DOI: https://dx.doi.org/10.4314/gjpas.v24i2.11

GLOBAL JOURNAL OF PURE AND APPLIED SCIENCES VOL. 24, 2018: 215-222
COPYRIGHT© BACHUDO SCIENCE CO. LTD PRINTED IN NIGERIA ISSN 1118-0579
www.globaljournalseries.com, Email: info@globaljournalseries.com

PHEROMONE DEPOSITION/UPDATING STRATEGY IN A NETWORK: USING ANT COLONY OPTIMIZATION (ACO) APPROACH

FELIX U. OGBAN AND ROY NENTUI

(Received 24 September 2018; Revision Accepted 26 October 2018)

ABSTRACT

The study and understanding of the social behavior of insects has contributed to the definition of some algorithms that are capable of solving several types of optimization problems. The most important and challenging problems that the ants encounters when routing through a network arc, is their ability to searching for the path with a shorter length as well as to minimize the total cost incurred in the process of routing through the network. In this paper, we introduced some features to the existing Ant Colony Optimization (ACO) algorithm to help tackle this problem. First, we defined two kinds of pheromone and then we also defined three kinds of heuristic information to guide the searching direction of ants for this bi-criteria problem. Each of the ants uses the heuristic types and the pheromone types in each iteration based on the probability, controlled by two parameters. These two parameters are adaptively adjusted in the process of the algorithm. Second, we used the information of the partial solutions to modify the bias of ants so that inferior choices will be ignored. Finally, we tested the performance of the experimental results of the algorithm in an application under different Deadline constraints and the performance of the algorithm prove to be more promising, for it outperformed the performance of most of the algorithm we downloaded on line.

KEYWORDS: Ant Colony Optimization algorithm, Pheromone Deposition, Pheromone Updating strategy, Cost Minimization, Network Routing, Optimization problem.
*Corresponding Author

INTRODUCTION

The application of an Ant Colony Optimization (ACO) algorithm to a specific problem usually requires some customization of the method. Ranging from simple parameter specification to a more delicate problem modification, such as the design of operators or searching strategy. This is a crucial step for the success of the optimization (Runka, 2009). As one aims at maintaining the ability of the algorithm to perform a robust exploration of the search space (Ogban, Asagba and Owolabi, 2014), while granting it some specific information that helps to efficiently discover good and quality solution for a given problem.
Ant colony optimization (ACO) algorithm draws its inspiration from pheromone-based strategies of the foraging process of ants. Initially, it was conceived to find the shortest path, soon after it was applied to different types of combinatorial optimization problems (Dorigo and Stuzle, 2004). Example of such situations addressed includes both static and dynamic variants of academic and real world problems. Usually, the problem is mapped into a fully connected graph. When seeking for a solution, the ants deposit pheromone while traveling across the graph edges, thus creating a virtual trail (Diosan and Oltean, 2009). A solution to the given problem will emerge from the interaction and cooperation that is made by the ants.

In this paper, we propose a framework to discover the effectiveness of the pheromone updating strategies for Ant Colony Optimization (ACO) algorithm. In the approach, the algorithm seeks to fine candidate solutions that can be used by the Ant Colony Optimization (ACO) algorithm to apply the Travelling Salesman Problem (TSP) (Diosan and Oltean, 2006). We will also use different instances to access the effectiveness of the proposed approach.
Finally, this paper is structured as follows: In section 2, a general description of an Ant Colony Optimization (ACO) algorithm was presented. Section 3 presents a detailed synopsis of the system used by ants deposit pheromone and the updating strategies. Section 4 reports the experimental results and analysis. Finally, section 5 summarizes and conclusion and highlight future work.

PROBLEM DEFINITION/FORMULATION

Generally speaking, network flow application can be modeled as a Direct Acyclic Graph (DAG), G = (N, A). Let n be the number of nodes in the network with the set of nodes given as N = (n_1, n_2… nm) corresponding to the nodes on the network. And the set of arcs A represents precedence relations. And the arc is written in the form of (n_i, n_j), where n_i is called the parent node of n_i, and n_j and n_j is called the child node of n_i. Typically, a child node in a

Felix U. Ogban, Department of Computer Science Faculty of Physical Sciences University of Calabar, Nigeria.
Roy Nentui, Department of Computer Science Faculty of Physical Sciences University of Calabar, Nigeria.

network cannot be executed until the execution of its parent node has been completed. The set of parent node of n_i is denoted as pred (n_i), and the set of the child node is denoted as succ (n_i). A very good example of a network flow described by a Direct Acyclic Graph (DAG) is given in figure (1).

For the sake of convenience, we have added to the Direct Acyclic Graph (DAG) n_{start} to represent the start node and n_{end} to represent the end node. Also, for all $n_{i,}$ $1 \leq i \leq m_t$ is given, if an only if pred (n_i) is empty. We also added n_i to all succ(n_{start}), so that pred(n_i) = {n_{start}}. Similarly, anywhere we see succ(n_i) empty, we added n_i to pred(n_{end}), so that succ(n_i) = {n_{end}}.

Furthermore, each node n_i ($1 \leq i \leq n$) has an implementation domain n_i = {np_i^1, np_i^2… np_{im}} where np_{ji} ($1 \leq j \leq m_t$) represents the node implementation and m_i is the total number of available nodes implemented by n_i. Also, denoted the total cost incurred in the process of np_i^jas c_i^j.

Finally, the objective function of the scheduling problem is to find an optimal schedule {k_1,….,k_m}, which means that n_i is being executed by np_i^{kt} ($1 \leq i \leq n$), so that the total cost of the network flow incurred in the process is minimized as described in equation (1)

$$\text{Minimized cost} = \sum_{i=1}^{n} c_i^{kt} \qquad (1)$$

Moreover, the end time of the whole network flow must not be later than the given deadline constraint (D). For (D) is the deadline constraint required for use.

ANT COLONY OPTIMIZATION (ACO) ALGORITHM FOR NODE CONNECTION PROBLEM

The general idea of Ant Colony Optimization (ACO) algorithm is to simulate the foraging behavior of real ant colonies. When a group of ants set out from their nest to search for food, they deposit some chemical substance called pheromone on the path to their food source. By sensing this pheromone on the ground, other ants from the same colony can follow the path to food source discovered by the ants (Oltean, 2005). As this process continues, most of the ants tend to choose the path to food with the shorter distance knowing that there have been huge amounts of pheromone accumulated on the path. This collective pheromone deposition and pheromone following behavior of ants becomes the inspiring source of Ant Colony Optimization (ACO) algorithm. (Runka, 2009)

In this paper, we applied the Ant System (AS) algorithm which is the first Ant Colony Optimization (ACO) algorithm developed to tackle network flow problem. Informally, the algorithm can be viewed as interplay of the following procedure:

(i) **Initialization of the algorithm**. All pheromone values are parameterized and initialized.

(ii) **Initialization of ants**. Let us assume that groups of M ants are used in the algorithm. At the beginning of each of the iteration, all the ants are set to an initial state, and each ant chooses a constructive type (say forward or backward) and a heuristic type (say duration – greedy, cost – greedy or overall – greedy), based on the constructive type. Each of the ants builds its tackling sequence accordingly.

(iii) **Solution construction.** M group of ants are set to build M solution to the problem and the construction procedure includes n steps. N is the number of nodes in the network. At each step, an ant will pick up the next node in its tackling sequence and map it to one implementation outside the node implementation domain using pheromone and heuristic information (Poli, and McPhee, 2008). The algorithm also estimates the earliest start time and the earliest end time in terms of the information of partial solution built by each ant. This information is what helps to guide the ant in its searching behavior.

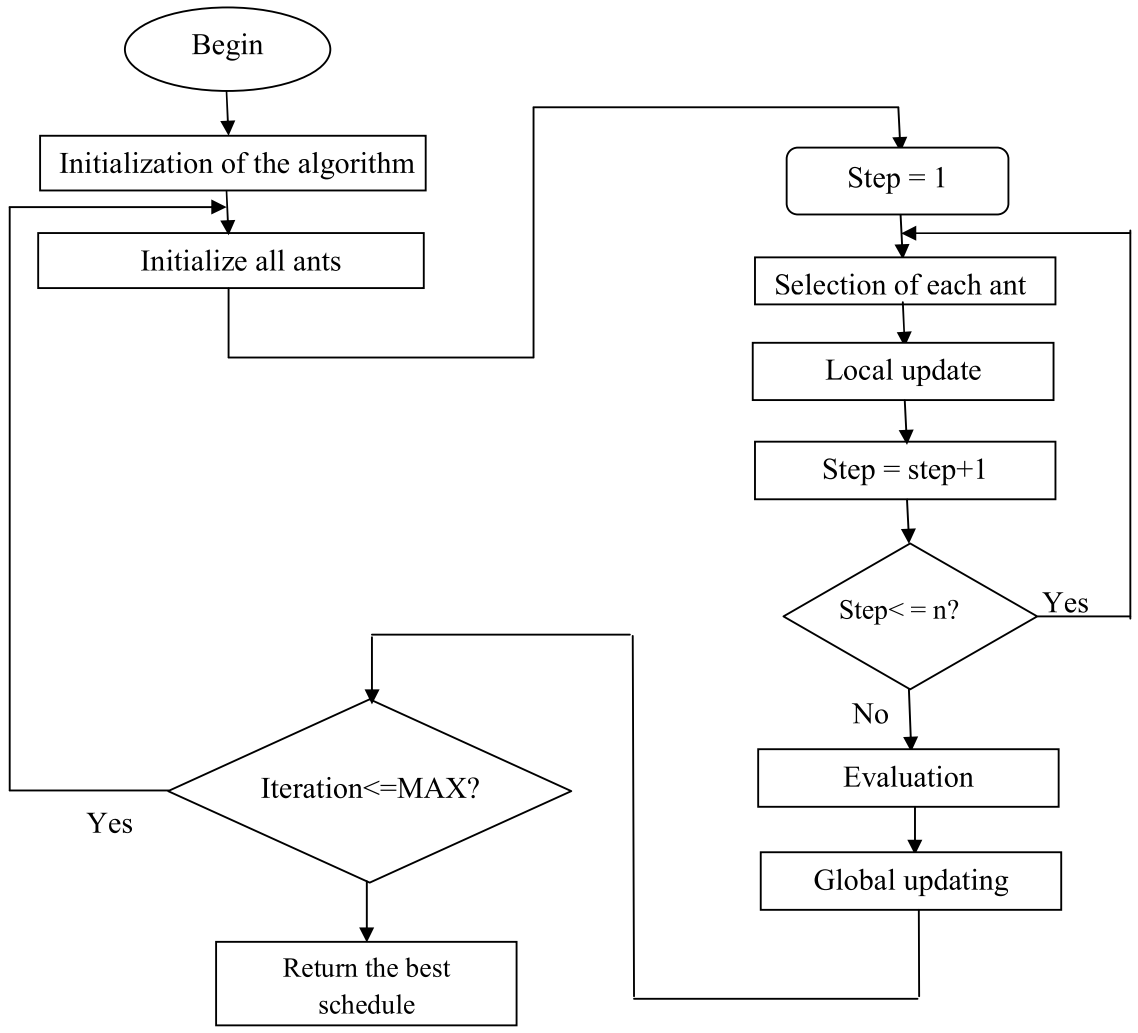
(iv) **Local Pheromone Updating.** As soon as an ant finish mapping a node n_i to np_i^j, the corresponding pheromone value is updated by a local pheromone updating rule.

(v) **Global Pheromone Update.** After all the ants have completed their construction, the global pheromone updating is applied to the best-so-far solution. The completion search time and the total cost of all the solutions are being evaluated and the pheromones that are related to the best-so-far solution is significantly increased. But in this case, some parameters are adaptively adjusted.

(vi) **Terminal Test.** If the test is passed, the algorithm will end. Otherwise, the algorithm will step up to begin a new iteration.

The flowchart of the algorithm is given in figure (1).

Fig. 1: The flowchart of the Ant System (AS) algorithm



(vii) Definition of Pheromone and Heuristic information

The problem considered is a bi-criteria problem. Therefore, it requires two types of pheromone to be used in solving the problem. One of the pheromone quantities is used for road mapping (path creation) and the other pheromone quantity is used to check the time spent (duration). We denote these two types of pheromone as $c\tau_{ij}$ and $d\tau_{ij}(1\le i \le n, 1\le j \le m_1)$. While, initializing the algorithm, all pheromone values are initialized. That is,

$$c\tau_{ij}= d\tau_{ij}, \quad d\tau_{ij}= d\tau_{o}, (1\le i\le n, 1\le j\le m_1) \qquad (2)$$

where $c\tau_{ij}$and $d\tau_{ij}$ are two parameters representing the initial values for $c\tau_{ij}$ and $d\tau_{ij}$ respectively. Similar to the Ant Colony System (ACS) for Traveling Salesman Problem (TSP), we set $c\tau_o= 1(n.\ c^{LB})$ and $d\tau_o = 1(n.d^{LB})$. Note that c^{LB} and d^{LB} are the lower bound estimation for the total cost respectively. Typically, c^{LB} can be set to the total cost when every service is set to its lowest-cost implementation and d can be set to the duration. So we have

$$c\tau_{o}=1/(n.\ \sum_{i=1}^{n}(\min_{1\le j\le m1} \quad c_i^{j}\ ) \qquad (3)$$

$$d\tau_{o}=1/(n.d) \qquad (4)$$

Note: the heuristic information for road mapping (path creation) service s_i to sp_i^j is denoted as η_{ij}. Finally, we used three different kinds of heuristic information to guide the searching direction of ants namely: duration-greedy, cost-greedy and overall-greedy. However, the definition is given below

$$\eta_{ij} = \begin{cases} {}^{1}\!/\!{}_{d_i^{j}} & \text{if selection type} = \text{duration} - \text{greedy} \\ {}^{1}\!/\!{}_{c_i^{j}} & \text{if selection type} = \text{cost} - \text{greedy} \\ {}^{1}\!/\!{}_{c_i^{j}.}\ \ d_i^{j} & \text{if selection type} = \text{overall} - \text{greedy} \end{cases} \qquad (5)$$

Following this definition, duration–greedy heuristic bias the implementation with the shortest execution time, Cost–greedy heuristic prefers the length with low-cost and the overall-greedy considers the both factors.

(viii) Initialization of Ants

At the beginning of each iteration, all ants are initialized and each ant chooses its type from any of the selection rule. Either from duration-greedy, cost-greedy or from the overall-greedy according to equation (6)

$$\text{Selection type} = \begin{cases} \text{duration} - \text{greedy}, 0 \le \text{ran} \le \ \ p_1 \\ \text{cost} - \text{greedy}, \ \ p_1 \ \ \le \text{ran} \le \ \ p_2 \\ \text{overall} - \text{greedy}, \ \ p_2 \ \ \le \text{ran} \le 1 \end{cases} \qquad (6)$$

Where p_1 and p_2 $((0 < p_1 \ \le p_2 \ < 1)$ are two parameters and ran∊ [0, 1] is a random number. Apparently, the probabilities of choosing duration-greedy, cost-greedy, and the overall greedy are p1, (p2,-p1) and (1-p2) respectively. The selection type of an ant is corresponding to the type of heuristic information it used while constructing a solution.
Furthermore, each ant has to select its constructive type of solution randomly (from either the forward or the backward ants) and builds its sequence of services. The tackling sequence is built following a simple illustration of a network flow application with 9 tasks is given below.

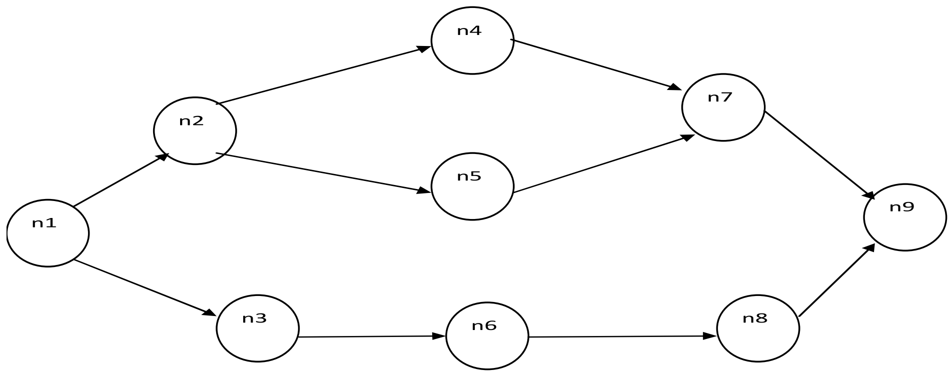


Fig.2: A simple illustration of a network flow application with 9 tasks.

Also, forward ant begins from the start $node_{start}$ and applies a random depth-first search to orderly connect all the nodes. For example, the possible sequence build by a forward ant in e-Economic workflow given by fig. 2 are (n1.n2.n4.n7.n9.n5.n3.n6.n8), (n1.n2.n5.n7.n9.n4.n3.n6.n8), (n1.n3.n6.n8.n9.n2.n4.n7.n5) and (n1.n3.n6.n8.n9.n2.n5.n7.n4), similarly, a backward ant begin its searching from the end node and uses a random backward depth-first search to orderly connect the nodes. The possible sequence build by the backward ant in the above example are (n9.n7.n4.n2.n1.n5.n8.n6.n3), (n9.n7.n5.n2.n1.n4.n8.n6.n3) (n9.n8.n6.n3.n1.n7.5n.n2.n4) and (n9.n8.n6.n3.n1.n7.n4.n2.n5)
The reason for using a depth-first search scheme is the information of the partial solutions (that is the earliest start

probabilities of selecting inferior components. The reason for constructing the tackling sequences from both sides (forward and backward) is to diminish the influence exerted by the relative order of nodes.

(ix) Solution Construction

After initialization, M ants set out to build solutions to the problem in a parallel order according to their tackling sequence. In step k (1.k.n), each ant picks up the kth node from its tackling sequence and map it for an implementation out of the node implementation domain. Assume that an ant is choosing one node out of $sp_i=\{sp_i^1, sp_i^2\ sp_i^3,\ldots\ldots sp_i^m\}$ to map to s_i, the selection rule is as follow:
Step 2: Evaluate the overall bias desirability of all implementation in terms of equation (6)

$$B_{ij} = \begin{cases} (dT_{ij})^{\alpha} \quad (\eta_{ij})^{,\beta} \quad \eta_{ij} = {}^{1}/_{d_i{}^j} \\ \text{if the selection type of ant is duration} - \text{greedy;} \\ (cT_{ij})^{\alpha} \quad (cT_{ij})^{,\beta} \quad \eta_{ij} = {}^{1}/_{c_i{}^j} \\ \text{if the selection type of ant is cost} - \text{gree dy;} \\ (cT_{ij})^{\alpha} \quad (cT_{ij})^{,\beta} \quad \eta_{ij} = {}^{1}/(c_i{}^j \ . \ d_i{}^j), \\ \text{if the selection type of the ant is overall} - \text{greedy;} \end{cases} \qquad (7)$$

Where B_{ij} represents the bias mapping of n_i to $np_{ij}(1\leq j\leq m_i;)$and are two parameters that determine the weight of the pheromone and the heuristic information respectively.
Step 2: Adapt the values of B_{ij} in terms of the information gotten from the partial solution. The earliest start time and the earliest end time of the node can be estimated for the current partial solution build by an ant. We denote the earliest start time of n_i as n_i.est and the earliest end time of n_i as n_i.eet. As the tackling sequence is built by depth-first search, it guarantees that a node is only considered by a forward ant until one of its parent nodes is considered. Similarly, a node is only considered by a backward ant until one of its child node is considered. We present a clear discussion of the situation with forward ant in the following text. However, the situation with the backward ant comes when regarding all parent nodes as child nodes and when regarding all children nodes as parent node.
Therefore, the forward ant which is considered as n_i n_i.est can be estimated as follow

$$n_i.\text{est} = \max n_{k\in \text{pred (nt)}}\ \text{nk.eet} \qquad (8)$$

For example, a forward ant uses the sequence of (n1. n2.n4.n7.n9.n5.n3.n6.n8) to build a solution for the workflow given in figure 2. After mapping all the nodes of the first branch, (n1.n2.n4.n7.n9) to create a corresponding implementation, we can then estimate n2.est = n1.eet, n4.est = n2.eet, n7.est = n4.eet, n9.est = n7.eet. Also, when considering the next node n5, we have n5.est = n2eet. On the other hand, it is also important to know that sometimes the earliest start time for the child node of n_i may also have been estimated. For instance, when considering n5 in the same example given in the last paragraph written as n7.est (n7 will be the son of n5) because it has already been estimated. However, because the available time slot for n5 is limited by n2.eet and n7.est. We can then define slot as

$$\text{Slot}_i = \begin{cases} \text{undifined,} \quad \text{if} \forall \text{ ni} \in \text{succ } n_i \\ \text{nk. est has not been evaluated} \\ (\min_{\text{nk} \in \text{succ.(ni)and nk.est has been evaluated}} \quad \text{nk. est}) \quad - \quad \text{ni. est} \\ \text{otherwise} \end{cases} \qquad (9)$$

Based on this definition, if n_i is mapped to sp_i^j to satisfy d_i^j> $slot_i$, then si.eet = si.est + d_i^j will be larger than at least one of its child's estimated earliest start time. In this situation, the estimated earliest start time for all the child node of n_i must be updated to be at least not smaller than n_i.est. Otherwise, for all the implementations that will satisfy $d_i^j \leq slot_i$, will only be successful with the one with the lowest cost because all other choices will result to a higher cost solution with the same path created. Therefore, the ants will ignore these inferior choices by modifying the preferences B_i^j using equation (10)

$$B_{ij} = \begin{cases} B_{ij,} \\ \text{if } d_i{}^j \ > \ \text{slot}_i \ \text{ or slot}_i \ = \ \text{udifined} \\ \dfrac{B_{ik}}{\sum_{\forall\ np_i{}^k}\ (d_i{}^k \ \leq \ \text{slot}_i)} \\ \text{if } d_i{}^j \ \leq \ \text{slot}_i \ \text{ and } c_i{}^j \ = \ \min_{\forall\ np_i{}^k\ (d_i{}^k \ \leq \ \text{slot}_i)} \ c_i{}^k \\ 0 \\ \text{if } d_i{}^j \ \leq \ \text{slot}_i \ \text{ and } c_i{}^j \ \min_{\forall\ np_i{}^k\ (d_i{}^k \ \leq \ \text{slot}_i)} \ c_i{}^k \end{cases} \qquad (10)$$

Step 3: An ant selects one implementation out of the following implementation
$Sp_i = \{sp_i^1, sp_i^2\ldots sp_i^m\}$ and map it to s_i in terms of the following selection rule:

$$n_i = \begin{cases} \arg\max_{\forall np_i/(1\leq j\leq m_1)} \ B_{ij,} \quad \text{if } q \ \leq q_0 \\ \text{routlette wheel scheme, otherwwise} \end{cases} \qquad (11)$$

$$p_i^j = \frac{B_{ij}}{\sum_{k=1}^{m_1} B_{ik}} \qquad (12)$$

equation (11) shows the pseudo random proportion selection rule. In this rule, a random number q∈[0.1] is generated and is compared to a parameter $q_o(q_0 \in [0,1])$. Only if $q \le q_0$, that the implementation sp_i^j with the largest value of B_i^j is chosen. Otherwise, a roulette wheel scheme is used. Also, the probability of mapping n_i to np_i^j is given by equation (12). In other words, the probability of selecting np_i^j is directly proportional to the value B_i^j

(x) Local Pheromone Updating

Immediately after an ant maps np_i^j to n_i, local pheromone updating procedure is implemented. The updating rule is given by equation (13)

$$\begin{cases} dT_{ij} = (1-\gamma) \cdot dT_{ij} + \gamma \cdot dT_{0,} \\ \text{if selection type is duration} - \text{greedy;} \\ cT_{ij} = (1-\gamma) \cdot cT_{ij} + \gamma \cdot cT_0, \\ \text{otherwise} \end{cases} \qquad (13)$$

Where $\boldsymbol{\gamma} \in [0,1]$ is a parameter value. The function of local pheromone update is to decrease the pheromone value corresponding to sp_i^j so that the following ants will have a higher probability to choose other implementation. Also, Local pheromone updating procedure enhances the diversity of the algorithm

(xi) Global pheromone updating

Global pheromone updating takes place after all the ants have built their solutions. Global pheromone updating only applies to the components on the best-so-far solution (Tavares and Pereira, 2006). Assume that the best-so-far solution is $\{k_1, k_2, \ldots, k_n\}$, which means that n_i is being executed by sp_i^{k1} $(1 \le i \le n)$. the cost and searching space of the best-so-far solution is denoted as $cost^{bs}$ and search $space^{bs}$. Hence, the global pheromone updating rule is given by equation (14)

Setting of parameters and characteristics of the algorithm

The parameters of the algorithm are set apriory as follows: The weight of pheromone and the heuristic information in equation (14) are set to $\boldsymbol{\alpha}=1$ and $\beta=0.45$, the probability of setting the implementation with the largest value is $q_o=0.8$.

$$P_{i,j} = \frac{[T_{i,j}]^{\alpha} \cdot [n_{i,j}]^{\beta}}{\sum_{i,j\,CA} [T_{i,j}]^{\alpha} \cdot [n_{i,j}]^{\beta}} \qquad (14)$$

Local pheromone updating rate is set at $p_L=0.1$, Global pheromone updating rate $P_G = 0.1$. In all the experiment, the total iteration number is set to 200 and the number of ants is set at the range of 100 - 500. We configured these parameters basically according to the Ant Colony System (ACS). The experimental result shows that these results still have good performance.

Table.1. Performance Comparison of the best solution between the strategies, with 200 iterations

Strategies	Iteration	Best Tour Length	Mean Best Fitness	Deviation	Branching
AS	200	198.00	159.10	7.46	5.25
EAS	200	199.72	147.37	9.76	3.54
RANK-BASE	200	189.40	146.17	8.83	2.77
MIN-MAX	200	199.00	143.00	9.50	2.50
DANTE	200	200.00	145.70	10.58	2.95

We compared the approach we proposed with other algorithms to tackle the routing problems. First, the algorithm works by dividing the Direct Acyclic Graph (DAG) into partitions and distribute sub-deadline to each partitions. The decision process is applied to find the best solution.

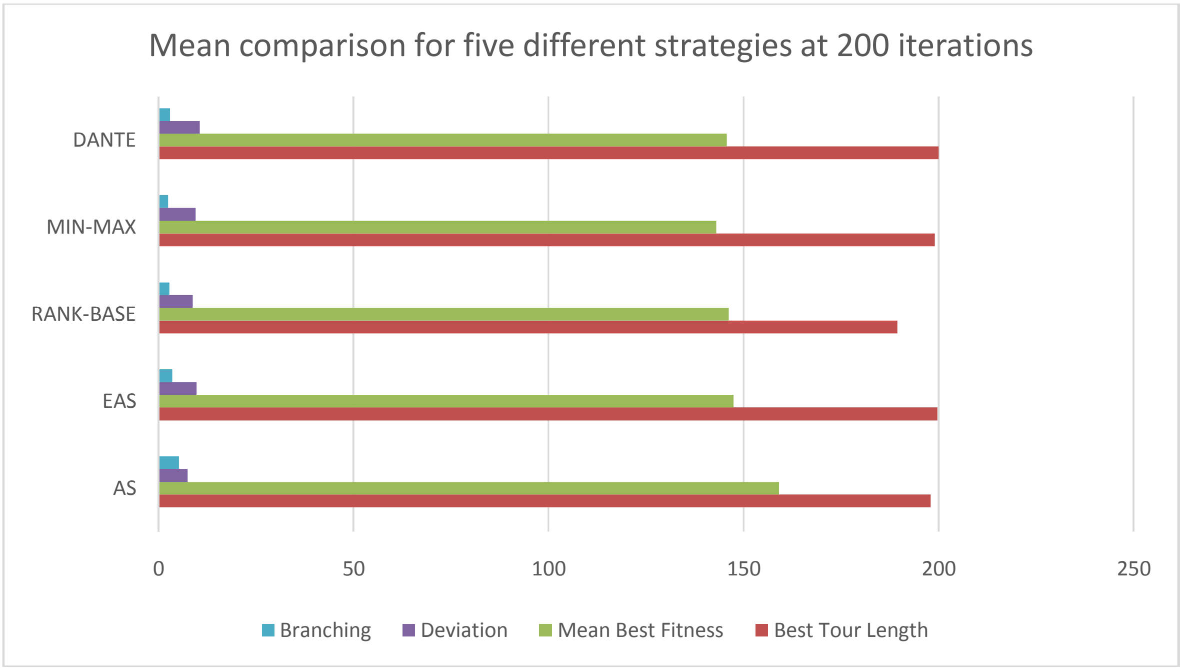


Figure 3: Mean Comparison for five different strategies at 200 iterations.

A clearer picture in fig. 4 of the closeness between the RANK-BASE strategy and the DANTE with respect to their Mean Best fitness and a slim variation from that of Min-Max is shwon.

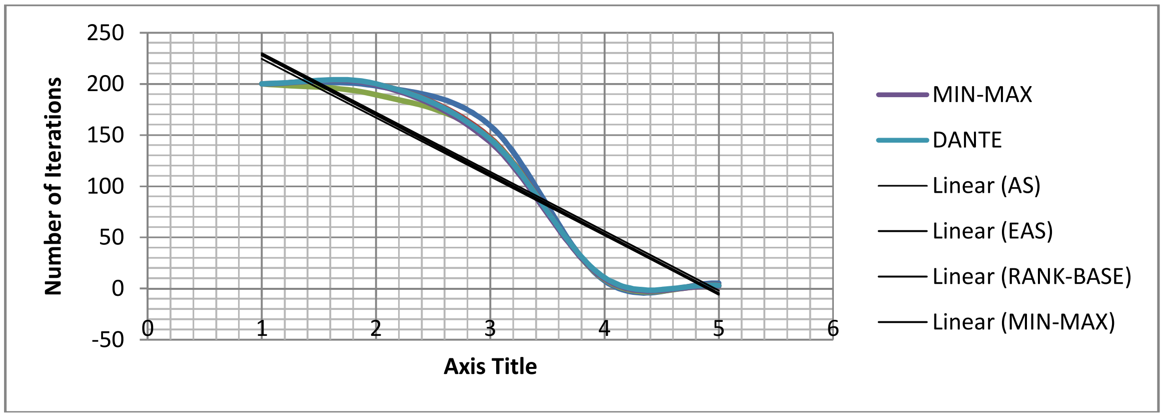


Figure 4: Best Tour Length and Mean Best Fitness compared.

The result obtained in the network flow application as illustrated in figures 3 and 4, can be seen that the performance of the algorithm outperformed other algorithm proposed in the literature. The result obtained was able to meet the deadline constraints and was able to make good use of the time to minimize cost.

REFERENCES

Diosan, L., and Oltean, M., 2009 Evolutionary design of evolutionary algorithm. Genetic programming and Evolving Machines (10) 263 – 306

Diosan, L., and Oltean, M., 2006 Evolving the structure of the particular swarm Optimization algorithms. In: EVOCOP Proceedings. PP. 25 – 36

Dorigo, M Stutzle, T., 2004 Ant Colony Optimization (ACO). MIT press

Ogban F. U, Asagba P.O, and Olumide O., 2014 An Efficient Clustering System for the measure of page (Document) Authoritativeness. Journal of Information Engineering and Application. ISSN 2224-5782 (print) ISSN 2225.0506 (online) Vol. 4. No. 6 2014.

Oltean, M., 2005 Evolving evolutionary algorithms using linear genetic programming. Evolutionary computation Journal 13, 387 – 410

Poli, R., Langdon, W.B., and MCPhee, N.F., 2008 A field guide to genetic programming. Published via http://LuLu.com and freely available at http://www. gp-field-guide.org.uk (with contribution by J.R. Koza)

Runka, A., 2009 Evolving and edge selection formula for Ant Colony Optimization (ACO) algorithm. In: GECCO proceeding. PP. 1075 – 1082

Tavares, J., Pereira, and F.B., 2006 Evolving Strategies for updating pheromone trails: a case study with the Traveling Salesman Problem (TSP). In: PPSN XI proceedings. Lecture note in Computer Science, Vol. 6239, pp. 523-532