Application of Ant Colony Optimisation Method

OLASUNMBO O. AGBOOLA[1], DEBORAH T. OBEMBE[1], MICHAEL C. AGARANA[2,3]

[1]Department of Mathematics,
Covenant University,
NIGERIA

[2]Department of Quality and Operations Management,
University of Johannesburg,
SOUTH AFRICA

[4]Artificial Intelligence Unit,
Canadian College of Business, Science & Technology,
CANADA

Abstract: Ant Colony Optimisation (ACO) is a meta-heuristic method that has drawn much interest in solving challenging optimisation issues. This paper examines the use of ACO to reduce the time spent on moving goods and people in Covenant University's transportation system. Covenant University is a private Christian university in Ota, Ogun State, Nigeria. Covenant University started on 21st October 2002, and since its opening, it has grown to be one of the world-class universities that have been ranked and recognised all around the globe. The data was collected from an article that computed the time per unit distance between origins and destinations within the university using Google Maps. The three traditional transportation techniques (Northwest corner method, least cost method, and Vogel's approximation method) alongside Ant colony meta-heuristics were used to get the initial basic feasible solution, which gave (361.3, 361.7, 359.8, 361.7) minutes, respectively. Then, the MODI approach was used to find the optimal solution (346.6 minutes). Also, the R programming language was used to run the Codes for the three traditional methods and the ACO method, which gave the result of solving manually. The optimal solution showed that if the time spent is reduced, the movement of people and goods would be enhanced, resulting in more time to handle important matters.

Key-Words: Ant Colony Optimisation (ACO), Transportation problem, Pheromone, Optimisation, Foraging, Meta-heuristic.

Received: April 9, 2024. Revised: March 15, 2025. Accepted: April 19, 2025. Published: July 17, 2025.

1 Introduction

ACO was developed by Marco Dorigo in 1992, and it was the first swarm intelligence-based algorithm [1]. It focuses on ants' foraging in their colony, which can be used to find the shortest route or path. Hitchcock developed the transportation problem in 1941, focusing on minimising cost and maximising deals and profit in transportation.

The application of ACO to transportation problems is a method for finding the initial basic feasible solution. It uses the behaviour of ants to find solutions to real-life issues, which makes it easier and more reliable because of ants' intrinsic nature.

A particular kind of Linear Programming Problem (LPP) called the "Transportation problem" entails transporting goods between several sources and destinations while accounting for supply and demand at each point to reduce the total cost of transportation; it is also known as the Hitchcock problem. Transportation problems can be solved using linear programming or by finding the initial essential feasible and optimal solutions. This paper focuses on the latter part of solving the transportation problem. There are two types of transportation problems, which are balanced and unbalanced.

The transportation problem can be solved in two steps. Finding the initial basic feasible solution is the first phase, and optimising the basic feasible solution achieved in the first phase is the second phase, which results in the optimal solution.

There are three methods of finding the initial basic feasible solution, which are i) the Northwest Corner Method, ii) the Least Cost Method, and iii) Vogel's Approximation Method. Finding the optimal solution is i) the Stepping Stone Method and ii) the Modified Distribution Method.

Modelled transportation problem has to do firstly with the source (i.e. where the transport is taking off from), which correlates with the supply (i.e. how much they can supply from that unit), then secondly, the destination (i.e. where the delivery or where the transport is going to stop) which correlates with the demand (i.e. what they ask for to be supplied to them). Transportation has been looked at in depth, including intelligent transportation systems that use big data to solve transportation problems [2].

Ant Colony Optimisation is a population-based meta-heuristic that can be applied to complex optimisation problems to find estimated solutions [3] [1]. An improvement of optimisation algorithms based on an ant colony's movements is known as ant colony optimisation (ACO). Artificial "ants," or model agents, navigate a bounded space representing all potential solutions to find the best answers; while examining their environment, real ants set aside the pheromones that direct their species to the resources. To aid in detecting better answers in subsequent reproduction rounds, the model 'ants' also register their places and the attributes of their solutions. The simulated ants move across the graph to gradually build solutions. The development of the stochastic solution is influenced by the pheromone model, a set of inventions related to the graph segment (either nodes or edges) whose values are modified by the ants at run-time. Numerous traditional combinatorial optimisation issues and unique optimisation problems with stochastic and active elements have shown functional positivity for ACO. Instances are the presentation of routing in communication networks and a stochastic form of the distinguished combinatorial optimisation problem, for instance, the probabilistic travelling salesperson problem. Similarly, ACO has been extended to handle infinite and mixed-variable optimisation situations. The most well-known example of an artificial/engineering swarm intelligence system with multiple applicability to real-world issues is ant colony optimisation.

Ant Behaviour: Like many other social insects, ants communicate with one another using impulsive chemical molecules called pheromones, whose concentration and bearing are discernible through their long, mobile antennae [4]. Karlson and Lüscher coined the term "pheromone" in 1959. It was derived from the Greek words pherein, which means to transport, and hormon, which means to stimulate. Social insects utilise pheromones in several ways. An example of a pheromone is the alarm pheromone, which ants release in response to danger to notify other ants in their nest and to set off behavioural alarm reactions [5].

A meal trail pheromone is a noteworthy subset of pheromones. Unlike flies, most ants are ground dwellers who use the soil's surface well to leave pheromone trails that other ants can follow as they search for food. If they select the quickest route to the food, ants will quickly return to the nest and mark their route with food trail pheromones. As more ants follow a road, it becomes increasingly appealing to other ants. This path will gradually draw other ants to follow. This auto-catalytic or optimistic response mechanism exemplifies ants' self-organising behaviour. The stigmatised way ants find the shortest route from their nest to food. The impulsive pheromone smell gradually disappears, and no fresh food pheromone tracks are muddied by returning ants after the food supply is exhausted. Ants use this unwanted response behaviour to help them adapt to environmental changes.

This paper seeks to address the transportation problem at Covenant University by employing ant colony optimisation (ACO) metaheuristics to move people and goods between multiple origins and destinations efficiently. In previous times and research, the three methods of solving transportation problems (Northwest corner method, least cost method and Vogel's approximation method) have led to either the feasible solution being far from the optimal solution or the process of getting to the possible solution have led to complication especially

the Vogel's approximation because of the long process it requires.

1.1 Literature Review

[6] used the new method (Direct Sum Method) to find the initial feasible solution in two numerical examples, which were then compared to the traditional methods (Least cost method, Northwest corner method, and Vogel's approximation method, at the end of the paper, the Direct Sum method was more efficient than the three methods.

[7] used Raval's approximation method to find the initial basic feasible solution and compared it to the three primary traditional methods (Least cost method, northwest corner method, and Vogel's approximation method).

[8] Examined how benchmark performance differs from real-world problems and showed that algorithms must be tested on both traditional benchmarks and real-world applications to generalise the results within Ant Colony Optimisation (ACO) context. ACO and its scaling dynamics with the parallel ACO designs of Parallel Ants (PA) and Independent Ant Colonies (IAC). The results showed that PA could achieve a greater solution quality in fewer iterations as the number of parallel instances increased. The three distinct hardware solutions' speeds were compared: 16 CPU cores, up to 4 GeForce GPUs, and 68 Xeon Phi cores. Using C++ and CUDA, cutting-edge ACO vectorisation methods like SS-Roulette were implemented. GPUs are deemed unsuitable for sophisticated real-world supply chain routing because of the necessary metadata access footprint, but 31 are great for routing straightforward TSPs. The research thus shows that generalised conclusions cannot be drawn from the established benchmarks.

[9] Reduced the amount of time spent travelling around Covenant University in this paper. The time and distance between locations within the campus were computed using Google Maps. Five assumptions were made, and the basic feasible solution was found using the usual methods (i.e., Vogel's approximation method, Northwest corner method, and least cost method). The optimal solution was then found using the MODI method. By the study's end, the findings aligned with what the university had anticipated.

[10] The ACO was examined and adjusted for usage in several iterations of the continuous space algorithm. Selecting the proper object (solution component) from an endless set to add to the ant's journey was critical to ACO's effectiveness when optimising across a continuous space. These stages were crucial in developing effective solutions. Significant changes were made to this component in this study, such as incorporating vector direction (Informative Pheromone) and 30 using a Gaussian distribution to estimate the expected amount of pheromone at each candidate solution component. The outcomes demonstrated that Informative Pheromone is used in any version of this algorithm to produce more accurate results than any other

[11] developed a linear weighting strategy based on the ACO algorithm to minimise the number of fixtures and the maximum completion time in the hybrid flow shop scheduling problem. The dual-objective optimisation problem could be effectively reduced to a single-objective optimisation problem. The paper used the scheduling issue of an automated precision part production line as an illustration. By the time the study ended, the suggested method's viability had been confirmed, and production scheduling and fixture resource allocation had been jointly optimised.

[12] developed a sinkhole detection technique, an enhancement of ant colony optimisation by including a hash table in the ant colony optimisation technique to advance sinkhole attack detection and reduce false alarm rate in a wireless sensor network. At the end of the paper, an increase in the detection rate of 96% was achieved, and the result outperformed other related research works when compared and further research was discussed.

However, in the application of ant colony optimisation, the time from one source to the destination would be changed to the probability of the ant passing the path, depending on the chemical" pheromone", leading to a different allocation method. Ants use pheromones to locate shorter paths. Therefore, it would aid the transporter in knowing from which origin to pick up goods and people, and which specific destination to drop off the goods and people to optimise the time. Assumptions were made by [9]:

i. Covenant University only has five origins and five destinations.
ii. Moving from one origin to a destination always takes the same time.
iii. Every route listed is always available.
iv. Identical mode of conveyance and steady speed.
v. People and products are constantly moved from their designated origins to their

designated destinations, never the other way around.

2 Problem Formulation

2.1 Ant Colony Optimisation Meta-heuristic

A meta-heuristic algorithm called ant colony optimisation (ACO) mimics an ant colony's foraging behaviour to find the quickest route to food. Ants leave behind pheromones during their food quest, which entices other ants to follow in their footsteps. The longer the path, the more pheromone is deposited. When evaporation occurs, the pheromone is evaporated, making it difficult for an ant to trace their smell, and as such, the probability of ants using the path is low. The shorter the route, the less pheromone is deposited, and the pheromone is not evaporated quickly, making the probability of ants passing that path large.

Below is the probability of choosing a particular path:

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{ij}[\tau_{ij}(t)]^{\alpha}\,[\eta_{ij}]^{\beta}} \qquad (1)$$

- Where $P_{ij}(t)$ is the probability of kth ant passing a chosen path with respect to time.
- $\tau_{ij}(t)$ is the concentration of pheromone associated with a path ij.
- $\eta_{ij} = {}^{1}/_{C_{ij}}$ is the visibility or heuristic factor favouring the path.
- C_{ij} is the cost of the path.
- α and β control the relative importance of the pheromone and the local heuristic factor.

Steps in applying ACO to the Transportation Problem:

- Step 1: Use the formula to find the probability of each distance between the two locations. Above the likelihood of an ant choosing a particular path.
- Step 2: Start allocation from the second highest probability (because we are dealing with vehicles which can cause congestion) to continue allocation till all the allocations are complete.
- Step 3: Test for degeneracy. If there is degeneracy, stop at step 4.
- Step 4: Find the total cost, i.e. the initial basic feasible solution.
- Step 5: Use the MODI method to find the optimal solution.

NOTE: In this paper, let,$\tau_{ij}(t) = 0.5$ and α, β = 1.

2.2 Transportation Problem: MODI Method

It is one method used to find the optimal solution for transportation problems.

Below are the steps in finding the optimal solution:

Table 1: MODI Procedure

STEPS	PROCEDURES
Step-1	Using the ant's movement probability, find the first basic possible solution.
Step-2	For each row and column, find U_i or V_j. To begin with • Put 0 in either U_i or V_j, depending on the maximum number of allocations in a row or column. • Using the formula $C_{ij} = U_i + V_j$, determine the other Ui′s and Vj′s for each occupied cell.
Step-3	Compute $d_{ij} = C_{ij} - [U_i + V_j]$ for every empty cell.
Step-4	• The current fundamental feasible solution is optimal and therefore terminates the process if $d_{ij} > 0$. • There is another approach with the same transportation cost but a different set of allocations if $d_{ij} = 0$. Now put an end to the process. • If $d_{ij} < 0$, then the given solution is not optimal, move to step 5.
Step-5	Choose the empty cell with the highest negative value of d_{ij} and add it to the following solution.
Step-6	From the empty cell chosen in the previous stage, create a closed path, also known as a loop. Starting from the original empty cell, the path should make a right angle and pass through the occupied cells, marking the (+) and (-) signs alternately at each corner.
Step-7	• Choose the closed path's minimal value from the cells with the (-) symbol next to them. • Give the chosen empty cell the assigned value. • In the other occupied cells indicated by the (+) symbol, add the value. • From the other occupied cells indicated by the (-) symbol, deduct the amount.
Step-8	Repeat steps 2 to 7 until an optimal solution is obtained, when $d_{ij} \geq 0$. Then

	the process ends because the optimal value has been found.

Where;

- U_i is for the rows where $i = (1,2, \dots)$
- V_j is for the rows where $j = (1,2, \dots)$
- C_{ij} is the value/ Distance in a cell.
- d_{ij} is used to locate the unoccupied cell to start the loop and process.
- Unoccupied cells have no allocations of the maximum demand and supply value.
- Occupied cells have allocations of the maximum value of the demand and supply.

Theorem 1 [13]: Let $P^*(t)$ be the probability that the algorithm finds an optimal solution at least once within the first t iterations. Then, for an arbitrary choice of a small $\epsilon > 0$ and a sufficiently large t, it holds that

$$P^*(t) \geq 1 - \varepsilon$$

And asymptotically ,

$$\lim_{t \to \infty} P^*(t) = 1$$

Proof: Due to the pheromone trail limits τ_{min} and τ_{max} we can guarantee that any feasible choice

$$P(C_k + 1 = C|T, x_k) = \begin{cases} \dfrac{\tau(C_k, C)^\alpha}{\sum_{y \epsilon C} \tau(C_k, C)^\alpha}, & if\ (C_k, C) \epsilon J_{C_k} \\ 0, & Otherwise \end{cases}$$

Is done with a probability $p_{min} > 0$. A trivial lower bound for p_{min} can be given as

$$p_{min} \geq \hat{p}_{min} = \frac{\tau_{min}^\alpha}{(N_C - 1)\tau_{max}^\alpha + \tau_{min}^\alpha} \qquad (2)$$

Where N_C is the cardinality of the set C of components. Then any generic solutions, including any optimal solution $s^* \epsilon S^*$, can be generated with a probability $\hat{p} \geq \hat{p}_{min}^n > 0$, where $n < +\infty$ is the maximum length of a sequence. Because it is enough that one ant finds an optimal solution, a lower bound for $P^*(t)$ is given by $\hat{P}^*(t) = 1 - (1 - \hat{p})^t$ by choosing t sufficiently large, this probability can be made larger than any value $1 - \varepsilon$ Because we have that

$$\lim_{t \to \infty} \hat{P}^*(t) = 1$$

2.3 Problem Statement

This paper examined ten sites at Covenant University; five are origins, and the other five are destinations. The transportation tableau is displayed in Table 3, while the names and comments of the origins and destinations are shown in Table 2.

Table 2: Origin and Destination [9]

	ORIGIN		DESTINATION
O1	Main Gate	**D1**	C.S.T
O2	Hebron Water	**D2**	C.D.S
O3	Camp House	**D3**	Library
O4	C.U. Gate	**D4**	Café 1
O5	Faith Tabernacle	**D5**	Guest House

Table 3: Transportation Tableau [9]

	D1	**D2**	**D3**	**D4**	**D5**	**Actual Distance**
O1	1.5	2.1	1.9	2.7	2.1	19
O2	2.0	2.6	2.5	3.8	2.6	25
O3	2.5	3.1	2.9	3.7	3.1	31
O4	0.8	1.4	1.2	2.0	2.0	14
O5	3.4	3.4	3.8	4.6	4.0	34
Expected Distance	17	24	23	32	24	

Each cell in the tableau represents the time it takes from one source (origin) to one destination. Consequently, the number entered in each cell represents the value of a choice variable. The current numbers in Table 3 represent the time per unit distance between one origin and one destination. Along the outside rim of the tableau are the needed distance and expected distance constraint quantity values, also referred to as rim requirements. Remember that the expected and actual distances total up to the same amount.

3 Problem Solution

3.1 Traditional Method

	D_1	D_2	D_3	D_4	D_5	Actual Distance
O_1	1.5 17	2.1 2	1.9	2.7	2.1	19/2/0
O_2	2.0	2.6 22	2.5 3	3.8	2.6	25/3/0
O_3	2.5	3.1	2.9 20	3.7 11	3.1	31/11/0
O_4	0.8	1.4	1.2	2.0 14	2.0	14/0
O_5	3.4	3.4	3.8	4.6 7	4.0 27	34/27/0
Expected Distance	17 0 - -	24 22 0 -	23 20 0 -	32 21 7 0	27 0 - -	

Figure 1: Northwest Corner Method

Test for Degeneracy: m + n − 1= Occupied Cells, Thus, 5 + 5 − 1 = 9, and no degeneracy exists.

Total time spent: (1.5 × 17) + (2.1 × 2) + (2.6 × 22) + (2.5 × 3) + (2.9 × 20) + (3.7 × 11) + (2.0 × 14) + (4.6 × 7) + (4.0 × 27) = 361.3 min

	D_1	D_2	D_3	D_4	D_5	Actual Distance
O_1	1.5 3	2.1	1.9 16	2.7	2.1	19/16/0
O_2	2.0	2.6 18	2.5 7	3.8	2.6	25/18/0
0_3	2.5	3.1 6	2.9	3.7	3.1 25	31/25/0
0_4	0.8 14	1.4	1.2	2.0	2.0	14/0
0_5	3.4	3.4	3.8	4.6 32	4.0 2	34/32/0
Expected Distance	17 3 0	24 6 0	23 7 0	32 0 -	27 2 0	

Figure 2: Least Cost Method

Test for Degeneracy: m + n − 1= Occupied Cells, thus 5 + 5 − 1 = 9, therefore there is no degeneracy. Total time spent: (1.5 × 3) + (1.9 × 16) + (2.6 × 18) + (2.5 × 7) + (3.1 × 6) + (3.1 × 25) + (0.8 × 14) + (4.6 × 32) + (4.0 × 2) = 361.7 min

	D_1	D_2	D_3	D_4	D_5	Actual Distance
O_1	1.5	2.1	1.9	2.7 19	2.1	19/0
O_2	2.0 3	2.6 22	2.5	3.8	2.6	25/22/0
0_3	2.5	3.1	2.9 23	3.7	3.1 8	31/8/0
0_4	0.8 14	1.4	1.2	2.0	2.0	14/0
0_5	3.4	3.4 2	3.8	4.6 13	4.0 19	34/21/2/0
Expected Distance	17 3 0	24 2 0	23 0 -	32 13 0	27 19 0	

Figure 3: Vogel's Approximation Method

Test for Degeneracy: m + n − 1= Occupied Cells, thus 5 + 5 − 1 = 9, therefore there is no degeneracy. Total time spent: (2.7 × 19) + (2.0 × 3) + (2.6 × 22) + (2.9 × 23) + (3.1 × 8) + (3.4 × 2) + (0.8 × 14) + (4.6 × 13) + (4.0 × 19) = 359.8 min

3.2 Ant Colony Optimisation Meta-heuristic

	D_1	D_2	D_3	D_4	D_5	Actual Distance
O_1	0.0587 17	0.0427	0.0463 2	0.0336	0.0427	19/2/0
O_2	0.0445	0.0338 7	0.0356	0.0231	0.0338 18	25/18/0
0_3	0.0356	0.0284	0.0302 23	0.0249	0.0284 8	31/7/0
0_4	0.1120	0.0640	0.0747	0.0445 14	0.0445	14/0
0_5	0.0267	0.0267	0.0231	0.0196 32	0.0222 2	34/32/0
Expected Distance	17 0 - -	24 0 - -	23 9 7 0	32 0 - -	27 9 2 0	

Figure 4: ACO Probability Tableau

	D_1	D_2	D_3	D_4	D_5	Actual Distance	U_i
O_1	1.5 17	2.1	1.9 2	2.7	2.1	19	$U_1 = 1.9$
O_2	2.0	2.6	2.5 7	3.8	2.6 18	25	$U_2 = 2.5$
0_3	2.5	3.1 24	2.9	3.7	3.1 7	31	$U_3 = 3.0$
0_4	0.8	1.4	1.2 14	2.0	2.0	14	$U_4 = 1.2$
0_5	3.4	3.4	3.8	4.6 32	4.0 2	34	$U_5 = 3.9$
Expected Distance	17	24	23	32	27		
V_j	$V_1 = -0.4$	$V_2 = 0.1$	$V_3 = 0$	$V_4 = 0.7$	$V_5 = 0.1$		

Figure 5: ACO Transportation Tableau 1

Test for Degeneracy: m + n − 1= Occupied Cells, thus 5 + 5 − 1 = 9, therefore there is no degeneracy. Total time spent: (1.5 × 17) + (1.9 × 2) + (2.6 × 18) + (2.5 × 7) + (3.1 × 24) + (3.1 × 7) + (1.2 × 14) + (4.6 × 32) + (4.0 × 2) = 361.7min

3.3 MODI Method

	D_1	D_2	D_3	D_4	D_5	ACTUAL DISTANCE
O_1	1.5 17	2.1	1.9 2	2.7	2.1	19
O_2	2.0	2.6	2.5 7	3.8	2.6 18	25
O_3	2.5	3.1 24	2.9	3.7	3.1 7	31
O_4	0.8	1.4	1.2 14	2.0	2.0	14
O_5	3.4	3.4	3.8	4.6 -32	4.0 2	34
EXPECTED DISTANCE	17	24	23	32	27	

Figure 6: First Loop Transition

	D_1	D_2	D_3	D_4	D_5	Actual Distance	U_i
O_1	1.5 17	2.1	1.9 2	2.7	2.1	19	$U_1 = 1.9$
O_2	2.0	2.6	2.5 7	3.8	2.6 18	25	$U_2 = 2.5$
0_3	2.5	3.1 22	2.9	3.7	3.1 9	31	$U_3 = 3.0$
0_4	0.8	1.4	1.2 14	2.0	2.0	14	$U_4 = 1.2$
0_5	3.4	3.4 2	3.8	4.6 32	4.0	34	$U_5 = 3.3$
Expected Distance	17	24	23	32	27		
V_j	$V_1 = -0.4$	$V_2 = 0.1$	$V_3 = 0$	$V_4 = 1.3$	$V_5 = 0.1$		

Figure 7: ACO Transportation Tableau 2

Test for Degeneracy: m + n − 1= Occupied Cells, thus 5 + 5 − 1 = 9, therefore there is no degeneracy. Total time spent: (1.5 × 17) + (1.9 × 2) + (2.6 × 18) + (2.5 × 7) + (3.1 × 22) + (3.1 × 9) + (1.2 × 14) + (4.6 × 32) + (3.4 × 2) = 360.5min

	D_1	D_2	D_3	D_4	D_5	ACTUAL DISTANCE
O_1	1.5 17	2.1	1.9 2	2.7	2.1	19
O_2	2.0	2.6	2.5 7	3.8	2.6 18	25
O_3	2.5	3.1 22	2.9	3.7	3.1 9	31
O_4	0.8	1.4	1.2 14	2.0	2.0	14
O_5	3.4	3.4 2	3.8	4.6 -32	4.0	34
EXPECTED DISTANCE	17	24	23	32	27	

Figure 8: The Second Loop Transition

	D_1	D_2	D_3	D_4	D_5	Actual Distance	U_i
O_1	1.5 17	2.1	1.9 2	2.7	2.1	19	$U_1 = 1.9$
O_2	2.0	2.6	2.5 7	3.8	2.6 18	25	$U_2 = 2.5$
0_3	2.5	3.1	2.9	3.7 22	3.1 9	31	$U_3 = 3.0$
0_4	0.8	1.4	1.2 14	2.0	2.0	14	$U_4 = 1.2$
0_5	3.4	3.4 24	3.8	4.6 10	4.0	34	$U_5 = 3.9$
Expected Distance	17	24	23	32	27		
V_j	$V_1 = -0.4$	$V_2 = -0.5$	$V_3 = 0$	$V_4 = 0.7$	$V_5 = 0.1$		

Figure 9: ACO Transportation Tableau 3

Test for Degeneracy: m + n − 1= Occupied Cells, thus 5 + 5 − 1 = 9, therefore there is no degeneracy. Total time spent: (1.5 × 17) + (1.9 × 2) + (2.6 × 18) + (2.5 × 7) + (3.7 × 22) + (3.1 × 9) + (1.2 × 14) + (4.6 × 10) + (3.4 × 24) = 347.3min

	D_1	D_2	D_3	D_4	D_5	ACTUAL DISTANCE
O_1	1.5 17	2.1	1.9 2	2.7	2.1	19
O_2	2.0	2.6	2.5 7	3.8	2.6 18	25
O_3	2.5	3.1	2.9	3.7 22	3.1 9	31
O_4	0.8	1.4	1.2 14	2.0	2.0	14
O_5	3.4	3.4 24	3.8	4.6 10	4.0	34
EXPECTED DISTANCE	17	24	23	32	27	

Figure 10: The Third Loop Transition

	D_1	D_2	D_3	D_4	D_5	Actual Distance	U_i
O_1	1.5 10	2.1	1.9 9	2.7	2.1	19	$U_1 = 0$
O_2	2.0 7	2.6	2.5	3.8	2.6 18	25	$U_2 = 0.5$
0_3	2.5	3.1	2.9	3.7 22	3.1 9	31	$U_3 = 1.0$
0_4	0.8	1.4	1.2 14	2.0	2.0	14	$U_4 = -0.7$
0_5	3.4	3.4 24	3.8	4.6 10	4.0	34	$U_5 = 1.9$
Expected Distance	17	24	23	32	27		
V_j	$V_1 = 1.5$	$V_2 = 1.5$	$V_3 = 1.9$	$V_4 = 2.7$	$V_5 = 2.1$		

Figure 11: ACO Transportation Tableau 4

Test for Degeneracy: m + n − 1= Occupied Cells, thus 5 + 5 − 1 = 9, therefore there is no degeneracy. Total time spent: (1.5 × 10) + (1.9 × 9) + (2.0 × 7) + (2.6 × 18) + (3.7 × 22) + (3.1 × 9) + (1.2 × 14) + (4.6 × 10) + (3.4 × 24) = 346.6min

Since there is no negative number, the process/iteration stops, so the total optimal time spent is 346.6 minutes.

3.4. R Program

*1 - Notepad

File Edit Format View Help

```
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> library(lpSolve)
> # Define actual_distance and expected_distance vectors
> actual_distance <- c(19, 25, 31, 14, 34)
> expected_distance <- c(17, 24, 23, 32, 27)
>
> # Define time matrix
> time_matrix <- matrix(c(1.5, 2.1, 1.9, 2.7, 2.1,
+ 2.0, 2.6, 2.5, 3.8, 2.6,
+ 2.5, 3.1, 2.9, 3.7, 3.1,
+ 0.8, 1.4, 1.2, 2.0, 2.0,
+ 3.4, 3.4, 3.8, 4.6, 4.0), nrow = 5, byrow = TRUE)
>
> # Initialize allocation matrix with zeros
> allocation <- matrix(0, nrow = length(actual_distance), ncol = length(expected_distance))
>
> # Make copies of actual_distance and expected_distance to work with
> actual_distance_copy <- actual_distance
> expected_distance_copy <- expected_distance
>
> # Apply the Least Cost Method
> while (sum(actual_distance_copy) > 0 && sum(expected_distance_copy) > 0) {
+
+   # Find the cell with the minimum time
+   min_time <- Inf
+   for (i in 1:length(actual_distance_copy)) {
+     for (j in 1:length(expected_distance_copy)) {
+       if (actual_distance_copy[i] > 0 && expected_distance_copy[j] > 0 && time_matrix[i, j] < min_time) {
+         min_time <- time_matrix[i, j]
+         min_pos <- c(i, j)
+       }
+     }
+   }
+
+  # Allocate as much as possible to the cell with the minimum time
+   i <- min_pos[1]
+   j <- min_pos[2]
+   allocation[i, j] <- min(actual_distance_copy[i], expected_distance_copy[j])
+   actual_distance_copy[i] <- actual_distance_copy[i] - allocation[i, j]
+   expected_distance_copy[j] <- expected_distance_copy[j] - allocation[i, j]
+ }
>
> # Calculate the total time
> total_time <- sum(allocation * time_matrix)
>
> # Print the allocation and total time
> cat("Allocation matrix:\n")
Allocation matrix:
> print(allocation)
     [,1] [,2] [,3] [,4] [,5]
[1,]    3    0   16    0    0
[2,]    0   18    7    0    0
[3,]    0    6    0    0   25
[4,]   14    0    0    0    0
[5,]    0    0    0   32    2
> cat("Total time:\n")
Total time:
> print(total_time)
[1] 361.7
>
```

Figure 12: R Code for Least Cost method

lastsave - Notepad

File Edit Format View Help

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> library(lpSolve)
>
> # Define actual_distance and expected_distance vectors
> actual_distance <- c(19, 25, 31, 14, 34)
> expected_distance <- c(17, 24, 23, 32, 27)
>
> # Define time matrix
> time_matrix <- matrix(c(1.5, 2.1, 1.9, 2.7, 2.1,
+ 2.0, 2.6, 2.5, 3.8, 2.6,
+ 2.5, 3.1, 2.9, 3.7, 3.1,
+ 0.8, 1.4, 1.2, 2.0, 2.0,
+ 3.4, 3.4, 3.8, 4.6, 4.0), nrow = 5, byrow = TRUE)
>
> # Initialize allocation matrix with zeros
> allocation <- matrix(0, nrow = length(actual_distance), ncol = length(expected_distance))
>
> # Apply the Northwest Corner Method
> i <- 1
> j <- 1
> while (i <= length(actual_distance) && j <= length(expected_distance)) {
+   allocation[i, j] <- min(actual_distance [i], expected_distance [j])
+   actual_distance [i] <- actual_distance [i] - allocation[i, j]
+ expected_distance [j] <- expected_distance [j] - allocation[i, j]
+   if (actual_distance [i] == 0) {
+     i <- i + 1
+   } else {
+     j <- j + 1
+   }
+ }
>
> # Calculate the total time
> total_time <- sum(allocation * time_matrix)
>
> # Print the allocation and total time
> cat("Allocation matrix:\n")
Allocation matrix:
> print(allocation)
     [,1] [,2] [,3] [,4] [,5]
[1,]   17    2    0    0    0
[2,]    0   22    3    0    0
[3,]    0    0   20   11    0
[4,]    0    0    0   14    0
[5,]    0    0    0    7   27
>
> cat("Total time:\n")
Total time:
> print(total_time)
[1] 361.3
>
```

Figure 13: R Code for Northwest Corner Method

*lastsave 3 - Notepad

File Edit Format View Help

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]
> # Define actual_distance and expected_distance vectors
> actual_distance <- c(19, 25, 31, 14, 34)
> expected_distance <- c(17, 24, 23, 32, 27)
>
> # Define time matrix
> time_matrix <- matrix(c(1.5, 2.1, 1.9, 2.7, 2.1,
+                         2.0, 2.6, 2.5, 3.8, 2.6,
+                         2.5, 3.1, 2.9, 3.7, 3.1,
+                         0.8, 1.4, 1.2, 2.0, 2.0,
+                         3.4, 3.4, 3.8, 4.6, 4.0), nrow = 5, byrow = TRUE)
>
> # Calculate the heuristic information matrix (inverse of time)
> heuristic_matrix <- 1 / time_matrix
>
> # Print the heuristic matrix
> cat("Heuristic information matrix:\n")
Heuristic information matrix:
> print(heuristic_matrix)
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.6666667 0.4761905 0.5263158 0.3703704 0.4761905
[2,] 0.5000000 0.3846154 0.4000000 0.2631579 0.3846154
[3,] 0.4000000 0.3225806 0.3448276 0.2702703 0.3225806
[4,] 1.2500000 0.7142857 0.8333333 0.5000000 0.5000000
[5,] 0.2941176 0.2941176 0.2631579 0.2173913 0.2500000
> # Define tau value
> tau <- 0.5
>
> # Multiply heuristic matrix by tau
> heuristic_tau_matrix <- heuristic_matrix * tau
>
> # Print the result
> cat("Heuristic matrix multiplied by tau:\n")
Heuristic matrix multiplied by tau:
> print(heuristic_tau_matrix)
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.3333333 0.2380952 0.2631579 0.1851852 0.2380952
[2,] 0.2500000 0.1923077 0.2000000 0.1315789 0.1923077
[3,] 0.2000000 0.1612903 0.1724138 0.1351351 0.1612903
[4,] 0.6250000 0.3571429 0.4166667 0.2500000 0.2500000
[5,] 0.1470588 0.1470588 0.1315789 0.1086957 0.1250000
> # Calculate the sum of the heuristic tau matrix
> sum_heuristic_tau <- sum(heuristic_tau_matrix)
>
> # Divide each element in the heuristic tau matrix by the sum
> normalized_heuristic_tau_matrix <- heuristic_tau_matrix / sum_heuristic_tau
>
> # Print the result
> cat("Normalized heuristic tau matrix:\n")
Normalized heuristic tau matrix:
> print(normalized_heuristic_tau_matrix)
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.05939238 0.04242313 0.04688872 0.03299576 0.04242313
[2,] 0.04454428 0.03426483 0.03563543 0.02344436 0.03426483
[3,] 0.03563543 0.02873825 0.03072819 0.02407799 0.02873825
[4,] 0.11136071 0.06363469 0.07424047 0.04454428 0.04454428
[5,] 0.02620252 0.02620252 0.02344436 0.01936708 0.02227214
>
> # Initialize allocation matrix with zeros
> allocation <- matrix(0, nrow = nrow(normalized_heuristic_tau_matrix), ncol = ncol(normalized_heuristic_tau_matrix))
>
> # Make copies of actual_distance and expected_distance to work with
> actual_distance_copy <- actual_distance
> expected_distance_copy <- expected_distance
>
> # Get unique values in the normalized heuristic tau matrix
> unique_values <- unique(normalized_heuristic_tau_matrix)
```

Figure 14: R Code for ACO Method

*lastsave 3 - Notepad

File Edit Format View Help

```
>
> # Sort unique values in descending order and remove the highest value
> sorted_values <- sort(unique_values, decreasing = TRUE)[-1]
>
> # Flag to track if allocation is completed
> allocation_completed <- FALSE
>
> # Start allocation from the second highest value to the lowest
> for (value in sorted_values) {
+   indices <- which(normalized_heuristic_tau_matrix == value, arr.ind = TRUE)
+   for (index in 1:nrow(indices)) {
+     row_index <- indices[index, 1]
+     col_index <- indices[index, 2]
+     if (actual_distance_copy[row_index] > 0 && expected_distance_copy[col_index] > 0) {
+       allocation_amount <- min(actual_distance_copy[row_index], expected_distance_copy[col_index])
+       allocation[row_index, col_index] <- allocation[row_index, col_index] + allocation_amount
+       actual_distance_copy[row_index] <- actual_distance_copy[row_index] - allocation_amount
+       expected_distance_copy[col_index] <- expected_distance_copy[col_index] - allocation_amount
+       # Check if actual_distance and expected_distance are satisfied
+       if (sum(actual_distance_copy) == 0 && sum(expected_distance_copy) == 0) {
+         allocation_completed <- TRUE
+         break
+       }
+     }
+   }
+   # If allocation is completed, break from the loop
+   if (allocation_completed) {
+     break
+   }
+ }
>
> # If actual_distance or expected_distance is not satisfied, put the last allocation on the highest value
> if (!allocation_completed) {
+   highest_value_indices <- which(normalized_heuristic_tau_matrix == max(unique_values), arr.ind = TRUE)
+   row_index <- highest_value_indices[1, 1]
+   col_index <- highest_value_indices[1, 2]
+   allocation_amount <- min(actual_distance_copy[row_index], expected_distance_copy[col_index])
+   allocation[row_index, col_index] <- allocation[row_index, col_index] + allocation_amount
+ }
>
> # Print the initial basic feasible solution
> cat("Initial Basic Feasible Solution (Allocation matrix):\n")
Initial Basic Feasible Solution (Allocation matrix):
> print(allocation)
     [,1] [,2] [,3] [,4] [,5]
[1,]   17    0    2    0    0
[2,]    0   18    7    0    0
[3,]    0    6    0    0   25
[4,]    0    0   14    0    0
[5,]    0    0    0   32    2
> # Define the time matrix
> time_matrix <- matrix(c(1.5, 2.1, 1.9, 2.7, 2.1,
+                         2.0, 2.6, 2.5, 3.8, 2.6,
+                         2.5, 3.1, 2.9, 3.7, 3.1,
+                         0.8, 1.4, 1.2, 2.0, 2.0,
+                         3.4, 3.4, 3.8, 4.6, 4.0), nrow = 5, byrow = TRUE)
>
> # Define the allocation matrix (replace this with your allocation matrix)
> allocation <- matrix(c(
+   17, 0, 2, 0, 0,
+   0, 18, 7, 0, 0,
+   0, 6, 0, 0, 25,
+ 0, 0, 14, 0, 0,
+ 0, 0, 0, 32, 2), nrow = 5, byrow = TRUE)
>
> # Calculate total time by multiplying original time matrix with allocation matrix
> total_time <- sum(time_matrix * allocation)
>
> # Print the total time
> cat("Total Time:", total_time, "\n")
Total Time: 361.7
>
```

Figure 15: R Code for ACO Method Cont.

lastsave 2 - Notepad

File Edit Format View Help

```
> # Define the time matrix, actual_distance, and expected_distance
> time_matrix <- matrix(c(1.5, 2.1, 1.9, 2.7, 2.1,
+                         2.0, 2.6, 2.5, 3.8, 2.6,
+                         2.5, 3.1, 2.9, 3.7, 3.1,
+                         0.8, 1.4, 1.2, 2.0, 2.0,
+                         3.4, 3.4, 3.8, 4.6, 4.0), nrow = 5, byrow = TRUE)
> actual_distance <- c(19, 25, 31, 14, 34)
> expected_distance <- c(17, 24, 23, 32, 27)
>
> # Vogel's Approximation Method (VAM)
> vogel_approximation_method <- function(time_matrix, actual_distance, expected_distance) {
+   allocation <- matrix(0, nrow(time_matrix), ncol(time_matrix))
+   total_time <- 0
+
+   while (sum(actual_distance) > 0 & sum(expected_distance) > 0) {
+     # Calculate penalties
+     row_penalty <- apply(time_matrix, 1, function(row) {
+       if (all(is.infinite(row))) return(-1) # Use -1 penalty for exhausted rows
+       sorted_row <- sort(row, na.last = NA)
+       return(sorted_row[2] - sorted_row[1])
+     })
+
+     col_penalty <- apply(time_matrix, 2, function(col) {
+       if (all(is.infinite(col))) return(-1) # Use -1 penalty for exhausted columns
+       sorted_col <- sort(col, na.last = NA)
+       return(sorted_col[2] - sorted_col[1])
+     })
+
+     # Determine the highest penalty
+     max_row_penalty <- max(row_penalty, na.rm = TRUE)
+     max_col_penalty <- max(col_penalty, na.rm = TRUE)
+
+     if (max_row_penalty > max_col_penalty) {
+       # Choose the row with the highest penalty
+       row_index <- which.max(row_penalty)
+       min_time_col <- which.min(time_matrix[row_index, ])
+
+       # Allocate the maximum possible to the lowest-time cell
+       allocation_amount <- min(actual_distance[row_index], expected_distance[min_time_col])
+       allocation[row_index, min_time_col] <- allocation_amount
+       total_time <- total_time + allocation_amount * time_matrix[row_index, min_time_col]
+
+       # Adjust actual_distance and expected_distance
+       actual_distance[row_index] <- actual_distance[row_index] - allocation_amount
+       expected_distance[min_time_col] <- expected_distance[min_time_col] - allocation_amount
+
+       # If actual_distance is exhausted, remove the row
+       if (actual_distance[row_index] == 0) {
+         time_matrix[row_index, ] <- Inf
+       }
+
+       # If expected_distance is exhausted, remove the column
+       if (expected_distance[min_time_col] == 0) {
+         time_matrix[, min_time_col] <- Inf
+       }
+
+     } else {
+       # Choose the column with the highest penalty
+       col_index <- which.max(col_penalty)
+       min_time_row <- which.min(time_matrix[, col_index])
+
+       # Allocate the maximum possible to the lowest-time cell
+       allocation_amount <- min(actual_distance[min_time_row], expected_distance[col_index])
+       allocation[min_time_row, col_index] <- allocation_amount
+       total_time <- total_time + allocation_amount * time_matrix[min_time_row, col_index]
+
+       # Adjust actual_distance and expected_distance
+       actual_distance[min_time_row] <- actual_distance[min_time_row] - allocation_amount
+       expected_distance[col_index] <- expected_distance[col_index] - allocation_amount
+
+       # If actual distance is exhausted, remove the row
```

Figure 16: R Code for Vogel's Approximation Method

lastsave 2 - Notepad

File Edit Format View Help

```
+
+     if (max_row_penalty > max_col_penalty) {
+       # Choose the row with the highest penalty
+       row_index <- which.max(row_penalty)
+       min_time_col <- which.min(time_matrix[row_index, ])
+
+       # Allocate the maximum possible to the lowest-time cell
+       allocation_amount <- min(actual_distance[row_index], expected_distance[min_time_col])
+       allocation[row_index, min_time_col] <- allocation_amount
+       total_time <- total_time + allocation_amount * time_matrix[row_index, min_time_col]
+
+       # Adjust actual_distance and expected_distance
+       actual_distance[row_index] <- actual_distance[row_index] - allocation_amount
+       expected_distance[min_time_col] <- expected_distance[min_time_col] - allocation_amount
+
+       # If actual_distance is exhausted, remove the row
+       if (actual_distance[row_index] == 0) {
+         time_matrix[row_index, ] <- Inf
+       }
+
+       # If expected_distance is exhausted, remove the column
+       if (expected_distance[min_time_col] == 0) {
+         time_matrix[, min_time_col] <- Inf
+       }
+
+     } else {
+       # Choose the column with the highest penalty
+       col_index <- which.max(col_penalty)
+       min_time_row <- which.min(time_matrix[, col_index])
+
+       # Allocate the maximum possible to the lowest-time cell
+       allocation_amount <- min(actual_distance[min_time_row], expected_distance[col_index])
+       allocation[min_time_row, col_index] <- allocation_amount
+       total_time <- total_time + allocation_amount * time_matrix[min_time_row, col_index]
+
+       # Adjust actual_distance and expected_distance
+       actual_distance[min_time_row] <- actual_distance[min_time_row] - allocation_amount
+       expected_distance[col_index] <- expected_distance[col_index] - allocation_amount
+
+       # If actual_distance is exhausted, remove the row
+       if (actual_distance[min_time_row] == 0) {
+         time_matrix[min_time_row, ] <- Inf
+       }
+
+       # If expected_distance is exhausted, remove the column
+       if (expected_distance[col_index] == 0) {
+         time_matrix[, col_index] <- Inf
+       }
+     }
+   }
+
+   return(list(allocation = allocation, total_time = total_time))
+ }
>
> # Solve the transportation problem
> result <- vogel_approximation_method(time_matrix, actual_distance, expected_distance)
>
> # Print the results
> print("Allocation Matrix:")
[1] "Allocation Matrix:"
> print(result$allocation)
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0   19    0
[2,]   17    8    0    0    0
[3,]    0    0   23    0    8
[4,]    0   14    0    0    0
[5,]    0    2    0   13   19
> print(paste("Total Time:", result$total_time))
[1] "Total Time: 359.8"
>
```

Figure 17: R Code for Vogel's Approximation Method Cont.

4 Conclusion

This paper used ACO to find the initial and optimal feasible solutions. It also compared ACO to other traditional methods (the Least Cost, Northwest Corner, and Vogel's Approximation methods) to find the initial basic feasible solution in the Covenant University Transportation System. The paper showed that ACO can be used as a method for finding the Initial Basic feasible solution, which was applied to the data of Covenant University and showed that the total time spent was 361.7 minutes, and then using the MODI method in finding the Optimal solution produced the total time spent was 346.6 minutes. In the comparison case, Vogel's Approximation method was the best because the total time spent (359.8 minutes) was 74 minutes less than the Least cost, Northwest corner, and ACO methods. But ACO was much easier to find the initial solution than Vogel's Approximation. Also, the application of ACO to transportation problems has covered a significant gap by not only focusing on cost but also on time and distance, and it also covers the gap of integration of campus-specific transportation policies. In this paper, the cost was not considered; therefore, the ACO method is recommended to be applied to transportation problems that consider cost, and it can also be applied. to unbalanced transportation problems and larger data on transportation problems.

References:

[1] M. Dorigo and T. Stützle, "Ant Colony Optimization and Swarm Intelligence," in *4th International Workshop*, M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, Eds., in Lecture Notes in Computer Science, vol. 3172. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 5–8. https://doi.org/10.1007/B99492

[2] O. F. Ademola, S. Misra, and A. Agrawal, "Improving Real-Time Intelligent Transportation Systems in Predicting Road Accident," *Lecture Notes in Electrical Engineering*, vol. 1011 LNEE, 2023, pp. 225–239. https://doi.org/10.1007/978-981-99-0601-7_18

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed Optimisation by Ant Colonies," 2019.

[4] Orkin, "Can Ants Smell? | Ant Behaviors & Habits | Orkin," Orkin LLC. Accessed: Mar. 19, 2024. [Online]. Available: https://www.orkin.com/pests/ants/can-ants-smell

[5] L. E. Lopes, E. T. Frank, Z. Kárpáti, T. Schmitt, and D. J. C. Kronauer, "The Alarm Pheromone and Alarm Response of the Clonal Raider Ant," *J Chem Ecol*, vol. 49, no. 1–2, Feb. 2023, pp. 1–10. https://doi.org/10.1007/s10886-023-01407-4.

[6] K. R. Ravi, G. Radha, and O. Karthiyayini, "A NEW APPROACH TO FIND THE INITIAL BASIC FEASIBLE SOLUTION OF A TRANSPORTATION PROBLEM," *International Journal of Research - GRANTHAALAYAH*, vol. 6, May 2018, Accessed: May 31, 2024. https://doi.org/10.29121/granthaalayah.v6.i5.2018.1457

[7] R. Shubham, "New Approach to Find Initial Basic Feasible Solution (IBFS) for Optimal Solution in Transportation Problem," *Open Journal of Applied Sciences* , vol. 13, Feb. 2023, pp. 207–211. Accessed: May 31, 2024. https://doi.org/10.4236/ojapps.2023.132017

[8] I. Dzalbs and T. Kalganova, "Accelerating supply chains with Ant Colony Optimization across a range of hardware solutions," *Comput Ind Eng*, vol. 147, Sep. 2020, pp. 1–14. https://doi.org/10.1016/j.cie.2020.106610.

[9] M. C. Agarana, A. E. Owoloko, and K. Adeshakin, "Enhancing the movement of people and goods in a potential world class University Using Transportation model," *Global Journal of Pure and Applied Mathematics*, vol. 12, no. 0973–1768, 2016, pp. 281–294. Accessed: Aug. 19, 2016. https://www.researchgate.net/publication/303763088

[10] Findley R., "Ant Colony Optimization for Continuous Spaces," 2016. https://scholarworks.uark.edu/csceuht/35

[11] L. Xu, J. Guo, and L. Zhu, "Ant Colony Optimisation Algorithm for the Hybrid Flow Shop Scheduling Problem with Integrated Consideration of Fixture Re-sources †," *Engineering Proceedings*, Vol.45, 2023, pp. 37. https://doi.org/10.3390/engproc2 023045037

[12] K. E. Nwankwo, S. I. Abdulhamid, J. A. Ojeniyi, S. Misra, J. Oluranti, and R. Ahuja, "A Panacea to soft computing approach for Sinkhole attack classification in a wireless sensor networks environment," In *Futuristic Trends in Network and Communication Technologies: Third International Conference, FTNCT 2020, Taganrog, Russia*, October 14–16, 2020, Revised Selected Papers, Part I 3 2021, pp. 78-87. Springer Singapore. https://doi.org/10.1007/978-981-16-1480-4_7

[13] T. St˜utzle, and M. Dorigo, "A Short Convergence Proof for a Class of Ant Colony Optimisation Algorithms," *IEEE Transactions on Evolutionary Computation*, Vol.6, 2002, pp. 358–365. https://doi.org/10.1109/TEVC.2002.802444

Contribution of Individual Authors to the Creation of a Science Article (Ghostwriting Policy)

The authors equally contributed to the present research, from formulating the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

We want to express our appreciation to Covenant University for financing the publication of this journal.

Conflict of Interest

The authors have no conflicts of interest to declare relevant to this article's content.