

Alessio Plebe, Angelo Marcello Anile

**A Neural Network Based Approach  
to the  
Double Traveling Salesman Problem**

*This is a preprint of the extended article on **Neural  
Computation**, 2001, 14(2) MIT Press, first reviewed  
on August 17 2000, finally accepted on April 26 2001*

# **A Neural Network Based Approach to the Double Traveling Salesman Problem**

Alessio Plebe <sup>1</sup>, Angelo Marcello Anile

Department of Mathematics and Informatics, University of Catania  
V.le Andrea Doria, 8 I-95125 Catania, Italy  
email: [alessio@dmi.unict.it](mailto:alessio@dmi.unict.it) [anile@dmi.unict.it](mailto:anile@dmi.unict.it)

This work was supported in part by the European Community ESPRIT Project 6715  
CONNYS (Robot Control based on Neural Network Systems)

---

<sup>1</sup>Correspondence author

## List of Symbols

### Abstract

The Double Traveling Salesman Problem is one of the variation of the basic Traveling Salesman Problem where targets can be reached by two salespersons operating in parallel. The real problem addressed by this work concerns the optimization of the harvest sequence for the two independent arms of a fruit harvesting robot. This application poses further constraints, like a collision avoidance function.

The proposed solution is based on a self-organizing map structure, initialized with as many artificial neurons as the number of targets to be reached. One of the key components of the process is the combination of competitive relaxation with a mechanism for deleting and creating artificial neurons. Moreover, in the competitive relaxation process information about the trajectory connecting the neurons is combined with the distance of neurons from the target. This strategy prevents tangles in the trajectory and collisions between the two tours. Results of tests indicate that the proposed approach is efficient and reliable for harvest sequence planning. Moreover, the enhancements added to the pure self-organizing map concept are of wider importance, as proved by a Traveling Salesman Problem version of the program, simplified from the “double” version for comparison purpose.

$\mathcal{C}$	=	set of neurons candidates for a target competition
$\mathcal{D}$	=	set of targets in disjointed areas not yet assigned
$\mathcal{H}$	=	shortest tour
$\mathcal{L}$	=	targets on the left tour
$\mathcal{O}$	=	set of targets in the overlap area not yet assigned
$\mathcal{R}$	=	targets on the right tour
$\mathcal{S}$	=	set of neurons in the neighborhood of a neuron
$\mathcal{T}$	=	set of all targets
$\mathcal{U}$	=	set of targets not yet assigned
$\mathcal{X}$	=	set of neurons
$D$	=	Euclidean distance function
$D^p$	=	Euclidean distance from a point to the target
$D^s$	=	Euclidean distance from a segment to the target
$E$	=	<i>activity</i> of a neuron
$E^T$	=	<i>activity</i> of a target
$F$	=	<i>frustration</i>
$K$	=	<i>knowledge</i>
$L$	=	size of the neighborhood of a neuron
$N$	=	total number of targets and number of neurons
$O$	=	number of targets in the overlap area
$T$	=	target

$U$	=	number of unassigned targets
$W$	=	winner history function
$X$	=	neuron
$h$	=	segment of a tour
$j$	=	current target index
$i$	=	current neuron index (unless otherwise stated)
$k$	=	tour index $\in \{1, r\}$
$n$	=	current iteration index
$s$	=	length variable along the trajectory
$w$	=	index of current winner neuron
$\mathbf{c}$	=	centroid of targets in a space subregion (vector of coordinates)
$\mathbf{t}$	=	vector of target coordinates
$\mathbf{x}$	=	vector of neuron weights
$\alpha$	=	decay of neural activity
$\beta$	=	weighting factor for the tour initialization
$\delta$	=	decay of neuron neighborhood size
$\gamma$	=	threshold for regeneration/adaptation state change
$\eta$	=	learning rate
$\eta_0$	=	initial learning rate
$\eta_N$	=	learning rate decay
$\nu$	=	linear density of neurons in the tour
$\varrho$	=	area density of targets
$\tau$	=	longest time lag of the winner history

## 1 Introduction

The traveling salesman problem (TSP) is probably the most famous topic in the field of combinatorial optimization, and many studies have searched for a reliable solution. In this problem the objective is to find the shortest route between a set of  $N$  randomly located cities, in which each city is visited only once. The number of possible routes grows exponentially with an increase in the number of cities. This makes an exhaustive search for the shortest route unfeasible, even if there are only a small number of cities. The high level of interest in the TSP has arisen from the wide range of related optimization problems in fields ranging from parts placement in electronic circuits, routing in communication network systems, and resource planning. For general overviews of TSP see (Lawler et al., 1985) and (Reinelt, 1994).

However, real problems are seldom defined in terms of just the pure TSP, often involve multiple traveling salesman planning routes between a single set of cities, and include additional constraints. A well-known case is the Vehicle Routing Problem (VRP), involving the design of a set of minimum cost delivery routes for a fleet of vehicles, originating and terminating at a central depot, serving a set of customers (Golden and Assad, 1988), (Laporte, 1992), (Fisher, 1995). In the multi-Vehicle Covering Tour Problem ( $m$ -CTP), only a subset of all vertices have to be collectively covered by the  $m$  vehicles (Hodgson et al., 1998), (Hachicha et al., 2000). Examples of the many possible additional constraints are time deadlines for serving customers (Thangiah et al., 1993), or time windows (Desrochers et al., 1992), or equity in customer satisfaction over periodic services (Dell et al., 1996).

From a theoretical point of view all such problems could be mapped in an equivalent TSP formulation, but in practice this is not a viable route to an efficient solution. Therefore, there is a vast literature on heuristics tailored to each abstract class in the multifarious family of TSP variants.

The work presented here considers the case in which there are two salesmen who each visit an independent subset of the cities. As in the TSP algorithm each city should be visited only once, and each tour should start and end in the same city. In this case the best solution is the one in which there is both the least difference between the length of the routes taken by the two salesmen and the total distance traveled in visiting all of the cities

is shortest.

This work has been developed to plan the path of the two arms of a robot that was designed to harvest oranges. The two arms of this robot are electrically driven and they move along the direction of unit vectors in a spherical coordinate system. The 2-TSP algorithm here described has been successfully implemented in the CRAM-OPR harvesting robot prototype, a general description of this robot and results of harvesting in real conditions are available in (Recce et al., 1996; Plebe and Grasso, *ress*).

The 2-TSP algorithm incorporates two of the constraints arising from the robotic application. One is the partition of the space allowed for the two salesmen. The total space where the targets lay, is divided into an *overlap* area where both salesmen can reach targets, and two *disjoint* areas where only one of the salesmen can move. The other constraint is that in the overlap area the paths planned by the two salesmen cannot cross.

Despite being conceived for a real application with specific constraints, the solving algorithm turns out to be also a very promising neural solution to the pure TSP.

## 1.1 Neural network approach

The idea of investigating neural networks as an alternative to the classical heuristics in the operations research domain in solving the TSP problem has gained widespread attention since the early work of Hopfield & Tank (Hopfield and Tank, 1985) and Durbin & Willshaw (Durbin and Willshaw, 1987) (see (Potvin, 1993) for a recent survey).

Both pioneer methods basically perform a gradient descent search of an appropriate energy function. The Hopfield-Tank model suffers from the expensive network representation, with a number of nodes proportional to the square of the cities. Despite subsequent efforts in the tuning of the involved parameters (Kagmar-Parsi and Kagmar-Parsi, 1992), (Lai and Coghill, 1992), and in several enhancements (Lo, 1992), (Van den Bout and Miller, 1989), there is an intrinsic difficulty in the quadratic non-convex formulation of the TSP objective function, with many more local minima than a standard linear formulation (Smith, 1996).

Instead, a linear formulation is shown to be the energy function of the so-called *elastic net* paradigm of Durbin-Willshaw, originated from models

of the biological visual cortex, which has been adopted and extended by several researchers (Boeres et al., 1992), (Vakhutinsky and Golden, 1995). Although quite accurate, this method is still computationally expensive.

Most recently, several algorithms have relied on the self-organizing feature map (SOM) (Kohonen, 1984), (Kohonen, 1990), (Kohonen, 1995), which is also a biologically motivated model, and apparently shares some similarity with the elastic net, but is not mathematically equivalent to an energy-function formalism any more (Erwin et al., 1992).

First attempts in using SOM to solve the TSP can be found in (Fort, 1988) and (Heuter, 1988). In this method the activity and weights of a set of artificial neurons is adapted to match the topological relationship of the targets within the input set, by means of iterative competitive learning. The input sample is presented to the neurons, and the connection weights to the neuron with the best match, together with those of neighboring neurons are modified in the direction which strengthens the match. This process results in a neural encoding that gradually relaxes towards a valid tour.

Although in its original form the SOM does not perform very well for the TSP, it is an attracting paradigm, which has been used by several researchers as a basis for more refined schemes: (Angéniol et al., 1988), (Fritzke and Wilke, 1991), (Burke and Damany, 1992), (Burke, 1994), (Burke, 1996) and (Aras et al., 1999).

In general the relaxation process of the SOM is not sufficient to successfully find an optimal tour, since in most cases there is a mismatch between the relative spatial distribution of randomly placed neurons and the location of the targets.

This drawback has been addressed in the algorithm of Angéniol et al. by a network, which is growing starting with just one neuron, and cloning neurons as long as they win on more than one city during the same iteration step. The same algorithm has been extended by Goldstein to the *m*-TSP problem (Goldstein, 1990). In the "guilty net" of Burke the number of neurons is fixed, but the function used in the competition is modified with a penalty term for neurons that win too often. Aras et al. included in the SOM adaptation a mechanism for restoring the centroid of the overall network to the centroid of the targets.

## 2 Description of the algorithm

First a formal definition of the problem dealt by the present algorithm will be given. In a 2-D space where the coordinate of the targets are normalized in the interval  $[0, 1]$ , it is assumed that the common area allowed to both salesmen is defined by two limits on the  $x$  dimension. The left salesman extends on the right side up to  $x_2$  and the right salesman extends to the left up to  $x_1$ , with  $x_1 < x_2$ .

Given a set of  $N$  targets  $\mathcal{T}$ , let  $\mathcal{H}(\mathcal{T})$  be the shortest tour for  $\mathcal{T}$  and  $d(\mathcal{H}(\mathcal{T}))$  be its length.  $\mathcal{H}(\mathcal{T})$  is a set of  $N$  segments  $h_i$  connecting two targets  $T_i^-, T_i^+$ , with  $T_i^+ = T_{i+1}^-$ ,  $T_N^+ = T_0^-$  and  $T_i^+, T_i^- \in \mathcal{T}$ . The double traveling salesman problem (2-TSP) addressed here is formally the following

**Problem 2.1 (2-TSP)** From all pairs of target sets  $\{\mathcal{L}, \mathcal{R}\}$ ,  $\mathcal{L} \in \mathcal{T}$ ,  $\mathcal{R} \in \mathcal{T}$  that satisfy:

- 1)  $\mathcal{L} \cup \mathcal{R} = \mathcal{T}$ ;
- 2)  $\mathcal{L} \cap \mathcal{R} = \emptyset$ ;
- 3)  $h_l \cap h_r = \emptyset$ ,  $\forall h_l \in \mathcal{H}(\mathcal{L}), \forall h_r \in \mathcal{H}(\mathcal{R})$ ;
- 4)  $0 < x_l < x_2$ ,  $\forall \{x_l, y_l\} \in h_l \in \mathcal{H}(\mathcal{L})$ ;
- 5)  $x_1 < x_r < 1$ ,  $\forall \{x_r, y_r\} \in h_r \in \mathcal{H}(\mathcal{R})$ .

find the two sets  $\{\mathcal{L}^*, \mathcal{R}^*\} \in \{\{\mathcal{L}, \mathcal{R}\}\}$  such that:

$$\max(d(\mathcal{H}(\mathcal{L}^*)), d(\mathcal{H}(\mathcal{R}^*))) = \min_{\mathcal{L}, \mathcal{R}} \{\max(d(\mathcal{H}(\mathcal{L})), d(\mathcal{H}(\mathcal{R})))\}$$

The solution is the shortest path, with the condition that the paths of the two arms do not cross. Since the arms work in parallel, the total time required is the longest time spent by one of the arms.

Theoretically, the 2-TSP can be reduced to several TSP problems, by assigning each combination of targets to one tour, from a minimum of 2 targets up to  $N/2$ , and the remaining to the other tour. In the worst case this requires  $\sum_{i=2}^{\frac{N}{2}} \frac{N!}{(N-i)!}$  TSP computations. All of these TSPs should first be checked for collisions, and the shortest tour which meets the conditions in Problem 2.1 is selected. The approach taken here is to find an approximate

solution which fully complies with the collision avoidance condition which is sufficiently close to the optimal solution.

The algorithm, summarized in Table 1, includes an initialization phase, and an iterative process described in details in the following sections. The input of the algorithm is a set of targets  $\mathcal{T}$ , which is divided according to the horizontal limitation of the two salesmen:

$$\mathcal{O}\{T \in \mathcal{T} : x_1 < x_T < x_2\}; \quad \mathcal{D}\{T \in \mathcal{T} : x_T < x_1 \vee x_T > x_2\};$$

where  $x_T$  is the horizontal component of the vector  $\mathbf{t}$  associated with the target  $T$ .

The 2-TSP works with a fixed number of neurons  $N$ , equal from the start to the number of targets. In most of the prior work based on SOM neural networks, the network is initialized with a small number of neurons (e.g. one neuron in the work of Angéniol (Angéniol et al., 1988)), and the number increases during the search for the best tour. Only Burke (Burke, 1994) (Burke, 1996) has started with a fixed number of neurons that is equal to the number of targets, but has no addition/deletion mechanism.

After an initial setup of the two tours the algorithm is iterated through a process of adaptation and regeneration until all targets are assigned to neurons. Throughout the iteration process the number of neurons never changes, since the creation and deletion of neurons is balanced.

### 2.1 The starting tours

The initialization process should pose in the space all the neurons organized in two initial left and right tours,  $\mathcal{X}^l, \mathcal{X}^r$ , with  $\mathcal{X} = \mathcal{X}^l \cup \mathcal{X}^r$ ,  $\mathcal{X}^l \cap \mathcal{X}^r = \emptyset$ ,  $|\mathcal{X}| = |\mathcal{T}|$ . The general principle for initializing the tours in the 2-TSP would be to lay two polygonal tours with vertex approximating the center of gravity of the targets in some discrete partition of the whole space, and with neurons on the edges approximating the target density in the same partition.

Calling  $A_i$  the areas into which the space related to one tour is partitioned, and  $C$  the closed polygonal interpolating the center of gravity of targets in  $A_i$ , the linear density of neurons on the initial tour,  $\nu(s)$ , shall

---

Input: the set of targets  $\mathcal{T}$ , the limits of the overlap area  $x_1, x_2$

$\mathcal{T}$  is split in the sets  $\mathcal{O}$ ,  $\mathcal{D}$  of targets in the overlap and in the non-overlap area

initialize the tours  $\mathcal{X}^l, \mathcal{X}^r$  with  $|\mathcal{X}^l| + |\mathcal{X}^r| = |\mathcal{T}|$ ,  $\mathcal{U} \leftarrow \mathcal{T}$ , default to *regeneration* mode

---

**while**  $\mathcal{U} \neq \emptyset$

**if** in *adaptation* mode:

    choose randomly a target  $T_j \in \mathcal{U}$

**else** (in *regeneration* mode):

    choose  $T_j \in \mathcal{U}$  with the largest *activity*  $E^T$

  identify one *section* of the tour if  $T_j \in \mathcal{D}$ , or two *sections* if  $T_j \in \mathcal{O}$ , and compute from the candidate set  $\mathcal{C}$  the winner neuron  $X_w \in \mathcal{X}^l \cup \mathcal{X}^r$

**if** in *regeneration* mode:

**if**  $E_w \geq 0$  ( $X_w$  is not yet assigned):

$\mathbf{x}_w \leftarrow \mathbf{t}_w$

**else** ( $X_w$  is already assigned):

      delete the *worst* neuron  $X$ , and add a new neuron near  $X_w$  with  $\mathbf{x} \leftarrow \mathbf{t}_w$

$\mathcal{U} \leftarrow \mathcal{U} - \{T_j\}$

**else** (in *adaptation* mode):

**if**  $X_w$  is a previous winner on  $T_j$  for  $\tau$  times:

$\mathbf{x}_w \leftarrow \mathbf{t}_w$ ,  $\mathcal{U} \leftarrow \mathcal{U} - \{T_j\}$

**else** ( $X_w$  is a (relatively) newly winner on  $T_j$ ):

      change the weights of the winner  $\mathbf{x}_w \leftarrow \mathbf{x}_w + (\mathbf{t}_w - \mathbf{x}_w)\eta$  and of its neighbors with a smaller learning rate

      update the process parameters  $K$  and  $F$ , the *activity*  $E$

      according to the value of parameters  $K, F$ , current  $\eta$ , number of cities with positive *activity*  $E^T$ , choose next process mode: *adaptation* or *regeneration*

---

Output the tours  $\mathcal{X}^l, \mathcal{X}^r$

---

Table 1: The DTSP algorithm

approximate the area density  $\varrho(A)$  of the targets:

$$\int_{C_i} \nu(s) ds = \iint_{A_i} \varrho(A) dA \quad (1)$$

Where  $C_i$  is the section of  $C$  inside  $A_i$ . The choice of the partition can vary in granularity and in geometry. An efficient solution is to divide each of the two subspaces in radial sectors, from the center of gravity of all targets in the subspace, at fixed angular steps.

Because of the weak sensitivity of the algorithm on the initial tour shape, a simpler solution is adopted, based on a tessellation of the space into 16 squares of equal area (4x4). The neurons are initially placed into these squares, where the number of neurons is set to be equal to the number of targets contained in the square. The center of gravity ( $\mathbf{c}_i$ ) of the targets is computed for each of the squares, and piece wise linear loops are connected between the squares (see Figure 1).

More precisely neurons in a square  $i$  are placed over a line segment in which the end points are:

$$\frac{\beta(i-b)\mathbf{c}_i + (1-\beta)\mathbf{c}_b}{1-\beta+\beta(i-b)}; \frac{\beta(f-i)\mathbf{c}_i + (1-\beta)\mathbf{c}_f}{1-\beta+\beta(f-i)}. \quad (2)$$

where  $b$  is the index of the first non-empty square backwards from  $i$ , and  $f$  the index of the first non-empty forwards. When all square are populated with neurons, equation (2) is reduced to  $\beta\mathbf{c}_i + (1-\beta)\mathbf{c}_{i-1}$ ;  $\beta\mathbf{c}_i + (1-\beta)\mathbf{c}_{i+1}$ . If an index is out of the boundary is wrapped circularly.

These segments are connected and the neurons are placed equidistant from each other along the segment within each square. The proportion of segments populated with neurons and joint segments between them is ruled by  $\beta$ . For example, with  $\beta = \frac{2}{3}$ , along the line connecting  $\mathbf{c}_i$  and  $\mathbf{c}_{i+1}$  at distance  $\frac{1}{3}D(\mathbf{c}_i, \mathbf{c}_{i+1})$  from  $\mathbf{c}_i$  there will be the end point of the segment carrying neurons of  $A_i$ . At distance  $\frac{2}{3}D(\mathbf{c}_i, \mathbf{c}_{i+1})$  there will be the starting point the segment carrying neurons of  $A_{i+1}$ . In between there will be a joint segment without neurons. See Figure 1 for an example with 50 targets.

While for a number of conventional TSP heuristics the choice of a good starting tour has an important impact (Perttunen, 1994), experimentation has revealed that for the 2-TSP the initialization is not critical. Even a

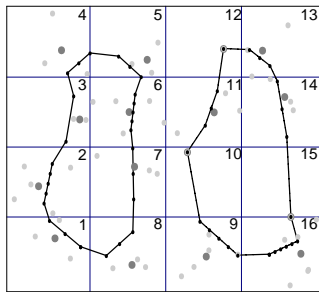


Figure 1: The partition of the problem space into 16 regions, and the shape of the two starting tours. The medium sized dots are the targets (oranges), the larger dots are centroids for each square, and the small dots are the starting locations for the artificial neurons.

very crude initialization, like two square with neurons placed without any relationship with the target distribution, lead to a worsening in time and quality less than 1% on a 1000 target problem, with respect to the solutions just described. The initial assignment of nodes is independent from the size of the mechanical overlap between the two arms, the partition of targets between the two tours will rely on the competition process, and each target initially included in the overlap set  $\mathcal{O}$  will be contended by both tours, in the way described below.

## 2.2 The iterative loop

In the 2-TSP algorithm one iteration step consists of the presentation of a single target  $T_j \in \mathcal{T}$ , the competition for the winner neuron  $X_w \in \mathcal{X}$ , followed by either the adaptation of synaptic weights or by the neural regeneration process.

In each iteration step the target is selected randomly, and the overall likelihood of selecting a target is independent of its membership in the sets  $\mathcal{O}$  or  $\mathcal{D}$ . However, the assignment of common target is more critical, since it is essential to avoid a premature assignment of free neurons in the overlapped area prior to an assessment of the remaining tours. For this reason

there is an explicit alternation in the selection process between distinct targets and common targets:

$$T_{j,n} \in \begin{cases} \mathcal{O}, & \text{when } \mathcal{D} = \emptyset \vee n \bmod \left(1 + \left\lceil \frac{|\mathcal{D}|}{|\mathcal{O}|} \right\rceil\right) = 0 \\ \mathcal{D}, & \text{otherwise} \end{cases} \quad (3)$$

where  $\lceil \cdot \rceil$  is ceiling function.

One conventional practice that reduces the time spent in finding the winner neuron for each target, is to restrict the search to a local neighborhood of the target (e.g. Fritzke & Wilke (Fritzke and Wilke, 1991)). In the present algorithm there is a modified version of this approach, based on shrinking the neighborhood of a target during the process. The set of neurons within the neighborhood of target  $T_j$  on the  $n^{\text{th}}$  iteration is referred to as  $\mathcal{S}_j^{k,n}$ , with  $k \in \{l, r\}$ . If  $T_j \in \mathcal{D}$ , there is only one  $\mathcal{S}_j^{k,n}$  selection, otherwise, if  $T_j \in \mathcal{O}$ , the selection is done for the left tour  $\mathcal{S}_j^{l,n}$  and the right tour  $\mathcal{S}_j^{r,n}$ .

In order to determine the initial definition of  $\mathcal{S}_j^{k,n}$ , the central neuron in the selection is found, using:

$$X = \begin{cases} W(T_j, 1) & \text{if } W(T_j, 1) \neq \not b \\ X_{A_i} & \text{otherwise} \end{cases} \quad (4)$$

where  $\not b$  is the symbol for an undefined winner. If the target has already been selected, there will be a previous winner, given by the function  $W$  defined later in (9), and is used as current center in  $\mathcal{S}_j^{k,n}$ . Otherwise, resorting to the information that  $T_j$  belongs to area  $A_i$  in the partition of the plane, (see Section 2.1), neuron,  $X_{A_i}$  is used.  $X_{A_i}$  is the most central neuron in the segment which endpoints are given by (2).

The neighborhood selection spans  $L$  neurons backwards and forwards from the central neuron along the connected chain of artificial neurons. The neighborhood set of all target neurons is changed in each iteration step. In particular the neighborhood set of the non-selected targets always decreases, while the neighborhood of the selected target may increase. More precisely, the value of  $L_p$  associated to each target  $T_p$ , is updated at each

iteration ( $n$ ) as:

$$\begin{aligned} L_{p,0} &= \delta N, & p=0..N; & \quad [initial\ value\ of\ L] \\ L_{p,n} &= \left(1 - \frac{1}{\delta N}\right) L_{p,n-1}, & p=0..N, p \neq j; & \quad [decreases\ L] \\ L_{j,n} &= \frac{L_{j,n-1} + L_{j,0}}{2}. & & \quad [increases\ L] \end{aligned} \quad (5)$$

with  $\delta < \frac{1}{2}$ . At the beginning of the search process,  $\mathcal{S}_j^{k,n}$  is large, and includes many potential neurons within the neighborhood of each target. As the iteration proceeds, the neurons in the tour move closer to the associated targets, and the search gradually becomes more of a local process. Nevertheless, the neighborhood of a target that has been selected but not assigned by the competition, is enlarged the next time that it is selected ( $L_j$  is increased). Actually, only a subset of the neurons in  $\mathcal{S}_j^{k,n}$  competes to be closest to  $T_j$ , as will be described in Section 2.3.

### 2.3 The competition rule

The orientation of the tour trajectory with respect to the target  $T_j$  is used to select possible candidates winner neurons out of the group defined by  $\mathcal{S}_j$ . There is a set of candidates  $\mathcal{C}_j^p$  for which the distance  $D^p(X_i, T_j)$  from target  $j$  to the candidate neuron  $i$  is measured, and an other set  $\mathcal{C}_j^s$  for which the segment distance  $D^s(\overline{X_i X_{i+1}}, T_j)$  from the segment of the tour connecting two candidate neurons and the target is used.

The two sets are constructed according to the sign changes of the derivative of the distance along the trajectory:

$$\begin{aligned} \mathcal{C}_j^p &\stackrel{\text{def}}{=} \left\{ X_i \in \mathcal{S}_j : \frac{dD}{ds} \Big|_{X_i^+} \geq 0 \wedge \frac{dD}{ds} \Big|_{X_i^-} \leq 0 \right\} \\ \mathcal{C}_j^s &\stackrel{\text{def}}{=} \left\{ X_i \in \mathcal{S}_j : \frac{dD}{ds} \Big|_{X_i^+} \leq 0 \wedge \frac{dD}{ds} \Big|_{X_{i+1}^-} \geq 0 \right\} \end{aligned} \quad (6)$$

As long as the tour trajectory is oriented towards the target the next neuron is closer to the target, and there is no need to compute the distance. It is only necessary to compute the distance when a change in this orientation

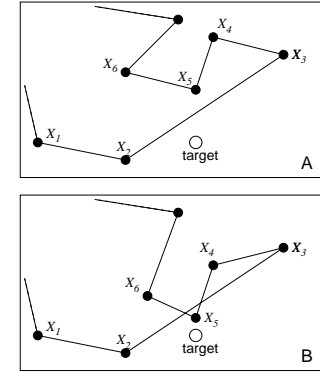


Figure 2: A typical situation when prior competition rules lead to a tangle in the tour (A). Neuron  $X_5$  will win, and move towards the current target, together with  $X_4$  and  $X_6$  (B). With the current algorithm, the winner will be  $X_2$ .

occurs. A change in orientation can occur either at a neuron, or along a part of the trajectory connecting two neurons. In the first case it is more appropriate to use the point to point distance as a score in the competition for the winner neuron. In the second case the distance is measured to the segment, and if the segment is closest than the nearest of the two neurons connected by the segment, it is selected as the winner.

The effect of including the segment distance in the competition is illustrated in an example in Figure 2. In this case if the segment distance is not included in the algorithm the tour becomes longer and tangled. In part A of Figure 2 the closest neuron to the selected target is  $X_5$ , but there is a part of the tour between  $X_2$  and  $X_3$ , that is much closer to the target. However neither  $X_2$  or  $X_3$  is close enough to win. Without including the segment distances,  $X_5$  would win and be moved, together with its neighbors  $X_4$  and  $X_6$ , towards the selected target (Figure 2 B). Instead  $X_2$  is selected as the winner, and is moved towards the target.

This safeguard feature inside the competition is one of the reasons why

this algorithm can use values of the learning rate much higher than in conventional SOM, as will be shown in 3.1, with faster convergence. The speed of the algorithm is further increased by computing the distances only when a neuron or a segment could be a winner in the competition. The orientation of the tour in relation to the target is computed at both endpoints of each segment within the neighborhood  $S_j$  of a target. The segments and neurons that participate in the competition are found by computing the sign of the change in distance to the selected target at the endpoints of the segments. This is most easily calculated by taking the sign of the dot product of the vector from  $X_i$  to  $X_{i+1}$  along the segment of the tour and the vector from an endpoint of the segment to the target:

$$\text{sign} \left( \frac{dD}{ds} \Big|_{X_i^+} \right) \equiv \text{sign} \left( (\mathbf{x}_{i+1} - \mathbf{x}_i)^T (\mathbf{x}_i - \mathbf{t}_j) \right) \quad (7)$$

$$\text{sign} \left( \frac{dD}{ds} \Big|_{X_{i+1}^-} \right) \equiv \text{sign} \left( (\mathbf{x}_{i+1} - \mathbf{x}_i)^T (\mathbf{x}_{i+1} - \mathbf{t}_j) \right)$$

since, for example, the  $\text{sign} \left( (\mathbf{x}_{i+1} - \mathbf{x}_i)^T (\mathbf{x}_i - \mathbf{t}_j) \right)$  is the sign of  $\cos \angle TX_i X_{i+1}$ . Table 2 shows the possible cases that need to be considered in the evaluation of a neuron in the trajectory, in accordance with (6). In the first case the tour is progressing toward the target, so the neuron is not shortlisted. The trajectory is directed away from the target in the second case. In the third case the tour has just passed the selected target, and the distance from the nearest point in the trajectory is used, so  $X_i \in \mathcal{C}_j^s$ . The fourth case is when the trajectory is directed as previously directed toward the target and changes on  $X_i$ , the distance from its position is used:  $X_i \in \mathcal{C}_j^p$ . Note that case  $\frac{dD}{ds} \Big|_{X_i^+} > 0$  and  $\frac{dD}{ds} \Big|_{X_{i+1}^-} < 0$  is physically impossible.

All the  $X_i \in \mathcal{C}_j^p \cup \mathcal{C}_j^s$  are selected for the final competition. In this competition the neuron with the smallest distance  $D$  wins. An example of the role of the segment orientation in the competition process is shown in Figure 3. The sign of the derivatives are shown in brackets, and thin lines show the distances  $D$  taken into account. Note that some of these are point to point distances, and others are segment distances.

This competition process is also part of the mechanism for selecting

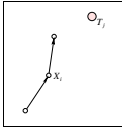
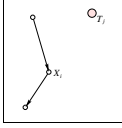
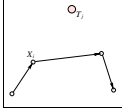
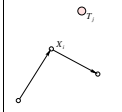
#	$\frac{\partial D}{\partial s} \Big _{X_i^-}$	$\frac{\partial D}{\partial s} \Big _{X_i^+}$	$\frac{\partial D}{\partial s} \Big _{X_{i+1}^-}$	distance type	pattern
1	$\forall$	$\leq 0$	$\leq 0$	none	
2	$\geq 0$	$\geq 0$	$\forall$	none	
3	$\forall$	$\leq 0$	$\geq 0$	$D^s$	
4	$\leq 0$	$\geq 0$	$\forall$	$D^p$	

Table 2: The rules for selecting candidates for the competition to the selected target. Columns 2,3,4 are the inputs to the table, including: the two distance derivatives and the previous status  $S_{i-1}$ . The outputs are the new status  $S_i$ , and the distance used as score in the competition, if any. The last column shows an example of the trajectory configuration treated for each case. A distance is calculated only in cases 2 and 3.

collision free tours. Let the two tours  $\mathcal{X}^l$  and  $\mathcal{X}^r$  be in any state of the competition process, but not yet colliding, and let  $T_j$  be an arbitrary next selected target. A tour intersection cannot take place if  $T_j$  is already included in one of the tours. Figure 4 shows an example in which the present algorithm avoids a collision between the two trajectories that would have occurred without the competition process described above.

There is no absolute means to avoid a collision when the contended target is outside of both tours. The rule described in (6) is applied independently on the trajectory of each tour, and at the end the two results are

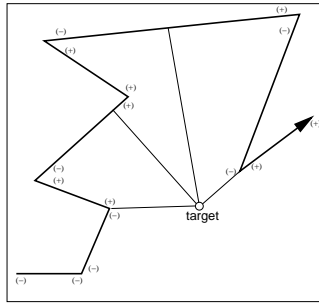


Figure 3: An example of the competition process. This is a part of an example tour, in which the thick solid lines correspond to set segments in the current tour, and there is an artificial neuron (not shown) at each end of each segment. The sign of the distance derivatives with respect the target are shown in brackets for each of the neurons. Following the rules described in (6), the distance to only 4 neurons out of 9 are evaluated. Two of the measurements are to segments of the tour and two of the distances are to artificial neurons. The thin lines show the minimum (perpendicular) distance to the evaluated line segments and to the evaluated neurons.

compared. For an absolute guarantee of collision free tours, it would be necessary to extend the rule to simultaneously take into account the orientation of the two trajectories with respect to the current target. In practice, as discussed in 3.3, the current algorithm is sufficient.

It is interesting to note that collision avoidance is included in the algorithm as a "soft" constraint, without imposing any check or penalty term where crossing of the two tours is mathematically described. This pertains to the self-adaptive nature of the SOM, where it is not required an explicit formulation of the desired objective function, and this is an other advantage of the chosen approach. It is known that the most efficient local heuristics for the TSP, like Lin-Kernighan, are not suitable for accepting additional constraints, but also methods based on explicit list of constraints, like linear and integer programming (Nemhauser and Wolsey, 1988), (Jünger et al., 1995), are limited to modeling linear equalities and inequalities. For the

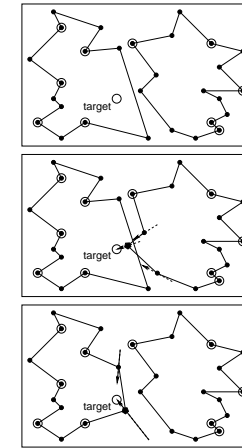


Figure 4: An example of the mechanism for avoiding a collision between the two tours. **A:** A potential initial situation that could lead to a crossing of the two tours. **B:** The modification to the tours that would result using conventional competition rules. A neuron of on right hand tour wins, and is pushed towards the target, crossing the left tour. Its two neighbors are also moved, as shown by the dashed arrows. **C:** The modification of the tours that results using the current DTSP competition rule. The winner is part of the left tour, and it is moved towards the target, together with the unassigned neighboring neuron.

collision avoidance, the mathematical representation of the necessary dis-equality constraints would require a number of binary variables and extra constraints which result in a significant increase in size of the model in terms of variable and constraints. There are several alternatives accepting directly non linear constraints, like special branch-and-bound formulations (Hajian and Richards, 1996), or translators from constraint-logic to mixed-integer programming (Rodosek and Wallace, 1996), which are too expensive in terms of computation for the 2-TSP application.

## 2.4 Neural adaptation

In the 2-TSP algorithm, the state of the neurons is not only reflected in the weight values, there is also an other measure, the *activity* of the neurons, which is updated during the process. The *activity* is an accumulation of winning events for a same neuron, and is a measure of the local mismatch between neuron and target density, which will be exploited during the regeneration phase, as discussed later. The *activity* will grow high for neurons in an area crowded with targets, and remain low where many neurons compete in an area with sparse targets. It is also a kind of energy level of the neuron in the transition from the free state to the state of being assigned to a target. Once the neuron is assigned, it will continue to compete, without the possibility to conquer further targets. Its activity will be still updated, but will have negative sign, being actually a negative potential for all the free neurons competing for same targets.

At the end of the competition at the  $n$ -th iteration, the activity  $E$  of the winner is updated by the following rule:

$$E_{w,n} \leftarrow \begin{cases} E_{w,n-1} + 1 + \alpha \frac{n}{N}, & \text{when } E_{w,n-1} > 0 \\ E_{w,n-1} - 1 - \alpha \frac{n}{N}, & \text{when } E_{w,n-1} < 0 \end{cases} \quad (8)$$

The absolute value given by equation (8) will be motivated in the next subsection, the only effect for the adaptation process is that in the first case  $X_w$  is a free neuron, in the second case is already assigned. The target activity is updated by replacing the previous value:

$$E_{j,n}^T \leftarrow |E_{w,n}|$$

and the winner is registered in the winner history for  $T_j$ , which is a function defined as:

$$W(T, t) : \mathcal{T} \times [0, \tau] \rightarrow \mathcal{X} + \{\emptyset\} \quad (9)$$

$W$  is a function giving the winner for a target at discrete time lags.  $t$  is the delay of drawing on a target, being  $t = 0$  the most recent time that the target has been drawn. Note that  $t$  is not an absolute time metrics, since drawing events are not synchronous for the different targets.  $\emptyset$  is the symbol for indeterminate winner. At iteration  $n = 0$  (at the beginning of the process),

$W = \emptyset, \forall T, \forall t$ .  $\tau$  is the *memory* of  $W$ : the longest delay for which the winner is still recorded for every target.

$W$  is updated as following:

$$\begin{aligned} W(T_j, t) &\leftarrow W(T_j, t-1), \quad t \in [1, \tau] \\ W(T_j, 0) &\leftarrow X_w \end{aligned} \quad (10)$$

Further updating of the activity of a neuron takes place only if the process is not in the regeneration state. If the following condition is met:

$$W(T_j, t) = W(T_j, t+1), \quad t \in [0, \tau-1] \quad (11)$$

$X_w$  is the same winner on the whole history of  $T_j$ , and is accepted as node assigned to the target  $T_j$ :

$$\mathbf{x}_w \leftarrow \mathbf{t}_j, E_w \leftarrow -E_w \quad (12)$$

Otherwise, the winner is updated towards the target, using the current learning rate, in a similar fashion to that used in the Kohonen self-organizing maps (Kohonen, 1984):

$$\mathbf{x}_{w,n} \leftarrow \mathbf{x}_{w,n-1} + \eta_n (\mathbf{t}_j - \mathbf{x}_{w,n-1}) \quad (13)$$

The learning rate  $\eta$  is normally reduced at each iteration by

$$\eta_n \leftarrow \left( \frac{\eta_N}{\eta_0} \right)^{1/N} \eta_{n-1} \quad (14)$$

$\eta_0$  is the value at  $n = 1$ , the parameter  $\eta_N$  is the value  $\eta$  would reach after  $N$  iteration steps in adaptation mode.  $\eta$  is also reduced to half its value each time the process moves into the regeneration state.

The weights of all of the neurons in the neighborhood region  $\mathcal{S}_j$  are updated by an amount that depends on the distance of the neuron along the connected segments from the winner neuron. The amount of movement of the  $i$ th neighboring neuron is given by:

$$\mathbf{x}_{i,n} \leftarrow \mathbf{x}_{i,n-1} + \eta_n \left( \frac{U}{N} \right)^{|w-i|} (\mathbf{x}_{i,n-1} - \mathbf{t}_j) \quad (15)$$

where  $U$  is the number of unassigned targets. The changes are symmetric in the neighborhood in each direction along the tour away from  $X_w$ . This update process is stopped for a particular neuron as soon as the neuron is assigned to a target.

The parameterization of the neural adaptation in equations (8) and (15) normalize the behavior with respect to  $N$ , so that the values of the parameters is almost independent of the size of the 2-TSP problem, and furthermore this stabilizes the relaxation process throughout the search for a tour. The ratio  $(\frac{U}{N})$  in equation (15) for example provides a more global change in the weights when the tour is still in an early stage, and restricts the changes to nearby neurons in the later stages of the search process.

## 2.5 Neuron regeneration

In each iteration of the algorithm, the value of two key parameters are used to determine if the search process is in the state of neural adaptation or neural regeneration. These two parameters are called *frustration* ( $F$ ) and *knowledge* ( $K$ ).

Let define the sets:

$$\mathcal{X}_n^- \stackrel{\text{def}}{=} \{X_{w,n'} : E_{w,n'-1} < 0, n' \in [n_o, n]\} \quad (16)$$

$$\mathcal{X}_n^{\neq} \stackrel{\text{def}}{=} \{X_{w,n'} : X_{w,n'} \neq W(T_{j,n'}, 1), \\ W(T_{j,n'}, 1) \neq \beta, n' \in [n_o, n]\} \quad (17)$$

$$\mathcal{X}_n^= \stackrel{\text{def}}{=} \{X_{w,n'} : X_{w,n'} = W(T_{j,n'}, t), t \in [1, \tau], n' \in [n_o, n]\} \quad (18)$$

$$\mathcal{T}_n^+ \stackrel{\text{def}}{=} \{T_{j,n'} : W(T_{j,n'}, t) = \beta, t \in [1, \tau], n' \in [n_o, n]\} \quad (19)$$

$n_o$  is the last iteration since the process was in adaptation mode: at  $n = 0$ , or immediately after the end of a regeneration state.

During neural adaptation  $F$  and  $K$  are computed as follows:

$$\begin{aligned} F_{n_o} &= 0, \\ F_n &= |\mathcal{X}_n^-| + |\mathcal{X}_n^{\neq}| - \frac{|\mathcal{X}_n^=|}{2}, \\ K_{n_o} &= 0, \end{aligned} \quad (20)$$

$$K_n = |\mathcal{T}_n^+| - |\mathcal{X}_n^=|; \quad (21)$$

The *frustration*  $F$  is increased when the result of the competitions is not progressing the construction of the tours: every time the winner is already assigned to a different target (16), or is different from the previous winner (17). On the contrary, it is decreased by successful neural assignments (18) (see also 11). The *knowledge*  $K$  is simply increased by any competition involving a new target (19), and decreased by an assignment of a neuron to a target. The regeneration state occurs when the following condition is met:

$$F K > \gamma U \quad (22)$$

When the conditions described in equation (22) switches the process to the regeneration state, the competition process continues, but in these iteration steps the target  $T_j$  is not drawn randomly, instead the target with the largest activity  $E_{j,n}^T$  is selected. The competition calculation finds the winning neuron  $X_w$ , and the action performed by the regeneration process depends on the activity  $E_w$  of this neuron.

If  $E_w < 0$ , a real create and delete process occurs:

$$\mathcal{X} \leftarrow \mathcal{X} - \{\tilde{X}\} + \{\hat{X}\} \quad (23)$$

$\tilde{X}$  is the *worst* neuron, using as metric the activation of unassigned neurons:

$$\tilde{X} = \arg \min_{\tilde{X}} \{E_X : E_X \geq 0\} \quad (24)$$

From equation (8),  $E_X$  is basically increased by 1 every time neuron  $X$  is the winner, therefore rule (24) selects the less successful neuron. There is an additional term  $\alpha \frac{n}{N}$  in (8), typically smaller than 1, increasing as the algorithm runs. Due to this term neurons which become inactive earlier will be processed first. For example, if two neurons have been both winners only once, the first to be removed by the regeneration mechanism will be the oldest winner.

If there are more than one neuron with  $E = 0$ ,  $\tilde{X}$  is chosen randomly between such neurons.

$\hat{X}$  is the newly generated neuron, placed in the same tour  $\mathcal{X}^k$  of  $X_w$ , in a position identified by the index  $\hat{i}$  given by:

$$\hat{i} = \begin{cases} w, & \text{when } D(\overline{X_{w-1}X_w}, T_j) > D(\overline{X_wX_{w+1}}, T_j) \\ w + 1, & \text{when } D(\overline{X_{w-1}X_w}, T_j) < D(\overline{X_wX_{w+1}}, T_j) \end{cases} \quad (25)$$

and with weights given by:

$$\hat{\mathbf{x}} = \mathbf{t}_j \quad (26)$$

If the activity of the winning neuron  $E_w$  is non negative then it has not been assigned to a target, and the regeneration process assigns it to the current target, with  $\mathbf{x}_i \leftarrow \mathbf{t}_j$ , and  $E_w \leftarrow 0$ , and no create and delete process takes place.

When the process is in the regeneration state, the condition of equation (22) is not checked, and the next iteration step remains in the regeneration state, causing the next  $T_j$  to be the one with the largest activity  $E_{j,n}^T$ . The normal state of neural adaptation is resumed only when  $\max\{E^T\} \leq 0$ , unless the learning rate is below a fixed threshold  $\eta < \eta_\epsilon$ , with  $\eta_\epsilon > \eta_N$ . This first of these conditions allows the regeneration process to continue until all of the unassigned targets that have been in a prior competition process are assigned. When the latter learning rate condition is met, the neural adaptation never resumes, and the regeneration process continues on all of the remaining targets until the end of the search for the tours.

This last condition is a very efficient shortcut in the final stage of the search process, when the neural convergence is very slow, and the few targets left may take many iterations to attract neurons. The regeneration process leads to assignment of a target to a neuron at every step, and therefore rapidly completes the development of the tour. When the neural adaptation process resumes, frustration  $F$  and knowledge are always set to zero, as in (20), (21).

## 3 Results

### 3.1 Parameter sensitivity

Several of the mathematical expressions in the algorithm include working parameters (2), (5), (8), (9), (14), (22). However, the algorithm works reasonably well in a broad range of most of the parameters, and in general the effort of fine tuning will bring a very marginal improvement in terms of tour quality. The same has been already noted for the starting tour (which actually can be interpreted as an other parameter choice) in Section 2.1. More-

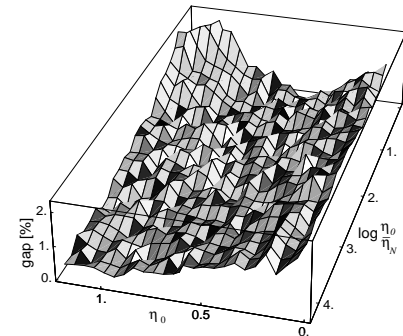


Figure 5: Sensitivity of the DTSP quality to different learning rate parametrization.  $\eta_0$  is the initial learning rate, and  $\frac{\eta_0}{\eta_N}$  is the indication of the span of the learning rate during the iteration.

over, the effect of the parameters is almost independent from the problem size, which is automatically accounted, as in (5), (8), (14), (22).

The learning rate is a classic crucial parameter in neural-based TSP algorithms in balancing the speed of convergence on the final tour and the avoidance of local minima. Kohonen suggest in general a decrease from unity in the initial adaptation phase, and a low value, like 0.02 during the final phase (Kohonen, 1995, p. 88). In (Angéniol et al., 1988) the learning rate is decreased from unity at every iteration using the rule  $\eta_m \leftarrow (1 - C_\eta)\eta_{m-1}$ , where a typical value of the constant  $C_\eta$  is 0.02, the same value is used in (Goldstein, 1990). With this setting on a 1000 target run the average learning rate will be 0.05. In the 2-TSP, the alternative mechanism of regeneration, and the selection of candidates (6), allow for a higher learning rate to be applied without errors due to local minima. For example, with a typical setting, the 2-TSP will apply an average learning rate of 0.5 when processing a 100-targets problem, and an average learning rate of 0.6 for a 1000-targets problem.

Figure 5 shows the performances of the 2-TSP on a 250 target problem, with 20% overlap, at various settings of the learning rate. From equation (14), the learning rate is ruled by its initial value,  $\eta_0$ , and the value decayed

after  $N$  iterations  $\eta_N$ . In the figure  $\eta_0$  is varied in  $[0.1, 1.2]$ , and the ratio  $\frac{\eta_0}{\eta_N}$  spans over 4 decades. The quality of a computed tour is measured as the gap from the best value:  $100(\frac{D}{\hat{D}} - 1)$ , where  $D$  is the total distance traveled by the tours, and  $\hat{D}$  is the minimum  $D$  achieved over all runs, which in this case happens for  $\eta_0 = 0.85$  and  $\eta_N = 0.000114$ . All data points are averages on 1000 runs, since results from one run is always affected by the random sequence of presentation of the targets during the process. As mentioned in Description, there is a constraint placed on the selection of targets from the set  $\mathcal{D}$  or  $\mathcal{O}$  that reduces the likelihood of problematic sequences.

It is evident in the figure that for a broad range of values ruling  $\eta$  the surface of the gap is contained within 1% from the best case.

The most important parameter for the behavior of the 2-TSP is  $\gamma$ , that determines the transition between the regeneration and adaptation states. As is evident in equation (22), with lower values of  $\gamma$  the algorithm easily switches into the regeneration state. This results in a reduction of the time required for the search, but may lead to a premature assignment of targets and less optimal final tours.

Figure 6 shows how the performance of the algorithm depends on changes in the value of  $\gamma$ , using the same problem described for Figure 5. The curve A of Figure 6 confirms that when the neural adaptation is the prevailing process (larger values of  $\gamma$ ), the convergence of the tours towards the targets requires many more iterations. Up to  $\gamma$  equal to 10, most of the targets are actually assigned during the regeneration state, and since each iteration step in this state results in one assignment, there are nearly 250 steps of regeneration. For values  $\gamma > 2000$  the regeneration is almost inhibited, in other words, the algorithm behaves more and more like an ordinary SOM, apart from the competition rule, and the number of iterations grows exponentially. It has to be noted that with a total exclusion of the regeneration mechanism ( $\gamma \rightarrow \infty$ ) also the final convergence will not be ensured any more, as know from the mathematics of the SOM, as well as from experimental results (Jeffries and Niznik, 1994), (Aras et al., 1999).

Convergence of the algorithm with the regeneration is guaranteed, since during this process one city is assigned at each iteration, and in adaptation state, from (14),  $\lim_{i \rightarrow \infty} \eta_i = 0$ . Therefore, for a number of iterations  $i$

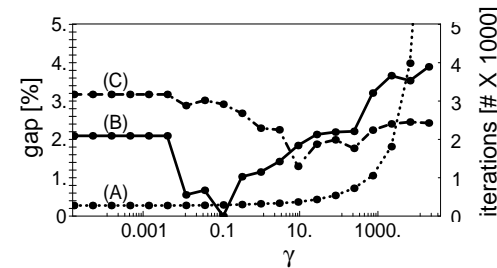


Figure 6: Results from running the DTSP algorithm with different values of  $\gamma$  (x-axis) on 250 targets with 20% overlap of the search area of the two tours. **A:** the number of iterations performed by the algorithm; **B:** The computation time in milliseconds; **C:** the mean difference between the two tours. **D:** the total length of the two tours; **C** and **D** are given as % over the shortest length. Each data point corresponds to the average from 1000 different random tours.

large enough,  $\eta_i < \eta_e$  so that regeneration will never give chances to the adaptation process to resume again.

The curve plotted in part B of Figure 6 shows the overall optimization of the tours, expressed as percentage gap like in Figure 5. The curve in part C is the success of the algorithm in balancing the length of the two tours, measured as  $100 \frac{\sigma_D}{\hat{D}}$ , where  $\sigma_D$  is the deviation between left and right tour lengths, and  $\hat{D}$  the shortest total length over all trials. Both parameters do not vary substantially with  $\gamma$ . Note that the best results lay in an area where the computation time is still very close to the lower limit. Other tests have shown that similar behavior occurs with a larger or smaller number of targets.

Figure 7 shows an example of the progress of the search process. In particular it shows the times at which the algorithm switched between the regeneration and competition states. In this case there are one hundred targets, and  $\gamma$  is equal to two. The plot of  $U$  shows the rate of assignment of the targets, as a function of the number of iteration steps. Whenever the amount of *frustration* and *knowledge* enable the regeneration to take

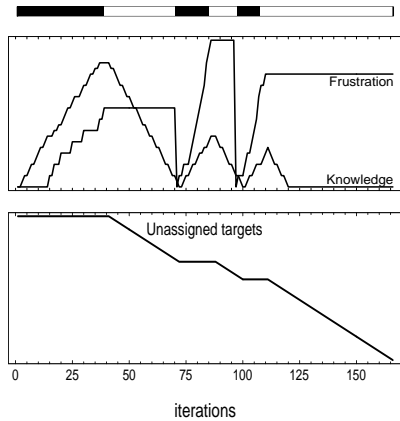


Figure 7: Plot of target assignment during the iteration of the DTSP algorithm. The middle panel shows the values of the two controlling parameters *frustration* and *knowledge*, and the upper panel shows the state of the process.

place (e.g. near step 40), the number of unassigned targets decreases almost linearly. In this case, when  $\gamma$  is two, there is no assignment during the neural adaptation state.

As a result of applying equation (22), in the final part of the search process the regeneration state is more likely than the adaptation state. In the example (after step 120) regeneration is the sole process, and there is a target assignment in each iteration step.

Figure 8 shows the evolution of the 2-TSP algorithm on a large problem. In this case there are 2000 targets and the two tours are completely overlapping. This figure contains the initial configuration, an intermediate step, and the final tours.

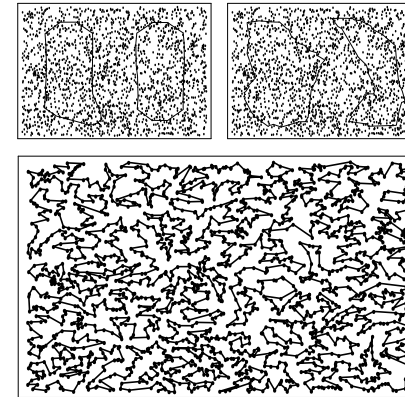


Figure 8: An example DTSP result with 2000 targets and with 100% overlap of the search space of the two arms. The upper two panels contain the starting tour and one of the intermediate steps. The lower panel shows the two final tours.

### 3.2 Performances on double problems

Since the 2-TSP has been designed for a specific real application, it is not easily comparable with respect to all its features with other algorithms, because  $m$ -TSP with additional constraints available in literature are quite different, as mentioned in the introduction, and no one was exactly formulated as for this robotic application. However, it is possible to compare the two most important figures: computation speed and tour lengths, with other  $m$ -TSP, neglecting other features of the algorithms.

A comparison has been done with `goldstein`, an other SOM-based algorithm (Goldstein, 1990), which is a direct extension of the algorithm of Angéniol. When a target  $T_j$  is selected, the distance from a neuron  $X_i$  is weighted by an additional term:

$$D^*(T_j, X_i) = D(T_j, X_i) \left( 1 + \frac{d^{(k)} - \bar{d}}{\bar{d}} \right) \quad (27)$$

where  $d^{(k)}$  is the total current distance of tour  $k \in l, r$ , and  $\bar{d}$  is the mean of the two total distances. In such way neurons on an overloaded tour are less likely to win.

Problem size	goldstein			2-tsp		
	length	dev	time	length	dev	time
100	10.33	0.68	127	10.13	0.08	7
200	13.37	0.27	264	12.08	0.24	34

Table 3: The algorithm compared with an other  $m$ -TSP neural solver in double tour problems. For each algorithm is reported the length, as sum of the two tours, the deviation between tours, and the running time in milliseconds.

Table 3 reports comparisons made on the same two problems used in the original work of Goldstein. The tour quality is slightly better for 2-tsp, but the major difference is in time performances, where `goldstein` is at least an order of magnitude slower. Both algorithm has been run on a 266 MHz Pentium II PC with Linux OS, C code compiled using GNU compiler, O1 level optimization, and timing is measured with the `getrusage` system call. These are the conditions used in all computations of this work.

### 3.3 Collision avoidance

As discussed in Section 2.3, the current algorithm does not absolutely guarantee that the two tours do not cross. Using a simulation the probability that the rules given in (6) result in two tours that cross has been evaluated.

A large number of different tours has been run, with sizes of the problem in the range of interest for the harvesting application, results are shown in Table 4.

Simulations were run using 50% of the total space as overlap area, and several distribution of targets in the overlap and disjointed areas. The case with 50% of targets in the overlap area correspond to the uniform distribution. The number of tours in which the trajectories cross is less then 10 cases over 100,000 runs, for most of the conditions.

The values shown in the table has been computed with a value  $\gamma = 0.1$ , other tests has been done with  $\gamma = 2000$  to check if the regeneration

Problem size	Fraction of targets in the overlap area		
	0.50	0.70	0.90
10	2	12	5
20	7	8	16
30	12	5	4
40	6	6	8
50	6	12	7
60	6	6	3

Table 4: Number of final tours with collision over 100,000 DTSP runs, for various problem size and condition.

process has an influence on the collision avoidance. For such value of  $\gamma$  regeneration is almost inactive, and an inevitable side effect is the increase of computation time, of at least a factor of 1000, so only 10,000 runs has been performed. For most of the cases one single collision occurred, and two collisions where detected for 60 targets and 0.9 overlap. These numbers are too small for a statistical investigation, however it can be stated that the avoidance is mainly due to the competition strategy, and the regeneration has no evident influence. The most important evidence is that collision episodes are very marginal. These trajectory crossings would actually lead to a potential collision if the two arms are simultaneously in the overlapping section of the corresponding tours. A real collision would never occur since there are safety mechanisms in the low level control system of the arms. If a collision condition occurs the robot controller is forced into a non-nominal feedback system, which protects the arms but takes up valuable time that could be spent in the harvesting process. The very low collision probability makes the avoidance function of the algorithm sufficient for the application.

### 3.4 TSP version of the algorithm

The 2-TSP algorithm has been designed with the robotic application in mind, however, its key mechanisms, like the adaptation/regeneration pro-

cess, and the competition based on trajectories, seems to be an enhancement of the SOM paradigm of a more general significance. In order to assess the effectiveness of the 2-TSP solutions, the algorithm has been simplified to solve the classical "one-TSP" problem, where competitors abound. A benchmark set which is getting more and more used in the TSP community is the TSPLIB (Reinelt, 1991), a growing collection of sample instances.

A recent work (Aras et al., 1999) has included, for the first time, a comparison of neural TSP solutions on some TSPLIB benchmarks, therefore runs of the 2-TSP<sup>2</sup> has been performed on the same instances.

Results are given in table 5. All the 14 problems have a known optimum tour, therefore the output is given in terms of gap from the optimum value. The compared algorithms are: the Burke's "guilty-net" `guilty`, the Angéniol's SOM extended by Aras et al. `aras`, the original Angéniol's SOM `ang` and the 2-TSP `2-tsp`. All algorithms has been described in the Introduction, and compared using best results, as in the original work. The `guilty` algorithm has clearly a lower quality output, of about an order of magnitude worst then the other neural approaches. It does not converge for problem `pcb442`, so statistics for this algorithm has been computed on one less sample. The `aras`, `ang` and `2-tsp` are of similar quality, with the best average quality for `aras` (4.00), followed by `2-tsp` (4.79) and `ang` (6.47). For the `2-tsp` the quality is less affected by the specific instance of the problem, as indicated by the standard deviation.

### 3.5 Comparison with non neural solutions

It is well known that neural networks approaches to the TSP, while of theoretical importance in developing and demonstrating capabilities of neural models, are reputed as far of being effective compared to operational research state-of-the-arts. Therefore comparisons with classical operational research methods is a necessary step in assessing progress of neural network approaches.

Figure 9 shows results of running the TSP version of the algorithm 500 times on one of the benchmark problems, **p654**, compared with sev-

<sup>2</sup>For sake of clarity the algorithm of this work will be always called "2-TSP", even when applied to a "single" problems

Problem name	Size [#]	Gap [%]			
		<code>guilty</code>	<code>aras</code>	<code>ang</code>	<code>2-tsp</code>
<code>bier127</code>	127	31.2	2.76	3.71	5.92
<code>ei151</code>	51	10.6	2.82	3.99	3.05
<code>ei176</code>	76	14.1	5.02	6.13	5.76
<code>ei1101</code>	101	22.7	4.61	6.68	6.83
<code>kroA200</code>	200	37.8	5.72	8.50	8.49
<code>lin105</code>	105	7.6	1.98	6.49	4.93
<code>pcb442</code>	442	-	11.07	17.47	11.08
<code>pr107</code>	107	81.7	0.73	1.79	1.93
<code>pr124</code>	124	47.1	0.08	5.12	0.33
<code>pr136</code>	136	40.4	4.53	6.89	2.54
<code>pr152</code>	152	42.8	0.97	1.30	1.17
<code>rat195</code>	195	65.6	12.23	15.41	9.25
<code>rd100</code>	100	10.4	2.10	4.50	4.07
<code>st70</code>	70	12.0	1.48	2.67	1.19
Average		32.6	4.00	6.47	4.79
Standard Deviation		23.0	3.66	4.7	3.27

Table 5: Quality of the TSP-version of the present algorithm compared with other neural algorithms, run on benchmark TSP problems which have known optimum solutions. Quality is measured as % gap from the known optimum.

eral other classical heuristics of the operational research domain (Reinelt, 1994).

The Nearest-Neighbor (**NN**) and Insertion (**IN**) methods successively build a tour according to some construction rules (Rosenkrantz et al., 1977). In the **NN** algorithm, at every step a new node is added by simply choosing the nearest target that is not yet assigned, and the tour is closed at the end by connecting the last and first nodes. The results from an implementation of the **NN** algorithm shown in Figure 9 use an additional heuristic

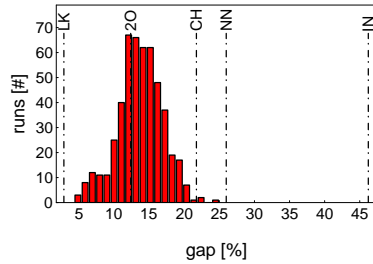


Figure 9: The distribution of TSP tour lengths from 500 runs of the standard problem **p654**. These results are compared with Reinelt’s measurements of the performance of several methods for solving the TSP (Reinelt, 1994) including: **NN**: Nearest neighbor, **IN**: Insertion, **CH**: Christofides, **2O**: 2-Opt, **LK**: Lin-Kernighan.

to avoid a characteristic problem with **NN**. At the end of the search with the **NN** algorithm isolated targets (the so-called *forgotten nodes*) have to be connected at a high cost. The **IN** algorithm starts from a small tour, and at every iteration an edge is broken to insert a new node. In the version of the algorithm presented in the figure the target is inserted whose minimal distance to the current tour is maximal. Moreover, the candidate set is reduced to the targets connected to the tour by a subgraph edge. If this set is empty, an arbitrary target is inserted.

The Christofides (**CH**) heuristics (Christofides, 1976) is one of a number of methods that use a *minimum spanning tree*<sup>3</sup> as a basis for generating tours. In the **CH** algorithm an *Eulerian graph*<sup>4</sup> is built from the minimum spanning tree by connecting every odd-degree node using exactly one edge. Then multiple paths are eliminated to get the final tour. The 2-opt algorithm (**2O**) is a general and widely used improvement principle that consists of eliminating two edges and reconnecting the two resulting paths in a different way. The shortest between the original and the new obtained tour is selected at each iteration. In this algorithm the initial tour must be constructed using a different technique. In its simplest form, the **2O** algo-

<sup>3</sup>the shortest tree connecting all targets and without cycles

<sup>4</sup>a graph connecting all targets, not necessarily with unique paths

gorithm can be just iterated on every node with a certain sequence, until the tour stops improving. This requires a very large and unbounded number of iterations. For this reason, the number of **2O** iteration steps are restricted using some criteria, like on the  $k$ -nearest neighbor subgraph<sup>5</sup>, but this can reduce the quality of the final tour. In the case presented in Figure 9, the starting tour is obtained with **NN**, and the termination of the **2O** algorithm is not restricted.

A more flexible set of heuristics is found in the **LK** algorithm, which was originally developed by Lin & Kernighan (Lin and Kernighan, 1973). This algorithm accepts **2O** moves that temporarily increase the length of the tour. This is clearly a way to avoid local minima, but it may increase considerably the running time. In practical implementations of **LK**, a move is defined as a sequence of simple sub-moves, like **2O**, and the number of sub-moves in a move, and the alternatives for each sub-move, are limited (Mak and Morton, 1993). Despite aged more than two decades, Lin-Kernighan-based algorithms are still the most successful tour-finding approaches (Codenotti et al., 1996), (Johnson and McGeoch, 1995), (Helsgaun, 2000). The solution compared here has moves with a maximum of 15 sub-moves, and up to two alternatives for each of the first three sub-moves. The starting tour was again obtained with the **NN** algorithm.

### 3.6 Time performances

The speed of the algorithm, in the simple TSP version, has been tested and compared with neural and classical solutions, using again the benchmarks of the TSPLIB library, now with a range of problem size up to 13509. The Lin-Kernighan heuristic has been at the heart of many solvers, with different design and performances. The version `linkern` here compared is a state-of-the-art implementation including several enhancements for speed efficiency (Helsgaun, 2000). For the neural solutions, comparison has been made with an improvement of the Durbin-Willshaw “elastic net” algorithm `elnet`, which runs about 15 times faster than the original one (Vakhutinsky and Golden, 1995), and with `ang`. No original time performances are available for `aras`, however from the description of the algorithm we be-

<sup>5</sup>a graph built on the nodes of the tour, by connecting to each target its  $k$ -nearest neighbors

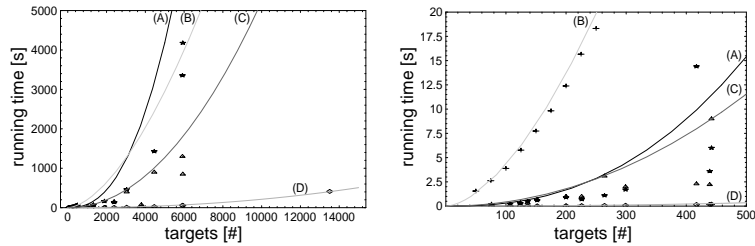


Figure 10: Comparison of time performance on 23 TSP benchmarks for the following algorithms: *linkern*, curve **A** and star marks; *elnet*, curve **B** and cross marks; *ang*, curve **C** and triangle marks; *2-tsp*, curve **D** and diamond marks. The plot on the left spans all the size of the TSP problems, plot on the right is a zoom in the smallest problems, up to 500 targets.

lieve it cannot be faster than *ang*, since it is based on *ang* itself with an added mechanism counteracting the SOM relaxation.

The runs of *elnet* in the original work were computed on a Sparc-10 machine, and has been downscaled by a factor of 3.0, a figure derived by several tests of the *2-tsp* algorithm on the 266 MHz Pentium II and on a Sparc-10. In the original work of Helsgaun *linkern* were run on a 300 MHz G3 Power Macintosh, which is about the same class of the 266 MHz Pentium II, so data has been left unchanged. Angéniol's SOM algorithm *ang* has been run on the same 266 MHz Pentium II machine as the *2-TSP*.

Figure 10 presents the comparison, the left graph spans all the problem sizes, and the right graph is a zoom close to the origin. The data points correspond to the 23 benchmarks: *pr76*, *pr107*, *pr124*, *pr136*, *pr152*, *pr226*, *pr264*, *pr299*, *pr439*, *pr1002*, *pr2392*, *kroA200*, *pcb442*, *r11304*, *r11323*, *r11889*, *pcb3038*, *f1417*, *f13795*, *fn14461*, *r15915*, *r15934*, *usa13509*. For the *elnet* the original data points have been used, 19 random problems of sizes from 50 to 500.

The *2-tsp* is clearly much faster than the compared algorithm in the range of problem sizes used for the evaluation. time required is almost a

function of the sole problem size, while for both *linkern* and *ang* one problem may require much running time than a larger one. It has to be noted that *linkern* is the one giving impressive quality performances, even on the larger problem, with 13509 cities, the gap is 0.008%. For this problem the gap of the *2-tsp* was of 15%, solved in 6 minutes against the 13 hours of *linkern*. Gap of *ang* are of the same order of magnitude of the *2-tsp*, in accordance with the previous analysis, while computation time is quite larger. For *2-tsp* the running time is a function of the problem size, and is weakly affected by the problem typology. On the contrary, for both *linkern* and *ang* one problem may require much more running time than a larger depending on the target typology.

The curves are fitted with  $N^a$  functions, in order to express the asymptotic time complexity of the algorithms. Values of  $a$ , for *elnet*, *linkern*, *ang* and *2-tsp* are of 1.7, 2.4, 2.0 and 2.1. This complexity would suggest that *elnet* will perform better on larger problems (as reported by the authors), however it should be noted that data for this algorithm are only available up to 500 cities, and it is not known if a similar figure will be maintained over a wider range. Also the *2-tsp* exhibits a lower complexity when fitting data in a limited range of sizes, for example up to 4000 cities the fit is  $N^{1.8}$ . In the regeneration process, it is possible to compute the theoretical complexity, since one target is assigned in each iteration, and during each iteration a number of scans proportional to the total number of targets is performed. Therefore, total time complexity becomes  $O(N^2)$ . There are a couple of mechanisms reducing this figure: the rule (6) selecting candidates, and the reduction of the scan section during the process (5), explaining the measured complexity less than quadratic in the range up to 4000 targets. During the adaptation process, complexity cannot be computed theoretically, since it is not only dependent on the problem size, and this portion of the overall process accounts for the higher measured complexity in the full range of problem sizes. Also for *linkern* the theoretical assessment of time complexity is difficult, a typical reported value is of  $O(N^{2.2})$  (Helsgaun, 2000). However, taking for granted the above complexity figures over a wide range of sizes, *elnet* would outperform *ang* on problems larger than 50000, and *2-tsp* only for problems with more than  $3 \times 10^7$  cities.

The *2-tsp* is clearly much faster than the compared algorithm, and

the time required is almost a function of the sole problem size, while for both `linkern` and `ang` one problem may require much running time than a larger one. It has to be noted that `linkern` is the one giving impressive quality performances, even on the larger problem, with 13509 cities, the gap is 0.008%. For this problem the gap of the `2-tsp` was of 15%, solved in 6 minutes against the 13 hours of `linkern`. Gap of `ang` are of the same order of magnitude of the `2-tsp`, in accordance with the previous analysis, while computation time is quite larger.

## 4 Conclusions

The 2-TSP algorithm described here, based on Neural Network concepts, proved to be a reliable yet simple solution to the 2-TSP problem. In the case of the harvesting robot application, the algorithm satisfies the real constraints of the working space and collision avoidance of the two arms. Thanks to its simplicity and limited memory requirements it is suitable for implementation on embedded real-time hardware, without the need of powerful general purpose computer hardware.

The evaluation of the algorithm revealed that there was no substantial dependence on the particular values of the parameters of the algorithm, and no significant discrepancy between the speed of the algorithm and the optimality of the resulting tours.

The mechanics of the harvesting robot, with two telescopic arms in spherical coordinates, allows a simple projection of the path optimization problem in a 2-D Euclidean space, which has been used all during the algorithm description. Multi-arm robots with different joint geometry in general requires higher dimensional space for representing their path planning, especially with respect to collision avoidance. Furthermore, time optimization in robotics involves joint dynamics as well, requiring non Euclidean space representations, typically Riemannian (Shin and McKay, 1985). While in principle the algorithm can be directly extended to any space representation, some of its features will certainly not work any more (like the collision avoidance), and its overall efficiency may not be as valid as in Euclidean 2-D space.

While the 2-TSP algorithm was intended initially for the harvesting ap-

plication, a TSP variant of the algorithm performed well in comparison with the prior solutions to the TSP problem, especially as a quick finder of tours where very high quality is not the main requirement.

In particular, several concepts of the 2-TSP algorithm, like combining neural adaptation and regeneration, or taking into account the trajectory distance in the neural competition, are very general in nature, and proved to considerably enhance the SOM paradigm, which appears to be the most promising neural strategy for solving TSP-class problems.

## References

- Angéniol, B., Vaubois, G. d. I. C., and Texier, J.-Y. L. (1988). Self-organizing feature maps and the travelling salesman problem. *Networks*, 1:289–293.
- Aras, N., Oommen, B. J., and Altinel, I. K. (1999). The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem. *Neural Networks*, 12:1273–1284.
- Boeres, M., Carvalho, L., and Barbosa, V. (1992). A faster elastic-net algorithm for the travelling salesman problem. In *Proc. IJCNN-92*, Baltimore.
- Burke, L. (1994). Neural methods for the traveling salesman problem: insights from operations research. *Neural Networks*, 7(4):532–541.
- Burke, L. (1996). Conscientious neural nets for tour construction in the traveling salesman problem: the vigilant net. *Computers and Operations Research*, 23(2):121–129.
- Burke, L. and Damany, P. (1992). The guilty net for the traveling salesman problem. *Computers and Operations Research*, 19:255–265.
- Christofides, N. (1976). Worst case analysis of a new heuristic for the travelling salesman problem. Research report, Carnegie-Mellon University, Pittsburgh.

- Codenotti, B., Manzini, G., Margara, L., and Resta, G. (1996). Perturbation: an efficient technique for the solution of very large instances of the euclidean tsp. *INFORMS Journal on Computing*, 8(2):125–133.
- Dell, F. R., Batta, R., and Karwan, M. H. (1996). The multiple vehicle TSP with time windows and equity constraints over a multiple day horizon. *Transportation Science*, 23(2):121–129.
- Desrochers, M. J., Desrosiers, J., and Solomon, M. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *Operations Research*, 40:342–354.
- Durbin, R. and Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691.
- Erwin, E., Obermayer, K., and Schulten, K. (1992). Self-organizing maps: Ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55.
- Fisher, M. L. (1995). Vehicle routing. In Ball, M., Magnanti, T. L., Momma, C. L., and Nemhauser, G. L., editors, *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam.
- Fort, J. C. (1988). Solving a combinatorial problem via self-organizing process: an application of Kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, 59:33–40.
- Fritzke, B. and Wilke, P. (1991). A neural network for the travelling salesman problem with linear time and space complexity. In *Proc. IJCNN-91*, Singapore.
- Golden, B. L. and Assad, A. A. (1988). *Vehicle Routing: Methods and Studies*. North Holland, New York.
- Goldstein, M. (1990). Self organizing feature maps for the multiple travelling salesmen problem. In *Proc. INNC '90*, Paris.
- Hachicha, M., Hodgson, M., Laporte, G., and Semet, F. A. (2000). Heuristics for the multi-vehicle covering tour problem. *Computers and Operations Research*, 27:29–42.

- Hajian, M. T. and Richards, E. B. (1996). Introduction of a new class of variables to discrete and integer programming problems. In *Proc Intern. Symposium on Combinatorial Optimization*, London.
- Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130.
- Heuter, G. J. (1988). Solution of the travelling salesman problem with an adaptive ring. In *Proceedings of the IEEE International Conference on Neural Networks*.
- Hodgson, M., Laporte, G., and Semet, F. A. (1998). A covering tour model for planning mobile health care facilities in Suhum district, Ghana. *Journal of Regional Science*, 38:621–638.
- Hopfield, J. J. and Tank, D. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, 4:141–152.
- Jeffries, C. and Niznik, T. (1994). Easing the conscience of the guilty net. *Computers and Operations Research*, 21(4):961–968.
- Johnson, D. S. and McGeoch, L. A. (1995). The traveling salesman problem: a case study in local optimization. In Aarts, L. E. H. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimization*. Wiley, New York.
- Jünger, M., Reinelt, G., and Rinaldi, G. (1995). The traveling salesman problem. In Ball, M., Magnanti, T. L., Momma, C. L., and Nemhauser, G. L., editors, *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam.
- Kagmar-Parsi, B. and Kagmar-Parsi, B. (1992). Dynamical stability and parameter selection in neural optimization. In *Proc. IJCNN-92*, Baltimore.
- Kohonen, T. (1984). *Self-organization and associative memory*. Springer-Verlag, Berlin.

- Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78:74–90.
- Kohonen, T. (1995). *Self-organizing maps*. Springer-Verlag, Berlin.
- Lai, W. K. and Coghill, G. C. (1992). Genetic breeding of control parameters for the Hopfield-Tank neural net. In *Proc. IJCNN-92*, Baltimore.
- Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358.
- Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. (1985). *The travelling salesman problem: a guided tour of combinatorial optimization*. Wiley, Chichester.
- Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21:498–516.
- Lo, J. T. H. (1992). A new approach to global optimization and its application to neural networks. In *Proc. IJCNN-92*, Baltimore.
- Mak, K. T. and Morton, A. J. (1993). A modified Lin-Kernighan for the travelling salesman heuristic. *Operations Research letters*, 13:127–132.
- Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley, Chichester.
- Perttunen, J. (1994). The vehicle routing problem: An overview of exact and approximate algorithms. *Journal of the Operational Research Society*, 45:1131–1140.
- Plebe, A. and Grasso, G. (in press). Localization of spherical fruits for robotic harvesting. *Machine Vision and Applications*.
- Potvin, J. Y. (1993). The travelling salesman problem: a neural network perspective. *ORSA Journal on Computing*, 5:328–348.

- Recce, M., Taylor, J., Plebe, A., and Tropiano, G. (1996). Vision and neural control for orange harvesting robot. In *International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*, Venice, Italy. IEEE Computer Society Press.
- Reinelt, G. (1991). Tspplib - a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384.
- Reinelt, G. (1994). *The Traveling Salesman*. Springer-Verlag, Berlin.
- Rodosek, R. and Wallace, M. (1996). Translating CLP(R) programs to mixed integer programs, a first step towards integrating two programming paradigms. In *Proc Intern. Symposium on Combinatorial Optimization*, London.
- Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M. (1977). An analysis of several heuristics for the travelling salesman problem. *SIAM Journal of Computing*, 6:563–581.
- Shin, K. G. and Mckay, N. D. (1985). minimum time control of robotic manipulator with geometric path constraints. *IEEE Transactions on Automatic Control*, 31(6):532–541.
- Smith, K. (1996). An argument for abandoning the traveling salesman problem as a neural-network benchmark. *IEEE Transactions on Neural Networks*, 7(6):1542–1544.
- Thangiah, S. R., Osman, I. H., Vinayagamoorthy, R., and Sun, T. (1993). Algorithms for the vehicle routing problems with time deadlines. *American Journal of Mathematical and Management Sciences*, 13:323–355.
- Vakhutinsky, A. I. and Golden, B. (1995). A hierarchical strategy for solving traveling salesman problem using elastic nets. *Journal of Heuristics*, 2:67–76.
- Van den Bout, D. E. and Miller, T. K. (1989). Improving the performance of the Hopfield-Tank neural network through normalization and annealing. *Biological Cybernetics*, 62:129–139.