A Kohonen-like Decomposition Method for the Euclidean Traveling Salesman Problem—KNIES_DECOMPOSE[*]

Necati Aras, İ.d.e.a. A.Ş., İstanbul, Türkiye
İ. Kuban Altınel, Ind. Eng. Dept., Boğaziçi Univ., İstanbul, Türkiye
John Oommen, Sch. of Comp. Sci., Carleton Univ., Canada

November 22, 2000

Abstract

In addition to the classical heuristic algorithms of operations research there have also been several approaches based on artificial neural networks for solving the traveling salesman problem. Their efficiency, however, decreases as the problem size (number of cities) increases. A technique to reduce the complexity of a large-scale TSP instance is to decompose or partition it into smaller subproblems. In this paper we introduce an all-neural decomposition heuristic that is based on a recent self-organizing map called KNIES, which has been successfully implemented for solving both the Euclidean traveling salesman problem (TSP) and the Euclidean Hamiltonian path problem (HPP). Our solution for the TSP proceeds by solving the HPP for the subproblems, and then patching these solutions together. No such solution has ever been reported.

Keywords : Combinatorial Optimization, Decomposition, Kohonen, Neural networks, Self-organizing maps, Euclidean traveling salesman problem.

1 Introduction

Of all the families of neural networks described in the literature, the Kohonen self-organizing neural network has been one of the most widely investigated one [24, 25]. This is not surprising, given its simplicity and the wide variety of problem areas to which it may be applied. The beauty of the self-organizing map (SOM) is the fact that the individual neurons adaptively tend to learn the properties of the underlying distribution of the space in which they operate. Additionally, they also tend to learn their places topologically. This feature is

[*]The computational experiments of this research were realized on the hardware donated to the Department of Industrial Engineering at Boğaziçi University by Hewlett–Packard. The first and second authors are partially supported by the Boğaziçi Research Fund Grant 98A301D. The third author is partially supported by the National Science and Engineering Research Council of Canada (NSERC). John Oommen is a Senior Member of IEEE. A preliminary version of this paper was presented in Berlin at ECAI'2000, and can be found in the Proceedings.

particularly important for problems which involve two and three-dimensional physical spaces, and is indeed, the principal motivation for the SOM being used in solving the Euclidean traveling salesman problem (TSP) [29, 33] and its close cousin the Euclidean Hamiltonian path problem (HPP).

The decomposition into subproblems is a widely used strategy in solving large mathematical programs. It is easier to solve the subproblems because the size of each subproblem is much smaller, and their solutions can usually be combined to approximate the solution of the original problem. The idea of decomposition is also applied to the TSP, which is one of the oldest "hard nuts" of operations research and mathematical programing. It is known to be *NP*-complete [28]. There are many exact and heuristic algorithms in the literature for the TSP [26, 31]. In addition to the classical heuristic algorithms of operations research, there have also been several methods based on artificial neural networks which solve the TSP [29, 33]. To this collection we now add a new all-neural decomposition heuristic, which we call KNIES_DECOMPOSE. The new method is based on a recent self-organizing map called the Kohonen Network Incorporating Explicit Statistics (KNIES) [4]. KNIES has been successfully implemented to solve both the TSP (KNIES_TSP) [5] and HPP (KNIES_HPP) [1]. The results which we have obtained using the new KNIES-based decomposition heuristic KNIES_DECOMPOSE are very good—it has been tested for instances obtained from TSPLIB problem library [30]. We believe that our present results are among the most accurate of all reported neural network schemes reported in the literature. Besides the increased accuracy, the decomposition strategy is also very fast. Our solution for the TSP proceeds by solving the HPP for the subproblems and then patching these solutions together. No such solution has ever been reported.

In the next section an overview of major Kohonen type neural approaches to solve the TSP are given along with KNIES_TSP. Section 3 summarizes neural network methods solving HPP, including KNIES_HPP. The new all-neural decomposition heuristic KNIES_DECOMPOSE is explained in Section 4 and Section 5, and experimental results are reported in Section 6. Section 7 concludes the paper.

2 Kohonen Type Neural Solutions to the TSP

The best neural solutions to the TSP proposed ever since the very first successful attempt of Hopfield and Tank [19], (which initiated much of the excitement for the neural network approaches to optimization [12]), use the basic principles of Kohonen's Vector Quantization (VQ) and the SOM [24, 25]. In order to place our current work in the right perspective we shall briefly review some of these methods.

2.1 Pure Kohonen Network Solutions

The self-organizing map of Kohonen in its virgin form has been used to solve the TSP [16, 20, 32]. The main idea is as follows. The network is fully specified by a fixed number of neurons, M, and the input to the network are the coordinates of the cities, say $\mathcal{P}$. These cities are mapped onto M neurons which are located on a ring, where the ordering on the ring represents the traversal of the cities. Observe that by virtue of the SOM, this mapping preserves the neighborhood relationships among the cities and also reflects this topological information.

The cities are first arranged in a random order and presented at each epoch in this order to the network. At every instant one city $\mathbf{X}_i$ from $\mathcal{P}$ is presented to the network. The neurons now compete and the closest one to $\mathbf{X}_i$, $\mathbf{Y}_{j*}(t)$, is determined. The coordinates of this neuron and a group of neurons within its *bubble of activity*, $B_{j*}(t)$, are moved in the direction of $\mathbf{X}_i$ according to the following equation:

$$\mathbf{Y}_j(t+1) = \begin{cases} \mathbf{Y}_j(t) + \Lambda(j, j^*)(\mathbf{X}_i - \mathbf{Y}_j(t)) & \text{if } j \in B_{j*}(t) \\ \mathbf{Y}_j(t) & \text{otherwise} \end{cases}, \tag{1}$$

The set $B_{j*}(t)$ may consist of a subset of neurons from the neighborhood of $\mathbf{Y}_{j*}(t)$. These neurons are given equal weights during the updates while those which are not contained in this subset have zero weights (not migrated at all). Another possibility for $B_{j*}(t)$ is the inclusion of all neurons. In this case neurons are given different weights by using a (Gaussian) kernel function. Experimentally, the former strategy has proven to be quite poor and so the use of a weighting function is generally chosen to be the scheme by which the neurons are moved towards the presented city.

The Kohonen Network in its original form performs poorly for the TSP. First of all, the scheme is not guaranteed to converge. Furthermore, even when it does, the results obtained are quite suboptimal. If the variance of the Gaussian kernel is large the convergence is slower. If, on the other hand, the variance is made small, the algorithm may not converge (because in the limit the presented city affects only a single neuron) and even if it does, the quality of the tour is far from optimal. These drawbacks have motivated other researchers to consider improvements on the scheme.

2.2 The Guilty Net

The main drawback of using the SOM for the TSP is the fact that the scheme changes the weights based only on the distance of the currently examined city to the various neurons on the ring. In [9], Burke and Damany updated the above strategy in their "guilty net" algorithm, by introducing two modifications. The first modification involves incorporating a penalty term to the distance function for determining the winning neuron. The second modification involves specifying an alternate method for the neighborhood function which determines the strength by which the weights of the neighbor neurons are modified.

Just like in the pure Kohonen scheme, the number of neurons on the ring is fixed (Burke and Damany recommend that it be equal to the number of cities N) and remains the same throughout the algorithm [9]. The neurons on the ring are indexed from 1 to M, where $M = N$, and the distance between a neuron and its two immediate neighbors is equal to one. The guilty net tries to avoid the mapping of more than one city on the same neuron by using the idea of a "conscience" introduced by DeSieno [13]. This mechanism gives a chance to other neurons by inhibiting those which win too often. To achieve such an inhibition, the winning neuron is selected according to the following formula:

$$j^* = \operatorname*{argmin}_j \left\{ \|\mathbf{X}_i - \mathbf{Y}_j\| + \lambda \cdot \frac{win_j}{1 + \sum_k win_k} \right\}, \tag{2}$$

where win_j is the number of wins for neuron j, and λ is a penalty factor, a parameter which assumes real values. Notice now that the winner is not the one which is closest to $\mathbf{X}_i$—the criterion to select the winning neuron must also take into consideration the number of times a particular neuron has won, and this combined effect tends to distribute the winning position among the others. The neighborhood function in this method is defined as:

$$\Lambda\left(j, j^*\right) = \begin{cases} \eta\left(1 - d_{jj^*}/L\right)^{\beta} & \text{if } d_{jj^*} < L \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $d_{jj^*} = \min\left\{\left|j - j^*\right|, M - \left|j - j^*\right|\right\}$ and $L = N/2$. Notice that this neighborhood function is identical to the Gaussian kernel except that, from a probabilistic point of view, it is more of a Geometric type—the indices of the neurons instead of their coordinates are used in determining the proximity of the neurons within a neighborhood. It should be noted that $\Lambda\left(j, j^*\right)$ is equal to unity for the winning neuron j^*, and its value decreases as d_{jj^*} increases during one cycle or epoch (the complete presentation of all cities). Furthermore, the updates of the weight vectors of the neighboring units are reduced over time by increasing the value of β at each cycle. This means that the neurons have a large neighborhood at the beginning of the procedure, and this gets smaller from epoch to epoch.

Reported computational results indicate that the guilty net approach performs better than the Hopfield-Tank model [19]. Our experience shows that the scheme is not too accurate, primarily because the number of neurons is not allowed to increase or decrease (due to the absence of insertion/deletion operations) which other methods utilize [2]. When λ is small, convergence is not guaranteed. However, in contrast, when λ increases the quality of the tour deteriorates, because it is no longer a simple tour—the edges (which may be distant from each other) tend to cross each other. Experimentally, the scheme is inferior to the scheme due to Angéniol et al. [2].

In her more recent works, Burke [10, 11] has extended the guilty net to automate the separation of neurons in order to guarantee the convergence. This new network is called the "vigilant net" and uses a simpler measure inspired by the vigilance of the adaptive resonance theory. In adaptive resonance theory, neuron j^* is selected as the winner upon presenting an input to the network if it satisfies two criteria: it must be the closest neuron to the input and must be sufficiently close to it. Namely, the output

$$O_{j^*} = \begin{cases} 1 & \text{if } \left\|\mathbf{X}_i - \mathbf{Y}_{j^*}\right\| = \min_j \left\|\mathbf{X}_i - \mathbf{Y}_j\right\| \text{ and } \left\|\mathbf{X}_i - \mathbf{Y}_{j^*}\right\| < \zeta \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where ζ is the vigilance parameter. Winning neurons are now determined according to a vigilance level. In the vigilance net the ratio between the costs of the longest and the average links in the TSP tour has to be within some desirable interval.

2.3 The Procedure of Angéniol, Vaubois, and Le Texier

The procedure introduced by Angéniol et al. [2] is different from the guilty net in two aspects. The first difference is that the weight update equation does not have the fractional learning rate parameter η. The neighborhood function is the Gaussian kernel defined as:

$$\Lambda(j, j^*) = \frac{1}{\sqrt{2}} \exp\left(-\frac{d_{jj^*}^2}{\sigma^2}\right) \tag{5}$$

where σ (the standard deviation) is called the "gain parameter" and d_{jj^*} is defined as before. The value of σ is decreased at the end of each complete pass by a factor K_σ (a user-defined constant) i.e.,

$$\sigma \leftarrow K_\sigma \sigma, \qquad 0 < K_\sigma < 1. \tag{6}$$

When $\sigma \to \infty$, neurons on the ring move toward the city being presented with the same strength $1/\sqrt{2}$. On the other hand, as σ decreases only neurons closer to winning neuron j^* tend to move toward the city. Furthermore, as σ decreases the intensity with which the neurons move towards the city decreases. Thus the decrement of σ has the dual effect of decreasing the window size and of decrementing α, the adaptation parameter of the SOM. Typically, the algorithm begins with a high initial value of σ so that the neurons make large moves during the initial epochs. Afterwards, as σ becomes sufficiently low, the neurons on the ring settle down. It is to be noted that the only parameters that are to be adjusted are the initial value of σ, and K_σ.

The second and primary difference between this approach and the guilty net is the fact that this scheme permits creation and deletion of neurons as the algorithm proceeds. If a neuron wins the competition for two different cities in the same epoch, then a new neuron is created with the same coordinates as the winner one. This newly created neuron is inserted into the ring as the neighbor of the winner. Both of them are inhibited for the rest of the current epoch, which guarantees that they will disperse by the movements of the neighboring neurons. In the same way, a neuron is deleted if it does not win a competition during three epochs. The algorithm starts with only one neuron (located at the origin) in the Euclidean plane. Experimentally, the number of neurons created turns out to be less than twice the number of cities, but the algorithm terminates when the number of neurons is exactly equal to the number of cities, and when each neuron is associated with a single city.

With this model, Angéniol et al. solved the 30-city problem taken from Hopfield and Tank [19] and the five sets of 50-city problems taken from Durbin and Willshaw [15]. According to their simulation results, the algorithm found solutions within 3 per cent of the optimum for the first data set. The results for the five sets of 50 cities show that, on the average, this approach performs as good as the elastic net of Durbin and Willshaw, but is significantly faster.

There are also other approaches based on self-organizing maps in the literature which are similar to the approaches explained above. For a good review of them the reader is referred to Section 4 of [29], to Section 7.2 of [22], and to [33]. In addition, it is worth mentioning the recent works by Budinich [7], and by Budinich and Rosario [8]; their neighborhood function considers not only the distances between the neurons on the ring (i.e., unlike the d_{jj*} of formula (5)) but also their physical distances (i.e., a function of the coordinates, such as the Euclidean distance).

2.4 The Kohonen Network Incorporating Explicit Statistics (KNIES)

Although the overall effects of using the self-organizing map is that the neurons ultimately obey the distribution of the original points, there is unfortunately a lot of information that it does not use. Indeed, in the effort to preserve the "non-parametric" form of the underlying algorithm the SOM completely ignores the statistical information that is already *collectively* resident in the data points. For example, as soon as $\mathcal{P}$, the set of data points is known, all the associated statistics of the points such as the mean, covariance etc. can be computed quickly. Thus we can conceive of enhanced SOM methods which, apart from only using the information in the data point presented, uses this information in conjunction with these pieces of statistical information. In other words, at every iteration, we seek a strategy by which we will utilize both the local information present in the point presented and also the global information latent in the entire data set, $\mathcal{P}$. How this is achieved is described below.

Let us suppose that the mean of the set of points in $\mathcal{P}$ to be represented by the neurons is $\boldsymbol{\mu}$. The intention is that we maintain $\boldsymbol{\mu}$ to be invariant for the entire training. The question which remains is merely one of determining how the neurons can be collectively updated so as to retain their mean to be also $\boldsymbol{\mu}$. To do this every training step is decomposed into two phases, namely the *attracting* and *dispersing* phases.

2.4.1 The Attracting Phase

When the data point P_i is presented to the network, the closest neuron and all the points within the bubble of activity (the neighborhood of the winner) are migrated using an equation identical to Equation (5). This constitutes the attracting module, and in this phase the neurons within the bubble use the local information present in the points as they appear individually, and move towards them.

Observe that since the neurons are being partitioned as those which participate in the attracting phase and those which do not, in KNIES it is assumed that the bubble of activity is maintained constant for a specified value of M. The question of decreasing the effect of the attraction as the neuron $j \in B_{j*}(t)$ is more distant from j^* must necessarily be achieved by using a kernel function. For the sake of simplicity the kernel is selected as the Gaussian kernel [14, 17] given by:

$$\Lambda(a,b) = K_g \cdot e^{\frac{-\|Y_a - Y_b\|^2}{\sigma^2}}, \tag{7}$$

where K_g is a normalizing constant, σ specifies the standard deviation of the kernel which is progressively decremented (for example, linearly [2]) to get the effect of reducing the bubble of activity, and $\mathbf{Y}_a$ and $\mathbf{Y}_b$ are the coordinates of the points in question. Thus the attraction phase obeys:

$$\mathbf{Y}_j(t+1) = \begin{cases} \mathbf{Y}_j(t) + \Lambda(j,j^*)(\mathbf{X}_i - \mathbf{Y}_j(t)) & \text{if } j \in B_{j*}(t) \\ \mathbf{Y}_j(t) & \text{otherwise} \end{cases}, \tag{8}$$

where $\mathbf{X}_i$ is the input vector presented to the system and j^* is the winning neuron closest (using the Euclidean distance) to $\mathbf{X}_i$.

2.4.2 The Dispering Phase

As a result of the migration of the neurons within the bubble of activity it is observed that the mean of all the neurons moves away from the initially assigned value, $\boldsymbol{\mu}$. Thus, the dispersing phase now forces the neurons outside the bubble to migrate *away* from their current locations to render the mean of the neurons to continue to be invariant. This is done as follows. The total migration that has been affected as a result of the attraction module is the vector $\Delta\mathbf{Y}(t)$, where,

$$\Delta\mathbf{Y}(t) = \sum_{j\in B_{j*}(t)} \left[\mathbf{Y}_j(t+1) - \mathbf{Y}_j(t)\right]. \tag{9}$$

This change $\Delta\mathbf{Y}(t)$, is distributed among the neurons which did not participate in the attraction as:

$$\mathbf{Y}_j(t+1) = \mathbf{Y}_j(t) - \frac{1}{M(t)-\left|B_{j^*}(t)\right|}\Delta\mathbf{Y}(t), \quad \text{if } j \notin B_{j*}(t). \tag{10}$$

Notice that in (10) the change $\Delta\mathbf{Y}(t)$ can also be distributed among the neurons in a non-uniform manner—for example by using a second weighting kernel function for the dispersing neurons. However, in the interest of simplicity (since there is no such neuron as the loser "neuron") the change is distributed uniformly. It is easy to see that as a result of both (8) and (10) the attracted neurons move towards the point $\mathbf{X}_i$, and the net result of both the phases retains the mean of the neurons to be $\boldsymbol{\mu}$, as desired.

2.4.3 KNIES_TSP

Although KNIES is philosophically superior to the traditional SOM, the additional effort used in maintaining the statistics is generally not useful for most problems. The reason for this is quite straightforward: the advantage that is gained is purely in terms of the rate of convergence of the neurons. Thus by enhancing the rate of convergence nothing drastic is achieved, unless the quality of the final solution of the underlying problem itself depends on the intermediate distribution of the neurons. For most problems this is not the case, and so, it is not anticipated that the KNIES will be more effective in solving problems than the traditional SOM. However, there are some problems where the intermediate location of the neurons can cause a difference in the final solution and one such problem is the Euclidean traveling salesman problem (TSP). Aras et al. [5] make use of this property as described below.

First of all it is assumed that the number of neurons at any time is $M(t)$, and that this number can increase or decrease by the insertion or deletion of neurons. At any instant 't' we say that a neuron j located at $\mathbf{Y}_j(t)$ is associated with a city $\Gamma_j(t)$ if:

$$\Gamma_j(t) = \min_i \|\mathbf{X}_i - \mathbf{Y}_j(t)\| \qquad j = 1, \ldots, M(t). \tag{11}$$

Thus, at this instant, $\Gamma_j(t)$ is the city closest to the neuron j, and the intention is that it is represented by $\mathbf{Y}_j(t)$.

At every time instant a city $\mathbf{X}_i$ is presented to the network. The neuron j^* closest to $\mathbf{X}_i$ is now determined as the winner and it, together with all its neighbors in the bubble of activity, is moved toward $\mathbf{X}_i$ with a strength determined by an appropriate kernel function. In all the experiments the Gaussian kernel is used with a variance σ^2. However, rather than using the kernel associated with the coordinates of the neurons, the kernel associated with the indices is employed as described by Λ in Equation (5). Subsequently, the rest of the neurons outside this bubble are moved in such a way that the mean of the neurons currently on the ring coincides with the mean of the cities represented by these neurons. Notice that the mean is not forced to converge towards the mean of all the cities—it is only made to move onto the mean of the cities, the set $\mathcal{P}$, represented by the subset $\left\{\mathbf{X}_{\Gamma_j(t)} : j = 1, \ldots, M(t)\right\}$.

Observe now that the number of neurons $M(t)$ changes with time. Insertions and deletions are permitted in a manner analogous to the scheme by Angéniol et al. [2]. If a neuron wins the competition for two different cities in the same epoch, a new neuron is created with the same coordinates as the winner neuron. This newly created neuron is inserted into the ring as the neighbor of the winner. Both neurons are inhibited until they disperse by the movements of the neighboring neurons. In the same way, a neuron is deleted if it does not win a competition during a fixed number of (say, three) epochs (complete presentation of all the cities).

The difference between our insertion/deletion processes and Angéniol et al.'s is quite subtle. Consider the situation when a new neuron, k, is inserted because a neuron l, wins the competition twice. Thus $\mathbf{Y}_k(t)$ is exactly the same as $\mathbf{Y}_l(t)$ since they are both at the same location. However, $\Gamma_k(t) \neq \Gamma_l(t)$ since they both represent completely different cities. The dispersing phase of KNIES now augments this effect since they are both on the ring and so they both contribute identically to the mean of the set of neurons although the cities which they represent are different. Thus the effect of the dispersion module causes the neurons to be at completely different locations than if they had been merely affected by Angéniol et al.'s procedure. In the same way, a neuron when deleted will no more be on the ring, and so its coordinates disappear when the mean of the neurons is made to coincide with the mean of the cities they represent. Based on the above, one can argue that, in one sense, the present scheme is a generalization of the algorithm due to Angéniol et al. because it introduces two modules, the attraction and the dispersion modules, (as opposed to a single one used by the latter)—to incorporate the learning of the position of the neurons and the identity of the cities which they represent. The pseudocode and other algorithmic details are given in [5].

2.4.4 KNIES_TSP_Global: A Simplification of KNIES_TSP

KNIES_TSP can be computationally expensive. This is just because we try to maintain the mean of the neurons to be the mean of the cities they represent, and so, we are continuously posed with the problem of identifying the city associated with each neuron. Consequently, $\{\Gamma_j(t) : j = 1, \ldots, M(t)\}$ has to be identified after the presentation of each city. A modification of KNIES_TSP (called KNIES_TSP_Global) approximates this by assuming that the migrations are always done towards the global mean of all the cities. The only difference between KNIES_TSP and KNIES_TSP_Global is that in the latter, the mean of the neurons on the ring always moves towards (not onto) the global mean of all the cities, instead of moving exactly onto the local mean only of the cities represented by the neurons currently on the ring.

This is accomplished by bookkeeping the amount by which the neurons inside the activation bubble are updated along the x and y coordinate axes. The total amount of updates along both coordinate axes (which are computed by summing up the recorded quantities algebraically) make up the components of a vector $\Delta\mathbf{Y}(t)$ which represents the total change in the coordinates of the neurons in $B_{j*}(t)$. This total change is equally distributed among the neurons which are not in $B_{j*}(t)$ as follows:

$$\mathbf{Y}_j(t) = \mathbf{Y}_j(t) - \frac{1}{M(t) \cdot (M(t) - |B_{j^*}(t)|)} \Delta\mathbf{Y}(t) \qquad j \notin B_{j^*}(t) \tag{12}$$

The above assignment replaces their respective steps in KNIES_TSP to yield KNIES_TSP_Global. This procedure has also been implemented and tested. It is computationally faster than KNIES_TSP since it does not have to identify $\Gamma_j(t)$ after the presentation of each city, which in turn is done by identifying the nearest city to a neuron on the ring. It is also quite accurate even though (in its transient behavior) it is theoretically a deviation from the principle motivating KNIES.

3 Neural Solutions to the Hamiltonian Path Problem

The Euclidean Hamiltonian path problem (HPP), which is also known to be *NP*-complete [18], can be posed as follows: Given a set $\{\mathbf{X}_i : 1 \leq i \leq N\}$ of N cities with starting and terminal cities $\mathbf{X}_s$ and $\mathbf{X}_t$, respectively and distances for each pair of cities, what is the shortest path that starts at $\mathbf{X}_s$, terminates at $\mathbf{X}_t$, and visits each city exactly once?

The HPP has interesting applications such as on-line optimization of flexible manufacturing systems [6] and sequencing by hybridization of DNA [34]. A detailed, excellent survey of the techniques used to solve the HPP can be found in [6]. It is interesting to note that the independent solutions to the HPP are few; indeed, as explained in [31], most solutions utilize the underlying solution to the TSP. This is because the HPP can be solved using the solution to the TSP as follows:

1. The distance between $\mathbf{X}_s$ and $\mathbf{X}_t$ is set to an arbitrarily small value ($-\infty$), or,
2. A new city is added to the set of cities whose intercity distances with $\mathbf{X}_s$, and $\mathbf{X}_t$ are set to zero.

It is easy to see that since the path between $\mathbf{X}_s$ and $\mathbf{X}_t$ has to be included in the corresponding TSP, both of the above strategies will indeed yield a solution to the underlying HPP.

Although there are many solution techniques based on neural networks (presented in the previous section) which yield approximate solutions to the TSP, none of them have been extended for the HPP except the one explained below. The reason for this is primarily because the constraints that the TSP imposes permit the neurons to be placed on a ring, and force their migration to satisfy the cyclic nature of their locations with respect to each other. Generalizing this for a "linear" structure is far from trivial since there is no straightforward way by which the energy functions can be correspondingly extended, and this is probably why no such solutions have been reported.

3.1 The Guilty Net Solution to the Hamiltonian Path Problem

Jeffries and Niznik [21] present a scheme to solve HPP by extending the basic ideas of the guilty net. In their method, a regression line going through the cities (i.e., data points) is first computed, and then the cities are projected onto this line. The locations of these projections form the initial starting coordinates of the neurons. The algorithm then invokes the guilty net by Burke and Damany [9]. In this algorithm the neurons are migrated until they stabilize at their final positions where they coincide with cities. However, the guilty net does not necessarily converge, and so when such a non-convergence condition is attained, neurons are inserted and deleted using the following technique:

1. If a neuron is not assigned to a city (data point) at the end of the guilty net algorithm, the neuron is deleted. Similarly, if multiple neurons are associated with the same city, the neurons which are the furthermost to it are deleted.

2. If a neuron has more than one city associated with it, a new neuron is introduced, and each of these is associated with a specific city.

Finally, after the entire process converges, a 2-Opt-like heuristic strategy [31] is invoked to remove edge-crossings.

By using this simple strategy, Jeffries and Niznik [21] claim that they can yield near optimal Hamiltonian paths and demonstrate their scheme for a simple data set consisting of 22 cities. It should be mentioned that their scheme *does not solve the HPP* since the endpoints of their path are not specified and fixed. In other words their algorithm tries to compute the shortest Hamiltonian path between all possible city pairs. Furthermore, the scheme is not an all-neural approach as it resorts to a final heuristic to remove the crossings. Finally, the scheme has the inherent drawbacks of the guilty net—it is not guaranteed to converge except after using the above mentioned Angéniol et al.'s like insertion/deletion steps. This scheme is the only reported neural method that moves in the direction of solving the HPP without resorting to the underlying solution of the TSP. In [1], new self-organizing neural network approaches were presented which indeed yielded good approximate solutions to the HPP itself. Since they are later used for our patched algorithm, we describe them briefly below.

3.2 GSOM_HPP: A Generalized SOM Solution to the Hamiltonian Path Problem

The self-organizing map as suggested by Angéniol et al. [2] has been modified to yield a solution to the Euclidean Hamiltonian path problem (HPP) and is referred to as the Generalized SOM Solution (GSOM_HPP) in a recent work [1]. This is done as follows.

The algorithm is initialized by using M neurons, $\{\mathbf{Y}_j(0) : 1 \leq j \leq M\}$, where M is a user specified input parameter. Two of these neurons, $\mathbf{Y}_1$ and $\mathbf{Y}_M$ are located at the coordinates of $\mathbf{X}_s$ and $\mathbf{X}_t$, the starting and terminal cities respectively, and are called the *anchor* neurons. The locations of the other $M-2$ neurons, $\{\mathbf{Y}_j(0) : 2 \leq j \leq M-1\}$, are on the straight line joining $\mathbf{X}_s$ and $\mathbf{X}_t$ where they are spaced equally. Thus,

$$\mathbf{Y}_j(0) = \mathbf{X}_s + (j-1)\,\frac{(\mathbf{X}_t - \mathbf{X}_s)}{M-1} \qquad j = 2, \ldots, M-1. \tag{13}$$

The cities are now presented to the algorithm one by one, and at each step a SOM procedure is invoked. If the winning neuron is an internal neuron (which is not anchored at the endpoints) it is moved towards the presented city, and all the other neurons are moved towards the city as well using the Gaussian kernel with the specified variance whose value also decreases with the epochs. The crucial issue, of course, is that the anchor neurons at $\mathbf{X}_s$ and $\mathbf{X}_t$ are not migrated even if they become winners upon presentation of some city $\mathbf{X}_i$. It is also to be noted that at each epoch they win at least one competition which occurs upon presenting the starting and terminal cities. As usual, if a neuron does not win a competition during three epochs, it is deleted, and if a neuron wins a competition twice in the same epoch a new neuron is created at the same position. The anchor neurons participate in the creation process, but under slightly different conditions. Thus, if the winning neuron is one of the anchor neurons upon presenting a city other than the starting or terminal city, i.e., $(\mathbf{Y}_1$ or $\mathbf{Y}_M)$, a new neuron is created instantaneously at the same location. Unlike the anchor neuron, however, the newly created one is permitted to migrate. When the algorithm terminates, the number of neurons can be larger than the number of cities because any city can be represented by more than one neuron located exactly on the city at the final configuration. Although this has no effect on the length of the final Hamiltonian path (the distance between two such neurons is zero), duplicate neurons may be eliminated by a pass over all the neurons.

3.3 KNIES_HPP: A KNIES Solution to the Hamiltonian Path Problem

Since KNIES is a useful scheme to solve the TSP, it can be adapted for the HPP. First of all, the initialization process is employed to determine the location of the neurons. As in the case of GSOM_HPP, the algorithm is initialized by using M neurons, where M is a user specified input parameter. Two of these neurons are located at the coordinates of $\mathbf{X}_s$ and $\mathbf{X}_t$, the starting and terminal cities respectively, and are the anchor neurons. The remaining neurons, namely $\{\mathbf{Y}_j(t) : 2 \leq j \leq M-1\}$, are first initialized according to equation (13) at $t = 0$. Then they are moved so that their mean falls exactly on the global mean of the cities. The intention is to maintain the mean of the neurons to be invariant. This is done as follows. If $\overline{\mathbf{X}}$ is the mean of the coordinates of the cities, and $\overline{\mathbf{Y}}(0)$ is the mean of the coordinates of the neurons, $\mathbf{Y}_j(0)$ is set as:

$$\mathbf{Y}_j(0) = \mathbf{Y}_j(0) + \frac{M}{M-2}\left(\overline{\mathbf{X}} - \overline{\mathbf{Y}}(0)\right) \qquad 2 \leq j \leq M-1. \tag{14}$$

It is easy to see that as a result of the above assignment, the new $\{\mathbf{Y}_j(0) : 2 \leq j \leq M-1\}$ are all located on the same side of the line joining $\mathbf{X}_s$ and $\mathbf{X}_t$, and that they are located on the side where the global mean falls. This is distinct from the scheme of Jeffries and Niznik [21], where the initial neurons fall on the regression line. Furthermore, the mean of the cities and that of the neurons coincide and thus, we can attempt to maintain this as an invariant using the two modules of KNIES. The attracting phase is similar to KNIES_TSP's. However the dispersing phase is somewhat different.

Let $\mathbf{X}_{total}(t)$ be the vector sum of the coordinates of the cities represented by the neurons and $\mathbf{Y}_{total}(t)$ be the vector sum of the coordinates of the neurons at any epoch t. In order to bring the mean of the cities and the mean of the neurons to the same position, the difference between the two vector sums is evenly distributed among those neurons which are not in the bubble of activity. It is interesting to note that unlike the TSP (where the neurons are either inside the bubble or outside it), in the case of the HPP these neurons fall either to the left of the bubble or to its right. Let $L(t)$ and $R(t)$ be the number of neurons respectively on the left and right hand sides of the bubble at any epoch t. These $L(t) + R(t)$ neurons are moved in such a way that the mean of the neurons currently on the band coincides with the mean of the cities they represent. Notice that we do not make the mean converge towards the mean of all the cities—it is only made to move onto the mean of the cities represented by the subset $\left\{\mathbf{X}_{\Gamma_j(t)} : j = 1, \ldots, M(t)\right\}$ of the set $\mathcal{P}$. Here $\Gamma_j(t)$ is defined as in Equation (11). It can be easily verified that this is achieved by updating:

$$\mathbf{Y}_j(t) = \mathbf{Y}_j(t) + \frac{\mathbf{X}_{total}(t) - \mathbf{Y}_{total}(t)}{L(t) + R(t)} \qquad j \notin B_{j^*}(t). \tag{15}$$

The above is surprising, because one would expect that the changes made on the left are proportional to $L(t)$, and that the changes made on the right are proportional to $R(t)$. It is indeed the case that the latter is true, but when the total change on the left is averaged by $L(t)$, and the total change on the right is averaged by $R(t)$, it turns out that all the neurons "outside" the bubble are migrated by exactly the same amount. Thus, the dispersing phase is really a very straightforward operation.

3.4 KNIES_HPP_Global : A Simplification of KNIES_HPP

KNIES_HPP has the same drawback as KNIES_TSP; it can be computationally expensive. This is just because we try to maintain the mean of the neurons to be the mean of the cities they represent. A modification of KNIES_HPP (called KNIES_HPP_Global) approximates this by assuming that we attempt to always migrate towards the global mean of all the cities. The only difference between KNIES_HPP and KNIES_HPP_Global is that in the latter, the mean of the neurons on the band always moves towards (not onto) the *global* mean of all the cities, instead of moving exactly onto the *local* mean only of the cities represented by the neurons currently on the band. This is accomplished again by applying Equation (12).

The experimental results obtained by KNIES_HPP and KNIES_HPP_Global on instances of the TSPLIB can be found in [1].

4 Decomposition Approach to the Euclidean TSP

Our strategy to reduce the complexity of a large-scale traveling salesman problem instance is to decompose or partition it into smaller HPP subproblems, which are easier to solve. In essence, we solve a large TSP by spawning, smaller HPPs. Once these smaller HPP instances are individually solved, the solution to the original TSP is obtained by patching the solutions to the HPP subproblems. We are not aware of any previous (all neural) solutions which attempt such a strategy.

The partitioning into smaller HPP subproblems is achieved by clustering the cities of the original problem in a way that the structural properties of the problem instance are preserved. Generally speaking, the problem of size n is divided into k nonoverlapping clusters C_i of size n_i ($C_i \cap C_j = \emptyset \quad \forall\ i \neq j$), where $\max\{n_i : i = 1, \ldots, k\} \ll n$.

Clustering is a research topic in itself and there are numerous mathematical methods which have been proposed to solve it [27]. After the cities are partitioned into different clusters, it would be possible to proceed in two different ways. One is (as is the case in the TSP-heuristic of Karp [23]) to solve the traveling salesman subproblem for each cluster and then to join the subtours to form a global tour. The traveling salesman subproblems can be solved by using any known method. For the generation of the global tour the following procedure might be applied: We begin with an arbitrary subtour T_1. This subtour is then connected to another subtour T_2 to form a new tour. In general, a subtour T_i is connected to the global tour $\hat{T}_i$ generated so far by removing one edge of T_i and one edge of $\hat{T}_i$ and then by replacing them by two new edges connecting T_i and $\hat{T}_i$ to form a new tour $\hat{T}_{i+1}$. This procedure, however, has an important disadvantage: the subtours are not optimized with respect to the edges connecting the clusters. So a better approach is to obtain the global tour *before* the subproblems are solved in order to integrate the obtained information into the local TSP-heuristic. In this way the local solution also considers the city where the global tour enters the cluster, and the city, where it leaves the cluster and so takes into account the outline of the global tour. As a result better global solutions are obtained.

This latter approach is the second way of proceeding after the cities are partitioned. We give the steps of this approach, which we will adopt for KNIES_DECOMPOSE, in the following.

1. Partition the cities into clusters for a given number of clusters.

2. Determine an order for visiting the clusters.

3. Identify a city to enter and a city to leave for each cluster. They obey the ordering obtained in step 2 and form the bridges between the clusters.

4. Find a Hamiltonian path in every cluster between the entrance and exit cities.

5. Connect the Hamiltonian paths by using the bridges obtained in step 3.

As mentioned above, there has been no reported TSP solutions of an analogous nature.

5 All-Neural Decomposition: KNIES_DECOMPOSE

The first step of KNIES_DECOMPOSE is to partition the cities into clusters. The clustering is accomplished using vector quantization. Here the number of clusters is a parameter and there are as many codebook vectors as the number of clusters. The codebook vectors are moved in the plane until they find their final places. The closest codebook vector to a city determines the cluster to which it belongs. Hence the input space is divided into clusters each of which is represented by a codebook vector. At this point the codebook vectors and the centroid of the clusters coincide. Therefore the codebook vectors representing the clusters can be used to find the global tour through the clusters. However, in order to obtain a better discrimination of the clusters (i.e., near-optimal cluster boundaries) we make use of the *intra-regional* and *inter-regional polarizing*, as suggested by Kohonen for his Learning Vector Quantization (LVQ2) algorithm [25].

5.1 The Intra-Regional Polarizing

The aim of intra-regional polarizing is to represent each cluster C_k by a number of codebook vectors M_k where M_k increases with the number of cities located in that cluster which is denoted as N_k. Specifically, we have set M_k as: $M_k = \lfloor 0.3\ N_k \rfloor$. If there are three or less cities in a cluster, then M_k is set equal to one.

The set of codebook vectors for cluster k, $\{Q_{k,j} : 1 \leq j \leq M_k\}$ are initially located on a circle the center of which is the mean of the cities belonging to that cluster. The codebook vectors are then updated according to the formula given below:

$$Q_{k,j}(t+1) = \begin{cases} (1-\alpha(t))\, Q_{k,j}(t) + \alpha(t)\, P_{k,i} & \text{if } Q_{k,j} \text{ is the closest codebook vector to the data point } P_{k,i} \\ Q_{k,j}(t) & \text{otherwise} \end{cases} \tag{16}$$

$\alpha(t)$ is decremented linearly from unity for the initial learning phase and then switched to 0.2 and is decreased linearly for the fine-tuning phase. After the individual clusters have been represented by M_k codebook vectors they are tested to see whether they adequately classify the cities within their clusters. Therefore, the inter-regional polarizing phase has been employed where the codebook vectors do not find their places by learning only from the cities within their own clusters (as in the intra-regional polarizing phase) but they are also migrated in such a way that they polarize away from the cities of the neighboring clusters. The principle by which this is done is as follows:

5.2 The Inter-Regional Polarizing

Suppose that a point $P \in C_k$ is examined. Also assume that the two closest codebook vectors to P (among all the codebook vectors) are Q_a and Q_b. If both Q_a and Q_b do not belong to the cluster C_k, clearly, the information content in P (with respect to Q_a and Q_b) is misleading, and so it is futile to migrate Q_a and Q_b using this information. However, if both of them are intended to represent C_k, clearly, the information in P can be used to achieve an even finer tuning to their locations. Thus, in this scenario, both Q_a and Q_b are moved marginally from their current locations along the line towards P. The final scenario is the case when one of them, Q_a (Q_b), correctly belongs to C_k, and the other, Q_b (Q_a), belongs to a different cluster. In this case, the information in P can be used to achieve an even finer tuning to their locations by migrating Q_a (Q_b) marginally from its current location along the hyperline towards P, and migrating the other codebook vector Q_b (Q_a) marginally from its current location along the line away from P. Since we do not want the "straggler" points (the points which are misclassified, but which probably would not have been correctly classified even by an optimal classifier) to completely dictate (and thus, disturb) the polarizing, this migration is invoked only if the node P lies within a pre-specified window of interest, W. This restriction has also been recommended in the literature [24, 25], and typically, this window, W, is a hypersphere centered at the bisector between the codebook vectors Q_a and Q_b. Also, as recommended in the literature, the polarizing of both Q_a and Q_b (when both of them correctly classify P) is made to be of much smaller magnitude than in the scenario when either of them misclassifies it. These steps are formally given below:

$$\begin{array}{ll}
Q_a(t+1) = (1-\epsilon\gamma)\,Q_a(t) + \epsilon\gamma P & \text{if } Q_a, Q_b \in C_k;\ P \in W \\
Q_b(t+1) = (1-\epsilon\gamma)\,Q_b(t) + \epsilon\gamma P & \text{if } Q_a, Q_b \in C_k;\ P \in W \\
Q_a(t+1) = (1-\gamma)\,Q_a(t) + \gamma P & \text{if } Q_a \in C_k;\ Q_b \in C_j \neq C_k;\ P \in W \\
Q_b(t+1) = (1+\gamma)\,Q_b(t) - \gamma P & \text{if } Q_a \in C_k;\ Q_b \in C_j \neq C_k;\ P \in W \\
Q_a(t+1) = (1+\gamma)\,Q_a(t) - \gamma P & \text{if } Q_a \in C_j \neq C_k;\ Q_b \in C_k;\ P \in W \\
Q_b(t+1) = (1-\gamma)\,Q_b(t) + \gamma P & \text{if } Q_a \in C_j \neq C_k;\ Q_b \in C_k;\ P \in W \\
Q_a(t+1) = Q_a(t) & \text{if } P \notin W \\
Q_b(t+1) = Q_b(t) & \text{if } P \notin W
\end{array} \tag{17}$$

There are three parameters in these update equations; γ, ϵ, and the diameter of the hypersphere W centered at the bisector of the two nearest codebook vectors. Except ϵ, which is kept constant at 0.25, experiments were performed with different values of parameters γ and the diameter of W in order to see the effect of the inter-regional polarizing on the quality of the solution (i.e., final tour or path length). Both γ, and the diameter of W assumed values in the interval $(0, 1)$ with increments of 0.1.

5.3 Obtaining the TSP Tour for Clusters

The output of the inter-regional polarizing phase is the final partitioning of the cities into clusters. The next step is to determine the global tour through the clusters. To accomplish this, the centroid of each cluster is found by computing the coordinatewise average of the cities located in that cluster and the algorithm KNIES_TSP_Global is invoked. KNIES_TSP_Global quickly yields a tour passing through the centroids. Hence, the sequence at which the global tour visits the clusters is determined. The next step is to find out the entrance and exit cities for each cluster. This is done as follows. For two subsequent clusters in the global tour the two nearest cities is found such that one city belongs to the first cluster and the other belongs to the second cluster. These two cities constitute the bridge between the two clusters. In other words the global tour passes from one cluster to another through these two cities.

After the entrance and exit cities for each cluster are found, the remaining task involves the determination of the Hamiltonian paths between these cities, which is done by running the algorithm KNIES_HPP_Global [1]. Finally Hamiltonian paths are glued together to obtain a good TSP tour.

6 Computational Results

The crucial parameter for the decomposition approach is the number of clusters, since the remaining parameters of algorithms KNIES_TSP_Global and KNIES_HPP_Global are fixed. Particularly, in our experiments, the following values were set as the parameters of KNIES_TSP_Global used to determine the order in which the clusters are visited: $K_\sigma = 0.8$, $\omega = 0.2$, and $\sigma = 20$ initially. The parameter ω determines which neurons will be moved in the attracting phase. Thus, $\omega = 0.2$ means that 20 per cent of all the neuron currently on the ring (10 per cent on one side of the winning neuron and the remaining 10 per cent on the other side) are moved towards the presented city. Furthermore, the initial number of neurons is set equal to the number of clusters. The same parameter setting is also used for KNIES_HPP_Global. The initial number of neurons is set equal to 0.3 times the number of cities (including the entrance and exit cities) in the cluster for which the Hamiltonian path is to be found. If the number of neurons turns out to be less than three, then the algorithm is run with three neurons initially. These values for the parameters are selected since the algorithms KNIES_TSP_Global and KNIES_HPP_Global provided satisfactory results with these parameters for most of the instances as reported in recent works [1, 5].

The steps of KNIES_DECOMPOSE are illustrated in Figure 1 while solving the instance `eil101` for 10 clusters. Figure **??** (a) shows the result of the partitioning. The cities are divided into 10 clusters. Figure **??** (b) contains the global tour through the clusters computed by using KNIES_TSP_Global. The global tour gives the order in which clusters are visited on the final tour. Hence, it is now possible to determine the entrance and exit cities of the clusters. The exit city of a cluster and the entrance city of the subsequent cluster (according to the order in which clusters are visited) are connected and this connection constitutes the bridge between the two clusters. This can be seen also in Figure **??** (b). Figure **??** (c), is a snapshot taken after the Hamiltonian path is determined by KNIES_HPP_Global for one of the clusters. As the Hamiltonian path is found for each cluster and glued with that of the subsequent cluster, the TSP tour for the original problem grows gradually and gets its final shape. The final TSP tour for the instance `eil101` can be seen in Figure **??** (d) without the cluster borders. Darker edges are the inter-cluster bridges.

It should be noted that since the computation for the various clusters are not interrelated, the entire process can be trivially rendered parallel.

For each instance of Table 2, experiments are performed with different number of clusters so that on the average no more than 50 and no less than 14 cities belong to each cluster which is achieved when the following is satisfied: $\lfloor 0.02\,N \rfloor \leq$ number of clusters $\leq \lfloor 0.07\,N \rfloor$. Here N is the problem size, i.e., the number of cities. This is because when the number of cities in any cluster exceeds 50 or so, the computational time of determining the Hamiltonian path for that cluster increases significantly. Hence, our experience dictated that we choose the number of clusters not less than $\lfloor 0.02\,N \rfloor$. On the other hand, when the cities are divided into too many clusters, then the number of cities in each cluster gets smaller and the preclustered structures are destroyed, and we move away from the global optimum. Again, we found that the number of clusters had to be less than $\lfloor 0.07\,N \rfloor$. To give an example, the number of clusters varied between $\lfloor 0.02 \times 532 \rfloor = 10$ and $\lfloor 0.07 \times 532 \rfloor = 37$ for `att532`.

As pointed out in the previous section, the parameters of the inter-regional polarizing were set to the following values: $\epsilon = 0.25$, and both γ and the diameter of W range between zero and one. Table 1 includes the best results obtained for `att532` as a function of the number of clusters. The values of γ and the diameter of W, which provide the best tour lengths, are also given. The overall best result is achieved when the cities are divided into 13 clusters. The overall best results obtained by KNIES_DECOMPOSE for different TSP instances selected from the TSPLIB [30] and the best values provided by KNIES_TSP (referred to as KL) and KNIES_TSP_Global (referred to as KG) for the same instances are given in Table 2.

Table 3 contains the relative deviations of the three approaches from the optimal values. As it can be observed in the table the success of KNIES_DECOMPOSE increases as the problem size increases. For `att532`, `pcb442`, `kroA200`, `rat195`, KNIES_DECOMPOSE provides shorter tour lengths than both KNIES_TSP and KNIES_TSP_Global. If only these four instances are considered, the average relative deviations from the optimal tour lengths become 7.03, 8.94, and 8.93 for KNIES_DECOMPOSE, KNIES_TSP, and KNIES_TSP_Global, respectively.

The most important advantage of the new method which cannot be perceived in the table is its speed. Compared with the other two approaches it is possible to obtain solutions quickly even for large problems. As for example KNIES_DECOMPOSE runs in 3.02 seconds on an HP–9000 K220 server with 256 MByte RAM working within an HP–UX 10.1 environment to obtain a tour length of 29,388.88 for `att532`. The number of clusters is 13, $\gamma = 0.6$ and the diameter of W is equal to 0.8. The CPU time goes up to 8.92 and 45.29 seconds for the large TSPLIB instances `pr1002` and `pr2392` on the same setup. Final tour lengths determined with KNIES_DECOMPOSE are respectively 281,557.81 and 429,772.31 and their relative deviations from the optimal tour lengths (259,045 for `pr1002` and 378,032 for `pr2392`) are 8.69 % and 13.69 %. These solutions are obtained with the following parameters:
`pr1002`: $\gamma = 0.8$, the diameter of W is equal to 0.8 and 17 clusters
`pr2392`: $\gamma = 0.4$, the diameter of W is equal to 0.8 and 16 clusters.
Note that the number of clusters are even less than the lower bounds 20 and 47, which provides additional clues about the success of the new method. The codes by Applegate et al. run 94.7 seconds and 342.2 seconds on a single processor of a Digital AlphaServer 4100 [3]. The same setup runs 294.3 seconds to compute the optimum tour for `att532`. These results are very much encouraging for neural algorithms, since they have been severely criticized for their time inefficiency. It is also possible to solve individual Hamiltonian path problems in a distributed environment. This will speed up the computations even more.

7 Conclusion

For large size instances neural (and actually, even non-neural) approaches to solve the TSP tend to be prohibitively time consuming. Decomposing the original problem into subproblems and solving each subproblem separately by paying attention to the quality of the overall solution turns out to be a very effective strategy. The method we have introduced in this paper is the first all-neural decomposition approach for solving large

Table 1: Best results for `att532` as a function of number of clusters.

Number of Clusters	**Tour Length**	γ	**Diameter of** W
10	30018.16	0.7	0.6
11	29891.98	0.6	0.8
12	29927.55	0.7	0.6
13	29388.88	0.6	0.8
14	29883.40	0.2	0.6
15	29782.35	0.5	0.6
16	29628.22	0.4	0.6
17	30078.65	0.8	0.9
18	29821.84	0.3	0.5
19	30231.76	0.5	0.3
20	29921.52	0.8	0.9
21	30239.64	0.7	0.9
22	30581.60	0.5	0.7
23	30286.34	0.7	0.5
24	30269.26	0.1	0.6
25	30215.12	0.6	0.6
26	29949.10	0.9	0.8
27	30278.20	0.4	0.5
28	30017.96	0.9	0.6
29	30404.09	0.5	0.8
30	30507.37	0.2	0.8
31	30407.92	0.4	0.3
32	30874.54	0.8	0.2
33	29862.38	0.5	0.3
34	30299.79	0.5	0.3
35	30871.63	0.6	0.9
36	30148.04	0.1	0.9
37	30682.40	0.5	0.5

Table 2: Comparison of the results obtained for different algorithms instances.

Instance	**Opt. Value**	**Decompos.** (# of clust.)	**KL** $(M, \sigma, K_\sigma, \omega)$	**KG** $(M, \sigma, K_\sigma, \omega)$
`att532`	27686	29388.9 (13)	29551.6 (200,30,0.8,0.15)	29569.8 (400,45,0.8,0.1)
`bier127`	118282	126080.8 (9)	121548.7 (100,15,0.8,0.1)	121923.7 (125,50,0.8,0.1)
`eil51`	426	440.9 (5)	438.2 (10,25,0.8,0.1)	438.2 (20,50,0.8,0.05)
`eil76`	538	572.9 (6)	564.8 (90,20,0.8,0.15)	567.5 (15,5,0.8,0.2)
`eil101`	629	672.0 (4)	658.3 (20,25,0.8,0.8)	664.4 (40,35,0.8,0.25)
`kroA200`	28568	30184.9 (12)	30200.8 (200,25,0.8,0.25)	30444.9 (160,10,0.8,0.05)
`lin105`	14379	14693.1 (8)	14664.4 (100,50,0.8,0.2)	14564.6 (100,35,0.8,0.05)
`pcb442`	50778	54838.6 (3)	56399.9 (500,30,0.8,0.1)	56082.9 (450,40,0.8,0.15)
`pr107`	44303	49103.9 (4)	44628.3 (100,45,0.8,0.1)	44491.1 (60,25,0.8,0.25)
`pr124`	59030	60931.5 (3)	59075.7 (25,25,0.8,0.10)	59320.6 (125,10,0.8,0.05)
`pr136`	96772	98641.2 (8)	101156.8 (75,40,0.8,0.15)	101752.4 (30,35,0.8,0.05)
`pr152`	73682	76072.1 (15)	74395.5 (180,30,0.8,0.05)	74629.0 (60,10,0.8,0.2)
`rat195`	2323	2517.0 (2)	2607.3 (200,25,0.8,0.25)	2599.8 (200,25,0.8,0.1)
`rd100`	7910	8296.9 (9)	8075.7 (80,20,0.8,0.25)	8117.4 (60,10,0.8,0.05)
`st70`	675	699.8 (3)	685.2 (40,10,0.8,0.05)	690.7 (30,45,0.8,0.05)

Table 3: Relative deviation from the optimal tour length (per cent) of the various algorithms

Instance	**Decompositon**	**KL**	**KG**
att532	6.15	6.74	6.80
bier127	6.59	2.76	3.08
eil51	3.49	2.86	2.86
eil76	6.49	4.98	5.48
eil101	6.84	4.66	5.63
kroA200	5.66	5.72	6.57
lin105	2.18	1.98	1.29
pcb442	7.99	11.07	10.44
pr107	10.84	0.73	0.42
pr124	3.22	0.08	0.49
pr136	1.93	4.53	5.15
pr152	3.24	0.97	1.29
rat195	8.35	12.24	11.92
rd100	4.89	2.09	2.62
st70	3.67	1.51	2.33
Average Relative Deviations	5.41	4.19	4.42

TSP instances. The new algorithm, KNIES_DECOMPOSE is based on the foundational principles of Kohonen's SOM, and its recent variant KNIES. The final tour is obtained by combining Hamiltonian paths that KNIES_HPP constructs for the clusters determined according to Kohonen's clustering method.

Experimental results for instances taken from TSPLIB indicate that it is, generally speaking, the most accurate neural strategy for the TSP currently reported. Besides accuracy, KNIES_DECOMPOSE cuts the running time necessary for KNIES_TSP_Global to solve `att532`, from 49,888.68 seconds down to 3.02 seconds, with no deterioration in the solution quality. Besides, as the problem size increases KNIES_DECOMPOSE produces higher quality tours. This can be observed through the figures of Table 2. Also, an increase in the number of clusters does not necessarily imply superior approximations. In fact the quality of the final tour usually increases first up to certain value, and then decreases as long as the number of clusters increases. As a consequence we claim that it is not the number of clusters, but the quality of the clustering which is crucial; this results both in a higher speed and accuracy at the same time.

It is clear that there is still a lot of work to be done when it concerns using KNIES_DECOMPOSE. The solution quality is affected by both the quality of the Hamiltonian paths joining the entering and exiting cities in the clusters, and the bridges between the clusters. However, these two aspects are not independent since the entrance and exit cities also determine the input of the Hamiltonian path problems to be solved in each cluster. Hence, the determination of mechanisms which consider potential entry and exit cities, namely the inter-cluster bridges, is still an open problem.

References

[1] Altınel, İ.K., N. Aras, and J. Oommen, "Fast, Efficient and Accurate Solutions to the Hamiltonian Path Problem Using Neural Approaches," *Computers & Operations Research*, Vol 27, pp. 461–494, 2000.

[2] Angéniol, B., C. Vaubois, and J.Y. Le Texier, "Self-Organizing Feature Maps and the Traveling Salesman Problem," *Neural Networks*, Vol.1, pp. 289–293, 1988.

[3] Applegate, D., Bixby, R.. and W. Cook, On the solution of Traveling Salesman Problems, Technical Report No. CRPC - TR98744, Rice University, Houston, 1998.

[4] Aras, N., *New Neurocomputational Approaches for Estimating Road Travel Distances and for Solving the Euclidean Traveling Salesman Problem*, Ph. D. Dissertation, Boğaziçi University, 1999.

[5] Aras, N, J. Oommen and İ.K Altınel, "Kohonen Network Incorporating Explicit Statistics and its Application to the Traveling Salesman Problem," *Neural Networks*, Vol. 12, pp. 1273–1284, 1999.

[6] Ascheuer, N., Hamiltonian Path Problems in the on-line Optimization of Flexible Manufacturing Systems, Technical Report No. TR 96–3, ZIB, Berlin, 1996.

[7] Budinich, M., "A Self-Organizing Neural Network for the Traveling Salesman Problem That is Competitive with Simulated Annealing," *Neural Computation*, Vol. 8, pp. 416–424, 1996.

[8] Budinich, M., and B. Rosario, "A Neural Network for the Traveling Salesman Problem with a Well Behaved Energy Function," in S. W. Ellacott, J . C. Mason, I. J. Anderson (Eds.), *Mathematics of Neural Networks, Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston–London–Dordrecht, 1997.

[9] Burke, L.I. and P. Damany, "The Guilty Net for the Traveling Salesman Problem," *Computers & Operations Research*, Vol. 19, No. 3/4, pp. 255–265, 1992.

[10] Burke, L.I., "Neural Methods for the Traveling Salesman Problem: Insights From Operations Research," *Neural Networks*, Vol. 7, pp. 681–690, 1994.

[11] Burke, L.I., "Conscientious Neural Nets for Tour Construction in the Traveling Salesman Problem: The Vigilant Net," *Computers & Operations Research*, Vol. 23, No. pp. 121–129, 1996.

[12] Cichocki, A., and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, Wiley, Chichester, 1994.

[13] DeSieno, D., "Adding a Conscience Mechanism to Competitive Learning," *Proceedings of the IEEE International Conference on Neural Networks* 1, 117–124, 1988.

[14] Duda, R.O., and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, 1973.

[15] Durbin, R. and D. Willshaw, "An Analogue Approach to the Travelling Salesman Problem using an Elastic Net Method," *Nature*, Vol. 326, pp. 689–691, 1987.

[16] Fort, J.C., "Solving a Combinatorial Problem via Self-Organizing Process: An Application of the Kohonen Algorithm to the Travelling Salesman Problem," *Biological Cybernetics*, Vol. 59, pp. 33–40, 1988.

[17] Fukunaga, K, *Introduction to Statistical Pattern Recognition*, 2nd edition, Academic Press, San Diego, 1990.

[18] Garey, M.R., and D.S. Johnson, *Computers and Intractability*, W.H. Freeman, New York, 1979.

[19] Hopfield, J.J. and D.W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141–152, 1985.

[20] Hueter, G.J., "Solution of the Traveling Salesman Problem with an Adaptive Ring," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, CA, pp. I-85–92, 1988.

[21] Jeffreys, C., and T. Niznik, "Easing the Conscience of the Guilty Net," *Computers & Operations Research*, Vol. 21, pp. 961–968, 1994.

[22] Johnson, D.S. and L.A. McGeoch, "The Traveling Salesman Problem: a Case Study," in E. Aarts and J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, pp. 215–310, Wiley, Chichester, 1997.

[23] Karp, R., “Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane,” *Mathematics of Operations Research*, Vol. 2, pp. 209–224, 1977.

[24] Kohonen, T., “The Self-Organizing Map,” *Proc. IEEE*, Vol 78, pp. 1464–1480, 1990.

[25] Kohonen, T, *Self-Organizing Maps*, Springer-Verlag, Berlin, Germany, 1995.

[26] Lawler, E. L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985.

[27] Mirkin, B., *Mathematical Classification and Clustering*, Kluwer Academic Publishers, Boston–London–Dordrecht, 1996.

[28] Papadimitriou, C.H., “The Euclidean Traveling Salesman Problem is *NP*-Complete,” *Theoretical Computer Science*, Vol. 4, pp. 237–244, 1978.

[29] Potvin, J.-Y., “The Traveling Salesman Problem: A Neural Network Perspective,” *ORSA Journal on Computing*, Vol. 5, pp. 328–348, 1993.

[30] Reinelt, G., “TSPLIB—A Traveling Salesman Problem Library,” *ORSA Journal on Computing*, Vol. 3, No.4, pp. 376–384, 1991.

[31] Reinelt, G., *The Travelling Salesman. Computational Solutions for TSP Applications*, Springer-Verlag, Berlin, 1994.

[32] Ritter, H.J., and K.J. Schulten, “Kohonen’s Self-organizing Maps: Exploring Their Computational Capabilities,” *Proceedings of the International Joint Conference on Neural Networks*, pp. 2455–2460, 1988.

[33] Smith, K., “Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research,” *INFORMS Journal on Computing*, Vol. 11, No. 1, pp. 15–34, 1999.

[34] Waterman, M.S., *Introduction to Computational Biology*, Chapman and Hall, Cambridge, 1995.