International Journal on
"Technical and Physical Problems of Engineering"
(IJTPE)
Published by International Organization of IOTPE

ISSN 2077-3528
IJTPE Journal
www.iotpe.com
ijtpe@iotpe.com

December 2014 | Issue 21 | Volume 6 | Number 4 | Pages 66-74

SHORT TERM POWER LOAD FORECASTING BASED ON COMPARISON OF ACS TO PROBABILISTIC TRAVELING SALESMAN PROBLEM

Naser M. Tabatabaei [1,2]   K. Asadian [1]   N.S. Boushehri [1,2]

1. Electrical Engineering Department, Seraj Higher Education Institute, Tabriz, Iran
n.m.tabatabaei@gmail.com, kiomars002@yahoo.com, nargesboush@yahoo.com
2. Taba Elm International Institute, Tabriz, Iran

Abstract- Accurate forecasting of power load has been one of the important issues in the electricity industry. Recently, along with the privatization and the deregulation, accurate forecasting of power load draws more and more attentions. There are many difficulties in the application of BP neural network, which is a very useful tool for the forecasting, such as the defining for the network structure and the local solution, which is easy to fall in to. To solve these problems, the back-propagation (BP) neural network short-term load forecasting method based on improved variable learning rate back propagation (IVL-BP) is present in this paper. Though introducing two threshold parameters for the mean square increasing and decreasing, the learning algorithm is sensitive to the error and convergence speed. Then use genetic algorithm to train network parameters until the error tending to some stable value. Then conduct BP algorithm with the optimized weights to achieve short-term load forecasting. The experimental results show that the load forecasting system based on this method has higher accuracy and real-time.

Keywords: Power Load Forecasting, ACS, Traveling Salesman.

I. INTRODUCTION

Power load prediction has attracted a great deal of attention from both the practice and academia. The short-term power load forecasting is very significant for the electric network's reliability and economic development. As short-term power load prediction is of crucial importance to the reliability and economic utilization of electric networks, it is drawing more and more attention from both the practice and academia. The aim of load forecasting is to make the best use of electric energy and relieve the conflict between supply and demand.

Inaccurate forecast of power load will leads to a great deal of loss for power companies. Bunn and Farmer pointed out that a 1% increase in forecasting error implied a 10 million increase in operating costs. The short-term forecasts refer to hourly prediction of electricity load demand for a lead-time ranging from 1h to several days ahead. In certain instances, the prediction of the daily peak load is the objective of short-term load forecasting, since it is the most important load during any given day. The quality of short-term hourly load forecasts has a significant impact on the economic operation of the electric utility since many decisions based on these forecasts have significant economic consequences.

II. INTRODUCTION TO ACS

The ACS differs from the previous ant system because of three main aspects: 1- The state transition rule provides a direct way to balance between exploration of new edges and exploitation of a priori and accumulated knowledge about the problem. 2- The global updating rule is applied only to edges, which belong to the best ant tour, and iii), while ants construct a solution a local pheromone updating rule (local updating rule, for short) is applied.

Informally, the ACS works as follows: ants are initially position on cities chosen according to some initialization rule (e.g., randomly). Each ant builds a tour (i.e., a feasible solution to the TSP) by repeatedly applying a stochastic greedy rule (the state transition rule). While constructing its tour, an ant also modifies the amount of pheromone on the visited edges by applying the local updating rule. Once all ants have terminated their tour, the amount of pheromone on edges modify again (by applying the global updating rule).

As was the case in ant system, ants are guide, in building their tours, by both heuristic information (they prefer to choose short edges) and by pheromone information. An edge with a high amount of pheromone is a very desirable choice. The pheromone updating rules are design so that they tend to give more pheromone to edges, which should be visit by ants. The ACS algorithm is report in Figure 3.

In the following, we discuss the state transition rule, the global updating rule, and the local updating rule. Thus, the distribution company could economically purchase a dynamic reactive power service from customers for perhaps 6 $/kVar. This practice would provide for better voltage regulation in the distribution system and would provide an alternate revenue source to help amortize the cost of PV and CHP installations.

A. Behavior of ACS

ACS uses a very aggressive search that focuses from the very beginning around the best-so-far tour T^{bs}. In other words, it generates tours that differ only in a relatively small number of arcs from the best-so-far tour T^{bs}. This is achieved by choosing a large value for q_0 in the pseudorandom proportional action choice rule (Equation (1)), which leads to tours that have many arcs in common with the best-so-far tour.

An interesting aspect of ACS is that while ants, their associated, traverse arcs Pheromone is diminishing, making them less attractive, and therefore favoring the exploration of still unvisited arcs. Local updating has the effect of lowering the pheromone on visited arcs so that they will choose with a lower probability by the other ants in their remaining steps for completing a tour. As consequence, the ants never converge to a common tour, as is also shown in Figure 1.

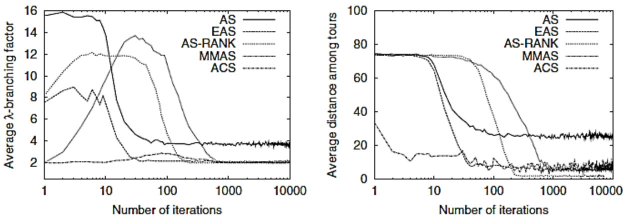

Figure 1. Comparing AS extensions: The plots give (left) the average *l*-branching factor, *l* ¼ 0:05, and (right) the average distance among the tours for several ACO algorithms on the symmetric TSPLIB instance kroA100. Parameters were set as in box 3.1, except for b which was set to *b* ¼ 2 for all the algorithms

B. Comparison of Ant System with its Extensions

There remains the final question about the solution quality returned by the various ACO algorithms. In Figure 2 we compare the development of the average solution quality measured in twenty-five trials for instance d198 (left side) and in five trials for instance rat783 (right side) of several ACO algorithms as a function of the computation time, which is indicated in seconds on the *x*-axis. We found experimentally that all extensions of AS achieve much better final solutions than AS, and in all cases the worst final solution returned by the AS extensions is better than the average final solution quality returned by AS. In particular, it can be observe that ACS is the most aggressive of the ACO algorithms and returns the best solution quality for very short computation times.

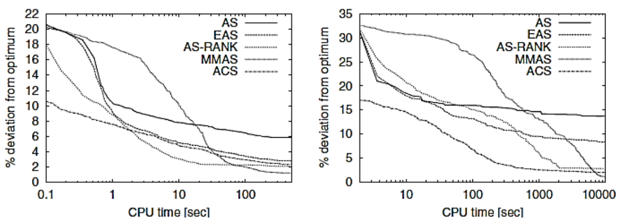

Figure 2. Comparing AS extensions: The plots give the development of the average percentage deviation from the optimum as a function of the computation time in seconds for AS, EAS, AS_{rank}, MMAS, and ACS for the symmetric TSPLIB instances d198 (left), and rat783 (right)

Differently, MMAS initially produces rather poor solutions and in the initial phases it is outperformed even by AS. Nevertheless, its final solution quality, for these two instances, is the best among the compared ACO algorithms. These results are consistent with the findings of the various published research papers on AS extensions: in all these publications, it found that the respective extensions improved significantly over AS performance. Comparisons among the several AS extensions indicate that the best performing variants are MMAS and ACS, closely followed by AS_{rank}.

C. ACS State Transition Rule

In the ACS, the state transition rule is as follows: an ant positioned on node chooses the city to move to by applying the rule given by Equation (3).

$$s = \begin{cases} \arg_{\max u \in j_k(r)} \left\{ \left[\tau(r,u)\right] \left[\eta(r,u)^{\beta}\right] \right\} & \text{if} \quad q \le q_0 \\ S & \text{otherwise(biasedexploration)} \end{cases} \tag{1}$$

where, q is a random number uniformly distribute in [0,...,1], q_0 is a parameter $0 \le q_0 \le 1$, and S is a random variable selected according to the probability distribution given in Equation (1).

The state transition rule resulting from Equations (3) and (1) is call pseudo-random-proportional rule. This state transition rule, as with the previous random-proportional rule, favors transitions toward nodes connected by short edges and with a large amount of pheromone. The parameter q_0 determines the relative importance of exploitation versus exploration: every time an ant in city r has to choose a city to move to, it samples a random number $0 \le q \le 1$. If $q \le q_0$ then the best edge, according to (3), is chosen (exploitation), otherwise an edge is chosen according to Equation (1) (biased exploration).

D. ACS Global Updating Rule

In ACS, only the globally best ant (i.e., the ant that constructed the shortest tour from the beginning of the trial) is allow to deposit pheromone. This choice, together with the use of the pseudo-random-proportional rule, is intend to make the search more directed: ants search in a neighborhood of best tour found up to the current iteration of the algorithm. Global updating performed after all ants have completed their tours. The pheromone level is update by applying the global updating rule of Equation (2).

$0 < \gamma < 1$

where: (2)

$$\Delta\tau(r,s) = \begin{cases} (\log)^{-1} & \text{if} \quad (r,s) \in \alpha \Delta\tau(r,s) \\ 0 & \text{otherwise} \end{cases}$$

where, $0 < \alpha < 1$ is the pheromone decay parameter, and L_{gb} is the length of the globally best tour from the beginning of the trial. As was the case in ant system, global updating is intend to provide a greater amount of pheromone to shorter tours.

Equation (2) dictates that only those edges belonging to the globally best tour will receive reinforcement. We also tested another type of global updating rule, called iteration-best, as opposed to the above called global-best, which instead used L_{ib} (the length of the best tour in the current iteration of the trial), in Equation (2). In addition, with iteration-best the edges, which receive reinforcement, are those belonging to the best tour of the current iteration. Experiments have shown that the difference between the two schemes is minimal, with a slight preference for global-best, which us is therefore in the following experiments.

E. ACS Local Updating Rule

While building a solution (i.e., a tour) of the TSP, ants visit edges and change their pheromone level by applying the local updating rule of Equation (3).

$$\tau(r,s) \leftarrow (1-\rho)\tau(r,s) + \rho\Delta\tau(r,s) \tag{3}$$

where, $0 < r < 1$ is a parameter.

We have experimented with three values for the term $\Delta\tau(r,s)$. The first choice was loosely inspire by Q-learning an algorithm developed to solve reinforcement learning problems [26]. Such problems are face by an agent that must learn the best action to perform in each possible state in which it finds itself, using as the sole learning information a scalar number, which represents an evaluation of the state, entered after it has performed the chosen action.

Q-learning is an algorithm, which allows an agent to learn such an optimal policy by the recursive application of a rule similar to that in Equation (3), in which the term $\Delta\tau(r,s)$ is set to the discounted evaluation of the next state value. Since the problem our ants have to solve is similar to a reinforcement-learning problem (ants have to learn which city to move to as a function of their current location), we set [19] $\Delta\tau(r,s) = \gamma_{\max z \in J_k(s)}\tau(s,z)$, which is exactly the same formula used in Q-learning ($0 < \gamma < 1$ is a parameter).

The other two choices were: 1- we set $\Delta\tau(r,s) = \tau_0$, where τ_0 the initial pheromone level is, and 2- we set $\Delta\tau(r,s) = 0$. Finally, we also ran experiments in which local updating was not applied (i.e., the local updating rule is not us, as was the case in ant system).

Results obtained running experiments (Table 1) on a set of five randomly generated 50-city TSP's [13], on the Oliver30 symmetric TSP and the ry48p asymmetric TSP [35], essentially suggest that local updating is definitely useful and that the local updating rule with $\Delta\tau(r,s) = 0$ yields worse performance than local updating with $\Delta\tau(r,s) = \gamma_{\max z \in J_k(s)}\tau(s,z)$ or with $\Delta\tau(r,s) = \tau_0$. The ACS with:

$$\Delta\tau(r,s) = \gamma_{\max z \in J_k(s)}\tau(s,z) \tag{4}$$

Which we have called Ant-Q in [11] and [19], and the ACS with called simply ACS hereafter, resulted to be the two best performing algorithms, with a similar performance level. Since the ACS local updating rule requires less computation than Ant-Q, we chose to focus attention on the ACS, which will be us to run the experiments presented in the rest of this paper.

As will be discussed in Section IV-A, the role of the ACS local updating rule is to shuffle the tours, so that the early cities in one ant's tour may be, explore later in other ants' tours. In other words, the effect of local updating is to make the desirability of edges change dynamically: every time an ant uses an edge, this becomes slightly less desirable (since it loses some of its pheromone). In this way, ants will make a better use of pheromone information: without local updating all ants would search in a narrow neighborhood of the best previous tour

III. ANT COLONY OPTIMIZATION

In ACO algorithms, a colony of artificial ants iteratively constructs solutions for the problem under consideration using artificial pheromone trails and heuristic information. The pheromone trails are modifying by ants during the algorithm execution in order to store information about 'good' solutions. Most ACO alga rhythms follow the algorithmic scheme given in Figure 3.

```
procedure ACO metaheuristic for combinatorial optimization problems
    Set parameters, initialize pheromone trails
    while (termination condition not met)
        ConstructSolutions
        (ApplyLocalSearch)
        UpdateTrails
    end while
```

Figure 3. High-level pseudo code for the ACO met heuristic

ACO are solution construction algorithms, which, in contrast to local search algorithms, may not find a locally optimal solution. Many of the best performing ACO algorithms improve their solutions by applying a local search algorithm after the solution construction phase. Our primary goal in this work is to analyze the PTSP tour construction capabilities of ACO, hence in this first investigation we do not use local search.

We apply to the PTSP Ant Colony System (ACS) [9, 10], a particular ACO algorithm which was successfully applied to the TSP. We also consider a medication of ACS which explicitly takes into account the PTSP objective function (we call this algorithm probabilistic ACS, that is, P^{ACS}). In the following, we describe how ACS and P^{ACS} build a solution and how they update pheromone trails.

A. Solution Construction in ACS and P^{ACS}

A feasible solution for an n-city PTSP is an a priori tour, which visits all customers. Initially m ants are position on their starting cities chosen according to some initialization rule (e.g., randomly). Then, the solution construction phase starts (procedure Construct Solutions in Figure 2). Each ant progressively builds a tour by choosing the next customer to move to because of two types of in- formation, the pheromone τ and the heuristic information η. To each arc, joining two customers i, j it is associated a varying quantity of pheromone T_{ij}, and the heuristic value $\eta_{ij} = 1/d_{ij}$, which is the inverse of the distance between j and i.

When an ant k is on city i, the next city is chosen as follows. {With probability q_0 city j, that maximizes $\tau_{ij}\eta_{ij}^{\beta}$ is chosen in set $J_k(i)$ of the cities not yet visited by ant k.

Here, β is a parameter, which determines the relative influence of the heuristic information. {With probability 1-q_0, a city j is chosen randomly with a probability given by:

$$p_k(i,j) = \begin{cases} \dfrac{\tau_{ij}\eta_{ij}^{\beta}}{\sum \tau_{ij}\eta_{ij}^{\beta}} & \text{if} \quad j \in j_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Hence, with probability q_0 the ant chooses the best city according to the pheromone trail and to the distance between cities, while with probability 1-q_0 it explores the search space in a biased way.

B. Pheromone Trails Update in ACS and P^{ACS}

Pheromone trails are update in two stages. In the first stage, each ant, after it has chosen the next city to move to, applies following local update rule $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\tau_0$, where ρ, $0 < \rho \le 1$, and τ_0, are two parameters. The effect of the local updating rule is to make less desirable an arc, which has already been chosen by an ant, so that the exploration of different tours is favored during one iteration of the algorithm.

The second stage of pheromone update occurs when all ants have terminated their tour. Pheromone is modified on those arcs belonging to the best tour since the beginning of the trial (best-so-far tour), by the following global updating rule.

$$\tau_{ij} \leftarrow (1-\alpha)\tau_{ij} + \alpha\Delta\tau_{ij} \tag{6}$$

$$\Delta\tau_{ij} = ObjectiveFunc_{best}^{-1} \tag{7}$$

where, $0 < \alpha \le 1$ being, the pheromone decay parameter, and $ObjectiveFunc_{best}$ is the value of the objective function of the best-so-far tour. In ACS, the objective function is the a priori tour length, while in P^{ACS} the objective function is the PTSP expected length of the a priori tour. In the next section, we discuss in more detail the differences between ACS and P^{ACS}.

C. Discussion of Differences between ACS and P^{ACS}

Differences between ACS and P^{ACS} are because the two algorithms exploit different objective functions in the pheromone-updating phase. The first and most important difference is set of arcs on which pheromone is globally increased, which is in general different in ACS and P^{ACS}, in fact, the 'best tour' in eq. Therefore, in ACS the search will be bias toward the shortest tour, while in P^{ACS} it will be bias toward the tour of minimum expected length.

The second difference between ACS and P^{ACS} is in the quantity $\Delta\tau_{ij}$ by which pheromone is increase on the selected arcs. This aspect is less important than the first, because ACO in general is more sensitive to the difference of pheromone among arcs than to its absolute value. The length of an a priori tour (ACS objective function) may be considered as an $o(n)$ approximation to the $o(n)^2$ expect length (P^{ACS} objective function). In general, the worse the approximation, the worse will be the solution quality of ACS versus P^{ACS}. The quality of the approximation depends on the set of customer probabilities p_i.

In the homogeneous PTSP, where customer probability is p for all customers, it is easy to see the relation between the two objective functions. For a given a priori tour L_λ :

$$\Delta = L_\lambda - E[L_\lambda] = (1-P^2)L_\lambda - \sum_{r=1}^{n-2}(1-p)^r L_\lambda^{(r)} \tag{8}$$

$$\Delta \sim o(qL_\lambda) \tag{9}$$

for $(1-p) = q \to 0$.

Therefore the higher the probability, the better is the a priori tour length L_λ as an approximation for the expected tour length $E[L_\lambda]$. In the heterogeneous PTSP, it is not easy to see the relation between the two objective functions, since each arc of the a priori tour L_λ is multiplied by a different probabilistic weight (Equation (3)), and a term with L_λ cannot be isolated in the expression of $E[L_\lambda]$, as in homogeneous case. ACS and P^{ACS} differ in time complexity.

In both algorithms one iteration (i.e., one cycle through the while condition of Figure 2) is $o(n)^2$ [11], but the constant of proportionality is bigger in P^{ACS} than in ACS. To see this one should consider the procedure Update Trail of Figure 2, where the best-so-far tour must be evaluate in order to choose the arcs on which pheromone is to be updated. The evaluation of the best-so-far tour requires $o(n)$ time in ACS and $o(n)^2$ time in P^{ACS}. ACS is thus faster and always performs more iterations than P^{ACS} for a fixed CPU time.

IV. IMPLEMENTING OF ACO ALGORITHMS

This section describes in detail the steps that have to be, taken to implement an ACO algorithm for the TSP. Because the basic considerations for the implementation of different ACO algorithm variants are very similar, we mainly focus on AS and indicate, where appropriate, the necessary changes for implementing other ACO algorithms. A first implementation of an ACO algorithm can be quite straightforward. In fact, if a greedy construction procedure like a nearest-neighbor heuristic is available, one can use as a construction graph the same graph used by the construction procedure.

Then it is only necessary to Equation (1) add pheromone trail variables to the construction graph and Equation (2) define the set of artificial ants to be used for constructing solutions in such a way that they implement, according to Equation (8), a randomized version of the construction procedure. It must be not, however, that in order to have an efficient implementation, often-additional data structures are required, like arrays to store information, which, although redundant, make the processing much faster.

In the following, we describe the steps to be taken to obtain an efficient implementation of AS. We will give a pseudo-code description of a possible implementation in a C-like notation. This description is general enough to allow for implementing an efficient version of any of the ACO algorithms.

A. Data Structures

As a first step, the basic data structures have to be defining. These must allow storing the data about the TSP instance and the pheromone trails, and representing artificial ants. Figure 4 gives a general outline of the main data structures that are we for the implementation of an ACO algorithm, which includes the data for the problem representation and the data for the representation of the ants.

B. Problem Representation

B.1. Intercity Distances

Often a symmetric TSP instance is given as the coordinates of a number of n points. In this case, one possibility would be to store the x and y coordinates of the cities in two arrays and then computes on the fly the distance between the cities as needed. However, this leads to a significant computational overhead: obviously, it is more reasonable to precompile all intercity distances and to store them in a symmetric distance matrix with n^2 entries. In fact, although for symmetric TSPs we only need to store $n(n-1)/2$ distinct distances, it is more efficient to use a n^2 matrix to avoid performing additional operations to check whether, when accessing a generic distance $d(i, j)$, entry(i, j) or entry (j, i) of the matrix should use.

```
% Representation of problem data

integer dist[n][n]          % distance matrix
integer nn_list[n][nn]      % matrix with nearest neighbor lists of depth nn
real    pheromone[n][n]     % pheromone matrix
real    choice_info[n][n]   % combined pheromone and heuristic information

% Representation of ants

structure single_ant
begin
   integer tour_length   % the ant's tour length
   integer tour[n + 1]   % ant's memory storing (partial) tours
   integer visited[n]    % visited cities
end

single_ant ant[m]        % structure of type single_ant
```

Figure 4. Main data structures for the implementation of an ACO algorithm for the TSP

Note that for very large instances it may be necessary to compute distances on the fly, if it is not possible (or too expensive) to keep the full distance matrix in the main memory. Fortunately, in these cases, there exist some intermediate possibilities, such as storing the distances between a city and the cities of its nearest-neighbor list that greatly reduce the necessary amount of computation. It is also important to know that, for historical reasons, in almost all the TSP literature, the distances are stored as integers. In fact, in old computers integer operations used to be much faster than operations on real numbers, so that by setting distances to be integers, much more code that is efficient could be obtain.

B.2. Nearest-Neighbor Lists

In addition to the distance matrix, it is convenient to store for each city a list of its nearest neighbors. Let d_i be the list of the distances from a city i to all cities j, with j, $j = 1,\ldots, n$ and $i \neq j$ (we assume here that the value d_{ij} is assigned a value larger than $d_{\max}$, where $d_{\max}$ is the maximum distance between any two cities).

The nearest neighbor list of a city i is obtained by sorting the list d_i according to no decreasing distances, obtaining a sorted list d_i ties can be broken randomly. The position r of a city j in city i is nearest-neighbor list *nn-list*[i] is the index of the distance d_{ij} in the sorted list d_i that is, *nn-list*[i] gives the identifier (index) of the r-the nearest city to city $i\,(i.e., nn\_list[i][r] = j)$. Nearest neighbor, lists for all cities can be constructing in $o\left(n^2 \log n\right)$ (in fact, you have to repeat a sorting algorithm over n-1 cities for each city).

An enormous speedup is obtain for the solution construction in ACO algorithms, if the nearest-neighbor list is cut off after a constant number *nn* of nearest neighbors, where typically *nn* is a small value ranging between 15 and 40. In this case, an ant located in city i chooses the next city among the *nn* nearest neighbors of i; in case the ant has already visited all the nearest neighbors, then it makes its selection among the remaining cities. This reduces the complexity of making the choice of the next city to $o(1)$ unless the ant has already visited all the cities in *nn-list*[i]. However, it should be not that the use of truncated nearest-neighbor lists could make it impossible to find the optimal solution.

V. PHEROMONE TRAILS

In addition to the instance-related information, we also have to store for each connection (i, j) a number T_{ij} corresponding to the pheromone trail associated with that connection. In fact, for symmetric TSP_s this requires storing $n(n-1)/2$ distinct pheromone values, because we assume that $T_{ij} = T_{ji}$. Again, as was the case for the distance matrix, it is more convenient to use some redundancy and to store the pheromones in a symmetric n^2 matrix.

A. Combining Pheromone and Heuristic Information

When constructing a tour, an ant located on city i chooses the next city j with a probability which is proportional to the value of $\left[T_{ij}\right]^{\alpha}\left[n_{ij}\right]^{\beta}$. Because these very same values need to be compute by each of the m ants, computation times may be significantly reduce by using an additional matrix choice info, where each entry choice info [i][j] stores the value $\left[T_{ij}\right]^{\alpha}\left[n_{ij}\right]^{\beta}$.

Again, in the case of a symmetric TSP instance, only $n(n-1)/2$ values have to be compute, but it is convenient to store these values in a redundant way as in the case of the pheromone and the distance matrices. Additionally, one may store the η_{ij}^{β} values in a further matrix heuristic (not implemented in the code associated with the book) to avoid recompiling these values after each iteration. Because the heuristic information stays the same throughout the whole run of the algorithm (some tests have shown that the speedup obtained when no local search is used is approximately 10%, while no significant differences are observe when local search is used). Finally, if some distances are zero, which is in fact the case for some of the benchmark instances in TSPLIB, then one may set them to a very small positive value to avoid division by zero.

VI. THE ALGORITHM

The main tasks to be considering in an ACO algorithm are the solution construction, the management of the pheromone trails, and the additional techniques such as local search. In addition, the data structures and parameters need to be initialize and some statistics about the run need to be maintain. In Figure 5, we give a high-level view of the algorithm, while in the following we give some details on how to implement the different procedures of AS in an efficient way.

A. Data Initialization

In the data initialization, 1- The instance has to be read, 2- The distance matrix has to be computed, 3- The nearest neighbor lists for all cities have to be computed, 4- The pheromone matrix and the choice info matrix have to be initialized, 5- The ants have to be initialized. 6- The algorithm's parameters must be initialized, and 7- Some variables that keep track of statistical information, such as the used CPU time, the number of iterations, or the best solution found so far, have to be initialized. A possible organization of these tasks into several data initialization procedures is indicating in Figure 6.

```
procedure ACOforTSP
  InitializeData
  while (not terminate) do
    ConstructSolutions
    LocalSearch
    UpdateStatistics
    UpdatePheromoneTrails
  end-while
end-procedure
```

Figure 5. High-level view of an ACO algorithm for the TSP

B. Termination Condition

The program stops if at least one termination condition applies. Possible termination conditions are: 1- The algorithm has found a solution within a predefined distance from a lower bound on the optimal solution quality, 2- A maximum number of tour constructions or a maximum number of algorithm iterations has been reach; 3- a maximum CPU time has been spent, or 4- The algorithm shows stagnation behavior.

C. Solution Construction

The tour construction is managed the procedure Construct Solutions, shown in Figure 7. The solution construction requires the following phases.

1- First, the ants' memory must be empty. This done in lines 1 to 5 of procedure Construct Solutions by marking all cities as unvisited, that is, by setting all the entries of the array ants. Visit to false for all the ants.

2- Second, each ant has to be assigning an initial city. One possibility is to assign each ant a random initial city. This is accomplishing in lines 6 to 11 of the procedure. The function random returns a random number chose according to a uniform distribution over the set $\{1,...,n\}$.

3- Next, each ant constructs a complete tour. At each construction step (see the procedure in figure 7) the ants apply the AS action choice rule [Equation (9)]. The procedure ASD excision Rule implements the action choice rule and takes as parameters the ant identifier and the current construction step index; this is discuss below in more detail.

4- Finally, in lines 18 to 21, the ants move back to the initial city and the tour length of each ant's tour is compute. Remember that, for the sake of simplicity, in the tour representation we repeat the identifier of the first city at position n this is done in line 19.

```
procedure InitializeData
  ReadInstance
  ComputeDistances
  ComputeNearestNeighborLists
  ComputeChoiceInformation
  InitializeAnts
  InitializeParameters
  InitializeStatistics
end-procedure
```

Figure 6. Procedure to initialize the algorithm

```
     procedure ConstructSolutions
1      for k = 1 to m do
2        for i = 1 to n do
3          ant[k].visited[i] ← false
4        end-for
5      end-for
6      step ← 1
7      for k = 1 to m do
8        r ← random{1, . . . , n}
9        ant[k].tour[step] ← r
10       ant[k].visited[r] ← true
11     end-for
12     while (step < n) do
13       step ← step + 1
14       for k = 1 to m do
15         ASDecisionRule(k, step)
16       end-for
17     end-while
18     for k = 1 to m do
19       ant[k].tour[n + 1] ← ant[k].tour[1]
20       ant[k].tour_length ← ComputeTourLength(k)
21     end-for
     end-procedure
```

Figure 7. Pseudo-code of the solution construction procedure for AS and its variants

As stated above, the solution construction of all of the ants is synchronizing in such a way that the ants build solutions in parallel. The same behavior can be obtain, for all AS variants, by ants that construct solutions sequentially, because the ants do not change the pheromone trails at construction time (this is not the case for ACS, in which case the sequential and parallel implementations give different results). While phases (1), (2), and (4) are very straightforward to code, the implementation of the action choice rule requires some care to avoid large computation times.

In the action choice rule an ant located at city i probabilistically chooses to move to an unvisited city j based on the pheromone trails T_{ij}^{α} and the heuristic information η_{ij}^{β}.

Here we give pseudo-codes for the action choice rule with and without consideration of candidate lists. The pseudo-code for the first variant ASD excision Rule is given in Figure 8. The procedure works as follows: first, the current city *c* of ant *k* is determined (Line 1). The probabilistic choice of the next city then works analogously to the roulette wheel selection procedure of evolutionary computation (Goldberg, 1989). Each value choice info [*c*][*j*] of a city *j* that ant *k* has not visited yet determines a slice on a circular roulette wheel, the size of the slice being proportional to the weight of the associated choice (lines 2-10). Next, the wheel is spun and the city to which the marker points is chosen as the next city for ant *k* (lines 11-17). This is implement by

1- Summing the weight of the various choices in the variable sum probabilities,

2- Drawing a uniformly distributed random number from the interval ½0; sum_ probabilities,

3- Going through the feasible choices until the sum is greater or equal to r.

Finally, the ant is move to the chosen city, which is marked as visited (lines 18 and 19).

```
procedure ASDecisionRule(k, i)
    input  k   % ant identifier
    input  i   % counter for construction step
1   c ← ant[k].tour[i − 1]
2   sum_probabilities = 0.0
3   for j = 1 to n do
4       if ant[k].visited[j] then
5           selection_probability[j] ← 0.0
6       else
7           selection_probability[j] ← choice_info[c][j]
8           sum_probabilities ← sum_probabilities + selection_probability[j]
9       end-if
10  end-for
11  r ← random[0, sum_probabilities]
12  j ← 1
13  p ← selection_probability[j]
14  while (p < r) do
15      j ← j + 1
16      p ← p + selection_probability[j]
17  end-while
18  ant[k].tour[i] ← j
19  ant[k].visited[j] ← true
end-procedure
```

Figure 8. AS without candidate lists, pseudo-code for the action choice rule

These construction steps repeated until the ants have completed a tour. Since each ant has to visit exactly n cities, all the ants complete the solution construction after the same number of construction steps. When exploiting candidate lists, the procedure ASD excision Rule needs to be adapted, resulting in the procedure Neighbor List ASD excision Rule, given in Figure 9. A first change is that when choosing next city, one needs to identify the appropriate city index from the candidate list of the current city *c*.

These results in changes of lines 3 to 10 of Figure 8: the maximum value of index j is change from *n* to *nn* in line 3 and the test performed in line 4 is apply to the *j*-the nearest neighbor given by $nn\_list[c][j]$. A second change is necessary to deal with the situation in which all the cities in the candidate list have already been visit by ant *k*. In this case, the variable sum probabilities keeps its initial value 0.0 and one city out of those not in the candidate list is chosen: the procedure Choose Best Next is used to identify the city with maximum value of $\left[T_{ij}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}$ as the next to move to.

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. Yet, the computation time is reduced only if procedure Choose Best Next does not need to be applying too often. Fortunately, as also suggested by computational results presented in Gambardella & Dorigo (1996) that this seems not to be the case.

```
procedure NeighborListASDecisionRule(k, i)
    input  k   % ant identifier
    input  i   % counter for construction step
1   c ← ant[k].tour[i − 1]
2   sum_probabilities ← 0.0
3   for j = 1 to nn do
4       if ant[k].visited[nn_list[c][j]] then
5           selection_probability[j] ← 0.0
6       else
7           selection_probability[j] ← choice_info[c][nn_list[c][j]]
8           sum_probabilities ← sum_probabilities + selection_probability[j]
9       end-if
10  end-for
11  if (sum_probabilities = 0.0) then
12      ChooseBestNext(k, i)
13  else
14      r ← random[0, sum_probabilities]
15      j ← 1
16      p ← selection_probability[j]
17      while (p < r) do
18          j ← j + 1
19          p ← p + selection_probability[j]
20      end-while
21      ant[k].tour[i] ← nn_list[c][j]
22      ant[k].visited[nn_list[c][j]] ← true
23  end-if
end-procedure
```

Figure 9. AS with candidate lists: pseudo-code for the action choice rule

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the potentialities of ACO algorithms for the PTSP. In particular, we have shown that the P^{ACS} algorithm is a promising heuristic for homogeneous TSP instances. Moreover, for customer's probabilities close to 1, the ACS heuristic is a better alternative than P^{ACS}. At present, we are investigating the heterogeneous PTSP, for different probability configurations of customers. This is an interesting direction of research, since it is closer to a real-world problem than the homogeneous PTSP.

We are also trying to improve P^{ACS} performance by inserting in the ants' tour con-striation criterion information about the customers probabilities. Work that will follow this paper also comprehends a comparison of P^{ACS} with respect to other PTSP algorithms. Moreover, P^{ACS} should be improving by adding to the tour construction phase a local search algorithm. The best choice and design of such a local search is also an interesting issue for the PTSP.

```
procedure ChooseBestNext(k, i)
  input k   % ant identifier
  input i   % counter for construction step
  v ← 0.0
  c ← ant[k].tour[i − 1]
  for j = 1 to n do
    if not ant[k].visited[ j] then
      if choice_info[c][ j] > v then
        nc ← j        % city with maximal τ^α η^β
        v ← choice_info[c][ j]
      end-if
    end-if
  end-for
  ant[k].tour[i] ← nc
  ant[k].visited[nc] ← true
end-procedure
```

Figure 10. AS pseudo-code for the procedure Choose Best Next

REFERENCES

[1] S. Baluja, R. Caruana, "Removing the Genetics From the Standard Genetic Algorithm", 12th Int. Conf. Machine Learning, Palo Alto, CA: Morgan Kaufmann, pp. 38-46, 1995.

[2] A.G. Barto, R.S. Sutton, P.S. Brower, "Associative Search Network - A Reinforcement Learning Associative Memory", Biological Cybern., Vol. 40, pp. 201-211, 1981.

[3] R. Beckers, J.L. Deneubourg, S. Goss, "Trails and U-turns in the Selection of the Shortest Path by the Ant Lasius Niger", J. Theoretical Biology, Vol. 159, pp. 397-415, 1992.

[4] J.L. Bentley, "Fast Algorithms for Geometric Traveling Salesman Problems", ORSA J. Comput., Vol. 4, pp. 387-411, 1992.

[5] H. Bersini, M. Dorigo, S. Langerman, G. Seront, L.M. Gambardella, "Results of the First International Contest on Evolutionary Optimization (1st ICEO)", IEEE Int. Conf. Evolutionary Computation, IEEEEC 96, pp. 611-615, 1996.

[6] H. Bersini, C. Oury, M. Dorigo, "Hybridization of Genetic Algorithms", Tech. Rep. IRIDIA/95-22, Free University of Brussels, Belgium, 1995.

[7] A. Colorni, M. Dorigo, V. Maniezzo, "Distributed Optimization by Ant Colonies", ECAL91-Eur. Conf. Artificial Life, Elsevier, pp. 134-142, New York, 1991.

[8] M. Dorigo, V. Maniezzo, "An Investigation of Some Properties of an Ant Algorithm", Parallel Problem Solving from Nature Conference (PPSN 92), Elsevier, pp. 509-520, New York, 1992.

[9] J.L. Deneubourg, "By Application of the Order Fluctuations in Description of Certain ETAPS Construction in Nest from Termites", Insect Sociaux, Vol. 24, pp. 117-130, 1977.

[10] M. Dorigo, "Optimization, Learning, and Natural Algorithms", Ph.D. Dissertation, DEI, Polytechnic of Milan, Italy, 1992.

[11] M. Dorigo, L.M. Gambardella, "A Study of Some Properties of ANT-Q", PPSN IV-4th Int. Conf. Parallel Problem Solving from Nature, Springer-Verlag, pp. 656-665, Berlin, Germany, 1996.

[12] M. Dorigo, V. Maniezzo, A. Colorni, "The Ant System - Optimization by a Colony of Cooperating Agents", IEEE Trans. Syst., Man, Cybern., Vol. 26, No. 2, pp. 29-41, 1996.

[13] R. Durbin, D. Willshaw, "An Analogue Approach to the Travelling Salesman Problem Using an Elastic Net Method", Nature, Vol. 326, pp. 689-691, 1987.

[14] S. Eilon, C.D.T. Watson Gandy, N. Christofides, "Distribution Management - Mathematical Modeling and Practical Analysis", Oper. Res. Quart., Vol. 20, pp. 37-53, 1969.

[15] D.B. Fogel, "Applying Evolutionary Programming to Selected Traveling Salesman Problems", Cybern. Syst. Int. J., Vol. 24, pp. 27-36, 1993.

[16] M.L. Fredman, D.S. Johnson, L.A. McGeoch, G. Ostheimer, "Data Structures for Traveling Salesmen", J. Algorithms, Vol. 18, pp. 432-479, 1995.

[17] B. Freisleben, P. Merz, "Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman problems", IEEE Int. Conf. Evolutionary Computation, IEEE-EC 96, pp. 616-621, 1996.

[18] M. Dorigo, "New Genetic Local Search Operators for the Traveling Salesman Problem", PPSN IV-4th Int. Conf. Parallel Problem Solving from Nature, Springer-Verlag, pp. 890-899, Berlin, Germany, 1996.

[19] L.M. Gambardella, M. Dorigo, "Ant-Q - A Reinforcement Learning Approach to the Traveling Salesman Problem", ML-95, 12th Int. Conf. Machine Learning, Palo Alto, CA: Morgan Kaufmann, pp. 252-260, 1995.

[20] M. Dorigo, "Solving Symmetric and Asymmetric TSP's by Ant Colonies", IEEE Int. Conf. Evolutionary Computation, IEEE-EC 96, pp. 622-627, 1996.

[21] B. Golden, W. Stewart, "Empiric Analysis of Heuristics in the Traveling Salesman Problem", Shmoys, Eds., Wiley, New York, 1985.

[22] S. Goss, S. Aron, J.L. Deneubourg, J.M. Pasteels, "Self Organized Shortcuts in the Argentine Ant", Natural Sciences, Vol. 76, pp. 579-581, 1989.

[23] G. Reinelt, "The Traveling Salesman - Computational Solutions for TSP Applications", Springer-Verlag, New York, 1994.

[24] D.S. Johnson, L.A. McGeoch, "The Travelling Salesman Problem - A Case Study in Local Optimization", in Local Search in Combinatorial Optimization, Eds., Wiley, New York, 1997.

[25] L.P. Kaelbling, L.M. Littman, A.W. Moore, "Reinforcement Learning - A Survey", J. Artif. Intell. Res., Vol. 4, pp. 237-285, 1996.

[26] P.C. Kanellakis, C.H. Papadimitriou, "Local Search for the Asymmetric Traveling Salesman Problem", Oper. Res., Vol. 28, No. 5, pp. 1087-1099, 1980.

[27] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy-Kan, D.B. Shmoys, Eds., "The Traveling Salesman Problem", New York, Wiley, 1985.

[28] F.T. Lin, C.Y. Kao, C.C. Hsu, "Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems", IEEE Trans. Syst., Man, Cybern., Vol. 23, pp. 1752-1767, 1993.

[29] S. Lin., "Computer Solutions of Traveling Salesman Problem", Bell Syst. J., Vol. 44, pp. 2245-2269, 1965.

[30] S. Lin, B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem", Oper. Res., Vol. 21, pp. 498-516, 1973.
[31] O. Martin, S.W. Otto, E.W. Felten, "Large Step Markov Chains for the TSP Incorporating Local Search Heuristics", Oper. Res. Lett., Vol. 11, pp. 219-224, 1992.
[32] J.Y. Potvin, "The Traveling Salesman Problem - A Neural Network Perspective", ORSA J. Comput., Vol. 5, No. 4, pp. 328-347, 1993.

BIOGRAPHIES

Naser Mahdavi Tabatabaei was born in Tehran, Iran, 1967. He received the B.Sc. and the M.Sc. degrees from University of Tabriz (Tabriz, Iran) and the Ph.D. degree from Iran University of Science and Technology (Tehran, Iran), all in Power Electrical Engineering, in 1989, 1992, and 1997, respectively. Currently, he is a Professor in International Organization of IOTPE. He is also an academic member of Power Electrical Engineering at Seraj Higher Education Institute (Tabriz, Iran) and teaches power system analysis, power system operation, and reactive power control. He is the General Secretary of International Conference of ICTPE, Editor-in-Chief of International Journal of IJTPE and Chairman of International Enterprise of IETPE all supported by IOTPE. He has authored and co-authored of six books and book chapters in Electrical Engineering area in international publishers and more than 130 papers in international journals and conference proceedings. His research interests are in the area of power quality, energy management systems, ICT in power engineering and virtual e-learning educational systems. He is a member of the Iranian Association of Electrical and Electronic Engineers (IAEEE).

Kiomars Asadian was born in Bijar, Iran in 1987. He received the B.Sc. degree in Electrical Engineering from Shabestar Branch, Islamic Azad University, Shabestar, Iran in 2010 and he is currently a M.Sc. student in Seraj Higher Education Institute, Tabriz, Iran. His research interests include power systems operation and control.

Narges Sadat Boushehri was born in Iran. She received her B.Sc. degree in Control Engineering from Sharif University of Technology (Tehran, Iran), and Electronic Engineering from Central Tehran Branch, Islamic Azad University, (Tehran, Iran), in 1991 and 1996, respectively. She received the M.Sc. degree in Electronic Engineering, 2009. She is the Member of Scientific and Executive Committees of International Conference of ICTPE and also the Scientific and Executive Secretary of International Journal of IJTPE supported by International Organization of IOTPE (www.iotpe.com). Her research interests are in the area of power system control and artificial intelligent algorithms.