A neural network algorithm for the traveling salesman problem with backhauls

Hassan Ghaziri*, Ibrahim H. Osman

School of Business, Center for Advanced Mathematical Studies, American University of Beirut, P.O. Box 11-0236, Bliss Street, Beirut, Lebanon

Abstract

This paper introduces a new heuristic based on Kohonen’s self-organizing feature map for the traveling salesman problem with backhauls (TSPB). The TSPB is an extension of the traveling salesman problem in which a set of customers is partitioned into a set of linehaul customers to be visited contiguously at the beginning of the route and a set of backhaul customers to be visited once all linehaul customers have been visited. The major innovation of the proposed heuristic is based on the design of a new network architecture, which consists of two separate chains of neurons. The network evolves into a feasible TSPB tour using four types of interactions: (1) the first chain interacts with the linehaul customers, (2) the second chain interacts with the backhaul customers, (3) the tails of the chains interact together, and (4) the heads of the two chains interact with the depot. The generated tour is then improved using the 2-opt procedure. The new heuristic is compared to the best available TSPB heuristics in the literature on medium to large-sized instances up to 1000 customers. The computational results demonstrate that the proposed approach is comparable in terms of solution quality and computational requirements.

Keywords: Asymmetric traveling salesman problem; Traveling salesman problem with backhaul; Competitive neural network; Meta-heuristics; Self-organizing feature maps; Variable neighborhood search

1. Introduction

We consider an extension of the traveling salesman problem (TSP) known as the traveling salesman problem with backhauls (TSPB), in which a set of customers is partitioned into two subsets: linehaul and backhaul customers. Each linehaul customer requires the delivery of a certain quantity of products from the depot, whereas each backhaul customer requires the collection of a certain quantity of products to the depot. The partition of the set of customers is frequently arising in many practical situations, such as the grocery industry, in which, supermarkets and shops are the linehaul customers, and grocery suppliers are
the backhauls customers. The Interstate Commerce Commission estimated a yearly savings of $160 millions in the USA grocery industry due to the introduction of the backhaul customers (Casco, Golden, & Wasil, 1988). Other TSPB applications arise in the automated warehouse routing (Chisman, 1975); in operation sequencing on numerically controlled machines (Lotkin, 1978); in quality stores (Yano et al., 1987) and in children bus transportation (Mosheiov, 1994).

The TSPB can be defined as follows. Let $G = (V, A)$ be a graph, where $V = \{v_0, \ldots, v_n\}$ is the set of nodes and $A = \{(v_i, v_j) : v_i \neq v_j, v_i, v_j \in V\}$ is the set of arcs. It is assumed that the arc $(v_i, v_j)$ is undirected and associated with a cost $C_{ij}$ representing the travel cost/time or distance between customers $i$ and $j$ located at nodes $v_i$ and $v_j$, respectively. The node set is partitioned into $V = \{\{v_0\}, \mathcal{L}, \mathcal{B}\}$ where $v_0$ is the depot, $\mathcal{L}$ is the set of linehaul customers and $\mathcal{B}$ is the set of backhaul customers. The objective of the TSPB is to find the least cost Hamiltonian cycle on $G$ that starts from $v_0$ visiting contiguously all customers in $\mathcal{L}$ and followed by all customers of $\mathcal{B}$ before returning to the depot.

The TSPB belongs to the general class of TSP with precedence constraints. First, it is a special case of the pickup and delivery TSP in which each pickup customer is associated with exactly one delivery customer, the number of pickups equals to the number of deliveries and each pickup must be visited before its associated delivery customer (Kalantari, Hill, & Arora, 1985; Renaud, Doctor, & Quenniche, 2000). Second, it is a special case of the clustered TSP in which $V$ is partitioned into clusters $V_1, \ldots, V_m$ and the clusters, $V_i$'s can be visited in any order, but the customers in each cluster must be visited contiguously (Chisman, 1975; Jongens & Volgenant, 1985; Potvin & Guertin, 1996). Last, it is a sub-problem in the vehicle routing problem with backhauls in which capacity/time restrictions are imposed on the vehicle visiting the linehaul and backhaul customers (Anily, 1996; Osman & Wassan, 2002; Thangiah, Potvin, & Sung, 1996; Toth & Vigo, 1999). For an extensive bibliography on routing problems and their applications, we refer to Laporte and Osman (1995).

The TSPB is NP-hard in the strong sense, since it generalizes the TSP when $\mathcal{B} = \emptyset$. The TSPB can be transformed into a standard asymmetric TSP by adding an arbitrarily large constant to each $C_{ij}$ where $(v_i, v_j)$ is any arc linking any two sets, $\{v_0\}, \mathcal{B}$ and $\mathcal{L}$ (Chisman, 1975). Therefore, the asymmetric TSP exact algorithm of Fischetti and Toth (1992) can be used to solve TSPB. However, when $n$ is large solving the TSPB exactly becomes impractical and approximate methods (heuristics/meta-heuristics) must be used (Osman, 1995).

In the literature, there are a few heuristics for the TSPB. They are modifications of the TSP approaches. For example, Gendreau, Hertz, and Laporte (1992) developed a heuristic for the TSP and called it GENUIS. It consists of a tour construction phase (GENI) and a tour improvement phase (US). Gendreau, Hertz, and Laporte (1996) stated that “the current state of knowledge on the TSPB is still unsatisfactory and more powerful algorithms must be designed”. They modified GENUIS to generate six heuristics for the TSPB: H1, GENUIS is applied to the transformed cost matrix by adding a large constant to solve the TSPB as an asymmetric TSP; H2, a double-cycle approach, which applies GENUIS separately to the set nodes $\mathcal{L}$ and $\mathcal{B}$, then joining these two cycles and the depot to construct a TSPB solution; H3 is H2 but applied to the sets $\mathcal{L} \cup \{v_0\}$ and $\mathcal{B} \cup \{v_0\}$; H4 is the cheapest-insertion with the US post-optimization; H5 is the GENI followed by Or-Opt tour improvement procedure and H6 is the cheapest-insertion followed by the Or-Opt procedure. It was found that H1 is the best performing heuristic. Recently, Mladenovic and Hansen (1997) used GENUIS within a variable neighborhood search framework (GENIUS-VNS) to solve the TSPB. They found that the GENIUS-VNS is better than the original GENUIS by an average of 0.40% with an increase of 30% in running time.

To our knowledge, no work has been published using the neural networks (NN) approach for the
TSPB. In this article, we attempt to fill in this gap by designing a self-organizing feature map (SOFM) for the TSPB. We also show that this heuristic produces good results when compared to the exact TSPB solutions obtained by the asymmetric algorithm of Fischetti and Toth (1992) on small-sized instances and comparable results when compared to the recent heuristics and meta-heuristics on large-sized instances. The remaining part of the paper is organized as follows. In Section 2, we present a brief review of NN on routing problems. In Section 3, we describe our implementation of the SOFM algorithm to the TSPB. Computational results are reported in Section 4 with a comparison to the best in the literature on randomly generated instances of sizes up to 1000 cities. In Section 5, we conclude with some remarks.

2. Neural networks for routing problems

In this section, we give highlight of the most promising NN applications for routing problems. The reader is referred to the following NN surveys for more details on routing problems (Burke, 1994; Looi, 1992; Potvin, 1993; Smith, 1999). The idea of using NN to provide solutions to difficult NP-hard optimization problems originated with the work of Hopfield and Tank (HT) on the TSP (Hopfield \& Tank, 1985). It has been widely recognized that the HT formulation is not ideal for problems other than TSP. The HT method also utilizes a nature of energy function, which causes infeasible solutions to occur most of the time. However, Aiyer, Niranjan, and Fallside (1990) introduced a subspace approach, where a single term is used to represent all constraints into the energy function. As a result, the HT feasibility can now be guaranteed. However, difficulty with the HT approach still exists, it is the convergence to local optima of the energy function. Recently, an investigation into the improvement of the local optima of the HT network to the TSP was conducted by Peng, Gupta, and Armitage (1996). They obtained good results for improving the local minima on small-sized TSP of less than 14 cities. But it was very difficult to obtain similar results for larger-sized instances. Another recent application to the generalized TSP (GTSP) by Andresol, Gendreau, and Potvin (1999) found that the HT is competitive to a recent heuristic for the GTSP with respect to solution quality. But it was computationally more expensive and failed to find feasible solutions when the search space is too large. In our view, much more research is still needed to make the HT network competitive with meta-heuristics (Osman \& Laporte, 1996).

The other more promising NN approach for routing problems is based on the Kohonen’s SOFM (Kohonen, 1982, 1995). The SOFM is an \textit{unsupervised} NN, which does not depend on any energy function. It simply inspects the input data for regularities and patterns and organizes itself in such a way to form an ordered description of the input data. Kohonen’s SOFM converts input patterns of arbitrary dimensions into the responses of a one- or two-dimensional array of neurons. \textit{Learning} is achieved through \textit{adaptation} of the weights connecting the input pattern (layer) to the array on neurons. The learning process comprises a \textit{competitive} stage of identifying a \textit{winning} neuron, which is closest to the input data, and an \textit{adaptive} stage, where the weights of the winning neuron and its \textit{neighboring} neurons are adjusted in order to approach the presented input data.

Applications of the SOFM to the TSP started with the work of Fort (1988). In this approach, one-dimensional circular array of neurons is generated, and mapped onto the TSP cities in such a way that two neighboring points on the array are also neighbors in distance. Durbin and Willshaw (1987) already introduced a similar idea for the TSP and called it an \textit{elastic network}. The elastic network starts with \textit{k} nodes (\textit{k} is greater than \textit{n} ) lying on an artificial ‘elastic band’. The nodes are then moved in the Euclidean space, thus stretching the elastic band, in such a way to minimize an energy function. The main
difference between the two approaches is that Fort’s approach incorporates stochasticsities into the weight adaptations, whereas the elastic network is completely deterministic. Also there is no energy function in Fort’s approach.

Encouraged by the improved NN results, researchers began combining features of both elastic network and SOFM to derive better algorithms for solving TSPs including: Angeniol, De-La-Croix, and Le-Texier (1988), Burke (1994) and Vakhutinsky and Golden (1995). It should be noted that the best NN results for TSP instances taken from the TSPLIB literature were obtained by Aras, Oommen, and Altinel (1999) using a modified SOFM algorithm. Finally, there are also few SOFM applications extended to the vehicle routing problems by Ghaziri (1991, 1996), Modares, Somhom, and Enkawa (1999) and Potvin and Robillard (1995).

3. SOFM for the TSP with backhauls

In this section, we first describe briefly the application of the SOFM algorithm to the TSP. Next, the SOFM approach to the TSPB is explained, followed by its algorithmic steps.

3.1. The SOFM for the TSP

The SOFM approach is applied to the Euclidian TSP, where cities are defined by their coordinates in the two-dimensional Euclidian space. The SOFM algorithm starts by specifying the architecture of the network, which consists of a one ring upon which the artificial neurons are spatially distributed. The coordinates in the Euclidian space and the position on the ring identify each neuron. The positions are initially created as follows. Assuming a ring of \( m \) neurons, the neurons are equally positioned on a circle of radius equal to 1/2 for TSP instances uniformly generated in the unit square, and the angle between two consecutive neurons is equal to \( \frac{360^\circ}{m} \). The Euclidian distance will be used to compare the positions of neurons with the positions of cities. The lateral distance will be used to define the distance on the ring between two neurons. The lateral distance between any two neurons is defined to be the smallest number of neurons separating them plus one. In the first step, a city is randomly picked up; its position is compared to all positions of neurons on the ring. The nearest neuron to the city is then selected and moved towards the city. The neighbors of the selected neuron move also towards the city with a decreasing intensity controlled by the lateral distance. Fig. 1 illustrates the evolution of the algorithm starting from the initial state of the ring, (a) reaching an intermediate stage after some iterations (b) and stopping at the final state (c). The number of neurons should be greater than the number of cities (\( m \) is greater or equal to \( 3n \)) to avoid the oscillation of a neuron between different neighboring cities. An extensive analysis of this algorithm could be found in the works published by Angeniol et al. (1988), Fort (1988) and Smith (1999).

3.2. The SOFM for the TSPB

In this section, we will explain how to extend the SOFM-TSP algorithm to the TSPB. The basic idea is to design a new architecture in which the TSP ring is replaced by two chains: one for the linehauls and one for the backhauls. The linehaul customers will interact exclusively with the first chain, the backhaul customers will interact exclusively with the second chain and the depot will interact with both chains.
Running the SOFM-TSP algorithm on each separate chain will lead to a chain for the linehaul customers and a chain for the backhaul customer. Consequently, the extremities of the two chains will remain free. Connecting the extremities to the depot and to them at the end of the process was not an efficient and successful strategy. To avoid such inefficiency, four types of interactions are introduced to generate a feasible TSPB tour (with $\mathcal{C}_1$ and $\mathcal{C}_2$ representing the first and second chains, respectively):

(i) **Interaction between the first chain $\mathcal{C}_1$ and the linehaul customers in $\mathcal{L}$.** In this interaction, the linehaul customers in $\mathcal{L}$ will be presented to $\mathcal{C}_1$ one by one in a random order. The nearest neuron of $\mathcal{C}_1$ will be selected as the winner neuron. The positions of all neurons belonging to $\mathcal{C}_1$ will be updated according to an adaptation rule to be explained later.

(ii) **Interaction between the second chain $\mathcal{C}_2$ and the backhaul customers in $\mathcal{B}$.** In this interaction, the backhaul customers in $\mathcal{B}$ will interact with $\mathcal{C}_2$ in a similar way to the interaction of type (i) and use the same adaptation rule.

(iii) **Interaction between the two chains $\mathcal{C}_1$ and $\mathcal{C}_2$.** During the previous two types of interactions, the chains are evolved independently. Nothing is forcing them to be connected together to form a feasible route. For this reason, an interaction between the two chains $\mathcal{C}_1$ and $\mathcal{C}_2$, is introduced. We assume that each chain has a head and a tail. The tail and the head are represented by the last and the first neurons, respectively. After presenting all backhaul and linehaul customers, the last neuron of the first chain $\mathcal{C}_1$ will interact with the second chain $\mathcal{C}_2$. The objective of this interaction is to make the tail of the first chain and the head of the second one converge. This convergence will allow the formation of a single ring representing a tour visiting the linehaul and backhaul customers consecutively. The first neuron of the second chain is assigned as the winner neuron in this interaction. This means that the algorithm at this level is not anymore a competitive algorithm but a supervised one in the sense that the last neuron of the first chain has to be attracted by the first neuron of the second chain. After this assignment, the adaptation rule has to be applied on the neurons of $\mathcal{C}_2$. We apply the same procedure to the second chain, by presenting the first neuron of $\mathcal{C}_2$ to the first chain, assigning the last neuron of $\mathcal{C}_1$ as the winner neuron and updating the positions of the neurons of $\mathcal{C}_1$ according to the same adaptation rule.

(iv) **Interaction between the two chains $\mathcal{C}_1$, $\mathcal{C}_2$ and the depot.** This type of interaction is similar to the last one, where the depot is presented to the first chain. The first neuron of $\mathcal{C}_1$ is assigned to the depot and considered as the winner neuron. Once this neuron is assigned, we update this neuron and its neighboring neurons according to the usual adaptation rule. The same procedure is applied to the last of $\mathcal{C}_2$. When we apply all these interactions, the different chains will evolve towards a single ring, passing through the depot, the linehaul customers, the backhaul customers and returning back to the depot. At the end, we
will obtain a feasible route defined by the ring. The position of each customer will coincide with the position of one neuron. The position of the neuron in the chains will give the position of the customers in the route. Fig. 2 illustrates the evolution stages of the SOFM algorithm into a feasible TSPB tour.

The interactions are repeated iteratively at the end of presentations of all cities to the chains and the depot. The SOFM algorithm terminates when the position of the last neuron of \( C_1 \) and the first neuron of \( C_2 \) coincide at which stage a continuous single chain is formed representing a TSPB feasible solution.

The SOFM-TSPB algorithm

**Notations:** let us define the following terms:

- \( \mathcal{L} = \{ l_i = (x_{1i}, x_{2i}) \text{, for } i = 1, \ldots, N_L \} \) be the set of \( N_L \) linehaul customers, where \( (x_{1i}, x_{2i}) \) are the coordinates of the linehaul customer \( l_i \).
- \( \mathcal{B} = \{ b_j = (y_{1j}, y_{2j}) \text{, for } j = 1, \ldots, N_B \} \) be the set of \( N_B \) backhaul customers, where \( (y_{1j}, y_{2j}) \) are the coordinates of the backhaul customer \( b_j \).
- \( D = (x_d, y_d) \) be the coordinates of the depot.
- \( \mathcal{C}_1 = \{ L_i = (X_{1i}, X_{2i}) \text{, for } j = 1, \ldots, N_L \} \) be the set of \( N_L \) connected neurons forming the first linehaul chain.
- \( \mathcal{C}_2 = \{ B_j = (Y_{1j}, Y_{2j}) \text{, for } j = 1, \ldots, N_B \} \) be the set of \( N_B \) connected neurons forming the second backhaul chain.
- \( t \) is the iteration counter.
- \( \eta \) is the control parameter of the adaptation rule.
- \( d_L \) is the lateral distance.
- \( \sigma \) is the parameter controlling the width of the Gaussian function in the adaptation rule. A large \( \sigma \) means that the neighborhood of the winner neuron is large. Decreasing this parameter will allow the algorithm to converge faster.
SOFM algorithmic steps:

Step 1. Initialization

a. Read the input data for the linehaul and backhaul customers.
b. Generate the positions of \(N_L\) neurons located on the first chain \(C_1\), where \(N_L = 3N_1\).
c. Generate the positions of \(N_B\) neurons located on the first chain \(C_2\), where \(N_B = 3N_2\).
d. Set the initial state of each chain to a vertical line, see Fig. 2.
e. Set \(t = 0\), \(\sigma(0) = 1\) and \(\eta(0) = 3\).

Step 2. Select an input

a. Select randomly a customer \(C \in \mathcal{L} \cup \mathcal{B}\). Every customer will be selected exactly once during one iteration.
b. Let us define the city to be represented by \(C = (x_{1c}, x_{2c})\) or \(C = (y_{1c}, y_{2c})\) depending on whether it belongs to \(\mathcal{L}\) or \(\mathcal{B}\).

Step 3. Select the winner neuron

**If** \((C \text{ belongs to } \mathcal{L})\) **Then**

a. Selection of the nearest neuron. Let \(L^*\) be the winning neuron belonging to \(\mathcal{L}_1\), i.e. \(L^* = (X_{1c}, X_{2c})\) such that \((x_{1c} - X_{1c})^2 + (x_{2c} - X_{2c})^2 \leq (x_{1c} - X_{1c})^2 + (x_{2c} - X_{2c})^2\), \(\forall i = 1, ..., N_L\).
b. Adaptation rule. Update the coordinates of each linehaul neurons, \(L_i = (X_{1i}, X_{2i})\). For example, the update of the \(X\)-coordinate of \(L_i\) is done as follows
   \[
   X_{1i}(t+1) = X_{1i}(t) + \eta(t) \times \Gamma(L_i, L^*) \times (x_{1c} - X_{1i}(t)), \quad \forall i = 1, ..., N_L,
   \]
   where
   \[
   \Gamma(L_i, L^*) = \frac{1}{\sqrt{2}} \exp \left( -\frac{d_i^2(L_i, L^*)}{2\sigma^2} \right).
   \]

**Else** \((C \text{ belongs to } \mathcal{B})\)

c. Selection of the nearest neuron. Let \(B^*\) be the winning neuron belonging to \(\mathcal{B}_2\), i.e. \(B^* = (Y_{1c}, Y_{2c})\) such that \((y_{1c} - Y_{1c})^2 + (y_{2c} - Y_{2c})^2 \leq (y_{1c} - Y_{1c})^2 + (y_{2c} - Y_{2c})^2\), \(\forall j = 1, ..., N_B\)
d. Adaptation rule. Update the coordinates of each backhaul neurons, \(B_j = (Y_{1j}, Y_{2j})\). For example, the update of the \(Y\)-coordinate of \(B_j\) is done as follows
   \[
   Y_{1j}(t+1) = Y_{1j}(t) + \eta(t) \times \Gamma(B_j, B^*) \times (y_{1c} - Y_{1j}(t)), \quad \forall j = 1, ..., N_B
   \]
   where
   \[
   \Gamma(B_j, B^*) = \frac{1}{\sqrt{2}} \exp \left( -\frac{d_j^2(B_j, B^*)}{2\sigma^2} \right).
   \]

Step 4. Extremities interactions
a. Interaction with the Depot. Consider the depot as a customer that should be served by both chains. Assign the heads of both chins as the winning neuron. Apply Step 3.b or 3.d.
b. Interaction of the two chains. Consider the tail of each chain as a customer for the other chain and apply step 3 accordingly.

Step 5. End-iteration test

If Not {all customers are selected at the current iteration} Then go to Step 2.

Step 6. Stopping criterion

If {all customers are within $10^{-4}$ of their nearest neurons in the Euclidean space} Then Stop
Else

a. Update $\sigma(t + 1) = 0.99\sigma(t)$ and $\eta(t + 1) = 0.99\eta(t)$.
b. Set $(t = t + 1)$ and Go to Step 2.

If the SOFM algorithm is followed by the post-improvement phase using the 2-opt improvement procedure applied separately to the TSP segments associated with the backhaul and linehaul customers of the TSPB, then the new variant is denoted by SOFM*. It will be shown that the solution quality of SOFM algorithm can be improved substantially by SOFM* with a little extra effort.

4. Computational experience

Our computational experience is designed to analysis the performance of the SOFM heuristic and its variant SOFM* in terms of solution quality and computational requirements. The experimental design consists of two experiments. The first experiment is designed to compare the performance of the proposed heuristics with a branch and bound (B&B) tree search procedure for the asymmetric TSP developed by Fischetti and Toth (1992) on small-sized instances. The second experiment is designed to compare their performance with the best existing heuristics in the literature on medium to large-sized instances.

4.1. Test instances and performance measure

The first experiment uses a set of 100 test instances. Each instance involves 50 backhaul and linehaul customers to be served from a single depot. The customers and depot coordinates are randomly generated in the unit square $[0, 1]^2$ according to a continuous uniform distribution. The set of 100 test instances is divided into five groups. Each group contains 20 instances with a given ratio for randomly selecting the backhaul customers. The ratio is computed by $\rho = |B/V|$, the number of backhaul customers over the total number of backhaul and linehaul customers. The $\rho$ values are given in the set $\Omega = \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

The second experiment uses a set of 750 test instances, which are generated according to the experimental design defined by Gendreau et al. (1996), used by Mladenovic and Hansen (1997) and
explained as follows. The coordinates of the customers and the depots are generated in the interval [0, 100]^2 according to a continuous distribution. For each pair \( (n, \rho) \) where \( n = 100, 200, 300, 500 \) and 1000 and \( \rho \in \Omega \), 30 instances are generated.

The quality of an algorithm is measured by the relative percentage deviation (RPD) of the solution value from its lower bound, optimal solution, or best-found value depending on whatever available. For each given pair of \( (n, \rho) \), the average of solutions and the average of RPD values (ARPD) over the set of instances in the group are reported.

4.2. Comparison with the B&B tree search algorithm

The objective of this section is to analyze the quality of the SOFM and SOFM^p heuristics, and the effect of backhaul ratio (\( \rho \)) on the performance of all compared algorithms. For a given instance, the associated cost matrix of the TSPB is converted into an equivalent cost matrix associated with an asymmetric TSP by adding large distances to arcs connecting the subsets \{D\}, \( \mathcal{L} \) and \( \mathcal{B} \). The modified cost matrix is then solved by the B&B tree search algorithm to obtain its exact optimal solution.

The computational results for the compared algorithms are reported in Table 1. The ARPD values in this table are measured from lower bound values used in the B&B tree search algorithm. A number of observations can be deduced from the results. First, it can be seen that the CPU time increases with increasing value of \( \rho \) for all algorithms, which indicates that the more backhaul customers the more difficult it is to solve the associated instance. Second, the SOFM^p heuristic improves substantially the quality of SOFM by an average of 0.33 with a small increase in CPU time of 0.05 on the average.

Third, the deterioration behavior of the SOFM algorithm as the \( \rho \) values decreases is smoothed by SOFM^p. More specific, the SOFM solution quality is worst when \( \rho = 0.1 \) and best when the number of linehauls and backhauls is balanced at \( \rho = 0.5 \). The unbalance in customer types creates two chains of different lengths. For example, the TSP on \( \mathcal{B} \) is very long, whereas the TSP on \( \mathcal{L} \) is very short when \( \rho = 0.1 \). For the long chain, the TSP solution obtained by SOFM seems to be worse than that for a shorter chain. This is expected as the NN algorithms are not the best one for solving TSPs. Hence, more improvements are expected with small ratios and this explains the success of SOFM^p. Its ARPD values from optimal solutions are leveled off at less than 0.1. Consequently, SOFM^p seems to be able to remove the behavioral deterioration of SOFM and the worst ARPD of SOFM is improved the best, as shown in Fig. 3. Moreover, SOFM^p obtained the optimal solutions for half of the instances, and when a solution

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>SOFM</th>
<th></th>
<th>SFOM^p</th>
<th></th>
<th>B&amp;B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARPD</td>
<td>CPU</td>
<td>ARPD</td>
<td>CPU</td>
<td>ARPD</td>
</tr>
<tr>
<td>0.1</td>
<td>7.47</td>
<td>2.00</td>
<td>7.02</td>
<td>2.04</td>
<td>6.97</td>
</tr>
<tr>
<td>0.2</td>
<td>7.98</td>
<td>2.20</td>
<td>7.53</td>
<td>2.25</td>
<td>7.43</td>
</tr>
<tr>
<td>0.3</td>
<td>8.01</td>
<td>2.30</td>
<td>7.72</td>
<td>2.37</td>
<td>7.63</td>
</tr>
<tr>
<td>0.4</td>
<td>8.13</td>
<td>2.33</td>
<td>7.89</td>
<td>2.39</td>
<td>7.82</td>
</tr>
<tr>
<td>0.5</td>
<td>8.25</td>
<td>2.38</td>
<td>7.98</td>
<td>2.42</td>
<td>7.94</td>
</tr>
<tr>
<td>Average</td>
<td>7.95</td>
<td>2.24</td>
<td>7.63</td>
<td>2.29</td>
<td>7.56</td>
</tr>
</tbody>
</table>
was not optimal, its RPD value did not exceed 0.23. Last, the computational requirement by SOFM and its variants are a fraction of that required by the B&B algorithm, making them a good tool for large-sized instances.

4.3. Comparison with existing heuristics

The objective of this section is to compare our heuristics with the best heuristics in the literature. The second data set of medium to large-sized instances is used for comparing SOFM, GENI, GENIUS, GENIUS-VNS and SOFM*. The results for GENI and GENIUS heuristics developed by Gendreau et al. (1996) and for the GENIUS-VNS heuristic developed by Mladenovic and Hansen (1997) using GENIUS within a variable neighborhood framework are taken from the paper of Mladenovic and Hansen (1997). For a given pair \((n, \rho)\), for each compared algorithm, the average of the objective values and the average of CPU times are reported over the set of 30 instances in Table 2. The computational results for SOFM and SOFM* are obtained from a single run on each data instance.

From the results in Table 2, the followings can be observed after conducting a paired-sample analysis on the difference of the overall averages between SOFM and GENI, SOFM* and GENIUS, and SOFM* and GENIUS-VNS. First, it was found that the overall average solutions of SOFM is significantly better than GENI with a probability of rejection such significance at \(p\)-value = 0.000. Hence, SOFM is the best among the two constructive methods. Second, it was similarly found that SOFM* is significantly better than GENIUS with a \(p\)-value of 0.035, and GENIUS-VNS is significantly better than SOFM* with a \(p\)-value of 0.000. Looking more closely at the ARPD plotted in Fig. 4, we can see clearly that the ranking from best to worst is GENIUS-VNS, SOFM*, GENIUS, SOFM and GENI. From the figure, it is interesting to notice that as the number of customers increases the relative performance of the neural heuristic improves. Hence, the neural heuristics are behaving better on large-sized instances. Such behavior is going against the results obtained by other NN type, such as the Hopfield–Tank approach. Another observation from the figure, with \(n = 200\), the ARPD of GENIUS-VNS is not coinciding with the X-axis, this is due to the fact that SOFM* has found better solutions for some instances than GENIUS-VNS. It should be noted that SOFM* uses a very simple 2-opt improvement procedure compared to the more sophisticated improvement procedure used in GENIUS and GENIUS-VNS. If the sophisticated improvement (US) is embedded in SOFM*, further improvements can be obtained.
Table 2
Computational results on medium to large-sized instances

<table>
<thead>
<tr>
<th>n</th>
<th>r</th>
<th>Average of solution values</th>
<th>Average CPU time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SOFM</td>
<td>GENI</td>
</tr>
<tr>
<td>100</td>
<td>0.1</td>
<td>1043.56</td>
<td>1012.50</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>1072.30</td>
<td>1068.70</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1108.54</td>
<td>1109.66</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1131.83</td>
<td>1125.63</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>1123.29</td>
<td>1133.87</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1095.90</td>
<td>1090.07</td>
</tr>
<tr>
<td>200</td>
<td>0.1</td>
<td>1436.12</td>
<td>1418.63</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>1489.91</td>
<td>1498.83</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1545.00</td>
<td>1550.52</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1554.69</td>
<td>1585.76</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>1553.90</td>
<td>1586.93</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1515.92</td>
<td>1528.13</td>
</tr>
<tr>
<td>300</td>
<td>0.1</td>
<td>1702.60</td>
<td>1720.82</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>1787.14</td>
<td>1824.62</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1877.15</td>
<td>1886.48</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1876.49</td>
<td>1903.29</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>1891.39</td>
<td>1927.34</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1826.95</td>
<td>1852.51</td>
</tr>
<tr>
<td>500</td>
<td>0.1</td>
<td>2168.59</td>
<td>2197.16</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>2310.70</td>
<td>2342.99</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2398.49</td>
<td>2409.80</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>2441.94</td>
<td>2443.12</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2428.72</td>
<td>2464.11</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>2349.69</td>
<td>2371.44</td>
</tr>
<tr>
<td>1000</td>
<td>0.1</td>
<td>3083.59</td>
<td>3099.17</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>3279.02</td>
<td>3281.34</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>3327.04</td>
<td>3366.07</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>3392.17</td>
<td>3451.02</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>3408.41</td>
<td>3455.69</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>3298.05</td>
<td>3330.66</td>
</tr>
<tr>
<td></td>
<td>Overall^c</td>
<td>1973.14</td>
<td>1989.87</td>
</tr>
</tbody>
</table>

^a CPU time in seconds on an Intel Pentium MMX, 233 MHz (32 Mflop/s).
^b Average CPU time in seconds on Sun SPARC 10 (23 Mflop/s).
^c CPU time in seconds converted into equivalent Sun SPARC 10.
Last, we have investigated the effect of the backhauls ratio and the size on the performance of SOFM\(^*\) in Fig. 5. It can be seen that when the ratio is small, the instance becomes like at classical TSP, SOFM\(^*\) seems not to be competitive with other heuristics. The ARPDs vary in the interval \([0.3\text{–}0.9]\). In such cases, the sophisticated improvement (US) procedure would be helpful to improve the SOFM solution. However, when the ratio is high, it produces good solutions with ARPDs varying in the interval \([0.05\text{–}0.60]\). The ARPDs get even better results when the instance size increases.

4.4. Comparison of CPU time

The compared algorithms are coded in different programming languages and run on different computers. The SOFM and SOFM\(^*\) heuristics are coded in C, the B&B tree search algorithm of Fischetti and Toth (1992) in Fortran and they are run on a PC Intel Pentium MMX, 233 MHz under the Windows 98 operating system. Whereas, GENI, GENIUS, and GENIUS-VNS are coded in C\(^++\) programming language and executed on a Sun SPARC 10, Mladenovic (2001). Hence, it becomes very difficult to compare algorithms, as it is not only the speed of the CPU that indicates the performance, but also the memory, compilers play a role. However, the benchmark results in Dongarra (2002) can be used to give a rough guideline on the performance of different computers in terms of millions floating-point
operations completed per second, Mflop/s. The Sun SPARC 10 performance is 23 Mflops, whereas the Intel Pentium MMX 233 MHz performance is 32 Mflops. Hence, our CPU time must be multiplied by a factor of 1.39 to get an equivalent Sun SPARC 10 time. Our Mflop/s is estimated from Kennedy (1998) and Dongarra’s results. Looking at the results in Table 2, it can be seen that GENIUS-VNS improves over SOFM* by 0.29% and it is also 46% faster, on the average. However, looking at the large-sized instances with \( n = 1000 \), SOFM* uses 20% more CPU time than GENIUS-VNS with comparable results. This indicates that the speed gap might be further reduced for larger sized instances. However, larger sized instances were not considered due to the lack of comparative benchmark results.

5. Conclusion

The SOFM heuristic and its variant with 2-opt post-optimization SOFM* are designed and implemented for the TSPB. Their comparisons with the branch and bound exact method and the best existing heuristics show that they are competitive with respect to solution quality, but they require more computational effort, similar to other NN in the literature. In particular, the constructive SOFM heuristic gives better results than GENIUs constructive heuristic. Moreover, SOFM* produces better results than GENIUS and very comparable to the later variant GENIUS-VNS, which is based a variable neighborhood framework. From the experimental analysis, the B&B exact method must be recommended for small-sized instances, GENIUS-VNS for medium-sized instances and SOFM* for large-sized instances.

Our neural approach is by far more powerful, flexible and simple to implement than the Hopfield–Tank NN method. The major limitation of the SOFM algorithm is its inability to address non-Euclidean instances. Further research can be conducted by applying the SOFM heuristic on the modified cost matrix associated with the equivalent asymmetric TSP to find out whether better results can be obtained. In addition, the effect of replacing the 2-opt post-optimization procedure by a more sophisticated one similar to one used in the competitive heuristics should be investigated.

Acknowledgments

We thank Professor Poalo Toth for making the exact method code available and the four referees for their comments that improved the clarity of the paper. We also thank Mr Iyad Wehbe from RCAM (Research and Computer Aided Management Company, Lebanon) for helping us in coding and running the various experiments.

References


Mladenovic, N (2001). Personal communication.


