Complementary Architecture of E-Learning Path Personalization and Optimization

Asare Y. Obeng and Samuel K. Opoku

***Abstract* — Using computing algorithms to generate personalized learning resources to provide the needs and improve the capabilities, preferences, and academic performance of diverse learners is creating preferred learning environments. As more learning resources, strategies and techniques are frequently added to these e-learning systems, input data to personalize the learning path has been growing exponentially making swift responses to learner's requests difficult. This study proposed a complementary learning path personalization architecture using ant colony (ACO) with nearest neighbour technique and genetic algorithm (GA) to extend the functions of the Spark framework purposely to develop a robust evolutionary computing algorithm. Experimental results indicate that complementing ACO, GA and Spark frameworks improved the generation of personalized learning resources and best-fitted-optimized learning paths. Spark-ACO took less computational time than standalone ACO. Combining ACO and GA improved the likelihood of an ant colony being trapped at a local optimum, and Spark-ACO-GA significantly enhanced the accuracy of the solutions.**

***Keywords* — Ant Colony, E-Learning System, Learning Path Personalization, Genetic Algorithm, Spark.**

I. Introduction

The ability to automate E-learning systems has created preferred learning environments for diverse learners [1]. However, designing appropriate course content to propel knowledge transfer through e-learning systems mostly lack personal support and guidance from facilitators [2] and learners' preferences [3]. Adopting a learning path clustering strategy in e-learning systems can facilitate the generation of personalized learning objects to provide the needs and improve the capabilities, preferences, and academic performance of learners. A study by Nabizadeh *et al.* in 2020 that spanned from 2009 to 2020 showed course generation as the most used learning path personalization method rather than course sequence [4]. It was therefore laudable for this study to focus on course generation mechanisms. Generally, a computational intelligence technique of evolutionary computing is suitable to provide the desired solution to the challenges of personalized learning paths as evolutionary computing seeks to optimize the objective function of scientific problems [5].

Kardan *et al.* presented a path generation method using an ant colony optimization algorithm and MapReduce framework [6]. Focusing on the course generation category, Li *et al.* presented a path generation method using a genetic algorithm and particle swarm optimization [7]. Vanitha *et al.* used an ant colony and genetic algorithm to construct a learning path in E-learning [8]. However, as more learning objects, diverse learners with preferred changing learning needs, strategies and techniques are added to these e-learning systems frequently, input data to personalize learning paths has been growing exponentially [4]. Regardless of such growth of data, learning path personalization algorithms ought to manage extensive datasets and respond swiftly to the requests of learners. This has necessitated employing a more robust algorithm or complement algorithms to efficiently generate a personalized learning path that fits the requirements of a learner.

Sachdeva *et al.* developed an Android application to personalize e-learning using peer recommendations and ant colony. The application enables learners to complete a course within a short time [9]. However, their work was inefficient and not accurate in determining the learning path of individual learners. Zhang *et al.* proposed a mechanism to simplify e-learning paths using differential evolution computing algorithms [10]. However, their work did not consider the learning performance of the learner during the learning process.

E-learning systems are creating preferred learning environments for several learners, learners prefer e-learning systems that can generate personalized learning objects, and computing algorithms are used to generate and optimize personalized learning paths. These require resolving related problems with sequencing learning objects, enhancing the local search, reducing computational activities and generating the best path. The Apache Spark-based framework has been designed to provide scalability and adaptability for handling big data and can be used in an e-learning environment to capture big data streaming [11]. This study then proposes a complementary learning path personalization architecture that incorporates ant colony and genetic algorithms to extend the functions of the Spark framework (Spark-ACO-GA). Section II of this paper discusses the problem formulation and the learning path construction mechanisms of the study. Section III looks at the Spark-ACO-GA learning path personalization architecture and its implementation. Section IV looks at the results. The conclusion of the paper is captured in Section V.

Submitted on April 20, 2023.
Published on June 13, 2023.
A. Y. Obeng, Kumasi Technical University, Ghana.
(e-mail: yaw.aobeng@kstu.edu.gh)

S. K. Opoku, Kumasi Technical University, Ghana.
(corresponding e-mail: samuel.kopoku@kstu.edu.gh)

II. Problem Formulation and Study Parameters

The learning path personalization problem results from clustering and classifying several learning resources and learner profiles (personal profile, learner preference, and learner portfolio) to address specific learner needs. The personal profile and learner preference were obtained after each learner provided such details through an online form. Learner portfolio and performance were obtained from activity logs after each learner had attempted the learning resources and assessment. Progressing to study the next level of learning resources depended mostly on the assessment results or performance of the learner. The goal was to generate the best-fitting learning paths to help learners make the best use of learning resources to improve performance.

Several studies have been conducted into finding accurate characteristics or profiles of learners that use e-learning systems to identify and cluster learning paths and ultimately improve learning activities. However, the standard characteristics [8], [12] considered in this study are personal profile, learner preference, and learner portfolio. Given that each learner is represented as Le; *Le* = {Le1, Le2, Le3 … Le n} and assigned with specific attributes. The learner profile is made up of 4 tuples. *Le* = {PP, LP, LPo, Pe}, where PP represents Personal Profile, LP represents Learner Preference, LPo represents Learner Portfolio and Pe represents Performance. Representations are given below.

- Personal Profile (PP) = {PP1, PP2, PP3, …, PPN}. Basic learner information such as id, name, age, email, username, and educational level were used.
- Learner Preference (LP) = {LP1, LP2, LP3 …. LPN}. The values of LP include learning styles {Visual, Auditory, Read/write, Kinesthetic, and Multimodal}.
- Learner Portfolio (LPo) = {LPo1, LPo2, LPo3…. LPoN}. Here, *learner goal* (objective and purpose of completing a knowledge point, a unit, a course, or a subject), and *competency* (acquired skills, experience and knowledge level) were considered.
- Performance (Pe) = {Pe1, Pe2, Pe3….PN}. Academic performance attributes included content studied, date and time, state of completion, accumulated usage time, score (range: 0 - 100), and grade were used.

To affirm a specific learning goal of a learner, learning resources that constitute learning objects (assembled into related learning content) should be developed. Learning objects facilitate learner requirements customization that makes personalization of learning content feasible [13]. The authors reflected on learning resources as learning contents that are reusable and denoted LR to their attributes such that LR = {LR1, LR2, LR3 …LR N} where LR = {DCi, TCi, Ri} are outlined below:

- Difficulty of learning content (DC) = {DC1, DC2, DC3…. DC N} represents the values {1=Easy, 2=Moderate, 3=Difficult, 4=Most difficult} at various levels.
- Time to complete (TC) = {TCi} where TCi is the time required to finish the LR.
- Requirement (R) = {R1, R2, R3 …. RN}. It shows knowledge of prior learning content required to progress to the current LR.

Thus, the fundamental elements to represent the problem include node(s) and edge(s); distance between edges of nodes; weight(s) on edge(s) and ant, the search robot. Similar to a graph, the generation of a personalized learning path is represented as *LP* = (X, Y). X represents the nodes which consist of a set of LR while Y represents the edge, a link that connects the LRs. A value V_{jk} that represents learners' scores is assigned to the edges of node *j* and node *k*. The usage data were serialized and mapped to the learning styles while results were tagged to personal profiles.

III. Spark-ACO-GA Learning Path Personalization Architecture and Implementation

A. Spark-ACO-GA Architecture

The main components of the Spark-ACO-GA architecture are illustrated in Fig. 1.

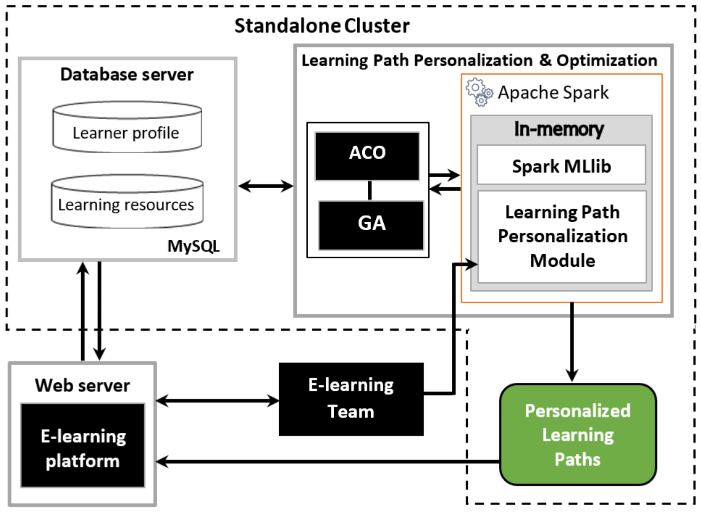


Fig. 1. Spark-ACO-GA Learning Path Personalization Architecture of E-Learning System.

The Apache web server houses the E-learning platform. The Apache web server was used since it is fast, scalable, handles simultaneous requests from browsers and runs under multitasking. The E-learning platform was Moodle. The proposed system runs in standalone cluster mode. With this, Spark runs on a single machine, precisely, the database server and uses a local file system instead of Hadoop's distributed file system. As a 3-tier system, multiuser and concurrent access from the web server is feasible; and data quality, system maintenance, improved security, backup and recovery are guaranteed. Data on e-learning can grow larger due to an increase in the number of users. It is then appropriate to place web and database applications on separate servers for eventual scalability and achieving operational efficiency.

Structured data including learning resources, learner profiles and other usage/log data on the e-learning system are extracted from the web server and converted to comma-separated values (CSV) files to facilitate the processes involved in Spark. The output is stored in MySQL (a database management system). Assigning a learning object with identification (ID) and linking it with metadata of the learning object can reduce the time required to obtain learning information. Moreover, the difficulty of learning content, the time taken to complete the learning object, and the requirement (knowledge of prior learning content) must be captured and stored in the database server. The learner profile is automatically updated anytime a learner communicates with the e-learning system. The structured data is subsequently loaded to the learning path personalization and optimization component for processing.

Complementing Apache Spark with MySQL is appropriate because Spark increases query performance due to in-memory computing (cache data into an in-memory table) [14] and utilizes all the processing units in data processing which is a major limitation of MySQL. Spark SQL as one of Apache Spark's libraries uses distributed and parallel processing approaches to process and analyse a high volume of batch and real-time data. Spark SQL is highly scalable to large jobs, fault-tolerant and uses a cost-based optimizer for faster execution of queries. Spark SQL works with structured and semi-structured data and uses Java Database Connectivity (JDBC) interfaces to connect to relational databases. Spark SQL interacts with MySQL server through JDBC to execute SQL queries and update the database with data from the path clustering system.

To improve efficiency, Zabbix, a system monitoring software is used to monitor and control elements of system performance, hosts and applications activities. Zabbix is compatible with MySQL and Spark. The high capacity and performance, ability to auto-discover added servers, network nodes and interfaces with centralized web administration make Zabbix more appropriate for the learning path personalization architecture than other system monitoring software [15].

The Spark SQL libraries load the data into the in-memory table for processing using the distributed and parallel processing approaches of the Apache Spark framework. Apache Spark is responsible for scheduling computational tasks, distribution and manipulation of data in parallel using resilient distributed datasets (RDDs). An RDD as a fault-tolerant is cached in memory to support multiple parallel operations and recovery of lost partitions.

Using the scalable machine learning library, MLlib in Spark, data is clustered using the K-Means algorithm. Genetic Algorithm (GA) is integrated with Ant Colony (ACO) to resolve related problems with sequencing learning objects, local search, computational activities and obtaining the best path. The Learning Path Personalization Module receives best-fitted-optimized learning paths and forwards them to the Personalized Learning Paths component. Then, the generated personalized learning paths are delivered to the learners to improve their capabilities, preferences, and academic performance.

B. *Building Ant Colony on Spark (Spark-ACO)*

Ant colony should perform several iterations before attaining the best results in a traversal. The main element of the pheromone matrix is updated by the best route outcomes after the completion of every iteration process. Each updated pheromone matrix is broadcast to all nodes in the cluster for use in the subsequent iteration. As a result, the Spark-ACO algorithm takes advantage of the Spark platform's sharing functionality. Through broadcasting, each node in a cluster receives the distance matrix between nodes for achieving practicable solutions.

In the ant colony algorithm, ants construct their workable solution independently for each iteration. Since each RDD in the cluster represents an ant, the ant colony is packaged by the Spark-ACO algorithm as analogous to RDD sets in individual nodes. Following that several operations are planned by Spark's capabilities to carry out RDD's transformation and operation. According to the number of cluster nodes, the Spark application is split into many partitions, and the RDD operation mode is fully parallel in each partition.

C. *Combing Spark-ACO with Genetic Algorithm*

The ACO acts as a constructive process to cluster learning objects, personal profiles, learner preferences, and learner portfolios into a learning path while GA optimizes the process of selecting the best-fitted path among the paths that ACO generates. Spark schedules computational tasks and manipulates data in parallel using RDD. Integrating Spark with ACO is necessary since the K-Means algorithm takes extra time in the initial selection stage. The sharing mechanism of the Spark platform is utilized by the Spark-ACO algorithm. Since the ant independently builds its feasible solution in each iterative process, the Spark-ACO algorithm considers the ant colony as RDD sets where each RDD represents an ant in each node in the cluster. Per the number of nodes in a cluster, partitions are created to represent the Spark program. The operations and transformational activities of RDD are carried on by a series of Spark functions in a complete parallel mode to construct optimal solutions. Each node also functions per the stages described below when GA is incorporated into a cluster:

- **Stage 1**: The outcomes of the sequence of city travelling are used as the genetic algorithm's preliminary population once the ant colony has traversed each node. The starting population of individuals is equal to the total number of ants where every single individual represents the sequences of each ant's city touring. Every city sequence matches a specific cost value while the cost value equals the genetic algorithm's fitness.
- **Stage 2**: The selection, crossover, and mutation operations are executed by a genetic algorithm.
- **Stage 3**: Where the least cost of the ant colony's tour is below the least value of the individual's fitness of genetic algorithm, and the least value among the two is below the lowest value of the previous iteration, revise chromosomal arrangements of the least individual GA with the optimized path of the ant colony. Alternately, the optimization path of the ant colony should be updated using the least distinct chromosomal sequences of GA.
- **Stage 4**: Where the iteration count is a prime digit, the key process compiles the best outcomes from more processes, computes process figures using the least value, and broadcasts the least value and route.
- **Stage 5**: The key method generates the cities' best sequence for this iteration and modifies the global pheromone matrix and certain aspects of chromosomal sequences of GA using certain routes of ant colony.
- **Stage 6**: The procedure is repeated until the algorithm reaches the desired result.

D. Spark-ACO-AG Implementation Mechanism

The optimization algorithm of the ant colony creates ant-like agents (alumni) to construct a solution while the genetic algorithm produces the best chromosomes to enhance the generation of a quality population. ACO and GA are initialized. At this point, predetermination is made for the trail value of the pheromone, heuristic facts, evaporation factor of the pheromone, and total ants of the ant colony. The number of times iteration is performed, the population size of the ants, and the probability of crossover and mutation are also initialized. The expected best solution (X%) from ACO, as well as the best and worst population (K% and L%) of GA, are initialized respectively. We denote chromosomes as the representation of a probable solution to the challenge of clustering analogous learners (alumni). Represented in binary format, a single chromosome comprises five genes [Visual, Auditory, Read/write, Kinesthetic, Multimodal …. Visual, Auditory, Read/write, Kinesthetic, Multimodal] that are considered learner characteristics.

Each ant traverses all cities; records the list of cities it has traversed, the path length between cities and the total length covered and comes back to the city it started with the traversal. As a result, the study employs a nearest neighbour (NN) technique in which ants randomly choose the next city from a group of *w* nearby cities. An ant has to choose one from the remaining cities that haven't been passed if all *w* cities have been traversed. As per the probability $p^{y}_{ij}(t)$ of state transfer, an ant will select the nearest city as it traverses [16].

$p^{y}_{ij}(t)$ shows ant *y* at *t* time and the probability between cities *i* and *j* as (1) indicates:

$$p_{ij}^{y}(t) = \int \frac{\tau_{ij}^{\alpha}(ij) \times \eta_{ij}^{\beta} \quad if\ j \in allowed\ \gamma}{\Sigma_{s \epsilon allowed_y} \tau_{ij}^{\alpha}(ij) \times \eta_{ij}^{\beta} \quad otherwise} \tag{1}$$

Group of cities in (1) is *allowed*$_y$ where ant *y* allows entry at now; $\tau_{ij}(t)$ denotes residue of pheromone between cities *i* and *j* at time *t*; $\eta_{ij}(t)$ denotes anticipated level of city *i* moving to city *j* at time *t*, the value of $\eta_{ij}(t) = 1/d_{ij}$, d_{ij} is distance between cities *i* and *j*; α denotes heuristic information factor; β denotes anticipated heuristic factor [16].

The ant colony is created, parameter variables are broadcasted, the best solution is updated, and the pheromone matrix is updated and broadcasted to each RDD that represents an ant in a node. Trial and error with iteration set at 300 were used to identify the ideal ant population that was set at 1, 25, 100, 200, and 500. Finding an optimal solution using a few ants is not possible. Likewise, increasing the ants' number consequently increases running time. Hence, setting the number of ants/alumni (n) at 300 was appropriate to find the optimal solution.

The selection of appropriate LR at the subsequent level after a learner finishes a specific LR is determined by the parameters of pheromone value, heuristic information, and node visitation information (information stored in the memory of ant to avoid looping). Learner score and total time to study LR are inputs to update pheromone value. Total time to study LR factoring ability to remember learned concept are used to determine the heuristic information. The fitness of LR is determined by the strength of the pheromone. The updated pheromone trails are a key input to the selection of fit ants to reproduce the new population in GA. With the introduction of the evaporation factor and applying updating (2), the volume of pheromone information at each edge is adjusted.

$$\tau_{(nm)}^{k\ \ new} = (1-\rho) \times \tau_{(nm)}^{k\ \ old} + \Delta\tau_{(nm)}^{k} \rho \in (0,1) \tag{2}$$

where $\tau_{(nm)}^{k\ \ new}$ represents pheromone information on the edge (*n, m*) after an update in the *k* node; $\tau_{(nm)}^{k\ \ old}$ represents pheromone information on the edge (*n, m*) before the update in the *k* node; ρ represents the pheromone evaporation information factor; and $\Delta\tau_{(nm)}^{k}$ represents updated pheromone information on the edge (*n, m*) in the *k* node.

Assessments that a learner takes after studying a concept and/or completing a course (i.e., reaching the end of a node) is/are used as input to compute a value(s) of pheromone at the edges.

GA performs the optimization process by initializing, selecting, crossover, and mutating. In a search space called population, GA starts with specific solutions. Utilizing a user-defined mathematical fitness function (3) as presented below, the fitness of each ant in a population is evaluated and stored as RDD sets in the cluster. In GA, the fitness function (performance index) is used to assess the quality of learning paths that had been generated.

$$f = \sum_{i=1}^{n} (y \times t_{(i-1)i} + (1-y) \times d_i \tag{3}$$

where *f* represents the fitness function of GA, $t_{(i-1)i}$ denotes the degree of the relation of the (*i* - 1) learning resources with *i* learning resource in the created learning path, d_i is the parameter to determine the difficulty of learning resources obtained from the metadata, *y* is a degree of learning (*f* (TimeSpent/StipulatedTime)), and *n* represents the entire number of learning resources selected for generating personalized learning path.

When all the nodes are visited, the percentage (X%) of the best solutions (best ants) of ACO is input into the mating pool of fit ants of GA to reproduce a new population of ants. Reproduction of a new population of ants occurs when the best fitness or maximum generation is not reached. At this stage, fit ants are selected to perform cross-over and mutation to generate new off-springs which are subsequently evaluated. Randomly, the roulette wheel is used to stochastically select individuals with the highest fitness to form the next generation. The probability of crossover generations was set at 0.8 to reduce the chance of an offspring having most of the characteristics of the parent. Generally, the value of the probability of crossover ranges between 0.5 and 1.0. The probability of mutation was set at 0.005 to induce diversity in the population and overcome the chance of an offspring inheriting similar qualities from the parents. Generally, the value of the probability of mutation ranges between 0.005 and 0.05. This stage injects new LR into the learning route.

Once the best fitness is reached or maximum generation is obtained, the best population (K%) updates the global pheromone value of ACO and the worst population (L%) applies to pheromone evaporation as given in (4), (5) and (6).

$$\tau_{nm} \leftarrow (1 - p)\, \tau_{nm} \tag{4}$$

For updating the global pheromone, the whole effect of pheromone updates of the entire ants in each path (n, m) is shown in (5), (6).

$$\tau_{nm}(t+1) = \rho\tau_{nm}(t) + \Delta\tau_{nm}^{k}(t) \tag{5}$$

$$\tau_{nm}(t) = \sum_{k=1}^{w} \Delta\tau_{nm}^{k}(t)\ \forall(n,m) \in \mathrm{C} \tag{6}$$

where $\tau_{nm}(t)$ and $\tau_{nm}(t+1)$ are the value of pheromone on the edge (nm) at time *t* and (*t* + 1) respectively. The increment in pheromone value is represented by $\Delta\tau_{nm}^{k}(t)$.

The following pseudocode illustrates the entire mechanism of the Spark-ACO-GA:

```
0.0 Begin
        1.0   Initialize Spark-ACO parameters
        2.0   Create ant colony
        3.0   Broadcast ant colony parameter variables
        4.0   For each node
                  4.1   For each cluster
                        4.1.1   Get ant colony from
                                RDD transformation
                                and operations.
                        4.1.2   Construct solutions
                        4.1.3   Record performance
                        4.1.4   Update local
                                performance
                  4.2   End each cluster
        5.0   End each node
        6.0   Update best solutions or generate optimal
              path length
        7.0   Update and broadcast pheromone matrix
        8.0   If all ant colony is created then
                  8.1   Go to step 14.1 with X% best ants
        9.0   Else
                 9.1  Go to step 2.0
        10.0  End if
        11.0  Generate Spark-ACO-GA initial population
        12.0  Evaluate the fitness of each ant within each
              cluster
        13.0  If reached the best fitness or maximum
              generation then
                 13.1   Return best ants
                 13.2   Update global pheromone (K%
                        best population)
                 13.3   Pheromone evaporation (L% worst
                        population)
                 13.4   If all ant colony created then
                            13.4.1   Best path obtained
                            13.4.2   Go to 16.0
                 13.5Else
                            13.5.1   Go to step 2.0
                 13.6End if
        14.0  Else
                 14.1   Reproduce new population
                 14.2   Select the fit ants
                 14.3   Do crossover
                 14.4   Perform mutation
                 14.5   Create new population
                 14.6   Go to 12.0
        15.0  End if
  16.0  Stop
```

The genetic algorithm is combined with the ant colony-based Spark algorithm to construct a solution, update the pheromone, generate population, evaluate fitness, perform genetic operations, and generate the best path. Repeatedly, the whole collaborative process of Spark-ACO-GA runs till the stipulated number of iterations are reached to create all ant colony. At this point, the best path is created.

IV. Results and Analysis of the Experiment

Designing, setting and determining specific parameters that facilitate the experimentation are performed. The intent is to test a complementary learning path clustering architecture that incorporates an ant colony algorithm with the nearest neighbour technique, genetic algorithm and Apache Spark framework. In addition, the effects of node(s) in the spark cluster on execution time, total execution time and correctness of ACO, Spark-ACO with and without nearest neighbour technique and Spark-ACO + GA are determined.

A. Setting and Determining Parameters of the Experiments

As part of setting up the environment to perform the experiments, values of pheromone trail (α), the number of ants/alumni (n), heuristic information (β), probability of mutation (P_m), pheromone evaporation coefficient (ρ), probability of crossover generations (P_c), and population size were all determined through experiments. The parameters of ρ and β were constantly held at 0.3 and 1 to 4 respectively to ascertain the value of α. β was set with total iterations of 300. Choosing 0.3 for ρ was appropriate to balance between optimal values of 0.2 and 0.6 as a low value converges quickly [8]. Initially, the parameter of α was varied between 0.1, 0.3, 0.6, 0.75, and 1.0. The value of 0.1 causes it to converge rapidly. When the deterioration constant is set to 1.0, convergence to a solution was slow. Fewer evaporation results in slower pheromone intensity degradation once a lower value of constant was set. Naturally, a high number for this element suggests that the effect of the preceding course taken will swiftly fade. The outcome will be significantly impacted by the new route. Finally, the value of α was set between 0.5 and 2 (0.5, 1.0, 1.5, 2.0), and total iterations of 300.

The probability of crossover (P_c) generations was set at 0.8 to reduce the chance of an offspring having most of the characteristics of the parent. The probability of mutation (P_m) was set at 0.005 to induce diversity in the population and overcome the chance of an offspring inheriting similar qualities from the parents. The total ants/alumni (n) set at 300 was appropriate to find an optimal solution. The nodes in a cluster of the Spark executor were configured as [1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50].

For this research, four elements including software development techniques, CASE tools, scope management, and project schedule were selected from an undergraduate course in software engineering. There were several subjects for each concept, and two themes were selected for each concept. Each subject included learning items with varying degrees of complexity and presenting style.

To make each subject more individualized for the learners, various combinations of LRs were used to convey it. Delivering the appropriate LR to the appropriate student based on those learners' adaption characteristics was essential. All students were assessed regardless of the LRs studied to comprehend the concepts. Assessments were the same for all students. The proposed architecture recommends a learning route based on the prior experiences of a learner (alumnus). Data needed for the experiments were gathered from 214 second-year computer science students studying software engineering course. The whole course was scheduled to complete in 12 weeks. Formative assessments were conducted after each theme. The students completed 20 summative assessment questions after the course to assess their understanding of all the concepts. Additional data obtained from alumni were also documented to aid the learning path personalization.

The hardware set up for the experiment included an IBM x3850 X6 Server – 4 processors, each with 20 cores, CPU clocked at 3.0 GHz, and 384 GB capacity of memory. For software, the RHEL7.6 Server operating system, spark-3.0.0-bin-hadoop3.2, and JDK 8 were used to set up the experiment's environment.

B. *Effects of Spark Cluster Nodes, Pheromone Trail and Heuristic Information on Execution Time*

A specific time cost is required for communication and cooperation across cluster nodes while the ant colony algorithm is running on a cloud computing platform. Therefore, given a certain ant colony size, adding a sufficient number of nodes to a cluster may speed up the execution of the algorithm. However, the algorithm's execution time rises as the total cluster nodes exceed a certain crucial amount. Considering these issues, experiments were designed to examine how the clustering of nodes impacts the ant colony algorithm's running time. In addition, it was appropriate to determine how long an algorithm will converge for various pheromone evaporation factors, how ants converge and avoiding the limitless accumulation of trails affect the execution time and solution.

The following ant colony algorithm settings are used in the experiment to determine the appropriate number of nodes per cluster: the total number of iterations N_c=100, values of pheromone trail α = 1.5, total ants n = 300, the pheromone evaporation coefficient = 0.5, the predicted heuristic factor = 2.0, and the heuristic information factor = 1.0. The nodes in a cluster of the Spark executor are configured as [1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]. We execute the Spark-ACO algorithm 20 times for each executor value before calculating the average time. The findings of the experiment are shown in Fig. 2 with milliseconds serving as the time unit (ms).

From Fig. 2, setting the cluster node executor to 1 or having a single node in the Spark cluster, the algorithm took the longest amount of time to run. The algorithm took less total time to run when the cluster node executor number is between 10 and 30 which confirms the work of Aharia *et al.* [17]. The Spark cluster node is then set to 20 considering an ant colony size of n = 300, which saves the maximum time.

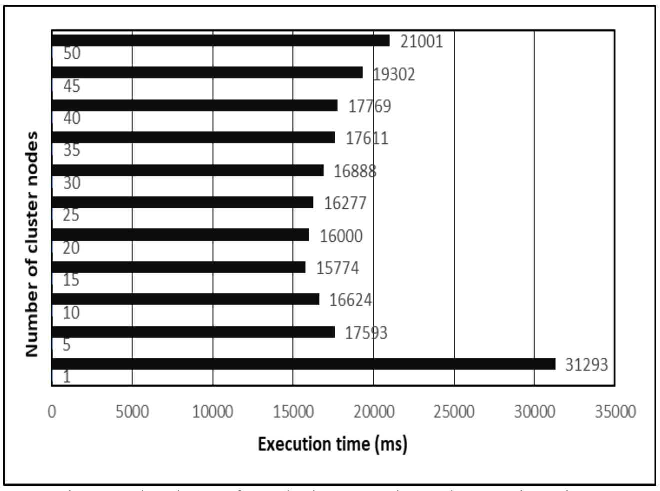
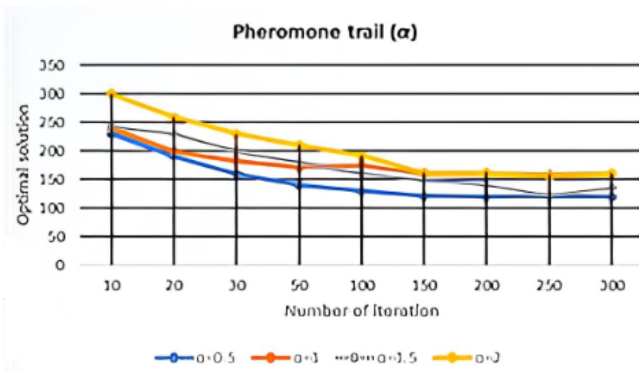
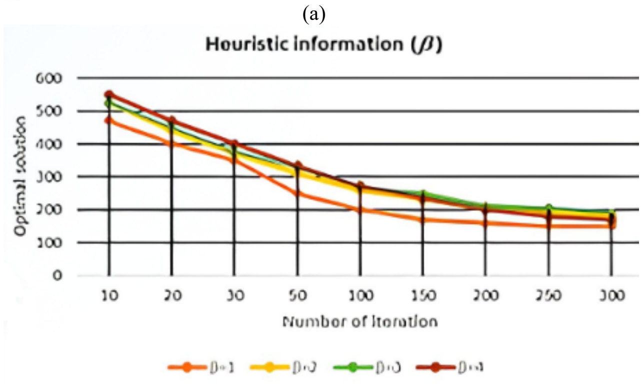


Fig. 2. Relatedness of Total Cluster Nodes and Execution Time.

Using the initial values set earlier, Figs 3a and 3b were generated.



(a)



(b)

Fig. 3. a) Pheromone trail (α); b) Heuristic information (β).

The alpha (α) value was calculated by measuring the standard of an ideal solution. The values of α were initially set between 0.5 and 2, and up to 300 iterations were carried out. Fig. 3a plots the best solution against time (in seconds) for various values of (0.5, 1.0, 1.5, 2.0). The alpha value was calculated by measuring the standard of an ideal solution. After plotting to determine the optimal solution against time, the appropriate scope of values of α was 1 and 2.

The experiment was repeated to identify the parameter of β in Fig. 3b. Before plotting to determine the optimal solution against time for β, β varied from 1 to 4 with a constant value of 1, and up to 300 iterations were carried out. After the plotting, the appropriate scope of values of β became 2 and 3.

C. Determining the Total Execution Time of ACO and Spartk-ACO

The study performs experiments to contrast the execution times of ACO and Spark-ACO algorithms to demonstrate their time effectiveness of them. Execution of the Spark-ACO algorithm was performed on the Spark platform server and the ACO algorithm in Java ran on a stand-alone server. These two platforms had the same configurations. The ant colony algorithm's settings are as follows: total iterations N_c=100, values of pheromone intensity = 1.5, total ants n = 300, the pheromone evaporation coefficient = 0.5, the predicted heuristic factor = 2.0, and the heuristic information factor = 1.0. From earlier findings, the nodes in a cluster of the Spark executor were set to 20. Additionally, the ant colony's size, given by the *n* value was in the range [100, 300, 500, 700, 900, 1100, 1300, 1500]. For each number of ant colonies, run each of the two algorithms separately 20 times before calculating the average run time.

As shown in Fig. 4, ACO only runs faster than Spark-ACO when the size of the ant colony was 100 or less. Thus, in stand-alone mode, the standard ACO algorithm may execute faster comparable to the Spark-ACO algorithm when the total number of ant colony is small. The premise is that interactions among nodes in a clustering result in a significant amount of the total time of operation once the ant colony algorithm is employed. A cluster of parallel computing will progressively start to outperform stand-alone mode as colony size increases. The Spark-ACO method uses the Spark framework to compute its calculations and packages the ant colony in a customizable distributed data RDD set. Each node in the cluster has the same amount of the ant colony RDD, which effectively utilizes the spark platform's computational memory features. Ant colonies build the best solutions via a succession of RDD conversion operations in each iteration step. Spark thus demonstrates its benefits in memory-based computing and avoiding data from landing. The Spark-ACO algorithm executes faster than ACO as the colony size increases.

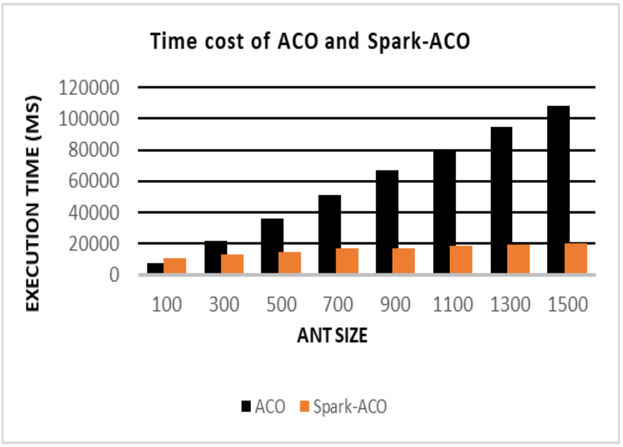


Fig. 4. Time Cost of ACO and Spark-ACO.

D. Determining the Precision of the Hybridized Complementary Algorithms

This research focused on using the nearest neighbour (NN) technique and a hybrid genetic algorithm as the main elements of the proposed complementary architecture to enhance the outcome of the simple ant colony algorithm.

The following experiment is set up to test and compare the outcomes of hybridized complementary algorithms in terms of generating quality solutions.

The ant colony algorithm's settings are as follows: total iterations N_c=300, values of pheromone intensity = 1.5, total ants n = 300, the pheromone evaporation coefficient = 0.5, the predicted heuristic factor = 2.0, and the heuristic information factor = 1.0. From earlier findings, the Spark cluster node executor was set to 20. In addition, the study chooses cases of kroA100, rat783, berlin52, d198, rd400, kroB150, and ch130 for investigations. The variants of the algorithms (Spark-ACO, Spark-ACO with NN, and Spark-ACO with NN and GA) were executed 20 times for each TSP instance, and each best outcome and the TSPLIB each algorithm produced are contrasted. The outcomes of the experiment are shown in Fig. 5.

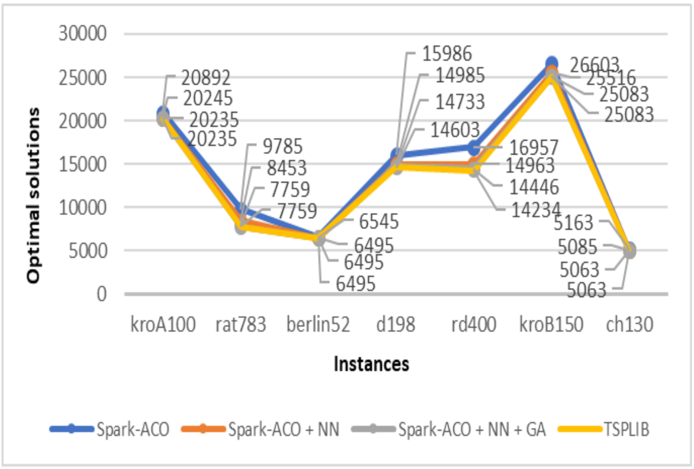


Fig. 5. Accuracy of hybridized complementary algorithms.

From Fig. 5, the operating time of the Spark-ACO algorithm greatly shortens and attains improved running speed than ACO. The Spark-ACO with NN algorithm attains improved running speed than Spark-ACO while Spark-ACO with NN and GA significantly enhanced the quality of solutions and execution time than all the other algorithms. This result confirms the observation made by Gaifeng *et al.* [16].

V. Conclusion

The proposed architecture focused on integrating the ant colony algorithm and genetic algorithm with Spark; a parallel computing framework to run in standalone cluster mode. Spark can run on a single machine and allows user-defined functions to add custom optimizations. Spark increases query performance since it is an in-memory distributed computing engine. Spark distributes and manipulates data using RDDs. RDDs are fault tolerant that are cached in memory to support multiple parallel operations and recovery of lost partitions. This distribution mechanism of Spark improves the execution of the K-Means algorithm, and computational time, and solves the issue of limited resources. Though the ant colony algorithm facilitates the clustering of data, it takes a longer time to solve large-scale problems and it is likely to be trapped at a local optimum. The ant colony generates ant-like agents to construct solutions while GA produces quality chromosomes to enhance the generation of a quality population of ants. The fundamental ant colony technique is devised and executed in Spark since an individual machine is not able to handle the computation time of huge dataset issues. Combining ACO and GA eliminated the likelihood of an ant colony being trapped at a local optimum. The execution time of the Spark-ACO significantly decreased compared to the standalone ACO system. The NN approach and GA were purposely integrated into Spark-ACO, which significantly increased the accuracy of the results. Thus, the ant colony algorithm, genetic algorithm and Spark framework are complementary to generate and optimize personalized learning paths. Our research leads us to the conclusion that the proposed complementary architecture could help improve the generation of personalized learning objects and best-fitted-optimized learning paths to enhance the performance of learners and boost technology-driven learning as preferred learning environments.

REFERENCES

[1] Obeng, AY, Coleman A. Evaluating the effects and outcome of technological innovation on a web-based e-learning system. *Cogent Education*, 2020; 7: 1836729.
[2] Choudhury S, Pattnaik S. Emerging themes in e-learning: A review from the stakeholders' Perspective. *Computers & Education*, 2020; 144: 103657.
[3] Opoku SK, Appiah S. Automating Students' Activities in Higher Educational Institutions. *International Journal of Computer Applications Technology and Research*, 2016; 5(11): 693-697.
[4] Nabizadeh AH, Leal JP, Rafsanjani HN, Shah RR. Learning path personalization and recommendation methods: A survey of the state-of-the-art. *Expert Systems with Applications*, 2020; 159: 113596.
[5] Opoku SK. A Robust Mechanism for Categorizing Context-Aware Applications into Generations. *European Journal of Electrical Engineering and Computer Science*, 2021; 5(6): 10-16.
[6] Kardan AA, Ebrahim MA, Imani MB. A new personalized learning path generation method: Aco-map. *Indian Journal of Scientific Research*, 2017; 5: 17.
[7] Li JW, Chang YC, Chu CP, Tsai CC. A self-adjusting e-course generation process for personalized learning. *Expert Systems with Applications*, 2021; 39: 3223-3232.
[8] Vanitha V, Krishnan P, Elakkiya R. Collaborative optimization algorithm for learning path construction in E-learning. *Computers and Electrical Engineering*, 2019; 77: 325-338.
[9] Sachdeva S, Singh M, Kumar N., Goswami, P. Personalized e-learning based on ant colony optimization, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* 2022.
[10] Zhang YW, Xiao Q, Song YL, Chen MM. Learning Path Optimization Based on Multi-Attribute Matching and Variable Length Continuous Representation. *Symmetry*, 2022; 14(11): 2360.
[11] Fernández-Gómez AM, Gutiérrez-Avilés D, Troncoso A, Martínez-Álvarez F. A new Apache Spark-based framework for big data streaming forecasting in IoT networks. *The Journal of Supercomputing,* 2023: 1-23.
[12] Kolekar SV, Pai RM, Pai MM, Adaptive User Interface for Moodle based E-learning System using Learning Styles. *Procedia Computer Science*, 2018; 135: 606-615.
[13] Guarino N, Oberle D, Staab S. What Is an Ontology? In *Handbook on Ontologies*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009 (pp. 1–17).
[14] ProjectPro.io. *Spark SQL for Relational Big Data Processing* [Internet], 2023 [updated 2023 Apr 13, cited 2023 Apr 17]. Available from: https://www.projectpro.io/article/spark-sql-for-relational-big-data-processing/355.
[15] Zabbix.com, Zabbix documentation [Internet], 2023 [cited 2023 April 17]. Available from: https://www.zabbix.com/documentation/current/en/manual/introduction/manual_structure.
[16] Gaifang D, Xueliang F, Honghui L, Pengfei X. Cooperative ant colony-genetic algorithm based on spark. *Computers and Electrical Engineering*, 2016: 1-10.
[17] Aharia M, Das T, Li H. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Usenix conference on hot topics in cloud computing, 1-10. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters*; 2012