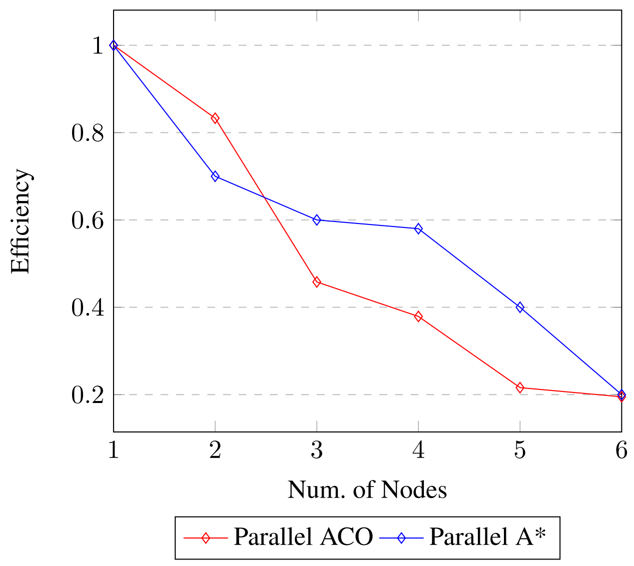

FIGURE 11. Efficiency results of Parallel ACO vs. Parallel A* with a different number of nodes over a huge graph on a cluster environment.

Apache Spark on a cluster environment but at some number of nodes the runtime will stop decreasing as the problem size will become smaller for the available number of nodes. Overall, it can be seen from Fig. 11 that efficiency decreased due to the Spark scheduling workload process.

A. COMPLEXITY ANALYSIS

The time complexity of the sequential approach is presented in Equation 10, and the time complexity for the parallel approach is presented in Equation 11

$$T_s m + km + k + Nk + 7 + 4m * \sum_{i=1}^{k} (N - CV) = O(Nmk) \tag{10}$$

where, T_s is the sequential time, N represents the set of randomly generated nodes between the source and destination, m is the maximum number of iterations, k is the maximum number of ants, and CV is a constant value.

$$T_p = 3N + 3m + 4\frac{A}{p}C + 7 + \frac{A}{p} + \frac{A}{p}\sum_{i=1}^{k}(N - CV) = O\left(\frac{ANm}{p}\right) \tag{11}$$

where, T_p is the parallel time, N represents the set of randomly generated nodes between the source and destination, m is the maximum number of iterations, p is the number of processors, A is the number of nodes in RDD file, C is the set of nodes that satisfy constraints, and CV is a constant value.

So, the main performance metrics for ACO parallel algorithms are:

- Total Parallel Overhead T_o

$$T_o = pT_p - T_s = P\left(\frac{ANm}{p}\right) - Nmk = Nm(A - k) \tag{12}$$

- Speedup Equation 13 shows the speedup for the proposed approach.

$$Speed\ Up = \frac{T_s}{T_p} = \frac{Nmk}{\frac{ANm}{p}} = \frac{p}{A} \tag{13}$$

- Efficiency E

$$E = \frac{Speedup}{p} = \frac{p}{pA} = \frac{1}{A} \tag{14}$$

- Cost

$$Cost = pT_p = p\frac{ANm}{p} = ANm \tag{15}$$

If $A = k$, then $cost = Ts$, thus the parallel algorithm is cost-optimal.

VII. CONCLUSION

In this paper, we present performance evaluations of the parallel ACO algorithmic in terms of the runtime, speedup, efficiency, and cost to find the shortest path between two points in a mountain environment. In this problem, we consider the slope and distance requirements such that the path is not too steep. Assessments of the parallel ACO consider different numbers of cores and nodes using Apache Spark. Overall, our findings demonstrate that parallel ACO features shorter running times for various numbers of the node with a limited number of cores. Concerning efficiency, it seems that paralyzing the ACO using Apache Spark is efficient only with large graph sizes; otherwise, the cost of Apache Spark will be unaccepted.

In addition, the parallel ACO performance was compared with one of the most recent research for the same optimization problem which is parallel A* with Apache Spark. Parallel ACO outperformed the other algorithm significantly as was obvious from the experimental results. Also, it is obvious from the results that the overhead of Spark is unnecessary unless using a huge dataset like a dataset with 50000 points.

In the future, the parallel version of ACO's performance using Spark can be compared with other optimization algorithms or used for several optimization problems. Moreover, ACO can be parallelized using Spark, Hadoop (MapReduce), or MPI, and the results of each would be compared.

REFERENCES

[1] C. Blum, "Ant colony optimization: Introduction and recent trends," *Phys. Life Rev.*, vol. 2, no. 4, pp. 353–373, 2005.

[2] W. Li, L. Xia, Y. Huang, and S. Mahmoodi, "An ant colony optimization algorithm with adaptive greedy strategy to optimize path problems," *J. Ambient Intell. Humanized Comput.*, vol. 13, pp. 1–15, Mar. 2021.

[3] O. Salzman and R. Z. Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems blue sky ideas track," in *Proc. 19th Int. Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2020, pp. 1711–1715.

[4] H. Alazzam and A. Sharieh, "Parallel DNA sequence approximate matching with multi-length sequence aware approach," *Int. J. Comput. Appl.*, vol. 975, no. 26, pp. 1–6, 2018.

[5] Y. Zhang, A. Azad, and A. Buluç, "Parallel algorithms for finding connected components using linear algebra," *J. Parallel Distrib. Comput.*, vol. 144, pp. 14–27, Oct. 2020.

[6] S. Mezzoudj, A. Behloul, R. Seghir, and Y. Saadna, "A parallel content-based image retrieval system using spark and tachyon frameworks," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 33, no. 2, pp. 141–149, Feb. 2021.

[7] K. Madduri, D. A. Bader, J. W. Berry, and J. R. Crobak, "An experimental study of a parallel shortest path algorithm for solving large-scale graph instances," in *Proc. 9th Workshop Algorithm Eng. Experiments (ALENEX)*. Philadelphia, PA, USA: SIAM, pp. 23–35, 2007.

[8] B. Alatas and H. Bingol, "A physics based novel approach for travelling tournament problem: Optics inspired optimization," *Inf. Technol. Control*, vol. 48, pp. 373–388, Sep. 2019.

[9] S. Akyol and B. Alatas, "Plant intelligence based metaheuristic optimization algorithms," *Artif. Intell. Rev.*, vol. 47, no. 4, pp. 417–462, May 2017.

[10] H. Bingol and B. Alatas, "Chaos based optics inspired optimization algorithms as global solution search approach," *Chaos, Solitons Fractals*, vol. 141, Dec. 2020, Art. no. 110434.

[11] J. R. Sampson, "Adaptation in natural and artificial systems (John H. Holland)," *SIAM Rev.*, vol. 18, no. 3, pp. 529–530, Jul. 1976.

[12] H. Abedinpourshotorban, S. M. Shamsuddin, Z. Beheshti, and D. N. A. Jawawi, "Electromagnetic field optimization: A physics-inspired metaheuristic optimization algorithm," *Swarm Evol. Comput.*, vol. 26, pp. 8–22, Feb. 2016.

[13] A. Naik, S. C. Satapathy, A. S. Ashour, and N. Dey, "Social group optimization for global optimization of multimodal functions and data clustering problems," *Neural Comput. Appl.*, vol. 30, no. 1, pp. 271–287, Jul. 2018.

[14] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *J. Simul.*, vol. 76, no. 2, pp. 60–68, Feb. 2001.

[15] B. Alatas, "ACROA: Artificial chemical reaction optimization algorithm for global optimization," *Exp. Syst. Appl.*, vol. 38, no. 10, pp. 13170–13180, 2011.

[16] H. Shah-Hosseini, "The intelligent water drops algorithm: A nature-inspired swarm-based optimization algorithm," *Int. J. Bio-Inspired Comput.*, vol. 1, nos. 1–2, pp. 71–79, 2009.

[17] R. L. Perez and K. Behdinan, "Particle swarm approach for structural design optimization," *Comput. Struct.*, vol. 85, nos. 19–20, pp. 1579–1588, 2007.

[18] T. Stützle, "Ant colony optimization," in *Proc. Int. Conf. Evol. Multi-Criterion Optim.* Springer, 2009, p. 2.

[19] V. Selvi and R. Umarani, "Comparative analysis of ant colony and particle swarm optimization techniques," *Int. J. Comput. Appl.*, vol. 5, no. 4, pp. 1–6, 2010.

[20] X. Dai, S. Long, Z. Zhang, and D. Gong, "Mobile robot path planning based on ant colony algorithm with A* heuristic method," *Frontiers Neurorobotics*, vol. 13, p. 15, Apr. 2019.

[21] J. Tang, G. Liu, and Q. Pan, "A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 10, pp. 1627–1643, Oct. 2021.

[22] E. Alhenawi, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Feature selection methods on gene expression microarray data for cancer classification: A systematic review," *Comput. Biol. Med.*, vol. 140, Jan. 2022, Art. no. 105051.

[23] M. Pedemonte, S. Nesmachnow, and H. Cancela, "A survey on parallel ant colony optimization," *Appl. Soft Comput.*, vol. 11, no. 8, pp. 5181–5197, 2011.

[24] Q. Wang, M. Kaul, C. Long, and R. C.-W. Wong, "Terrain-toolkit: A multi-functional tool for terrain data," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1645–1648, Aug. 2014.

[25] Y. Zhou, "Runtime analysis of an ant colony optimization algorithm for TSP instances," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1083–1092, Oct. 2009.

[26] N. Attiratanasunthron and J. Fakcharoenphol, "A running time analysis of an ant colony optimization algorithm for shortest paths in directed acyclic graphs," *Inf. Process. Lett.*, vol. 105, no. 3, pp. 88–92, Jan. 2008.

[27] W. J. Gutjahr and G. Sebastiani, "Runtime analysis of ant colony optimization with best-so-far reinforcement," *Methodology Comput. Appl. Probab.*, vol. 10, no. 3, pp. 409–433, Sep. 2008.

[28] F. Neumann, D. Sudholt, and C. Witt, "Computational complexity of ant colony optimization and its hybridization with local search," in *Innovations in Swarm Intelligence*. Springer, 2009, pp. 91–120.

[29] W. J. Gutjahr, "First steps to the runtime complexity analysis of ant colony optimization," *Comput. Operations Res.*, vol. 35, no. 9, pp. 2711–2727, Sep. 2008.

[30] Y. Wang and Z. Han, "Ant colony optimization for traveling salesman problem based on parameters optimization," *Appl. Soft Comput.*, vol. 107, Aug. 2021, Art. no. 107439.

[31] D. Sudholt and C. Thyssen, "Running time analysis of ant colony optimization for shortest path problems," *J. Discrete Algorithms*, vol. 10, pp. 165–180, Jan. 2012.

[32] J. Ning, Q. Zhang, C. Zhang, and B. Zhang, "A best-path-updating information-guided ant colony optimization algorithm," *Inf. Sci.*, vols. 433–434, pp. 142–162, Apr. 2018.

[33] L.-F. Du and Y.-J. Niu, "Implementation of ant colony algorithm in MATLAB," *Inf. Technol.*, vol. 6, pp. 1–7, 2011.

[34] Y. Zhang, F.-Z. He, N. Hou, and Y. M. Qiu, "Parallel ant colony optimization on multi-core SIMD CPUs," *Future Gener. Comput. Syst.*, vol. 79, pp. 473–487, May 2018.

[35] M. T. Islam, P. Thulasiraman, and R. K. Thulasiram, "A parallel ant colony optimization algorithm for all-pair routing in MANETs," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, p. 8.

[36] J. Yu, R. Li, Z. Feng, A. Zhao, Z. Yu, Z. Ye, and J. Wang, "A novel parallel ant colony optimization algorithm for warehouse path planning," *J. Control Sci. Eng.*, vol. 2020, pp. 1–12, Aug. 2020.

[37] D. Thiruvady, A. T. Ernst, and G. Singh, "Parallel ant colony optimization for resource constrained job scheduling," *Ann. Oper. Res.*, vol. 242, no. 2, pp. 355–372, 2016.

[38] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen, "Finding shortest paths on terrains by killing two birds with one stone," *Proc. VLDB Endowment*, vol. 7, no. 1, pp. 73–84, Sep. 2013.

[39] P. Sewal and H. Singh, "A critical analysis of apache Hadoop and spark for big data processing," in *Proc. 6th Int. Conf. Signal Process., Comput. Control (ISPCC)*, Oct. 2021, pp. 308–313.

[40] A. Singh, A. Khamparia, and A. K. Luhach, "Performance comparison of apache Hadoop and apache spark," in *Proc. 3rd Int. Conf. Adv. Informat. Comput. Res. (ICAICR)*, 2019, pp. 1–5.

[41] X. Hong, "Basketball data analysis using spark framework and K-Means algorithm," *J. Healthcare Eng.*, vol. 2021, pp. 1–7, Jul. 2021.

[42] J. King and R. Magoulas, *2015 Data Science Salary Survey*. Sebastopol, CA, USA: O'Reilly Media, 2015.

[43] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, and S. Owen, "MLlib: Machine learning in Apache Spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[44] A. Spark, "Apache spark," *Développez en Python Pour le Big Data*, vol. 17, pp. 1–17, Jan. 2018.

[45] D. Jong, I. Kwon, D. Goo, and D. Lee, "Safe pathfinding using abstract hierarchical graph and influence map," in *Proc. IEEE 27th Int. Conf. Tools with Artif. Intell. (ICTAI)*, Nov. 2015, pp. 860–865.

[46] S. Josef and A. Degani, "Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6748–6755, Oct. 2020.
[47] A. Sinodkin, T. Evdokimova, and M. Tiurikov, "A method for constructing a global motion path and planning a route for a self-driving vehicle," *IOP Conf. Mater. Sci. Eng.*, vol. 1086, Nov. 2021, Art. no. 012003.
[48] K. Zhigalov, D. K. Bataev, E. Klochkova, O. Svirbutovich, and G. Ivashchenko, "Problem solution of optimal pathfinding for the movement of vehicles over rough mountainous areas," *IOP Conf. Mater. Sci. Eng.*, vol. 1111, Sep. 2021, Art. no. 012033.
[49] R. Masadeh, A. Sharieh, S. Jamal, M. Haj, and B. Alsaaidah, "Best path in mountain environment based on parallel Hill climbing algorithm," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 9, pp. 1–10, 2020.
[50] O. Montiel, R. Sepúlveda, and U. Orozco-Rosas, "Optimal path planning generation for mobile robots using parallel evolutionary artificial potential field," *J. Intell. Robotic Syst.*, vol. 79, no. 2, pp. 237–257, Aug. 2015.
[51] A. Ismail Ebada, I. Elhenawy, C.-W. Jeong, Y. Nam, H. Elbakry, and S. Abdelrazek, "Applying apache spark on streaming big data for health status prediction," *Comput., Mater. Continua*, vol. 70, no. 2, pp. 3511–3527, 2022.
[52] M. Belouch, S. El Hadaj, and M. Idhammad, "Performance evaluation of intrusion detection based on machine learning using apache spark," *Proc. Comput. Sci.*, vol. 127, pp. 1–6, 2018.
[53] A. Zarindast and A. Sharma, "Big data application in congestion detection and classification using apache spark," 2021, *arXiv:2101.06524*.
[54] A. Bhuvaneswari, R. Jayanthi, and A. L. Meena, "Improving crisis event detection rate in online social networks Twitter stream using apache spark," *J. Physics, Conf.*, vol. 1950, no. 1, Aug. 2021, Art. no. 012077.
[55] M. F. Aljunid and D. Manjaiah, "Movie recommender system based on collaborative filtering using apache spark," in *Data Management, Analytics and Innovation*. Springer, 2019, pp. 283–295.
[56] A. Mohan and G. Remya, "A parallel implementation of ant colony optimization for TSP based on MapReduce framework," *Int. J. Comput. Appl.*, vol. 88, no. 8, pp. 9–12, Feb. 2014.
[57] H. Alazzam, O. AbuAlghanam, and A. Sharieh, "Best path in mountain environment based on parallel A* algorithm and apache spark," *J. Supercomput.*, vol. 78, pp. 1–20, Sep. 2021.
[58] C.-S. Stan, A.-E. Pandelica, V.-A. Zamfir, R.-G. Stan, and C. Negru, "Apache spark and apache ignite performance analysis," in *Proc. 22nd Int. Conf. Control Syst. Comput. Sci. (CSCS)*, May 2019, pp. 726–733.
[59] V.-D. Hoang and K.-H. Jo, "Path planning for autonomous vehicle based on heuristic searching using online images," *Vietnam J. Comput. Sci.*, vol. 2, no. 2, pp. 109–120, May 2015.
[60] S. Basu and J. Snoeyink, "Terrain representation using right-triangulated irregular networks," in *Proc. CCCG*. Princeton, NJ, USA: Citeseer, 2007, pp. 133–136.

ESRA'A ALHENAWI received the bachelor's degree in computer engineering from the Jordan University of Science and Technology, the master's degree in computer science from Yarmouk University, in 2015, and the Ph.D. degree in computer science from The University of Jordan, in 2022. She is currently working as an Assistant Professor with the Software Engineering Department, Al-Ahliyya Amman University. Her research interests include cybersecurity, artificial intelligence especially in machine learning, and optimization problems.

RUBA ABU KHURMA received the bachelor's and master's degrees in computer science from Yarmouk University, in 2004 and 2007, respectively, and the Ph.D. degree in computer science from The University of Jordan, in 2020. She worked as a Researcher in cybersecurity and machine learning with the Department of Cybersecurity and Digital Forensics, Al-Balqa Applied University. She is currently working as an Assistant Professor with the Computer Science Department, Al-Ahliyya Amman University. She has published many articles in reputable journals, book chapters, and international conferences, such as CEC, ICPRAM, ECTA, ITISE, PICIT, and Evoapplications.

AHMAD A. SHARIEH was the Dean of the King Abdullah School for Information Technology, The University of Jordan. He has published articles in journals (24), conferences (18), and authored and prepared books (14). He gained grants for eight research projects from UJ and Europe. He developed several software systems, such as teaching sign language, e-learning modeling and simulation, and online (automated) exams. He is on the editorial board of several journals and conferences and a referee of several others. He has been supervising several master's and Ph.D. research students. His research interests include distributing systems, expert systems, e-government, e-learning, parallel processing, pattern recognition, software engineering, wire/wireless communication, modeling, and simulation.

OMAR AL-ADWAN received the bachelor's degree in computer science from Eastern Michigan University, MI, USA, in January 1987, the master's degree in computer science software engineering from The George Washington University, Washington, DC, USA, in January 1998, and the Ph.D. degree in computer science systems engineering from The George Washington University. The subject of the thesis is to increase the effectiveness of the emergency department through project engineering system analysis. He is currently the Dean of the Faculty of Information Technology, AL-Ahliyya University. His research interests include software engineering, systems engineering tools, requirements engineering, security, control, and audit of information systems and machine learning and neural networks. He passed the General Secondary School Certificate Examination Scientific Branch, in June 1982.

AREEJ AL SHORMAN received the Ph.D. degree in computer science from The University of Jordan, in 2020. She is currently working as an Assistant Professor at Petra University.

FATIMA SHANNAQ received the B.Sc. and M.Sc. degrees in computer information systems (CIS) from Yarmouk University, Jordan, in 2006 and 2009, respectively, and the Ph.D. degree in computer science from The University of Jordan, Jordan, in 2022. She is currently working as an Assistant Professor at Amman Arab University, Jordan. Her research interests include evolutionary computation, machine learning, and natural language processing in social network analysis.