Applied Soft Computing 11 (2011) 1256–1262

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesmen problems

Soheil Ghafurian*, Nikbakhsh Javadian

Department of Industrial Engineering, Mazandaran University of Science and Technology, Iran

ARTICLE INFO

Article history:
Received 26 June 2007
Received in revised form 7 March 2010
Accepted 14 March 2010
Available online 19 March 2010

Keywords:
Ant system
Ant colony optimization
Multiple traveling salesmen problem
Multi-depot
Fixed destination

ABSTRACT

The fixed destination multi-depot multiple traveling salesmen problem (MmTSP) is a problem, in which more than one salesmen depart from several starting cities and having returned to the starting city, form tours so that each city is visited with exactly one salesman, and the tour lengths stay within certain limits. This problem is of a great complexity and few investigations have been done on it previously. In this paper an ant system is designed to solve the problem and the results are compared to the answers obtained by solving the same problems by Lingo 8.0 which uses exact methods.

1. Introduction

Multiple traveling salesmen problem (mTSP) is a generalized form of the well-known Traveling Salesman Problem (TSP), in which instead of one salesman, several of them should visit the customer nodes so that each customer node will be visited by exactly one salesman. If the salesmen depart from several depots instead of just one, then we will have the multi-depot multiple traveling salesmen problem (MmTSP), which represents a higher complexity. In this problem, if every salesman has to end his tour at its starting node, we will have the fixed destination MmTSP which is investigated in this paper; otherwise, the case will be a non-fixed destination MmTSP. A few previous works have been found on the MmTSP by the authors and that is the reason why the algorithm provided here is compared with an exact method.

The ant system (AS) was introduced by Colorni, Dorgio and Maniezzo [1,3,4] and has proved to be a perfectly acceptable meta-heuristic for a lot of NP-hard problems. Perhaps the most known application of the Ant System is the TSP case [1], in which the AS has shown a very high efficiency. Due to the similarities between TSP and our problem the ant system seemed to be a good meta-heuristic choice to solve the problem with.

The AS was inspired by the behavior of the real ants in nature. Every ant leaves an amount of pheromone on the path it passes and chooses the path with more pheromone left on it from the previous ants. Then, since more ants pass the shorter roots, gradually, more and more ants choose the shorter root and as a result, the colony finds the best way to get to the food.

Ant colony algorithms have proved to be really fast meta-heuristics, but they should be designed for every problem specifically, and sometimes finding an efficient design gets so hard that combining the ant system with another meta-heuristic gets inevitable. The ant system has also disadvantages, like having many parameters, which make the parameter tuning harder.

2. Related work

Many works have been done concerning various cases of the mTSP but to the best knowledge of the authors, there were no heuristics for the special case of the mTSP addressed in this paper and that is why the algorithm presented here is compared with the exact method used by Lingo software. Some of the heuristics tackling the mTSP problem are as follows.

One of the first heuristics addressing the problem of m tours in TSP with some side conditions is due to Russell [6], although the solution procedure is based on transforming the problem to a single TSP on an expanded graph. The algorithm is an extended version of the Lin and Kernighan [7] heuristic originally developed for the TSP. Another heuristic based on an exchange procedure for the mTSP is given by Potvin et al. [8].

A parallel processing approach to solve the mTSP using evolutionary programming is proposed by Fogel [9]. The approach considers two salesmen and an objective function minimizing the

* Corresponding author.
E-mail addresses: SoheilGhafurian@gmail.com (S. Ghafurian), NiJavadian@ustmb.ac.ir (N. Javadian).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.03.002

difference between the lengths of the routes of each salesman. Problems with 25 and 50 cities were solved and it is noted that the evolutionary approach obtained exceedingly good near-optimal solutions.

Several artificial neural network (NN) approaches have also been proposed to solve the mTSP, but they are generally extended versions of the ones proposed for the TSP.

Wacholder et al. [10] have extended the Hopfield-Tank ANN model to the mTSP but their model has been evaluated to be too complex with its inability to guarantee feasible solutions [11]. Hsu et al. [12] presented a neural network approach to solve the mTSP, based on solving m standard TSPs. The authors state that their results are superior to that of Wacholder et al. [10]. A self-organizing NN approach for the mTSP is due to Vakhutinsky and Golden [13], which is based on the elastic net approach developed for the TSP. Another self-organizing NN approach for the mTSP is proposed by Goldstein [14]. Torki et al. [15] describe a self-organizing NN for the VRP based on an enhanced mTSP NN model.

Recently, Modares et al. [16] and Somhom et al. [11] developed a self-organizing NN approach for the mTSP with a min-max objective function, which minimizes the cost of the most expensive route among all salesmen. Their approach seems to outperform the elastic net approach.

Utilizing genetic algorithms (GAs) for the solution of mTSP seems to be first due to Zhang et al. [17]. A recent application by Tang et al. [18] use genetic algorithms to solve the mTSP model developed for hot rolling scheduling. The approach is based on modeling the problem as an mTSP, converting it into a single TSP and applying a modified genetic algorithm to obtain a solution. Yu et al. [19] also use GAs to solve the mTSP in path planning.

Tabu search is used in solving a mTSP with time windows by Ryan et al. [20]. The authors offer an integer linear programming formulation, but solve the problem through a reactive tabu search algorithm within a discrete event simulation framework.

Recently, Song et al. [21] proposed an extended simulated annealing approach for the mTSP with fixed costs associated with each salesman. The approach is tested on an mTSP with 400 cities and three salesmen, which, however, required about 51 min to be solved on an IBM PC-586 (400 MHz).

Gomes and Von Zuben [22] present a neuro-fuzzy system based on competitive learning to solve the mTSP along with the capacitated VRP. The proposed method employs unsupervised learning of the network guided by a fuzzy rule base. Sofge et al. [23] implemented and compared a variety of evolutionary computation algorithms to solve the mTSP, including the use of a neighborhood attractor schema, the shrink-wrap algorithm for local neighborhood optimization, particle swarm optimization, Monte-Carlo optimization, genetic algorithms and evolutionary strategies (according to Bektas [5]).

3. Problem statement

3.1. Problem definition and notation

Consider a complete directed graph $G=(V,A)$, where V is a set of n nodes (vertices), A is a set of arcs and $C=(c_{ij})$ is the cost (distance) matrix associated with each arc $(i,j)\in A$. The cost matrix C can be either symmetric or asymmetric. Now let the node set be partitioned such that $V = V' \cup D$, where the first d nodes of V are depot set D. There are initially m_i salesmen located at depot i and the total number of salesmen is m. Also let $V' = \{d+1, d+2, \ldots, n\}$ be the set of customer nodes.

The MmTSP consists of finding tours for all the salesmen such that all customers are visited exactly once; the number of customers visited by a salesman lies between a predetermined interval and the total cost of all the tours is minimized (or maximized). If the problem is to determine a total of m tours such that the salesmen must return to their original depots, it is referred to as the fixed destination MmTSP. On the other hand, if the salesmen do not have to return to their original depots, but the number of salesmen at each depot should remain the same at the end as it was in the beginning, we have the non-fixed destination MmTSP [4]. In some cases (including ours) limitations are imposed to the number of customer cities in a tour, we show the upper and lower bounds for the number by L and K, respectively (for real life examples the reader is referred to Bektast [5]).

3.2. Problem formulation

The formulation for the fixed destination MmTSP presented in [4] is adopted here. But, because the problem was investigated in a maximization form, the objective function is turned to the maximization case.

First, the following binary variable is defined:

$$x_{ijk} = \begin{cases} 1, & \text{if the salesman departured from the } k\text{th node, passes the } ij \text{ link} \\ 0, & \text{otherwise} \end{cases}$$

Notice that x_{iik} is equal to zero.

For any traveler, u_i is the number of nodes visited on the traveler's path from the origin up to node i (i.e., the visit number of the ith node). L is the maximum number of nodes a salesman may visit; thus, $1 \le u_i \le L$ for all $i \ge 2$. In addition, let K be the minimum number of nodes a salesman must visit, i.e., if $x_{ikk}=1$, then $K \le u_i \le L$ must be satisfied:

$$\text{Maximize} \quad \sum_{k\in D}\sum_{j\in V'}(c_{kj}x_{kjk} + c_{jk}x_{jkk}) + \sum_{k\in D}\sum_{i\in V'}\sum_{j\in V'} c_{ij}x_{ijk} \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in V'} x_{kjk} = m_k, \quad k\in D, \tag{2}$$

$$\sum_{k\in D} x_{kjk} + \sum_{k\in D}\sum_{i\in V'} x_{ijk} = 1, \quad j\in V', \tag{3}$$

$$x_{kjk} + \sum_{i\in V'} x_{ijk} - x_{jkk} - \sum_{i\in V'} x_{jik} = 0, \quad k\in D, \quad j\in V', \tag{4}$$

$$\sum_{j\in V'} x_{kjk} - \sum_{j\in V'} x_{jkk} = 0, \quad k\in D, \tag{5}$$

$$u_i + (L-2)\sum_{k\in D} x_{kik} - \sum_{k\in D} x_{ikk} \le L-1, \quad i\in V' \tag{6}$$

$$u_i + \sum_{k\in D} x_{kik} + (2-K)\sum_{k\in D} x_{ikk} \ge 2, \quad i\in V' \tag{7}$$

$$\sum_{k\in D} x_{kik} + \sum_{k\in D} x_{ikk} \le 1, \quad i\in V', \tag{8}$$

$$u_i - u_j + L\sum_{k\in D} x_{ijk} + (L-2)\sum_{k\in D} x_{jik} \le L-1, \quad i \ne j, \quad i,j\in V', \tag{9}$$

$$x_{ijk} \in \{0,1\}, \quad i,j\in V, \quad k\in D. \tag{10}$$

In this formulation, constraints (2) ensure that exactly m_k salesmen depart from each depot $k\in D$. Constraints (3) ensure that each customer is visited exactly once. Route continuity for customer nodes and depot nodes is represented respectively by constraints (4) and (5). Constraints (6) and (7) impose upper and lower limits to the tours, respectively. In addition, if i is the first node on a tour, these constraints oblige u_i to be equal to 1. Tours with just one customer node are prohibited with constraints (8). Finally, constraints (9) are subtour elimination constraints (SECs) in that they break all subtours between customer nodes. Subtours are closed tours made of customer nodes and with no depots as the starting or ending point, which might be found in the solutions if there are no SECs considered in the model.

4. Solution construction

Any solution is constructed using one artificial ant. In order to produce a solution for the fixed destination mMTSP problem, the ant should complete one tour for any salesman. At the beginning of any tour, the ant chooses one depot randomly and chooses the next cities consecutively. The tour continues until the depot is chosen again. Having completed a tour, the ant chooses the next depot for the new tour randomly.

The number of tours started and ended at any depot should be equal to the number of agents associated with that depot.

4.1. Transition probability

The transition probability function is basically very similar to the one defined in [1], but some modifications are done in order to produce feasible solutions for the fixed destination mMTSP.

As in [1] we define a list of nodes which the kth ant cannot chose as the next node. This list is called $\mathbf{tabu_k}$, which includes all the customer nodes which have been visited by the kth ant until the current state in addition to all the depots except the one which the current tour has been started from.

The transition probability, p_{ij} is the probability of the node j being chosen by the ant currently at the node i, and is obtained by the following formula:

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha [c_{ij}]^\beta}{\sum\limits_{k \in \text{allowed}_k} [\tau_{ik}(t)]^\alpha [c_{ik}]^\beta} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases}$$

where $\text{allowed}_k = \{N - \text{tabu}_k\}$ and α and β are parameters that control the relative importance of trail versus greediness of the algorithm.

The following modifications are imposed on the transition probability in special cases not to produce infeasible solutions:

- In any of the conditions below, the ant is coerced to return to the depot and finish the tour:
  - The ant has visited L customer cities in its current tour.
  - The number of customer cities per left agent falls below K if the ant continues its current tour. If the ant does not return to the depot node at this situation, it will not be able to complete the consequent tours with the minimum customer cities.
- In any of the conditions below, the tabu list will include the depot city of the current tour too:
  - The ant has not visited K customer cities in its current tour yet.
  - The ant is at its last tour and not all the customer nodes are visited.
  - The number of customer cities per left agent exceeds L. In this case if the ant returns to the depot node, it will not be able to complete the consequent tours with intermediate nodes less than the upper limit L.

4.2. Trail update

At the end of any cycle the trail matrix will be updated using the solutions produced by the artificial ants. Letting t be the number of the finished cycle and τ_{ij}^t the amount of trail associated with the edge ij at the tth cycle, the new trail amount will be obtained by the following formula:

$$\tau_{ij}^{t+1} = \rho \tau_{ij}^t + \Delta\tau_{ij}$$

in which ρ is the persistence coefficient of the trail and $(1 - \rho)$ the evaporation rate $(0 \le \rho \le 1)$ and

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k$$

where $\Delta\tau_{ij}^k$ is the quantity of trail laid on edge ij by the kth ant between cycles t and $t+1$; it is given by

$$\Delta\tau_{ij}^k = \begin{cases} F_k & \text{if the } k\text{th ant uses edge } (i,j) \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

Having supposed F_k is the objective function of the solution produced by the kth ant.

4.3. The algorithm

The ant algorithm consists of repetitive cycles. At any cycle, first, *antnum* artificial ants produce solutions using the trail matrix and the solution construction mentioned above. Then the trail matrix updates, using the solutions produced.

The mentioned cycles continue running consequently until a special number of successive cycles with no improvement in the best solution found so far are observed. The number is shown as the variable *cyclenum* here:

- Initialize *antnum*, *cyclenum*, *inittrail*, α, β and ρ.
- Set all the τ_{ij}s to *inittrail*.
- Do:
  - Generate *antnum* solutions by the solution construction method explained above.
  - Update the trail matrix by the solutions generated.
- While (there is been an improvement in the best answer found on the last *cyclenum* cycles).
- Return the best solution found.

5. Experimental results

5.1. Generating test problems

To the best knowledge of the authors, there were no test problems for the mMTSP on the accessible libraries on the Internet. Therefore it seemed inevitable to produce test problems using a proper method.

In order to show the growth of the problems on a two-dimensional graph, we tried to produce our test problems using a single parameter n which is the same as the total number of the cities in the problem including the depots. Given this parameter, the problem will be generated by the following process:

1. The number of the depot nodes, d, is set to $[n/10]$.
2. K is a randomly generated integer so that $2 \le K \le (n-d)/d$.
3. L is a randomly generated integral integer so that $\lceil (n-d)/m \rceil \le L \le n-d$. Notice that in case $k=2$ and $L=n-d$, the problem turns into a mMTSP with no limitations on the number of intermediate nodes in each tour.
4. m is a randomly generated integer so that $d \le m \le (n-d)/K$. The m agents are randomly assigned to the depot nodes so that every depot has at least one agent associated with.
5. c_{ij}'s are random integers generated uniformly in the span $100 \le c_{ij} < 200$.

As seen, though the problem generation is not completely random due to the dependency of parameters on n, they are random enough to show the efficiency of our algorithm.

For each $n \in \{10,20,30,40,50,60\}$, five problems were generated.

5.2. Lingo 8.0

Lingo 8.0 is a software tool designed to efficiently build and solve linear, non-linear, and integer optimization models. All the methods used by this application are exact methods, which ensure that the solutions obtained are globally optimal.

In the meanwhile, Lingo 8.0 suffers all the downsides of the exact algorithms, including the enormous increase of the solution time by the growth of the size of NP-hard problems. For example, only the test problems with $n < 50$ could be solved by Lingo 8.0 in a reasonable time, which shows both the complexity of the mMTSP and the inability of the exact methods to address such problems.

Throughout the experimental results section, the proportion of the obtained solution to the best solution is used to measure the efficiency of the algorithm and the parameters. The best solution consists of the optimal solution given by Lingo 8.0 for the problems with $n < 5$ and the best solutions found in all the experiments by the ant algorithm for those with $n > 4$.

5.3. Parameter tuning

Finding the best values for the parameters of the AS could be a complicated problem itself, because changing the value of any parameter could affect the optimum value of the others. What is usually done instead is an experimental approximation of the best values.

In our work, we adopted the values 1 and 5 for α and β from [1] due to the similarity of the problems. For the rest of the parameters we implemented a set of experiments on the test problem generated with $n \in \{30,40,50,60\}$ (five problems for each n). For each problem, the algorithm was run for ten times and the average answer and time were registered.

5.3.1. Number of the ants

Though the complexity of the problem has a dependency on the number of agents, we noticed that for the problems in the same group (each group with the same n value) and different number of agents, the solution time does not differ significantly. The reason is that as the number of the agents grows, the tour lengths they could have shorten and this retaliates the complexity caused by the growth of the agents' number.

Considering this fact, we tried to find a liaison between the optimum number of ants and the number of the nodes as an approximate index of the problem complexity. In order to examine the existence of such a liaison, we solved the problems with zn ants, z spanning from 1 to 15, hoping the optimum ant number will be a coefficient of the number of the nodes. The results can be seen in Tables 1 and 2 and Fig. 1. In these implementations the values for the other parameters were $\rho = 0.6$, *cyclenum* = 5, and *inittrail* = 1.

As seen in the charts, the major improvements in the answer, for all groups of the problems, occur before 5. After 10 no noticeable improvement has occurred, showing the existence of the liaison mentioned above. Since the average CPU time for the coefficient 10 is almost twice as large as that of 5, we spared the improvements after 5 and chose the value $5n$ as the number of ants in our algorithm.

Table 1
Average proportion to the best solution obtained by solving the AS algorithm with zn ants.

z	$n = 30$	$n = 40$	$n = 50$	$n = 60$	Average
1	0.972	0.969	0.977	0.977	0.974
2	0.973	0.972	0.981	0.982	0.977
3	0.975	0.975	0.982	0.982	0.979
4	0.975	0.975	0.983	0.983	0.979
5	0.977	0.975	0.983	0.983	0.980
6	0.978	0.976	0.984	0.984	0.981
7	0.978	0.977	0.984	0.984	0.981
8	0.979	0.977	0.985	0.984	0.981
9	0.979	0.978	0.985	0.985	0.982
10	0.979	0.978	0.985	0.985	0.982
11	0.980	0.978	0.985	0.986	0.982
12	0.979	0.978	0.986	0.985	0.982
13	0.981	0.978	0.985	0.986	0.983
14	0.981	0.979	0.986	0.986	0.983
15	0.981	0.979	0.986	0.986	0.983

Table 2
Average CPU times in seconds for the implementations with different number of ants.

z	$n = 30$	$n = 40$	$n = 50$	$n = 60$	Average
1	0.13	0.34	0.64	1.46	0.64
2	0.25	0.65	1.52	2.89	1.33
3	0.41	1.01	2.40	4.49	2.08
4	0.60	1.36	3.06	6.17	2.80
5	0.70	1.81	4.04	6.98	3.38
6	0.84	2.24	4.80	9.16	4.26
7	1.04	2.54	5.20	10.25	4.76
8	1.14	2.84	6.73	12.07	5.69
9	1.34	3.62	7.05	13.58	6.40
10	1.46	3.65	7.40	14.36	6.72
11	1.70	4.08	8.84	16.70	7.83
12	1.63	4.36	9.81	18.56	8.59
13	1.96	4.88	10.78	20.00	9.41
14	2.19	5.46	11.37	22.54	10.39
15	2.01	6.25	11.82	23.37	10.86

5.3.2. Evaporation rate

Having the previously used number of ants replaced with the one found, all the mentioned problems were solved again with ρ ranging from 0.1 to 0.9 increasing by 0.1 to find the best value. The results can be seen in Tables 3 and 4 and Fig. 2.

As seen, the best solutions have been obtained by $\rho = 0.1$ and increasing the value, the efficiency declines significantly. These results brought about the question as to if there are any better values less than 0.1. To find the answer, the problems with $n = 30$ were solved again with ρ's from 0.02 to 0.2 with the intervals of 0.02. The results showed now significant difference between the values less than 0.1. Though, the answers from 0.1 were slightly better than others. The results can bee seen in Table 5. In the next implementations ρ is set to 0.1.

a larger percentage indicates a better performance

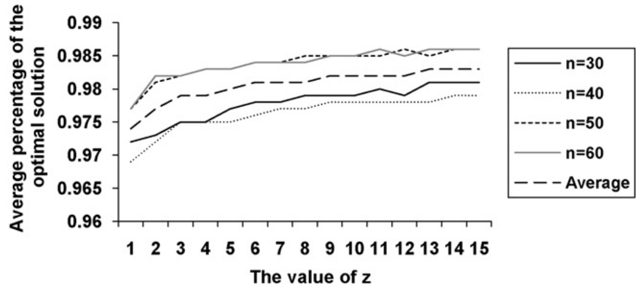

Fig. 1. The average proportion to the best solution obtained by the AS grows as the ant number increases but in all groups of the problems, the major improvements occur before z value of 5.

Table 3
Average proportion to the best solution obtained by solving the AS algorithm with different values of ρ.

ρ	$n = 30$	$n = 40$	$n = 50$	$n = 60$	Average
0.1	0.980	0.979	0.987	0.986	0.983
0.2	0.979	0.978	0.986	0.986	0.982
0.3	0.978	0.979	0.985	0.985	0.982
0.4	0.979	0.978	0.985	0.985	0.982
0.5	0.979	0.978	0.984	0.985	0.981
0.6	0.977	0.977	0.983	0.984	0.980
0.7	0.977	0.974	0.983	0.982	0.979
0.8	0.974	0.973	0.980	0.981	0.977
0.9	0.972	0.970	0.977	0.976	0.974

Table 4
Average CPU times in seconds for the implementations in Table 3.

ρ	$n = 30$	$n = 40$	$n = 50$	$n = 60$	Average
0.1	0.67	1.64	3.92	6.89	3.28
0.2	0.68	1.64	3.80	6.91	3.25
0.3	0.62	1.75	4.01	6.82	3.30
0.4	0.71	1.79	4.41	7.23	3.53
0.5	0.73	1.86	4.07	8.26	3.73
0.6	0.68	1.93	4.48	9.27	4.09
0.7	0.76	1.76	4.44	8.16	3.78
0.8	0.67	1.77	4.72	8.70	3.96
0.9	0.58	1.70	4.32	7.68	3.57

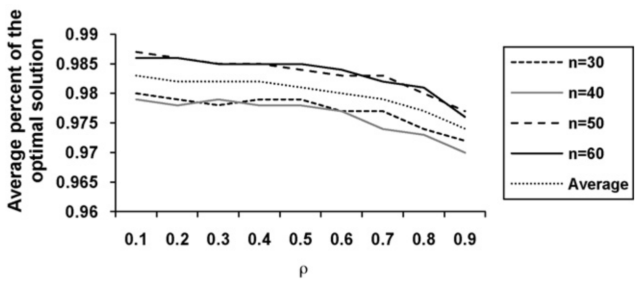

Fig. 2. As seen the efficiency of the algorithm declines as the amount of ρ increases and the best value for ρ seems to be 0.1 for all groups of problems solved.

5.3.3. Initial amount of trail

It was expected that the initial amount of trail not affect the efficiency of the algorithm as indicated in [1]. To assure, one of problems with $n = 30$ and the initial trail amount of 1, 10, 100, 1000 and 10,000 was solved (ten times for each value) and the results, shown in Table 6, showed no major differences in the average answers. In the remainder of the implementations, the initial amount of trail is arbitrarily set to 1 as before.

5.3.4. Stopping criteria

As mentioned above, when a certain number of consecutive cycles with no improvement in the best answer found are observed, the algorithm stops. The best value of this parameter – called *cyclenum* hereafter – was found by solving the previously mentioned problems with *cyclenums* belonging to the [1,13] span. Again the results can be seen in Tables 7 and 8. Considering the results, the improvements made with *cyclenums* greater than 5 can be spared due to their minority and the increasing CPU time. Therefore, a value of 5 is supposed to be favorable and used in the proceeding sections as it luckily was in the previous ones! (Fig. 3).

Table 6
Even major changes in the initial amount of trail laid on the edges cause no significant changes in the average proportion to the best solution (APBS).

Inittrail	APBS	CPU time
1	0.982	0.73
10	0.980	0.76
100	0.979	0.64
1000	0.981	0.66
10000	0.979	0.65

Table 7
Average proportion to the best solution obtained by solving the AS algorithm with different values of the variable *cyclenum* which is the allowed number of consecutive cycles with no improvements in the answer and works as the stopping criteria of the AS.

Cyclenum	$n = 30$	$n = 40$	$n = 50$	$n = 60$	Average
1	0.968	0.968	0.977	0.977	0.972
2	0.975	0.974	0.982	0.982	0.978
3	0.978	0.976	0.983	0.984	0.980
4	0.979	0.978	0.985	0.985	0.982
5	0.981	0.978	0.986	0.986	0.983
6	0.980	0.980	0.986	0.986	0.983
7	0.981	0.980	0.987	0.987	0.984
8	0.982	0.980	0.987	0.987	0.984
9	0.982	0.981	0.989	0.988	0.985
10	0.984	0.982	0.988	0.988	0.985
11	0.984	0.982	0.988	0.988	0.985
12	0.984	0.982	0.989	0.988	0.986

Table 8
Average CPU times in seconds for the implementations in Table 7.

Cyclenum	$n = 30$	$n = 40$	$n = 50$	$n = 60$	Average
1	230.20	659.20	1327.20	2305.60	1130.55
2	327.40	1020.20	2024.20	3473.00	1711.20
3	446.80	1260.80	2777.20	4659.00	2285.95
4	590.20	1514.80	3407.40	5207.20	2679.90
5	680.00	1708.60	4025.80	5957.80	3093.05
6	769.20	2018.20	4236.20	7162.20	3546.45
7	818.60	2348.60	4905.60	7372.60	3861.35
8	913.80	2374.00	5462.20	8487.80	4309.45
9	990.40	2644.00	5335.60	9763.20	4683.30
10	1198.60	2846.60	5308.60	9606.00	4739.95
11	1278.60	3135.40	6292.00	10756.00	5365.50
12	1333.20	3080.00	6329.40	10824.60	5391.80

5.4. Comparison with Lingo 8.0

In order to examine the accuracy and efficiency of the algorithm a comparison between it and Lingo 8.0 is done. The problems with $n = \{10,20,30,40\}$ are solved with the ant system 30 times each and also once with Lingo 8.0. The results are shown in Tables 9 and 10. In the ant implementations, the parameter tuning done before is considered.

The algorithm is coded in Microsoft Visual C++ 6.0 and the implementations were done on a PC with Intel 3.0 GHz Pentium 4 CPU.

Table 5
Average proportion to the best solution (APBS) obtained by solving the AS algorithm with different values of ρ between 0.02 and 0.2. As seen for the values less than 0.1 no significance is seen in the behavior of the algorithm. The better answers are accompanied with longer CPU times and are probably not caused by the pheromone evaporation rate change.

ρ	0.02	0.04	0.06	0.08	0.1	0.12	0.14	0.16	0.18	0.2
APBS	0.981	0.980	0.981	0.981	0.981	0.980	0.981	0.981	0.980	0.980
Time	0.73	0.68	0.70	0.72	0.67	0.64	0.70	0.63	0.66	0.68

comparision results between AS and Lingo results with 5 problems

Table 9
Comparison results between our AS and Lingo 8.0. As seen, for $n = 10$ the AS shows the same properties of Lingo 8.0.

Group of the problem	Number of the problem	AS answer	Lingo answer	AS proportion to the optimal solution	AS time (s)	Lingo time (s)
	1	2872	2872	1.000	0.009	0
	2	2846	2846	1.000	0.015	0
$n = 10$	3	2807.7	2813	0.998	0.019	0
	4	3122	3122	1.000	0.013	0
	5	2817	2817	1.000	0.014	0
	1	5762.63	5811	0.992	0.191	2
	2	5747.33	5803	0.990	0.192	14
$n = 20$	3	6271.57	6343	0.989	0.173	2
	4	6052.9	6116	0.990	0.160	1
	5	5773.5	5842	0.988	0.157	2
	1	10311.3	10533	0.979	0.648	14
	2	8671.97	8814	0.984	0.622	46
$n = 30$	3	9479.17	9695	0.978	0.679	80
	4	8668.63	8825	0.982	0.628	1220
	5	9895.47	10099	0.980	0.779	388
	1	12071.3	12370	0.976	1.567	4348
	2	11552.3	11807	0.978	1.584	2012
$n = 40$	3	11564.9	11800	0.980	1.628	897
	4	11580.2	11812	0.980	1.694	5908
	5	11536.3	11773	0.980	1.656	363

Table 10
Average values of results for each group of the problems. Notice the huge difference in CPU time between Lingo 8.0 and AS in the last group while the average answer got from AS is 0.979 of that of Lingo 8.0.

	Average proportion to the optimal solution	Average AC time in seconds	Average Lingo time in seconds
$n = 10$	1.000	0.01	0
$n = 20$	0.990	0.17	4.2
$n = 30$	0.981	0.67	351.2
$n = 40$	0.979	1.63	87665.6

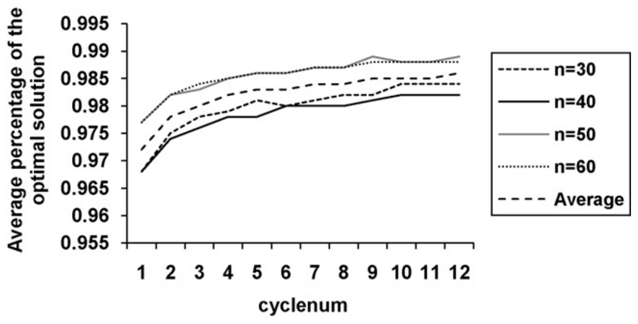

Fig. 3. The average proportion to the best solution obtained by the AS grows as *cyclenum* increases but in all groups of the problems, the major improvements occur before *cyclenum* value of 5.

5.5. Discussion of the results

As seen in Table 10, the ant system shows almost the same results as Lingo 8.0. But, as the dimensions of the problems increase, the Lingo 8.0 time magnificently grows. Whereas, just a little growth is observed in the AS time, to the point that for the problems with $n = 40$, the average CPU time for Lingo 8.0 is about 24 min versus the AS average CPU time of just 1.63 s which is 883 times shorter and shows no considerable growth.

While offering this considerable thrift in the solving time, the AS shows just a little decline in the quality of the answer. As seen for the fourth group of the problems, just 2.1 percent of the optimal solution is lost in return of the time saved.

6. Conclusion

In this paper we presented an ant system algorithm for the fixed destination multi-depot multiple traveling salesmen problem. Also, in order to examine the efficiency of the algorithm, test problems with various sizes were produced. Moreover, an applicable parameter tuning is done that makes it possible to solve new problems without any renewed tunings. Compared to Lingo 8.0 the experimental results demonstrate the efficiency of the algorithm which obtains answers very close to the optimal solution in a very short time compared to the exact approach used in Lingo 8.0. In addition to the MmTSP presented here, there are other kinds of the MmTSP with very few meta-heuristics designed for, which can be investigated.

References

[1] M. Dorgio, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systematic, Man and Cybernetics 26 (1996) 29.
[3] M. Dorgio, Optimization, Learning and Natural Algorithms, Doctoral Dissertation, Politecnico di milano, Italy (1992).
[4] I. Kara, T. Bektas, Integer linear programming formulations of multiple salesmen problems and its variations, European Journal of Operational Research 174 (2006) 1449–1458.
[5] T. Bektas, The multiple traveling salesman problem: an overview of formulations and solution procedures, The International Journal of Management Science 34 (2006) 209–219.
[6] R.A. Russell, An effective heuristic for the m-tour traveling salesman problem with some side conditions, Operations Research 25 (3) (1977) 517–524.
[7] S. Lin, B. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Operations Research 21 (1973) 498–516.
[8] J. Potvin, G. Lapalme, J. Rousseau, A generalized k-opt exchange procedure for the MTSP, INFOR 27 (4) (1989) 474–481.
[9] D.B. Fogel, A parallel processing approach to a multiple traveling salesman problem using evolutionary programming, in: Proceedings of the fourth annual symposium on parallel processing, Fullerton, CA, 1990, pp. 318–326.
[10] E. Wacholder, J. Han, R.C. Mann, A neural network algorithm for the multiple traveling salesmen problem, Biology in Cybernetics 61 (1989) 11–19.
[11] S. Somhom, A. Modares, T. Enkawa, Competition-based neural network for the multiple traveling salesmen problem with min max objective, Computers and Operations Research 26 (4) (1999) 395–407.
[12] C. Hsu, M. Tsai, W. Chen, A study of feature-mapped approach to the multiple travelling salesmen problem, IEEE International Symposium on Circuits and Systems 3 (1991) 1589–1592.

[13] I.A. Vakhutinsky, L.B. Golden, Solving vehicle routing problems using elastic net, Proceedings of the IEEE International Conference on Neural Network (1994) 4535–4540.
[14] M. Goldstein, Self-organizing feature maps for the multiple traveling salesmen problem, in: Proceedings of the IEEE International Conference on Neural Network, San Diego, CA, 1990, pp. 258–261.
[15] A. Torki, S. Somhon, T. Enkawa, A competitive neural network algorithm for solving vehicle routing problem, Computers and Industrial Engineering 33 (3–4) (1997) 473–476.
[16] A. Modares, S. Somhom, T. Enkawa, A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems, International Transactions in Operational Research 6 (1999) 591–606.
[17] T. Zhang, W.A. Gruver, M.H. Smith, Team scheduling by genetic search, in: Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials, vol. 2, 1999, pp. 839–844.
[18] L. Tang, J. Liu, A. Rong, Z. Yang, A multiple traveling salesman problem model for hot rolling scheduling in Shangai Baoshan Iron and Steel Complex, European Journal of Operational Research 124 (2000) 267–282.
[19] Z. Yu, L. Jinhai, G. Guochang, Z. Rubo, Y. Haiyan, An implementation of evolutionary computation for path planning of cooperative mobile robots, in: Proceedings of the Fourth World Congress on Intelligent Control and Automation, vol. 3, 2002, pp. 1798–1802.
[20] J.L. Ryan, T.G. Bailey, J.T. Moore, W.B. Carlton, Reactive Tabu search in unmanned aerial reconnaissance simulations, in: Proceedings of the 1998 Winter Simulation Conference, vol. 1, 1998, pp. 873–879.
[21] C. Song, K. Lee, W.D. Lee, Extended simulated annealing for augmented TSP and multi-salesmen TSP, in: Proceedings of the International Joint Conference on Neural Networks, vol. 3, 2003, pp. 2340–2343.
[22] L.D.T. Gomes, F.J. Von Zuben, Multiple criteria optimization based on unsupervised learning and fuzzy inference applied to the vehicle routing problem, Journal of Intelligent and Fuzzy Systems 13 (2002) 143–154.
[23] D. Sofge, A. Schultz, K. De Jong, Evolutionary Computational Approaches to Solving the Multiple Traveling Salesman Problem Using a Neighborhood Attractor Schema. Lecture Notes in Computer Science, vol. 2279, Springer, Berlin, 2002, pp. 151–60.