KOHONEN-LIKE NEURAL SOLUTIONS TO THE HAMILTONIAN PATH PROBLEM

I. Kuban Altinel, Necati Aras, B. John Oommen

TR-98-07 SEPTEMBER 1998

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

Kohonen-like Neural Solutions to the Hamiltonian Path Problem

İ. Kuban Altınel[*]
Dept. of Industrial Eng.
Boğaziçi University
İstanbul, TÜRKİYE
altinel@boun.edu.tr

Necati Aras[*]
İ.d.e.a. A. Ş.
Tuzla 81719
İstanbul, TÜRKİYE
necatia@idea.com.tr

B. John Oommen[†]
School of Comp. Science
Carleton University
Ottawa, CANADA
oommen@scs.carleton.ca

Abstract : Unlike its cousin, the Euclidean Traveling Salesman Problem (TSP), to the best of our knowledge, there has been no documented *all-neural* solution to the Euclidean Hamiltonian Path Problem (HPP). The reason for this is the fact that the heuristics which map the cities onto the neurons "lose their credibility" because the underlying cyclic property of the *order* of the neurons used in the TSP is lost in the HPP. In this paper we present three neural solutions to the HPP. The first of these, GSOM_HPP, is a generalization of Kohonen's Self Organizing Map (SOM) as modified by Angéniol *et al.* [5]. The second and third methods use the recently-introduced self-organizing neural network, the Kohonen Network Incorporating Explicit Statistics (KNIES)[2]. The primary difference between KNIES and Kohonen's Self-Organizing Map (SOM) is the fact that unlike SOM, every iteration in the training phase includes *two distinct modules* - the attracting module and the dispersing module. As a result of SOM and the dispersing module introduced in KNIES the neurons *individually* find their places both statistically and topologically, and also *collectively* maintain their mean to be the mean of the data points which they represent. The new philosophy, which has previously [2] been used to effectively solve the Euclidean Traveling Salesman Problem (TSP), is now extended to solve the Euclidean Hamiltonian Path (HPP). These algorithms, which to our knowledge, are the first all-neural solutions to the HPP, have also been rigorously tested. Experimental results for problems obtained by modifying selected TSPLIB instances [46] for the HPP indicate that they are both accurate and efficient. Apart from the computational results presented, the paper also contains a systematic strategy by which the quality of any HPP algorithm can be quantified.

Keywords : Combinatorial Optimization, Hamiltonian Path, Neural Networks, Self-organizing maps.

1 Introduction

In this paper we present two Neural Network (NN) solutions to the Euclidean Hamiltonian Path Problem (HPP), which, to our knowledge, are the first *all-neural* solutions to the problem. Apart from explaining the solutions and presenting a survey of the related literature, the paper contains a complete report on how HPP algorithms can be tested, and a survey of the experimental results computed for problems obtained by modifying TSPLIB instances [46] has been included. The paper can thus be considered as a comprehensive survey of the field and the particular problem studied.

1.1 The Traveling Salesman Problem

The Euclidean Traveling Salesman Problem (TSP) has been one of the oldest "hard nuts" of Operations Research and Mathematical Programming. It is known to be NP-complete [44]. Any algorithm devised to solve the TSP tries to answer the following question: Given a set of N nodes (cities) and distances for each pair of nodes, what is the shortest tour that visits each node exactly once ? There are many exact and heuristic algorithms in the literature for the TSP [35, 47]. The exact algorithms, which are computationally expensive, provide an optimal solution, while the heuristic algorithms do not guarantee an optimal solution most of the time, but they "quickly" find near-optimal solutions which are within a

*Partially supported by the Boğaziçi Research Fund Grant 97H301D and 98A301D

†Senior Member IEEE. Partially supported by the National Science and Engineering Research Council of Canada (NSERC). The second author gratefully acknowledges the NSERC grant which permitted his travel to Türkiye.

few percent of the optimum [36]. The heuristic algorithms can be divided into three broad categories, i.e., tour construction heuristics, tour improvement heuristics and composite heuristics which make use of the first two kinds of heuristics. The well-known Lin-Kernighan heuristic [36] which is able to find solutions within 1 % of the optimum belongs to the third category.

In addition to the classical heuristic algorithms of Operations Research, there have also been several approaches based on artificial NNs which solve the TSP [12, 28, 45]. A *brief* overview of these schemes has been included in the forthcoming section. More recently, we have added to this collection of methods a scheme which uses the recently-introduced self-organizing neural network, the Kohonen Network Incorporating Explicit Statistics (KNIES) [2]. The primary difference between KNIES and Kohonen's Self-Organizing Map (SOM) is the fact that unlike SOM, every iteration in the training phase includes two distinct modules - the *attracting module* and the *dispersing module*. As a result of SOM and the dispersing module introduced in KNIES, the neurons *individually* find their places both statistically and topologically, and also *collectively* maintain their mean to be the mean of the data points which they represent. The paper [2], describes in detail, two particular versions of the latter. Based on the results obtained for instances computed from the TSPLIB problem set [46], we believe that the solutions presented in [2] are the most accurate of all NN schemes reported in the literature.

1.2 The Hamiltonian Path Problem

The TSP has a close cousin, the Euclidean Hamiltonian Path Problem (HPP)[1], which is also known to be NP-complete. The HPP can be posed as follows : Given a set $\{X_i : 1 \leq i \leq N\}$ of N nodes (cities) and distances for each pair of nodes, and starting and terminal nodes X_s and X_t respectively, what is the shortest path that starts at X_s, terminates at X_t, and which visits each node exactly once ?

The HPP has interesting applications such as on-line optimization of flexible manufacturing systems [3] and sequencing by hybridization of the DNA [58]. The literature reports a few independent exact and heuristic algorithms, but for the most part, algorithms for the HPP utilize fast solutions to its cousin, the TSP. As in the case of all NP-complete problems, the exact algorithms are computationally expensive and often prohibitive even though they provide an optimal solution. The heuristic algorithms do not guarantee an optimal solution - they "quickly" find an approximate, near-optimal solution. A detailed, excellent survey of the techniques used to solve the HPP can be found in [3].

It is interesting to note that the independent solutions to the HPP are few; indeed, as explained in [47], 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 X_s and X_t is set to an arbitrarily small value ($-\infty$), or,

2. A new node is added to the set of nodes with distances between it and X_s, as well as X_t are set to zero.

It is easy to see that since the path between X_s and X_t has to be included in the corresponding TSP, both of the above strategies will indeed yield a solution of the underlying HPP.

In this paper we attempt to solve the HPP without resorting to an underlying TSP solution. The reasons for this are many. First of all, we are motivated by the scientific nature of the problem itself. Secondly, in many real-life cases, it is well known that the cities (nodes) are specified in terms of their coordinates (Cartesian, geographical or grid) and it is really quite *artificial* to assign a very large negative value to one single distance while all the other distances in the distance matrix are Euclidean (or geographical). A third reason involves the computation of the TSP for large problems. When the problem size is very large, it is often impossible to compute even an approximate solution to the problem in a reasonable time. In such cases, it is often advantageous to cluster the points into patches[2], and an entry and an exit point for each cluster is determined. The TSP can then be solved by computing the HPP for each cluster from the respective entry and exit points of clusters. Clearly, it would be advantageous if the HPP could be solved in a fast and efficient manner for *each* of these clusters without having to resort to an artificial underlying TSP problem.

[1]There is a web-page on the world-wide web dedicated to results related to the Hamiltonian paths and cycles. Its address is : http://www.ing.unlp.edu.ar/cetad/mos/Hamilton.html monitored and maintained by Gregory Gutin and Pablo Moscato. The reader would do well to visit the page. Amazingly enough, there are only a *few* papers which describe *independent* algorithms for the HPP.

[2]We are currently working on such a patched-function all-neural solution to the TSP.

Apart from all of the above, the HPP is interesting in its own right because, the problem itself opens new avenues for attractive research problems. First, among these is the question of how we can justify the accuracy of an approximate solution which we have obtained. Second, and more important, it is very interesting to note that various neural solutions to the TSP do not lead directly to corresponding solutions to the HPP. This is because the assumption that the ordering of cities has a cyclic property vanishes. Instead, the cities have to assume a *linear* ordering, and the two end points, which otherwise were close to each other on a TSP tour, can become very distant from each other on an HPP path.

The format of the paper is as follows. Since we are studying neural solutions to the HPP, in the next section we shall present a brief overview of the neural solutions which have been devised for the Euclidean Traveling Salesman Problem (TSP). We shall explain the only reported solution to the Euclidean Hamiltonian Path Problem (HPP) which extends the basic work of Burke and Damany's Guilty Net (GN) [29]. However, as we shall see, this solution is not an all-neural solution, because it has to resort to an additional heuristic strategy to eliminate the crossings that are obtained after the convergence. Although [29] does not yield a solution to the HPP because the end points are not fixed, we do feel that this work is to be commanded and respected because, in our opinion, it is a pioneering work in the direction of trying to solve the HPP using some neural principles.

In Section 3 we explain KNIES, the Kohonen Network which Incorporates Explicit Statistics, which we recently introduced [2]. Section 4 is about the only neural approach to HPP we ran into in the literature. Section 5 is where we extend the results of Angéniol *et al.* [5] to yield our first neural method for the HPP. In Section 6 we show how KNIES can be extended to solve HPP. Finally, we discuss in some detail the question of how a solution method for the HPP can be evaluated for its quality in Section 7, and conclude the paper in Section 8.

2 Non-Kohonen Neural Solutions to the TSP

In addition to the classical heuristic algorithms of Operations Research, several NN solutions have been proposed to solve the TSP. The first successful attempt in this direction was the work of Hopfield and Tank [26]. The Hopfield-Tank model basically performs a gradient descent search to find a local minimum of an appropriate energy function E. Hence, it is expected that the local minimum corresponds to a good solution of the TSP being solved. However, the model suffers from the fact that the feasibility is not guaranteed, namely, not all the minima of the energy function represent feasible solutions to the TSP. This and some other deficiencies of the model have led researchers to improve the original model [1]. As a conclusion, however, it can be stated that both the original model as well as its variants with many refinements provide solutions for TSP instances which consist of 200 cities or less.

Besides the Hopfield-Tank model two other NN schemes for the TSP have been reported. The first one is the elastic net approach where a local minimum of an appropriately defined energy function is found by performing a gradient descent search [16]. Actually, Simic [54] has shown that both the Hopfield-Tank model and the elastic network can be derived from the principles of statistical mechanics. The second one is the self-organizing map approach which is the main topic of this work. It should be reminded that while the Hopfield-Tank model can also be applied for the asymmetric TSP (ATSP) the latter two approaches are applicable only for the Euclidean TSP (TSP).

The Hopfield-Tank algorithm is quite slow and is computationally extremely expensive, even when the number of cities considered is small (less than 50). The elastic net method [16] is also quite slow, but it has been recently improved by Vakhutinsky and Golden [56] by using hierarchical strategies. Of all the NN methods that are available, we believe that the algorithms derived from the Kohonen based SOM are both the most accurate and the fastest. In the interest of brevity and to present our results in the correct perspective, we shall review here only such methods. A more detailed survey of other NN schemes for solving the TSP can be found in [12, 28, 45].

Although all of these techniques have been used to yield approximate solutions to the TSP, none of them have been extended for the HPP. 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 TSP Solution Using the Kohonen Network

The best neural solutions for the TSP use the basic principles of Kohonen's Vector Quantization (VQ) and the self-organizing map (SOM) [24, 33, 34, 37, 40] as designed by Kohonen and his group. A *brief* overview would not be out of place.

The foundational ideas motivating VQ and the SOM are classical concepts that have been applied in the estimation of probability density functions. Traditionally, (in the realms of both statistical analysis and statistical pattern recognition) distributions have been represented either parametrically or non-parametrically. In the former, the user generally assumes the form of the distribution function and the parameters of the function are learned using the available data points. In pattern recognition (classification), these estimated distributions are subsequently utilized to generate the discriminant hyper-surfaces (which are, for example, hyperspheres or hyperellipsoids in the case of normally distributed random vectors) whence the classification is achieved. Unfortunately, this first of all assumes that the *parametric form* of the distributions are known, and this assumption is often a point of contention debated by the proponents of the two schools of thought.

As opposed to the latter, in non-parametric methods, the practitioner assumes that the data must be processed in its entirety (and not just by using a functional form to represent the data). The corresponding pattern recognition (classification) algorithms which result are generally of the nearest neighbor (or k-nearest neighbor) philosophy and are thus computationally expensive. The comparison of these two perspectives is found in standard pattern recognition textbooks [15, 19], and bounds on the classification error rate of non-parametric strategies (as compared to the optimal Bayesian parametric strategies) have also been derived.

The concept of the SOM can be perceived as a compromise between the above two schools of thought. Rather than assuming that the parametric form of the distributions are known and thus representing the entire data in a compressed form using *only* the estimates (and in the estimate domain), the SOM opts to represent the data in the actual feature space. However, as opposed to the non-parametric methods which use all the data in the training and testing phases of classification, the SOM compresses the information by representing it using a "small" set of vectors, **neurons** - which are called the set of **codebook** vectors in the SOM literature. These neurons are migrated in the **feature** domain so that they collectively represent the distribution under consideration. We shall refer to this phase as the *Intra-Regional Polarizing* phase. In a multi-class problem the neurons for each region are subsequently migrated so as to ensure that they adequately represent their own regions and furthermore distinguish between the other regions. This phase can be considered to be an *Inter-Regional Polarizing* phase, and can also be utilized to implicitly learn the discriminant function to be used in a subsequent classification module. Note that these discriminant functions are of a nearest neighbor philosophy, except that the nearest neighbors are drawn from the set of neurons - as opposed to the entire set of training samples. They thus drastically reduce the computational burden incurred in the testing of traditional non-parametric methods.

It is not appropriate for us to explain the theoretical details of VQ and the SOM here; they can be found in an excellent survey by Kohonen [33] and in [34]. However, in the interest of completeness and continuity and to better show how the KNIES scheme is built on the foundation of the SOM, we shall in all brevity, explain its various phases.

3.1 Vector Quantization and the SOM

We assume that we are given a set of N points $\mathcal{P} = \{X_i : 1 \leq i \leq N\}$ in a d-dimensional space. The primary aim of both VQ and the SOM is to represent these points in $\mathcal{P}$ by M representative points, neurons, $\mathcal{Y} = \{Y_j : 1 \leq j \leq M\}$. In a typical *pattern recognition* setting it is generally assumed that $M \ll N$, although, as we shall see in the Traveling Salesman Problem (TSP), this is not necessary. Indeed, in the TSP, it is not uncommon that in the learning phase $M \geq N$. Furthermore, although strictly speaking, we can represent $\mathcal{P}$ by M_N neurons (where M_N increases with N), in the interest of simplicity, in this paper we assume that the *final* number of representative neurons is a pre-determined constant.

The neurons $\{Y_j : 1 \leq j \leq M\}$ are initially assigned random positions which are typically within the convex hull of the points they are to represent, $\mathcal{P}$. We now consider how the migration of the neurons is achieved.

In both VQ and the SOM the polarizing algorithm is repeatedly presented with a point X_i from $\mathcal{P}$. The neurons now try to incorporate the topological information present in X_i. This is done as follows.

First of all the closest neuron to X_i, Y_{j*}, is determined. This neuron and a group of neurons in its neighborhood, B_{j*}, are now moved in the direction of X_i. The set B_{j*} is called the "Activation Bubble", and we shall presently specify how it is determined. The actual migration of the neurons is achieved by rendering the new Y_j to be a convex combination of the current Y_j and the data point X_i for all $j \in B_{j*}$. More explicitly, the updating algorithm is as follows :

$$Y_j(t+1) = \begin{cases} (1-\alpha(t))Y_j(t) + \alpha(t)X_i & if\ j \in B_{j*}(t) \\ Y_j(t) & otherwise \end{cases} \tag{1}$$

where 't' is the discretized (synchronized) time index.

This basic algorithm has two fundamental parameters, $\alpha(t)$ and the size of the bubble $B_{j*}(t)$. We shall now explain how these two parameters are assigned in a traditional setting [24, 33, 34, 37, 40, 57]. $\alpha(t)$ is called the adaptation constant and satisfies $0 < \alpha(t) < 1$. Kohonen and others [24, 33, 34, 37, 40, 57] recommend steadily decrementing $\alpha(t)$. Initially, while the neurons migrate to find their places the transitions made are drastic. In this phase it is generally recommended that the value of $\alpha(t)$ be close to unity. After the neurons are relatively close to their final locations the migrations must be much smaller and so, during this phase $\alpha(t)$ is generally made closer to zero. For example, as recommended in the literature [33, 34, 57], $\alpha(t)$ can be decremented linearly from unity for the initial learning phase and then switched to small values which decrease linearly from 0.2 for the fine-tuning phase.

The neighborhood bubble $B_{j*}(t)$ is the quantity which makes VQ differ from the SOM. Indeed, if the bubble is always taken as empty set, only the closest neuron is migrated and this yields a VQ scheme. However, in the SOM, the nearest neuron and the neurons within the bubble are also migrated, and it is this which permits the algorithm to be both *topology preserving* and self-organizing. The size of the bubble is initially assigned to be fairly large to allow a global ordering to develop. Consequently all the neurons tend to tie themselves into a knot for a value of $\alpha(t)$ that is close to unity ; they subsequently quickly disperse. Once this coarse spatial resolution is achieved the size of the bubble is steadily decreased. Hence only those neurons which are most relevant to the processed input point will be affected by it. Thus the ordering which has been achieved by the coarse resolution is not disturbed, but fine tuning on this ordering is permitted.

There are two ways of achieving the effect of the bubble size. The first is by explicitly decreasing the size of the bubble, and is done by linearly decrementing its diameter as :

$$W(t+1) = K_w \cdot W(t), \tag{2}$$

where K_w is a user-defined constant, and $[-W, W]$ specifies the activation bubble. An alternate method for reducing the effect of updating on the more distant neurons is by introducing a weighting function whose value decreases with the distance from the winning (or closest) neuron. Typical weighting functions are either of a Gaussian Window form (with a variance which steadily decreases), or can be of any kernel form used in non-parametric Parzen window methods [15, 19]. We shall discuss the use of such kernel functions in greater detail later. The effect of the polarizing is best perceived by observing how the neurons migrate in a 2 or 3-dimensional space. This is omitted here in the interest of brevity but found in [33, 34].

3.2 Dynamic Properties of VQ and SOM

Although the SOM is described and implemented in such an elementary manner it has extremely elegant properties. A collection of these is found in [34, 57].

First of all Kohonen showed that the resulting map forms a Voronoi tessellation of a given input space which is a kind of VQ partitioning of the input space into nonoverlapping regions [34]. Furthermore, apart from these tessellations, the neurons also *ultimately* collectively possess the stochastic properties of the underlying data points. Indeed, if the width of the neighborhood is fixed and $\alpha(t)$ is very small Ritter [48] showed that for the scalar case, the density of weight vectors $G(Y)$ is proportional to $F(X)^\gamma$ where $F(X)$ is the distribution of the input vectors, $\mathcal{P}$, and

$$\gamma = ((W+1)^2/3)/((W+1)^2 + W^2), \tag{3}$$

where again, $[-W, W]$ specifies the activation bubble. To the best of our knowledge, the analogous results for vectors of higher dimensions are presently unknown.

The situation is more hopeful in the case of VQ in its purest form. Zador [59] showed that the Linde, Buzo and Gray algorithm [37] for VQ (i.e., the Kohonen map for $W = 0$) produces neurons with asymptotic density $G(Y)$ proportional to $F(X)^{n/(n+d)}$ where n is the dimension of the input space and the $L_d - norm$ is used to measure the distortion between the neurons and the input vectors. Moreover, when $n = 1$ and $d = 2$, these two asymptotic distributions become the same, namely, $G(Y) \propto F(X)^{1/3}$.

By considering the effects of the SOM in the transformed domain Wong [57] argues that

$$\overline{Y}(t+1)(\omega) = (1-\alpha)\overline{Y}(t)(\omega) + \alpha X(t)\delta(\omega), \tag{4}$$

where $\delta(\cdot)$ is the delta function and $\overline{Y}$ is the Fourier transform of Y. Thus, all frequency components in $Y(t)$ are scaled by $1-\alpha$ while its d.c. component is increased by $\alpha X(t)$. If the same updating is repeatedly applied to the weights in the activation bubble all the weights would be equal. Of course, since the input vectors are presented randomly, the net effect is that the neighboring neurons have their weights as similar as possible. Thus, the Kohonen updating rule also smooths the weights. Indeed, the dynamics of the Kohonen map can be seen to consist of two components - the nearest-neighbor component achieved in a non-parametric manner, and a smoothing effect obtained as a consequence of the windowing effect. Whereas the clustering component tends to gather the neurons' weights, the smoothing keeps the spacing among the weights. Notice that as the neighborhood $[-W, W]$ decreases, the width of the smoothing effect decreases and the clustering effect increases. Fine tuning of the map is a result of the changing balance between these two effects. However, if W is progressively decreased [33, 34], initially all the neurons tend to coalesce close to the mean of the overall distribution when W is large, and subsequently unwind to find their relative places statistically and topologically as W is progressively decreased.

The SOM has been used in a variety of applications. In statistical pattern recognition it has been used in the recognition of Finnish and Japanese speech [30, 31, 32], sentence understanding [53], in classification of sea-ice [43] and even in the classification of insect courtship songs [42]. From a hardware point of view the SOM has been used in the design of algorithms, which at the lowest level can control the production of semi-conductor substrates [39, 55], and at a higher level the synthesis of digital systems [25].

The beauty of the 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 particularly important for problems which involve two and three-dimensional **physical** spaces, and is indeed, the principal motivation for the SOM being used in path planning and obstacle avoidance in Robotics [22, 23, 41, 49, 50, 51].

3.3 Pure Kohonen Network Solutions to the TSP

The Kohonen Network (described in detail earlier) in its virgin form has been used to solve the TSP [18, 27, 52]. The idea that is used is essentially 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 then presented in this order to the NN during several epochs. At every instant one city X_i from $\mathcal{P}$ is presented to the NN. The neurons now compete and the closest one to X_i, Y_{j*}, is determined. This neuron and a group of neurons within its activation bubble, B_{j*}, are now moved in the direction of X_i as per equation (10). As mentioned earlier, the set B_{j*} can consist of a subset of neurons from the neighborhood of Y_{j*}. 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*} is the inclusion of all neurons. In such situation 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 drawn 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. Experimental results clarifying this are given in [2]. These drawbacks have motivated other researchers to consider improvements on the scheme and these are presented in the next few subsections.

3.4 The Guilty Net Solution to the TSP

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 winning neuron to the various neurons. In [8], Burke *et al.* modified the above strategy in their "Guilty Net" (GN) algorithm, by introducing two modifications. The first modification involved incorporating a penalty term to the distance function for determining the "winning neuron". The second modification involved specifying an alternate method for the neighborhood function which determined the strength by which the weights of the neighbor neurons are modified.

Just like in the pure Kohonen scheme, the number of neurons is fixed (Burke *et al.* recommend that it be equal to the number of cities) and remains the same throughout the algorithm [8]. 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 one or more cities on the same neuron by using a "conscience mechanism". 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:

$$j^* = \arg\min_j \left\{ \|X_i - Y_j\| + \lambda \cdot \frac{win_j}{1 + \sum_k win_k} \right\} \tag{5}$$

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 X_i — besides this distance, the number of times a particular neuron has won must also be taken into consideration.

The neighborhood function in this method is defined as

$$\Lambda(j, j^*) = \begin{cases} (1 - d_{jj^*}/L)^\beta, & \text{if } d_{jj^*} < L \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $d_{jj^*} = \min\{|j - j^*|, M - |j - j^*|\}$ and $L = N/2$. Notice that this distance function is identical to the Gaussian kernel except that, from a probabilistic point of view, it is more of a Geometric type - which seems to be a reasonable kernel to be used when the coordinates are integers.

It should be noted that $\Lambda(j, j^*)$ is equal to unity for the winning neuron j^*, and its value decreases as d_{jj^*} increases during one pass or cycle (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 epoch. This means that the neurons have a large neighborhood at the beginning of the procedure which gets smaller from epoch to epoch.

Reported computational results indicate that the guilty net approach performs better than the Hopfield-Tank model, but it is inferior to the elastic net [45]. 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 [5]. 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.* presented in the next subsection [5]. The weaknesses of guilty net have also been pointed by Burke later on [6] and improved recently [7].

3.5 The Procedure of Angéniol, Vaubois, and Le Texier

The procedure introduced by Angéniol *et al.* [5] is different from the guilty net in two aspects. The first difference is that the weight update equation does not have a fractional learning rate parameter. The neighborhood function is the Gaussian kernel defined as:

$$\Lambda(j, j^*) = \frac{1}{\sqrt{2}} \exp\left(-\frac{d_{jj^*}^2}{\sigma^2}\right) \tag{7}$$

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.,

$$\sigma \leftarrow K_\sigma \sigma, \qquad 0 < K_\sigma < 1.$$

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 nodes 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 one more iteration. A neuron is deleted if it does not win a competion 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, the authors solved the 30-city problem taken from Hopfield and Tank [26] and the five sets of 50-city problems taken from Durbin and Willshaw [16]. According to their simulation results, the algorithm found on the average solutions within 3 % 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, 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 Potvin [45], and to Section 7.2 of Johnson and McGeoch [28]. In addition, it is worth mentioning recent works by Budinich [9], and by Budinich and Rosario [10]. Their neighborhood function considers not only the distances between the neurons on the ring (i.e., unlike the d_{jj*} of formula (6)) but also their physical distances (i.e., a function of the coordinates, such as the Euclidean distance).

4 The Guilty Net Solution to the Hamiltonian Path Problem

Jeffries and Niznik [29] recently presented a scheme by which they computed the minimum cost Hamiltonian *path* on the plane. They did this by extending the basic ideas of the guilty net (GN). Their strategy is as follows.

A regression line going through the cities (i.e., data points) is first computed. The cities are then projected on to this line. The locations of these projections form the initial starting coordinates of the neurons. The algorithm then invokes the GN [8] due to Burke and Damany, and repeatedly presents the cities to the scheme and migrates the neurons till they have reached their final positions. As we explained earlier, the GN does not necessarily converge, and so if (or rather 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 GN algorithm, the neuron is deleted. Similarly, if multiple neurons are associated with the same city, the neuron furthermost to it is 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, Jeffries and Niznik observed that the resulting solution often contained edge-crossings, and these were removed by invoking a 2-OPT-like crossing elimination strategy.

By using this simple strategy, Jeffries and Niznik [29] claim that they can yield near optimal Hamiltonian path and demonstrate their scheme for a simple data set consisting of 22 cities. The results are fairly impressive. The reader should observe that their scheme *does not solve the HPP* since the end points of their path is not specified and fixed. In other words their algorithm tries to compute the shortest Hamiltonian Path between all possible city pair. 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 GN - it is not guaranteed to converge except after using the above-mentioned Angéniol *et al.*'s like insertion/deletion steps.

We have mentioned this scheme only in the interest of completeness, because it is, in one sense the only reported neural method that moves in the direction of solving the HPP without resorting to the underlying solution of the TSP. We shall now present new self-organizing NN approaches which indeed yield good approximate solutions to the HPP itself.

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

The first contribution of this paper is to use the SOM (as suggested by the results of Angéniol *et al.* [5]) to yield a solution to the HPP. We shall refer to this solution as GSOM_HPP : a Generalized SOM Solution to the Euclidean Hamiltonian Path Problem. Observe that as mentioned earlier, the method due to Angéniol *et al.* [5] differs from the GN by using a Gaussian kernel defined (7) in which σ (the standard deviation) is called the "gain parameter" and d_{jj^*} is defined as before. Also, as mentioned earlier, the value of σ is decreased at the end of each complete pass, namely epoch, by a factor K_σ (a user-defined constant) as:

$$\sigma \leftarrow K_\sigma \sigma, \qquad 0 < K_\sigma < 1.$$

The significance of the variation of σ was explained earlier.

More importantly, the second and primary difference between Angéniol *et al.*'s approach and the GN is the fact that, as explained earlier, this scheme permits creation and deletion of neurons as the algorithm proceeds.

The way by which we extend this algorithm to solve the HPP is as follows. The algorithm is initialized by using M neurons, $\{Y_j(0) : 1 \leq j \leq M\}$, where M is a user specified input parameter. Two of these neurons, Y_1 and Y_M are located at the coordinates of X_s and X_t, the starting and terminal neurons respectively, and are called the *anchor* neurons. The locations of the other $M-2$ neurons, $\{Y_j(0) : 2 \leq j \leq M-1\}$, are on the straight line joining X_s and X_t, and is equally spaced on this line. Thus,

$$Y_j(0) = X_s + (j-1) \cdot \frac{(X_t - X_s)}{M-1} \qquad j = 2, \ldots, M-1 \tag{8}$$

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 (not the ones 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 X_s and X_t are not migrated even if they become winners upon presentation of some city 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 complete passes, it is deleted, and if a neuron wins a competition twice in the same epoch a new neuron is created at the same position. Unlike the deletion operation, the anchor neurons do participate in the *creation* process, but under slightly different conditions. Thus, if the winning neuron is one of the anchor neurons, (Y_1 or Y_M) a new neuron is created instantaneously at the same location. Unlike the anchor neuron, however, the newly created is itself permitted to migrate.

When the algorithm terminates the number of neurons can be larger than the number of cities because either the starting and terminal cities or both can be represented by more than one neuron located exactly on it at the final configuration (i.e., more than two anchor neurons). Observe that, since this multiple representation in the final configuration occurs exactly on the starting and terminal cities the length of the final Hamiltonian Path is not affected. Indeed, we have observed multiple neurons representing only the terminal city during our experiments.

Experimental results demonstrating the power of GSOM_HPP are presented in a subsequent section where we shall discuss, in detail, the results obtained using various techniques.

6 KNIES_HPP : A KNIES Solution to the Hamiltonian Path Problem

Although the overall effects of using SOM in any pattern recognition (and distribution learning) problem is that the neurons ultimately obey the distribution of the original data 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 both VQ and SOM completely ignore the statistical information that is already collectively resident in the data points. As introduced in [2], the Kohonen Network Incorporating Explicit Statistics (KNIES) explicitly utilizes this information. Thus, 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. Apart from only using the information in the data point presented, KNIES uses these pieces of statistical information as well. In other words, at every iteration, KNIES utilizes both the local information present in the point presented and also the global information latent in the entire data set, $\mathcal{P}$. We shall *briefly* describe how this is achieved below - the reader is referred to [2] for more specific details of the scheme.

Let us suppose that the mean of the set, $\mathcal{P}$, of data points to be represented by the neurons is μ. The intention is that we maintain μ 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 μ. To do this, as alluded to earlier, we shall decompose every training step into two phases, namely the **attracting** and **dispersing** phases.

When the data point X_i is presented to the network, the closest neuron and all the neurons within the activation bubble are migrated using an equation identical to (1). This constitutes the attracting module, and in this phase the neurons within the bubble use the local information present in the city coordinates and move towards it.

Observe that since in KNIES the neurons are partitioned as the ones which participate in the attracting phase and those which do not, we assume that the activation bubble is maintained constant for a specified value of M. The question of decreasing the effect of the attraction for the neurons in B_{j*} as its lateral distance from j^* increases, *must necessarily* be achieved by using a kernel function. For the sake of simplicity we assume that the kernel is the Gaussian kernel [15, 19] given by :

$$G(a,b) = K_g \cdot e^{\frac{-\|Y_a - Y_b\|^2}{2\sigma^2}}, \tag{9}$$

where K_g is a normalizing constant, σ specifies the standart deviation of the kernel which is progressively decremented (for example, linearly [5, 57]) to get the effect of reducing the activation bubble, and Y_a and Y_b are the coordinates of the neurons in question. Thus the attraction phase obeys :

$$Y_j(t+1) = \begin{cases} Y_j(t) + G(j,j^*)(X_i - Y_j(t)) & \text{if } j \in B_{j*}(t) \\ Y_j(t) & \text{otherwise,} \end{cases} \tag{10}$$

where X_i is the i th data point presented to the system and j^* is the winning neuron closest (using the Euclidean norm) to X_i.

Recall that, for the TSP, Angéniol *et al.* [5] have recommended using the Gaussian kernel given with (7). Note that it is a function of the *indices* of the neurons — as opposed to being a function of their coordinates — and of the variance. Besides the normalizing constant is set to $1/\sqrt{2}$. The advantage of such a kernel function is that a single parameter σ, can take care of both the activation bubble and their intensity of attraction.

As a result of the migration of the neurons within the activation bubble we observe that the mean of all neurons moves away from the initially assigned value, μ. Thus, the dispersing phase now forces the neurons outside the activation 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 effected as a result of the attraction module is the vector $\Delta Y(t)$, where,

$$\Delta Y(t) = \sum_{j \in B_{j*}(t)} \left[Y_j(t+1) - Y_j(t) \right]. \tag{11}$$

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

$$Y_j(t+1) = \; Y_j(t) - \frac{1}{M(t)-|B_{j*}(t)|}\Delta Y(t), \quad if\ j \notin B_{j*}(t) \tag{12}$$

Notice that in (12) the change $\Delta Y(t)$ 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") we have opted to distribute the change uniformly. It is easy to see that as a result of both (10) and (12) the attracted neurons move towards X_i, and the net result of both phases retains the mean of the neurons to be μ, as desired. A formal description of the algorithm and its application to the TSP are given in [2].

6.1 Using KNIES to solve the HPP

As mentioned in [2], KNIES by itself would not be too powerful in solving problems for which the SOM suffices. This is because the distribution of the neurons will ultimately follow the distribution of the cities whether the global statistical properties are utilized or not. However, in the case of the TSP, KNIES is a useful scheme because the *quality* of the final solution depends on the *transient* stochastic distribution of the neurons. Experimental results shown in [2] demonstrate that the new scheme, KNIES_TSP, is a very accurate neural solution to the TSP. We shall adapt now KNIES for the HPP.

First of all we shall consider the initialization process of 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 X_s and X_t, the starting and terminal cities respectively, and are the *anchor* neurons. The remaining neurons, namely $\{Y_j(t) : 2 \le j \le M-1\}$, are first initialized according to equation (8) 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 an invariant. This is done as follows. If $\overline{X}$ is the mean of the coordinates of the cities, and $\overline{Y}(0)$ is the mean of the coordinates of the neurons, $Y_j(0)$ is set as :

$$Y_j(0) = Y_j(0) + \frac{M}{M-2}\left(\overline{X} - \overline{Y}(0)\right) \qquad 2 \le j \le M-1. \tag{13}$$

It is easy to see that as a result of the above assignment, the new $\{Y_j(0) : 2 \le j \le M-1\}$ are all located on the same side of the line joining X_s and 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 [29], 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.

To show how this is done, first of all we assume that the number of neurons at any time is $M(t)$, and that this number can be increased or decreased by the insertion or deletion of neurons. At any instant 't' we say that a neuron j located at $Y_j(t)$ is associated with a city $\Gamma_j(t)$ if :

$$\Gamma_j(t) = \min_i \|X_i - Y_j(t)\| \qquad j = 1, \ldots, M(t). \tag{14}$$

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

At every time instant a city X_i is presented to the network. The neuron j^* closest to X_i is now determined as the winner and both this neuron as well as all its neighbors in the activation bubble are moved toward X_i with a strength determined by an appropriate kernel function. In our experiments we have used the Gaussian kernel with a standart deviation σ. However, rather than using the kernel associated with the coordinates of the neurons, we have used the kernel associated with the indices as described by Λ as in (7) because decrementing σ takes care of decreasing the width of activation bubble and also of the adaptation constant, $\alpha(t)$.

The dispersing phase is as follows: Let $X_{total}(t)$ be the vector sum of the coordinates of the cities represented by the neurons and $Y_{total}(t)$ 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 activation bubble. 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* $\{X_{\Gamma_j(t)} : j = 1, \ldots, M(t)\}$ of the set $\mathcal{P}$. It can be easily verified that this is achieved by updating :

$$Y_j(t) = Y_j(t) + \frac{X_{total}(t) - Y_{total}(t)}{L(t) + R(t)} \qquad j \notin B_{j^*}(t). \tag{15}$$

The above is actually quite amazing, because we 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.

Observe now that the number of neurons $M(t)$ changes with time, t. Insertions and deletions are permitted in a manner analogous to the scheme by Angéniol *et al.* [5]. If a neuron wins the competition for two different cities in the same pass, a new neuron is created with the same coordinates as the winner neuron. This newly created neuron is inserted into the line as the neighbor of the winner. Both neurons are inhibited until they disperse by the movements of the neighboring units. In the same way, a neuron is deleted, if it does not win a competition during a fixed number of (say, three) complete passes.

The strategy for the anchor neurons is slightly different. If the winning neuron is one of the anchor neurons, ($Y_1(t)$ or $Y_M(t)$) a new neuron is created instantaneously at the same location. Unlike the anchor neurons, however, the newly created one is permitted to migrate.

As in the case of the TSP, the difference between our insertion/deletion processes in GSOM_HPP and the corresponding processes in Angéniol *et al.*'s scheme is quite subtle. Consider the situation when a new neuron, k, is inserted because a neuron l, wins the competition twice. Thus $Y_k(t)$ is exactly the same as $Y_l(t)$ since they are 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 band 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. In the same way, a neuron when deleted will no more be on the band, and so its coordinates disappear when we attempt to make the mean of the neurons coincide with the mean of the cities they represent. Based on the above, one can argue that, in one sense, our 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 formal algorithm, **Algorithm** KNIES_HPP, included in the interest of completeness, is given on the next page.

6.2 KNIES_HPP_Global : A simplification of KNIES_HPP

KNIES_HPP 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. 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 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 $\Delta Y(t)$ 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 :

```
For all $j \notin B_{j*}(t)$ Do
    $Y_j(t) \leftarrow Y_j(t) - \frac{1}{M(t)\cdot(M(t)-|B_{j*}(t)|)}\Delta Y(t)$
EndFor
```

The above loop replaces their respective steps in KNIES_HPP to yield KNIES_HPP_Global. This procedure has also been implemented and tested. It is computationally faster than KNIES_HPP since it does not have to identify the cities associated with the neurons currently on the band after the presentation of each city. It is also quite accurate even though (in its transient behavior) it is theoretically a deviation from the principle motivating KNIES.

```
Algorithm KNIES_HPP
 Input       : The coordinates of the $N$ cities $\{X_i : 1 \leq i \leq N\}, X_s$ and $X_t$
 Output      : A solution to the Euclidean HPP.
 Parameters  : $W$, the width determining the window, $B_{j*}$,$\sigma$, the standart deviation of the Gaussian kernel
               decremented as in KNIES ; $M$ is the initial number of neurons on the starting line.
 Notation    : Won is an array which records the number of times neuron $j$ wins the competition
               in any epoch. It is set to zero at the beginning of each epoch.
               Insert($j$) inserts a neuron at the same location as neuron $j$ which is indexed as $j+1$.
               Inhibit($j$) is a Boolean array which indicates that neuron $j$ is inhibited or not.
               Delete($j$) deletes neuron $j$ from the current set of neurons.
Method
Begin
    Initialize $M(0)$ with $M$ and the location of the $M(0)$ neurons on a piecewise linear line as :
    $Y_1(0) = X_s$
    $Y_M(0) = X_t$
    For $j = 2, \ldots, M-1$ Do
        $Y_j(0) = X_s + (j-1) \cdot \frac{(X_t - X_s)}{M-1}$
    EndFor
    For $j = 2, \ldots, M-1$ Do
        $Y_j(0) = Y_j(0) + \frac{M}{M-2}\left(\overline{X} - \overline{Y}(0)\right)$
    EndFor
    $t = 0$
    Repeat
        For $j = 1, \ldots, M(t)$ Do
            Won($j$) =0
            Inhibit($j$) = False
        EndFor
        For every point $X_i \in \mathcal{P}$ Do
            $j^* \leftarrow \text{argmin}_j \|X_i - Y_j(t)\|$
            If (($j^* = 1$) or ($j^* = M(t)$)) Then
                MustInsert $\leftarrow$ True
            EndIf
            If (Inhibit($j^*$)) and
                        (neuron $j^*$ overlaps with its inhibited neighbor) Then
                go to jump
            EndIf
            If (Won($j^*$) = 2) Then
                MustInsert $\leftarrow$ True
                Won($j^*$) = 0
            EndIf
            For all $j \in B_{j*}(t)$ Do
                $\Delta Y_j(t) \leftarrow \Lambda(j, j^*)(X_i - Y_j(t))$, with $\Lambda$ defined by (7)
                $Y_j(t) \leftarrow Y_j(t) + \Delta Y_j(t)$
            EndFor
            $X_{total}(t) = \sum_j X_{\Gamma_j(t)}$
            $Y_{total}(t) = \sum_j Y_j(t)$
            $\Delta Y(t) = X_{total}(t) - Y_{total}(t)$
            For all $j \notin B_{j*}(t)$ Do
                $Y_j(t) \leftarrow Y_j(t) + \frac{1}{M(t) - |B_{j*}(t)|} \Delta Y(t)$
            EndFor
            If (MustInsert) Then
                Insert($j^*$)
```

```
                    $M(t) \leftarrow M(t)+1$
                    If $(j \neq 1)$ and $(j \neq M(t))$ Then
                        Inhibit($j^*$) $\leftarrow$ True
                        Inhibit($j^*+1$) $\leftarrow$ True
                    EndIf
                EndIf
jump:       EndFor
            For $j = 2, \ldots, M(t)-1$ Do
                If ($j$ has not won for three consecutive epochs) and ($j \neq 0$)Then
                    Delete($j$)
                    $M(t+1) \leftarrow M(t)-1$
                EndIf
            EndFor
            Decrement $\sigma$
            $t \leftarrow t+1$
        Until Satisfied
        HPP_Path $\leftarrow$ Path joining the cities in the order of their location on the final path
    END Algorithm KNIES_HPP
```

7 Measuring the Performance of the New Heuristics

The first major problem we encounter when we test our neural HPP algorithms is to determine their accuracy. Indeed, this is because unlike the TSP, there does not exist a standard test library with known optimal solutions. Although it is possible to transform a TSP instance with a known optimal solution (i.e., an optimal tour) into an HPP instance with an optimal path whose length is equal to the length of an optimal tour [4], any such transformation is useful for methods working with intercity distances rather than the city coordinates; and this is not the case with GSOM_HPP, KNIES_HPP and KNIES_HPP_Global as well as any other Kohonen type SOM method which solves the HPP and TSP. We are therefore forced to resort to statistical means to justify the accuracy of the solution. To do this we consider the classical result due to Fisher and Tippett [17] which has been adapted by various authors [14, 20, 21, 38] to compare the efficiency of experimental results in combinatorial optimization.

7.1 Confidence Intervals for the Length of an Optimal Hamiltonian Path

Consider a set of S independent samples $\Sigma_1, \Sigma_2, \ldots, \Sigma_S$each of size m drawn from a parent population which is both continuous and bounded from below by a. If x_i is the smallest value in sample i, and we let

$$v = min\{x_i : 1 \leq i \leq S\}, \tag{16}$$

then, as m gets larger the distribution for x_i converges to a Weibull distribution with location parameter a [17]. Consequently, by arriving at S independent estimates (each of which is a local optimal value) of the length of an optimal Hamiltonian path, we can argue that :

$$Pr\left[v - b \leq a \leq v\right] \approx 1 - e^{-S} \tag{17}$$

where S is the sample size, and a and b are the location and scale parameters of the Weibull distribution.

This is exactly the approach introduced first by Golden and Alt [20], and applied and tested for the TSP by Golden and Stewart [21] later. Los and Lardinois suggest two modifications on this approach [38]. First, instead of using S local optima, only a subset of size $S' \leq S$ *distinct* local optima and their values $x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(S')}$ are used to fit a Weibull distribution. The reason for this modification is the fact that the Fisher - Tippett theorem assumes the independence of S samples [17]; nevertheless having identical local optima in the sets $\Sigma_1, \Sigma_2, \ldots, \Sigma_S$ is equivalent to repeating the same sample several times. Second, allowing T to be any real number, they then developed the following formula :

$$Pr\left[v - b/T \leq a \leq v\right] \approx 1 - e^{-S'/T^c} \tag{18}$$

The main advantage of this formula over the one suggested by Golden and Alt (i.e. formula (17)) is its explicit dependence on the level of confidence. In other words, $100(1-\alpha)\%$ confidence can be achieved by letting

$$T = \left(-\frac{S'}{\ln \alpha}\right)^{1/c}. \tag{19}$$

The confidence level is fixed to $100(1-e^{-S})\%$ in (17). However, there is a specific problem with the Los and Lardinois formula: it involves the shape parameter c, which is not the case for (17). According to the computational experiments reported in the literature, the computation of good estimators (e.g. maximum likelihood estimators (MLE)) for three-parameter Weibull distribution is not a trivial task [60]. The difficulty is mainly due to the location parameter a. Once its value is determined the estimation of the other parameters (the scale parameter b and the shape parameter c) becomes fairly easy.

In addition to MLEs some non-MLEs are also suggested for the shape parameter c of the Weibull distribution [11, 13, 61]. These estimators are tested by Derigs while computing confidence limits for the global optimum of the TSP and the quadratic assignment problem (QAP) [14]. Based on the results he advocates the use of Golden and Alt formula (i.e., formula (17)) with the following estimates of the shape and scale parameters :

$$\widehat{c} = \left(x_{(1)} \cdot x_{(S)} - x_{(2)}^2\right) / \left(x_{(1)} + x_{(S)} - 2x_{(2)}\right) \tag{20}$$

$$\widehat{b} = x_{(\lfloor 0.63S \rfloor + 1)} - \widehat{c}. \tag{21}$$

These estimates are introduced by Zanakis [61]. Here $x_{(1)}$, $x_{(2)}$ and $x_{(S)}$ are respectively the smallest, the second smallest and the largest of S values. The main argument was that the lower confidence limits obtained by using these non-MLEs with Golden and Alt formula on his test problems were that the greatest lower bounds on the length of optimal TSP tours (the gap between the lower bound and the length of an optimal) is the smallest. This was not the case for Los and Lardinois interval (i.e. formula (18)) : it failed to give confidence intervals which cover an optimal value significantly more often than expected.

Given the intercity distances and endpoints of an Hamiltonian path we can assign an arbitraily small (i.e., $-\infty$) distance between the first and last cities and use one of the TSP heuristics with random initial conditions to generate S local optimal values. Note that, if we exclude the artificial distance, the sum of the distances associated with the arcs on the Hamiltonian path computed by one of the TSP heuristics can be used for estimating a confidence interval for the length of an optimal Hamiltonian path. In this work, being convinced by Derigs' results, we use Golden and Alt formula with non-MLEs (20) and (21). Because of the reasons stated above, we agree that we can get good interval estimates for the length of an optimal Euclidean Hamiltonian path which we can use for measuring the performance of GSOM_HPP, KNIES_HPP and KNIES_HPP_Global.

Summing up the above arguments, in a given TSPLIB instance, we first randomly chose two cities as the starting and terminal points of a Hamiltonian path. Then we assign an arbitraily small intercity distance for them and run one of the known TSP heuristics many times with different initial conditions. Each run produces a local optimum, namely a near optimal Hamiltonian path whose length corresponds to one of the S sample values (i.e., $x_i : 1 \leq i \leq S$) used in the estimation of $100(1-e^{-S})\%$ confidence intervals for the length of an optimal Euclidean Hamiltonian path. Note that once the endpoints are randomly selected they remain unchanged during the replications.

7.2 Verification of the Interval Estimates

A typical quality measure for a heuristic solution is the relative deviation of its value from the value of an optimal solution. Using such a measure the average of the relative errors obtained on a standard test library can be utilized as a good experimental performance measure for the heuristic: the smaller it is, the better the performance of the heuristic. We follow this approach with one exception; we use the lower limit of the interval estimate instead of the length of an optimal Hamiltonian path to determine relative deviations. The essential valid reason for the use of lower confidence limits is the fact that optimal Hamiltonian paths are unavailable and very expensive to compute. Furthermore, it is clear that the estimation strategy is definitely more robust if the optimal value remains within the confidence interval.

Table 1: Verification of the statistical method for procuring confidence intervals for the length of optimal Hamiltonian Paths between X_s and X_t. Cases where the optimal value lies in the interval are marked with a '*'.

Instance	Endpoints (X_s, X_t)	Opt. Path Length	NNb	NI	FI
att532	(112, 96)	27408.88	[27982.03, 28359.97] 28701.49	[29897.31, 30377.94] 30763.05	[28361.04, 28946.83] 29527.43
berlin52	(24, 48)	7528.55	[7318.04, 7643.15] * 7865.05	[7815.91, 8195.21] 8559.92	[6942.44, 7528.55] * 7977.02
eil51	(41, 13)	420.55	[416.41, 424.41] * 433.65	[417.4, 437.3] * 451.16	[399.50, 420.81] * 438.90
eil76	(11, 53)	537.39	[525.14, 543.75] * 560.89	[565.49, 583.07] 598.77	[523.23, 551.16] * 574.30
eil101	(54, 55)	633.83	[635.51, 651.09] 663.61	[665.60, 677.40] 694.02	[630.72, 653.25] 673.53
kroA100	(34, 83)	21139.47	[20755.17, 21249.41] * 21647.81	[20973.48, 22432.01] * 23643.20	[20658.88, 21484.37] * 22244.06
lin105	(87, 66)	13999.78	[13752.08, 14155.66] * 14473.51	[13758.57, 14568.89] * 15082.25	[13462.44, 14211.07] * 14841.26
pcb442	(226, 411)	50591.51	[51406.66, 52378.21] 53183.72	[54252.59, 54830.73] 55306.98	[53296.61, 55108.42] 56579.12

To verify the above we test this assumption before using interval estimates, and for this purpose we consider some of the TSPLIB instances whose optimum tours are available. First we randomly select an edge on the given optimum tour, make its length arbitrarily small and treat the remaining edges of the tour as an optimal Hamiltonian path. The length of this path is simply the length of the optimum tour less the original length of the randomly selected edge, and its endpoints are the edge's endpoints. Then we compute confidence intervals by using one of the known TSP heuristics with the modified instance (the length of the randomly selected edge is very small and this guarantes their endpoints to be adjacent in a tour). To achieve this we have used six TSP heuristics : *cheapest insert* (CI), *nearest* neighbor (NNb), *Clark-Wright savings method* (CW), *random insert* (RI), *farthest insert* (FI) and *nearest insert* (NI). Detailed information on the performances of these heuristics can be found in an excellent work [47]. They are not only easy to implement but also the ones used in most of the previous studies [20, 21]. Except for the Clark-Wright savings method, we further performed 2-OPT on the solution generated by the heuristics for further improvement as advised in the literature [14, 21]. All of the TSP heuristics we performed for the same endpoints (i.e., endpoints of the randomly selected edge) for fifty different initializations. In other words, 50 values, generated by each one of these heuristics, are used in the estimation of the shape and scale parameters of the Weibull distribution according to formulae (20) and (21). The estimates are then employed in establishing confidence limits for the length of an optimal Hamiltonian path according to (16). Note that six heuristics yielded six samples of size 50, whence we obtained six different confidence intervals. Since, as suggested by Golden and Stewart [21] $S = 50$ it implies almost 100 % confidence since for $S = 50$, $1 - e^{-S}$ is practically unity.

The results as well as the test bed for the verification experiments are summarized in Table 1 and Table 2. The first column contains references for the TSPLIB instances. The corresponding optimum tour lengths can be found in [46]. The pairs listed in the second column are the endpoints of the randomly selected edge whose length is made arbitrarily small and represent the starting and terminal cities of an optimal Hamiltonian path. The length of an optimal Hamiltonian path between these endpoints, which is obtained by substracting the length of the selected edge from the length of the optimum tour, is given in the third column. The starting and terminal cities are represented by their TSPLIB sequence number. The last three columns consist of the confidence intervals computed according to formula (17) as described in the previous section. As it can be observed, the optimal path length of the instances (column 3) is covered by 27 of the 48 interval estimates (columns 4 – 6). Successful intervals are marked with "$*$". Note that, for the confidence intervals computed with NNb, the optimal values which are not included are almost equal to the lower limit.

Table 2: Verification of the statistical method for procuring confidence intervals for the length of optimal Hamiltonian Paths between X_s and X_t. Cases where the optimal value lies in the interval are marked with a '*'.

Instance	Endpoints (X_s, X_t)	Opt. Path Length	CI	RI	CW
att532	(112, 96)	27408.88	[29112.36, 29880.26] 30545.77	[28344.20, 29056.11] 29668.59	[28854.35, 29585.69] 30220.21
berlin52	(24, 48)	7528.55	[7369.04, 7825.38] * 8247.02	[6787.68, 7528.55] * 8134.99	[7315.33, 7802.52] * 8213.32
eil51	(41, 13)	420.55	[400.02, 425.17] * 447.30	[414.51, 429.37] * 443.13	[403.77, 424.82] * 442.81
eil76	(11, 53)	537.39	[516.25, 551.94] * 581.56	[540.28, 560.23] 577.89	[529.48, 556.29] * 579.53
eil101	(54, 55)	633.83	[643.27, 664.32] 682.47	[616.88, 649.96] * 677.04	[634.43, 661.94] 684.88
kroA100	(34, 83)	21139.47	[20105.87, 22047.52] * 23617.26	[19968.92, 20164.70] 22187.48	[19718.48, 21480.60] * 22928.
lin105	(87, 66)	13999.78	[13194.70, 14343.56] * 15307.39	[13134.91, 14727.06] * 15157.35	[13729.92, 14433.60] * 15051.33
pcb442	(226, 411)	50591.51	[51927.08, 54248.71] 56091.78	[53599.85, 55162.32] 56480.68	[52382.87, 545117.87] 55593.77

Another interesting observation is the effect of the heuristics used in the generation of these intervals. All the heuristics performed equally well except the nearest insertion heuristic (column 5 in Table 1 and Table 2). Columns 4 - 6 of Table 1 and Table 2 contain one additional number under the intervals : it designates the average length of 50 different Hamiltonian paths computed with the corresponding heuristic. If we measure the performance of a heuristic by the average values, it is reasonable to assign the one with the smallest average as the best. Thus the nearest neighbor followed by 2-OPT (NNb) is the winner for all test instances. Note that 5 out of 8 intervals generated by NNb includes the optimal value listed in column (3). Therefore, the use of lower confidence limits in measuring the performance of our HPP heuristics is indeed a good approach, as justified for the TSP and QAP.

Table 3: Confidence intervals for the length of optimal Hamiltonian Paths between X_s and X_t.

Instance	**Endpoints** (X_s, X_t)	**NNb**	**NI**	**FI**
att532	(239, 451)	[28054.48, 28486.41] 28904.50	[29347.82, 29686.19] 30025.90	[27949, 28666.18] 29264.74
berlin52	(10, 31)	[7394.56, 7549.32] 7675.93	[7254.99, 7699.84] 7977.14	[7096.24, 7504.68] 7844.28
bier127	(14, 122)	[114970.39, 119102.93] 122972.70	[126551.31, 127988.16] 129220.59	[115506.66, 120347.06] 1243553.15
eil51	(5, 14)	[405.69, 422.95] 437.67	[400.13, 427.01] 451.79	[422.36, 430.90] 439.65
eil76	(60, 38)	[526.20, 548.07] 566.28	[560.02, 576.06] 590.07	[527.06, 555.70] 579.16
eil101	(7, 50)	[631.69, 649.28] 663.71	[634.38, 663.99] 687.67	[605.86, 639.24] 668.59
kroA100	(239, 451)	[20400.44, 21124.34] 21692.45	[21425.49, 22558.40] 23270.29	[19884.22, 21084.09] 22202.97
kroA200	(65, 134)	[28878.35, 29473.94] 30114.63	[32636.36, 33081.95] 33447.02	[29163.98, 30197.34] 31103.06
lin105	(37, 96)	[13885.61, 14338.09] 14814.50	[14199.53, 14442.39] 14728.55	[14088.46, 14529.35] 14858.00
pcb442	(315, 169)	[51315.36, 52195.61] 52947.12	[53860.96, 55416.70] 56689.56	[51761.16, 53771.12] 55585.26
pr107	(22, 91)	[39860.24, 40213.38] 40719.66	[40912.53, 41907.45] 42710.16	[39577.44, 40190.95] 40673.17
pr124	(8, 76)	[57309.63, 58823.04] 60338.93	[60665.69, 61191.00] 61605.94	[56243.79, 59101.27] 61514.62
pr136	(40, 105)	[100272.44, 102345.32] 104171.05	[97836.82, 100201.46] 102432.70	[97235.45, 101667.10] 103521.56
pr152	(103, 2)	[67171.17, 67701.35] 71765.97	[72013.72, 73121.88] 73941.44	[65791.64, 67701.35] 69297.67
rat195	(166, 128)	[2293.74, 2330.87] 2366.34	[2480.62, 2521.24] 2557.77	[2386.37, 2471.97] 2543.19
rd100	(26, 73)	[7685.11, 7996.26] 8289.27	[8343.94, 8497.79] 8677.11	[7823.07, 8157.85] 8455.82
st70	(7, 43)	[634.58, 671.35] 701.11	[684.37, 707.87] 720.42	[644.46, 677.09] 705.93

Table 4: Confidence intervals for the length of optimal Hamiltonian Paths between X_s and X_t.

Instance	**Endpoints** (X_s, X_t)	**CI**	**RI**	**CW**
`att532`	(239, 451)	[28481.55, 29407.97] 30172.52	[28085.37, 28859.65] 29473.89	[28996.04, 29698.70] 30308.43
`berlin52`	(10, 31)	[7242.09, 7558.88] 7868.82	[6975.84, 7504.68] 7956.14	[7167.17, 7686.31] 8109.39
`bier127`	(14, 122)	[113237.42, 122426.45] 131539.83	[115126.03, 121040.12] 126051.13	[118105.45, 121833.82] 125360.92
`eil51`	(5, 14)	[393.62, 421.04] 445.14	[407.83, 425.27] 440.63	[399.27, 423.42] 443.81
`eil76`	(60, 38)	[531.70, 560.44] 584.44	[538.27, 559.73] 577.95	[527.52, 558.50] 585.01
`eil101`	(7, 50)	[635.90, 658.49] 676.83	[636.21, 655.82] 675.23	[640.07, 665.80] 687.16
`kroA100`	(239, 451)	[20135.53, 22024.95] 23520.29	[19817.05, 21047.05] 22039.85	[20320.10, 21690.25] 22851.23
`kroA200`	(65, 134)	[29217.27, 30908.65] 32374.05	28591.49, 29942.70] 31033.10	[29839.39, 30950.97] 31904.49
`lin105`	(37, 96)	[13856.84, 14475.32] 15020.69	[13954.35, 14596.09] 15149.65	[14101.05, 14830.78] 15494.39
`pcb442`	(315, 169)	[52549.31, 54145.65] 55559.21	[51390.02, 53847.92] 56480.68	[53838.73, 54838.37] 55728.19
`pr107`	(22, 91)	[39372.19, 40722.93] 41816.26	[39281.91, 40164.25] 40900.35	[40542.34, 42662.08] 44347.41
`pr124`	(8, 76)	[56409.17, 59750.24] 62513.57	[56651.56, 58959.58] 61326.05	[56773.39, 60619.76] 63613.63
`pr136`	(40, 105)	[93215.71, 97631.93] 101302.04	[94762.04, 98531.97] 101853.31	[92518.23, 97757.66] 101999.69
`pr152`	(103, 2)	[66267.42, 69220.83] 71765.97	[65580.40, 67854.34] 69956.44	[63990.73, 69611.87] 75018.80
`rat195`	(166, 128)	[2329.85, 2450.32] 2562.35	[2349.12, 2463.14] 2559.40	[2332.97, 2419.72] 2491.89
`rd100`	(26, 73)	[7485.46, 8151.63] 8776.48	[7541.79, 7949.27] 8304.01	[7758.64, 8222.84] 8621.75
`st70`	(7, 43)	[633.15, 671.79] 705.16	[643.68, 674.06] 700.06	[662.87, 693.66] 720.24

Table 5: Comparison of the results obtained by neural HPP algorithms for various TSP instances. In each case we report the best results and the values of the parameters associated with these results.

Instance	Endpoints (X_s, X_t)	Test Interval	GS (M, σ, K_σ)	KL $(M, \sigma, K_\sigma, \omega)$	KG $(M, \sigma, K_\sigma, \omega)$
att532	(239, 451)	[28054.48, 28486.41]	29379.84 (400,30,0.8)	29281.37 (80,5,0.8,0.20)	29191.64 * (560,20,0.8,0.25)
berlin52	(10, 31)	[7394.56, 7549.32]	7660.33 (16,10,0.8)	7353.49 * (56,10,0.8,0.05)	7522.04 (48,5,0.8,0.15)
bier127	(14, 122)	[114970.39, 119102.93]	124153.90 (75,40,0.8)	121468.76 * (25,30,0.8,0.05)	121512.26 (125,45,0.8,0.05)
eil51	(5, 14)	[405.69, 422.95]	438.74 (50,30,0.8)	431.36 * (55,35,0.8,0.10)	432.13 (30,40,0.8,0.05)
eil76	(60, 38)	[526.20, 548.07]	575.02 (45,5,0.8)	567.66 * (75,35,0.8,0.25)	570.72 (60,30,0.8,0.15)
eil101	(7, 50)	[631.69, 649.28]	661.58 (40,50,0.8)	653.33 (80,45,0.8,0.10)	649.09 * (60,40,0.8,0.10)
kroA100	(92, 59)	[20400.44, 21124.34]	21219.57 (45,25,0.8)	21176.29 (75,25,0.8,0.20)	21129.63 * (45,10,0.8,0.10)
kroA200	(65, 134)	[28878.35, 29473.94]	29867.79 (160,35,0.8)	29867.79 * (200,45,0.8,0.20)	30174.88 (160,40,0.8,0.05)
lin105	(37, 96)	[14199.53, 14442.39]	14816.53 (20,5,0.8)	14619.93 * (80,10,0.8,0.15)	14666.96 (40,5,0.8,0.25)
pcb442	(315, 169)	[51315.36, 52195.61]	55642.20 (360,25,0.8)	55843.27 (450,30,0.8,0.20)	55379.15 * (450,50,0.8,0.05)
pr107	(22, 91)	[39860.24, 40213.38]	40445.14 (60,25,0.8)	40334.50 * (75,40,0.8,0.2)	40467.20 (30,25,0.8,0.25)
pr124	(8, 76)	[57309.63, 58823.04]	62137.92 (100,35,0.8)	59725.41 * (100,20,0.8,0.15)	60474.06 (50,5,0.8,0.20)
pr136	(40, 105)	[93215.71, 97631.93]	101946.20 * (120,20,0.8)	102126.91 (20,10,0.8,0.05)	103094.13 (140,30,0.8,0.15)
pr152	(103, 2)	[65791.64, 67701.35]	68005.73 (90,20,0.8)	67771.59 (30,25,0.8,0.25)	66191.84 * (90,25,0.8,0.20)
rat195	(166, 128)	[2293.74, 2330.87]	2611.38 (120,50,0.8)	2609.37 (40,15,0.8,0.10)	2575.93 * (40,20,0.8,0.25)
rd100	(26, 73)	[7685.11, 7996.26]	7958.17 (40,15,0.8)	7947.67 * (40,50,0.8,0.05)	7983.43 (20,20,0.8,0.20)
st70	(7, 43)	[634.58, 671.35]	693.19 (40,5,0.8)	685.03 * (40,10,0.8,0.15)	689.29 (60,30,0.8,0.10)

7.3 Computational Results

We have tested the new neural methods by using problem instances taken from TSPLIB [46]. The selected problems are Euclidean instances which means that each city is represented by its coordinates on the two-dimensional Euclidean space and the distances between the cities are computed according to the Euclidean norm. Furthermore, the sizes of the problems range between 51 and 532 cities. The test bed includes the instances used for the verification of interval estimates of the previous subsection as a subset. From this perspective, we believe that this paper is also of documentary importance because, in the literature, almost all of the reported results on the NN algorithms for the TSP and its relatives are based on randomly generated test problems, and thus we think that our results are also more realistic. All experiments were run on an HP-C110 workstation with 128 MByte RAM.

In order to compare the quality of the solutions obtained with our algorithms, confidence intervals for all instances in the test bed are computed by using the same six TSP heuristics. They are summarized in Table 3 and Table 4. The first column of the table includes references to TSPLIB instances. The second column contains the endpoints of the Hamiltonian path for which the intervals are calculated; each endpoint is selected randomly (i.e., they are not the endpoints of a randomly selected edge of an

Table 6: Relative (%) deviation from the lower confidence limits. The associated parameters of each case are specified in Table 5.

Instance	GS	KL	KG
att532	4.72	4.37	4.05
berlin52	3.59	-0.56	1.72
bier127	7.99	5.65	5.69
eil51	8.15	6.33	6.52
eil76	9.28	7.88	8.46
eil101	4.73	3.43	2.75
kroA200	3.44	3.43	4.49
lin105	4.34	2.96	3.29
pcb442	7.78	8.11	7.30
pr107	1.47	1.19	1.52
pr124	8.42	4.22	5.52
pr136	9.37	9.56	10.59
pr152	3.37	3.01	0.61
rat195	13.85	13.76	12.30
rd100	3.55	3.42	3.88
st70	9.44	7.95	8.62
Ave. Rel. (%) Deviations	6.47	5.29	5.46

optimal tour, which is the case in Section 7.2). The next three columns include average lengths of locally optimal Hamiltonian (X_s, X_t)-paths computed by the six TSP heuristics. The averages are taken over 50 solutions obtained from 50 randomly started runs. Recall that heuristic solutions are improved further by 2–OPT, except the one computed by Clark - Wright heuristic. As it can be observed, in addition to average path lengths, each cell includes also an interval. These are the confidence intervals computed as explained before. Observe that the higher the quality of the TSP heuristic (i.e., smaller average), the narrower is the confidence interval. This fact was also observed for the TSP [21].

Table 5 summarizes the results obtained on the test bed with our neural heuristics. The first column specifies the TSPLIB instance modified to obtain an HPP instance. The second column shows the starting and terminal cities for these instances also listed in Table 3 and Table 4. The third column consists of the confidence intervals we used in the comparisons selected from Table 3 and Table 4, and represent the heuristics with the smallest average. Note that, except the ones obtained for `lin105`, `pr136`, and `pr152`, all are generated by using nearest neighbor followed by 2-OPT (NNb). The numbers in the fourth to sixth columns are obtained by using GSOM_HPP (referred to as GS in the table), KNIES_HPP, (referred to as KL, L standing for "local", in the table) and KNIES_HPP_Global (referred to as KG, G standing for "global", in the table), respectively. These values are the best ones recorded after running the algorithms for a large number of different parameter settings. The numbers in parentheses are the parameter values for which these best results are obtained. As a reminder M represents the number of neurons on the initial band connecting X_s and X_t, σ is the standart deviation and K_σ is the multiplier used to decrease its value from epoch to epoch, and ω determines the width of the activation bubble. They are explained in Section 3 in large. In each row the cell with the smallest value is marked with a "$*$".

During the computations the neighborhood function is based on the lateral distance between two neurons on the piecewise linear line rather than their Euclidean distance. More explicitly, the distance between the winner neuron and any other neuron on the piecewise linear line is the absolute value of the difference between the indices of the two neurons. Furthermore, in all methods the parameter K_σ is kept fixed at 0.8 throughout the iterations; this is advised in the literature [5].

Figures 1, 2, and 3 illustrate the Hamiltonian paths associated with (a) optimal solutions and (b) the solution of KNIES_HPP for the problem instances `eil51`, `eil76`, and `eil101` by deleting the edges {41, 13}, {11, 53} and {54, 55} respectively. They are chosen randomly on the optimum tour listed in TSPLIB. Hamiltonian paths of Figure 4 are obtained by KNIES_HPP as well. Except in these cases, both of their endpoints are randomly selected, and thus they are not the endpoints of a randomly selected edge of an optimal TSP tour.

As it can be observed, KNIES_HPP (KL) is the winner for `berlin52`, `bier127`, `eil51`, `eil76`, `kroA200`, lin105, pr107, pr124, rd100, and st70. In the remaining instances KNIES_HPP_Global (KG) provides the best results except pr136. Also, note the negative relative deviation obtained with KL for berlin52. This is the only case where the best neural solution is lower than the lower confidence limit. For example, in the case of eil76, the lower confidence limit was 526.20. The length of the path obtained by KNIES_HPP is 567.66, which is 7.88 % higher than the lower confidence limit. Similarly, the length of the paths obtained by KNIES_HPP_Global and GSOM are respectively 570.72 and 575.02 corresponding to 8.46 % and 9.28 % of relative deviations from the lower confidence limits.

With the risk of being dubbed "immodest" we claim that these results are very good. This is because all of them have been obtained without resorting to a solution to the TSP. Besides, since the neurons are ordered linearly (as opposed to cyclically), we have had to forfeit all the information which we would have otherwise obtained by processing the neurons that would have been neighbors on the ring. In spite of having to discard this information, the results which we obtain are always within a few percent of the lower confidence limits !

The relative deviations from the lower confidence limits are summarized in Table 6. These values are calculated according to the following formula :

$$\frac{Z_{NH} - LCL}{LCL} \tag{22}$$

where Z_{NH} and LCL are respectively the length of the near optimal Hamiltonian path and lower confidence limits of the intervals listed in Table 5. Note that $NH \in \{GS, KL, KG\}$. Of course, it is impossible to make categorical statements about the various algorithms from the results presented here. However, to get an overall picture, we have computed the average deviations for GS, KL, and KG, which are respectively 6.47 %, 5.29 % and 5.46 %. Based on these results, it is possible to draw the conclusion that KNIES_HPP (KL) performs better than the other two neural methods. However, the performance of KNIES_HPP_Global (KG) is close to the KNIES_HPP's. This can also be observed from Table 5 where KG outperforms KL for att532, eil101, kroA100, pcb442, pr152, and rat195. This fact makes KG's use more attractive since it is computationally less expensive than KL.

Finally, we should mention that the performance of all methods is not "proportional" to the number of cities. For example, in the case of 532 city instance, att532, GS, KL, and KG yielded relative deviations of 4.72 %, 4.37 %, 4.05 % respectively. However, these deviations are 9.28 %, 7.88 %. 8.46 % for the 51 city instance eil51.

8 Conclusions

In this paper we have presented new algorithms, which to our knowledge, are the first documented *all-neural* solutions to the Euclidean Hamiltonian Path Problem (HPP). The reason why similar neural algorithms from the TSP cannot be easily generalized for the HPP is that the heuristics which map the cities onto the neurons "lose their credibility". This is because the underlying cyclic property of the *order* of the neurons used in the TSP is lost in the HPP. The first of the new algorithms, GSOM_HPP, is a generalization of the SOM as adapted by Angéniol et al. [5]. The second method uses the recently-introduced self-organizing neural network, the Kohonen Network Incorporating Explicit Statistics (KNIES) [2]. The primary difference between KNIES and Kohonen's Self-Organizing Map (SOM) is the fact that unlike SOM, every iteration in the training phase includes *two distinct modules* - the attracting module and the dispersing module. As a result of SOM and the dispersing module introduced in KNIES the neurons *individually* find their places both statistically and topologically, and also *collectively* maintain their mean to be the mean of the data points which they represent. KNIES, which has previously been used to yield a very accurate neural solution to the Euclidean TSP[2], has now been extended to solve the Euclidean HPP. Experimental results for problems obtained by modifying various TSPLIB instances [46] for randomly selected starting and terminal cities indicate that these algorithms are very accurate. Since the paper reports results based on such a well-documented and standardized test bed and uses state-of-the-art statistical methods for performance analysis, we believe that it is of documentary importance too.

Acknowledgement. The authors would like to acknowledge that the computational experiments of this research were realized on the hardware donated to the Department of Industrial Engineering at Boğaziçi University by Hewlett – Packard. They are also grateful to Norbert Ascheuer from Konrad–Zuse–Zentrum–für Informationstechnik Berlin (ZIB) for his suggestions on the generation of optimal

Hamiltonian paths from optimum TSP tours and Gerhard Reinelt from the University of Heidelberg for pointing TSP instances distributed with optimum tours.

References

[1] Aiyer, S.V.B., M. Niranjan and F. Fallside, (1990), "A Theoretical Investigation into the Performance of the Hopfield Model", *IEEE Transactions on Neural Networks* 1:2, pp. 204–215.

[2] Oommen, B. J., Aras, N. and Altinel, K., "Solving the Travelling Salesman Problem using the Kohonen Network Incorporating Explicit Statistics". To appear in the *Proceedings of WIRN/VIETRI-98, the Tenth Italian Workshop on Neural Nets*, Vietri sul Mare, Italy, May 1998. Also available as Research Paper No. FBE–IE–14/97–18, Boğaziçi University, İstanbul, TÜRKİYE.

[3] Ascheuer, N., (1996), "Hamiltonian Path Problems in the On–line Optimization of Flexible Manufacturing Systems," Technical Report No. TR 96–3, ZIB, Berlin, GERMANY.

[4] Ascheuer N., (1998), private communication.

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

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

[7] Burke, L. I., (1996), " 'Conscientious' Neural Nets for Tour Construction in the Traveling Salesman Problem : The Vigilant Net," *Computers & Operations Research* 23, pp. 121–129.

[8] Burke, L. I. and P. Damany, (1992), "The Guilty Net for the Traveling Salesman Problem," *Computers & Operations Research* 19, pp. 255–265.

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

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

[11] Cooke, P., (1979), "Statistical Inference for Bounds of Random Variables," *Biometrika 66*, pp. 367–374.

[12] Cichocki, A. and R. Unbehauen, (1993), *Neural Networks for Optimization and Signal Processing*, Wiley, Chichester.

[13] Dannenbring, D., (1977), "Estimating Optimal Solutions for Large Combinatorial Problems," *Management Science* 23, pp. 1273–1283.

[14] Derigs, U., (1985), "Using Confidence Limits for the Global Optimum in Combinatorial Optimization," *Operations Research* 33, pp. 1024–1049.

[15] Duda, R. O. and P. E. Hart, (1973), *Pattern Classification and Scene Analysis*, Wiley, Chichester.

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

[17] Fisher, R. and L. Tippet, (1928), "Limiting Forms of the Frequency Distribution of the Largest or Smallest Number of a Sample," *Proc. Cambridge Phil. Soc.* 24, pp. 180–191.

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

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

[20] Golden, B. and F. B. Alt, (1979), "Interval Estimation of a Global Optimum for Large Combinatorial Problems," *Naval Res. Logistics Quart.* 26, pp.69–77.

[21] Golden, B. and W. R. Stewart (1985), "Empirical Analysis of Heuristics," in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, edited by E.L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, Wiley, Chichester.

[22] Graf, D. H. and W. R. Lalonde, (1988), "A Neural Controller for Collision-free Movement of General Robot Manipulators," *Proc. IEEE Int. Conf. on Neural Networks*, pp. 177-I-84.

[23] Graf, D. H. and W. R. Lalonde, (1989), "Neuroplanners for Hand / Eye Coordination," *Proc. Int. Joint Conf. on Neural Networks*, II-543-II-548.

[24] Gray, D. H., (1984), "Vector Quantization," *IEEE ASSP Mag* 1., pp. 4–29.

[25] Hemani, A. and A. Postula, (1990), "Scheduling by Self Organization," *Proc. Int. Joint Conf. on Neural Networks*, IJCNN-90-WASH-DC, II-543-II-548.

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

[27] Hueter, G. J., (1988), "Solution of the Traveling Salesman Problem with an Adaptive Ring," *Proc. of the IEEE Int. Conf. on Neural Networks*, San Diego, CA, pp. I-85–92.

[28] Johnson, D.S. and L. A. McGeoch, (1997), "The Traveling Salesman Problem : a case study," in *Local Search in Combinatorial Optimization*, edited by E. Aarts and J. K. Lenstra, Wiley, Chichester.

[29] Jeffries, C. and T. Niznik, (1994), "Easing the Conscience of the Guilty Net," *Computers and Operations Research* 21, pp. 961–968.

[30] Kohonen, T., K. Makisara and T. Saramaki, (1984), "Phonotic Maps - Insightful Representation of Phonological Features for Speech Recognition," *Proc. Seventh. Int. Conf. on Pattern Recognition*, pp. 182–185

[31] Kohonen, T., K. Torkkola, M. Shozokai, J. Kangas and O. Venta, (1987), "Microprocessor Implementation of a Large Vocabulary Speech Recognizer and Phonetic Typewriter for Finish and Japanese", *Proc. European Conference of Speech Technology*, pp. 377–380.

[32] Kohonen, T., (1988), "The Neural Phonotic Typewriter", *Computer*, 21, pp.11–22.

[33] Kohonen, T., (1990),"The Self-Organizing Map," *Proc. IEEE*, 78, pp. 74-90.

[34] Kohonen, T., (1995), *Self-Organizing Maps*, Springer-Verlag, Berlin.

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

[36] Lin, S. and B. Kernighan, (1973), "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research* 21, pp. 498–516.

[37] Linde, Y., A. Buzo and R. M. Gray, (1980), "An Algorithm for Vector Quantization," *IEEE Trans. Communication COM-28*, pp. 61–71.

[38] Los, M. and C. Lardinois, (1982), "Combinatorial Programming, Statistical Optimization and the Optimal Transportation Network Problem," *Trans. Res.* 16B, pp. 89–124.

[39] Marks, K. M. and K. F. Goser, (1988), "Analysis of VLSI Process Data Based on Self-Organizing Feature Maps," *Proc. Nuero-Nimes '88*, pp.337–347.

[40] Makhoul, J., S. Roucos and H. Gish, (1985), "Vector Quantization in Speech Coding," *Proc. IEEE*, 73, pp. 1551–1588.

[41] Martinetz, J., H. J. Ritter and K. J. Schulten, (1990), "Three-dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm," *IEEE Trans. Neural Networks*, 1, pp. 131–136.

[42] Neumann, E. K., D. A. Wheeler, A. S. Burnside, A. S. Bernstein and J. C. Hall, (1990), "A Technique for the Classification and Analysis of Insect Courtship Song," Proc. Int. Joint Conf. on Neural Netwrorks, IJCNN-90-WASH-DC, II-257-II-262.

[43] Orlando, G. A., R. Mann and S. Haykin, (1990), "Radar Classification of Sea-ice Using Traditional and Neural Classifiers", *Proc. Int. Joint Conf. on Neural Networks*, IJCNN-90-WASH-DC, II-263-II-266.

[44] Papadimitriou, C. H., (1978), "The Euclidean Traveling Salesman Problem is NP-Complete", *Theoretical Computer Science*, 4, pp. 237–244.

[45] Potvin, J.-Y., (1993), "The Traveling Salesman Problem: A Neural Network Perspective," *ORSA Journal on Computing* 5, pp. 328–348.

[46] Reinelt, G., (1991), "TSPLIB— A Traveling Salesman Problem Library", *ORSA Journal on Computing* 3, pp. 376–384.

[47] Reinelt, G., (1994), *The Traveling Salesman. Computational Solutions for TSP Applications*, Springer-Verlag, Berlin.

[48] Ritter, H. J., (1991), "Asymptotic Level Density for a Class of Vector Quantization Processes," *IEEE Transaction on Neural Networks* 2, pp. 173–175

[49] Ritter, H. J., J. Martinetz and K. J. Schulten, (1989), "Topology Conserving Maps Learning Visuomotor Coordination," *Neural Network* 2, pp. 159–168.

[50] Ritter, H. J. and K. J. Schulten, (1986), "Topology Conserving Mapping for Learning Motor Tasks," *Proc. Neural Networks for Computing, AIP Conference*, pp. 376-380.

[51] Ritter, H. J. and K. J. Schulten, (1988a), "Extending Kohonen's Self-organizing Mapping to Learn Ballistic Movements," *NATO ASI Series*, pp. 393-406.

[52] Ritter, H. J. and K. J. Schulten, (1988b), "Kohonen's Self-organizing Maps : Exploring Their Computational Capabilities" *Proc. of the Int. Joint Conf. on Neural Networks*, pp. 2455–2460.

[53] Samarabandu, J. K. and O. E. Jakubowicz, (1990), "Principles of Sequential Feature Maps in Multi-Level Problems," *Proc. Int. Joint Conf. on Neural Networks*, IJCNN-90-WASH-DC II-683-II686.

[54] Simic, P. D., (1990), "Statistical Mechanics as the Underlying Theory of Elastic and Neural Optimization," *Network* 1, pp. 363–374.

[55] Tryba, V., K. M. Marks, U. Rütcker and K. Goser, (1988), "Selbst-organizierende Karten Als Lernende Klassifizierende Speicher," *IFG Fachbericht* 102, pp. 407–419.

[56] Vakhutinsky, A. I. and B. L. Golden, (1995), "A Hierarchical Strategy for Solving Traveling Salesman Problem Using Elastic Nets," *Journal of Heuristics*, 2, pp. 67–76.

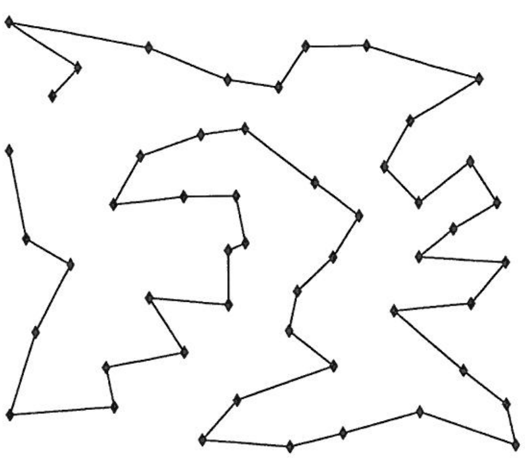
[57] Wong, Y., (1996), "A Comparative Study of the Kohonen Self-Organizing Map and the Elastic Net," *Computational Learning Theory and Natural Learning Systems*, 2, pp. 401–413.

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

[59] Zador, P. L., (1982),"Asymptotic Quantization of Error of Continuous Signals and the Quantization Dimension," *IEEE Transaction on Information Theory* 28, pp. 139–149.

[60] Zanakis, S.H., (1979a),"Extended Pattern Search with Transformation for the Three–Parameter Weibull MLE Problem," *Management Science* 25, pp. 1149–1161.

[61] Zanakis, S.H., (1979b),"A Simulation Study of Some Simple Estimators of the Three–Parameter Weibull Distribution," *J.Statist. Comput. Simulat.* 9, pp.101–106.


(a) Optimal

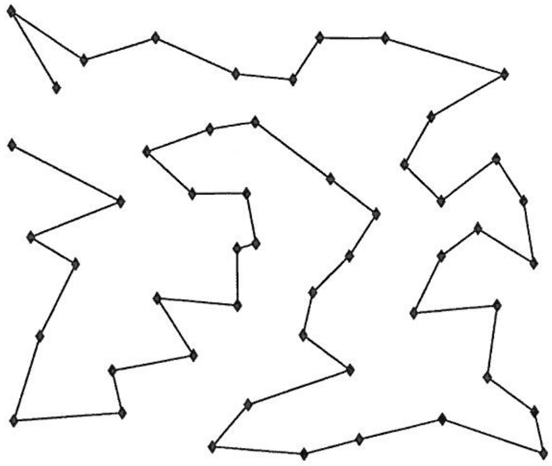
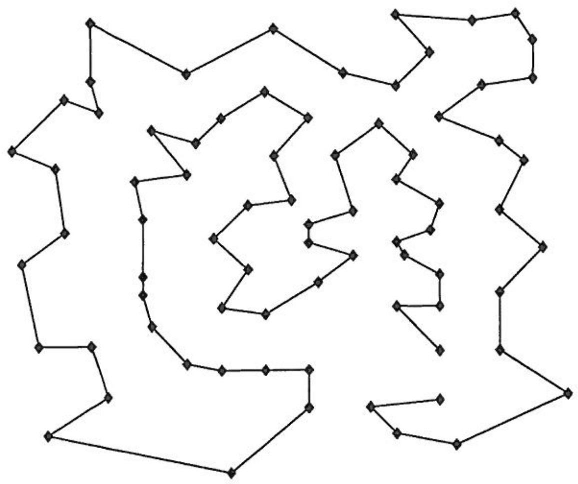

(b) Obtained with KNIES_HPP

Figure 1: Hamiltonian Paths for eil51 from 41 to 13


(a) Optimal

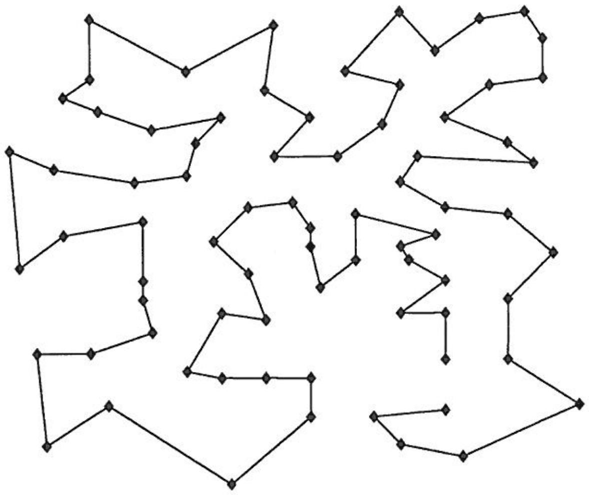
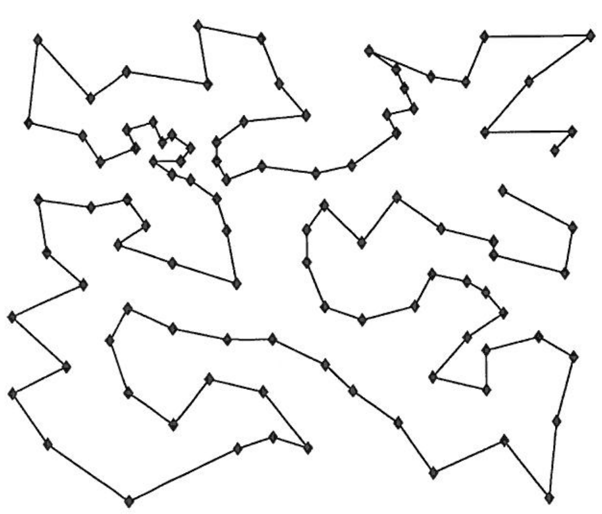

(b) Obtained with KNIES_HPP

Figure 2: Hamiltonian Paths for eil76 from 11 to 53


(a) Optimal

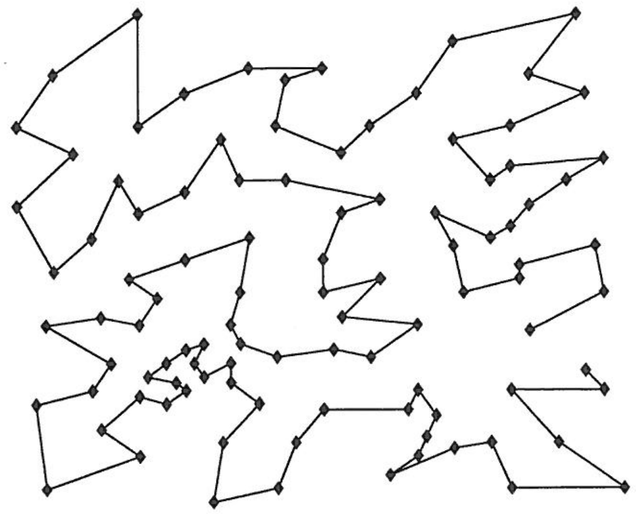
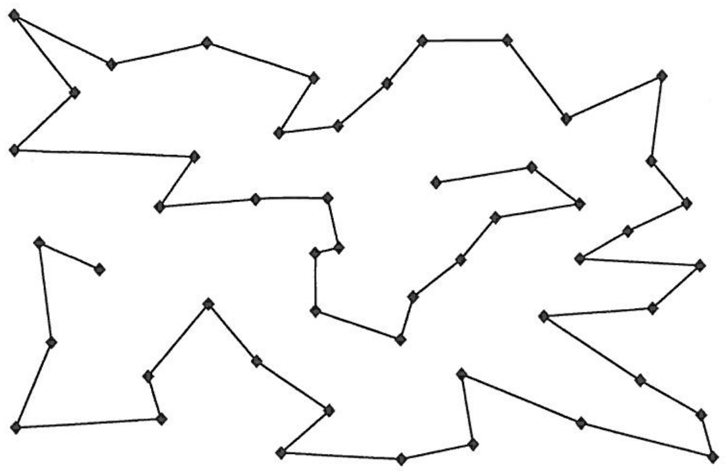
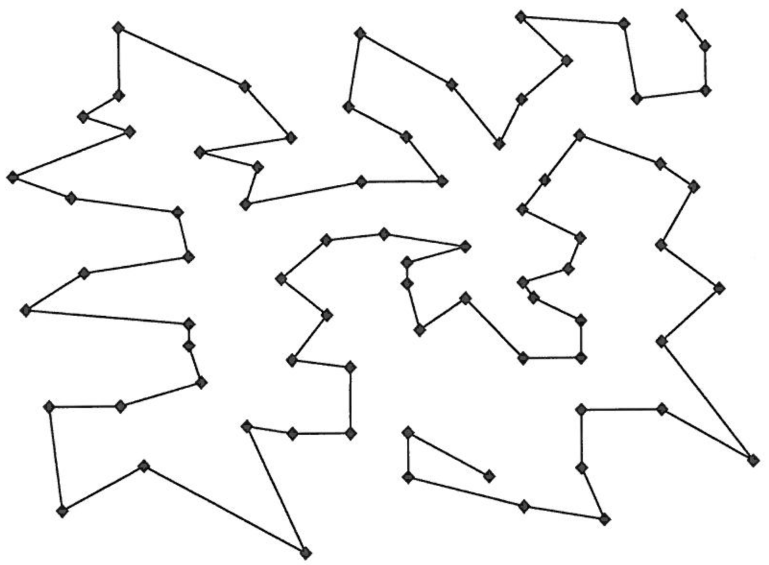

(b) Obtained with KNIES_HPP

Figure 3: Hamiltonian Paths for eil101 from 54 to 55


(a) For eil51 from 5 to 14


(a) For eil76 from 60 to 38

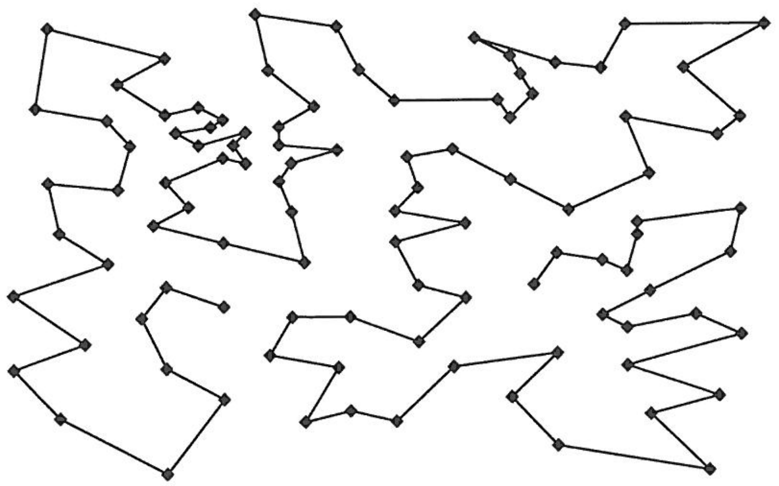

(a) For eil101 from 7 to 50

Figure 4: Hamiltonian Paths obtained with KNIES_HPP for random starting and terminal cities