Parallelized neural network system for solving Euclidean traveling salesman problem

Bihter Avşar[a], Danial Esmaeili Aliabadi[a,*]

[a]*Sabanci University, Faculty of Engineering and Natural Science, Istanbul, Turkey.*

Abstract

We investigate a parallelized divide-and-conquer approach based on a self-organizing map (SOM) in order to solve the Euclidean Traveling Salesman Problem (TSP). Our approach consists of dividing cities into municipalities, evolving the most appropriate solution from each municipality so as to find the best overall solution and, finally, joining neighborhood municipalities by using a blend operator to identify the final solution. We evaluate performance of parallelized approach over standard TSP test problems (TSPLIB) to show that our approach gives a better answer in terms of quality and time rather than the sequential evolutionary SOM.

Keywords: Euclidean Traveling Salesman Problem, Artificial Neural Network, Parallelization, Self-organized map, TSPLIB

1. Introduction

1.1. Problem description

The Traveling Salesman Problem (TSP) is one of the oldest and well-studied problems in operational research: it has been subject of study for more than three decades.

The problem at hand is to find the shortest tour between N cities which covers all cities each exactly once. For a $N-$city problem, there exists $(N-1)!/2$ roundtrips. Therefore, the problem is NP-complete and by increasing the number of cities, the computation time of optimal solution increases drastically [1]. Consequently, obtaining a near-optimal solution in rational time has enormous value. This is why heuristic and metaheuristic have been developed and disclosed very good empirical results over TSP. It is also worth mentioning that they have no mathematical proof for effectiveness.

We can categorize major metaheuristics for solving TSP as evolutionary algorithm (EA) [2, 3], tabu search [4, 5], simulated annealing [6, 7], particle swarm optimization [6], ant colony optimization [6], and neural network as well. Also, by combining these categories, hybrid systems were taken into account [8, 9].

1.2. Application

TSP is naturally applied in transportation and logistic problems but because of its simplicity and comprehensibility, TSP can model many other interesting problems. More specifically, in the biology area that is the host of huge problems, with the advent of the genome projects, the research has focused to shift utilizing the well studied computer problem, Traveling Salesman Problem, approach to study group of genes or proteins. Because of the effectiveness of the TSP, it is used in different applications in genomics and proteomics areas. In one study, Johnson and Liu [10] utilized TSP to predict proteins functions. They found a promising tool to predict functions of uncharacterized proteins. Their prediction method was more advantageous than the traditional methods.

Another study was related to chemotaxis process of neutrophils and macrophages which are the main responsible elements in the defense system in all mammalian bodies. They use chemotaxis to locate their preys and implementing of TSP performed successfully even in the absence of information about target location [11]. Korostensky and Gonnet [12] used TSP solution for evolutionary tree construction that shows relationship between members and they had better results than the other methods.

TSP methods are also used for the DNA sequencing processes [13]. Sequencing By Hybridization (SBH) is proposed as a promising approach however it has an intrinsic problem (i.e. two types of errors associated with nucleotide hybridization) so it has been less widely applicable for unknown sequences. TSP algorithm has provided better and more accurate results [14].

Some other examples are printing circuit-boards [15, 16], clustering a data array [17], encoding DNA [18, 19], image processing [20], and so forth. Nowadays, diversified applications require large-scale TSPs to be solved with acceptable precision.

1.3. Related work

For approximately three decades, neural networks have absorbed much attention. Mostly, two types of neural networks are applied for solving TSP. Hopfield neural network [21, 22] performs weakly in solving big problems and the

*Corresponding Author
Email addresses: `Bihteravsar@sabanciuniv.edu` (Bihter Avşar), `Danialesm@sabanciuniv.edu` (Danial Esmaeili Aliabadi)

self-organizing map (SOM) [23] which exhibits better performance in the large-scale problems.

Many researchers have focused on altering learning rule of neural networks for better results. Aras et al. [24] have tried to improve performance by exploiting the topology of the problem. They called their network Kohonen Network Incorporating Explicit Statistics (KNIES) and claimed that by keeping the mean of the synaptic weights of the network the same as the mean of cities, better results can be achieved. Cochrane and Beasley [25] demonstrate that considering cooperation between neurons in addition to competition can improve quality of solution. They called their network as the Co-adaptive neural network. The obtained results highlighted that none of other self-organized networks can individually compete with Co-adaptive network. Cheung and Law [26] in 2007 introduced a new network which prohibits neurons to be always-winner. Zhang et al. [27] in 2012 presented a new SOM algorithm that categorizes competition between the neurons into overall and regional groups. In their proposed algorithm, overall competition is designed to make the winning neuron and its neighborhood neurons less competitive for outlining the tour, and regional competition is designed to make them more competitive for refining the tour.

Although much research was put into refining the network structure and rules, other research has focused on parallelizing conventional neural networks to deal with bigger problems.Obermayer et al. [28] had applied SOM over a large-scale biological problem (18K data, 900 dimensions). To cope with the size issue, they had to use parallelized computers to solve the problem. Mulder and Wunsch [29] divided huge TSP by clustering algorithms and solved each cluster with adaptive resonance neural network. They were claiming that proposed divide and conquer paradigm can increase scalability and parallelism. Créput and Koukam [30] have tried to improve the neural network by focusing on a heuristic that follows a metaphor within biologic systems that exhibit a high degree of intrinsic parallelism. Such a system has two advantages; firstly, it is intuitive; secondly easy to implement. They have indicated that by improving neural network via an evolutionary manner, they can get better results than the Co-adaptive network [25], Expanding SOM (ESOM) [31], and evolved ISOM(eISOM) [32].

1.4. Current work

Our focus in this paper is to adopt the evolving mechanism of memetic SOM in Créput and Koukam [30] but in a different way so that it is made more parallel-friendly. In order to show performance of system, we will use TSPLIB [33] sample problems with different levels of parallelization. As one can check, distances between cities are rounded to the nearest integer value in the TSPLIB optimal tour report. To keep consistency of literature, we adopt this assumption of TSPLIB for distances between cities as well. Hereafter, we call our new system as Parallelized Memetic Self-Organizing Map (PMSOM). For the sake of simplicity, we will apply our algorithm over Euclidean TSP samples but some researchers have demonstrated that SOM is applicable on non-Euclidean TSP as well [34].

At the beginning, PMSOM divides cities between municipalities by the well-known K-means clustering algorithm [35]. Municipalities are completely independent of each other and evolving separately. Each municipality contains a population of individuals which are evolving by SOM rules and by adopting evolutionary mechanism of Créput and Koukam [30], the weakest answer is replaced by the best answer at some periods. After the convergence of the municipality to a sub-optimal tour of assigned cities, it will wait for the neighborhood municipalities to converge. Then, the blend operator merges two adjacent converged municipalities and this process continues until one municipality is left. The answer for the final municipality is the final answer of TSP.

Therefore, the major contributions of this study are as follows:

1. We introduce a parallel-friendly system based on a self-organizing map to solve large-scale traveling salesman problems.
2. We present a divide and conquer method to split large problems to a set of small problems and collect the results in an efficient way.
3. We experiment new system over TSPLIB sample problems with different levels of parallelization.

Although, aforementioned mechanism has some advantages, it has also one drawback. It gives the final answer when all partitions are solved and merged together.

The rest of article is organized as follows. Firstly, the building-block of method is introduced in Section 2. The principle of PMSOM is presented in Section 3. Then, Section 4 reports experimental analysis of proposed method. Finally, Section 6 concludes.

2. The Kohonen network incorporating explicit statistics

Although, Pure Kohonen network works well enough in Créput and Koukam [30] but we decided to use KNIES [24] as the building block of our methodology because of the following reasons:

1. Créput and Koukam [30] mentioned that trying more advanced learning rules by individuals may improve the algorithm.
2. KNIES [24] has shown better performance than the Pure Kohonen network, the Guilty Net of Burke and Damany [36], and the approach of Angniol et al. [37].
3. KNIES uses the statistical properties of the data points which seems necessary when we divide the map into municipalities by K-means clustering algorithm.

As alluded previously, each municipality includes population of SOM networks. KNIES uses global statistical information of cities to improve the final answer. Indeed, the basic idea lies in the fact that the mean of the given set of cities is known and the mean of final tour by SOM should be similar to the mean of the coordinates of the cities. In other words, in the two-dimensional case, average horizontal and vertical positions of all neurons of a tour should be equal to average horizontal and vertical coordinates of all the given cities, respectively. To do this, KNIES decomposes every training step into two phases, namely the attracting and dispersing phases to keep the mean of the tour the same as for the given set of cities in each iteration.

In the learning phase, we introduce a city (X_i) randomly from N cities to the network and neurons compete with each other. The closest neuron (Y_{j^*}) will win the competition. After that, all neurons in the activation bubble (Y_j, $j \in B_{j^*}$) migrate toward introduced city by Eq.(1) and the rest of neurons outside of activation bubble (Y_j, $j \notin B_{j^*}$) dispersed in such a way that mean of the tour coincide with the mean of the cities coordinates.

$$Y_j(t+1) = \begin{cases} Y_j(t) + W(j, j^*)(X_i - Y_j(t)) & j \in B_{j^*} \\ Y_j(t) + \frac{\sum_{i \in B_{j^*}} (Y_i(t+1) - Y_i(t))}{M - |B_{j^*}|} & j \notin B_{j^*} \end{cases} \quad (1)$$

The farther the neuron is from the winner, the less affected it is. This rule can be implemented by defining $W(j, j^*)$ as gaussian kernel function as Eq.(2).

$$W(j, j^*) = e^{\frac{-d(j,j^*)}{2\sigma^2}} \quad (2)$$

where M denotes number of neurons, $d(j, j^*) = min\{|j - j^*|, M - |j - j^*|\}$ and σ is standard deviation of kernel function. σ reduces over time to decrease effect of bubble of activity to more distant neurons and play role of adjustment at the end of learning phase [38].

Although, Eq.(2) has proven its eligibility but because of usage frequency, we employ another simplified version of the function to accelerate computation.

$$W(j, j^*) = \left(1 - \frac{d(j, j^*)}{|B_{j^*}|}\right)^{\beta_t} \quad (3)$$

where β_t is an increasing value from zero at the beginning ($t = 0$) to β_T at the end of time span ($\beta_t = \frac{t\beta_T}{T}$).

Figure 1 demonstrates effect of the attracting and dispersing phases on expansion of neuron ring. Initially, network starts from a ring at the center of map and then expands by considering location of introduced cities in the next iterations.

Angniol et al. [37] had done extensive analysis on the number of neurons in SOM. To avoid oscillation of neurons between different cities, they proposed that the number of neurons should be greater than number of cities ($M \geq 3N$). In our study we assume a fixed number of neurons ($M = 5N$) but a variable number of neurons was also studied by Angniol et al. [37] and Boeres et al. [39].

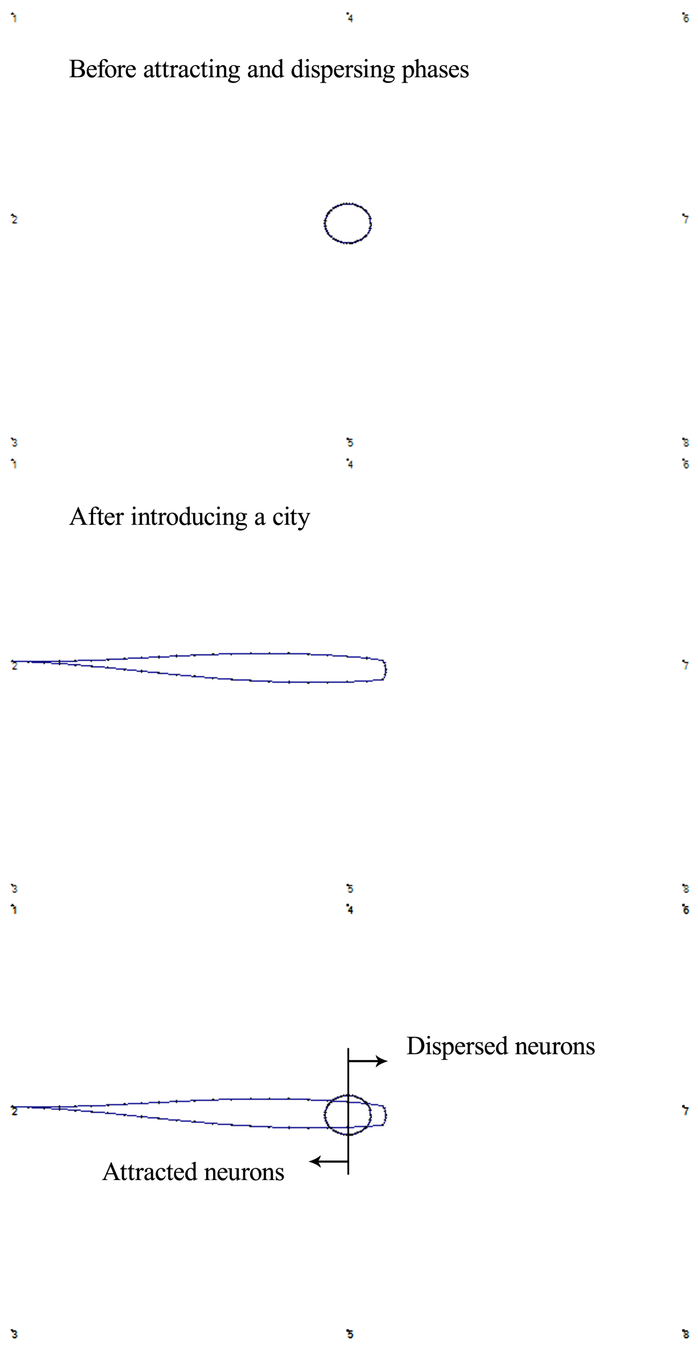

Figure 1: Effect of the attracting and dispersing phases between two subsequent iterations

3. Parallelized memetic self-organizing map

In this section, the parallelized memetic self-organizing map (PMSOM) will be explained in detail. At the beginning, we will introduce a dividing mechanism that creates municipalities. After that, a parallel-friendly evolutionary mechanism for each municipality will be discussed in detail and then we will elaborate the blend operator which aggregates sub-tours. It is worth mentioning that in each section we also talk about the time complexity of algorithms. Lastly, explaining termination condition of the

PMSOM and the adjusted values for the parameters will finalize this section.

3.1. Creating municipalities

In order to have a parallel-friendly method for solving TSP, we need a good algorithm to divide huge maps into smaller regions. However, this algorithm needs some prerequisites as explained below.

- First of all, created regions should be continuous, i.e. cities assigned to a region should not be separated from each other by another region.
- Secondly, the devised algorithm has to keep and represent the topological information of the problem (e.g. outlying cities may need to be considered as different regions).
- Thirdly, the clustering algorithm should be fast enough to handle large problems.

One simple way to implement such an algorithm is to divide the whole map into K groups using a rectangular grid and to assign cities in each group to one municipality. This method, however, has one major drawback: the algorithm might assign close-by city to an undesirable municipality just because of falling into assigned municipality grid cell. Another method is to apply a well-known clustering algorithm such as the K-means algorithm to find regions and their centroids. Figure 2 depicts *att*48 after dividing cities into three groups with the K-means algorithm. The time complexity of the K-means algorithm is NP-hard but by sacrificing accuracy, we can reach the proper clusters in a reasonable time.

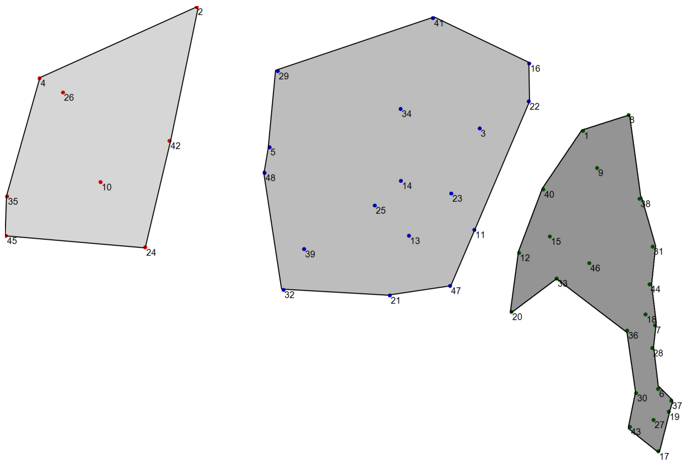

Figure 2: *att*48 after dividing into three groups: light gray, gray, dark gray

After finding clusters (municipalities)[1], we need to determine adjacent municipalities. If two clusters are far from each other or they have another cluster in between, merging them together is erroneous because they may forget the local information of the sub-tours. For this reason, we will implement an algorithm to find the most reasonable adjacent clusters. Algorithm 1 determines which clusters are adjacent and which are not.

Algorithm 1 determining adjacent municipalities.

```
 1: for $a, b \in C$ do
 2:     Suppose $a, b$ are adjacent
 3:     for  $k, l \in C$ and $k, l \neq a, b$ do
 4:         if $(\overrightarrow{P_aP_b} \cap \overrightarrow{P_kP_l} \neq \emptyset)$ then
 5:             $a, b$ are not adjacent
 6:             Break
 7:         else
 8:             if $\|\overrightarrow{P_aP_b}\| \geq \mathbf{E}_{r \in C}\|\overrightarrow{P_aP_r}\|$ then
 9:                 $a, b$ are not adjacent
10:                 Break
11:             end if
12:         end if
13:     end for
14: end for
```

C is defined as the set of all K clusters and P_a is centroid coordinates of cluster $a \in C$. Algorithm 1 starts by choosing two cluster's $(a, b \in C)$ centroids for the line segment $\overrightarrow{P_aP_b}$, if there are two other clusters $(k, l \in C)$ that their centroids line segment intersect then it means a and b are not adjacent. In addition to the previous condition, we add the distance condition so that two adjacent clusters should not be too faraway from each other (i.e. the distance between a and b should be less than the average distance of a to all other neighborhoods). Time complexity of Algorithm 1 is in order of $O(K^4)$ where K is the number of clusters.

3.2. Evolving mechanism

After creating municipalities by the K-means clustering algorithm, we need to create a parallel-friendly system to let municipalities separately evolve. Thus, for each municipality, we create a population of KNIES neural networks to create the sub-tour of that municipality under an evolutionary mechanism. The evolutionary mechanism consistently replaces the worst answer with the best answer. This mechanism guides the system to enhance the quality of the answer iteratively and has proven its eligibility in the memetic SOM [30]. Because the evolving mechanism in each municipality is completely independent of the other municipalities, we can easily employ parallel programming (or multi-thread programming on a single computer) [40]. It is also easy to prove that by considering just one municipality as a number of clusters, PMSOM will diminish to the memetic SOM [30]. Consequently, PMSOM could be a more generalized version of memetic SOM and later on we will compare the result of PMSOM with memetic SOM

[1]In this work, cluster and municipality words are interchangeable.

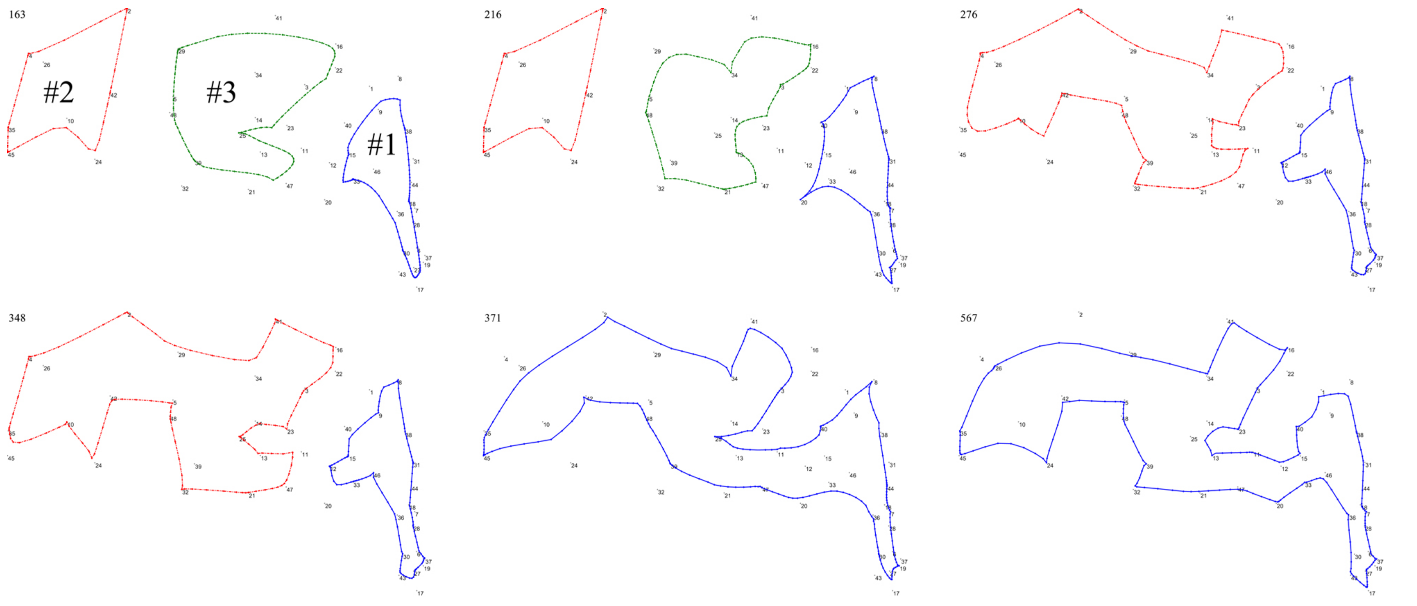

Figure 3: Evolution of *att*48 by PMSOM

as the latest winner of the neural network-based methods for solving TSP.

After some iteration (which is dependent on the assigned cities' topology and KNIES's parameters), the learning rule of the KNIES causes the synaptic weights of an individual to converge to a sub-tour. If the total transformation of all individuals becomes negligible, we call the municipality converged: a converged municipality needs to wait (be inactive) for other municipalities to converge as well. After that, the blend operator combines inactive municipalities. This activation-deactivation mechanism enables us to efficiently use system resources. Figure 3 shows the evolution of *att*48 in 6 different steps from top-left to down-right. At the beginning, the three municipalities individually evolved. In iteration #216, the second municipality deactivated and was waiting for the third one to converge. After convergence of the third cluster, it combined with the second cluster, thus creating to a bigger cluster. Finally, in iteration #371, all municipalities are combined and evolved until finding the optimal tour exactly after iteration #567. It is worth mentioning, after iteration #567, termination phase (see Subsection 3.4) will be called which results in a tour (in this case optimal tour) passing through all cities. However, we exclude the final tour to keep the figure as simple as possible.

Algorithm 2 shows the learning mechanism of each municipality. If the municipality has not yet converged, it will start to randomly choose a subset of assigned cities and apply KNIES learning role over each member of the population. Due to implementing evolutionary behavior, the best answer (line 8) and worst answer (line 11) of the municipality should be updated. The best solution has to replace the worst solution (line 13). Moreover, the algorithm observes the deformation of the best solution of population to keep track of changes and while the improvement is less than threshold (ϵ), it increases the *Stopped* variable. Otherwise, the *Stopped* variable resets. Eventually, if the number of the *Stopped* frames exceeds the *Maximum Pause* the municipality has converged and should be deactivated but if there is just one municipality, it means the system found the solution of given TSP and the refining step is necessary. The *Maximum Pause* can be a function of active clusters; therefore, at the beginning with the maximum number of active clusters, we will wait less and at the end we need to wait more for fine tuning.

Because municipalities are separately evolving, the worst case time complexity of Algorithm 2 in each iteration is exactly equal to the memetic SOM when assigned cities to a municipality are $N-K$ (again the order of N when we assume $N \gg K$) and for the rest, each a city was assigned. By mathematical notation, it has $O(|Population| \times N \times 2M_p)$ where the p is an individual of the municipality with the highest number of assigned cities. But the best case time complexity is when all municipalities have the same number of cities and then we will have $\theta(|Population| \times \frac{N}{K} \times 2M_p)$.

Finally, the aforementioned method for parallelization is more computational-friendly than memetic SOM. because, problem is divided into several subproblems that can be easily solved by simple feasible computers at the beginning of the process. At the final stages by aggregating results in a potent computer, we will reach convergence in a fewer number of iterations.

3.3. Blend operator

As mentioned in Subsection 3.2, the blend operator plays a pivotal role in reaching a high-quality solution. The blend operator also serves as a bottleneck while two in-

Algorithm 2 Learning mechanism of each municipality.

```
while Municipality is not converged do
    $t = t + 1$
    for each $KNIES_p \in$ Population do
        for $X_i \in$ Assigned cities do
            $KNIES_p$.Learn($X_i$, $\beta_t = \frac{t\beta_T}{T}$)
        end for
        if $KNIES_p > \overline{KNIES}$ then
            $\overline{KNIES(t)} = KNIES_p$
        end if
        if $KNIES_p < \underline{KNIES}$ then
            $\underline{KNIES(t)} = KNIES_p$
        end if
        $\underline{KNIES(t)} = \overline{KNIES(t)}$
    end for
    if $\frac{\overline{KNIES(t-1)} - \overline{KNIES(t)}}{\overline{KNIES(t)}} < \epsilon$ then
        Stopped = Stopped + 1
        if Stopped $\geq$ Maximum pause then
            Municipality converged
        end if
    else
        Stopped = 0
    end if
end while
```

active municipalities are waiting for this operator to combine them. First of all, the blend operator should take care of the adjacency of clusters. When, two frozen municipalities are adjacent then joining them is possible. Secondly, the algorithm should prefer to mix closer neighborhoods between adjacent clusters. Therefore, the blend operator should have following properties.

1. Create a tour as short as possible
2. Cover all cities in both candidates
3. Avoid creating kinks in our resultant tour.

Suppose the algorithm finds two municipalities a and b as candidates to be blended (in here, we are assuming each municipality has just one KNIES in its population).

- Finding the closest neurons between a and b. We name A as the closest neuron in a to b and B as the closest neuron in b to a.
- Find two other neurons between a and b so that they become close to A and B and we are not missing any city between them. We call C as the closest neuron after B in b to a and D as the closest neuron after A in a to b.
- Creating a tour by following the proper path starting from A to B and continuing from B to C by picking the proper direction that neglects no cities in b. Then adding an arc from C to D and completing the tour by going from D to A so that it covers all cities in a. Figure 4 explains the blend operator by means of a graphic.
- Generally, kinks make tours more complex and lengthy; therefore, removing the kinks is a wise action. Algorithm 3 helps to remove kinks of mixed clusters. In spite of some similarities with the famous 2-opt algorithm [41], there are also some differences to keep it fast.

  Differences between Algorithm 3 and 2-opt [41] algorithms are as follows:

  1. Contrary to complete 2-opt, Algorithm 3 does not check all the combinations of j_1 and j_2. Its focus is on collided lines and removing kinks by first diagnosing location of collision.
  2. In Algorithm 3, we do not check quality of resulted tour after removing kink since calculations for each pair of neighborhoods can be time consuming.

Algorithm 3 Omitting kinks from tour.

```
for $j_1 = 1$ to $M - 1$ do
    for $j_2 = 1$ to $j_1 - 2$ do
        if $(\overrightarrow{Y_{j_1}Y_{j_1+1}} \cap \overrightarrow{Y_{j_2}Y_{j_2+1}}) \neq \emptyset$ then
            Swap($j_1$, $j_2$)
        end if
    end for
end for
```

Algorithm 3, tries to find the intersection of two arcs in the sequence and opens the kinks by reversely reading the sequence between j_1 and j_2 (Swap(j_1, j_2)). Figure 5 demonstrates Algorithm 3 steps over a sample tour. Time complexity of Algorithm 3 is $O(M^2)$ when M neurons are connected with $M - 1$ links.

By introducing a penalty parameter for the winning numbers of each neuron, we can avoid kinks in the learning phase in each municipality; therefore in order to accelerate process, one can just check $\overrightarrow{AB}$ and $\overrightarrow{CD}$. Because blending two municipalities as illustrated in Figure 4 can sometimes result in kink (created line between point A and point B and line between point C and point D may cross one another).

The population in the new generation (the population of the mixed municipality) should be constituted in such a way that first keeps the best solution in the pool and second preserves the variance of previous generations as well. Therefore, we insert a solution by mixing the best solutions of two neighborhood clusters at the beginning, and then, randomly mixing the rest of the population to avoid narrowing down solutions rapidly.

Eventually, updating the adjacency matrix of clusters is necessary. To do that, first we need to update the centroid of the mixed municipalities from Eq. 4

$$P_{a\cup b} = \frac{N_a P_a + N_b P_b}{N_a + N_b} \tag{4}$$

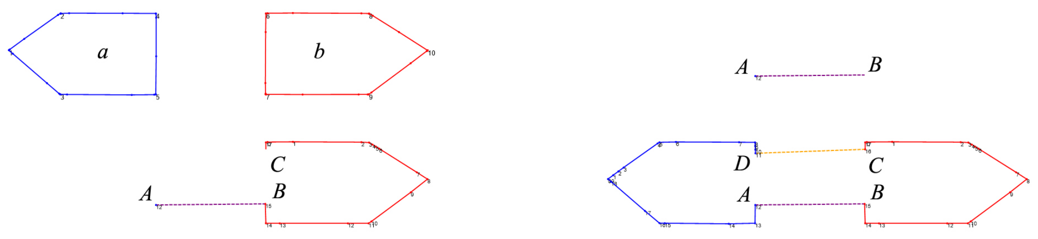

Figure 4: Blending two municipalities by mean of (A, B) and (C, D) arcs.

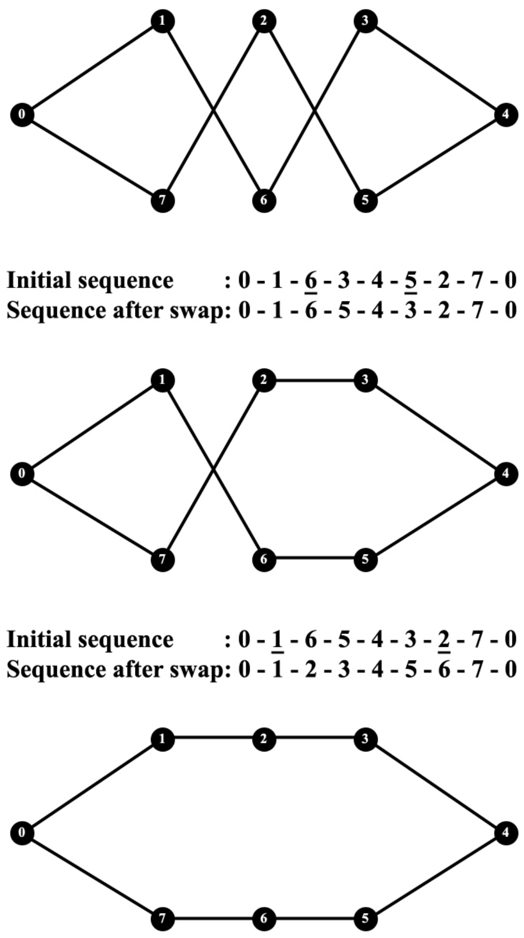
Initial sequence : 0 - 1 - 6 - 3 - 4 - 5 - 2 - 7 - 0
Sequence after swap: 0 - 1 - 6 - 5 - 4 - 3 - 2 - 7 - 0
Initial sequence : 0 - 1 - 6 - 5 - 4 - 3 - 2 - 7 - 0
Sequence after swap: 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 0

Figure 5: Removing kinks from sample tour.

where $P_{a \cup b}$ denotes centroid coordinates of the blended municipality and N_a, N_b are the number of assigned cities to the municipality $a \in C$ and $b \in C$ respectively. Then, Algorithm 1 updates the adjacency matrix. Finally, the resultant municipality will be activated again to evolve.

3.4. Termination phase

This subsection explains the termination condition and final refinement processes. As mentioned before, when the algorithm mixes all municipalities together, the last and only municipality continues to evolve when the convergence condition is satisfied. This means, PMSOM found the final solution and just we need to go over all neurons of the best individual and find the index of the nearest city for each one. Then, after creating such a sequence we might have more than one neuron for each city. To avoid this, we filter out repeated indices in the resultant sequence. Finally, by going through the whole sequence, the final tour length is computed. The Figure 6 describes our proposed methodology.

3.5. Parameter adjustment

To maintain consistency with the memetic SOM [30] report, we adopt their parameter setting; therefore, we assume 10 individuals for each municipality [2] with the consecutive update and 60 generations in the PMSOM [3] since in the memetic SOM, Créput and Koukam have determined 20 and 40 iterations, respectively, for the construction loop and the improvement loop.

Additionally, we adopted two terms $\%PDM$ and $\%PDB$ from Créput and Koukam [30], as the percentage deviation from the optimum of the mean solution value over different runs and the percentage deviation from the optimum of the best solution value, respectively. We use these terms in this subsection and the next section.

β_T in Eq. (3) depends on the number of cities (N); thus, for the fewer number of cities we need more gentle transformation, consequently, smaller β_T is necessary but for the bigger problems, bigger β_T could be handy. By trial-and-error, we formulate $\beta_T = 1 + cN$ as a suitable candidate where the c is a random value between 0.08 and 0.3. Because these values are randomly assigned between individuals; therefore, the weaker values will be replaced by the better ones in a evolutionary mechanism for each.

In the Algorithm 2, finding proper values for ϵ and *Maximum Pause* are interdependent; therefore, we conduct experiments on 5 different case studies from TSPLIB (*Eil*51, *Eil*76, *pr*299, *pr*439, and *rl*1304). In each case study, 16 different experiments are conducted on each setting (with different number of clusters from 1 to 4) and the results are aggregated. Due to the fact that the goal of this method is to solve large problems, we utilize the weighted average method based on the size of the problems

[2] $Population = 10$ in the Algorithm 2
[3] $T = 60$ for the computing β_t in the Eq.(3)

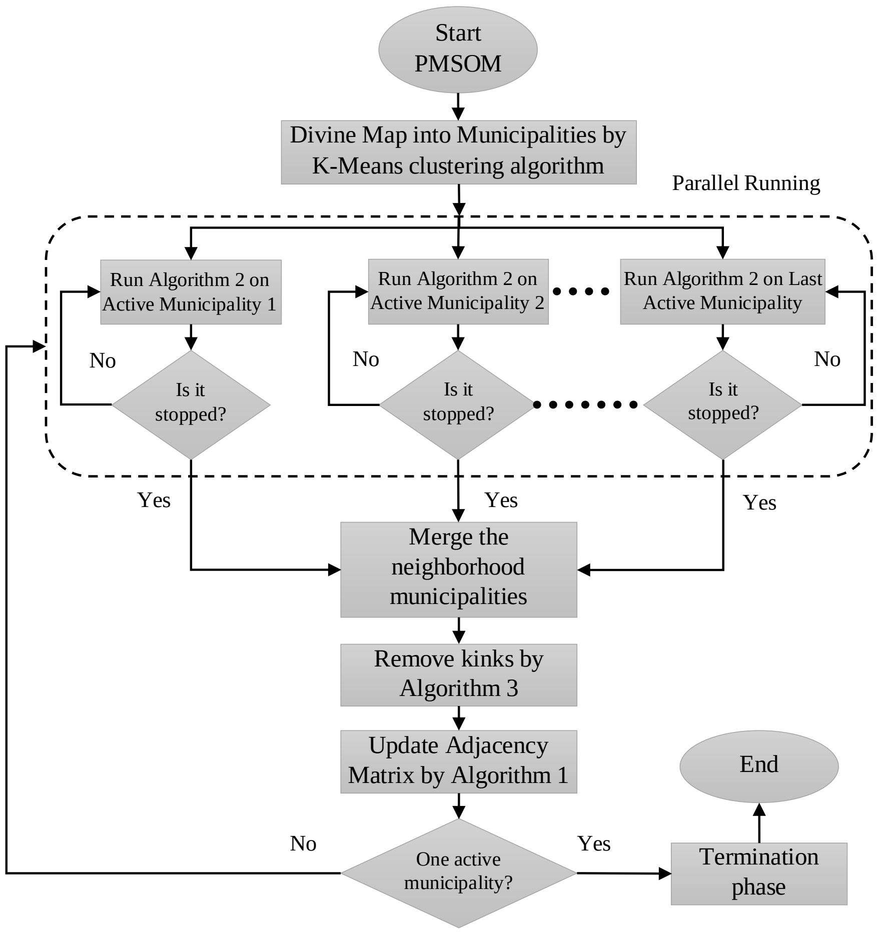
Start PMSOM
Divine Map into Municipalities by K-Means clustering algorithm
Parallel Running
Run Algorithm 2 on Active Municipality 1
Run Algorithm 2 on Active Municipality 2
Run Algorithm 2 on Last Active Municipality
No
Is it stopped?
No
Is it stopped?
Is it stopped?
No
Yes
Yes
Yes
Merge the neighborhood municipalities
Remove kinks by Algorithm 3
Update Adjacency Matrix by Algorithm 1
End
No
Yes
One active municipality?
Termination phase

Figure 6: Flowchart of parallelized memetic self-organizing map (PMSOM).

to aggregate *PDM*%s of each case. Finally, to protect our judgement in the next section unaffected, we will not use these problems in our comparison with the memetic SOM.

Table 1: Adjusting parameters for PMSOM

Weights				
Eil51	Eil76	pr299	pr439	rl1304
0.024	0.035	0.138	0.202	0.601

	PDM (%)	ϵ		
		0.1K	0.01K	0.001K
Max. Pause	$\frac{10}{e^{-0.1K}}$	8.61	7.57	7.44
	$\frac{10}{e^{-0.5K}}$	7.93	7.44	**7.41**
	$\frac{10}{e^{-0.9K}}$	8.32	7.83	8.22

As Table 1 declares, the best values for our ϵ and *Maximum Pause* are $0.001K$ and $\frac{10}{e^{-0.5K}}$, respectively, where K is the number of municipalities. Figure 7 represents the interval plot with 95% confidence for all settings and all the case studies.

4. Numerical analysis

In this section, we compute the performance of the proposed method on different case studies with different levels of parallelization. Afterwards, we compare our result with the memetic SOM [30] as the previous winner of NN-based methods for solving TSP.

4.1. Computation analysis

To corroborate the effectiveness of PMSOM, we employ the Multi-threading technology implemented in $C\#$. We utilize a typical 64-bit computer which benefits from the Intel Core2 Quad CPU (Q8200) with the frequency of 2.34GHz and 8 GB memory. Each case study was examined with different numbers of municipalities (K from 1 to 8, depending on the result and the size of problem) and for each K four times.

To have a fair comparison with Créput and Koukam [30], those municipalities that are evolving even after iteration T will be imposed to be inactive and merge with others. Therefore, our method lets municipalities evolve at most T iterations.

In the beginning, we applied PMSOM on the different cases which are categorized based on the number of cities.

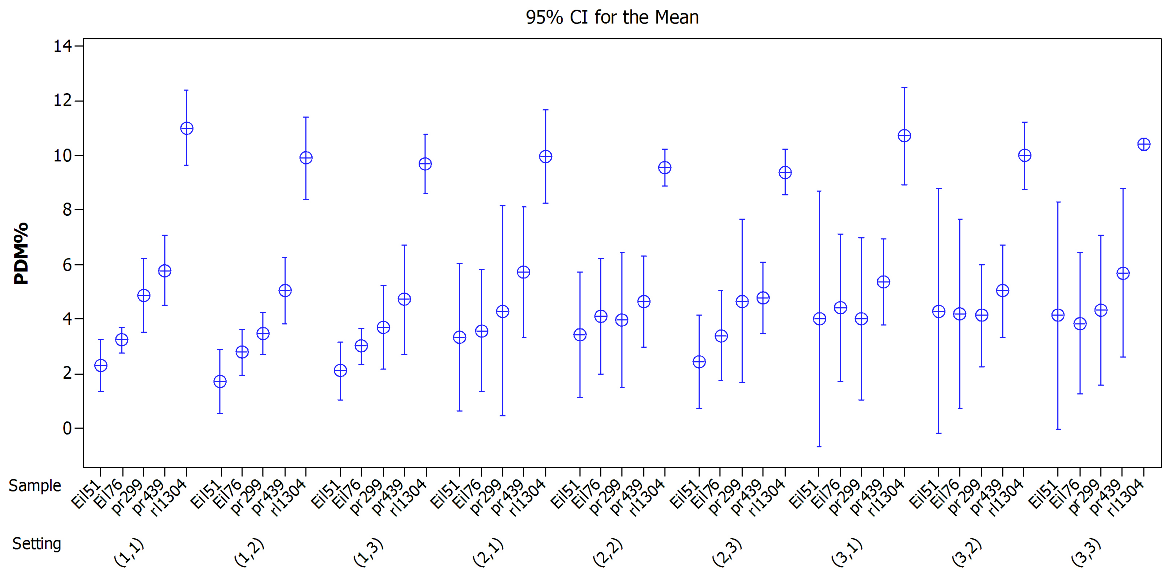

Figure 7: Interval Plot of *PDM%* with 95% confidence level.

Figure 8 depicts effect of the cluster number on computation time on the small case studies. As it was expected, increasing the number of municipalities causes the computation time to increase. It indicates that for the small problems, the sequential memetic SOM could be a better choice than PMSOM.

On the contrary, Figure 9 delineates the effectiveness of the proposed method, especially, for the large case studies. As one can see, computations are accelerating when the number of municipalities (K) are ascending in all the cases. Accordingly, this figure points out that most of the speed derives from changing K from 1 to 2.

4.2. Comparing the quality of solutions

After showing the positive effect of the parallelized method on CPU time, the only remaining part of our comparison would be the quality of results. In here to keep consistency, we are enforcing our method so as to merge all remaining clusters at the end of $T = 60$. The results are organized in Table 2. In the first column, we address our case studies from TSPBLIB, in the second column, the optimal solutions are specified. The next four columns show the optimal number of clusters, the computation time improvement ratio by conducting experiments on K^* parallel clusters rather than just one cluster, $\%PDM$, and $\%PDB$ of the PMSOM method respectively. In the last two columns, we borrow the best reported results of the memetic SOM for the mentioned case studies [30].

It is worth noting that the table is separated into three categories by two lines. The first category includes small-sized instances, the second category contains middle-sized instances and the last category contains large-sized instances.

By illustrating the computation time ratio instead of time in seconds, one can easily compute this number for the different types of computers and compare them instead of converting times between different computers inaccurately. It also shows how successful is our method in exploiting system resources.

The first result we obtain is that by increasing the number of cities, the number of clusters becomes more and more important. The size of problem is not, for sure, the only factor but it is an important one along with the topology of the problem. Accordingly, computations accelerates by increasing number of optimal clusters. Figure 10 shows effect of K^* over computation time ratio.

For those cases that K^* is one, it means that memetic SOM is as capable as PMSOM but because of different implementation, they could not reach to the better results for the reported cases. However from $KroA150$, PMSOM beats the memetic SOM approach. In Table 2 better results are bolded for ease of comparison. According to Table 2, by increasing the number of municipalities, we can get faster and even more accurate results, a finding which means utilizing PMSOM for large problems is beneficial. Finally, the average benefit of the using proposed parallel mechanism is a yield of more than 4 times faster than the non-parallel system. Additionally, PMSOM could obtain better results (both $\%PDM$ and $\%PDB$) on average and more interestingly, PMSOM gets a better $\%PDM$ in all the cases.

For the large-sized instances, Créput and Koukam [30] did not report their result for the $GenC = 20$ and $GenI = 40$. However, they reported for the $GenC = 80$ and $GenI = 400$. By comparing the PMSOM method for $T = 60$ rather than $T = 480$ (which has the less number of iterations) in the $pcb3038$ and $pla7397$ cases, we got acceptable results. However, for the $rl5915$ and $kz9976$, we could not obtain better results in $T = 60$ but we got better results with the same number of iterations ($T = 480$).

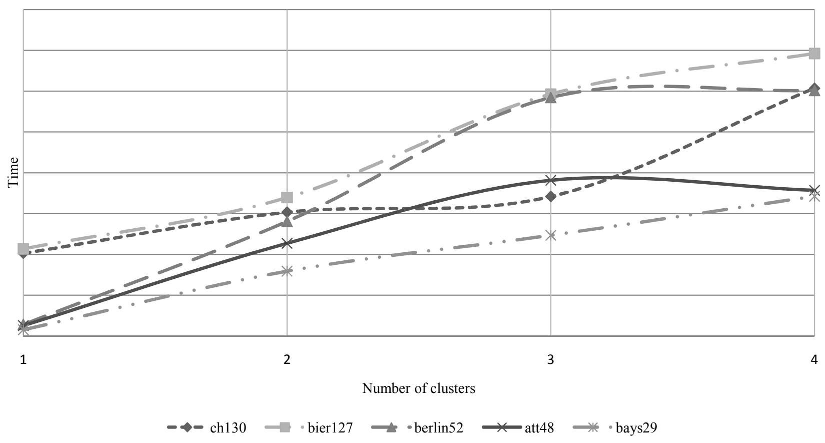

Figure 8: The trend of computation time for small case studies and different number of clusters.

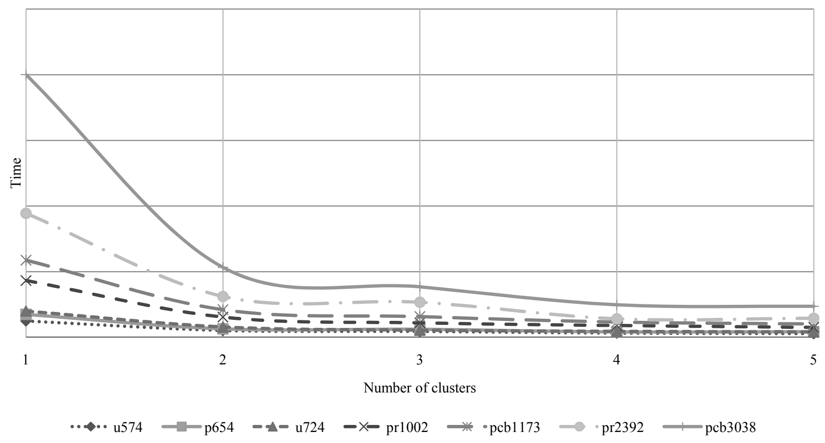

Figure 9: The trend of computation time for medium and large case studies on different numbers of clusters.

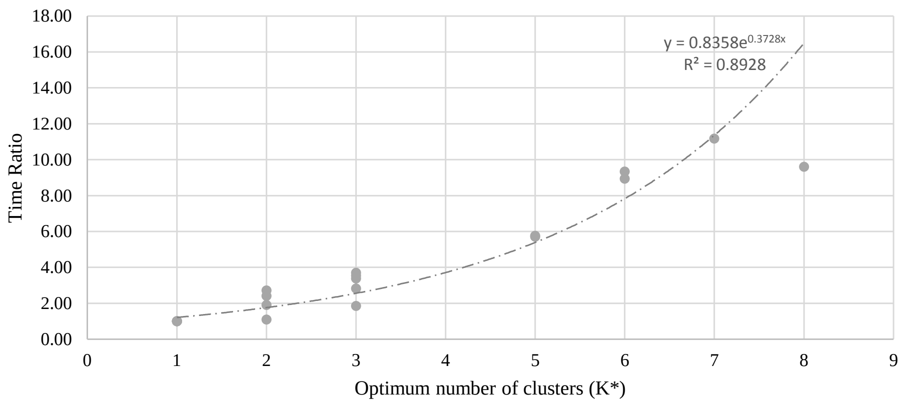

Figure 10: Computation time's trend over different K^*.

Even though, we could not find exact formulation for the optimal number of clusters, it seems there is a relationship between number of cities (N) and optimal number of clusters (K^*). Figure 11 suggests $\tilde{K}^* = 0.2352N^{0.3849}$ as a good approximation. For example, if we consider $fi10639$ then $\tilde{K}^* = 8.34$. Even if we check the clusters from 8 to 10 again it will be 5.53 times on average faster than non-parallelized version ($K = 1$).

5. Acknowledgement

Authors of this article would like to thank Edris Esmaeili because of his support for implementing parallelized system effectively. Also, authors would like to thank Nancy Karabeyoglu for delivering helpful advice.

Table 2: Comparison with best reported result of memetic SOM[30]

		Parallelized memetic SOM				Memetic SOM	
Problem	**Optimal**	**K***	**Time Ratio**	**%PDM**	**%PDB**	**%PDM**	**%PDB**
*bays*29	**2020**	1	1.00	0.62	0.10	-	-
*att*48	**33522**	1	1.00	0.80	0.19	-	-
*berlin*52	**7542**	1	1.00	**1.40**	**0.00**	1.63	0.00
*bier*127	**118282**	1	1.00	**1.56**	**0.39**	2.78	1.25
*ch*130	**6110**	1	1.00	**0.79**	**0.43**	2.83	0.80
*KroA*150	**26524**	2	1.09	**1.84**	**1.41**	2.73	1.64
*KroA*200	**29368**	2	1.90	**1.44**	1.18	2.20	**1.08**
*lin*318	**42029**	3	1.85	**4.60**	**2.88**	4.95	3.48
*pcb*442	**50778**	3	2.82	**5.09**	3.65	6.08	**3.57**
*u*507	**36905**	2	2.41	**4.01**	**3.35**	5.08	4.09
*att*532	**27686**	3	3.38	**3.76**	**3.04**	4.21	3.29
*p*654	**34643**	2	2.72	**5.07**	3.78	5.13	**2.51**
*u*724	**41910**	3	3.71	**4.94**	**4.28**	5.36	4.64
*pr*1002	**259045**	5	5.71	**4.67**	**4.37**	6.11	4.75
*pcb*1173	**56892**	5	5.76	**7.90**	**7.32**	8.66	8.20
*pr*2392	**378032**	3	3.54	**6.48**	**6.25**	8.16	7.32
*pcb*3038	**137694**	8	9.61	**7.74**	7.62	7.88	**7.10**
*rl*5915*	**565530**	6	9.33	**10.06**	**9.68**	12.94	12.02
*pla*7397	**23260728**	6	8.94	**9.84**	**8.71**	10.19	9.11
*kz*9976*	**1061881**	7	10.98	**6.58**	**5.90**	7.72	7.18
*fi*10639*	**520527**	9	16.62	**6.03**	**5.87**	6.93	6.66
Average			4.54	**4.53**	**3.83**	5.87	4.67

- there is no report for these cases of memetic SOM [30].
* [30] has no report for $T = 60$; so, we changed $T = 480$ when $GenC = 80$ and $GenI = 400$.

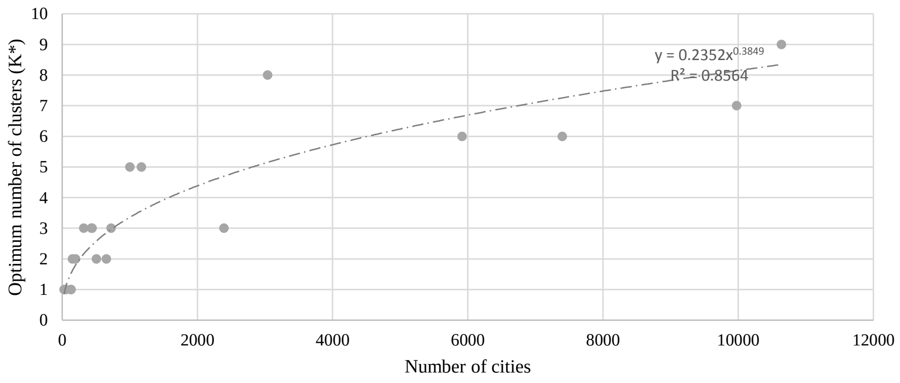

Figure 11: Optimum Number of clusters (K^*) over number of cities (N).

6. Conclusion

By incorporating a clustering algorithm, in addition to a self-organizing map in a evolutionary mechanism, we show that proposed mechanism utilizes system resources more effectively without losing accuracy. We claim that proposed system is more generalized form of memetic SOM [30] as the last winner of the neural-network applications for solving the Euclidean traveling salesman problem. To provide evidence of eligibility for the proposed method, Euclidean TSPs from TSPLIB are experimented. Our proposed algorithm is tested on various case studies with different levels of parallelization.

All in all, the results show for the large problems that invoking a higher number of clusters seems necessary and the presented methodology is more than 4 times faster than those of non-parallel systems on average. Furthermore, the presented method (PMSOM) seems accurate enough to compete with the best reported result of memetic SOM.

This study can be extended in some directions. As mentioned above, the topology of the problem has vital information which can enable us to exploit it in a parallelized fashion. We merely used the well-known K-means clustering algorithm to divide cities between neighborhoods. Although it works well and outperforms many cases, but in some cases our work could not catch the topology successfully; therefore, further studies in finding more effective clustering mechanism are needed. Considering different neural networks in individuals could be another direction for future works. Besides, employing proposed algorithm on other problems such as Vehicle Routing Problem is worth doing.

References

[1] R. G. Michael, D. S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, WH Freeman & Co., San Francisco (1979).

[2] Y. Nagata, D. Soler, A new genetic algorithm for the asymmetric traveling salesman problem, Expert Systems with Applications 39 (2012) 8947 –53.

[3] J. Yang, C. Wu, H. P. Lee, Y. Liang, Solving traveling salesman problems using generalized chromosome genetic algorithm, Progress in Natural Science 18 (2008) 887 –92.

[4] C.-N. Fiechter, A parallel tabu search algorithm for large traveling salesman problems, Discrete Applied Mathematics 51 (1994) 243 –67.

[5] M. Hasegawa, T. Ikeguchi, K. Aihara, Solving large scale traveling salesman problems by chaotic neurodynamics, Neural Networks 15 (2002) 271 –83.

[6] S.-M. Chen, C.-Y. Chien, Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques, Expert Systems with Applications 38 (2011) 14439 –50.

[7] Y. Chen, P. Zhang, Optimized annealing of traveling salesman problem from the nth-nearest-neighbor distribution, Physica A: Statistical Mechanics and its Applications 371 (2006) 627 –32.

[8] S.-M. Chen, C.-Y. Chien, Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques, Expert Systems with Applications 38 (2011) 14439–50.

[9] S.-M. Chen, C.-Y. Chien, Parallelized genetic ant colony systems for solving the traveling salesman problem, Expert Systems with Applications 38 (2011) 3873–83.

[10] O. Johnson, J. Liu, Source code for biology and medicine, Source Code for Biology and Medicine 1 (2006).

[11] A. Reynolds, Chemotaxis can provide biological organisms with good solutions to the travelling salesman problem, Physical Review E 83 (2011) 052901–4.

[12] C. Korostensky, G. H. Gonnet, Using traveling salesman problem algorithms for evolutionary tree construction, Bioinformatics 16 (2000) 619–27.

[13] R. Agarwala, D. L. Applegate, D. Maglott, G. D. Schuler, A. A. Schäffer, A fast and scalable radiation hybrid map construction and integration strategy, Genome Research 10 (2000) 350–64.

[14] T. A. Endo, Probabilistic nucleotide assembling method for sequencing by hybridization, Bioinformatics 20 (2004) 2181–8.

[15] K. Fujimura, K. Obu-Cann, H. Tokutaka, Optimization of surface component mounting on the printed circuit board using SOM-TSP method, in: Neural Information Processing, 1999. Proceedings. ICONIP '99. 6th International Conference on, volume 1, pp. 131–136 vol.1.

[16] K. Fujimura, S. Fujiwaki, O.-C. Kwaw, H. Tokutaka, Optimization of electronic chip-mounting machine using SOM-TSP method with 5 dimensional data, in: Info-tech and Info-net, 2001. Proceedings. ICII 2001 - Beijing. 2001 International Conferences on, volume 4, pp. 26–31 vol.4.

[17] J. K. Lenstra, A. H. G. R. Kan, Some simple applications of the travelling salesman problem, Operational Research Quarterly (1970-1977) 26 (1975) pp. 717–733.

[18] A. Han, D. Zhu, A new DNA encoding method for traveling salesman problem, in: D.-S. Huang, K. Li, G. Irwin (Eds.), Computational Intelligence and Bioinformatics, volume 4115 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 328–35.

[19] J. Y. Lee, S.-Y. Shin, T. H. Park, B.-T. Zhang, Solving traveling salesman problems with DNA molecules encoding numerical values, BioSystems 78 (2004) 39–47.

[20] D. Banaszak, G. Dale, A. Watkins, J. Jordan, An optical technique for detecting fatigue cracks in aerospace structures, in: Instrumentation in Aerospace Simulation Facilities, 1999. ICIASF 99. 18th International Congress on, IEEE, pp. 27–1.

[21] K. Smith, An argument for abandoning the travelling salesman problem as a neural-network benchmark., IEEE transactions on neural networks/a publication of the IEEE Neural Networks Council 7 (1995) 1542–4.

[22] G. Wilson, G. Pawley, On the stability of the Travelling Salesman Problem algorithm of Hopfield and Tank, Biological Cybernetics 58 (1988) 63–70.

[23] T. Kohonen, Self-organization and associative memory, Self-Organization and Associative Memory, 100 figs. XV, 312 pages.. Springer-Verlag Berlin Heidelberg New York. Also Springer Series in Information Sciences, volume 8 1 (1988).

[24] N. Aras, B. J. Oommen, I. Altınel, The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem, Neural Networks 12 (1999) 1273–84.

[25] E. Cochrane, J. Beasley, The co-adaptive neural network approach to the Euclidean travelling salesman problem, Neural Networks 16 (2003) 1499–525.

[26] Y.-m. Cheung, L.-t. Law, Rival-model penalized self-organizing map, Neural Networks, IEEE Transactions on 18 (2007) 289–95.

[27] J. Zhang, X. Feng, B. Zhou, D. Ren, An overall-regional competitive self-organizing map neural network for the Euclidean traveling salesman problem, Neurocomputing 89 (2012) 1 – 11.

[28] K. Obermayer, H. Ritter, K. Schulten, Large-scale simulations of self-organizing neural networks on parallel computers: application to biological modelling, Parallel Computing 14 (1990) 381 – 404.

[29] S. A. Mulder, D. C. Wunsch, II, Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks, Neural Netw. 16 (2003) 827–32.

[30] J.-C. Créput, A. Koukam, A memetic neural network for the Euclidean traveling salesman problem, Neurocomputing 72 (2009) 1250–64.

[31] K.-S. Leung, H.-D. Jin, Z.-B. Xu, An expanding self-organizing neural network for the traveling salesman problem, Neurocomputing 62 (2004) 267–92.

[32] H.-D. Jin, K.-S. Leung, M.-L. Wong, Z.-B. Xu, An efficient self-organizing map designed by genetic algorithms for the traveling salesman problem, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 33 (2003) 877–88.

[33] G. Reinelt, TSPLIBA traveling salesman problem library, ORSA journal on computing 3 (1991) 376–84.

[34] J. Faigl, M. Kulich, V. Vonsek, L. Peuil, An application of the self-organizing map in the non-Euclidean traveling salesman problem, Neurocomputing 74 (2011) 671 –9.

[35] V. Faber, Clustering and the continuous k-means algorithm, Los Alamos Science 22 (1994) 138–44.

[36] L. I. Burke, P. Damany, The guilty net for the traveling salesman problem, Computers & Operations Research 19 (1992) 255–65.

[37] B. Angniol, G. de La Croix Vaubois, J.-Y. L. Texier, Self-organizing feature maps and the travelling salesman problem, Neural Networks 1 (1988) 289 –93.

[38] R. O. Duda, P. E. Hart, et al., Pattern classification and scene analysis, volume 3, Wiley New York, 1973.

[39] M. Boeres, L. de Carvalho, V. Barbosa, A faster elastic-net algorithm for the traveling salesman problem, in: Neural Networks, 1992. IJCNN., International Joint Conference on, volume 2, pp. 215–220 vol.2.

[40] B. Wilkinson, M. Allen, Parallel programming, volume 999, Prentice hall New Jersey, 1999.

[41] D. S. Johnson, L. A. McGeoch, The traveling salesman problem: A case study in local optimization, Local search in combinatorial optimization 1 (1997) 215–310.

Bihter Avşar received her B.Sc. degree in Genetics and Bioengineering in Yeditepe University, Istanbul and she completed her M.Sc. in Molecular Biology and Genetics area in Izmir Institute of Technology. She is currently a Ph.D. candidate in Biological Sciences and Bioengineering Department in Sabanci University. Her research interests are bioinformatics, computational biology, genomics, proteomics, biotechnology, plant genetics and molecular biology.

Danial Esmaeili Aliabadi received his B.Sc. and M.Sc. in industrial engineering at Islamic Azad University, Iran. He has been participated in RoboCup international competitions from 2009 to 2011. He received many national and international awards in the robotics game events. He is currently a Ph.D. candidate in the Department of Industrial Engineering, Sabanci University. His research interests include artificial intelligence, agent-based simulation, optimization, game theory and machine learning.