Advanced Mathematical Models & Applications
Vol.5, No.1, 2020, pp.95-110

Jomard Publishing

A HYBRID OPTIMIZATION ALGORITHM OF ANT COLONY SEARCH AND NEIGHBOUR-JOINING METHOD TO SOLVE THE TRAVELLING SALESMAN PROBLEM

Warif B. Yahia[1*], Mohammed W. Al-Neama[2], Ghassan E. Arif[1]

[1]College of Education for Pure Sciences, Tikrit University, Tikrit, Iraq
[2]Education College for Girls, Mosul University, Mosul, Iraq

Abstract. In this paper, we are investigating an optimal solution of a typical Travelling Salesman Problem (TSP). Indeed, the results of the TSP consist of costs and complexity of time and space. In fact, many researchers have been working on solving the TSP and trying to reduce the complexity of this problem for the purpose of reaching an optimal solution efficiently. Therefore, we are going to structure a hybrid algorithm that consists of the Ant Colony Optimization method (ACO) combined with the Neighbour Joining method (NJ) and call it a Neighbour-joining Ant Colony Optimization method (NACO) to get accurate and efficient results. In addition, a variety of benchmarks from real-life problems will be taken as instances and implemented in our hybrid model using the MATLAB® platform in order to examine how our proposed method compares with the standard ACO method.

Keywords: Ant Colony Optimization, Travelling Salesman Problem TSP, discrete optimization, ACO, NACO, NJ method.
AMS Subject Classification: 49M30.
Corresponding author: Warif Yahia, College of Education for Pure Sciences, Tikrit University, Tikrit, Iraq, Tel.: +9647701667778, e-mail: warifb@gmail.com
Received: 20 November 2019; Revised: 20 March 2020; Accepted: 5 April 2020; Published: 29 April 2020.

1 Introduction

The travelling salesman problem (TSP) is one of the very important problems in the field of computational and applied mathematics regarding its applications in the real-life problems. In addition, it was classified as one of the most complicated problems because it falls within the complexity class of the NP-complete Du & Ko (2011). The complexity of this problem has been inspiring many researchers to find and modify new ways to reduce the complexity of this problem and trying to reach an optimal solution efficiently. Indeed, there are many algorithms have been used to solve this problem such as Particle Swarm Optimization (PSO) Shi et al. (2007), Genetic Algorithm (GA) Razali & Geraghty (2011), Monarch Butterfly Optimization (MBO) Wang et al. (2016), However, some of these algorithms were able to give an exact solution, while some others were able to give an approximate solution.

Basically, the TSP as shown in Fig. 1 can be defined and explained as there are number of cities that the seller is required to visit in a shortest possible way in order to reduce the cost of the trip and then return to the city from which he started. The importance of this problem is well brought out in many fields and applications in the real-life such as in the manufacture of electrical appliances circuits, the construction of roads between cities, aircraft guidance and drones, DNA computing, and many other applications. Moreover, TSP can be linked to many field in computer science and mathematics such as Eiegenproblems Ali (2017), Neural Networks Yamada et al. (1993), Projective Geometry Yassen et al. (2019).

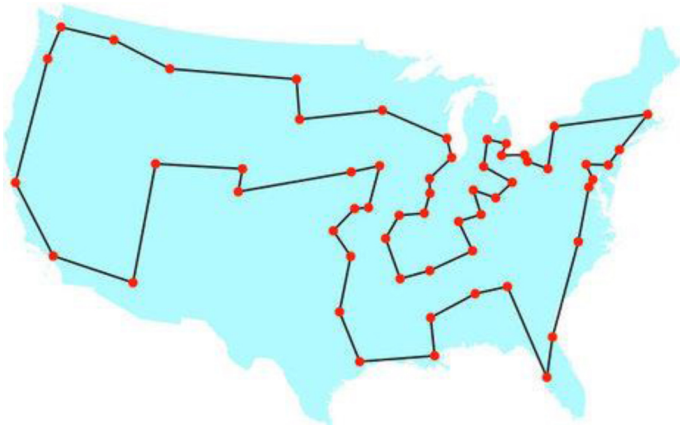


Figure 1: TSP Example

The TSP can be classified into three types such as symmetric TSP, asymmetric TSP, and dynamic TSP. As we mentioned earlier, in the computational complexity theory, this problem has been classified as an NP-complete problem, which means if there are n cities in a complete graph then there are (n-1)! Possible different tours for the seller. If n is a large number, then the classical method takes a huge amount of time to find the optimal solution. Therefore, researchers in the field of artificial intelligence always seek to find new ways to solve this problem and try to reduce its complexity. Scientists have simulated the behaviour of many living creatures that have less intelligence than human beings such as ants, bees, birds, and many other animals that live in swarms Farisi et al. (2016). This simulation helped the researchers to solve many complicated problems, including the travelling salesman problem.

The rest of this paper consists of the following sections. In Section 2, hybrid methods of ACO and other related work have been mentioned. In Section 3, we are going to mention the methods and techniques of the TSP, ACO, NJ and our proposed method NACO. Finally, we are going to apply and implement our proposed algorithm and see how it compares with the standard ACO in the experiments and the results analysis in Section 4.

2 Related Work

Regarding the studies that are conducted on the algorithm of the ant colony and its uses in solving the travelling salesman problem, Gong & Ruan (2004) introduced a hybrid algorithm consisting of an ant colony algorithm and a genetic algorithm, and they concluded that their algorithm increases the accuracy of the solution as well as reduces the travelling salesman problem search space.

In Duan & Yu (2007) combined the ant colony algorithm with the Metric Algorithm (MA), which was used to improve the selection of parameters in the ant colony algorithm, and they concluded that this method is more efficient and accurate.

A new method was presented in Chen & Chien (2011) which was called the genetic simulated annealing ant colony system with particle swarm optimization techniques. They concluded that their proposed algorithm has better solutions after comparing it with many algorithms.

Junqiang & Aijia (2012), were able to combine the ant colony algorithm and the delete-cross method to conclude that the resulting algorithm is better than six types of the original ant colony algorithm.

Girsang et al. (2014) introduced a new method deriving from hybridization the Bee Colony Algorithm BCO with the ant colony algorithm ACO, and they called it Ant Bee Colony Optimization (HABCO), and they concluded that their new algorithm resulting from the hybridiza-

tion overcame the ACO and the BCO in reaching the solution of travelling salesman problem.

A hybrid of Cuckoo Search algorithm and Ant Colony algorithm was proposed by Kumar & Tiwari (2015), and they compared it with the original ant colony algorithm and they found that the hybrid algorithm is more efficient than the original ant colony algorithm in solving the problem.

Mahi et al. (2015) found a new method consisting of using 3-Opt algorithms, Particle Swarm Optimization (PSO), and ant colony optimization. Their results show that the new method is better than many methods in terms of the performance by using less number of ants cities.

In Farisi et al. (2016) modified the ant colony algorithm ACO by combining it with the Firefly Algorithm (FA), and compared their results with the original ACO and FA. They found that the new method gets closer to the solution efficiently in a short time.

Gülcü et al. (2016) proposed a parallel hybrid algorithm by combining 3-Opt algorithm and ant colony optimization ACO, and they called it PACO-3Opt. They compared the results with PSO-ACO-Opt, and concluded that PACO-3Opt has higher efficiency in reaching the solution because it works in parallel, which reduces the time that is spent on mathematical calculations within the algorithm.

An improvement on ACO by updating the pheromone from the results of 2-Opt has been done by Ratanavilisagul & Pasaya (2018). The proposed method is called Modified Ant Colony Optimization with updating Pheromone by Leader and Reinitialization (MACO-LR). From the experiments in their paper on many instance of TSP, they found that the proposed algorithm overcomes ACO, ACO-2OPT, and PACO-3OPT in terms of the accuracy of the solutions.

A new hybrid algorithm has been proposed by Rokbani et al. (2019). They combine ACO with Gravitational Particle Swarm Optimization (PSOGSA), and called it Gravitational Particle Swarm Optimization with a Local Search (PSOGSA-ACO-LS). They concluded that the proposed algorithm is very close to the optimal solution of the standard TSP instances and also it has a great ability to solve problems that contain more than 100 cities.

Sahana (2019) modelled a new hybrid algorithm consisting of genetic algorithm, heuristics like remove-sharp, and local-opt with Ant Colony System (ACS). They concluded that the proposed method can reach the solution efficiently and requires fewer iterations compared to the standard ACS and hybrid GA. Also, it was competing with the exact algorithms if the number of cities is small as well as it outperforms many of the heuristic algorithms in instances with large cities.

We conclude from the mentioned studies above that hybridization of the ant colony algorithm ACO with other suggested algorithms mostly gives better results comparing with the standard ACO. However, most of the algorithms in the related work above have a lack in terms of the algorithm speed and the calculating performance. For this reason, we put our attention intensively in studying the complexity of time issue and tried to reduce it in order to obtain better results efficiently.

3 Methods and Techniques

3.1 Mathematical Model of a TSP

In graph theory, the problem is represented by a graph consisting of a number of nodes that represent the cities and these nodes are connected to each other by lines. Each line can be represented as the transportation between any two cities. The primary goal is to find a path that passes through all the cities with the lowest possible cost Jünger et al. (1995).

If $G = (V, E)$ is a weighted complete graph where V is the set of nodes such that $|V| = n$ and E is the set of edges. Each node in the graph is connected with $n-1$ edges. M is $n \times n$ cost or distance matrix. H is the set of the all possible Hamiltonian cycles in G which is represent

the seller tour. The mathematical formulation of the TSP is given as:

$$\min(h) = \sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij}\mathcal{L}_{ij},$$

where $i, j \in V$ $q \in Q$, $m_{ij} = m_{ji} = 0$ if $i = j$, $\mathcal{L}$ is the decision variable, and $h \in H$

$$\sum_{i=1}^{n} \mathcal{L}_{ij} = 1 \qquad j = 1, \ldots, k$$

$$\sum_{j=1}^{k} \mathcal{L}_{ij} = 1 \qquad i = 1, \ldots, n$$

otherwise $\mathcal{L}_{ij} = 0$ that mean $\mathcal{L}_{ij}$ is equal to 1 if the seller is travelling from the city i to the city j and $\mathcal{L}_{ij}$ is equal to 0 if the seller is travelling from the city i to another city that is not j.

3.2 Ant Colony Optimization (ACO)

Dorigo et al. (1998) introduced the algorithm of the ant colony. It is one of the heuristic algorithms that is derived from the natural behaviour of ants in the nature based on analysing the real ants thinking about finding and storing food in the colony. This algorithm has been tested in large optimization problems and it has been shown that this algorithm has a great ability to reach the optimal solution.

ACO depends on the behaviour of ant swarms, measure the reactions of these individuals in the past, and collecting new information and then building new possibilities in order to improve the solution in each iteration. Each ants in the colony communicates with the other ants using chemicals that are called pheromones. The transmission of ants is initially random, but after a while, the amount of pheromones will increase in the path that the ants will prefer, which will make most of the ants walk in this way.

Whenever this path is short, it will accelerate the process of returning the ants to the colony. The path that takes less time in returning is going to have large amount of pheromones. Since the pheromone is an evaporation substance, the probability of ants going to paths chosen by a few ants will be less likely. Therefore, the path that contains a large amount of pheromone will represent the optimal path.

3.2.1 Ant Colony on TSP

In ant colony algorithm, there is a set of solutions equal to the number of ants m in each iteration t. These solutions are based on information collected by pheromones (ζ) in each path. It is important to mention that there are two important concepts of ACO in TSP which determine the optimal solution among the set of solutions that are obtained in each iteration. These concepts can be explained as follows:

The Concept of Transition When the time $t = 0$, the ants are randomly placed on the cities, and all the directions will have the same amount of pheromones. In this case, the initial state of the pheromone can be described as $\zeta_0 = c$, where $c \epsilon \mathbb{R}$. In the next step, each ant will move to the next unvisited city depending on the transport rules that are followed by ants which can be described as below Gao (2020):

1. Pheromone density (ζ_{ij}) on each path from the city i to the city j and this data is inferred from the algorithm.

2. The heuristic information (η_{ij}) between every two cities can be calculated by $\eta_{ij} = \frac{1}{d_{ij}}$, where d_{ij} is the distance between the current city i and the next city j.

By using these rules, the probability (P_{ij}^k) of k^{th} ant to move to the next city will be generated, and this probability is calculated in the following way:

$$P_{ij}^k(t) = \left\{ \begin{array}{ll} \frac{(\zeta_{ij}(t))^\alpha (\eta_{ij})^\beta}{\sum (\zeta_{ij}(t))^\alpha (\eta_{ij})^\beta}, & \text{if } j \text{ is not visited city} \\ 0, & \text{otherwise} \end{array} \right\},$$

where i is the current city, j is the next city, t is the iteration number, and α, β are parameters used to determine the importance of the relation between pheromones and distance. The ant will move to the next city by comparing the probability on each path and choosing the path with the highest probability.

The Concept of Pheromone Updating In order to improve the solution, the pheromone must be updated in every iteration, and the formula of pheromone updating of the next city $(t+1)$ is given as

$$\zeta_{ij}(t+1) = (1-\rho)\,\zeta_{ij}(t) + \sum_{k=1}^{m} \triangle\zeta_{ij}^k(t, t+1),$$

where ρ is pheromone evaporation coefficient $(0 < \rho = 1)$, $\triangle\zeta_{ij}^k(t, t+1)$ is the amount of pheromone left by an ant in each iteration which is given by the following formula

$$\triangle\zeta_{ij}^k(t, t+1) = \left\{ \begin{array}{ll} \frac{Q}{L^k} & \text{if } (i,j) \in \psi^k \\ 0 & \text{otherwise} \end{array} \right\},$$

where Q is a fixed value that determines the amount of pheromone increment in each step, L^k is the length of the path ψ^k that the ant k visited Deng et al. (2019).

3.3 Neighbour Joining Method (NJ)

This method was first demonstrated by Naruya Saitou and Nei Saitou & Nei (1987), then it got developed in several researches later. This method can be described as if we have a matrix of distances calculated from a set of nodes, then a tree consisting of leaves and branches can be formed depending on these distances Al-Neama et al. (2014). Therefore, this method is one of the most important ways to form a phylogenetic tree, such as the formation of DNA trees and protein sequences. The idea of this method can be presented as the following steps:

1. If D is a distant matrix of n species S where $(s_1, s_2, \ldots, s_n \in S)$.
2. Calculate $\mathcal{U}_{s_i} = \sum_{s_k \neq s_i} \frac{D_{s_i,s_k}}{(n-2)}$ for all species $s_i \in S,\ i = 1, 2, , , , n$.
3. Select s_i, s_j which have the minimum values calculated from $D_{s_i,s_j} - \mathcal{U}_{s_i} - \mathcal{U}_{s_j}$.
4. Join the nodes s_i, s_j in a new node s_{ij} and compute the branch length from s_{ij} and nodes s_i, s_j by $d_{s_i,s_{ij}} = \frac{1}{2}D_{s_i,s_j} + \frac{1}{2}(\mathcal{U}_{s_i} - \mathcal{U}_{s_j})$, $d_{s_j,s_{ij}} = \frac{1}{2}D_{s_i,s_j} + \frac{1}{2}(\mathcal{U}_{s_j} - \mathcal{U}_{s_i})$.
5. Delete the s_i, s_j from D, and replace them by s_{ij}.
6. Update D by computing the distance between s_{ij} and other remaining species.
7. Redo the previous steps until formatting a complete tree.

Applying these steps on any distant matrix will lead to forming a complete tree, and that all species will be represented as leaves tied to branches.

3.4 Neighbour Joining Ant Colony Optimization (NACO)

Using hybridization among several algorithms usually leads to better results. However, this does not mean that hybrid algorithms do not fall into bad solutions. In our proposed algorithm that is resulted from the hybridization of the ant colony optimization ACO and Neighbour Joining method NJ (we briefly called the new algorithm NACO), we took an intensive attention to all of the criteria that might affect the solution and applied it to the travelling salesman problem TSP. We got many improvements in the solutions by comparing them with the original algorithm of the ant colony especially in reducing calculations as well as time.

In fact, our main concern of our proposed method (NACO) was to reduce the complexity of the space, and we leaned on selecting the NJ method in our hybridization because using the NJ algorithm will link all the cities in any TSP and create a complete tree as shown in Fig.2. This processing can be done by the calculations mentioned in (3.3).

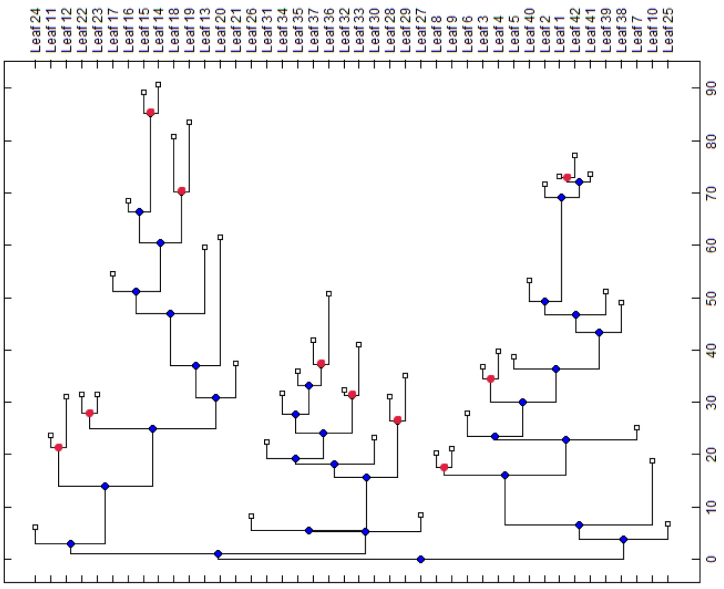


Figure 2: NJ tree of 42 cities in dantig42.

One of the advantages that the NJ method has in building trees, is building a reverse tree that starts from the leaves and ends in roots. This advantage made us able to create a matrix that it consists of the first stage of the joined leaves by connecting every two cities individually before connecting them to a third city. We called this matrix (LEAF1). This idea was applied to the TSP instance (dantig42), and we got the results that are shown below in Fig.2

From Fig.2, we can see that every two leaves are connected to a red dot which represents a connection of two cities in a TSP. Therefore, we conclude that the first stage of the joined leaves matrix can be revealed as:

$$\text{LEAF1} = \begin{bmatrix} 14 & 15 \\ 18 & 19 \\ 1 & 42 \\ 22 & 23 \\ 11 & 12 \\ 28 & 29 \\ 36 & 37 \\ 32 & 33 \\ 3 & 4 \\ 8 & 9 \end{bmatrix}$$

The explanation of the data in matrix LEAF1 can be described as that the cities in the first

column 14, 18, 1, 22, 11, 28, 36, 32, 3, 8 are linked to the cities in the second column 15, 19, 42, 23, 12, 29, 37, 33, 4, 9 respectively. After applying the information of the matrix LEAF1 in dantig42, we got the results that are shown below in the Fig. 3.

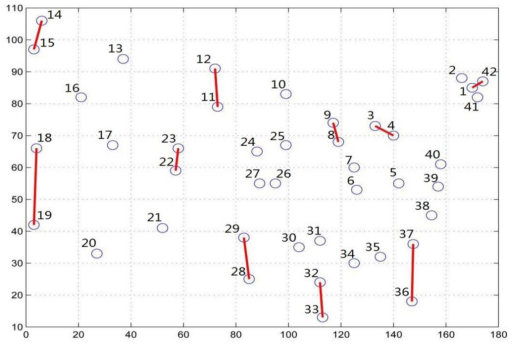


Figure 3: Applying LEAF1 matrix in dantig42.

From Fig. 3, it is clear that the 42 nodes in the graph have been reduced to 22 nodes and 10 edges. Now, by using mentioned procedure of ACO in (3.2), the seller will start selecting a city randomly to start his tour, then he will start moving from a city to another. Now, when the seller reaches any city within the matrix LEAF1, he will directly move to the city that is placed in the same row and the different column in the matrix LEAF1 without doing any extra calculations.

On the other hand, if the city is not listed in matrix LEAF1, the seller will carry on the usual way of the ant colony algorithm until he completes his tour and returns to the city from which he started. The main goal that we got here is the complexity was reduced by assuming that each two cities are linked together represent just one city. Therefore, moving to the city 14 in column 1 or the city 15 in column 2 have the same meaning. This procedure can be applied to the rest of the cities that are linked together in the matrix LEAF1.The entire processing of our proposed algorithm is shown in Algorithm (1).

Algorithm 1: NACO Algorithm

Input: cities number, ants number, iteration and ACO parameters $\rho, \alpha, \beta, \zeta_0, Q$
Output: Min of (local solution)

```
Compute distance matrix
Compute LEAF1 matrix using NJ algorithm
for t ← 1 to iteration do
    for m ← 1 to ants number do
        for n ← 2 to cities number do
            if i ∈ LEAF1matrix then
                go to j
            else
                compute P and chose j
            store the tour h and compute the cost (h)
        localsolution ← Min(cost(h))
    store local solution
    update pheromone matrix
globalsolution ← Min(localsolution)
```

4 Experiments and Results

4.1 Experimental Environment

The below implementations of all the experiments have been done using the following information (Devises and Software):

1. HP Laptop with 8GB of RAM and Intel core i7-8565U CPU with 8 cores.
2. 64-bit Windows operating system.
3. MATLAB® R2017a (Version 9.2).

Our proposed algorithm (NACO), was tested on many instances of TSP. In addition, we compared the results we got with the original ant colony algorithm method. The instances that we used in our experiments can be found in the following website: (http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/).

All the parameters of the ACO method and the hybrid ant colony algorithm NACO that are shown in Table 1, were taken from the range of the standard parameters without any changes Li & Zhu (2016).

Table 1: Parameters set-up

Parameter name and symbol	Value
Density of pheromones Q	100
the coefficient of Pheromone evaporation ρ	0.1
Data gathering factor α	1
Indicative prediction factor β	5
Number of iterations i	200
Number of ants k	Equal to the number of cities n

The initial pheromone value ζ_0 was chosen using the following equation

$$\zeta_0 = \frac{10 \times Q}{n \times mean(M)}.$$

We performed several experiments and then we selected the top 20 implementations for both algorithms individually. The instances that we used in the experiments are: (dantzig42, att48, eil51, eil76, eil101, pr107, bier127, krob200, rl1889, d2103, rl5915, rl11849).

4.2 Results and Analysis

The information below in the Table 2 have been arranged as the following:

1. **Instances**: dantzig42, att48, eil51, eil76, eil101, pr107, bier127, krob200.
2. **Algorithm**: ACO and NACO.
3. **Optimal solution**: which represents the best cost that is taken from the TSPLIB.
4. **n**: represents the number of cities in each instance.

5. **Worst**: represents the worst result of the two algorithms.
6. **Best**: represents the best result of the two algorithms.
7. **Average**: represents the average solution of the 20 independent implementations.
8. $\mathbf{D}_{best}$: denotes to the percentage deviation of the best solution.
9. $\mathbf{D}_{avg}$: denotes to the percentage average of the best solution.

D_{bes} and D_{avg} can be calculated respectively as follows:

$$D_{best}(\%) = \frac{optimal\ solution - best\ solution}{optimal\ solution} \times 100,$$

$$D_{avg}(\%) = \frac{optimal\ solution - \text{average}\ solution}{optimal\ solution} \times 10.$$

For ACO and NACO optimization algorithms, the experiments show that NACO can obtain the best optimization values in all of the taken TSP instances. For example, in eil51, NACO has obtained 439.5788, where the optimal solution is 426. Also, in att48, NACO reached 34286.56, which is very closed to the optimal solution 33522.

Table 2: The experiment and result

Instances	Algorithm	Optimal solution	n	Worst	Best	Average	D_{best}	D_{avg}
att48	NACO	33522	48	34945.7	**34286.56**	**34652.05**	2.28	3.37
	ACO			35413.11	34587.47	35128.09	3.17	4.79
eil51	NACO	426	51	441.8957	**439.5788**	**440.5195**	3.18	3.4
	ACO			452.4181	441.2533	446.8659	3.5	4.89
eil76	NACO	538	76	568.0236	**557.0516**	**565.9005**	3.54	5.18
	ACO			575.1985	562.0776	570.0606	4.47	5.95
eil101	NACO	629	101	690.2071	**679.8785**	**686.2631**	8.08	9.1
	ACO			692.9806	681.1936	694.0937	8.29	10.34
pr107	NACO	44303	107	46111.2	**45896.98**	**46017.12**	3.59	3.86
	ACO			46520.64	45974.25	46267.55	3.77	4.43
bier127	NACO	118282	127	122732	**121728.3**	**122444.8**	2.91	3.51
	ACO			124497.47	123159.04	123899.75	4.12	4.74
krob200	NACO	29437	200	32370.94	**31794.16**	**32203.08**	8	9.39
	ACO			32744.096	32078.586	32365.203	8.97	9.94

It is clear that the proposed algorithm NACO has greatly improved the time of the implementation. In addition, the acceleration towards the solution was much better than the original algorithm ACO as shown in Table 3. This table contains the instances, the time which is taken by the algorithms (ACO and NACO) in each instances, the difference in times, and time improvement percentage which represents the improvement at the time of the implementation which was calculated as the following way:

$$Time\ improvement\ percentage(\%) = \frac{ACO\ time - NACO\ time}{ACO\ time} \times 100$$

Table 3: The times improvement in experiments

instances	NACO	ACO	ACO-NACO	Time improvement percentage
dantzig42	7.59375	7.91875	0.325	4.104
att48	10.44531	11.06641	0.6211	5.612
eil51	11.55859	12.26641	0.70782	5.77
eil76	29.09375	38.65313	9.559383	24.731
eil101	57.07891	60.6523	3.57339	5.891
pr107	66.37266	72.46563	6.09297	8.408
bier127	95.14063	102.623	7.48237	7.291
krob200	279.6586	319.684	40.0254	12.52

It is worth to mention that relying on the linked cities in LEAF1 has clearly improved the results. The linked cities were identical to the optimal tour for most of the instances. As a result, there was an improvement in the time which was due to the reduction in the mathematical calculations within the algorithm. That is because, the number of times of calling the function *rouletteSelection* in MATLAB that is responsible for the seller moving from a city i to a city j in has decreased. Table 4 shows the details of the number of times that MATLAB recalls the function *rouletteSelection* in both algorithms NACO and ACO respectively.

Table 4: The *rouletteSelection* function recalling in MATLAB

Instances	NACO	ACO
att48	364800	451200
eil51	387600	510000
eil76	927200	1140000
eil101	1717000	2020000
pr107	1883200	2268400
bier127	2565400	3200400
krob200	6080000	7960000

The linked cities from the beginning of implementation are shown in Table 5.

Table 5: The linked cities

instances	Number of cities	Number of linked cities
att48	48	9
eil51	51	12
eil76	76	14
eil101	101	15
pr107	107	18
bier127	127	25
krob200	200	47

The relation between the number of the linked cities and the reduction in rouletteSelection function calling within the algorithm, and it is illustrated in percentage for each instance, as shown in Fig. 4.

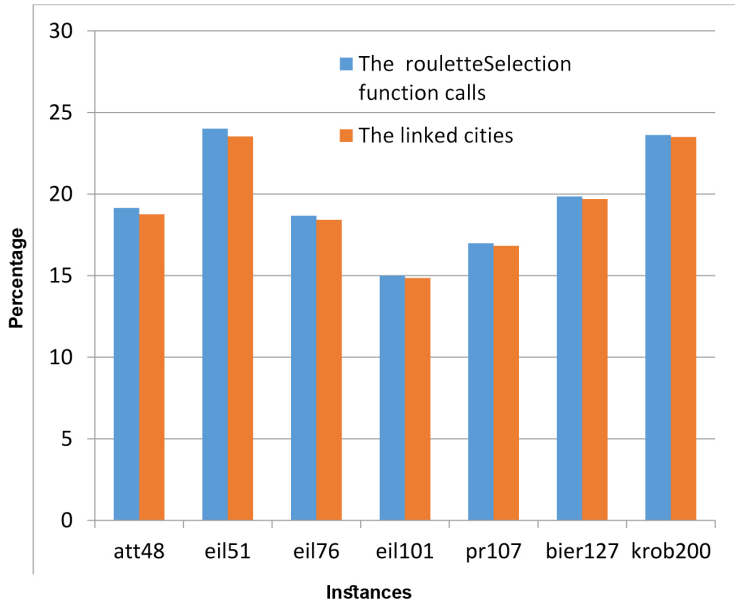


Figure 4: The relationship between the linked cities and reduction in the calculations

4.3 New Results

In the instance (dantzig42), the results of the two algorithms can be shown in Table (6). It is clear that NACO has recorded new results, where, the tour length that is recorded in TSPLIB for dantzig42 was (699), but the best tour in NACO was (684.9386). Also, the average of 20 runs was smaller than (699). Now, the time that is taken for NACO was less than ACO by 0.325 seconds, and this is a significant improvement in time and cost.

Moreover, the complexity of the space within the algorithm has been also reduced, because the recalling of *rouletteSelection* function has been decreased significantly by 24.39% as shown in the Table (7).

The convergence towards a solution in NACO is much faster than ACO, as shown in Fig. (5).

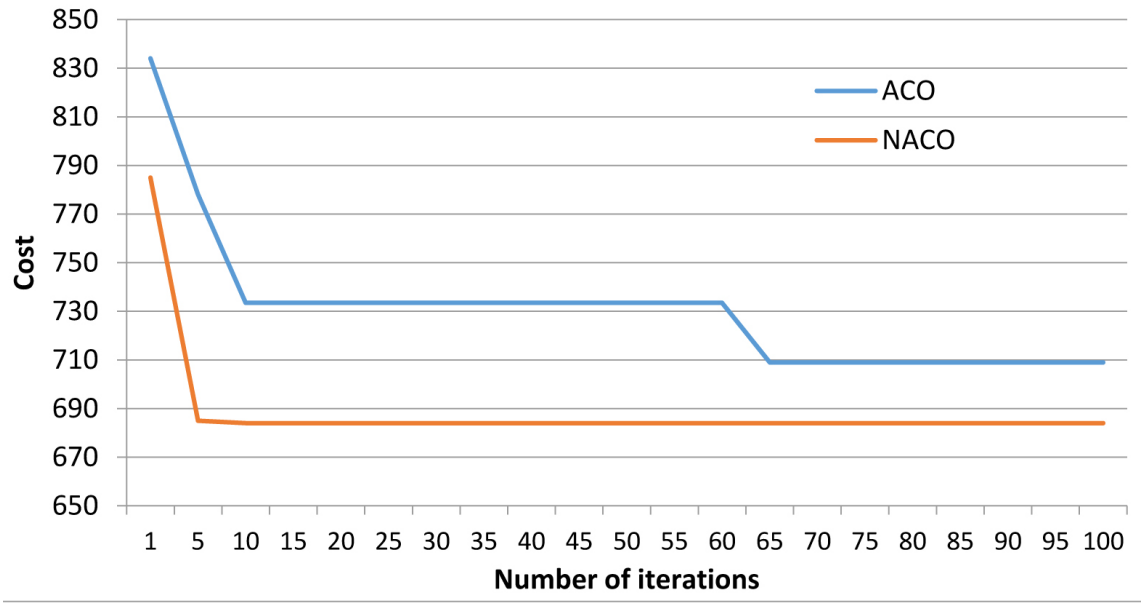


Figure 5: Solution convergence

Table 6: Result of 20 implementations on dantzig42 using ACO and NACO

dantzig42							
NACO				ACO			
No. it	No. Cities	Best Cost	Time(s)	No. it	No. Cities	Best Cost	Time(s)
200	42	684.9386	7.71875	200	42	704.8635	8.03125
200	42	687.8036	7.25	200	42	714.442	8.078125
200	42	692.3374	7.640625	200	42	716.0602	7.984375
200	42	692.7099	7.515625	200	42	716.1693	7.515625
200	42	693.2885	7.34375	200	42	718.404	7.90625
200	42	694.0555	7.828125	200	42	719.7429	7.671875
200	42	695.3411	7.609375	200	42	719.7945	8.0625
200	42	696.87	7.6875	200	42	721.4636	7.984375
200	42	697.7633	7.875	200	42	725.8268	8.015625
200	42	697.8502	7.09375	200	42	725.9792	8.109375
200	42	697.8502	7.421875	200	42	728.3959	7.859375
200	42	697.8502	7.859375	200	42	728.7708	8.0625
200	42	698.566	7.703125	200	42	729.0138	7.875
200	42	699.1626	7.703125	200	42	732.0614	7.625
200	42	699.6677	7.734375	200	42	732.1105	8.015625
200	42	700.909	7.765625	200	42	732.4635	7.984375
200	42	700.9959	7.765625	200	42	734.8429	8.078125
200	42	701.1678	7.625	200	42	735.8168	7.484375
200	42	701.6084	7.625	200	42	736.2496	7.921875
200	42	702.0231	7.109375	200	42	739.4587	8.109375
Average		696.63795	7.59375	Average		725.5964	7.91875

Table 7: The *rouletteSelection* function recalling in Matlab of dantzig42 instances

Algorithm	No. of recalling the function
ACO	260400
NACO	344400

Fig. 5 shows comparing with ACO, NACO needs a few iterations to reach the best solutions. In other words, it can converge towards a solution faster than ACO. Moreover, NACO was able to eliminate any intersection of edges that might appear in the graph, and that it is certainly the intersection of any two edges will lead to increase the cost of the tour as shown in the Fig. 6, where the ACO with the intersection on the left side, and the NACO on the right side.

4.4 Big Sample Instances

To make sure of the work of our proposed algorithm properly, NACO was applied to instances that have more than 10000 cities and was also compared with ACO. It is worth to mention that we made a significant improvement as follows.

In instance (rl1889), the number of cities are 1889 and the number of the joined cities in

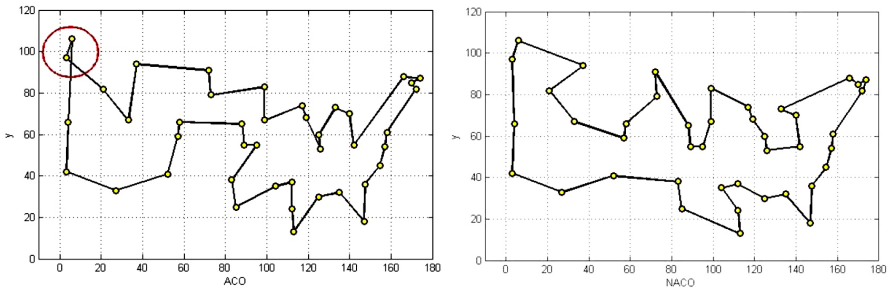


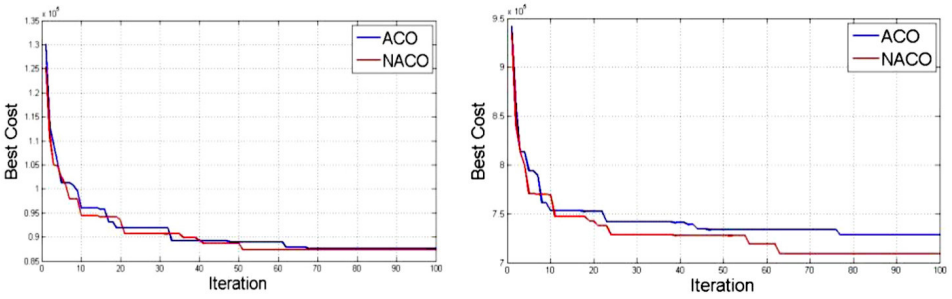
Figure 6: Intersection of edges eliminate by NACO.

NACO are 307. Now, after applied ACO and NACO 100 iterations and 100 ants, we got this results that are shown here in Table 8.

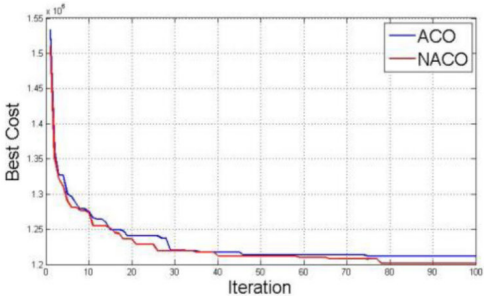
Table 8: Time, best cost, and RouletteSelection recalls in MATLAB of (rl1889)

rl1889	ACO	NACO
Best cost	382804.487	**380153.822**
Time (s)	6223.398	**5279.90156**
RouletteSelection recalls in MATLAB	145810000	**18870000**

In the instaces (d2103, rl5915, and rl11849) that contain 2103, 5915, and 11849 cities respectively, ACO and NACO have been applied with 100 iterations and 10 ants. NACO had a significant advantage of the convergence to the solution as shown in Fig. 7a, Fig.7b, and Fig. 7c.



(a) The convergence of (d2103) **(b)** The convergence of (rl5915)



(c) The convergence of (rl11849)

Figure 7: The convergence of the big sample instances

4.5 Complexity Analysis

In the complexity theory, it is known that the Big-O notation of finding different solutions of the ACO algorithms is equal to $O(n^3)$ by assuming the number of ants and cities are the same Yuan et al. (2009). In NACO, we were able to reduce this complexity to $O(n^2\,(n-\omega))$, where w is a positive number represents the number of rows in the matrix LEAF1.

5 Conclusion

To sum up, it is not a necessity to hybrid ACO method with a method that can deal with TSP. In other words, NACO method was a resultant of the NJ method that was not designed for the TSP and ACO method. In addition, NACO method has overcome the original ACO method in most of the instances that we implemented in our paper in terms of time and space complexity. Moreover, NACO method has shown a significant improvement in the big sample instances when it compares with the standard ACO method. Finally, the number of calculations within NACO has been decreased from $n^3$to $n^2(n-\omega)$ comparing with ACO in terms of the Big-O notation complexity.

References

Ali, A.H. (2017). Modifying Some Iterative Methods for Solving Quadratic Eigenvalue Problems. (Master's thesis, Wright State University).

Al-Neama, M.W. , Reda N.M. & Ghaleb F.F.M. (2014). Accelerated guide trees construction for multiple sequence alignment. *International Journal of Advanced Research, 2*(4), 14–22.

Balas, E., Toth, P. (1983). *Branch and Bound Methods for the Travelling Salesman Problem* (No. MSRR-488). Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Chen, S.M., Chien, C.Y. (2011). Solving the travelling Salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*, *38*(12), 14439-14450.

Deng, W., Xu, J., & Zhao, H. (2019). An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. *IEEE Access*, *7*, 20281-20292.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *26*(1), 29-41.

Du, D.Z., Ko, K.I. (2011). *Theory of Computational Complexity* (Vol. 58). John Wiley & Sons.

Duan, H., Yu, X. (2007, April). Hybrid ant colony optimization using memetic algorithm for travelling Salesman problem. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning* (pp. 92-95). IEEE.

Farisi , O.I.R., Setiyono, B., & Danandjojo, R.I. (2016). A Hybrid Firefly Algorithm of Ant Colony Optimization for travelling Salesman Problem. *Jurnal Buana Informatika*, *7*(1).

Gülcü Ş., Mahi, M., Baykan, Ö.K., & Kodaz, H. (2016). A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving travelling salesman problem. *Soft Computing*, *22*(5), 1669-1685.

Gao, W. (2020). New Ant Colony Optimization Algorithm for the travelling Salesman Problem. *International Journal of Computational Intelligence Systems*, *13*(1), 44-55.

Girsang, A.S., Tsai, C.W., & Yang, C.S. (2014). A Hybrid Ant-Bee Colony Optimization for Solving travelling Salesman Problem with Competitive Agents. In *Mobile, Ubiquitous, and Intelligent Computing* (pp. 643-648). Springer, Berlin, Heidelberg.

Gong, D., & Ruan, X. (2004, June). A hybrid approach of GA and ACO for TSP. In *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No. 04EX788)* (Vol. 3, pp. 2068-2072). IEEE.

Jünger, M., Reinelt, G., & Rinaldi, G. (1995). The Travelling Salesman Problem. *Handbooks in Operations Research and Management Science*, *7*, 225-330.

Junqiang, W., Aijia, O. (2012, August). A hybrid algorithm of ACO and delete-cross method for TSP. In *2012 International Conference on Industrial Control and Electronics Engineering* (pp. 1694-1696). IEEE.

Kumar, S., Kurmi, J., & Tiwari, S. P. (2015). Hybrid ant colony optimization and Cuckoo search algorithm for travelling salesman problem. *International Journal of Scientific and Research Publications*, *5*(6), 1-5.

Li, P., Zhu, H. (2016). Parameter selection for ant colony algorithm based on bacterial foraging algorithm. *Mathematical Problems in Engineering*.

Mahi, M., Baykan, Ö.K., & Kodaz, H. (2015). A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for travelling Salesman Problem. *Applied Soft Computing*, *30*, 484-490.

Ratanavilisagul, C., & Pasaya, B. (2018, July). Modified Ant Colony Optimization with Updating Pheromone by Leader and Re-Initialization Pheromone for Travelling Salesman Problem. In *2018 International Conference on Engineering, Applied Sciences, and Technology (ICEAST)* (pp. 1-4). IEEE.

Razali, N.M., Geraghty, J. (2011, July). Genetic algorithm performance with different selection strategies in solving TSP. In *Proceedings of the World Congress on Engineering* (Vol. 2, No. 1, pp. 1-6). Hong Kong: International Association of Engineers.

Rokbani, N., Kromer, P., Twir, I., & Alimi, A.M. (2019). A new hybrid gravitational particle swarm optimisation-ACO with local search mechanism, PSOGSA-ACO-Ls for TSP. *International Journal of Intelligent Engineering Informatics*, *7*(4), 384-398.

Sahana, S.K. (2019). Hybrid optimizer for the travelling salesman problem. *Evolutionary Intelligence*, *12*(2), 179-188.

Saitou, N., Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution, 4*(4), 406-425.

Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., & Wang, Q.X. (2007). Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, *103*(5), 169-176.

Wang, G.G., Hao, G.S., Cheng, S., & Qin, Q. (2016, June). A discrete monarch butterfly optimization for Chinese TSP problem. In *International Conference on Swarm Intelligence* (pp. 165-173). Springer, Cham.

Yamada, T., Aihara, K., & Kotani, M. (1993, October). Chaotic neural networks and the travelling salesman problem. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)* (Vol. 2, pp. 1549-1552). IEEE.

Yaseen, M.T., Ali, A.H., & Shanan, I.A. (2019). Weighted (k, n)-arcs of Type $(n - q, n)$ and Maximum Size of (h, m)-arcs in $PG(2, q)$. *Communications in Mathematics and Applications, 10*(3), 361-368.

Yuan, Y., Wang, K., & Ding, L. (2009, August). A Solution to Resource-Constrained Project Scheduling Problem: Based on Ant Colony Optimization Algorithm. In *2009 Ninth International Conference on Hybrid Intelligent Systems* (Vol. 1, pp. 446-450). IEEE.