Information Sciences 179 (2009) 1454–1468

Contents lists available at ScienceDirect

Information Sciences

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

A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem

Thiago A.S. Masutti [a,*], Leandro N. de Castro [b]

[a] Laboratory of Intelligent Systems (LSIN), Catholic University of Santos (UNISANTOS), R. Dr. Carvalho de Mendonça, 144, Vila Mathias Santos/SP, 11070-906, Brazil
[b] Graduate Program in Electrical Engineering, Mackenzie University, R. da Consolação, 896 São Paulo/SP, 01302-907, Brazil

ARTICLE INFO

Article history:
Received 26 July 2007
Accepted 18 December 2008

Keywords:
Traveling salesman problem
Self-organizing networks
Artificial immune systems
Artificial neural networks
Combinatorial optimization

ABSTRACT

Most combinatorial optimization problems belong to the NP-complete or NP-hard classes, which means that they may require an infeasible processing time to be solved by an exhaustive search method. Thus, less expensive heuristics in respect to the processing time are commonly used. These heuristics can obtain satisfactory solutions in short running times, but there is no guarantee that the optimal solution will be found. Artificial Neural Networks (ANNs) have been widely studied to solve combinatorial problems, presenting encouraging results. This paper proposes some modifications on RABNET-TSP, an immune-inspired self-organizing neural network, for the solution of the Traveling Salesman Problem (TSP). The modified algorithm is compared with other neural methods from the literature and the results obtained suggest that the proposed method is competitive in relation to the other ones, outperforming them in many cases with regards to the quality (cost) of the solutions found, though demanding a greater time for convergence in many cases.

1. Introduction

The study of combinatorial optimization problems is relevant from a theoretical and practical perspective. They are commonly found in real-world situations, such as routing and scheduling, and usually belong to the NP-complete and NP-hard classes [17,25]. Thus, the time required to find a satisfactory solution may be infeasible, and the use of heuristics [18,32] is often preferred, for they are capable of providing a good trade-off between the quality of the solution and the time spent to find it.

Probably the most widely studied combinatorial optimization problem is the *Traveling Salesman Problem* (TSP), which can be described as follows: given a set of n cities (nodes in a graph), a salesman has to visit all the cities once and return to its origin minimizing the cost of the trip [24]. It is well-known that a symmetric TSP with n cities has $(n-1)!/2$ possible routes to perform, a value that grows factorially with the number of cities. Several algorithms have already been devised to solve TSP-like problems, such as *branch and cut* algorithms [9,33] which uses *cutting-plane* and *branch and bound* [4,35] techniques to find the global solution; the k-opt algorithm, which uses a technique that permutes k edges of the current route to find better routes; and *Lin–Kerninghan* [26], which adapts k-opt such that an ideal k is found. For the symmetric TSP, *Lin–Kerninghan* is considered one of the most effective methods to generate optimal or quasi-optimal solutions [20]. Concerning a broader range of heuristics within the Natural Computing [11] area of research, it is important to mention the use of Ant Colony Optimization [13] and Evolutionary Algorithms [28] to solve the TSP. Further information about the TSP and its solution approaches can be found in [2].

* Corresponding author.
E-mail addresses: thiago.masutti@gmail.com (T.A.S. Masutti), lnunes@mackenzie.br (L.N. de Castro).

0020-0255/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2008.12.016

Still under the natural computing umbrella, a class of algorithms that have been broadly applied to combinatorial problems, particularly to the TSP, are the *self-organized artificial neural networks* [6,27,38,39] or slight variations of Kohonen's *self-organizing feature map* (SOM) [23]. The application of these networks to optimization problems in general is important and intriguing. This is because they usually do not involve a quality measure to assess the performance of the solutions they generate, but are still capable of finding high quality solutions that can be used either as the final solution to the problem or that can be combined with another technique to find the global optimal. A self-organized competitive process is the main mechanism used to adjust the network weights and, thus, find a solution to the problem. In the particular case of competitive networks, which will be the focus of this paper, the network architecture is almost invariably of single layer with a circular neighborhood. In SOM-like networks for solving the TSP, the number N of neurons in the network is usually greater than or equal to the number n of cities in the TSP ($N \geqslant n$) and, usually, never greater than $2n$.

The present paper proposes some modifications in the RABNET-TSP [34] algorithm aiming at improving its performance in terms of computational *efficiency* (time to solve the problem) and *optimality* or *efficacy*(quality of the solution). The modified algorithm is directly compared with the original RABNET-TSP, termed here oRABNET-TSP, and two other well-known self-organized algorithms designed to solve TSP-like problems, namely Angeniol's network [1] and Somhom's network [40]. The performance of all methods in terms of optimality is compared with the best values in the literature for a number of TSP instances taken from the TSPLIB [37]. Besides, RABNET-TSP is compared with other approaches whose performances were taken directly from the literature.

This paper is organized as follows. Section 2 reviews the literature on self-organized neural networks applied to the TSP, and Section 3 describes RABNET-TSP stressing the modifications introduced. Section 4 presents the simulation results, including the heuristics used to tune the main parameters of all algorithms, and Section 5 concludes the paper with a discussion about the results presented and some perspectives for future investigation.

2. Self-organized networks applied to the TSP: a brief review

This section makes a brief review of self-organized neural nets to solve TSP problems, focusing on the main differences between the proposals and describing in greater detail the works of Angeniol et al. [1] and Somhom et al. [40], which were re-implemented and used for direct performance comparisons.

In the work of Durbin and Wilshaw [12], the authors proposed the *Elastic Net*, in which the neurons are moved according to two strengths that force the network to be expanded as an elastic band until all cities be covered by the neurons, thus forming a TSP route. These forces act upon the neurons aiming at minimizing the size of the band. The elastic net was used to solve 50-city instances of the traveling salesman problem with results 2% worse than those obtained with the *simulated annealing algorithm* [22], regarding the quality of the solutions.

The work of Angeniol et al. [1] was one of the pioneers in the use of self-organized neural networks to solve TSP-like problems. In their algorithm, the network is initialized with a single neuron and, through a process of duplication and pruning, the network architecture is allowed to vary with time. Based on experimental results the authors observed that the number of neurons is never greater than twice the number of cities in the instance investigated. Before the iterative process starts, the order of the cities is randomized and maintained fixed throughout all epochs. For each city (input pattern), the winner neuron is the one closest to it. All neurons in the network are moved towards the input city based on a function that takes into account the neighborhood between the neurons in a circular ring and the winner neuron:

$$f(G,d) = (1/\mathrm{sqrt}(2))\exp(-d^2/G^2), \tag{1}$$

where d is the neighborhood degree between a given neuron and the winner neuron, and G is a parameter that controls the neighborhood influence. Parameter G usually starts with high values and decreases along the iterative procedure of adaptation.

The updating rule of neuron j is given by:

$$\mathbf{w}_j = \mathbf{w}_j + \alpha f(G,d)(\mathbf{x}_i - \mathbf{w}_j), \tag{2}$$

where $\mathbf{w}_j$ is the weight vector of neuron j, α is a learning rate, and $\mathbf{x}_i$ is the coordinates' vector of city i. Angeniol's algorithm does not explicit use a learning rate, what is equivalent to making $\alpha = 1$ in Eq. (2) above.

After each epoch, all neurons that won the competition for more than one city are duplicated. The new copy is inserted in the network as a direct neighbor of the duplicated neuron and both neurons are inhibited in the following epoch; that is, if they win the competition for a certain city they will produce no updating in the network. This inhibition makes that 'twin' neurons be separated by the movements of their neighbors before they promote changes in the network. For a neuron to be excluded from the network, it must spend three iterations in a row without winning for any city.

Along with the work of Angeniol et al., the paper by Fort [15] was pioneer in applying self-organizing networks to the TSP. Unlike [1], however, Fort uses a fixed number of neurons in the network, being $2n$ neurons for small instances, $n \in [10,30]$, and $2.5n$ for larger instances. One of the main problems found by the author was the high computational time required to find reasonable solutions. The author also stresses the dynamic nature of the algorithm; that is, its capability of dealing with variations in the TSP instance.

Kim et al. [21] proposed a SOM-based network for TSP in which the number of neurons is always the same as the number of cities. A restriction was imposed in the competitive phase such that a neuron wins for a single city each epoch. The author compares his proposal with [1] and concludes that his algorithm is superior in terms of efficiency and optimality.

In the work of Favata and Walker [14], a modified SOM with a number of neurons equal to the number of cities in the TSP instance to be solved is proposed. In the updating phase, only the winner and its two direct neighbors are updated. No constraint in relation to the competition phase is used and, thus, one neuron can be related to more than one city. According to the author, this makes that the network does not become sensitive to the order of the cities in the final solution and, in order to make this solution feasible, a heuristic is used to sort the cities. Their proposal was compared with a simulated annealing algorithm [22] and concluded that, although their approach presented lower quality solutions, the processing time was smaller.

Burke and Damani [7] proposed the *Guilty Net*, in which the number of neurons used is equal to the number of cities in the TSP. In order to guarantee a feasible solution (one neuron mapping each city), the authors used a bias to avoid that a neuron wins many competitions. This bias is called *conscience*. In a later work [5], Burke proposed a new network, called *Vigilant Net*, to improve the separation among neurons, so that each neuron maps a single city in the TSP. To reach that goal, a vigilance parameter is used to avoid that a single neuron wins too many competitions.

In the work of Somhom et al. [40], a SOM-like neural network is proposed with a number of neurons fixed as twice the number of cities in the problem. To guarantee the convergence of the algorithm to feasible solutions, no neuron is allowed to win twice within the same epoch. The updating rule follows that of Eq. (2), with a neighborhood function given by:

$$f(G,d) = \begin{cases} \exp(-d^2/G^2), & d < 0.2n, \\ 0 & \text{otherwise}, \end{cases} \tag{3}$$

where n is the number of cities in the TSP instance, d is the neighborhood degree between a given neuron and the winning neuron, and G is the parameter that controls the neighborhood influence. Based on this neighborhood function, only 40% of the neurons are updated and, according to the authors, this restriction does not affect the quality of the solution, but improves the efficiency of the algorithm reducing the processing time.

To assess the performance of their algorithm, the authors applied it to several instances of the TSPLIB [37] and compared its results with those obtained by the *Guilty Net*, the *Elastic Net* and the approach introduced by Matsuyama [30]. The results presented in their paper showed a superior performance in terms of efficiency and optimality for most instances tested.

In the work of Aras et al. [3] the authors propose a neural network in which the number of neurons varies according to duplication and pruning procedures similar to that of [1]. An important feature of this paper is that the authors not only take into account the topological distribution of the cities in the TSP instance, they also account for the average position of the cities. The training algorithm has two adaptive modules: (1) an *attraction module* in which the winner neuron and its neighbors belonging to a certain ring move in the direction of the city presented; and (2) a *dispersion module* in which the neurons that were not updated in the attraction module are moved so that the neurons have the same average position as the cities. The authors assessed their method using the TSPLIB and compared the performance with the Guilty Net, the Angeniol's net and with a standard self-organizing map.

More recently, Cochrane and Beasley [8] proposed a network named *Co-Adaptive Net*, characterized by the use of a heuristic that decides if the winner neuron and its neighbors will be adapted or not. The algorithm also has a heuristics that determines one route per epoch. Thus, the best route found by the algorithm is not necessarily the one at the end of the iterative procedure of adaptation, but the best one found so far. The algorithm allows one neuron to map more than one city and, in this case, a heuristics has to be used to determine the proposed route. The performance of their algorithm is compared to that of [3,16,31,40] with very good results.

The work of Reilly and Tchimev [36] compares the Continuous Hopfield neural network [42] with SOM. To compare the performance between the algorithms, the authors proposed a 52-city instance based on a map of Germany. As the Hopfield network was unable to process the full 52 cities instance, a 15 cities subset was used. With the obtained results the authors concluded that the SOM algorithm is clearly superior to the Hopfield algorithm, with a 62% better solution than the Hopfield network. Moreover, the SOM was able to solve the original 52 cities instance, achieving a good solution.

3. Modified RABNET-TSP

The constructive self-organizing network to be presented here is a slight variation of the Real-Valued Antibody Network designed to solve the Traveling Salesman Problem (RABNET-TSP) [34]. The modifications proposed in the original algorithm aim at improving its efficacy (quality of the solutions found) and reducing the computational time of the algorithm. From now on, the acronym RABNET-TSP refers to the modified version to be presented here and oRABNET-TSP refers to its original version.

The main features of RABNET-TSP are: (1) *feedforward* neural network with no hidden layer; (2) competitive and unsupervised learning based on some immune principles [10]; (3) constructive network architecture with growing and pruning phases based on some immune principles; and (4) pre-defined circular neighborhood.

The goal of RABNET-TSP is to position one network cell on top of each city of the TSP instance to be solved. The sequence of labeled network cells will correspond to the sequence of cities to be traversed by the salesman. The adaptation algorithm can be divided into nine distinct phases: (1) network initialization; (2) presentation of input patterns; (3) competition; (4) cooperation; (5) adaptation; (6) growing; (7) stabilization of the winners; (8) network convergence; and (9) pruning. Each of these phases will be explained separately in the next sections and the proposed modifications will be stressed.

3.1. Network initialization

In RABNET-TSP the network is initialized with only one antibod (in artificial neural networks, the analogue for an antibody is the neuron). The attribute vector (weight vector in the neural network literature) describing this antibody is equal to the centroid of the spatial distribution of the cities.

3.2. Presentation of antigens (cities)

During the immune system evolution, an organism can meet a certain antigen several times. As the problem to be solved by the self-organizing network is the TSP, each city corresponds to one antigen (in artificial neural networks, the analogue for an antigen is an input pattern) and they are iteratively presented to the antibody network, simulating the meeting between the organism and an antigen. Prior to each epoch, the order of the cities is randomized so as to avoid that this order influences the network adaptation.

3.3. Competition

The competitive phase consists of finding the network antibody that is most similar to the antigen presented. Thus, that antibody J whose Euclidean distance to the antigen is minimal wins the competition and is selected:

$$J = \arg\min_{j} \|\mathbf{Ag}_i - \mathbf{Ab}_j\| \quad \forall j. \tag{4}$$

where the antigen $\mathbf{Ag}_i$ corresponds to the coordinate vector of city i and $\mathbf{Ab}_j$ is the coordinates vector of antibody j. Each antibody in the network is allowed to recognize zero, one, or more antigens. This information is used in the growing and pruning procedures of the algorithm, and is stored in two variables: (1) one, γ_j, containing the number of cities mapped by antibody j; and (2) another, υ_i, containing the index of the antibody that won the competition for city i. The main idea we use here from artificial immune systems [10] is that the most stimulated cell; that is, the one that wins the competition, will be the one with the highest affinity (most similar) to the city presented and that recognizes the largest number of cities (largest concentration of *antigens*).

3.4. Cooperation

The cooperative procedure used in RABNET-TSP is the same as the one for Kohonen's self-organizing maps [23]. The antibodies in RABNET-TSP are arranged in a linear array of cells with a one-cell ring neighborhood, as illustrated in Fig. 1 and expressed by Eq. (5).

$$d_{jJ} = \min[|j - J|, N - |j - J|], \tag{5}$$

where d_{jJ} is the neighborhood degree between the winner antibody J and the present antibody j, and N is the number of antibodies in the ring.

In Fig. 1, cell 1 is a one-degree neighbor of cells 2 and 5, and a two-degree neighbor of cells 3 and 4.

When an antibody J wins the competition, it is moved towards the antigen, and its neighbors are moved as well, but with a smaller step. The farther the neighbor, the smaller the movement. One form of simulating the influence of a cell with its neighbors is to define a general kernel function h_{iJ} that allows a decay of the cells' updating step smoothly with lateral distance [19,23]. This can be implemented by giving function h_{iJ} a 'bell curve' shape. More specifically, let h_{iJ} denote the topological neighborhood centered on the winning cell J and encompassing a set of neighbor cells, and let d_{iJ} denote the topological lateral distance between the winning antibody J and its neighbor j:

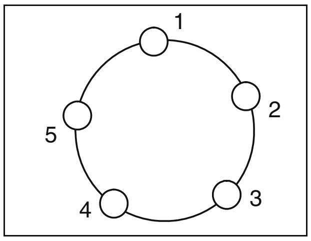


Fig. 1. Ring structure and neighborhood in RABNET-TSP.

$$f(\sigma, d_{jJ}) = h_{jJ} = \exp(-d_{jJ}^2/2\sigma^2), \tag{6}$$

where $f(\sigma, d_{jJ})$ plays the same role as $f(G,d)$ in Eq. (2), and the value of $\sigma = \sigma(t)$ will be responsible for controlling the neighborhood influence during the iterations, as will be explained in details shortly. The parameter $\sigma(t)$ is iteratively reduced until the network weights stabilize, as follows:

$$\sigma(t) = \sigma(0)\exp(-t/\tau_1), \tag{7}$$

where $\sigma(0)$ is the initial value of σ, t is the epoch counter, and τ_1 is defined as $\tau_1 = 1000/\log(\sigma(0))$.

3.5. Adaptation

To present an effective response to pathogenic agents, the immune system counts with a learning process that involves the increasing of the Ag–Ab affinity at each meeting of the organism with a determined antigen. In RABNET-TSP, the adaptation phase performs this increase in Ab–Ag affinity updating the attribute vectors $\mathbf{Ab}_i$ by moving them in the direction of the antigen presented $\mathbf{Ab}_i$, taking into account the neighborhood relationship defined in the cooperative phase. The adaptation rule to be proposed here takes into account the size of the topological neighborhood given by Eq. (6), as follows:

$$\mathbf{Ab}_j(t+1) = \begin{cases} \mathbf{Ab}_j(t) + \alpha.h_{jJ}(\mathbf{Ag}_i - \mathbf{Ab}_j(t)), & h_{jJ} > \kappa, \\ \mathbf{Ab}_j(t) & \text{otherwise}, \end{cases} \tag{8}$$

where $\mathbf{Ag}_i$ corresponds to the coordinates' vector of city i, α is the learning rate, and κ is a parameter that defines the minimum value of h_{jJ} so that $\mathbf{Ab}_j$ is updated.

This is one of the modifications we performed in relation to the original oRABNET-TSP. By constraining the updating to only those cases in which updating will be significant (achieved by setting κ), the computational time of the algorithm can be substantially reduced, as will be empirically investigated in Section 4.1.

It is also common in self-organizing networks that the learning rate decreases with the iterations. Here we follow the same schedule as described in [19]:

$$\alpha(t) = \alpha(0)\exp(-t/\tau_2), \tag{9}$$

where $\alpha(0)$ will depend on the general feature of the TSP instance to be solved and $\tau_2 = 1000$.

3.6. Network growing

The growing process is inspired by the clonal selection and affinity maturation principles from immunology [10]. Basically, the most stimulated antibody in the immune system is selected for cloning (splitting). The stimulation level of an antibody is determined by two parameters: (1) the affinity level of antibody j in relation to the antigen presented j, which will be measured here by their Euclidean distance; and (2) the concentration (number), γ_j, of antigens recognized by antibody j.

The possibility of growing is tested every iteration, as follows:

- Antibody I that recognizes the highest concentration of antigens is selected: $I = \arg\max_i(\gamma_j)\ \ \forall j$. If two antibodies have the same highest concentration γ, then one of them is selected randomly.
- Among all antigens (cities) recognized by antibody I, $\mathbf{Ag}_i$ with the greatest Euclidean distance to $\mathbf{Ab}_I$ is selected.
- If the distance between the selected antibody and $\mathbf{Ag}_i$ is greater than a pre-defined threshold ε, then $\mathbf{Ab}_I$ is cloned:
  If $\|\mathbf{Ag}_i\text{-}\mathbf{Ab}_I\| > \varepsilon$, then clone.

The vector describing the features of the newly created antibody is the same as the one from its parent antibody, and its neighborhood is of degree one in relation to it. The whole network neighborhood is then reconstructed.

3.7. Stabilization of the winners

To reduce the processing time of the algorithm, we propose the Winners Stabilization phase which suppresses cooperation as soon as no variation, Δv, in the winners is detected. This is obtained when the winners for each antigen in a given epoch are the same as the ones in the previous epoch:

$$\Delta v = \sum_{i=1}^{n} |J_i(t-1) - J_i(t)|, \tag{10}$$

where $J_i(t-1)$ and $J_i(t)$ are the indices of the winners for city i at epoch $t-1$ and t, respectively; and n is the number of cities.

To quantify the gain in performance with the addition of this phase, for a TSP instance with 76 cities, the average computational time was 5.5% smaller than with the original algorithm (oRABNET-TSP, 9.987s; RABNET-TSP, 9.462s).

3.8. Convergence criterion

In order for the algorithm to converge to a valid solution, a well-defined route has to be determined by the network antibodies. Therefore, at least one antibody has to map each city. To guarantee that this is going to occur, every antibody will have to be at least at a distance λ from a city. When this condition is met, the learning algorithm automatically stops.

3.9. Network pruning

Pruning in RABNET-TSP is very simple. After the iterative procedure of adaptation is concluded, all antibodies with concentration level $\gamma_j = 0\ \forall j$, are removed from the network. The expected result is an antibody mapping each city in the instance. This way, the pre-defined neighborhood will indicate the TSP route determined.

Algorithm 1 summarizes the pseudo-code for the modified RABNET-TSP.

4. Performance evaluation

To assess the performance of RABNET-TSP, several experiments were conducted using TSP instances available at TSPLIB [37]. Another instance with 64 cities symmetrically distributed, ncit64, proposed in [34] was also employed. To compare the performance of the modified RABNET-TSP, the following algorithms were implemented: (1) the original algorithm oRABNET-TSP [34]; (2) Angeniol's algorithm [1], named here AVT; (3) Somhom's algorithm [40], named here SME. The reason why we chose Angeniol's algorithm for comparison is because it is one of the pioneers in the literature and it also incorporates a growing mechanism in the network. The choice of Somhom's goes in the opposite direction, it constitutes a more recent approach with a fixed number of neurons in the network and with good results reported.

The algorithms were implemented in Matlab and run on a PIV 3.0 GHz with 1 GB RAM. The results presented are the minimum, average and standard deviation of the cost (tour length) over 30 runs. The average number of epochs, time for convergence and number of cells generated are also listed. Furthermore, the performance of other algorithms from the literature is considered when available.

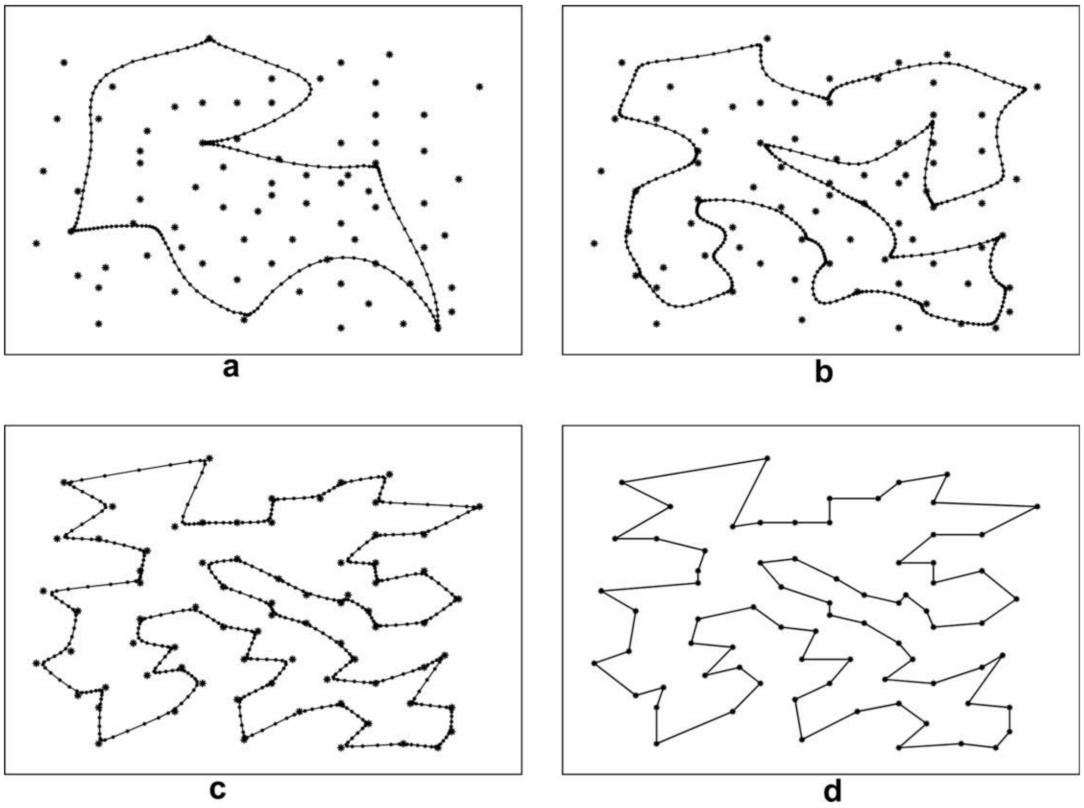


Fig. 2. RABNET-TSP learning for the instance eil76. (a) 200 epochs; (b) 400 epochs; (c) 600 epochs; (d) final solution after 837 epochs.

4.1. The influence of tunable parameters

All the algorithms investigated have some parameters to be adjusted before they can be applied to the problems at hand. This section briefly investigates the influence of the tunable parameters of all algorithms used for comparisons, with emphasis on the proposed RABNET-TSP, namely, $\alpha(0)$, $\sigma(0)$, ε, λ and κ. The analyses were undertaken using the eil76 TSP instance [37], and all results presented involve 30 independent runs.

RABNET-TSP is initialized with a single antibody and is allowed do grow during training. Fig. 2 illustrates the behavior of the network during the iterative procedure of adaptation. Note that the network behaves like a growing ring that molds itself to cover all nodes in the graph.

When an antibody wins for more than one city, it will be cloned when its distance from the farthest city is greater than or equal to ε, thus avoiding that an antibody is cloned unnecessarily. According to [34], a reasonable value for ε is:

$$\varepsilon = 0.2^{*}md, \tag{11}$$

where md is the smallest distance among all the cities in the map. Furthermore, in order for a route to emerge, each antibody must be at a minimum distance from each city. Parameter λ is used to guarantee this approximation and its value is also a function of md, as follows:

$$\lambda = 0.01^{*}md. \tag{12}$$

Parameter σ is responsible for controlling the influence of the neighborhood in the adaptation of the antibodies. Its initial value, $\sigma(0)$, directly influences the total number of antibodies used in the network and its convergence time, as can be observed from Fig. 3. As illustrated in Fig. 3d, one can note that, on average, the initial value of σ does not influences significantly the quality of the solution found, but its right tune had some crucial effect on founding the best solutions that will be presented in this paper. An appropriate value for $\sigma(0)$ varies according to the TSP instance to be tackled and should be obtained with some preliminary experiments. When this is not possible we suggest the use of $\sigma(0) = 16$, which performed well for most of our experiments.

Large values for parameter $\alpha(0)$ imply in major changes in the network structure at each epoch, and vice-versa. As can be observed from Fig. 4, large values for $\alpha(0)$ result in large numbers of antibodies in the network and large numbers of epochs for convergence, thus requiring large convergence time. Except for low values of $\alpha(0)$ (e.g., 0.1 or 0.3), the quality of the average solution remains almost unalterable, indicating a low influence of this parameter on the quality of the solution. As with the case of $\sigma(0)$, an optimized value for parameter $\alpha(0)$ can only be obtained empirically and for each case independently. However, as suggested in [34], the use of $\alpha(0) = 0.1$ is appropriate for symmetric TSP instances, and $\alpha(0) = 1.25$ for the other instances.

The value of parameter h in Eq. (8) directly influences the step size of the adaptation of a given antibody's attribute vector. In Fig. 5 we observe that after some epochs the h value is considerably small for large neighborhoods d, thus suggesting that the updating procedure could be constrained so as to avoid unnecessary computational effort. Some authors restrict the adaptation, decreasing the effective neighborhood of the winner neuron by performing the adaptation only to values of d lower than a pre-determined threshold [8,40]. This approach does not consider that, at the beginning of the learning process, some neighbors far from the winner neuron carry out a significant adaptation to the network and that, at the end of the adaptive process, neighbors close to the winner neuron no longer carry out significant adaptations. According to this, we propose that this restriction occurs on h, adding a threshold κ to it, thus allowing a higher flexibility of the method. Therefore, the standard value we propose for the threshold κ in Eq. (8) is $\kappa = 0.01$. To illustrate the gain in performance obtained with this approach for the example studied in this section, eil76, the average computational time was reduced from 17.80 to 14.43 s, a gain in performance of approximately 20%.

For the AVT algorithm [1], $G_i = \text{round}(n/2)$, where round(·) is the operator that rounds the value within parenthesis towards the closest integer and n is the number of cities in the instance. The parameter $\alpha(0)$ was chosen to be 0.002 for ncit64, 0.02 for all other instances of size less than 500, and 0.2 for the instances with more than 500 cities.

In [40] the authors suggest that $G(0)$ has to be defined according to the problem being tested, and in [41] the authors proposed the following heuristics to define $G(0)$:

$$G(0) = 0.06 + 12.41^{*}n. \tag{13}$$

Although the context of [41] is slightly different from that of their previous paper, we chose this heuristics to tune $G(0)$. Parameter α was set to 0.03 for all problems and the learning rate was set to 0.6 for all instances.

4.2. Experimental evaluation

The comparison to be performed here will take into account the total cost of the solutions found by the algorithms, the computational cost (in terms of time and epochs for convergence) of each algorithm, and the percentage deviation of the solutions found in relation to the best known solutions. All algorithms were run 30 times for each instance and the results presented include the best and average solutions found. The number of cities in each instance is the number following the letters that name the instances, for example, in fl1400 the number of cities is 1400.

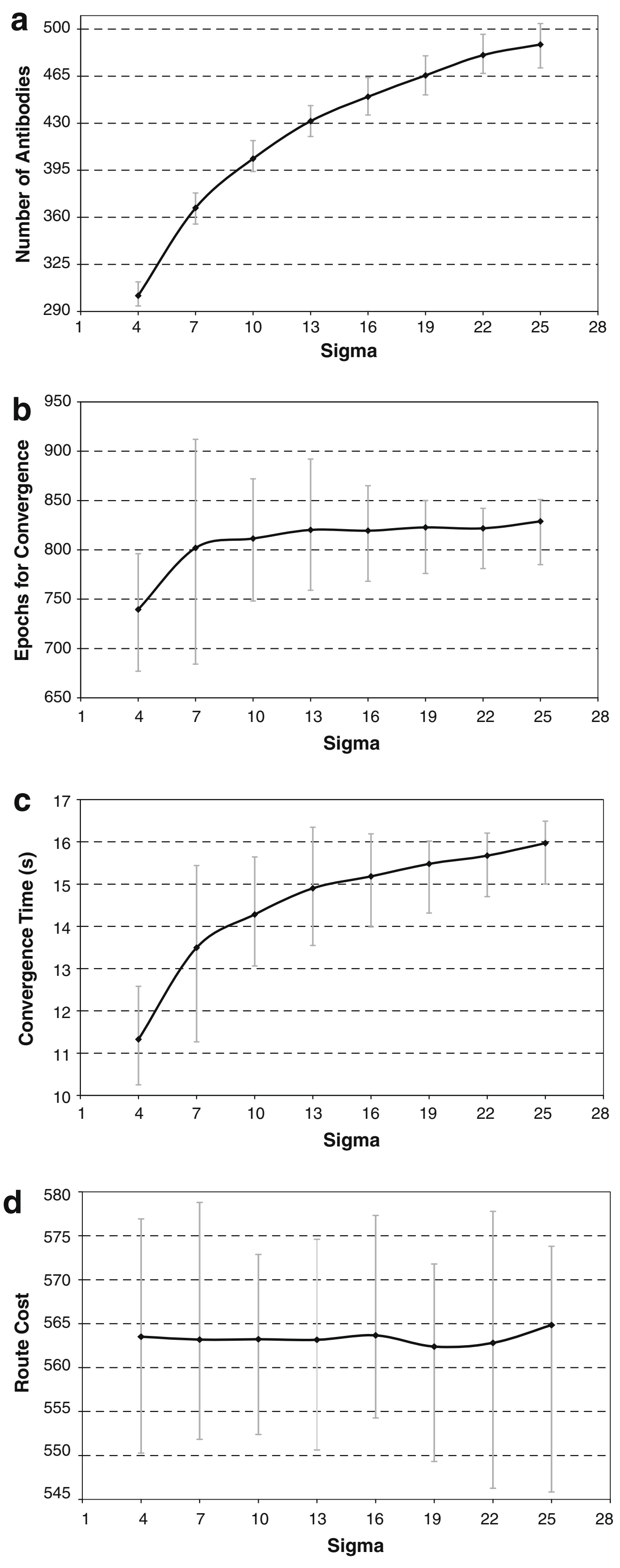


Fig. 3. Influence of the initial value of the parameter $\sigma(0)$ on the proposed algorithm. The black curve represents the average values and the edges of the bars represent the minimum and maximum values, all obtained after 30 runs. (a) Number of antibodies before the pruning phase; (b) Epochs for convergence; (c) convergence time; and (d) route cost.

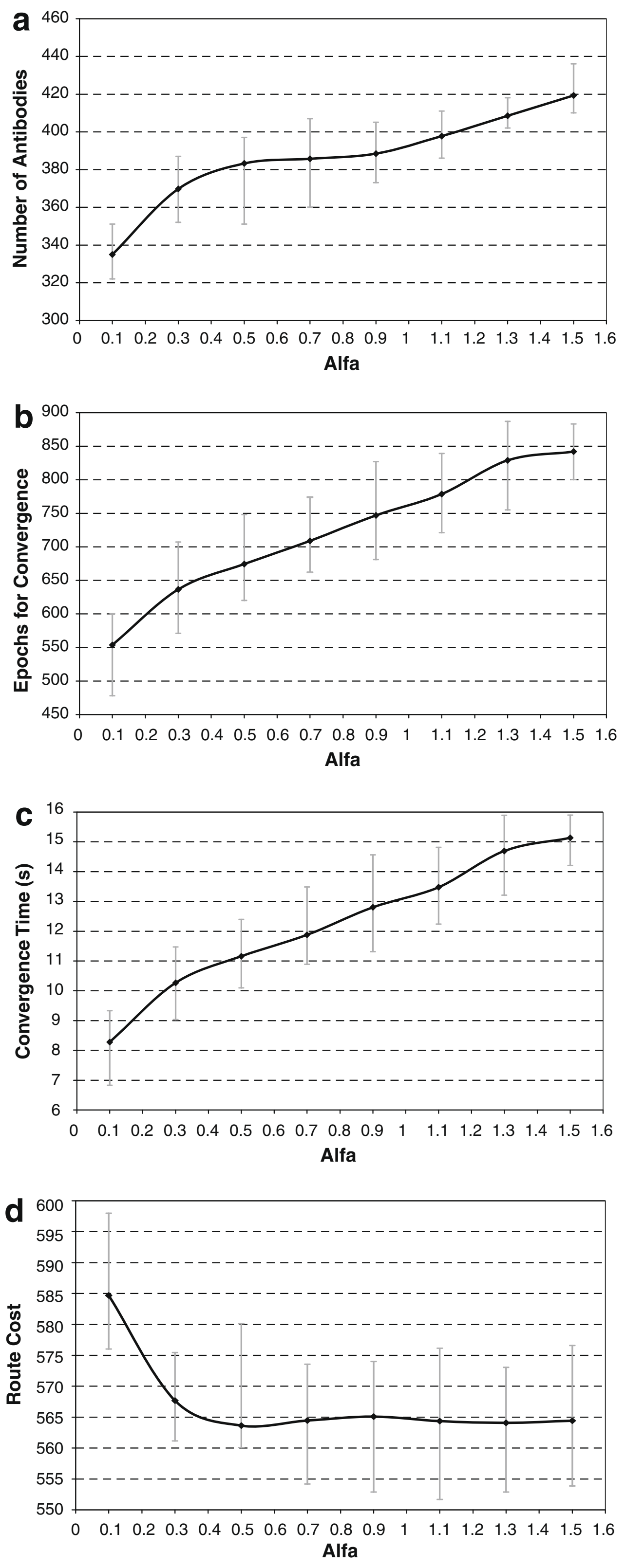


Fig. 4. Influence of the initial value of the parameter $\alpha(0)$ on the proposed algorithm. The black curve represents the average values and the edges of the bars represent the minimum and maximum values, all obtained after 30 runs. (a) Number of antibodies before the pruning phase; (b) Epochs for convergence; (c) Convergence time; and (d) Route cost.

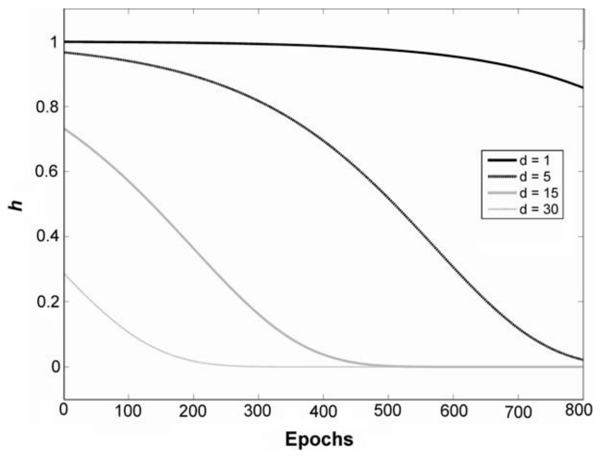


Fig. 5. Influence of the neighborhood function *h* along the iterations for different neighborhood sizes *d*.

Table 1 compares the performance of RABNET-TSP, oRABNET-TSP [34], AVT [1], and SME [40] in terms of the best and average solutions found for thirty independent runs. The best results found are detached in bold, and the table is shadowed for the five instances with more 500 cities. In this table, all algorithms were re-implemented using Matlab 7.0. We chose to re-implement the algorithms in order to reduce the bias introduced by the implementation and programming language when comparing the computational cost of the different algorithms. It is possible to observe from Table 1 that for the smaller instances the modified algorithm presents slightly superior results in absolute terms. For the instances with more than 1000 cities, however, AVT and SME performed slightly better.

Table 2 compares the computational time and number of epochs for convergence for each algorithm. It also brings the percentage deviation of the cost of each solution found by the algorithms when compared to the best known solutions. In this table it is possible to observe that the SME algorithm is the least computational intensive one in terms of time and number of epochs for convergence, but it is the least efficient one in terms of quality (tour length) of the solutions found. By

Table 1
Computational results of RABNET-TSP, oRABNET-TSP, AVT and SME. BKS is the best known solution; Mean is the average solution; SD is the standard deviation; and Best is the best solution found. All results are presented for 30 runs.

TSP Instance	BKS	RABNET-TSP			oRABNET-TSP			AVT			SME		
		Mean	**SD**	**Best**	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Best**
ncit64	6400	6400.00	0.00	**6400**	6400.00	0.00	**6400**	6594.07	72.97	6482	6547.60	66.02	6482
eil51	426	437.47	4.20	**427**	438.70	3.52	429	442.90	4.59	432	440.57	3.44	433
eil76	538	556.33	5.30	**541**	556.10	8.03	542	563.20	6.45	554	562.27	5.23	552
eil101	629	648.63	3.85	**638**	654.83	6.57	641	665.93	7.27	655	655.57	5.95	640
berlin52	7542	7932.50	277.25	**7542**	8073.97	270.14	7716	8363.70	241.82	7778	8025.07	248.79	7715
bier127	118282	120886.33	1158.79	118970	121780.33	1564.02	**118760**	128920.33	6563.13	120110	121733.33	1240.04	119840
ch130	6110	6282.40	60.15	6145	6291.77	64.98	**6142**	6416.80	92.97	6265	6307.23	62.95	6203
ch150	6528	6738.37	76.14	**6602**	6753.20	83.01	6629	6842.80	105.97	6634	6751.13	62.18	6631
rd100	7910	8199.77	80.77	7982	8253.93	148.61	**7947**	8444.50	144.26	8088	8239.40	103.91	8028
lin105	14379	14400.17	44.03	**14379**	14702.23	328.37	**14379**	16111.37	609.10	14999	14475.60	118.24	**14379**
lin318	42029	43696.87	410.06	**42834**	43704.97	391.48	42975	45832.83	541.67	44869	43922.90	383.29	43154
kroA100	21282	21522.73	93.34	**21333**	21868.47	245.76	21369	24678.80	862.09	23009	21616.77	164.21	21410
kroA150	26524	27355.97	327.91	**26678**	27346.43	223.31	26932	29960.90	564.39	28948	27401.33	252.03	26930
kroA200	29368	30190.27	273.38	29600	30257.53	341.98	**29594**	33228.33	638.63	31669	30415.67	132.86	30144
kroB100	22141	22661.47	193.47	**22343**	22853.60	208.56	22596	25966.40	713.12	24026	22622.50	75.33	22548
kroB150	26130	26631.87	232.86	**26264**	26752.13	210.48	26395	29404.53	730.02	27886	26806.33	250.14	26342
kroB200	29437	30135.00	276.78	29637	30415.60	349.50	29831	33838.13	662.85	32351	30286.47	301.23	29703
kroC100	20749	20971.23	108.16	**20915**	21231.60	215.92	**20915**	23496.13	844.30	22344	21149.87	187.99	20921
kroD100	21294	21697.37	157.03	**21374**	22027.87	303.22	21457	23909.03	544.06	23076	21845.73	154.30	21500
kroE100	22068	22714.63	260.16	22395	22815.50	268.37	22427	24828.03	699.55	23642	22682.47	214.05	**22379**
rat575	6773	7115.67	37.47	7047	7125.07	40.91	**7039**	8301.83	106.88	8107	7173.63	39.47	7090
rat783	8806	9343.77	46.97	9246	9326.30	48.45	**9185**	10721.60	108.81	10532	9387.57	39.44	9316
rl1323	270199	305314.33	2315.81	300770	300286.00	2678.56	295060	301424.33	2891.50	**293350**	300899.00	2717.10	295780
fl1400	20127	21110.00	163.29	20851	21070.57	205.16	20745	21174.67	247.33	20649	20742.60	115.80	**20558**
d1655	62128	72113.17	698.57	70918	71431.70	589.16	70323	71168.07	587.19	69975	68046.37	379.33	**67459**

Table 2
Computational effort for each algorithm and percentile deviation in relation to the best known solution. Time is the average running time in seconds, Epochs refers to the average number of epochs for convergence and PDbest is the percentile deviation of the best solution found by each algorithm in relation to the best known solution. All results are presented for 30 runs.

TSP Instance	RABNET-TSP			oRABNET-TSP			AVT			SME		
	Time (s)	**Epochs**	**PDbest**	**Time (s)**	**Epochs**	**PDbest**	**Time (s)**	**Epochs**	**PDbest**	**Time (s)**	**Epochs**	**PDbest**
ncit64	6.87	556.2	**0.00**	16.72	1028.6	**0.00**	50.90	4339.0	1.28	**5.22**	**262.0**	1.28
eil51	7.95	747.3	**0.23**	10.64	764.3	0.70	**3.24**	412.4	1.41	3.34	**249.9**	1.64
eil76	15.07	818.6	**0.56**	19.09	843.9	0.74	6.91	450.2	2.97	**6.77**	**262.6**	2.60
eil101	21.72	845.8	**1.43**	30.60	943.1	1.91	12.14	477.8	4.13	**11.37**	**271.6**	1.75
berlin52	8.91	730.8	**0.00**	11.43	772.4	2.31	**3.33**	415.4	3.13	3.48	**254.7**	2.29
bier127	32.02	872.3	0.58	47.48	1027.3	**0.40**	19.57	507.5	1.55	**17.97**	**286.9**	1.32
ch130	33.31	864.1	0.57	51.43	1037.3	**0.52**	20.70	503.4	2.54	**18.55**	**283.2**	1.52
ch150	39.90	894.4	**1.13**	64.90	1117.8	1.55	27.63	513.2	1.62	**26.02**	**288.1**	1.58
rd100	19.88	803.2	0.91	31.34	960.0	**0.47**	11.91	475.8	2.25	**11.31**	**275.8**	1.49
lin105	20.62	800.9	**0.00**	31.75	927.2	**0.00**	13.27	485.4	4.31	**12.44**	**277.3**	**0.00**
lin318	**110.28**	1038.4	**1.92**	204.44	1393.9	2.25	155.07	586.3	6.76	115.27	**314.0**	2.68
kroA100	15.87	693.2	**0.24**	30.77	944.9	0.41	12.09	479.9	8.11	**11.35**	**277.0**	0.60
kroA150	39.89	616.0	**0.58**	63.27	1075.2	1.54	28.00	518.0	9.14	**26.31**	**290.2**	1.53
kroA200	61.88	954.2	0.79	101.91	1215.2	**0.77**	52.20	542.3	7.84	**45.58**	**299.5**	2.64
kroB100	22.06	630.0	**0.91**	29.94	928.6	2.06	11.90	478.2	8.51	**11.34**	**277.0**	1.84
kroB150	40.61	892.7	**0.51**	64.09	1093.8	1.01	27.93	514.7	6.72	**26.05**	**290.1**	0.81
kroB200	57.91	894.0	**0.68**	107.51	1258.1	1.34	52.49	542.2	9.90	**45.48**	**299.4**	0.90
kroC100	21.65	817.1	**0.80**	30.96	947.0	0.80	11.93	476.7	7.69	**11.34**	**276.9**	0.83
kroD100	21.92	814.5	**0.38**	31.24	948.1	0.77	12.05	475.7	8.37	**11.36**	**277.4**	0.97
kroE100	21.49	834.4	**1.48**	30.48	940.6	1.63	11.92	475.0	7.13	**11.43**	**277.1**	1.41
rat575	215.28	1037.4	4.05	427.10	1473.4	**3.93**	**48.76**	**61.0**	19.70	372.08	330.4	4.68
rat783	352.03	1152.9	5.00	609.14	1499.9	**4.30**	**112.66**	**63.2**	19.60	709.82	340.8	5.79
rl1323	**946.32**	1502.6	11.31	1222.16	1590.0	**9.20**	969.18	**70.63**	8.57	2124.02	362.1	9.47
fl1400	1105.80	1552.0	3.60	1273.23	1551.1	3.07	**502.64**	**69.8**	2.59	2353.00	359.0	**2.14**
d1655	**1569.25**	1766.3	14.15	1839.29	1763.3	13.19	1798.43	**71.27**	12.63	3346.77	367.4	**8.58**

Table 3
Computational results of RABNET-TSP in comparison with other results directly taken from the literature (when available). BKS is the best known solution, PDav is the percentage deviation of the average solution found in relation to the best known solution, and PDbest is the percentage deviation of the best solution found in relation to the best known solution. All results for the RABNET-TSP are taken from 30 runs.

TSP Instance	BKS	RABNET-TSP		SME		Aras et al. [3]		Cochrane and Beasley [8]	
		PDav	**PDbest**	**PDav**	**PDbest**	**PDav**	**PDbest**	**PDav**	**PDbest**
eil51	426	2.69	**0.23**	3.43	1.64	–	2.86	2.89	0.94
eil76	538	3.41	**0.56**	5.46	1.49	–	4.98	4.35	2.04
eil101	629	3.12	1.43	4.17	1.27	–	4.65	3.78	**1.11**
berlin52	7542	5.18	**0.00**	5.81	**0.00**	–	–	7.01	**0.00**
bier127	118,282	2.20	**0.58**	3.41	0.84	–	–	3.00	0.69
ch130	6110	2.82	**0.57**	2.82	1.02	–	–	2.82	1.13
ch150	6528	3.22	**1.13**	3.91	1.76	–	–	3.23	1.78
rd100	7910	3.66	**0.91**	3.99	1.28	–	2.09	3.64	1.19
lin105	14,379	0.15	**0.00**	2.75	0.62	–	1.98	1.08	0.00
lin318	42,029	3.97	**1.92**	4.16	2.17	–	–	4.31	2.65
kroA100	21,282	1.13	**0.24**	2.62	0.31	–	–	1.31	0.57
kroA150	26,524	3.14	**0.58**	4.61	1.47	–	–	3.06	1.55
kroA200	29,368	2.80	**0.79**	3.70	0.86	–	5.72	3.27	0.92
kroB100	22,141	2.35	**0.91**	2.91	1.43	–	–	2.20	1.53
kroB150	26,130	1.92	**0.51**	2.82	1.67	–	–	2.60	1.06
kroB200	29,437	2.37	**0.68**	4.83	1.55	–	–	2.31	0.88
kroC100	20,749	1.07	**0.80**	3.18	0.85	–	–	1.70	**0.80**
kroD100	21,294	1.89	**0.38**	2.28	0.89	–	–	1.87	0.80
kroE100	22,068	2.93	**1.48**	3.35	1.71	–	–	2.56	1.52
rat575	6773	5.06	**4.05**	–	–	–	–	6.20	4.89
rat783	8806	6.11	**5.00**	–	–	–	–	6.57	5.66
rl1323	270,199	13.00	11.31	–	–	–	–	11.85	**11.29**
fl1400	20,127	4.88	3.60	2.78	1.68	–	–	4.26	**2.12**
d1655	62,128	16.07	14.15	-	–	–	–	10.09	**9.10**

comparing the results of RABNET-TSP with those of oRABNET-TSP the effectiveness of the modifications introduced can be observed in the reduced time and number of epochs for convergence without a loss in performance.

In Table 3 a trade-off between the performance of RABNET-TSP and the results of other proposals from the literature is presented. Again it is possible to observe that RABNET-TSP is competitive when the target is to find the lowest cost

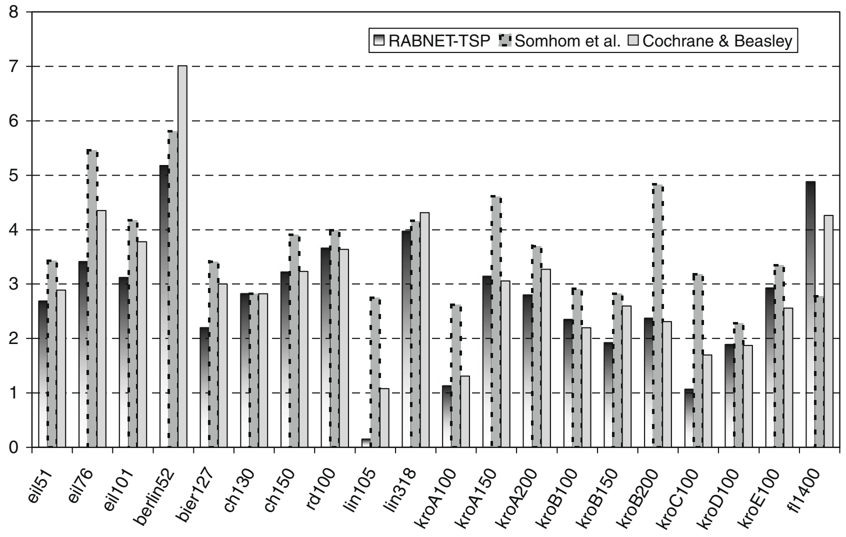


Fig. 6. Percentage deviation of the average solution to the best known solution of each TSP instance to the following algorithms: RABNET-TSP, Somhom et al. [40] and Cochrane and Beasley [8].

routes for the TSP. Besides, the average solution obtained with RABNET-TSP is of good quality, generally, better than the results presented with other algorithms from the literature for the set of instances used in this work, as can be observed in Fig. 6.

Fig. 7 illustrates some of the best routes found by RABNET-TSP and their costs. Note that the way the network grows, like an expanding ring, reduces the possibility of crossings in the routes, which are characteristic of locally optimal routes.

5. Discussion and future investigations

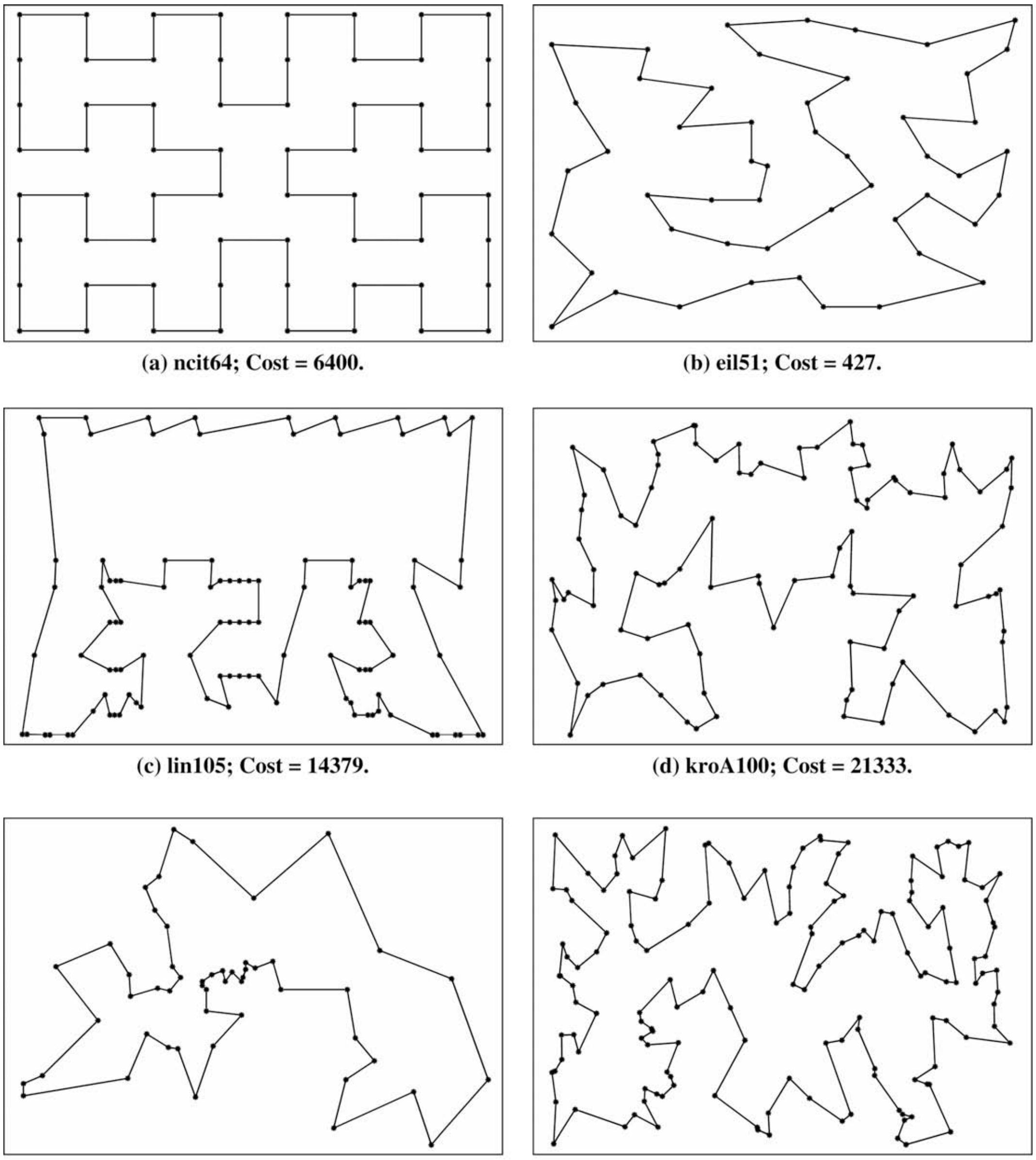
Aiming at improving the efficacy and the efficiency of an immune-inspired network for discrete optimization, this paper proposed two modifications in the original algorithm: (1) the addition of a threshold to the use of the kernel function, Eq. (8), for the antibodies updating; and (2) the use of a winners' stabilization mechanism. The resultant and original algorithms were tested on a much larger set of TSP instances than in the original publication [34] and the results were directly compared to that of two other well-known algorithms from the literature and to the best results available to date. Comparisons with results available in the literature were also made.

It is important to remark that the networks assessed in this work are unsupervised, in the sense that no information about the quality of the solutions found is used to guide the search. In spite of that, the modified algorithm proposed here was capable of finding solutions that are less than 1% worse than the best results found to date for almost all instance with less than 500 cities, thus showing the strength of the self-organizing learning processes. For larger instances, a percentage deviation of up to 14% was found in relation to the best known solutions, which means that either further tuning/improvements could be made in the algorithm or these would only be suitable for smaller scale problems. But this type of conclusion requires further investigation.

In relative terms, the results obtained show that RABNET-TSP is superior when compared with the other approaches implemented in terms of quality of the solutions found (tour length), but in most cases it required a greater computational time to solve the problems. However, for larger TSP instances, with $n \geqslant 1000$, RABNET-TSP was less computational intensive but was not the most efficient method in terms of tour length.

Although the better solutions for each instance were found with an appropriate set of parameters, empirical tests indicated that small variations of these values do not influence significantly the quality of the average solution found. Moreover, the average solution obtained with the proposed algorithm was satisfactory, usually better than those found by similar algorithms. These two characteristics suggest that, even with suboptimal values of the parameters, the algorithm is capable of providing, on average, good quality solutions. However, this also requires further experiments for a more solid conclusion.

The encouraging results presented here led us to adapt the algorithm to solve more complex routing problems, such as the multiple traveling salesmen problem (MTSP) [29], and the capacitated vehicle routing problem (CVRP). Also, further investigations must be performed with larger TSP instances and the hybrid use of the proposed algorithm with route improvement heuristics, such as k-opt and simulated annealing.



(a) ncit64; Cost = 6400. (b) eil51; Cost = 427.

(c) lin105; Cost = 14379. (d) kroA100; Cost = 21333.

(e) berlin52; Cost = 7542. (f) kroB150; Cost = 26264.

Fig. 7. Best routes found by RABNET-TSP: (a) ncit64; (b) eil51; (c) lin105; (d) kroA100; (e) berlin52 and (f) kroB150.

Algorithm 1. RABNET-TSP Pseudocode

```
function [Ab]=RABNET_TSP(Ag,$\alpha(0)$,$\sigma(0)$,$\varepsilon$,$\lambda$,$\kappa$, itMax)
//Ab->antibodies' attribute matrix
//Ag->antigens (matrix with the coordinates of the cities);
//$\alpha(0)$->initial learning rate;
//$\sigma(0)$->initial value for $\sigma$;
//$\varepsilon$->cloning threshold;
//$\lambda$->convergence threshold;
//$\kappa$->cooperation threshold;
//itMax->maximum number of epochs;
n $\leftarrow$ number of cities;
$\tau_1 \leftarrow$ 1000/log($\sigma_o$);
```

```
$\tau_2$ ← 1000;
**Ab** ← initialize a single antibody;
N ← 1;//initial number of antibodies
neighborhood ← 1;
for(it = 1:itMax)
  $\alpha$(it) ← $\alpha$(it − 1)*exp(−it/$\tau_1$);
  $\sigma$(it) ← $\sigma$(it − 1)*exp(−it/$\tau_2$);
  $\gamma$ ← **0**;//concentration vector
  $\upsilon$ ← **0**;//winners vector
  **Ag** ← random order
  for($i$ = 1:n)//for all antigens
    **ag** ← **Ag**$_i$;//current antigen
    J ← competition(**ag**,**Ab**);// Eq. (4)
    $\gamma$(J) ← $\gamma$(J) + 1;//increases the concentration of the winner
    $\upsilon$(k) ← J;//winner for antigen k
    if(neighborhood == 1)
      **H** = cooperation();// Eq. (6)
      **Ab** = adaptation();// Eq. (8)
    else
      **Ab**$_J$ = adaptation();// Eq. (8)
    endIf
  endFor
  mC ← max(**c**);
  if(mC == 1)
    p ← convergence_criterion();
    if(p == 1)
        break;
    endIF
  endIF
  $\Delta$v = winners_stabilize();// Eq. (10)
  if($\Delta$v == 0)
    neighborhood ← 0;
  endIF
  **Ab** ← networkgrow();
  N ← size(**Ab**);
endFor
**Ab** ← networkprune();
```

Acknowledgements

The authors thank CNPq and Fapesp for the financial support.

References

[1] B. Angeniol, G. Croix Vaubois, J.-Y. Le Texier, Self-organizing feature maps and the travelling salesman problem, Neural Networks 1 (1988) 289–293.
[2] D. Applegate, R. Bixby, V. Chvatal, W. Cook, The Traveling Salesman Problem: A Computational Study, Princeton University Press, 2007.
[3] N. Aras, B.J. Oommen, I.K. Altinel, The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem, Neural Networks 12 (1999) 1273–1284.
[4] R. Bronson, G. Naadimuthu, Schaum's Outline of Theory and Problems of Operations Research, second ed., McGraw-Hill, 1997.
[5] L. Burke, "Conscientious" neural nets for tour construction in the traveling salesman problem: the vigilant net, Computers and Operations Research 23 (1996) 121–129.
[6] L.I. Burke, Neural methods for the traveling salesman problem: insights from operations research, Neural Networks 7 (1994) 681–690.
[7] L.I. Burke, P. Damany, The guilty net for the traveling salesman problem, Computers and Operations Research 19 (1992) 255–265.
[8] E.M. Cochrane, J.E. Beasley, The co-adaptive neural network approach to the Euclidean travelling salesman problem, Neural Networks 16 (2003) 1499–1525.
[9] G. Dantzig, R. Fulkerson, S. Johnson, Solution of a large-scale traveling salesman problem, Operations Research 2 (1954) 393–410.
[10] L.N. de Castro, J.I. Timmis, Artificial Immune Systems: A New Computational Intelligence Approach, Springer-Verlag, 2002.
[11] L.N. de Castro, Fundamentals of natural computing: an overview, Physics of Life Reviews 4 (2007) 1–36.
[12] R. Durbin, D. Willshaw, An analogue approach to the travelling salesman problem using an elastic net method, Nature 326 (1987) 689–691.
[13] I. Ellabib, P. Calamai, O. Basir, Exchange strategies for multiple ant colony system, Information Sciences 177 (2007) 1248–1264.
[14] F. Favata, R. Walker, A study of the application of Kohonen-type neural networks to the travelling salesman problem, Biological Cybernetics 64 (1991) 463–468.
[15] J.C. Fort, Solving a combinatorial problem via self-organizing process: an application of the Kohonen algorithm to the traveling salesman problem, Biological Cybernetics 59 (1988) 33–40.

[16] B. Fritzke, P. Wilke, FLEXMAP – a neural network for the traveling salesman problem with linear time and space complexity, in: Proceedings of the International Joint Conference on Neural Networks, IJCNN'91, 1991, pp. 929–934.
[17] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, WH Freeman & Co., New York, NY, 1979.
[18] F.W. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research and Management Science, Springer, 2003.
[19] S. Haykin, Neural Networks: A Comprehensive Foundation, second ed., Prentice Hall, Upper Saddle River, NJ, 1998.
[20] K. Helsgaun, An effective implementation of the Lin–Kernighan traveling salesman heuristic, European Journal of Operational Research 126 (2000) 106–130.
[21] S.-J. Kim, J.-H. Kim, H.-M. Choi, An efficient algorithm for traveling salesman problems based on self-organizing feature maps, in: Proceedings of the Second IEEE International Conference on Fuzzy Systems, 1993, pp. 1085–1090.
[22] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[23] T. Kohonen, Self-Organizing Maps, third ed., Springer, Berlin, 2001.
[24] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.), The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley & Sons, Chichester, 1985.
[25] H.R. Lewis, C.H. Papadimitriou, Elements of the Theory of Computation, second ed., Prentice-Hall, Upper Saddle River, NJ, 1997.
[26] S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations Research 21 (1973) 498–516.
[27] C.-K. Looi, Neural network methods in combinatorial optimization, Computers and Operations Research 19 (1992) 191–208.
[28] S.J. Louis, G. Li, Case injected genetic algorithms for traveling salesman problems, Information Sciences 122 (2000) 201–225.
[29] T.A.S. Masutti, L.N. de Castro, Uma Abordagem Neuro-Imune para a Solução do Problema de Múltiplos Caixeiros Viajantes (in Portuguese), in: Proceedings of the VIII Brazilian Conference on Neural Networks, Florianópolis, SC, 2007, (CD-ROM).
[30] Y. Matsuyama, Self-organizing neural networks and various Euclidean traveling salesman problems, Systems and Computers in Japan 23 (1992) 101–112.
[31] E. Mérida-Casermeiro, G. Galán-Marín, J. Muñoz-Pérez, An efficient multivalued Hopfield network for the traveling salesman problem, Neural Processing Letters 14 (2001) 203–216.
[32] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag, New York, NY, 2000.
[33] M. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33 (1991) 60–100.
[34] R. Pasti, L.N. de Castro, A Neuro-immune network for solving the traveling salesman problem, in: Proceedings of International Joint Conference on Neural Networks, IJCNN'06, 2006, pp. 3760–3766.
[35] R.L. Rardin, Optimization in Operations Research, Prentice-Hall, Upper Saddle River, NJ, 1997.
[36] R. Reilly, P. Tchimev, Neural network approach to solving the traveling salesman problem, Journal of Computing Sciences in Colleges 19 (2003) 41–61.
[37] G. Reinelt, TSPLIB – a traveling salesman problem library, ORSA Journal on Computing 3 (1991) 376–384.
[38] K.A. Smith, M. Palaniswami, M. Krishnamoorthy, Neural techniques for combinatorial optimization with applications, IEEE Transactions on Neural Networks 9 (1998) 1301–1318.
[39] K. Smith, Neural networks for combinatorial optimization: a review of more than a decade of research, INFORMS Journal on Computing 11 (1999) 15–34.
[40] S. Somhom, A. Modares, T. Enkawa, A self-organising model for the travelling salesman problem, Journal of the Operational Research Society 48 (1997) 919–928.
[41] S. Somhom, A. Modares, T. Enkawa, Competition-based neural network for the multiple travelling salesmen problem with minmax objective, Computers and Operations Research 26 (1999) 395–407.
[42] L. Wang, On the dynamics of discrete-time, continuous-state Hopfield neural networks, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 45 (1998) 747–749.