A Novel Constructive Optimizer Neural Network for the Traveling Salesman Problem

M. Saadatmand-Tarzjan[*,1], M. Khademi[2], M. R. Akbarzadeh-T.[2], H. Abrishami Moghaddam[3]

1. Department of Electrical Engineering, Tarbiat Modares University
Nasr Bridge, P.O. Box 14155-111, Tehran, Iran
saadatmand@kiaeee.org

2. Department of Electrical Engineering, Ferdowsi University of Mashhad
Vakil Abad Blv., P.O. Box 91775-1111, Mashhad, Iran
akbarzadeh@ieee.org
khademi@um.ac.ir

3. Electrical Engineering Department, K.N. Toosi University of Technology,
Seyed Khandan P.O. Box 16315-1355, Tehran, Iran
moghadam@saba.kntu.ac.ir

[*]Corresponding Author

Corresponding Author

M. Saadatmand-Tarzjan (Ph.D. Student)

Electrical Engineering Department

Tarbiat Modares University

Intersection of Jalale-Ale-Ahmad and Shahid-Chamran Highways

P.O. Box 14155-194

Tehran, Iran

Phone: +98 912 515 7462

Fax: +98 21 8005040

E-mail: saadatmand@kiaeee.org

A Novel Constructive-Optimizer Neural Network for the Traveling Salesman Problem

M. Saadatmand-Tarzjan, M. Khademi, M. R. Akbarzadeh-T., H. Abrishami Moghaddam

Abstract

In this paper, a novel constructive-optimizer neural network (CONN) is proposed for the traveling salesman problem (TSP). CONN uses a feedback structure similar to Hopfield-type NNs and a competitive training algorithm similar to the Kohonen-type self-organizing maps. Consequently, CONN is composed of a constructive part which grows the tour and an optimizer part to optimize it. In the training algorithm, an initial tour is created first and introduced to CONN. Then, it is trained in the constructive phase for adding a number of cities to the tour. Next, the training algorithm switches to the optimizer phase for optimizing the current tour by displacing the tour cities. After convergence in this phase, the training algorithm switches to the constructive phase anew and is continued until all cities are added to the tour. Furthermore, we investigate a relationship between the number of TSP cities and the number of cities to be added in each constructive phase.

CONN was tested on nine sets of benchmark TSPs from TSPLIB to demonstrate its performance and efficiency. It performed better than several typical NNs, including KNIES_TSP_Local, KNIES_TSP_Global, Budinich's SOM, Co-Adaptive Net, and multi-valued Hopfield network as wall as computationally-comparable variants of the simulated annealing algorithm, in terms of both CPU time and accuracy. Furthermore, CONN converged considerably faster than expanding SOM (ESOM) and evolved integrated SOM (eISOM) and generated shorter tours compared to KNIES_DECOMPOSE. Although CONN is not yet comparable in terms of accuracy with some sophisticated computationally-intensive algorithms, it converges significantly faster than them. Generally speaking, CONN provides the best compromise between CPU time and accuracy among currently reported neural networks for TSP.

Key words

Traveling Salesman Problem (TSP), Constructive-Optimizer Neural Network (CONN), Hopfield-Type Neural Networks, Kohonen-Type Self-Organizing Maps

I. Introduction

Combinatorial optimization tasks such as the traveling salesman problem (TSP) belong to a family of NP-Complete problems [1] whose computational complexity rises exponentially by increasing the number of parameters. Finding suboptimal solutions with a reasonable cost may be more advantageous in many of current TSP applications such as printed circuit boards manufacturing [2, 3], data transmission in computer networks [4], power distribution networks [5], image processing and pattern recognition [6], robot navigation [7], and data partitioning [8].

TSP consists of finding the shortest closed tour visiting *n* cities. To date, several methods based on deterministic or probabilistic heuristics have been proposed for solving TSP. These include classical search maps [9], simulated annealing (SA) [10], artificial neural networks (Kohonen-type self-organizing maps [11-16], Hopfield-type neural networks [17-19], Boolean neural network [20], and chaotic neural network [21]), genetic algorithms (GA) [22, 23], evolutionary programming [24], ant colony optimization (ACO) [25, 26], population-based incremental learning [27], tabu search [28], and fine-tuned learning [29]. Since each of the above approaches has weak points as well as strengths, determining a superior approach is nontrivial. For example, several researchers reported good solution quality of the evolutionary methods such as GA for off-line applications [30], while others preferred algorithms such as ACO [25] and SA [31] for their efficiency. Although ACO and SA may be faster than evolutionary algorithms, they are still slower than neural approaches [14]. In fact, neural approaches are generally considered to be fast with inferior solution quality [32]. Therefore, developing a neural network (NN) structure that provides a good TSP solution with less computational complexity remains a challenging endeavor.

Among the above approaches, the Hopfield-type neural network (HNN) and Kohonen-type self-organizing map (K-SOM) are paradigmatically similar to the proposed approach. HNN generally has a second order energy function that determines its structure and behavior. Moreover, it uses a negative feedback to minimize the energy function during NN training. In spite of its fast convergence speed, a major drawback of HNN for solving TSP is getting caught in local minima of the energy function [29, 32]. A number of solutions have been proposed to avoid local minima of the energy function in HNN. For example, Lee and Sheu [33] addressed this problem by adding an adaptable corrective input to neurons.

In contrast to HNN, K-SOM has a slow convergence speed, mainly because of its competitive training algorithm [11]. Nevertheless, it attracted many research interests to explore and enhance its capability to solve TSP due to its intuitive appeal, relative simplicity, and promising performance [14]. Because of its relatively poor performance, it had been previously argued that K-SOM may not be the best benchmark to evaluate the effectiveness of NNs for TSP optimization [32]. However, recent improvements in K-SOM demonstrated their potential ability for solving TSP. Generally, there are three main streams to enhance the original SOM [11-13]: *i)* introducing a variable structure network, *ii)* amending the competition criterion, and *iii)* enhancing the learning rule. Recently, Leung *et al.* [14] proposed an expanding learning rule (ESOM) which can generate shorter tours than several typical K-SOMs such as convex elastic net (CEN) [15] and Budinich's SOM [16]. Furthermore, Jin *et al.* [11] developed an integrated SOM (ISOM) which incorporates the above learning mechanisms. They also optimized ISOM using a genetic algorithm to obtain evolved ISOM (eISOM). Another example is Co-Adaptive Net [34] which allows neurons to co-operate and compete amongst themselves depending on their situation.

In this paper, a novel constructive-optimizer NN (CONN) is introduced to provide the best compromise between the convergence speed and solution quality. The main idea of the proposed NN is taking advantage of HNN's fast convergence and K-SOM's solution quality. For this purpose, CONN uses a feedback structure similar to HNN and a competitive training algorithm similar to K-SOM. Consequently, CONN is composed of a constructive part which grows the tour and an optimizer part to optimize it. In the training algorithm, an initial tour is created first and introduced to CONN. We show that depending on the number of TSP cities, one of three different algorithms including the cheapest link, convex hull, and hull through four outermost cities can be used to generate the initial tour. Then, CONN is trained in the constructive phase for adding a number of cities to the tour. Next, the training algorithm switches to the optimizer phase for optimizing the current tour by displacing the tour cities. After convergence in this phase, the training algorithm switches to the constructive phase anew and is continued until all cities are added to the tour. Finally, we investigate a relationship between the number of TSP cities and the number of cities to be added in each constructive phase.

The paper is organized as follows. In the next section, we present the proposed basic optimizer NN (BONN). CONN is presented in Section III as an extension of BONN. Section IV is devoted to evaluate the performance of CONN compared to a large number of its counterparts using nine sets of TSP benchmarks. Finally, conclusions are drawn in Section V.

Notations used in this paper are fairly standard. Boldface symbols are used for vectors (in lower case letters). We also have the following notations:

n	total number of TSP cities;
m	number of cities on the tour (tour cities);
$\mathbf{c}_i$	i-th city;
$D(\mathbf{c}_i,\mathbf{c}_j)$	distance function between city pair $(\mathbf{c}_i,\mathbf{c}_j)$;
$\mathbf{x}_i^l$	output vector of the i-th neuron in the l-th layer;
$x_{i,j}^l$	j-th component of the output vector $\mathbf{x}_i^l$;
$\boldsymbol{\psi}^k$	current tour in the k-th step;
$\boldsymbol{\psi}_j^k$	j-th city on the tour in the k-th step;
e^k	energy function value in the k-th step;
Q^k	set of all tour cities in the k-th step;
R^k	set of all nontour cities in the k-th step.

II. Basic optimizer neural network

BONN [35] is considered as an elementary structure of CONN. It is a simple optimizer neural network which uses a feedback configuration as well as a competitive training algorithm. In each step of its training algorithm, BONN improves the current tour until no further improvement can be achieved (convergence to the final solution).

(Figure 1)

A. BONN structure

Figure 1 illustrates the BONN structure for a 3-city TSP which includes *i)* the tour, *ii)* link, *iii)* link competitive, and *iv)* tour competitive layers. Although the output of typical neurons is usually a scalar, the output of each neuron in BONN is a vector whose content is dependent on the layer. All cities in BONN are on the tour, i.e. $m = n$. This paper addresses the symmetric TSP, where $D(i,j) = D(j,i)$.

The first layer of BONN (the tour layer) specifies the current tour by *m* neurons in which the neuron output $\mathbf{x}_j^1$ ($j=1,2,\ldots,m$) indicates the *j*-th city on the tour. It means that the current tour is determined by:

$$\boldsymbol{\psi}^k=[\boldsymbol{\psi}_j^k]=[\mathbf{x}_1^1,\mathbf{x}_2^1,\ldots,\mathbf{x}_m^1], \qquad j=1,2,\ldots,m \tag{1}$$

The tour is a closed cycle; hence, the 0-th neuron is the same as the *m*-th one and the (*m*+1)-th neuron is the same as the first one in the layer. As will be shown later, the tour cities in the first layer are authorized to be displaced between neurons. The BONN energy function is simply defined as the tour length:

$$e^k=\sum_{j=1}^{m} D(\mathbf{x}_j^1,\mathbf{x}_{j+1}^1) \tag{2}$$

In order to visit each city only once, this energy function is minimized with the following constraint:

$$\forall j\neq i\in\{1,2,\ldots,m\}: \qquad \mathbf{x}_j^1\neq\mathbf{x}_i^1 \tag{3}$$

The training algorithm minimizes the BONN energy function constrained by (3) in two steps. First, BONN is initialized by an initial valid tour that satisfies the constraint. Second, the initial tour is iteratively improved (while satisfying the constraint) until no further improvement can be achieved.

The second layer (the link layer) also contains *m* neurons; each one indicates a tour link which connects two adjacent cities on the tour. The output of the neurons in this layer is given by:

$$\mathbf{x}_j^2=\left[\mathbf{x}_{j,1}^2,\mathbf{x}_{j,2}^2\right]=\left[\mathbf{x}_j^1,\mathbf{x}_{j+1}^1\right], \qquad j=1,2,\ldots,m \tag{4}$$

The third layer (the link competitive layer) has $n+1$ neurons. One of them is a threshold neuron whose output is a vector containing the length of the tour links:

$$\mathbf{t}^3=[t_j^3]=\left[D(\mathbf{x}_{j,1}^2,\mathbf{x}_{j,2}^2)\right], \qquad j=1,2,\cdots,m \tag{5}$$

The remaining neurons in the third layer are assigned to the cities in a one to one and ordered manner (hereafter, each neuron is indicated by its corresponding city and vice versa). In each neuron, the activation value of each link is given by:

$$v_{i,j}^3=D(\mathbf{x}_{j,1}^2,\mathbf{c}_i)+D(\mathbf{c}_i,\mathbf{x}_{j,2}^2)-t_j^3, \qquad i=1,2,\ldots n,\ j=1,2,\cdots,m \tag{6}$$

where $v_{i,j}^3$ indicates the tour length increase due to inserted city $\mathbf{c}_i$ between the cities in the *j*-th link $\left[\mathbf{x}_{j,1}^2,\mathbf{x}_{j,2}^2\right]$. The output of each neuron in the third layer is a vector including the smallest activation value and its index:

$$\mathbf{x}_i^3 = \left[x_{i,1}^3,\ x_{i,2}^3\right]^T = \left[\min_{j\in P_i}\left(v_{i,j}^3\right),\ \arg\left(\min_{j\in P_i}\left(v_{i,j}^3\right)\right)\right], \qquad i = 1,2,\ldots,n \tag{7}$$

where P_i is a set of links excluding $\mathbf{c}_i$:

$$P_i = \left\{j \middle| j = 1,2,\ldots m;\ \mathbf{x}_{j,1}^2, \mathbf{x}_{j,2}^2 \neq \mathbf{c}_i\right\}. \tag{8}$$

In more details, in each neuron of the third layer, a competition occurs among all of the second layer neurons and finally, the winning neuron activation value and its corresponding index are retained as the neuron's output.

The fourth layer (the tour competitive layer) also has $n+1$ neurons. One of them is again a threshold neuron whose output is a vector containing the decrease in the tour length due to removal of each city from the current tour:

$$\mathbf{t}^4 = [t_i^4], \qquad i = 1,2,\ldots,n \tag{9}$$

where

$$t_i^4 = \begin{cases} D(\mathbf{x}_{j,1}^2, \mathbf{x}_{j,2}^2) + D(\mathbf{x}_{j+1,1}^2, \mathbf{x}_{j+1,2}^2) - D(\mathbf{x}_{j,1}^2, \mathbf{x}_{j+1,2}^2) & \mathbf{x}_{j,2}^2 = \mathbf{x}_{j+1,1}^2 = \mathbf{c}_i \in Q^k \\ 0 & \mathbf{c}_i \in R^k \end{cases} \tag{10}$$

where,

$$Q^k = \{\mathbf{x}_i^1 | i = 1,2,\ldots,m\} \tag{11}$$

$$R^k = \{\mathbf{c}_i | i = 1,2,\ldots,n\} - Q^k\ . \tag{12}$$

The remaining neurons in the fourth layer are also assigned to the cities in a one to one and ordered manner (hereafter, each neuron is indicated by its corresponding city and vice versa). The activation value of each neuron in this layer is computed by subtracting its corresponding threshold from its input as follows:

$$v_i^4 = x_{i,1}^3 - t_i^4\ , \qquad i=1,2,\cdots,n \tag{13}$$

Indeed, the activation value of the i-th neuron indicates the tour length increase due to displaced city $\mathbf{c}_i$ from its current location to the new location between cities in the link specified by $x_{i,2}^3$. A competition is then occurred among the neurons in this layer and the neuron with the smallest increase in the tour length wins:

$$\omega_{\text{opt}} = \arg\left(\min_{i\in Q^k}(v_i^4)\right) \tag{14}$$

Finally, the outputs of all neurons except the winning one are set to zero as follows:

$$x_i^4 = 0, \qquad i \neq \omega_{\text{opt}} \tag{15}$$

If the activation value of the winning neuron is negative, its output will indicate a new location for city $\mathbf{c}_i$ according to (16). Otherwise, the NN convergence is achieved and the winning neuron's output is set to zero as well.

$$x_{\omega_{\text{opt}}}^4 = x_{\omega_{\text{opt}},2}^3 \times \varphi(v_{\omega_{\text{opt}}}^4) \tag{16}$$

where $\varphi(\cdot)$ is a hard limiter function defined by:

$$\varphi(\text{a}) = \begin{cases} 1 & \text{a} < 0 \\ 0 & \textit{otherwise} \end{cases} \tag{17}$$

(Table 1)

B. BONN training algorithm

The BONN training algorithm is summarized in Table 1. Since at each training step, only one city is displaced from its current location, the final tour usually satisfies constraint (3) like the initial tour. In more details, displacing a city from its current location to another location neither creates a loop on the tour nor changes the number of tour cities. Therefore, if the initial tour satisfies constraint (3), it will be remained satisfied after each displacement.

C. BONN convergence analysis

According to (14), in the *k*-th step of the training algorithm, the winning neuron ω_{opt} gives $x_{\omega_{opt}}^4 = p$ as its output in the optimizer phase. Furthermore, as stated in (19), the corresponding city $\mathbf{c}_{\omega_{\text{opt}}}$ is located in location *a* on the current tour. According to (4) and (5), the threshold value corresponding to the *p*-th link is given by:

$$t_p^3 = D(\mathbf{x}_p^1, \mathbf{x}_{p+1}^1) \tag{21}$$

Then, using (6) and (7), the output of the neuron ω_{opt} in the third layer is obtained as follows:

$$x_{\omega_{opt},1}^3 = D(\mathbf{x}_p^1, \mathbf{c}_{\omega_{opt}}) + D(\mathbf{c}_{\omega_{opt}}, \mathbf{x}_{p+1}^1) - D(\mathbf{x}_p^1, \mathbf{x}_{p+1}^1) \tag{22}$$

Similarly, the threshold value of the neuron ω_{opt} in the fourth layer can be computed from (4) and (10) as:

$$t^4_{\omega_{opt}} = D(\mathbf{x}^1_{a-1}, \mathbf{c}_{\omega_{opt}}) + D(\mathbf{c}_{\omega_{opt}}, \mathbf{x}^1_{a+1}) - D(\mathbf{x}^1_{a-1}, \mathbf{x}^1_{a+1}) \tag{23}$$

Using (13), the activation value of the neuron ω_{opt} in the fourth layer is given by:

$$v^4_{\omega_{opt}} = \left(D(\mathbf{x}^1_p, \mathbf{c}_{\omega_{opt}}) + D(\mathbf{c}_{\omega_{opt}}, \mathbf{x}^1_{p+1}) - D(\mathbf{x}^1_p, \mathbf{x}^1_{p+1}) \right) - \left(D(\mathbf{x}^1_{a-1}, \mathbf{c}_{\omega_{opt}}) + D(\mathbf{c}_{\omega_{opt}}, \mathbf{x}^1_{a+1}) - D(\mathbf{x}^1_{a-1}, \mathbf{x}^1_{a+1}) \right) < 0 \tag{24}$$

According to (2), the value of CONN energy function in the *k*-th step is given by:

$$e^k = \left[\sum_{\substack{j=1 \\ j \neq a-1, a, p}}^{m} D(\mathbf{x}^1_j, \mathbf{x}^1_{j+1}) \right] + D(\mathbf{x}^1_{a-1}, \mathbf{c}_{\omega_{opt}}) + D(\mathbf{c}_{\omega_{opt}}, \mathbf{x}^1_{a+1}) + D(\mathbf{x}^1_p, \mathbf{x}^1_{p+1}) \tag{25}$$

The training algorithm will displace the winning city $\mathbf{c}_{\omega_{\text{opt}}}$ from its current location, *a*, to the new location between cities in the link specified by $x^4_{\omega_{opt}}$. This will form the next tour as indicated by (20). Hence, the value of CONN energy function in the next step, $k+1$, will be given by:

$$e^{k+1} = \left[\sum_{\substack{j=1 \\ j \neq a-1, a, p}}^{m} D(\mathbf{x}^1_j, \mathbf{x}^1_{j+1}) \right] + D(\mathbf{x}^1_p, \mathbf{c}_{\omega_{opt}}) + D(\mathbf{c}_{\omega_{opt}}, \mathbf{x}^1_{p+1}) + D(\mathbf{x}^1_{a-1}, \mathbf{x}^1_{a+1}) \tag{26}$$

Finally, using (24)-(26), we have:

$$e^{k+1} = e^k + v^4_{\omega_{\text{opt}}} \tag{27}$$

The above equation states that in each step of the training algorithm, the energy function value is updated by adding $v^4_{\omega_{\text{opt}}}$ to its value in the previous step. According to (16), the activation value of the winning neuron, ω_{opt}, in the fourth layer should be negative; otherwise, the NN convergence is achieved. Consequently, we may write:

$$e^{k+1} < e^k \tag{28}$$

It means that the energy function decreases monotonically during BONN training. Therefore, BONN is stable in the sense of Lyapunov [36] and will converge to a local minimum.

D. Relation between BONN and typical heuristics

As explained in the previous subsections, BONN is trained by causing a competition among the tour neurons in each step and displacing the winning city from its current location to a better location on the tour.

In this regard, BONN is algorithmically similar to the family of 2.5-Opt heuristics [36]. However, BONN is presented here in the context of neural networks, since it uses a feedback configuration similar to HNN and a competitive training algorithm similar to K-SOM. Consequently, the computational complexity of the proposed neural network is comparable with typical NNs for TSP. Furthermore, as will be seen in the next section, BONN is used as an elementary structure to develop a constructive-optimizer neural network with the ability to avoid weak local minima of the energy function. Similar strategy has already been adopted by other researchers [37] who presented well-known 2-Opt and 3-Opt heuristics in terms of NN. Obviously, NN is just an appropriate framework for developing the proposed algorithm and it can be presented and used in different contexts like typical heuristics.

III. Constructive-optimizer neural network

Similar to all energy-based NNs, BONN is highly sensitive to the initial conditions and apt to be caught in local minima of its energy function [35]. Here, a constructive approach is proposed for the initialization and extension of BONN in order to avoid weak local minima of the energy function.

As stated in Section II, all cities in BONN are on the tour (m=n). According to (14), when $m < n$, the competition in the fourth layer occurs only among the tour cities (belonging to the set Q^k). Consequently, the nontour cities (belonging to the set R^k) do not take part in the training process. In each step of the BONN training algorithm, a tour city is displaced from its current location to a better one. Hence, the number of tour cities remains fixed at m during BONN training. A question arises here: how can the tour grow during training? The main idea behind the proposed CONN is that, in each constructive phase, a number of cities from the set R^k are added to the current tour and then, the new tour is optimized by BONN in the optimizer phase.

A. CONN structure

According to (10), the thresholds of nontour neurons are set to zero and their activation value is equal to the tour length increase due to inserting the corresponding city on the tour. Consequently, the nontour city with the smallest activation value in the fourth layer may be the best choice to be added to the tour. In other words, the winning nontour city in the fourth layer (belonging to R^k) is likely the best choice for inserting on the current tour.

(Figure 2)

The above procedure suggests the extension of BONN to a constructive-optimizer structure (CONN) as illustrated in Figure 2. Neurons of the fourth layer in CONN are divided into two parts: *i)* the optimizer part which consists of tour neurons (Q^k) and *ii)* the constructive part including nontour neurons (R^k). The optimizer and constructive parts are in charge of optimizing and growing the tour, respectively. All the neurons in CONN have the same operation as in BONN except the neurons in the fourth layer. In this layer, optimizer part neurons compete with each other according to (14) to optimize the current tour, while constructive part neurons compete according to (29) to extend the tour.

$$\omega_{\text{cns}} = \arg\left(\min_{i\in R^k}(v_i^4)\right) \tag{29}$$

Furthermore, the winning neuron output in the constructive part is obtained by (30):

$$x^4_{\omega_{\text{cns}}} = x^3_{\omega_{\text{cns}},2} \tag{30}$$

Hence:

$$x_i^4 = 0, \qquad i \neq \omega_{\text{opt}}, \omega_{\text{cns}} \tag{31}$$

(Figure 3)

B. CONN training algorithm

The training algorithm of CONN has two phases including constructive and optimizer as shown in the block diagram of Figure 3. The training algorithm starts with generating an initial γ-city tour as will be explained in Subsection III-D. Using the initial tour, the output of the neurons in the first layer is computed according to (18), by setting $m = \gamma$. CONN first extends the tour in the constructive phase. In each step of this phase, a nontour neuron in the fourth layer wins and its corresponding city is inserted on the tour as follows:

$$\boldsymbol{\psi}^{k+1} = [\mathbf{x}_1^k, \mathbf{x}_2^k, \ldots, \mathbf{x}^k_{x^4_{\omega_{\text{cns}}}}, \mathbf{c}_{\omega_{\text{cns}}}, \mathbf{x}^k_{x^4_{\omega_{\text{cns}}}+1}, \ldots, \mathbf{x}_m^k] \tag{32}$$

The winning neuron in the fourth layer is consequently displaced from the constructive part (R^k) to the optimizer part (Q^k). Obviously, adding a new city to the current tour increases the number of neurons in the first and second layers (m) by one. This process is repeated until the number of cities on the tour augments to $m=\gamma+\lambda$.

After construction, the training algorithm switches to the optimizer phase. In each step of this phase, the winning tour neuron in the fourth layer is displaced from its current location to a better location on the tour, according to the BONN training algorithm presented in Table 1. This process is repeated until the NN convergence is achieved in the optimizer phase. Then, the training algorithm switches again to the constructive phase and the same procedure is iterated until the tour includes all cities.

C. CONN convergence analysis

Since CONN uses the same optimization algorithm as BONN, its convergence in the optimizer phase can be demonstrated as in Subsection II-C. Similar to (27), it can be simply shown that in each step of the constructive phase, the energy function value increases as follows:

$$e^{k+1} = e^k + v^4_{\omega_{\text{cns}}} \tag{33}$$

Since in each step of the constructive phase, only one city is inserted on the tour and the number of cities is finite, this phase of the training algorithm can not make the algorithm instable. Indeed, the constructive phase initializes the optimizer phase in each switching stage. Therefore, CONN is also stable in the sense of Lyapunov as well as BONN and finally converges to a local minimum.

D. Initial tour generation

The initial tour may significantly affect the CONN performance. CONN uses only local information to grow and optimize the tour, while the initial tour can provide some global information for it. We studied three different algorithms to generate the initial tour: *i)* the cheapest link [38], *ii)* hull through four outermost cities [39], and *iii)* convex hull [40]. CONN uses one of these algorithms to generate the initial tour for each TSP, based on the total number of cities as will be explained in Subsection IV-B.

i) Cheapest link algorithm: This algorithm is used to create the initial tour by arranging a number of outermost cities on a tour. According to (34), γ outermost cities $\{\mathbf{c}_{\varphi_1},\ldots,\mathbf{c}_{\varphi_\gamma}\}$ can be found by maximizing the average intra-distance between these cities and all the TSP cities as well as maximizing the average inter-distance between them in γ steps.

$$\varphi_{p+1} = \arg\left(\max_{i\in\Phi} \left(\frac{\sum_{j=1}^{n} D(\mathbf{c}_i, \mathbf{c}_j)}{n} + \frac{\sum_{q=1}^{p} D(\mathbf{c}_{\varphi_q}, \mathbf{c}_i)}{p} \right) \right), \qquad \Phi = \{1,2,\ldots,n\} - \{\varphi_1, \varphi_2, \ldots, \varphi_p\} \tag{34}$$

The value of γ may depend on the topology and number of TSP cities. Our simulations resulted in the following experimental equation to determine γ as a function of n for good performance:

$$\gamma = \max(\min([0.032\times n + 6.94], n\text{-}10),\ 4) \tag{35}$$

ii) Hull through four outermost cities: This algorithm simply creates a 4-city tour that consists of four outermost cities ($\gamma = 4$).

iii) Convex hull: This algorithm makes a convex hull as the initial tour whose vertices are chosen from TSP cities such that the resultant hull surrounds all the remaining cities [14]. Obviously, the value of γ is determined by the algorithm and depends on the topology and number of TSP cities.

(Table 2)

E. Computational complexity

The most computationally-complex equations of CONN are (6), (7), and (10) which are of *mn* order (Table 2). These equations are used in both constructive and optimizer phases of CONN. Hence, the overall computational volume of CONN is $O(n^2)\times K$, where K is the total number of CONN iterations. Suppose that the total number of optimizer phase iterations is β. Obviously, the number of constructive phase iterations is $n-\gamma$. Hence, the total number of training algorithm iterations is $K = n + \beta - \gamma$. As will be demonstrated in Subsection IV-B, K is usually smaller than 2n, since we generally have $\beta < n$. Therefore, the overall computational complexity of CONN seems to be of $O(n^3)$.

However, in each step of the optimizer phase, only one tour city is displaced. Consequently, the above equations should not necessarily be computed for all cities. More specifically, suppose that the city $\mathbf{c}_{\omega_{\text{opt}}}$ has won in the optimizer phase of the training algorithm and it has been placed at the location a on the current tour ($\mathbf{x}_a^1 = \mathbf{c}_{\omega_{\text{opt}}}$). The next tour will be generated by displacing the city $\mathbf{c}_{\omega_{\text{opt}}}$ from the location a to the new location between cities in the link specified by $x^4_{\omega_{\text{opt}}}$ on the current tour. Hence, (10) should be computed only for five cities including $\mathbf{x}^k_{a-1}, \mathbf{x}^k_a, \mathbf{x}^k_{a+1}, \mathbf{x}^k_{x^4_{\omega_{\text{opt}}}}$ and $\mathbf{x}^k_{x^4_{\omega_{\text{opt}}}+1}$. Similarly, if in the constructive phase the city $\mathbf{c}_{\omega_{\text{cns}}}$ wins, (10) will be computed only for three cities including $\mathbf{x}^k_{x^4_{\omega_{\text{cns}}}}, \mathbf{x}^k_{x^4_{\omega_{\text{cns}}}+1}$ and $\mathbf{x}^k_{x^4_{\text{cns}}+2}$. Therefore, the computational complexity of (10) is reduced to $O(n)$ as shown in the third column of Table 2.

In the same manner, (6) should be computed for only five links including:

$$M_{\text{opt}} = \left\{ \mathbf{x}^2_{a-1}, \mathbf{x}^2_a, \mathbf{x}^2_{x^4_{\omega_{\text{opt}}}}, \mathbf{x}^2_{x^4_{\omega_{\text{opt}}}+1}, \mathbf{x}^2_{x^4_{\omega_{\text{opt}}}+2} \right\} \tag{36}$$

in each step of the optimizer phase and two links including:

$$M_{\text{cns}} = \left\{ \mathbf{x}^2_{x^4_{\omega_{\text{cns}}}}, \mathbf{x}^2_{x^4_{\omega_{\text{cns}}}+1} \right\} \tag{37}$$

in each step of the constructive phase. Hence, the computational complexity of (6) is also reduced to $O(n)$ as shown in Table 2.

Furthermore, efficient implementation of (7) can reduce its complexity. Note that in each step of the optimizer phase, only a limited number of $v^4_{i,j}$ are modified. In more details, for the *i*-th neuron in the third layer, if the best link in the previous step ($x^3_{i,2}$) does not change in the current step:

$$\mathbf{x}^2_{x^3_{i,2}} \notin M_{\text{opt}} \tag{38}$$

we will redefine P_i as follows:

$$P_i = M_{\text{opt}} \cup \left\{ \mathbf{x}^2_{x^3_{i,2}} \right\} - \left\{ j \middle| j = 1,2,\ldots m; \mathbf{x}^2_{j,1}, \mathbf{x}^2_{j,2} \neq \mathbf{c}_i \right\}. \tag{39}$$

Otherwise, the set P_i is computed by (8). Using (39) for all neurons results in the computational complexity of $O(n)$ for (7). However, (38) may not be satisfied for all cities. Consequently, using the above efficient implementation, the complexity order of (7) reduces to $o(n^2)$. In the same manner, similar results can be obtained for the constructive phase. Therefore, the overall computational complexity of CONN reduces to $o(n^3)$ as shown in Table 2 and Figure 7.

(Figure 4)

F. Example

We give an example to further clarify the CONN operation. Consider a 6-city TSP (n=6), consisting of the following cities:

$$\mathbf{c}_1 = [0,0],\ \mathbf{c}_2 = [0,1],\ \mathbf{c}_3 = [0,2],\ \mathbf{c}_4 = [1,2],\ \mathbf{c}_5 = [1,1],\ \mathbf{c}_6 = [1,0]$$

where each vector indicates the coordinates of the corresponding city in the 2-dimensional space. Suppose that the initial tour is $\boldsymbol{\psi}^0 = [\mathbf{c}_2, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_3, \mathbf{c}_6]$ as shown in Figure 4-a, setting m=5 and k=0. Furthermore, the function $D(.)$ computes the Manhattan distance between each city pair. According to (18), the outputs of the first layer neurons are determined as follows:

$$\mathbf{x}_1^1 = \mathbf{c}_2,\ \mathbf{x}_2^1 = \mathbf{c}_4,\ \mathbf{x}_3^1 = \mathbf{c}_5,\ \mathbf{x}_4^1 = \mathbf{c}_3,\ \mathbf{x}_5^1 = \mathbf{c}_6$$

Using (11) and (12), Q^0 and R^0 are defined as $Q^0 = \{\mathbf{c}_2, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_3, \mathbf{c}_6\}$ and $R^0 = \{\mathbf{c}_1\}$, respectively. The current value of the energy function is given by (2) as $e^0 = 2+1+2+3+2 = 10$. Now, suppose that the NN is trained in the optimizer phase. The output vectors of the second layer neurons are given by (4) as follows:

$$\mathbf{x}_1^2 = [\mathbf{c}_2, \mathbf{c}_4],\ \mathbf{x}_2^2 = [\mathbf{c}_4, \mathbf{c}_5],\ \mathbf{x}_3^2 = [\mathbf{c}_5, \mathbf{c}_3],\ \mathbf{x}_4^2 = [\mathbf{c}_3, \mathbf{c}_6],\ \mathbf{x}_5^2 = [\mathbf{c}_6, \mathbf{c}_2]$$

The output vector of the threshold neuron in the third layer is computed by (5) as $\mathbf{t}^3 = [2,1,2,3,2]$. The activation values of the remaining neurons in this layer are determined by (6) as follows:

$$\mathbf{v}_1^3 = [2,4,2,0,0],\quad \mathbf{v}_2^3 = [0,2,1,0,0],\quad \mathbf{v}_3^3 = [0,2,0,0,2]$$
$$\mathbf{v}_4^3 = [0,0,0,0,2],\quad \mathbf{v}_5^3 = [0,0,0,0,0],\quad \mathbf{v}_6^3 = [2,2,2,0,0]$$

According to (7), the outputs of these neurons are given by:

$$\mathbf{x}_1^3 = [0,4],\quad \mathbf{x}_2^3 = [0,4],\quad \mathbf{x}_3^3 = [0,1]$$
$$\mathbf{x}_4^3 = [0,3],\quad \mathbf{x}_5^3 = [0,1],\quad \mathbf{x}_6^3 = [2,1]$$

The output vector of the threshold neuron in the fourth layer is determined by (10) as $\mathbf{t}^4 = [0,2,4,2,2,4]$. Then, the activation values of the fourth layer neurons are computed by (13) as:

$$v_1^4 = 0,\quad v_2^4 = -2,\quad v_3^4 = -4,\quad v_4^4 = -2,\quad v_5^4 = -2,\quad v_6^4 = -2$$

Therefore, according to (14), the third neuron in the fourth layer wins ($\omega_{\text{opt}} = 3$). The outputs of the neurons in this layer are given by (16), (30) and (31) as:

$x_1^4 = 0,\quad x_2^4 = 0,\quad x_3^4 = 1,\quad x_4^4 = 0,\quad x_5^4 = 0,\quad x_6^4 = 0$

Hence, the city $\mathbf{c}_3$ will be displaced from the 4th location to the 2nd location on the tour (Figure 4-b):

$$\boldsymbol{\psi}^1 = [\mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6]$$

According to (18), the new outputs of the first layer neurons will be (setting k=1):

$$\mathbf{x}_1^1 = \mathbf{c}_2,\ \mathbf{x}_2^1 = \mathbf{c}_3,\ \mathbf{x}_3^1 = \mathbf{c}_4,\ \mathbf{x}_4^1 = \mathbf{c}_5,\ \mathbf{x}_5^1 = \mathbf{c}_6$$

The current value of the energy function is $e^1 = e^0 + v_{\omega_{\text{opt}}}^4 = 10 - 4 = 6$ (see Eqn. 27). It can be easily shown that CONN can not further improve the current tour and it converges in the optimizer phase. Now, the CONN training algorithm switches to the constructive phase. In this phase, CONN proceeds forward in the same manner as in the optimizer phase. The activation values of the fourth layer neurons are obtained as follows:

$v_1^4 = 0,\quad v_2^4 = 2,\quad v_3^4 = 2,\quad v_4^4 = 2,\quad v_5^4 = 0,\quad v_6^4 = 0$

According to (29), the first neuron in the fourth layer wins ($\omega_{\text{cst}} = 1, x_{\omega_{\text{cst}}}^4 = 5$). Using (30)-(32) we have:

$$\boldsymbol{\psi}^1 = [\mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_1]$$

It means that the nontour city $\mathbf{c}_1$ is inserted at the 6th location of the tour as shown in Figure 4-c. Now, the current value of the energy function is $e^2 = e^1 + v_{\omega_{\text{cns}}}^4 = 6 + 0 = 6$ (see Eqn. 33). At this stage, all cities have been added to the tour. Therefore, the CONN training algorithm switches to the optimizer phase again. The training algorithm can not further improve the tour. Hence, CONN converges to the final solution.

(Table 3)

IV. Experimental results

The performance of CONN was evaluated using nine sets of experiments. All the experimental results were obtained by an AMD ATHLON XP 1.4-GHz PC with 1-GB main memory using Matlab environment. For comparing CONN with other algorithms in terms of CPU time, we scaled the CPU time of each algorithm by an appropriate scaling coefficient related to its processing system. Similar to the approach used in [34], we utilized the results reported in [41] to obtain the scaling coefficients as shown in Table 3. Note that the codes made by 3GL programming languages such as C/C++, PASCAL, and FORTRAN are more efficient than M-codes in the MATLAB interpreter [42]. Nevertheless, we did not consider any scaling coefficient for comparing MATLAB M-codes with 3GL codes. In other words, it is expected to obtain better performance by implementing our algorithm using an efficient programming language like C++.

A. Adjusting the CONN parameters

Our primary experiments were performed on a set of 21 benchmark TSPs taken from a frequently-used TSP library called TSPLIB collected by Reinelt [43]. The number of cities ranges from 52 (small-scale) to 5915 (large-scale). CONN was used to solve each problem using 5 different values for λ. For each λ, CONN was executed 10 times to obtain the average CPU time (T). The percent differences have been computed using (40):

$$\delta = \frac{l - l_{\text{opt}}}{l_{\text{opt}}} \times 100 \tag{40}$$

where, l and l_{opt} are the algorithm and optimal tour lengths, respectively. Note that for each TSP, CONN gives the same solution in all runs. Hence, the average percent difference ($\bar{\delta}$) is equal to each percent difference for CONN, i.e. $\bar{\delta}_{\text{CONN}} = \delta_{\text{CONN}}$. The results are shown in Table 4. The best solution for each benchmark TSP is shown by bold-faced text. As shown, augmenting the number of TSP cities (n) increases the appropriate number of cities to be added in each constructive phase (λ_a). As illustrated in Figure 5, the following equation can be fitted for determining λ_a as a function of n:

$$\lambda_a = \max\left(\left\lceil 38\times(\ln n)^3 - 631\times(\ln n)^2 + 3476.9\times(\ln n) - 6334 \right\rceil, 10\right) \tag{41}$$

(Table 4)

(Figure 5)

B. CONN performance on TSPLIB problems

The second set of experiments was performed on 91 benchmark TSPs from TSPLIB with 14 to 5923 cities. CONN was executed for each TSP with three different initial tour generation algorithms including the cheapest link, convex hull, and hull through four outermost cities. The results are listed in Table 5. For each benchmark problem, the first four columns of this table are: *i)* the TSP name, *ii)* number of cities (n), *iii)* optimal tour length (optimal solution) as reported in TSPLIB, and *iv)* appropriate number of cities to be added in each constructive phase (λ_a) as given by (41), respectively. For each initial tour generation algorithm, the subsequent columns give the CONN percent difference ($\bar{\delta} = \delta$), number of cities on the initial tour (γ), number of optimizer phase iterations (β), number of total CONN iterations (K), and average CONN CPU Time per run, respectively. As shown, the cheapest link algorithm gives the best average results. More specifically, the cheapest link algorithm provides the best performance for TSPs with $n < 130$. The convex hull algorithm performs better for $130 < n < 900$. Finally, the hull through four outermost cities is more appropriate for $900 < n$. If TSP cities' coordinates are not strictly specified, the convex hull algorithm can not be used for the initial tour generation. In these cases, the cheapest link algorithm may be used. In the experiments reported hereafter, the above criteria were used to generate the CONN initial tour. The CONN solutions using this arrangement of algorithms are indicated in Table 5 by gray background.

(Table 5)

(Figure 6)

As shown in Table 5, for all benchmark TSPs we have $\beta/n < 0.5$, and consequently $K/n < 1.5$ (Figure 6). Hence, according to the discussion presented in Subsection III-E, the overall computational order of CONN is $o(n^3)$.

(Figure 7)

Figure 7 shows the CONN CPU time versus the number of cities for all benchmark TSPs. In this figure, the CPU time versus *n* has been fitted to a polynomial whose degree is less than 3 (2.7).

(Figure 8)

As illustrated in Figure 8, the solution qualities of CONN are between 0.0% and 14% for all benchmark TSPs. Furthermore, CONN gave the optimal solutions for 5 benchmark TSPs.

C. Comparing with computationally-complex algorithms

In this section, we compared CONN with several non-neural computationally-complex algorithms. First, the performance of CONN was compared with those of 2-Opt and 4-Opt heuristics, an accurate variant of SA (SA_1), and iterated tabu search (ITS), which were reported in [28]. The third set of experiments was performed on 10 benchmark TSPs from TSPLIB with 99 (small-scale) to 493 cities (medium-scale). The results are given in Table 6. CONN gave shorter tours than 2-Opt heuristic and converged several times faster than it. However, CONN caused 0.7%, 3.6%, and 3.7% suboptimalities with respect to 4-Opt heuristic, SA_1 and ITS, respectively and converged significantly faster than all of them. In more details, the computational complexity of these counterpart algorithms is $O(n^2)\times K$ where K is the total number of tour generations (iterations). As shown in Table 6, their CPU time as well as K increased rapidly by augmenting the number of cities. In other word, K increased as a function of n^l where $l \geq 1$. Hence, the overall computational complexity of these non-neural approaches is $\Omega(n^3)$. Therefore, CONN rapidly outstrips them, since its computational complexity is $o(n^3)$. For example, to solve d198, CONN converged 5 and 1440 times faster than 2-Opt and 4-Opt heuristics, respectively. Another example is d493 for which CONN converged 85 and 240 times faster than SA_1 and ITS. Figure 9 illustrates these results in more details by comparing the CPU times of CONN and SA_1 for 64 benchmark TSPs from TSPLIB (the fourth set of experiments) with 14 to 493 cities.

(Table 6)

(Figure 9)

The fifth set of experiments was performed on 14 benchmark TSPs from TSPLIB with 532 to 4461 cities in order to compare the performances of CONN and three non-neural computationally-complex algorithms including: *i)* greedy Lin–Kernighan (Greedy-LK) [44], *ii)* max-min ant system (MMAS) [25], and *iii)* an evolutionary algorithm called NEA that utilizes the natural crossover operator [30]. These algorithms are among the most accurate algorithms for TSP. According to Table 7, CONN resulted in significant efficiency improvement and caused 8.1%, 8.3%, and 6.3% suboptimalities with respect to Greedy-LK, MMAS, and NEA, respectively. It can be easily shown that the computational complexity of these counterparts may be also $\Omega(n^3)$; hence CONN outstrips them quickly. For example, to solve fl3795, CONN converged 18 times faster than Greedy LK. Another example is fl1577 for which CONN converged 65 times faster than MMAS. The final example is fnl4461 where CONN converged 37 times faster than NEA.

Note that classical well-known heuristics like Greedy LK may result in better performance in some problems by extensively tweaking and tuning the algorithm [44]. However, CONN is based on a much simpler and more general-purpose approach. In other words, CONN is not a tool to compete with designer algorithms [36]. Nevertheless, the above experimental results demonstrated that CONN can converge several times faster than such heuristics while providing a reasonable percent difference in the solution quality.

(Table 7)

(Table 8)

D. Comparing with computationally-comparable algorithms

Table (8) compares CONN with several well-known heuristics including nearest neighbor, greedy, Clarke-Wright, and Christofides whose computational complexities were reported in [36]. As shown, CONN has a competitive computational complexity with respect to these algorithms. However, as illustrated in Figure 8, it is apparent that CONN's result extrapolates to being more than 10% above optimal for $n \geq 10^4$. In this case, Christofides may give better results compared to CONN [36].

In the following experiments, we compared CONN with several state-of-the-art NNs, including KNIES NNs [45], Budinich's SOM [16], ESOM [14], eISOM [11], Co-Adaptive Net [34], and multi-valued Hopfield network (MVHN) [37] as well as a computationally-comparable variant of SA (SA_2) [14]. These algorithms have comparable computational complexities with respect to CONN. However, it was claimed that KNIES NNs had been the most efficient NN formerly presented in the literature for TSP [45, 46]. Similarly, ESOM [14] and eISOM [11] were introduced as the most accurate neural networks for TSP. Also, Co-Adaptive Net was proposed as a neural approach with better performance than other NNs in terms of accuracy and/or CPU time [34].

The basic idea of KNIES NNs is dispersing output neurons after SOM learning to make their statistics equal to that of some cities. If all cities participate, it leads to the global version, KNIES_TSP_Global (KG).

If only the represented cities are involved, it leads to the local version, KNIES_TSP_Local (KL). KNIES_DECOMPOSE (KD) firstly decomposes large-scale TSPs by clustering cities using the learning vector quantization approach. It then uses KNIES_TSP_Global to find a tour among the cluster centers. Finally, it glues the Hamiltonian paths which are computed by KNIES_HPP_Global [47] for each cluster. Budinich's SOM is an effective implementation of the traditional SOM that maps each city onto a linear order without ambiguity [16]. ESOM incorporates the neighborhood preserving and convex-hull properties of TSP to generate shorter tours than Budinich's SOM and CEN [14]. Furthermore, eISOM optimally integrates the above two TSP properties used in ESOM with the mechanism of dragging excited neurons toward the input cities [11]. Finally, Co-Adaptive Net involves not only unsupervised learning to train neurons, but also allows neurons to co-operate and compete amongst themselves depending on their situation [34].

The sixth set of experiments was conducted on 37 benchmark TSPs from TSPLIB with 51 to 5934 cities. Table 9 lists the results of CONN and its nine counterparts. The best solution for each benchmark problem is shown by bold-faced text. It may be observed from Table 9 that all algorithms can generate good tours. The range of CONN solution qualities is at least 1.7% less than those of its counterparts except eISOM. Furthermore, CONN produced a superior average solution quality than its counterparts except ESOM and eISOM. In more details, CONN provided 1.2%, 2.9%, 0.7%, 1.0%, 1.3%, 0.5%, and 6.1% improvements over SA_2, KD, KL, KG, Budinich's SOM, Co-Adaptive Net, and MVHN (two optimal), respectively. Although CONN caused 0.6% and 2.2% suboptimalities with respect to ESOM and eISOM, respectively, it converged significantly faster than both as will be further demonstrated. Moreover, CONN provided the best solution for 13 benchmark TSPs. CONN has a smaller computational complexity than those of its counterparts; hence, it eventually outstrips them as demonstrated in the following experiments.

(Table 9)

(Figure 10)

The seventh set of experiments were performed to compare the CPU time of CONN, Budinich's SOM, and ESOM [14], for 20 benchmark TSPs from TSPLIB. The number of cities in these benchmark problems is between 51 and 1748. As illustrated in Figure 10, CONN converged, at least, 10 times faster compared to its counterparts for all benchmark TSPs. Regarding the solution qualities reported in Table 9, CONN performed significantly better than Budinich's SOM and ESOM. The execution time of eISOM is about 1.6 times longer than those of ESOM and Budinich's SOM [11]. Consequently, CONN is more efficient than eISOM, ESOM and Budinich's SOM. For example, the CPU time of CONN, Budinich's SOM, ESOM, and eISOM are 12, 218, 228, and (about) 365 seconds, respectively, for vm1748.

The eighth set of experiments was run to compare CONN with Co-Adaptive Net in terms of CPU time for 20 benchmark TSPs from TSPLIB with 100 to 5934 cities. As shown in Figure 11 and Table 9, CONN beats Co-Adaptive Net in terms of CPU time and solution quality.

(Figure 11)

The last set of experiments was mainly designed to compare CONN with KNIES NNs including KD, KL, and KG in terms of CPU time for 18 benchmark TSPs from TSPLIB with 51 to 1002 cities. The experimental results illustrated in Figure 12 and Table 9 demonstrate that CONN beats KL in terms of both CPU time and solution quality. KG converged faster than CONN for TSPs with $n < 442$. However, for larger *n*, CONN outstripped it since its computational order is smaller than KG. Although CONN converged slower than KD, it provided 2.9% quality improvement over that (Table 9). Furthermore, the computational complexity of CONN is smaller than KD and it eventually outstrips KD for TSPs with larger size. Generally, among KNIES NNs, KL is the slowest NN with the highest accuracy, while KD is the fastest one with the poorest solution quality.

(Figure 12)

V. Conclusion

In this paper, we proposed a constructive-optimizer NN called CONN for TSP. The main purpose of designing CONN is to obtain a fast method while achieving near to optimal solution quality. CONN uses a feedback-type structure similar to HNN and a competitive training algorithm similar to K-SOM. The main idea behind the CONN training algorithm is as follows. CONN is firstly initialized by an initial tour. Then, it grows and optimizes the current tour until all cities are added. CONN utilizes three different algorithms including the cheapest link, convex hull, and hull through four outermost cities for generating the initial tour.

For each TSP, it chooses one of these algorithms depending on the number of TSP cities. Furthermore, CONN provides competitive efficiency compared to the previously introduced NNs for TSP. Its computational intensity is less than $o(n^3)$.

CONN was compared with several computationally-complex algorithms including 4-Opt heuristics, SA_1, ITS, Greedy-LK, MMAS, and NEA for performance evaluation. Although these algorithms slightly outperformed CONN in terms of accuracy, their computational complexity was several times larger which limits their applications.

The CONN performance was also compared with those of several computationally comparable NNs, including KNIES NNs, Budinich's SOM, ESOM, eISOM, Co-Adaptive Net, and MVHN as well as SA_2. These state-of-the-art NNs were outperformed by CONN in terms of accuracy and/or CPU time. It provided a better compromise between the CPU time and solution quality.

CONN uses a competitive training algorithm based on the minimum added tour length criterion. Its performance can be improved by considering other criteria such as the shortest link for growing and optimizing the tour in the training algorithm. Moreover, the initial tour has an important role in the CONN performance. Hence, extending the CONN training algorithm as well as its structure to include other criteria and improving the initial tour may result in better CONN performance in terms of both time and accuracy.

Acknowledgement

The authors appreciate the members of robotics group of Ferdowsi university of Mashhad for definition of the problem and Dr. Nikbakht for his useful editing comments.

References

[1] P. Crescenzi and V. Kann, "A compendium of NP optimization problems," Oct. 1995. Available: http:// www.zvne.fer.hr/ ~zmija/ resources/ science_resources/nn_for_optimization/index.html

[2] K. Fujimura, K. Obu-Cann, and H. Tokutaka, "Optimization of surface component mounting on the printed circuit board using SOM-TSP method," in Proc. *6th Int'l Conf. Neural Information Processing* (*ICONIP'99*), vol. 1, 1999, pp. 131–136.

[3] K. Fujimura, S. Fujiwaki, O.-C. Kwaw, and H. Tokutaka, "Optimization of electronic chip-mounting machine using SOM-TSP method with 5 dimensional data," in Proc. *Int'l Conf. Info-tech and Info-net (ICII'01)*, vol. 4, 2001, pp. 26–31.

[4] M. K. Mehmet Ali and F. Kamoun, "Neural networks for shortest tour computation and routing in computer networks," *IEEE Trans. Neural Networks*, vol. 4, no. 5, pp. 941–953, 1993.

[5] T. Onoyama, T. Maekawa, S. Kubota, Y. Taniguchi, and S. Tsuruta, "Intelligent evolutional algorithm for distribution network optimization," in Proc. *Int'l Conf. Control Applications*, vol. 2, 2002, pp. 802–807.

[6] D. Banaszak, G.A. Dale, A.N. Watkins, and J.D. Jordan, "An optical technique for detecting fatigue cracks in aerospace structures," in Proc. *18th Int'l Cong. Instrumentation in Aerospace Simulation Facilities (ICIASF)*, 1999, pp. 27/1–27/7.

[7] D. Barrel, J.-P. Perrin, E. Dombre, and A. Liengeois, "An evolutionary simulated annealing algorithm for optimizing robotic task ordering," *IEEE Int'l Sym. Assembly and Task Planning (ISATP)*, 1999, pp. 157–162.

[8] C.-H. Cheng, W.-K. Lee, and K.-F. Wong, "A genetic algorithm-based clustering approach for database partitioning," *IEEE Trans. Systems, Man, and Cybernetics-Part C: Applications and Reviews*, vol. 32, no. 3, pp.:215–230, 2002.

[9] J. Gu and X. Huang, "Efficient local search with search space smoothing: a case study of the traveling salesman problem (TSP)," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 24, no. 5, pp. 728–735, 1994.

[10] F. Tian and L. Wang, "Chaotic simulated annealing with augmented Lagrange for solving combinatorial optimization problems," in Proc. *26th Annual Conf. of the IEEE Industrial Electronics Society (IECON'00)*, vol. 4, 2000, pp. 2722–2725.

[11] H.-D. Jin, K.-S. Leung, M.-L. Wong, and Z.-B. Xu, “An efficient self-organizing map designed by genetic algorithms for the traveling salesman problem,” *IEEE Trans. Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 33, no. 6, pp. 877–888, 2003.

[12] B. Fritzke and P. Wilke, “FLEXMAP-A neural network with linear time and space complexity for the traveling salesman problem,” in Proc. *Int'l Joint Conf. Neural Networks*, pp. 929–934, 1991.

[13] L. I. Burke and P. Damany, “The guilty net for the traveling salesman problem,” *Computers and Operations Research*, vol. 19, no. 3/4, pp. 255–266, 1992.

[14] K. S. Leung, H. D. Jin, and Z. B. Xu, “An expanding self-organizing neural network for the traveling salesman problem,” *Neurocomputing*, vol. 62, pp. 267–292, 2004.

[15] H. Al-Mulhem and T. Al-Maghrabi, “Efficient convex-elastic net algorithm to solve the Euclidean traveling salesman problem,” *IEEE Trans. Systems, Man, Cybernetics-Part B: Cybernetics*, vol. 28, no. 4, pp. 618–620, 1998.

[16] M. Budinich, “A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing,” *Neural Computation*, vol. 8, no. 2, pp. 416–424, 1996.

[17] Y. Takahashi, “Mathematical improvement of the Hopfield model for feasible solution to the traveling salesman problem by a synapse dynamical system,” *IEEE Trans. Systems, Man, Cybernetics-Part B: Cybernetics*, vol. 28, no. 6, pp. 906–919, 1998.

[18] S. Abe and A. H. Gee, “Global convergence of the Hopfield neural network with nonzero diagonal elements,” *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 1, pp. 39–45, 1995.

[19] J. B. Shakleford, “Neural data structures: Programming with neurons,” *Hewlett-Packard Journal*, pp. 69–78, 1989.

[20] S. Bhide, N. John, and M. R. Kabuka, “A real-time solution for the traveling salesman problem using a Boolean neural network,” in Proc. *Int’l Conf. Neural Networks (ICNN’93)*, vol. 2, 1993, pp. 1096–1103.

[21] Y. He and L. Wang, “Chaotic neural networks and their applications,” *3rd World Cong. Intelligent Control and Automation*, vol. 2, 2000, pp. 826–830.

[22] L. Jiao and L. Wang, “A novel genetic algorithm based on immunity,” *IEEE Trans. Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 30, no. 5, pp. 552–561, 2000.

[23] R. Baraglia, J. I. Hidalgo, and R. Perego, “A hybrid heuristic for the traveling salesman problem,” *IEEE Trans. Evolutionary Computation*, vol. 5, no. 6, no. 613–622, 2001.

[24] D. B. Fogel, “Applying evolutionary programming to selected traveling salesman problems,” *Cybernetics and Systems*, vol. 24, pp. 27–36, 1993.

[25] T. Stützle and M. Dorigo, “ACO algorithms for the traveling salesman problem,” In K. Miettinen, M. Makela, P. Neittaanmaki, J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, Wiley, 1999.

[26] M. Dorigo and L. M. Gambardella, “Ant colonies for the traveling salesman problem,” *BioSystems*, vol. 43, pp. 73–81, 1997.

[27] Z. He, C. Wei, B. Jin, W. Pei, and L. Yang, “A new population-based incremental learning method for the traveling salesman problem,” in Proc. *Cong. Evolutionary Computation*, vol. 2, 1999, pp. 1152–1156.

[28] A. Misevičius, “Using iterated tabu search for the traveling salesman problem,” *Informacinės Technologijos ir Valdymas*, vol. 3, no. 32, pp. 29–40, 2004.

[29] S. P. Coy, B. L. Golden, and G. C. Runger, E. A. Wasil, “See the forest before the trees: fine-tuned learning and its application to the traveling salesman problem”, *IEEE Trans. Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 28, no. 4, pp. 454–464, 1998.

[30] S. Jung and B.-R. Moon, “Toward minimal restriction of genetic encoding and crossovers for the two-dimensional Euclidean TSP,” *IEEE Trans. Evolutionary Computing*, vol. 6, no. 6, pp. 557–565, Dec. 2002.

[31] L. Ingber, B. Rosen, “Genetic algorithms and very fast simulated reannealing: a comparison,” *Mathematical and Computer Modeling*, vol. 16, no. 11, pp. 87–100, 1992.

[32] K. Smith, “An argument for abandoning the traveling salesman problem as a neural network benchmark”, *IEEE Trans. Neural Networks*, vol. 7, no. 6, pp. 1542–1544, 1996.

[33] B. W. Lee and B. J. Sheu, “Modified Hopfield neural networks for retrieving the optimal solution”, *IEEE Trans. Neural Networks*, vol. 2, no. 1, pp. 137–142, 1991.

[34] E.M. Cochrane and J.E. Beasley, “The co-adaptive neural network approach to the Euclidean traveling salesman problem,” *Neural Networks*, vol. 16, pp. 1499–1525, 2003.

[35] M. Saadatmand-Tarzjan, M.-R. Akbarzadeh-T., M. Khademi, “A novel combinatorial constructive neural network for the traveling salesman problem and shortest path problem with specified city number,” *Journal of Engineering Faculty of University of Tehran*, vol. 39, no. 4, pp. 469-487, 2005.

[36] D.S. Johnson, L.A. McGeoch, “The traveling salesman problem: a case study in local optimization,” in the Book *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra (eds.), John Wiley and Sons, London, 1997, pp. 215-310.
Available: http://www.research.att.com/~dsj/papers/TSPchapter.pdf.

[37] E. Mérida-Casermeiro, G. Galán-Marín, and J. Muñoz-Peréz, “An efficient multivalued Hopfield network for the traveling salesman problem,” *Neural Processing Letters*, vol. 14, pp. 203–214, 2001.

[38] G. Gutin, A. Yeo, and A. Zverovitch, “Chapter 1: Exponential neighborhoods and domination analysis for the TSP,” In G. Gutin, A. Punnen, editors, *The traveling salesman problems and its variations*, Kluwer Academic Publishers, 2002.

[39] O. Lahyani, E. Oertli, and H. Eberle, “Optimizing Drill Tapes for Printed Circuit Boards”, *International Workshop on Autonomic Communication (WAC’96)*, vol. 3, 1996.

[40] G. Laporte, “The traveling salesman problem: An overview of exact and approximate algorithms,” *Eur. J. Oper. Res.*, vol. 59, pp. 231–247, 1992.

[41] J.J. Dongarra, Performance of various computers using standard linear equations software. Available: http://www.netlib.org/benchmark/performance.ps

[42] *MATLAB Compiler Version 3: User’s Guide*. MathWorks, 2002.

[43] G. Reinelt, “TSPLIB–A traveling salesman problem library,” *ORSA J. Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[44] R. Baraglia, J.I. Hidalgo, and R. Perego, “A hybrid heuristic for the traveling salesman problem,” *IEEE Trans. Evolutionary Computation*, vol. 5, no. 6, pp. 613–622, Dec. 2001.

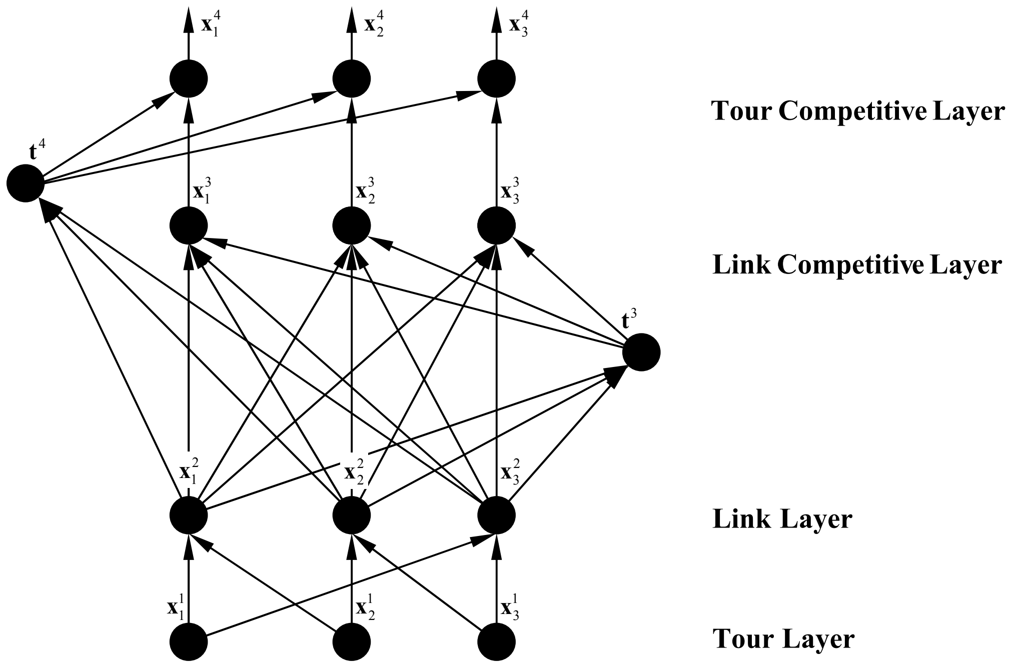
[45] N. Aras, İ.K. Altınel, and J. Oommen, “A Kohonen-like decomposition method for the Euclidean traveling salesman problem–KNIES_DECOMPOSE,” *IEEE Trans. Neural Networks*, vol. 14, no. 4, pp. 869–890, 2003.

[46] N. Aras, B. J. Oommen, İ.K. Altınel, “Kohonen network incorporating explicit statistics and its application to the traveling salesman problem,” Neural Networks, vol. 12, no. 9, pp. 1273–1284, 1999.

[47] İ.K. Altınel, N. Aras, and B. J. Oommen, “Fast, efficient and accurate solutions to the Hamiltonian path problem using neural approaches,” *Comput. Operations Res.*, vol. 27, pp. 461–494, 2000.

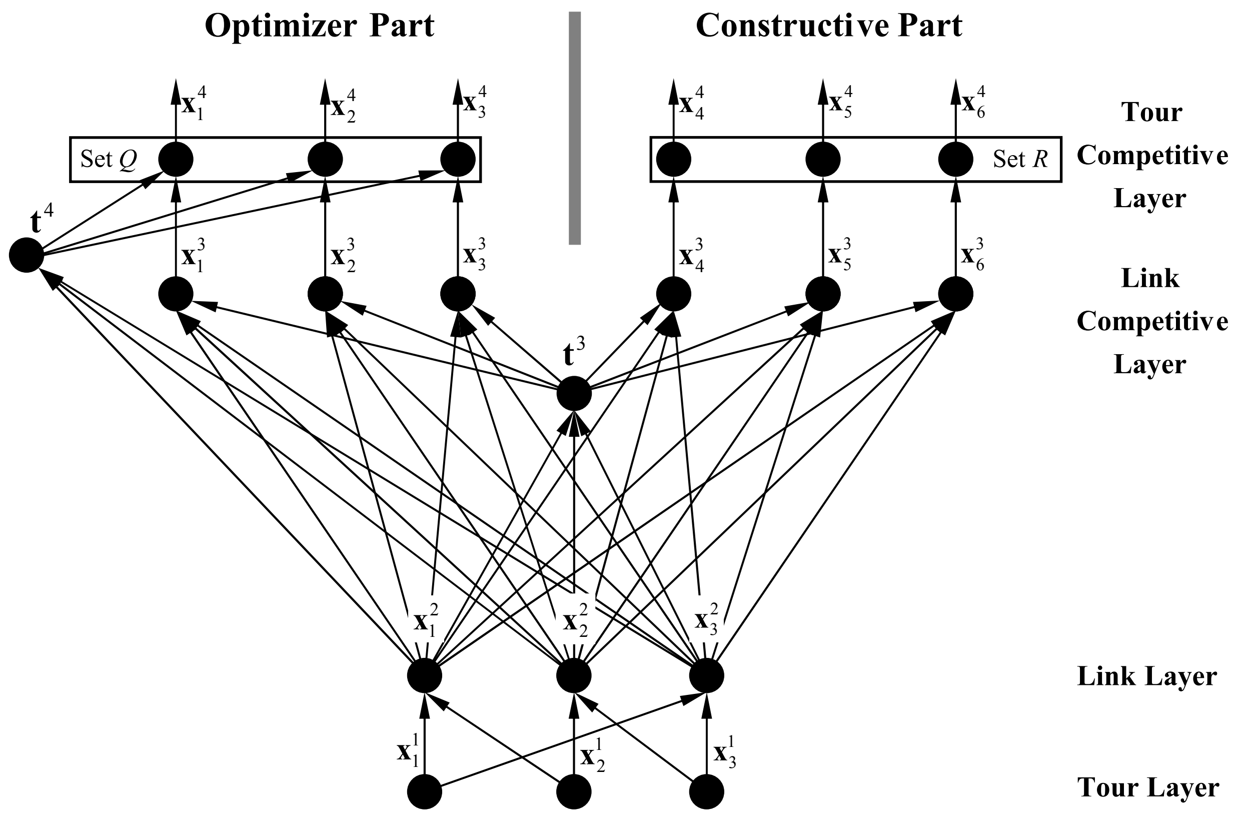
List of Captions of Figures

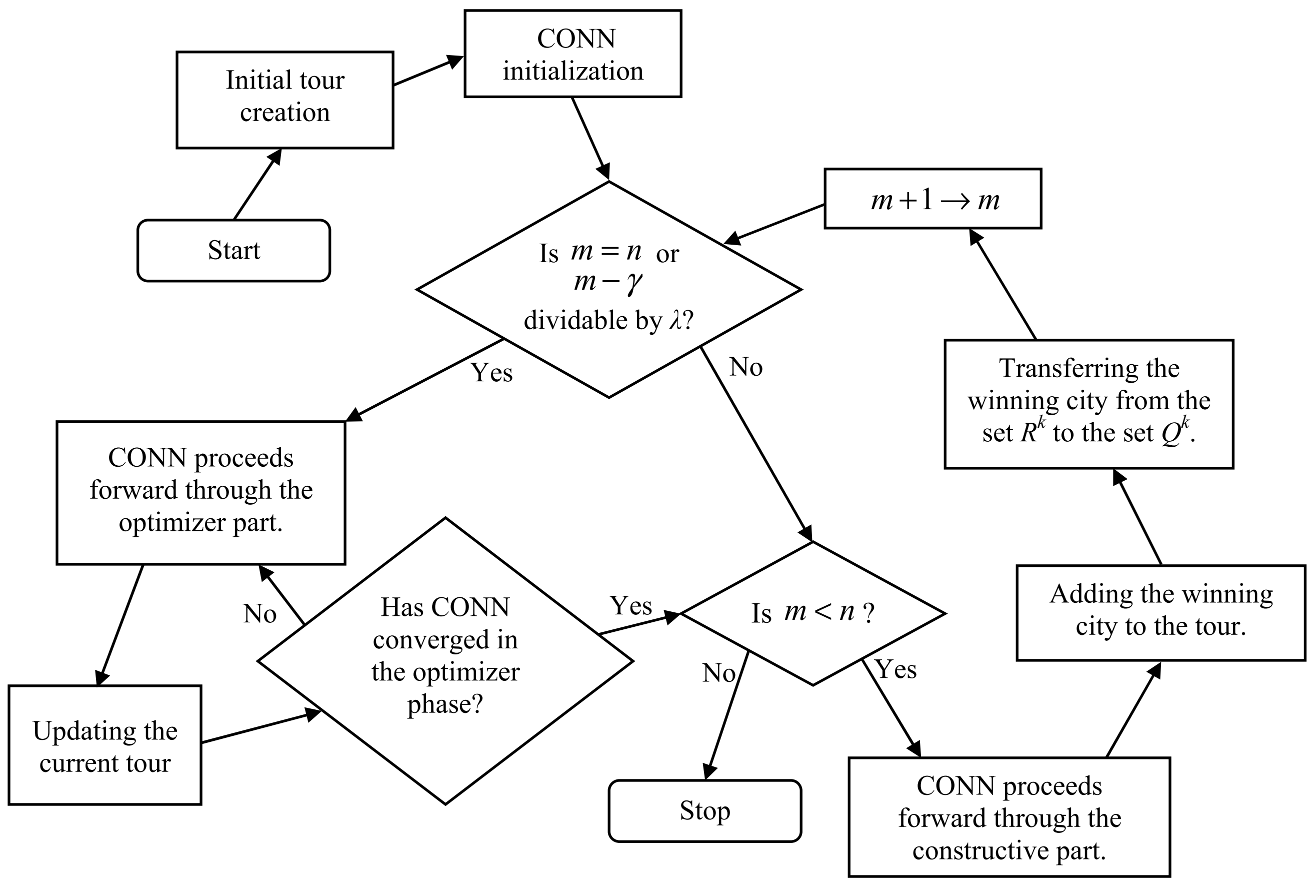
List of Captions of Tables



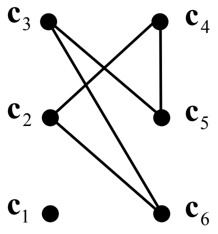
$\mathbf{x}_1^4$
$\mathbf{x}_2^4$
$\mathbf{x}_3^4$
Tour Competitive Layer
$\mathbf{t}^4$
$\mathbf{x}_1^3$
$\mathbf{x}_2^3$
$\mathbf{x}_3^3$
Link Competitive Layer
$\mathbf{t}^3$
$\mathbf{x}_1^2$
$\mathbf{x}_2^2$
$\mathbf{x}_3^2$
Link Layer
$\mathbf{x}_1^1$
$\mathbf{x}_2^1$
$\mathbf{x}_3^1$
Tour Layer

1. A valid tour satisfying (3) is selected as the initial tour, $\boldsymbol{\psi}^0$, setting k=0.
2. Outputs of the first layer neurons are initialized using the initial tour as follows:
   $$\mathbf{x}_j^1 = \boldsymbol{\psi}_j^k, \qquad j = 1,2,\ldots,m \tag{18}$$
3. NN is iterated once.
4. If no neuron in the fourth layer wins, NN is converged and the training is over.
5. The winning city in the fourth layer, $\mathbf{c}_{\omega_{\text{opt}}}$, is displaced from its current location to the new location specified by the winning neuron output. In other words, if $\mathbf{c}_{\omega_{\text{opt}}}$ is placed in location a on the current tour:
   $$\mathbf{x}_a^1 = \mathbf{c}_{\omega_{\text{opt}}} \tag{19}$$
   then, the new tour is configured as:
   $$\boldsymbol{\psi}^{k+1} = [\mathbf{x}_1^1, \mathbf{x}_2^1, \ldots, \mathbf{x}_{x_{\omega_{opt}}^4}^1, \mathbf{x}_a^1, \mathbf{x}_{x_{\omega_{opt}}^4+1}^1, \ldots, \mathbf{x}_{a-1}^1, \mathbf{x}_{a+1}^1, \ldots, \mathbf{x}_m^1] \tag{20}$$
6. Set $k = k+1$ and repeat steps 2-5 until convergence.

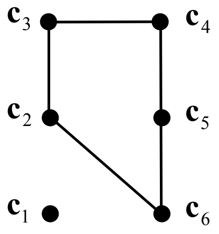

Optimizer Part
Constructive Part
$\mathbf{x}_1^4$
$\mathbf{x}_2^4$
$\mathbf{x}_3^4$
$\mathbf{x}_4^4$
$\mathbf{x}_5^4$
$\mathbf{x}_6^4$
Set Q
Set R
Tour Competitive Layer
$\mathbf{t}^4$
$\mathbf{x}_1^3$
$\mathbf{x}_2^3$
$\mathbf{x}_3^3$
$\mathbf{x}_4^3$
$\mathbf{x}_5^3$
$\mathbf{x}_6^3$
Link Competitive Layer
$\mathbf{t}^3$
$\mathbf{x}_1^2$
$\mathbf{x}_2^2$
$\mathbf{x}_3^2$
Link Layer
$\mathbf{x}_1^1$
$\mathbf{x}_2^1$
$\mathbf{x}_3^1$
Tour Layer


Start
Initial tour creation
CONN initialization
$m+1 \rightarrow m$
Is $m = n$ or $m-\gamma$ dividable by λ?
Yes
No
Transferring the winning city from the set R^k to the set Q^k.
CONN proceeds forward through the optimizer part.
Has CONN converged in the optimizer phase?
No
Yes
Is $m < n$?
No
Yes
Adding the winning city to the tour.
Updating the current tour
Stop
CONN proceeds forward through the constructive part.

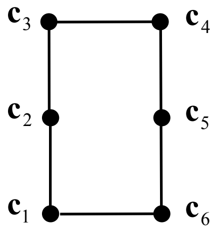
Equation Number	Before Efficient Implementation	After Efficient Implementation
8	$O(mn)$	$O(n)$
9	$O(mn)$	$O(n)$
12	$O(mn)$	$o(mn)$
Other Equations	$O(n)$	$O(n)$
Overall Computational Complexity	$O(n^3)$	$o(n^3)$


$\mathbf{c}_3$
$\mathbf{c}_4$
$\mathbf{c}_2$
$\mathbf{c}_5$
$\mathbf{c}_1$
$\mathbf{c}_6$

(a)


$\mathbf{c}_3$
$\mathbf{c}_4$
$\mathbf{c}_2$
$\mathbf{c}_5$
$\mathbf{c}_1$
$\mathbf{c}_6$

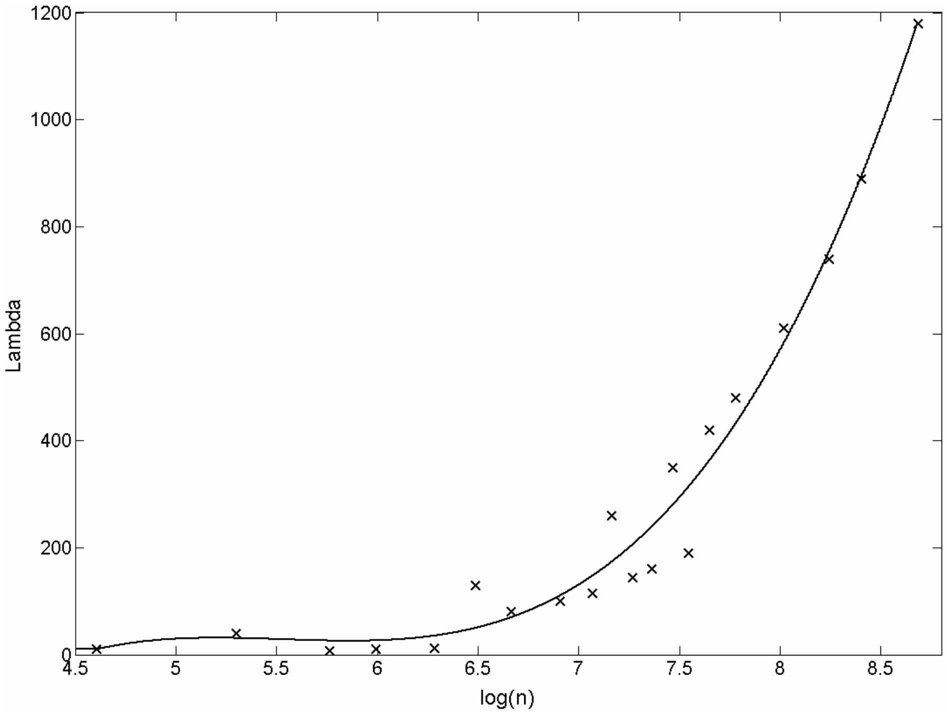
(b)


$\mathbf{c}_3$
$\mathbf{c}_4$
$\mathbf{c}_2$
$\mathbf{c}_5$
$\mathbf{c}_1$
$\mathbf{c}_6$

(c)

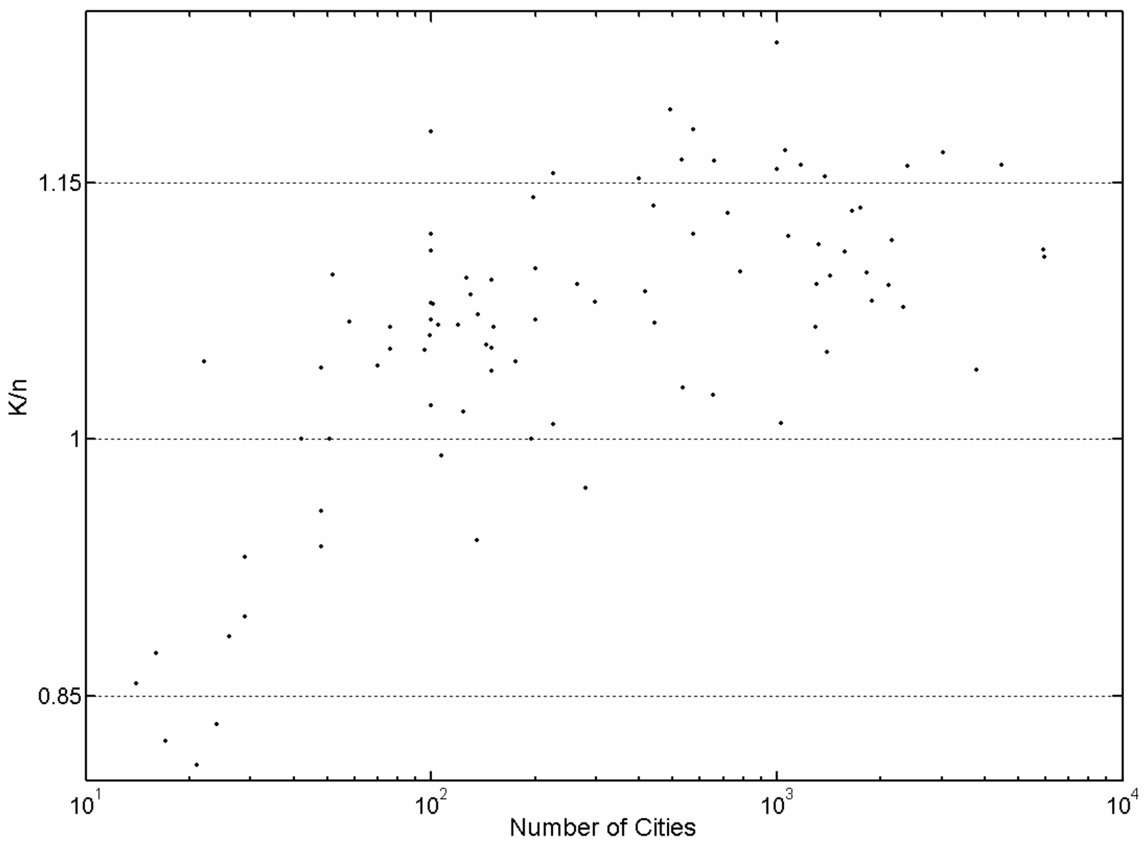
Counterpart Algorithm	**Processing System**	**Prog. Language**	**Scaling Coefficients**	
			AMD 1.4-GHz PC 3GL Prog. Lang.	AMD 1.4-GHz PC MATLAB
2-Opt, 4-Opt, SA_1, ITS [28]	Pentium III 900-MHz PC	PASCAL	0.32	>0.32
Greedy-LK [50]	PentiumPro 200-MHz PC	NA	0.07	>0.07
MMAS [25]	Sun UltraSparc II Works.	C++	0.24	>0.24
NEA [47]	Pentium III 866-MHz PC	C++	0.31	>0.31
KD, KL, KG [43]	Pentium III 700-MHz PC	FORTRAN	0.25	>0.25
eISOM, ESOM, Budinich's SOM [11]	Sun UltraSparc 5/270 Works.	C++	0.48	>0.48
Co-Adaptive Net [48]	Silicon Graphic O2 Works.	FORTRAN	0.42	>0.42

TSP name	Optimal answer	$\lambda \approx \max(1,[n/80])$			$\lambda \approx \max(3,[n/40])$			$\lambda \approx \max(5,[n/20])$			$\lambda \approx \max(10,[n/10])$			$\lambda \approx \max(20,[n/5])$		
		$\delta\%$	γ	T	$\delta\%$	γ	T	$\delta\%$	γ	T	$\delta\%$	γ	T	$\delta\%$	γ	T
berlin52	7627	6.0	1	0.09	6.0	3	0.08	6.0	5	0.08	**6.0**	**10**	**0.08**	7.1	20	0.06
kroA100	21282	2.6	1	0.27	2.6	3	0.22	2.6	5	0.22	**2.6**	**10**	**0.21**	2.6	20	0.20
kroB200	29437	4.3	3	0.53	4.2	5	0.58	4.2	10	0.44	4.2	20	0.41	**4.2**	**40**	**0.35**
lin318	42029	7.1	4	1.2	**7.3**	**8**	**0.89**	7.5	16	0.88	8.0	32	0.75	8.4	64	0.70
rd400	15281	5.4	5	1.7	**4.6**	**10**	**1.3**	6.1	20	1.2	6.3	40	1.1	7.1	80	1.0
ali535	202339	12.3	6	3.1	**11.9**	**13**	**2.4**	12.6	27	2.0	12.1	55	1.8	13.4	110	1.7
d657	48912	7.6	8	4.3	7.6	16	3.3	8.2	32	2.8	7.1	65	2.3	**7.0**	**130**	**2.1**
rat783	8806	8.2	10	5.4	7.9	20	4.2	8.0	40	3.5	**7.2**	**80**	**2.9**	8.2	160	2.8
pr1002	259045	6.7	12	8.9	6.7	25	6.5	6.9	50	5.1	**6.9**	**100**	**4.6**	7.2	200	3.9
pcb1173	56892	9.1	14	12.1	8.4	28	8.9	8.9	57	7.0	**8.5**	**115**	**6.2**	8.9	230	5.6
d1291	50801	11.0	16	12	11.2	32	9.1	11.3	65	7.4	11.3	130	6.3	**11.0**	**260**	**5.6**
u1432	152970	6.4	18	16	6.4	36	12	6.4	72	9.1	**6.4**	**145**	**7.6**	6.9	290	7.1
fl1577	22249	10.5	20	19	10.5	40	13	7.0	80	11	**6.9**	**160**	**9.2**	9.0	320	8.4
vm1748	336556	9.2	22	26	9.4	44	18	8.6	87	14	8.8	175	13	**8.8**	**350**	**12**
rl1889	316536	9.3	23	30	8.8	46	21	9.1	95	16	**9.1**	**190**	**13**	10.4	380	12
d2103	80450	3.1	26	34	3.1	52	25	3.1	105	18	3.1	210	15	**3.1**	**420**	**14**
pr2392	378032	8.4	30	99	7.7	60	63	7.6	120	46	7.4	240	38	**7.5**	**480**	**34**
pcb3038	137694	7.8	35	370	7.7	71	59	7.4	152	41	7.4	305	35	**7.3**	**610**	**32**
fl3795	28772	12	41	132	11.9	82	88	9.9	185	66	9.6	370	55	**9.4**	**740**	**50**
fnl4461	182566	7.9	55	233	7.9	111	144	7.6	222	105	7.6	445	88	**7.6**	**890**	**76**
rl5915	565530	13.0	98	398	12.7	197	254	12.7	295	234	12.6	590	179	**12.8**	**1180**	**154**
Average		8.0			7.8			7.7			**7.6**			8.0		

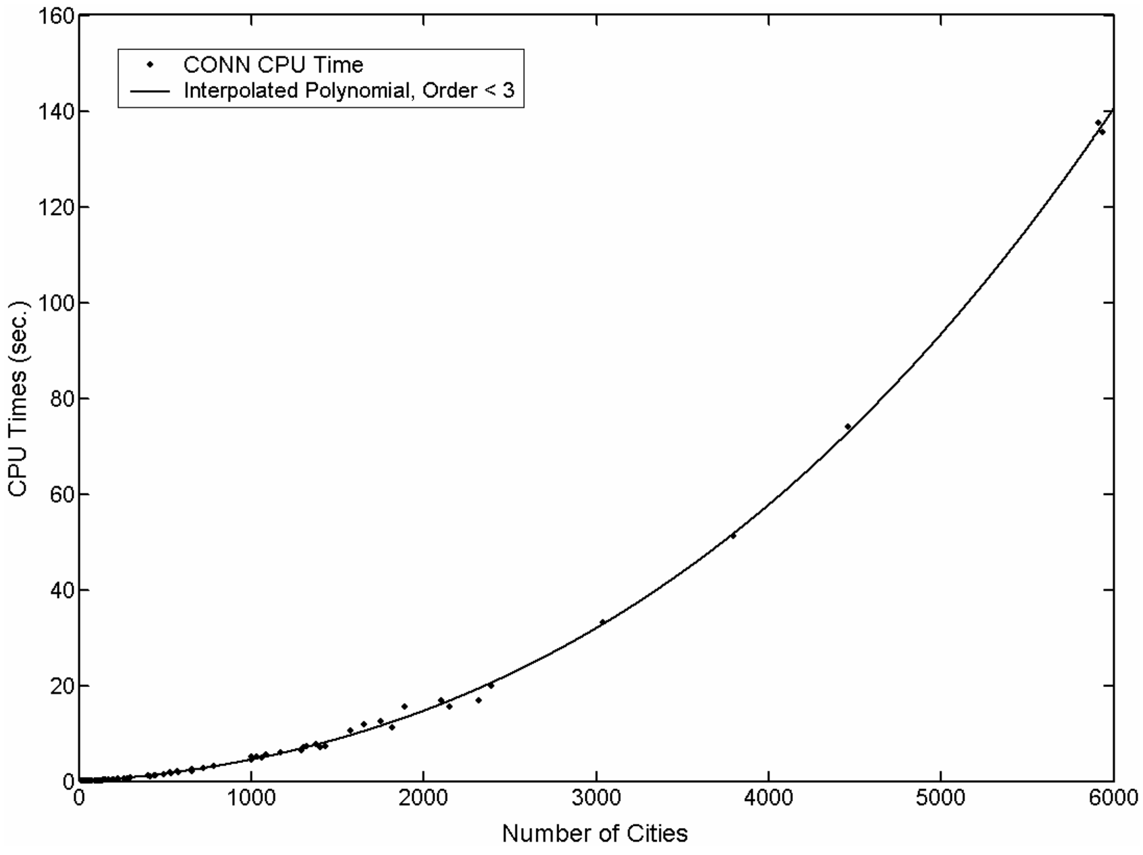

1200
1000
800
600
400
200
0
4.5
5
5.5
6
6.5
7
7.5
8
8.5
Lambda
log(n)

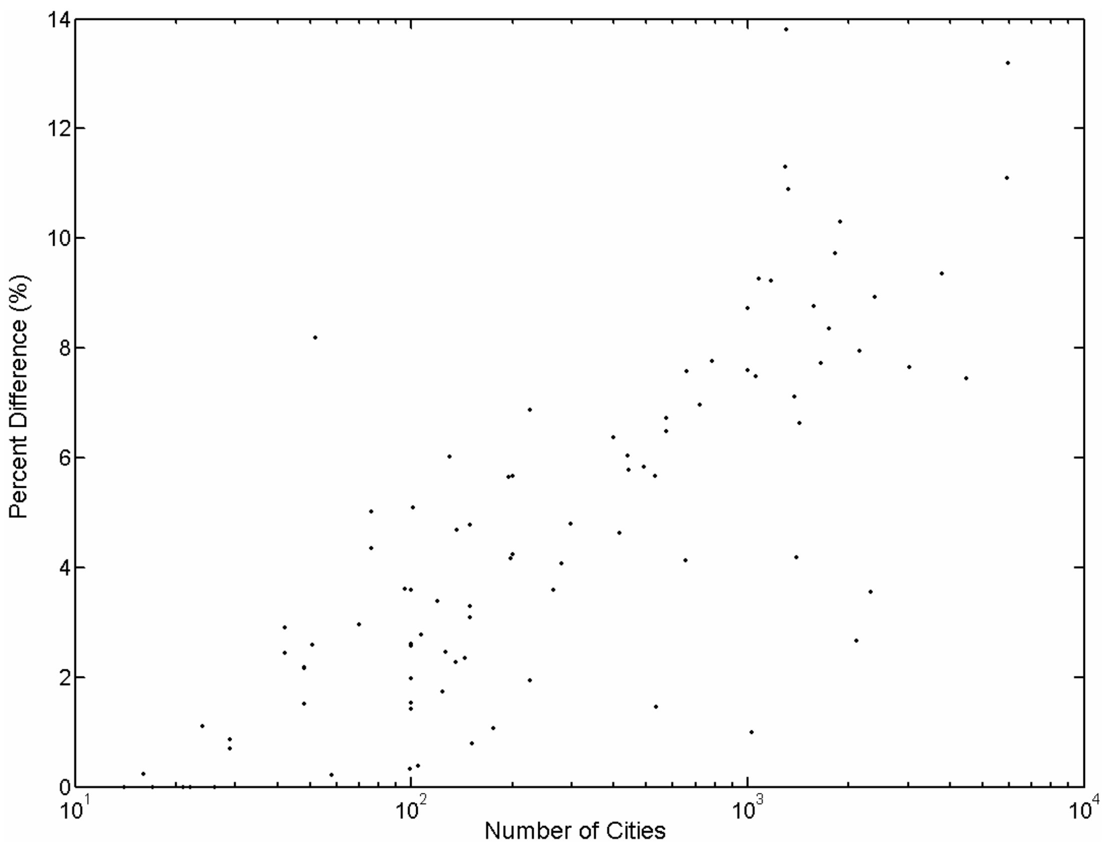
TSP Name	n	Optimal Solution	λ	Cheapest Link					Convex Hull					Hull Through Four Outermost Cities			
				δ (%)	γ	β	K	T (sec.)	δ (%)	γ	β	K	T (sec.)	δ (%)	β	K	T (sec.)
burma14	14	3323	10	**0.0**	4	2	12	0.01	**0.0**	5	2	11	0.01	**0.0**	2	12	0.01
ulysses16	16	6859	10	**0.23**	6	4	14	0.02	**0.23**	7	3	12	0.01	**0.23**	6	18	0.02
gr17	17	2085	10	**0.0**	7	4	14	0.02	—	—	—	—	—	3.07	5	18	0.03
gr21	21	2707	10	**0.0**	8	4	17	0.02	—	—	—	—	—	2.77	4	21	0.02
ulysses22	22	7013	10	**0.0**	8	9	23	0.03	**0.0**	7	5	20	0.03	**0.0**	8	26	0.03
gr24	24	1272	10	**1.10**	8	4	20	0.03	—	—	—	—	—	**1.10**	4	24	0.04
fri26	26	937	10	**0.0**	8	5	23	0.03	—	—	—	—	—	2.35	6	28	0.04
bayg29	29	1610	10	**0.87**	8	6	27	0.03	**0.87**	7	6	28	0.03	**0.87**	6	31	0.04
bays29	29	2020	10	**0.69**	8	5	26	0.03	**0.69**	7	6	28	0.03	**0.69**	7	32	0.04
dantzig42	42	699	10	**2.43**	9	9	42	0.05	**2.43**	8	8	42	0.05	6.58	10	48	0.06
swiss42	42	1273	10	**2.91**	9	9	42	0.06	—	—	—	—	—	7.46	12	50	0.06
att48	48	10628	10	**2.17**	9	6	45	0.06	**2.17**	11	6	43	0.05	2.58	6	50	0.06
gr48	48	5046	10	**1.51**	9	7	46	0.06	—	—	—	—	—	**1.51**	7	46	0.06
hk48	48	11461	10	**2.16**	9	11	50	0.07	—	—	—	—	—	**2.16**	11	55	0.07
eil51	51	426	10	**2.58**	9	9	51	0.06	**2.58**	9	8	50	0.06	**2.58**	9	56	0.07
berlin52	52	7542	10	8.18	9	14	57	0.07	**6.03**	8	10	54	0.07	8.76	12	60	0.07
brazil58	58	25395	10	**0.22**	9	13	62	0.09	—	—	—	—	—	5.13	18	72	0.09
st70	70	675	10	**2.96**	10	13	73	0.09	3.70	10	10	70	0.09	3.41	10	76	0.10
eil76	76	538	10	**5.02**	10	15	81	0.10	**5.02**	10	15	81	0.11	**5.02**	19	91	0.12
pr76	76	108159	10	**4.34**	10	14	80	0.11	4.90	7	11	80	0.10	**4.34**	12	84	0.11
gr96	96	55209	10	**3.61**	10	15	101	0.13	**3.61**	11	16	101	0.14	4.77	18	110	0.16
rat99	99	1211	10	**0.33**	11	17	105	0.14	0.50	14	18	103	0.14	0.50	19	114	0.16
kroA100	100	21282	10	**2.57**	11	29	118	0.16	**2.57**	12	21	109	0.16	**2.57**	28	124	0.18
kroB100	100	22141	10	**2.60**	11	19	108	0.15	**2.60**	13	17	104	0.14	**2.60**	21	117	0.16
kroC100	100	20749	10	**1.53**	11	22	111	0.15	**1.53**	11	17	106	0.14	**1.53**	20	116	0.16
kroD100	100	21294	10	**1.42**	11	13	102	0.14	**1.42**	14	13	99	0.13	1.73	18	114	0.16
kroE100	100	22068	10	**1.97**	11	18	107	0.15	2.14	14	17	103	0.14	4.37	18	114	0.15
rd100	100	7910	10	**3.59**	11	23	112	0.16	**3.59**	11	22	111	0.15	4.46	26	122	0.17
eil101	101	629	11	5.09	11	19	109	0.15	5.88	10	15	106	0.14	**4.61**	21	118	0.17
lin105	105	14379	14	**0.38**	11	18	112	0.15	**0.38**	13	17	109	0.15	**0.38**	20	121	0.17
pr107	107	44303	15	**2.77**	11	10	106	0.14	**2.77**	24	9	92	0.12	**2.77**	10	113	0.15
gr120	120	6942	22	**3.39**	11	19	128	0.19	5.68	12	16	124	0.18	4.84	19	135	0.20
pr124	124	59030	24	**1.74**	11	13	126	0.19	**1.74**	27	14	111	0.16	**1.74**	13	133	0.19
bier127	127	118282	25	**2.45**	11	23	139	0.20	**2.45**	10	23	140	0.21	5.07	21	144	0.21
Average of the Above Solutions				**2.08 (32)**					2.52 (20)					3.02 (18)			
ch130	130	6110	26	4.88	12	24	142	0.24	6.01	10	21	141	0.21	**4.66**	21	147	0.22
pr136	136	96772	27	2.79	12	15	139	0.21	**2.27**	26	18	128	0.19	2.92	15	147	0.22
gr137	137	69853	28	5.67	12	24	149	0.22	**4.69**	8	18	147	0.21	8.07	28	161	0.24
pr144	144	58537	29	**2.34**	12	15	147	0.22	**2.34**	10	18	152	0.22	4.12	16	156	0.24
ch150	150	6528	30	5.50	12	23	161	0.25	**3.29**	15	21	156	0.24	5.35	22	168	0.26
kroA150	150	26524	30	5.17	12	21	159	0.25	**4.78**	15	23	158	0.24	5.76	27	173	0.26
kroB150	150	26130	30	**3.09**	12	25	163	0.25	**3.09**	11	25	164	0.26	3.21	21	167	0.26
pr152	152	73682	30	**0.79**	12	22	162	0.25	**0.79**	13	23	162	0.25	**0.79**	23	171	0.26
si175	175	21407	31	**1.07**	13	21	183	0.30	—	—	—	—	—	1.48	29	200	0.32
rat195	195	2323	31	**5.64**	14	17	198	0.32	**5.64**	17	17	195	0.32	**5.64**	21	212	0.34
d198	198	15780	31	10.1	14	49	233	0.39	**4.16**	8	36	226	0.36	8.23	30	224	0.37
kroA200	200	29368	31	5.16	14	36	222	0.37	5.66	11	31	220	0.37	**4.40**	33	229	0.38
kroB200	200	29437	31	4.50	14	33	219	0.37	**4.24**	17	31	214	0.36	**4.24**	34	230	0.38
tsp225	225	3916	30	**5.54**	15	49	259	0.45	6.87	8	43	260	0.45	6.36	47	268	0.46
pr226	226	80369	30	2.35	15	20	231	0.39	**1.93**	19	21	228	0.39	2.43	21	243	0.41

TSP Name	n	Optimal Solution	λ	Cheapest Link					Convex Hull					Hull Through Four Outermost Cities			
				δ (%)	γ	β	K	T (sec.)	δ (%)	γ	β	K	T (sec.)	δ (%)	β	K	T (sec.)
pr264	264	49135	28	**3.58**	16	47	295	0.57	**3.58**	16	40	288	0.54	4.02	40	400	0.56
a280	279	2579	27	4.81	16	48	312	0.60	4.07	38	29	271	0.50	**3.57**	30	306	0.60
pr299	299	48191	27	4.85	17	48	330	0.65	**4.80**	22	46	323	0.68	5.44	48	343	0.71
rd400	400	15281	26	**5.77**	20	85	465	1.15	6.37	14	75	461	1.12	6.22	81	477	1.19
fl417	417	11861	27	4.54	21	69	465	1.15	4.62	32	68	453	1.06	**4.42**	66	479	1.15
pr439	439	107217	28	6.24	21	71	489	1.20	**6.03**	13	73	499	1.20	7.30	70	505	1.23
pcb442	442	50778	28	5.72	21	82	503	1.28	5.77	37	67	472	1.22	**5.56**	57	495	1.27
d493	493	35002	31	6.27	23	113	583	1.59	**5.83**	9	104	588	1.50	5.90	107	596	1.55
att532	532	27686	35	5.84	24	101	609	1.81	**5.66**	12	99	619	1.86	5.95	107	635	1.96
si535	535	48450	35	**1.45**	24	40	551	1.60	—	—	—	—	—	1.65	48	579	1.68
u574	574	36905	39	6.11	26	115	663	1.93	6.48	11	115	678	2.01	**5.90**	116	686	2.04
rat575	575	6773	39	7.84	26	83	632	1.90	**6.72**	19	88	644	1.92	6.90	84	655	1.86
p654	654	34643	49	4.41	28	88	714	2.21	**4.13**	55	72	671	2.10	4.34	89	739	2.43
d657	657	48912	50	7.74	28	127	756	2.40	**7.58**	10	117	764	2.45	7.94	130	783	2.53
u724	724	41910	60	7.61	30	131	825	3.65	**6.97**	25	121	820	2.78	7.89	123	843	2.94
rat783	783	8806	70	**7.59**	32	133	884	3.20	7.76	20	97	860	3.18	8.12	118	897	3.42
Average of the Above Solutions				5.00 (10)					**4.90 (20)**					5.12 (9)			
dsj1000	1000	18660188	111	8.73	39	218	1179	4.83	8.84	16	216	1200	4.90	**8.72**	236	1232	5.16
pr1002	1002	259045	111	**6.94**	39	174	1137	4.37	6.97	23	167	1146	4.38	7.59	162	1160	4.38
si1032	1032	92650	117	**0.91**	40	14	1006	4.79	—	—	—	—	—	0.99	14	1042	5.09
u1060	1060	224094	123	7.79	41	185	1204	5.04	7.72	24	171	1207	4.92	**7.49**	183	1239	4.99
vm1084	1084	239297	128	9.41	42	153	1195	5.46	**9.08**	13	140	1211	5.33	9.26	133	1213	5.63
pcb1173	1173	56892	147	**8.90**	45	205	1333	5.92	9.10	15	214	1372	5.91	9.23	192	1361	5.93
d1291	1291	50801	174	**10.8**	48	88	1331	6.60	11.1	13	94	1372	6.47	11.3	89	1376	6.45
rl1304	1304	252948	177	**12.6**	49	138	1393	6.83	12.9	19	143	1428	6.76	13.8	122	1422	7.12
rl1323	1323	270199	181	**9.6**	49	160	1434	7.10	10.4	14	166	1475	7.20	10.9	155	1474	7.20
nrw1379	1379	56638	194	6.72	51	224	1552	7.42	**6.06**	19	226	1586	8.02	7.12	216	1591	7.66
fl1400	1400	20127	198	4.38	52	98	1446	6.8	**3.89**	32	78	1446	6.70	4.19	75	1471	7.11
u1432	1432	152970	206	**6.47**	53	157	1536	7.12	6.56	11	128	1549	7.01	6.62	141	1569	7.26
fl1577	1577	22249	240	9.65	57	142	1662	9.87	9.16	85	201	1693	8.89	**8.76**	177	1750	10.6
d1655	1655	62128	258	8.28	60	180	1775	11.0	8.29	40	215	1830	11.4	**7.72**	225	1876	11.8
vm1748	1748	336556	280	9.23	63	251	1936	14.0	8.55	26	243	1965	12.4	**8.36**	241	1985	12.4
u1817	1817	57201	297	9.73	65	180	1932	10.8	**9.04**	31	181	1967	11.3	9.73	181	1994	11.2
rl1889	1889	316536	314	10.4	67	161	1983	12.4	**9.59**	20	144	2013	12.4	10.3	157	2042	15.5
d2103	2103	80450	365	3.10	74	196	2225	17.8	3.03	12	141	2232	16.3	**2.67**	194	2293	16.8
u2152	2152	64253	376	9.05	76	220	2296	14.9	8.50	94	270	2328	16.4	**7.95**	255	2403	15.5
u2319	2319	234256	426	**3.08**	81	182	2420	16.7	3.14	50	153	2422	16.2	3.55	183	2498	16.8
pr2392	2392	378032	433	9.37	83	408	2717	20.2	9.11	48	368	2712	19.4	**8.92**	386	2774	20.0
pcb3038	3038	137694	584	8.04	104	495	3429	34.4	**7.30**	10	506	3534	32.8	7.65	514	3548	33.2
fl3795	3795	28772	752	9.66	128	188	3855	48.2	**9.31**	32	196	3959	53.4	9.36	157	3948	51.1
fnl4461	4461	182566	894	7.94	149	696	5008	73.3	7.64	21	680	5120	73.5	**7.44**	721	5178	74.1
rl5915	5915	565530	1183	13.1	195	625	6345	143.1	12.8	14	588	6489	139.4	**11.1**	659	6570	137.5
rl5934	5934	556045	1187	13.1	196	690	6428	134.7	**12.2**	15	745	6664	139.9	13.2	636	6566	135.5
Average of the Above Solutions				8.35 (8)					8.41 (8)					**8.22 (10)**			
Total Average				**4.87 (50)**					5.22 (47)					5.22 (37)			
Total Average for CONN				4.72													



K/n
1.15
1
0.85
10^1
10^2
10^3
10^4
Number of Cities

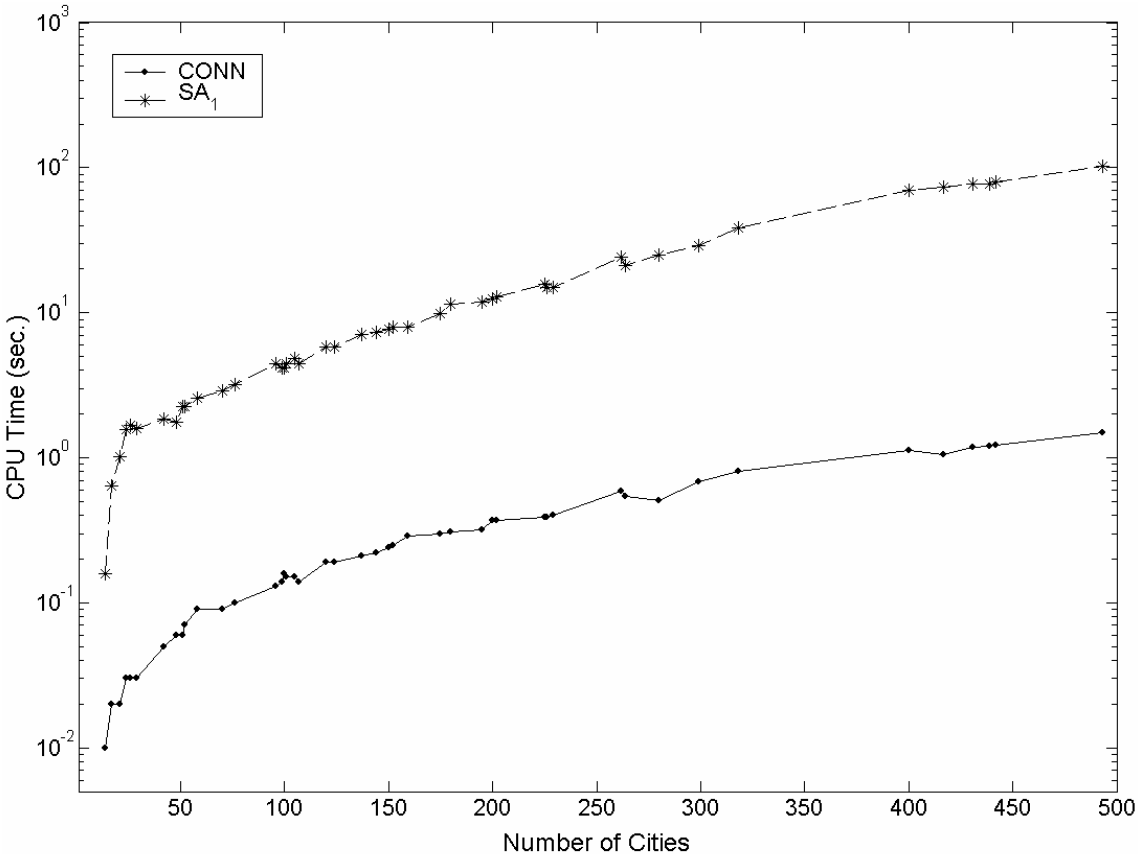

CONN CPU Time
Interpolated Polynomial, Order < 3
CPU Times (sec.)
Number of Cities
0
20
40
60
80
100
120
140
160
0
1000
2000
3000
4000
5000
6000



Percent Difference (%)
14
12
10
8
6
4
2
0
10^1
10^2
10^3
10^4
Number of Cities

		CONN		2-Opt		4-Opt		SA_1		ITS	
TSP name	Optimal solution	$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time
rat99	1211	0.33	**0.1**	4.5	0.2	3.0	15	**0.0**	4.2	**0.0**	0.3
pr152	73682	0.8	**0.3**	0.8	0.9	0.8	122	0.2	8.0	**0.0**	5.1
si175	21407	1.1	**0.3**	0.5	1.2	0.2	288	**0.0**	9.9	**0.0**	3.5
rat195	2323	5.6	**0.3**	7.5	1.6	3.4	544	0.2	12	**0.0**	48
d198	15780	4.2	**0.4**	2.2	1.9	1.3	576	0.1	13	**0.0**	19
pr264	49135	3.6	**0.5**	4.9	5.1	—	—	0.1	21	**0.0**	6.7
a280	2579	4.1	**0.5**	6.7	6.1	—	—	**0.0**	25	**0.0**	8.0
rd400	15281	6.4	**1.1**	6.6	30	—	—	0.5	70	**0.2**	179
pcb442	50778	5.8	**1.2**	7.1	38	—	—	0.6	80	**0.4**	163
d493	35002	6.0	**1.2**	5.6	54	—	—	0.6	102	**0.3**	288
Average $\bar{\delta}(\bar{\delta}_{\text{CONN}})$		**3.8** (3.8)		4.6 (**3.8**)		**1.7** (2.4)		**0.2** (3.8)		**0.1** (3.8)	
Comp. Complexity		$o(n^3)$		$\Omega(n^2)$		$\Omega(n^3)$		$\Omega(n^3)$		$\Omega(n^3)$	

† Solution qualities of 2-Opt, 4-Opt, SA_1, and ITS were quoted from Ref. [28] while their CPU time was scaled by 0.32 (see Table 3).


CONN
SA_1
CPU Time (sec.)
Number of Cities
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
50
100
150
200
250
300
350
400
450
500

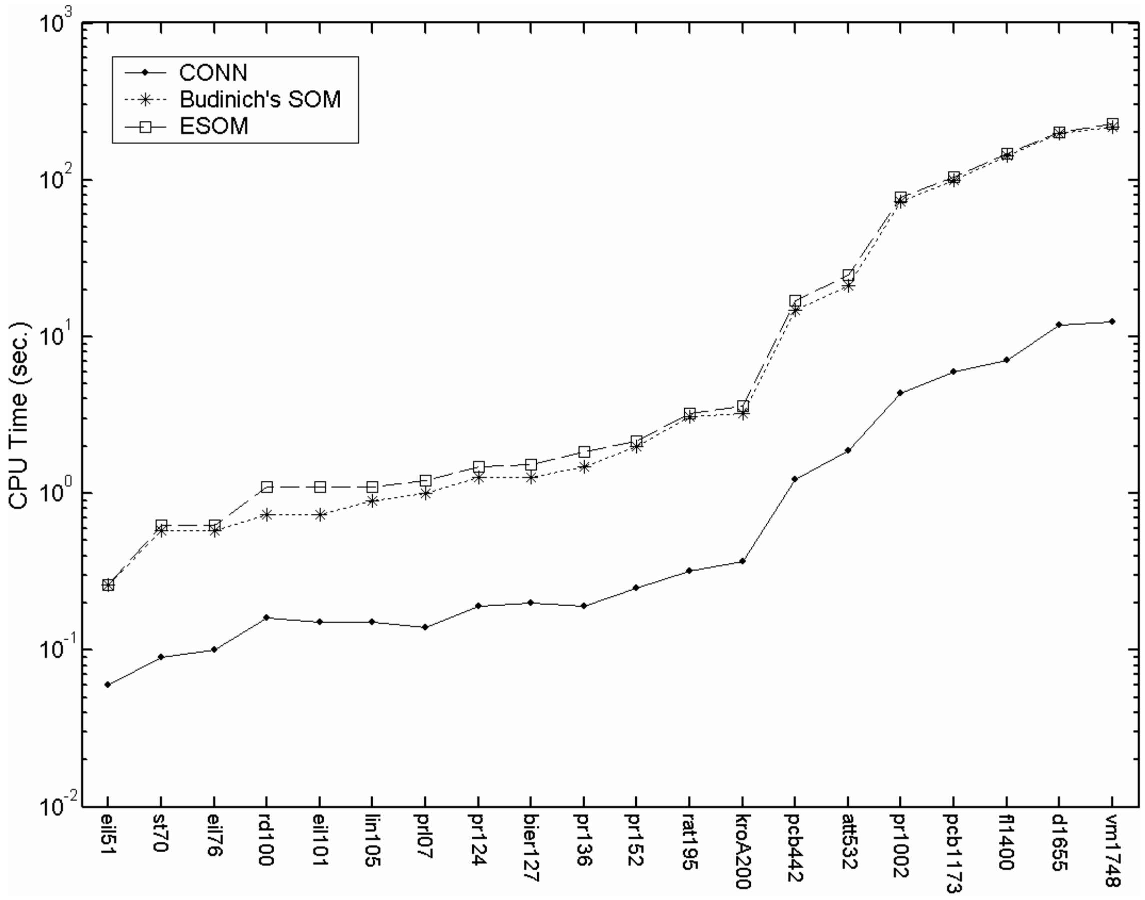
TSP name	Optimal solution	CONN		Greedy-LK		MMAS		NEA	
		$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time	$\bar{\delta}$	CPU Time
att532	27686	5.7	**1.9**	**0.0**	6.4	0.1	93	**0.0**	52
gr666	294358	10.8	**3.1**	**0.0**	49	—	—	—	—
rat783	8806	7.8	**3.2**	**0.0**	32	—	—	—	—
dsj1000	18660188	8.7	**5.2**	—	—	—	—	**0.0**	345
pr1002	259045	7.6	**4.4**	**0.0**	78	—	—	—	—
u1060	224094	7.5	**5.0**	—	—	**0.3**	619	—	—
pcb1173	56892	9.2	**5.9**	—	—	**0.1**	773	—	—
d1291	50801	11.3	**6.5**	—	—	**0.0**	455	—	—
fl1577	22249	8.8	**11**	—	—	**0.3**	720	—	—
d2103	80450	2.7	**17**	—	—	—	—	**0.0**	141
u2152	64253	7.9	**16**	**0.2**	350	—	—	—	—
pcb3038	137694	7.6	**33**	—	—	—	—	**0.0**	538
fl3795	38772	9.4	**51**	**0.1**	944	—	—	—	—
fnl4461	4461	7.4	**74**	—	—	—	—	**0.0**	2738
Average $\bar{\delta}(\bar{\delta}_{\text{CONN}})$		**8.0**		**0.1** (8.2)		**0.2** (8.5)		**0.0** (6.3)	
Comp. Complexity		$o(n^3)$		$\omega(n^3)$		$\omega(n^3)$		$\omega(n^3)$	

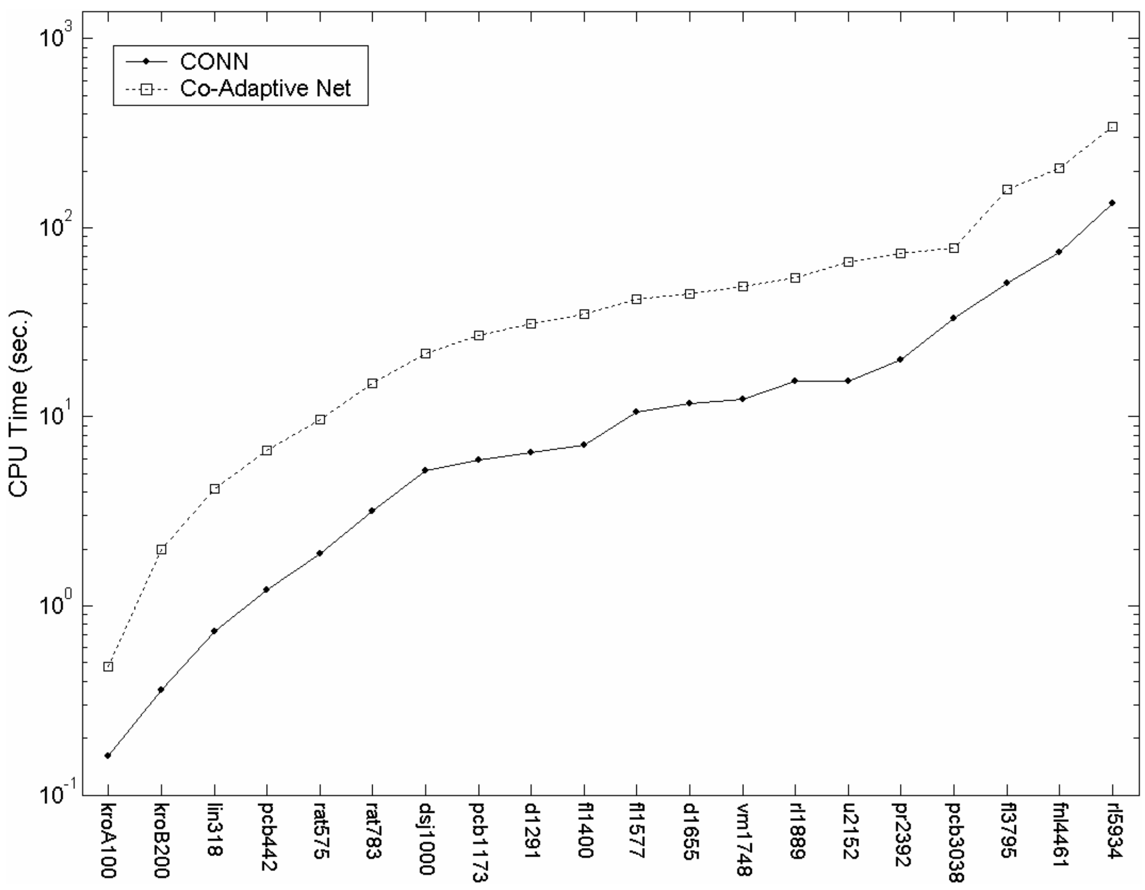
† Solution qualities of Greedy-LK, MMAS, and NEA were quoted from Refs. [50], [25], and [47], respectively, while their CPU time was scaled by 0.07, 0.24, and 0.31, correspondingly (see Table 3).

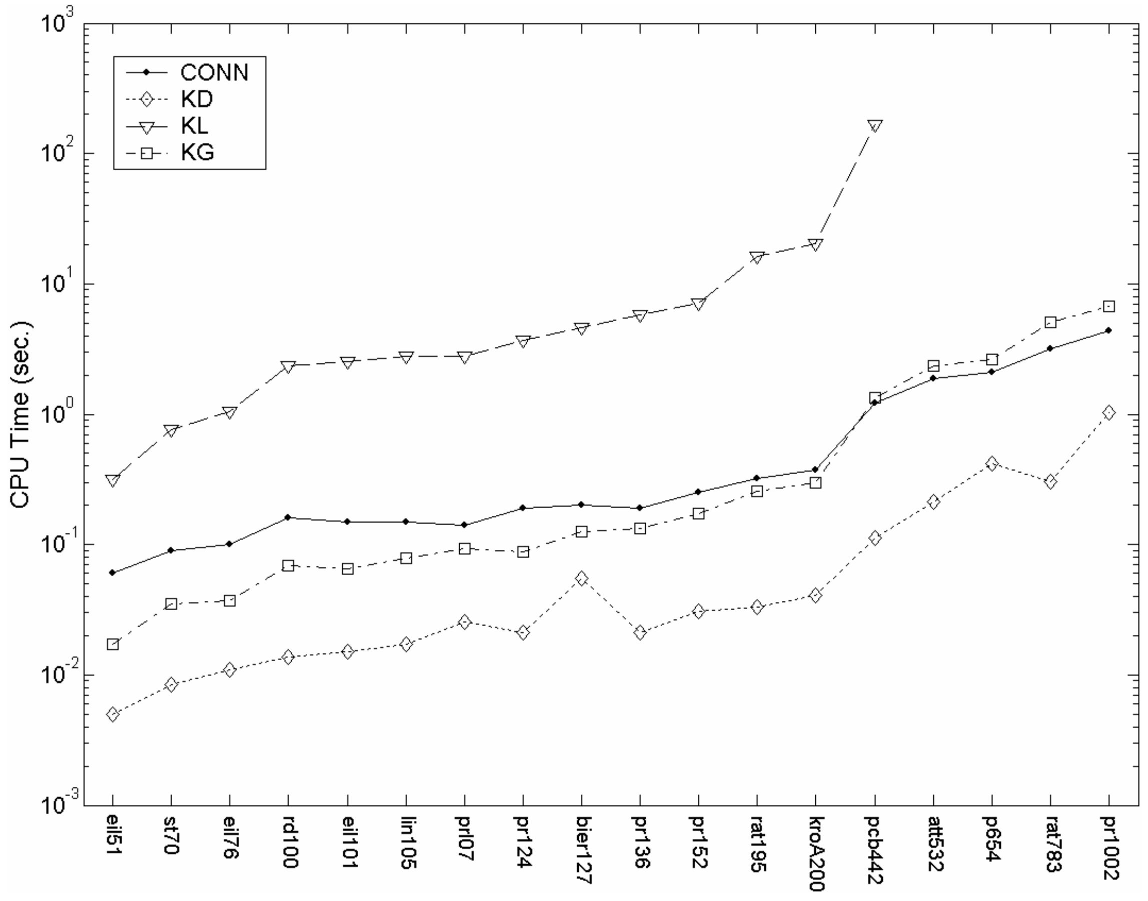
Algorithm	Computational Complexity
CONN	$o(n^3)$
Nearest Neighbor	$O(n^2)$
Greedy	$O(n^2 \log n)$
Clarke-Wright	$O(n^2 \log n)$
Christofides	$\Omega(n^{2.5})$

TSP name	Optimal solution	CONN	SA$_2$[a, b]	KD[c]	KL[c]	KG[c]	Budinich's SOM[a, b]	ESOM[a, b]	eISOM[b]	Co-Adaptive Net[d]	MVHN[e] two-optimal
eil51	426	2.6	2.3	3.5	2.9	2.9	3.1	**2.1**	2.6	2.9	7.0
st70	675	3.0	2.3	3.7	**1.5**	2.3	1.7	2.1	—	1.7	7.8
eil76	538	5.0	5.5	6.5	5.0	5.5	5.3	**3.9**	—	4.4	9.4
gr96	55209	3.6	4.1	—	—	—	2.1	1.0	**0.8**	—	—
rd100	7910	3.6	3.3	4.9	2.1	2.6	3.2	**2.0**	—	3.6	10.0
kroA100	21282	2.6	5.9	—	—	—	3.7	1.0	**0.6**	1.3	13.9
eil101	629	5.1	5.7	6.8	4.7	5.6	5.2	**3.4**	3.6	3.8	10.1
lin105	14379	0.38	1.9	2.2	2.0	1.3	1.7	**0.25**	—	1.1	8.3
pr107	44303	2.8	1.5	10.8	0.73	**0.42**	1.3	1.5	—	4.4	6.1
pr124	59030	1.7	1.3	3.2	**0.08**	0.49	1.6	0.67	—	2.9	6.1
bier127	118282	2.5	3.5	5.8	2.8	3.1	3.6	**1.7**	—	3.0	9.1
pr136	96772	2.3	4.9	**1.9**	4.5	5.2	5.2	4.3	—	4.7	9.9
gr137	69853	4.8	8.5	—	—	—	8.6	4.3	**3.2**	—	—
kroA150	26524	3.5	4.3	—	—	—	4.4	2.0	**1.8**	3.1	—
pr152	73682	**0.79**	2.6	3.2	0.97	1.3	2.0	0.89	—	2.1	5.6
rat195	2323	**5.6**	13.3	8.4	12.2	11.9	11.5	7.13	—	7.5	12.2
kroA200	29368	5.7	5.6	5.7	5.7	6.6	6.1	2.9	**1.6**	3.3	—
lin318	42029	7.3	7.6	—	—	—	8.2	4.1	**2.1**	4.3	—
pcb442	50778	**5.8**	9.2	8.0	11.1	10.4	8.4	7.4	6.1	7.6	12.8
att532	87550	5.7	5.4	6.2	6.7	6.8	5.7	5.0	**3.4**	5.3	—
p654	34643	**4.1**	—	5.0	—	5.6	—	—	—	4.6	—
rat783	8806	7.8	—	9.1	—	9.6	—	—	—	**6.6**	—
pr1002	259045	7.6	6.0	7.1	—	7.6	8.8	6.4	**4.8**	5.3	—
pcb1173	56892	**9.2**	11.1	12.7	—	—	11.4	9.9	—	9.6	—
d1291	50801	**11.3**	—	16.4	—	—	—	—	—	12.0	—
rl1323	270191	**10.9**	—	13.8	—	—	—	—	—	11.8	—
fl1400	20127	**4.1**	4.7	6.0	—	—	5.6	**4.1**	—	4.3	—
u1432	152970	6.6	—	9.0	—	—	—	—	—	**6.3**	—
fl1577	22249	**8.8**	—	14.3	—	—	—	—	—	14.4	—
d1655	62128	**7.7**	13.2	12.3	—	—	15.2	11.4	—	10.1	—
vm1748	336556	8.4	7.9	8.6	—	—	10.2	7.3	—	**6.8**	—
pr2392	378032	8.9	8.2	11.5	—	—	10.3	8.5	**6.4**	7.9	—
pcb3038	137694	**7.6**	—	12.6	—	—	—	—	—	8.9	—
fl3795	28772	**9.4**	—	15.6	—	—	—	—	—	14.9	—
fnl4461	182566	7.4	—	16.3	—	—	—	—	—	**6.6**	—
rl5915	565530	**11.1**	—	19.5	—	—	—	—	—	14.8	—
rl5934	556045	**13.2**	—	18.9	—	—	—	—	—	14.0	—
Average $\bar{\delta}(\bar{\delta}_{\text{CONN}})$		**5.9**	5.8 (**4.6**)	9.0 (**6.1**)	4.2 (**3.5**)	5.0 (**4.0**)	5.9 (**4.6**)	**4.0** (4.6)	**3.1** (5.3)	6.5 (**6.0**)	9.2 (**3.1**)
Computational Complexity		$o(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$\omega(n^3)$

† Solution qualities of the algorithms indicated by superscripts a, b, c, d and e are quoted from Refs. [14], [11], [43], [48] and [52], respectively.


CONN
Budinich's SOM
ESOM
CPU Time (sec.)
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
eil51
st70
eil76
rd100
eil101
lin105
prl07
pr124
bier127
pr136
pr152
rat195
kroA200
pcb442
att532
pr1002
pcb1173
fl1400
d1655
vm1748


CONN
Co-Adaptive Net
CPU Time (sec.)
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
kroA100
kroB200
lin318
pcb442
rat575
rat783
dsj1000
pcb1173
d1291
fl1400
fl1577
d1655
vm1748
rl1889
u2152
pr2392
pcb3038
fl3795
fnl4461
rl5934


CONN
KD
KL
KG
CPU Time (sec.)
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
eil51
st70
eil76
rd100
eil101
lin105
prl07
pr124
bier127
pr136
pr152
rat195
kroA200
pcb442
att532
p654
rat783
pr1002