International Journal of Innovation and Scientific Research
ISSN 2351-8014 Vol. 44 No. 2 Sep. 2019, pp. 131-143

http://www.ijisr.issr-journals.org/

Analysis for Travelling Salesman Problem using Improved Ant Colony Optimization Algorithm

Zar Chi Su Su Hlaing

Faculty of Information Science,
University of Computer Studies (Magway),
Magway, Myanmar

ABSTRACT: Ant Colony Optimization (ACO) is a recent algorithm used for solving optimization problems and is the model on the behavior of real ant colonies. It has been used exclusively for solving problems in the combinatorial optimization domain. Traveling salesman problem (TSP) is one of the well-known and extensively studied problems in combinational optimization and used to find the shortest roundtrip of minimal total cost visiting each given city (node) exactly once and it can be applied to solve many practical problems in real life. ACO is a good search capability and a high-performance computing method for TSP. But, the traditional ACO has many drawbacks such as stagnation behavior, trapping in local optimal and premature convergence. This paper implements and evaluates a specialized version of ant colony optimization capable of searching in travelling salesman problems and evaluates its performance under a range of conditions and test cases. The proposed system is an improved ant colony optimization algorithm with dynamic candidate set strategy is adopted to rapid convergence speed and adapting parameter to improve the performance in solving TSP. Algorithms are tested on benchmark problems from TSPLIB and test results are presented. From our experiments, the algorithm has better performance on TSP and analysis results are presented.

KEYWORDS: combinatorial optimization, stagnation behavior, adaptive behavior, dynamic candidate set, adapting parameter.

1 INTRODUCTION

Ant Colony Optimization is a perpetual topic in the research field of metaheuristic how to improve the convergence speed under the condition of guaranteeing the solution quality. ACO is a constructive metaheuristic techniques that allows each ant to add an element (such as the next city for the TSP) to its solution at each step of the algorithm. One possible difficulty encountered by ACO algorithms is when they are applied to problems with big-sized neighborhood in the solution construction. Possible problems are that the solution construction is significantly slow down because of scanning the set of all possible solution elements before choosing a particular one and that the probability which many ants visit the same state is very small. Hence, the computational time required for each step of the algorithm can be large. Such a situation can occur in the ACO application to large TSPs. There has been little work done in this area, despite the fact that this can potentially improve the efficiency of ACO, especially for large real world problems. So, both comparative slow convergence speed and comparative long runtime are the quite prominent problems in ACO.

In such situations, the above-mentioned problem can be considerably reduced by the use of candidate lists. Candidate lists comprise a small set of promising neighbors of the current state. Using a prior available knowledge on the problem, candidate lists are created, if available, or dynamically generated information. Their use allows ACO algorithms to focus on the more interesting components, strongly reducing the dimension of the search space.

Candidate set strategies have traditionally only been used as part of a local search procedure applied to the solutions generated by ACO. However, the strategies developed for local search heuristics such as 2-Opt and 3-Opt are inappropriate for

use in the construction phase of the ACO algorithm and it is only in later improvements of ant colony system that candidate set strategies were applied as part of the construction process.

TSP has also been widely used as a problem for testing various metaheuristics. Some work based on ant colony optimization technology was reported by Dorigo et al. [1], [2], [3] and [4]. MacGregor and Chu [6] provided a review of recent research on human performance on the travelling salesman problem and related combinatorial optimization problems. Various adaptations: an algorithm based on the basis of the ant system, dynamic control of solution construction and mergence of local search [9], new pheromone updating strategies [12], max-min ant system [11], a strategy is to partition artificial ants into two groups: scout ants and common ants [14], are studied to improve the quality of the final solution and lead to speedup of the algorithm.

2 REASONS FOR ADAPTING PARAMETER

The basic ant system faces with many drawbacks such as stagnation behavior, trapping in local optimal and premature convergence. Stagnation denotes the undesirable situation in which all ants construct the same solution over and over again, making further exploration of newer paths almost impossible. This derives from excessive trail levels on the edges of one solution, and can be observed in advanced phases of the search process. It happens when the ACO parameters (α, β, ρ, q_0) are not well tuned for undertaking the problem.

The following stagnation problem was observed on ant colony system. The individuals with the lowest values for q_0 and the highest values for q_0 dominated the construction very quickly. This is not unsurprising, since these individuals avoid exploration and focus only on exploitation. A low value ranks distance higher than pheromone information. A low q_0 influences the algorithm to explore more often, but if the value is too low, the search tends to be too erratic and unguided, despite the pheromone trail. Also, a high value for q_0 means that the locally best option is chosen rather than any option probabilistically. This leads to relatively short tours, since no exploration takes place. Exploration generally results in longer tours and therefore individuals who favor exploration will be seen as less fit and their genes will vanish from the gene pool. In rare cases exploration successfully finds a shorter tour, but generally 'exploration favoring' values will be eradicated before they get a chance to prove their worth. 'Exploitation favoring' on the other hand comes to dominate the ants due to this premature convergence. One would really like to defer the selection for a few iterations in order to give the 'exploring' ants a chance to find an improved tour.

To prevent this premature convergence and stagnation, a random number *q* is generated and compared to the trail on the edge. For a very high pheromone trail τ, *q* will be less and for a low value of the same, q is high. Thus, if the trail is not too high, the algorithm may overlook the best path at the selection point and take up an alternative edge. However, if the trail is very high, the algorithm has a tendency to stick to this edge. Therefore, it can be encouraged from the above discussion that the proposed system adapts the parameter q_0 and dynamic updating of parameter β which is to prevent the system stagnating at points and thus avoids locally optimized solutions and improve the performance of the algorithm.

3 IMPROVED ACO ALGORITHM

A modified version of the ant colony system is based on ACS [1], [2]. The state transition rule and pheromone updating rules are from the original ant colony system. The modifications include flexible state transition rule and dynamic candidate list strategy. The system introduces a mechanism for escaping from local optima as well as increasing time efficiency.

The rule used by ants to select the next city has been the same from the ant colony system. Some of the time an ant will choose a city using the same biased random choice as before. The rest of the time the ant chooses only the "best" city based on the visibility and the trail.

Formally, an ant *k* on city *r* chooses a city *s* to move to according to:

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ [\tau_{(r,u)}]^{\alpha} \cdot [\eta_{(r,u)}]^{\beta} \right\}, & if\ q \le q_0 \\ S, & otherwise \end{cases} \quad (1)$$

where q is a random number in the range (0,1), q_0 is a constant, $J_k(r)$ is the set of cities that have not been visited by the k^{th} ant, and *S* is a random city selected according to:

$$P_k(r,s) = \begin{cases} \dfrac{[\tau_{(r,s)}]^{\alpha} \cdot [\eta_{(r,s)}]^{\beta}}{\sum_{u \in J_k(r)} [\tau_{(r,u)}]^{\alpha} \cdot [\eta_{(r,u)}]^{\beta}}, & if\ s \in J_k(r) \\ 0, & other\ \ wise \end{cases} \quad (2)$$

where, $P_k(r,s)$ is the probability of choosing city s from city r. $J_k(r)$ is the set of unvisited cities by ant k. $\tau(r,s)$ is pheromone for moving from city r to city s. $\eta(r,s)$ is local heuristic for moving from city r to city s is $1/d(r,s)$. α determines the importance of pheromone information and β determines the importance of heuristic visibility.

3.1 EXPLORATION AND EXPLOITATION

The algorithm has to achieve an appropriate balance between the exploitation of the search experience so far and the exploration of unvisited or relatively unexplored search space regions. Exploitation is the decision of taking the path with highest calculated probability. Exploration is the process whereby the ants decide which path to take based on the probabilities calculated. Hence, there is a higher chance of taking a path with higher probabilities. So exploitation or exploration is an important factor of the algorithm. The decision of exploitation or exploration factor is defined by a factor q_0, exploitation factor, which is a floating point value between 0 and 1.

In ant colony system, the parameters q_0, β, and ρ are constants that do not change during the execution of the ACS algorithm. Dorigo and Gambardella [3] used 0.9 as the value for q_0. This means that the ants will build a tour by exploring 10% of the time, and exploiting 90% of the time. Intuitively, q_0 determines the relative importance of exploitation versus biased exploration, whereas β and ρ determine the desirability of the edges.

In the proposed algorithm, ants explore more at the beginning and toward the end of the algorithm they tend to exploit more, utilizing the information that has been accumulated. This is accomplished by allowing q_0 in each cycle. Specifically, in each cycle is calculated a new value for q_0 as follows:

$$q_0 = \frac{IterationCounter}{MaxIteration} \quad (3)$$

$$q_0 = \begin{cases} 0.1 & if\ q_0 < 0.1 \\ 0.9 & if\ q_0 > 0.9 \\ q_0 & otherwise \end{cases} \quad (4)$$

where *IterationCounter* is the current iteration (cycle) number and *MaxIteration* is the maximum number of iterations. As q_0 gets larger, exploitation occurs more frequently. More exploitation may direct ants to use the edges of the global best tour more frequently. To avoid a local optimum when ants find the same tour over and over again or do not improve the global best tour after a certain number of iterations, we encourage the exploration of edges not used frequently by perturbing the global best tour erasing memory from some percentage of randomly chosen edges of the global best tour.

GLOBAL PHEROMONE UPDATING RULE

The next change is that only the globally best ant is allowed to deposit pheromone. The global pheromone update that occurred at the end of an iteration is an important phase. If there is no pheromone update, each ant will repeatedly find the same probability on all moves. The evaporation and deposit of pheromone is only added to the best route found since the beginning of the trial. This makes the global updating rule:

$$\tau(r,s) = \begin{cases} (1-\rho).\tau(r,s) + \rho.(L_{gb})^{-1}, & if\ (r,s) \in global\_best\_tour \\ \tau(r,s), & otherwise \end{cases} \quad (5)$$

where ρ is the pheromone decay parameter, and L_{gb} is the length of the globally best tour. The effect of this change is to concentrate the search made by the ants around the best known tour. Again, this increases exploitation.

LOCAL PHEROMONE UPDATING RULE

Finally, it is a local updating rule. While the global updating rule is applied to all edges at the end of each iteration, when all ants have completed a tour, the local updating rule is applied each time an ant moves to a new city. If no such action is performed, each of the ants in the iteration will be non-collaborative and use only the pheromone trail at the beginning of the iteration. The ant change the trail on the edge is followed according to the rule:

$$\tau(r,s) = (1-\rho).\tau(r,s) + \rho.\tau_0 \quad (6)$$

where ρ is a constant in the range (0,1) and τ_0 is the initial trail level. This initial value for trail is

$$\tau_0 = \frac{1}{n.L_{nn}} \quad (7)$$

where n is the number of cities, and L_{nn} is the length of the tour obtained by a nearest-neighbour search. Using the local update strategy, the pheromone concentration on the traversed edges is decreased. So the subsequent ants are encouraged to choose other edges and to produce different solutions. This makes it less likely that several ants produce identical solutions during one iteration.

In the original ACS, when ants choose the next vertex to move to, they consider the entire set of vertices that have not been visited yet. This can be very time consuming. To improve the efficiency of IACO, we restrict the set of nodes (vertices) that the ants consider to the *k* nearest neighbors by using dynamic candidate list.

3.2 DYNAMIC CANDIDATE SET BASED ON NEAREST NEIGHBOUR

Candidate list (*CL*) is a strategy that tries to improve the performance of an ant algorithm. It was proposed by Gambardella and his colleagues to accommodate searching procedure of ACS on larger data. It is used fixed size candidate list. For example *CL* = 15, or *CL* =20 when involves local search. However, applying a fixed list in candidate list is not flexible when facing with various sizes of data. Due to the purpose of improving algorithm performances, the proposed system is also applying candidate list. The proposed candidate list is a dynamic candidate list (DCL) procedure which captures a suitable number of nodes based on the total number of nodes and which sets need to be calculated and recalculated throughout the search process. It is a static data structure that lists a limited number of preferred closed cities to be visited order by increasing distance for each city. The numbers of closest cities that allowed being included into the candidate list were different from one algorithm to another. Due to the purpose of improving algorithm performance, the proposed algorithm is also applying dynamic candidate list in its solution construction process. The proposed dynamic candidate list based on nearest neighbor approach. It would not allow ants to venture into cities outside the candidate list. The number of cities or the size of the candidate list is also restricted to one fourth of the cities *n*. For example, seven was chosen resulting from the candidate list computation to determine the size of candidate list element for Oliver30 data. The candidate list procedure is as follows:

Procedure: Candidate list procedure

1. Initialize an empty node *Node* and *MaxLength*
2. Set dynamic candidate_list D*CL*=n/4 /*size of candidate list*/
3. **if** *DCL*>*MaxLength* D*CL*=*MaxLength*
4. Determine cities that not yet visited
5. **repeat**
6. **for** *i*=1 to n
7. Find the unvisited city *j* within the nearest neighborhood of *k(j ∈ N(k))*
8. Determine distance between city *j* and city *r*
9. **if** *distance < distance of previous city j*
10. *Node*← *city j* (move city *j* to *NL*)
11. **end if**
12. **end for**
13. D*CL*←*Node*
14. **until** candidate_list *(DCL)* is full

Fig. 1. Candidate List Strategy

4 IACO FOR SOLVING TSP

Now this section explains how the IACO operates, the following simple example is employed where the IACO is applied to solve a TSP which is considered a problem and the candidate solution is defined as a sequence of cities. In this example, there are eight cities (A to H) and assume that ant one (1) is placed in city A, ant two (2) is placed in city B and so on. Every time an ant (*k*) needs to move from city *i* to city *j*, it first search in candidate set and it adds its current start location to its tabu list and then uses Equation (3) where it generates a value for the parameter q_0 and also generates a random number for parameter *q*, when a random parament $q \leq q_0$, ant (*k*) exploits the knowledge available about the problem and goes to city (j) which has the maximum product of the amount of pheromone on the edge *(i,j)* and the shortest distance between the two cities according to Equation (1). While when $q > q_0$, the ant (k) explores new solutions using a probability decision from Equation (2). Each city has a candidate list (*cl*); its length is defined by the number of cities listed. In this example, assumne that *cl*=2, where cities (B) and (H) are on the candidate list of city (A) and they will be explored by ant (1) before other cities. it is assumed that ant (1) moves from city (A) to city (B), then city (B) will be added to the tabu list to avoid being visited twice by the same ant. After moving from city (A) to city (B), ant (1) updates the pheromone on the link between the two cities using local update Equation (6). For the next step, ant (1) again calculates the possibilities of moving from its current city (B) to those other cities that are not in its tabu list (C to H) using the same Equation (1) and so on until ant (1) visits all the seven cities as shown in Figure 2.

The length of the tour made by ant (1) will be calculated by adding the length of the arc between each two cities from the tour. The process will be accomplished by each ant and at the end of the iteration there will be five tours generated by five ants. The local search improvement is applied to improve the tour consturcted by the ants, The shortest of these tours will be selected as the best tour and the arcs that form this tour will be updated using the global update formula in Equation (5). Then, calculate the current entropy by analyzing the pheromone information to update the heuristic parameter β. Then, the ants are placed again randomly for a second iteration and redefine the exploitation parameter q_0. The algorithm goes on until a stopping criterion is met such as the minimum number of iterations or the global tour length has been found.

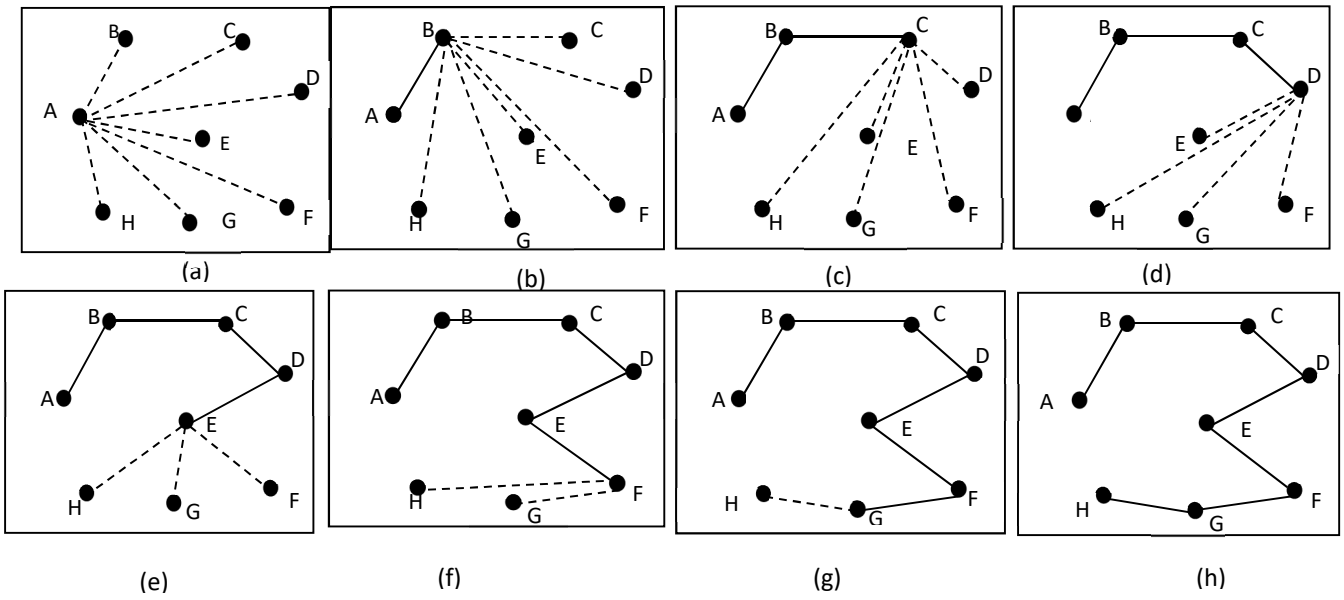

Fig. 2. IACO for a Simple TSP problem

5 PROCEDURE OF IACO

This section describes an implementation in a pseudo-code description.

```
Procedure: IACO for TSP
begin
InitializeData
    while (not termination) do
        ConstuctSolutions
        LocalSearch
        UpdateStatistics
        UpdatePheromoneTrails
 end-while
end
```

Fig. 3. General Procedure of IACO for TSP

In data initialization, (1) the TSP instance has to be read; (2) the distance matrix of the read instance has to be computed; (3) the candidate lists for all cities have to be computed; (4) the ants randomly place the their staring cities; (5) the algorithm's parameters must be initialized and (6) some variables that keep track of statistical informaiton, such as number of iterations, or best solution found (best tour length), and best tour.

```
Procedure: InitializeData
begin
 ReadTSPInstance
        ComputeDistances
    Determine candidate_list procedure
        InitializeAnts
        InitializeParameters
        InitializeStatistics
end
```

Fig. 4. Initialization procedure of algrithm

In construction steps the following two constuction steps are repeated until all ants have completed a tour. When exploiting the procedure *ConstructExploitDecisionRule* needs to be adapted. If not so, the procedure *ConstructExploreDecisionRule* needs to be computed. In *ConstructExploitDecisionRule* a first change is that when choosing the next city, one needs to find an unvisited city from the candidate list of the current city. A second change is necessay to deal with the situation in which all the cities in the candidte list have already been visited by ant *k*. In this case, the variable *node* keeps its initial value -1 and one city out of those not in the candidate list is chosen. The proceduer chooses the maximum product of the pheromone value and heuristic information $[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}$ as the next to move to. In *ConstructExploreDecisionRule* it has the two change as the *ConstructExploitDecisionRule*. But the exploring procedure chooses the next unvisited city according to acton choice rule as Equation (2).

```
Procedure: ConstructSolution
begin
        curNode ← startNode
q0←IterCounter/Iterations
        for k=1 to m ants do
            repeat
                q←random number
                if (q<q0) then
                    newNode ←ConstructExploitDecisionRule(k, curNode)
                else newNode ←ConstructExploreDecisionRule(k, curNode)
                end if
                Add newNode to ant k's tour
                LocalUpdatingRule(curNode,newNode)
                curNode ← newNode
            until ant k completes tour
        end for
end
```

Fig. 5. Construct solution procedure

```
Procedure: ConstructExploitDecisionRule(k, curNode)
begin
            sum_probability←0.0 // CandidateListConstructionRule
            node← -1
            for j=1 to DCL do
                if k^th ant's node j is not visited in candidate list then
                  selection_probability← value of transition probability
                  node← j          /* city with maximal τ^α η^β */
                end if
            end for
            if (node==-1) then                    // city outside candidate list
                for j=1 to n do
                  if k^th ant's node j is not visited outside the candidate list then
                  selection_probability← value of transition probability
                  node ← j          /* city with maximal τ^α η^β */
                end if
            end for
end if
return node
end
```

Fig. 6. Exploitation procedure of solution construction

```
Procedure: ConstructExploreDecisionRule(k,curNode)
begin
    sum_probability←0.0 // CandidateListConstructionRule
    node← -1
    for j=1 to DCL do
        if k^th ant's next node j is not visited in candidate list then
            partial_product← pheromone_value*exp(1/distance,β) /* city with τ^α η^β */
            sum_probability ← sum_probability+ partial_ product
        end if
    end for
    for j=1 to DCL do
        if k^th ant's next node j is not visited in candidate list then
            selection_probability ← partial_ product / sum_probability
            rno ← random number
            if selection_probability >= rno then
              node ← j
             break
            end if
        end if
    end for
        if node = -1 then                    // city outside candidate list
        for j=1 to n do
            if k^th ant's next node j is not visited outside the candidate list then
         partial_product← pheromone_value*exp(1/distance,β) /*city with τ^α η^β */
             sum_probability ← sum_probability+ partial_ product
            end if
        end for
        for j=1 to n do
        if k^th ant's next node j is not visited outside the candidate list then
            selection_probability ← partial_product / sum_probability
            rno ← random number
            if selection_probability >= rno then
              node ← j
             break
            end if
        end if
    end for
    return node
end
```

Fig. 7. Exploration procedure of solution construction

It is clear that by using candidate lists the computation time necessary for the ants to construct solutions can be significantly reduced, because the ants choose from among a much smaller set of cities. The next step is the local pheromone updating. It always triggers after the ants have moved to the next city.

```
Procedure: LocalUpdatingRule(curNode, newNode)
begin
 value←(1-ρ)*pheromone[curNode][newNode]+ρ* pheromone[curNode][newNode]
 pheromone[curNode][newNode] ← value
 pheromone[newNode][ curNode] ← value
end
```

Fig. 8. Local pheromone updating procedure

Once the solutions are constructed, they may be improved by a local search procedure (for example 2-Opt or 2.5-Opt). The next step in an iteration of the algorithm is the pheromone update (global pheromone updating). This is implemented by the procedure of *UpdatePheromoneTrails*, which comprises two pheromone update phases: pheromone evaporation and pheromone deposit. Pheromone evaporation decreases the value of the pheromone trails on only the best path by a constant factor ρ. Pheromone deposit adds pheromone to the edges belonging to tours constructed by the ant's best path length. *ComputeCurrentEntropy* computes the pheromone information of the current pheromone matrix to be used in the next step. *UpdateHeuristicParameter* dynamically updates the heuristic parameter based on the entropy value of current pheromone information.

```
Procedure: UpdatePheromoneTrails
begin
    for i=1 to n-1 do
        tau ← 1/ bestLength
        evaporation ← (1- ρ) * pheromone[bestPath[i]][bestPath[i+1]]
        deposition ← ρ * tau;
        pheromone[bestPath[i]][bestPath[i+1]] ← evaporation +deposition
pheromone[bestPath[i+1]bestPath[i]]←pheromone[bestPath[i]][bestPath[i+1]]
 end for
end
```

Fig. 9. Global pheromone updating procedure

6 EXPERIMENTAL RESULTS (ANALYSIS OF TOUR LENGTH RESULTS)

Firstly, the proposed algorithm was tested to see how it performed; the eil51 case was chosen as test case. The results were shown over 20 trials with 20 iterations per trial (run) for eil51. The parameters used in this test case are α =1, β=5, ρ= 0.1, number of ants is 10 and number of iterations is 20. The results show that how big deviation from optimal distance when running the proposed system several times both with and without optimization with local search optimization. The tour length (distance) is represented as in unit.

As seen in Figure 10, the proposed algorithm gets an average of tour length, 426.6 when using optimization with local search, which is 0.6 higher than the optimal of 426. The deviation of the proposed algorithm's result (degree of approximation) is 0.14% from the optimal distance. The proposed algorithm's results vary from 426 (0% deviation) to 428 (0.47% deviation) and considering half of the running results are converged to the optimal and average length of the runs are much closed to the optimal. Moreover, Figure 11 and 13 shows the analysis results that are commonly used (can be found in literature) can be shown graphically as in best tour-so far solution, tour best and standard deviation respectively.

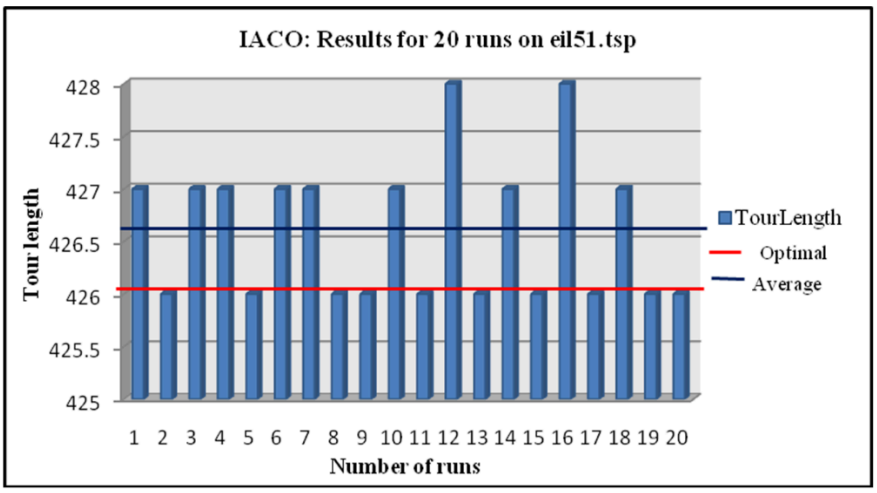

Fig. 10. Results the eil51 instance using local search (2-Opt)

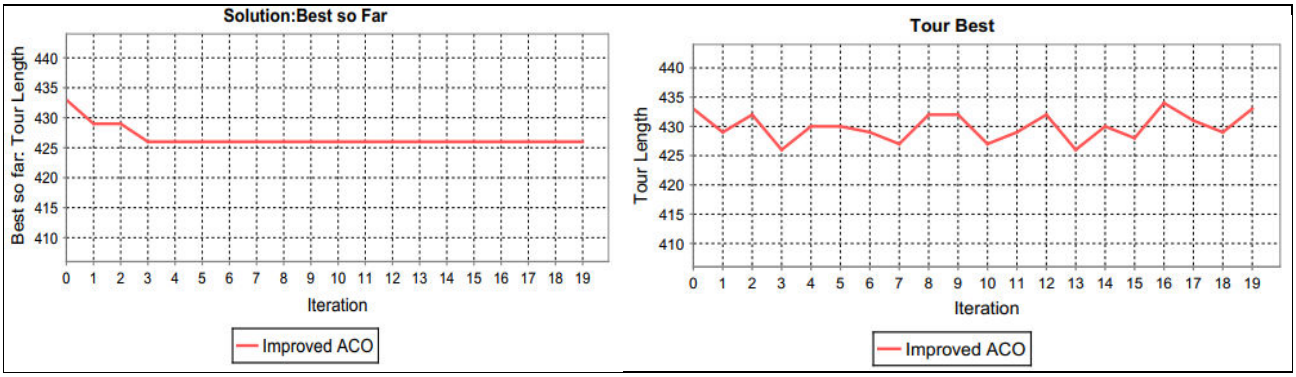

Fig. 11. Best so far solution of tour length and Tour best result for eil51 instance

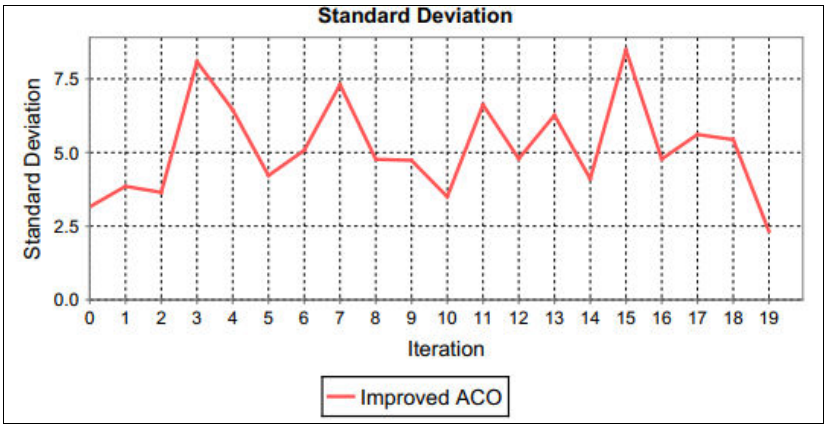

Fig. 12. Standard deviation of tour length for eil51 instance

Iteration	Best Tour	Average Tour Best	Standard Deviation
1	433.0	433.000	3.156
2	429.0	431.000	3.852
3	429.0	430.333	3.646
4	426.0	429.250	8.088
5	426.0	428.600	6.447
6	426.0	428.167	4.219
7	426.0	427.857	5.075
8	426.0	427.625	7.308
9	426.0	427.444	4.771
10	426.0	427.300	4.734
11	426.0	427.182	3.499
12	426.0	427.083	6.621
13	426.0	427.000	4.792
14	426.0	426.929	6.264
15	426.0	426.867	4.105
16	426.0	426.812	8.486
17	426.0	426.765	4.780
18	426.0	426.722	5.612
19	426.0	426.684	5.437
20	426.0	426.650	2.326

Founded solution: 426.0 at iteration: 4

Fig. 13. Tour length results for eil51 instance

6.1 COMPARISON OF CONVERGENCE SPEED

In second experiment, the proposed approach is compared with DSMACS [7] in terms of convergence speed. In order to compare the proposed system with DSMACS, some instances of TSP that are the same as ones used in DSMACS are chosen. A comparison of the final solution and convergence number between the proposed algorithm (Improved ACO) and DSMACS are shown in Table 5.2 (the results of DSMACS are directly taken from Reference [7]). It can be seen that the performance of the proposed IACO algorithm is wonderful. It not only finds out the global optimal solutions for the following TSP instances but also has very quick convergence speed.

Table 1. Comparison of the final solution and convergence number between IACO and DSMACS

TSP Problem	Best length of IACO	Best length of DSMACS	Convergence number of IACO	Convergence number of DSMACS
eil51	426	426	4	5
berlin52	7542	7542	3	4
st70	675	N/A	3	N/A

6.2 COMPARISON EXPERIMENTS

For the purpose of demonstrating the efficiency of the improved ACO algorithm proposed in this paper, there have constructed a simulation and applied it to problems from TSPLIB library [8]: In this study the proposed algorithm results compare to the results of ACS algorithm in the aspects of algorithm convergence and experiment results. The ACS algorithm combines with local search. In all cases, the proposed algorithm shows better performance than the ant colony system algorithm.

Table 2. Tour Length Results and Relative Errors (deviation) on several TSP instances

TSP problems	Optimum (1)	IACO			ACS		
		Best length (2)	Average tour length	Relative errors (deviation) ((2)-(1))/(1)	Best length (3)	Average tour length	Relative errors (deviation) ((3)-(1))/(1)
ch130	6110	6123	6175.3	0.21%	6144	6200.6	0.56%
ch150	6528	6528	6573.9	0%	6548	6600.83	0.31%
d198	15780	15815	15894.03	0.22%	15900	15994.77	0.76%
kroB100	22141	22141	22183.33	0%	22146	22278.5	0.02%
kroC100	20749	20749	20789.6	0%	20753	20906.1	0.02%
kroD100	21294	21294	21379.6	0%	21309	21584.7	0.07%
kroE100	22068	22068	22135.77	0%	22116	22284.67	0.22%
kroA150	26524	26524	26806.5	0%	26820	27189.47	1.08%
pr76	108159	108159	108303.5	0%	108304	108723.4	0.13%
pr124	59030	59030	59110.67	0%	59076	59228.23	0.08%
pr152	73682	73682	73772.73	0%	73818	74243.67	0.18%
pr226	80369	80377	80628.16	0.01%	80524	80959.67	0.19%
rat195	2323	2339	2356.17	0.69%	2352	2379.27	1.25%

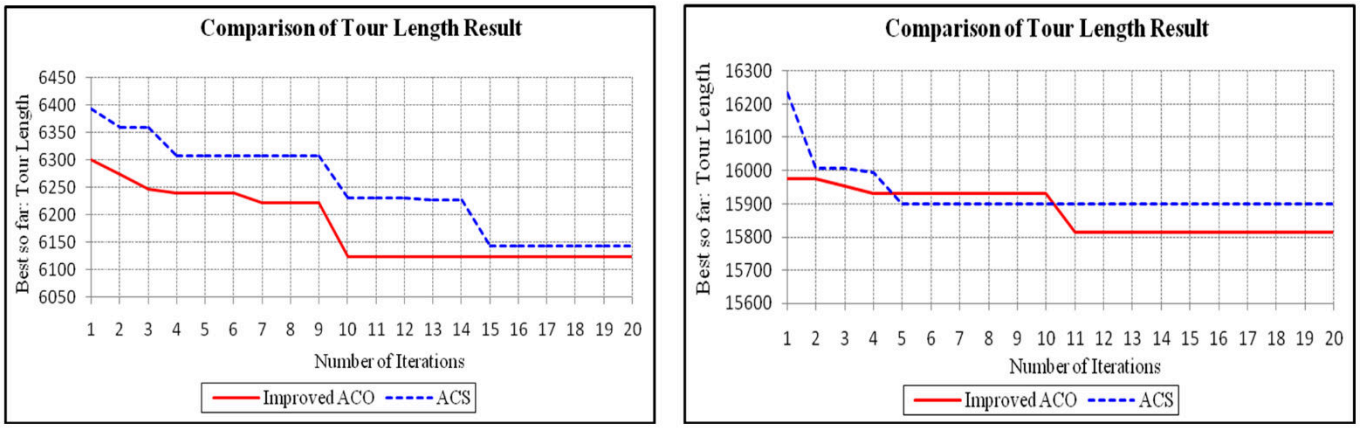

Fig. 14. Comparison of the tour length result of ch130 and d198

7 CONCLUSION

An algorithm based on the ACO approach called "Improved Ant Colony Optimization Algorithm" is designed and implemented to induce in combinatorial optimization problem as in travelling salesman problems. Compared to implementations of famous ant colony system, the implementations in this paper reduce the search space and computation time from the dynamic candidate list strategy and adapting parameter is applied to perform efficiently. Then, the resulted tours have been improved by the local search algorithm. The proposed algorithm showed many interesting features. It also meets the requirements of the research and figures out some new findings during the analysis. The proposed approach has proved to be better than using a single approach to solve a NP Complete problem like TSP. The solutions obtained for some instances of TSP, showed to be reaching to the optimum values. In some cases the solutions converged to the optimum values and some were very close to the optimum solutions. A very interesting observation seen was that time taken to find a solution was flexible.

ACKNOWLEDGEMENT

I would like to express my heartfelt thanks to all my teachers for their valuable advice, helpful comments, kindness support, and precious time for my research. Most importantly, none of this would have been possible without the love and patience of my family throughout my research. My heartfelt thanks also extend to all my colleagues and friends for their help, support, interest and valuable hints for discussions about research.

REFERENCES

[1] A. Colorni, M. Dorigo, and V.Maniezzo, "An Investigation of some properties of an Ant Algorithm", Apperaed in Proceedings of The Parallel Problem Solving from Nature Conference (PPSN 92), Brussels, Belgium, 1992, Elsevier Publishing, 509-520.

[2] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," IEEE Transactions on System, Man, and Cybernetics, Part B, Vol.26, pp. 29-41, 1996.

[3] M. Dorigo and L. M. Gambardella, "Ant Colony System: A cooperative learning approach to the traveling salesman problem," IEEE Transactions on Evolutionary Computation, Vol.1, No.1, April, 1997.

[4] M. Dorigo and T. St$\ddot{u}$tzle, "The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances", In: F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics. Kluwer Academic Publishers, 2002.

[5] J-Luc. Ngassa, J. Kierkegaard, K. Helsgam, "ACO and TSP", Roskilde University, May 2007.

[6] J. N. MacGregor and Y. Chu, "Human Performance on the Traveling Salesman and Related Problems: A Review", The Journal of Problem Solving, Volume 3, No. 2, 2011.

[7] C-Xue. Wang, D.-Wu. Cui, Y-Kun. Zhang and Z-Rong. Wang, "A Novel Ant Colony System Based on Delauney Triangulation and Self-adaptive Mutation for TSP", International Journal of Information TechnologyVol.12, No.3, 2006.

[8] TSPLIB WebPage, http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/tsp/

[9] C-Mihaela Pintea, D. Dumitrescu, "Improving Ant System Using a Local Updating Rule", Proceedings of the Seventh International Symposium and Numeric Algorithms for Scientific Computing (SYNASC'05), IEEE 2005.

[10] J. Han, Y. Tian, "An Improved Ant Colony Optimization Algorithm Based on Dynamic Control of Solution Construction and Mergence of Local Search Solutions", Fourth International Conference on Natural Computation, IEEE, 2008.

[11] T. St$\ddot{u}$tzle and H.H. Hoos, "Max-Min Ant System", Future Generation Computer Systems 16(8): 889-914, 2000.

[12] C-Mihaela Pintea, D. Dumitrescu, "Improving Ant System Using a Local Updating Rule", Proceedings of the Seventh International Symposium and Numeric Algorithms for Scientific Computing (SYNASC'05), IEEE 2005.

[13] H. Md. Rais, Z. A. Othman, A.R. Hamdan, "Improvement DACS3 Searching Performance using Local Search", Conference on Data Mining and Optimization, IEEE, 27-28 October 2009.

[14] R. Gan, Q. Guo, H. Chang, Y. Yi, "Improved Ant Colony Optimization Algorithm for the Traveling Salesman Problems", Jouranl of Systems Engineering and Electronics, April 2010, pp 329-333.