Applying Self-Organizing Kohonen Maps to the Traveling Salesman Problem

F. I. MAKSIMOV, V. V. BUKHVALOVA, I. G. BUROVA
Department of Computational Mathematics,
St. Petersburg State University,
7/9 Universitetskaya Embankment, St.Petersburg, 199034,
RUSSIA

Abstract: - This paper discusses solving the traveling salesman problem using Kohonen maps. We investigate the application of Self-Organizing Maps (SOM) to the Traveling Salesman Problem (TSP), develop Central Processing Unit (CPU)- and Graphics Processing Unit (GPU)-based implementations of the proposed approach, and evaluate their efficiency in SOM hyperparameter tuning using a multistage dimensionality reduction framework and Bayesian optimization. Bayesian optimization based on sequential trial selection was used to select the parameters. A new SOM configuration was defined for each trial, after which the map was trained, and the objective function value was calculated. This approach differs from exhaustive searches in that subsequent trials are selected based on the results already obtained.

Key-Words: - Self-Organizing Kohonen Maps, Traveling Salesman Problem, Bayesian optimization, CPU implementation, GPU implementation.

Received: August 16, 2026. Revised: September 2, 2026. Accepted: September 21, 2026. Published: November 6, 2026.

1 Introduction

The Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization and has retained both theoretical and applied significance for many decades [1]. In its substantive form, it consists of constructing the shortest closed route that passes through each vertex of a given set exactly once and returns to the starting point. Despite the simplicity of the formulation, the problem has high computational complexity: as the number of vertices increases, the number of possible routes quickly becomes so large that a direct enumeration ceases to be practically applicable. Therefore, the traveling salesman problem is a convenient model for studying methods of approximate optimization, intelligent search, and analysis of the computational efficiency of algorithms [1].

Traditional approaches to solving the traveling salesman problem include exact methods, simple heuristics, and metaheuristic algorithms. Exact methods are valuable because they yield an optimal solution, but their computational cost increases sharply with the number of nodes. Therefore, in applied research, approaches that focus not on a strict guarantee of optimality, but on achieving a reasonable compromise between route length, computation time, and result quality, are particularly important. In this context, self-organizing maps (SOMs) are of interest, allowing us to interpret route construction as a process of adaptation of a ring structure of neurons to the geometry of a set of points [2], [3].

SOMs occupy a prominent place among unsupervised neural network learning models. Their basic idea is that a set of neurons with weight vectors gradually restructures itself to match the structure of the input data, preserving the topological proximity of objects. When solving the traveling salesman problem, such a map can be viewed as a closed loop that, during the learning process, approximates the locations of cities or production points. After training is complete, the order in which neurons are traversed determines the route. This approach does not guarantee finding the global optimum, but it has a clear geometric interpretation, combines well with visual analysis, and allows for efficient software implementation.

One of the key challenges in applying SOM to the traveling salesman problem is the selection of hyperparameters. The quality of the resulting route is affected by the number of neurons, the initial learning rate, the neighborhood radius, their attenuation coefficients, and the number of training iterations. Inappropriate selection of these parameters can lead to slow convergence, premature stabilization of the map, or the appearance of inactive neurons not involved in route construction. Therefore, this paper considers hyperparameter

selection based on Bayesian optimization and a multi-step dimensionality reduction scheme, which allows for the automated search for effective configurations and reduces the dependence of the result on manual tuning. Another important aspect of the study relates to computational efficiency. Training an SOM involves repeatedly calculating distances to neurons, determining the winning neuron, calculating the neighborhood function, and updating the weight vectors. These operations are repetitive and amenable to vectorization, making the use of GPU computing and Compute Unified Device Architecture (CUDA) technology advisable. Comparison of CPU and GPU implementations allows us to evaluate how much transferring computations to the graphics processor speeds up training and expands the capabilities of conducting a series of computational experiments.

The object of this research is a method for solving the traveling salesman problem in its Euclidean setting and in applied routing problems. The subject of the study is Kohonen self-organizing maps, methods for their software implementation, methods for accelerating computations, and approaches to selecting hyperparameters for solving the traveling salesman problem. The goal of this work is to study the application of Self-Organizing Kohonen maps to solving the traveling salesman problem, develop CPU and GPU implementations of this approach, and evaluate its effectiveness using two approaches to finding SOM hyperparameters based on a multi-step dimensionality reduction scheme and Bayesian optimization.

To achieve this goal, the following tasks must be solved:
1. Consider the mathematical formulation of the traveling salesman problem, its application areas, and the main classes of methods for solving the TSP;
2. Study the theoretical foundations of SOMs and the specifics of their adaptation to the TSP;
3. Describe the SOM training algorithm taking into account the ring topology, neighborhood function, learning rate, and decay parameters;
4. Develop of CPU and GPU implementations of the SOM algorithm for solving TSP with further analysis of efficiency.

2 The Traveling Salesman Problem

Suppose a set of cities, service points, technological operations, or other objects are given, between which distances, travel times, or other costs are known. It is required to construct a closed route starting at a certain vertex, passing through each vertex exactly once, and returning to the starting point, with the total route length being minimal. This formulation makes the traveling salesman problem a universal model for a large number of processes related to the ordering of actions and the minimization of total costs [1]. Formally, the traveling salesman problem is defined on a complete weighted graph [1]:

$$G = (V, E), \quad V = \{v_1, v_2, \ldots, v_N\},$$

where V is the set of vertices, E is the set of edges, and each edge (v_i, v_j), is assigned a weight d_{ij}.

The value d_{ij} can be interpreted as distance, travel cost, travel time, or another cost indicator. If a route is defined by a permutation of vertices

$$\pi = (\pi_1, \pi_2, \ldots, \pi_N), \tag{1}$$

then its length is determined by the expression:

$$L(\pi) = \sum_{k=1}^{N-1} d_{\pi_k \pi_{k+1}} + d_{\pi_N \pi_1}.$$

The traveling salesman problem consists of finding a permutation π * for which the condition (1) is satisfied [1, 3]:

$$\pi^* = \arg\min_{\pi \in S_N} L(\pi),$$

where S_n denotes the set of all permutations of n vertices. This entry emphasizes that the solution to the problem involves finding a minimum on a set of discrete combinations, the size of which rapidly increases with increasing n.

3 Self-Organizing Kohonen Maps

3.1 The Basic Idea of the Method

Kohonen Self-Organizing Maps (SOMs) are among the most well-known unsupervised learning models [4]. Unlike neural networks, which are trained on labeled data to solve classification or regression problems, SOMs are designed to reveal the internal structure of the input set, cluster objects, and construct a topologically ordered representation of the data. The basic idea of the method is that a set of neurons, organized in a regular structure, gradually adapts to the distribution of the input vectors, thereby reflecting the geometry of the feature space under consideration.

Architecturally, a self-organizing map is a set of neurons, each of which is associated with a weight vector of the same dimension as the input data, [3],

[4]. If the input vector is of the form $x \in R^L$, then the weight vector of the j-th neuron is written as

$$W_j = \left(w_{j1}, w_{j2}, \ldots, w_{jL}\right) \in R^L, j = 1, 2, \ldots, M,$$

where M is the number of neurons in the map. Neurons can be located on a one-dimensional or two-dimensional lattice. In visualization problems, a two-dimensional map is often used, but for the traveling salesman problem, a one-dimensional ring topology is especially important, since the sought route is also a closed cycle [3]. The general idea of a self-organizing map as a topological representation of the input data is shown in Figure 1, [4], [5]. This example illustrates that SOMs do not simply group objects, but preserve the proximity relationships between them on a regular map.

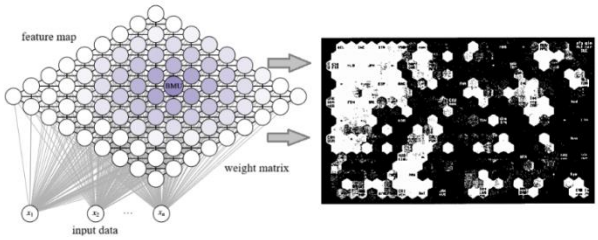


Fig. 1: Example of a Self-Organizing Map as a topological representation of a data structure.
Source: created by the authors

At each step t of network training, the input vector $x(t)$ is presented, after which the winning neuron, or BMU (Best Matching Unit), is determined. This neuron corresponds to the neuron whose weight vector is closest to the current input using the selected metric [3]. Using Euclidean distance, the winner's index e is defined by the expression:

$$e(t) = \arg\min_{1 \le j \le M} \left\| x(t) - W_j(t) \right\|. \quad (2)$$

This step associates the input object with the closest neuron and sets the center of subsequent map adaptation. A schematic representation of the Kohonen layer is shown in Figure 2.

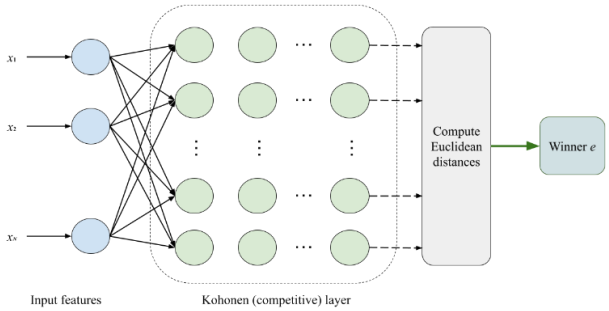


Fig. 2: Schematic diagram of the Kohonen layer and the selection of the winning neuron
Source: created by the authors

In the diagram, the input vector is mapped to a set of neurons, from which a winner is selected, after which the map weights are adjusted. SOM training is not limited to updating only the winning neuron. The weights of neurons located near the winner in the map topology are also updated. In general, the weight update rule is written as follows [4], [6]:

$$W_j(t+1) = W_j(t) + \alpha(t) h_{ej}(t) \left(x(t) - W_j(t)\right), \quad (3)$$

where $\alpha(t)$ is the learning rate at step t, $h_{ej}(t)$ is the neighborhood function between the winner neuron e and neuron j, and $x(t) - W_j(t)$ determines the direction of weight vector adjustment. This rule allows the map to be reconstructed, not as a set of independent points, but as a consistent topological structure.

The neighborhood function determines how strongly neuron j should be influenced by the winner e. It is often given in Gaussian form [4], [7], [8]:

$$h_{ej}(t) = \exp\left(-\frac{d_{ej}^2}{2\eta^2(t)}\right), \quad (4)$$

where d_{ej} is the distance between neurons e and j in the map topology, and $\eta(t)$ is the current neighborhood radius. The smaller the topological distance between a neuron and the winner, the more its weight vector is adjusted. As training progresses, the neighborhood radius decreases, and global map restructuring gradually gives way to local fine-tuning [4], [7], [8]. For a ring topology, the distance between neurons must take into account the closed nature of the map. If neurons are numbered from 0 to $M-1$, then the topological distance between the winner e and neuron j can be conveniently calculated as follows

$$d_{ej} = \min(|e - j|, M - |e - j|). \quad (5)$$

This formula means that the first and last neurons are considered neighbors, and the map itself forms a closed loop. It is this property that makes the ring SOM a natural model for constructing a Hamiltonian cycle in the TSP [3], [9].

To establish formal convergence of the ring-topology SOM within structural bounds, we consider the update dynamics within the framework of stochastic approximation and absorbing Markov chains [4]. Let:

$$W(t) = \left(W_0(t), W_1(t), \ldots, W_{M-1}(t)\right)^T$$

be the state vector of M weights arranged in a closed cycle.

In the deterministic continuous-time limit, following the stochastic-approximation treatment of the SOM in [4], the adaptation rule (3) can be described as a system of ordinary differential equations in which each weight $W_j(t)$ evolves toward a neighborhood-weighted local average of the input vectors for which the corresponding neuron is selected as the winner. The winner is determined by condition (2), and the local average is formed over the Voronoi region of the winning neuron, with the influence of neighboring units controlled by the neighborhood function. This continuous-time description preserves the essential convergence and ordering behavior of the discrete SOM; for detailed derivation and the associated stochastic-approximation conditions, see [4]. Under the conditions of order-preserving properties, the ring map constitutes a Markov process with an absorbing set of ordered configurations $\mathcal{S}^*$. Once $W(t) \in \mathcal{S}^*$, the probability of boundary-crossing transitions drops to zero, ensuring that the network converges almost surely to an ordered boundary trajectory.

It must be noted that a strict and comprehensive mathematical proof of global convergence for a ring topology mapped onto a two-dimensional Euclidean space remains an open and challenging problem. As recognized in foundational literature [4], multi-dimensional SOM models often constitute an "ill-posed" mathematical system, making an unconditional guarantee of global convergence unfeasible within standard analytical limits. Consequently, the application of the ring topology to the TSP in this paper is justified primarily by its robust topological preservation, structural efficiency, and well-documented geometric heuristic capabilities [4]. A more detailed and extended theoretical investigation of this convergence domain will be thoroughly addressed in future works.

The laws of change in the learning rate and the neighborhood radius play an important role. In the basic scheme, these parameters decrease over time, which allows for coarse global self-organization first, and then for local refinement. In this paper, we use the notation:

$$\alpha(t+1) = \lambda_\alpha \alpha(t), \quad \eta(t+1) = \lambda_\eta \eta(t),$$

where λ_α and λ_η are the decay coefficients of the learning rate and neighborhood radius, respectively. These coefficients are typically chosen close to unity so that the parameters decrease gradually [6], [9]. Too rapid a decay can lead to premature stabilization of the map, while too slow of a decay complicates final route tuning.

According to the Robbins–Monro stochastic approximation theorem, convergence to a stable equilibrium requires:

$$\sum_{t=0}^{\infty} \alpha(t) = \infty \text{ and } \sum_{t=0}^{\infty} \alpha^2(t) < \infty.$$

While a strict geometric decay sequence yields a finite sum, in practical computational implementations with T iterations, setting the decay coefficients λ_α, λ_η close to unity satisfies quasi-Robbins-Monro behavior over the operational horizon [0, T], [4]. To evaluate the Lyapunov stability of the weight trajectories across the entire set of input points $X = \{x_1, x_2, \dots, x_N\}$, we define the energy (distortion) functional over the ring as:

$$E\big(W(t)\big) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=0}^{M-1} h_{b(i),j}(t)\|x_i - W_j(t)\|^2$$

Here, $b(i) = \arg\min_{1\le j\le M} \| x_i - W_j \|$ denotes the index of the winning neuron corresponding to a specific input point x_i. This notation directly extends the single-step neighborhood function $h_{ej}(t)$ to cover the spatial relation between the winning node $b(i)$ of city x_i and every node j on the map.

The expectation of the Lyapunov candidate derivative satisfies:

$$\mathbb{E}\big[\Delta E\big(W(t)\big)\big] = -\alpha(t)\nabla_W E\big(W(t)\big) + \mathcal{O}\big(\alpha^2(t)\big)$$

Since the learning rate satisfies $\alpha(t) > 0$ and the neighborhood function satisfies $h_{b(i),j}(t) > 0$, the condition $\mathbb{E}\big[\Delta E\big(W(t)\big)\big] \le 0$ holds true for all iterations $t \le T$. This demonstrates that the map adaptation process is asymptotically stable in the sense of Lyapunov.

For TSP, not only the training scheme but also the initial placement of the weights is important. Random initialization can lead to instability of the result and the emergence of neurons that weakly contribute to the representation of the input points. Therefore, for the Euclidean setting, it is convenient to use ring initialization.

Suppose $c = (c_x, c_y)$ is the center of the set of input points, R is the radius of the initial ring, and M is the number of neurons. The initial weights can be specified by the formulas

$$W_j(0) = \begin{pmatrix} c_x + R\cos 2\pi j/M \\ c_y + R\sin 2\pi j/M \end{pmatrix},$$

$$j = 0, 1, \ldots, M - 1. \quad (6)$$

Such initialization aligns the map structure in advance with the cyclic nature of the route and reduces the risk of forming inactive map sections. The problem of so-called "dead" neurons arises when some neurons rarely become winners and participate weakly in map adaptation. This is undesirable for the TSP, since insufficiently active neurons worsen the distribution of support points along the route. The use of a neighborhood function, gradual decay of $\alpha(t)$ and $\eta(t)$, and ring initialization helps make training more robust and maintain consistent movement of the entire neuronal structure [2], [6], [7].

From a theoretical point of view, the key advantage of SOM is its ability to perform a topological mapping of the input space onto a lower-dimensional structure [2], [4]. Nearby input features tend to be mapped to nearby neurons of the map, while distant features tend to be mapped to more distant regions. This property is especially important for the TSP: a set of points on a plane must be transformed into an ordered sequence, along which a route is then constructed. Due to this, SOM can be viewed as a geometrically interpretable method for approximately solving the TSP [2], [3]. However, the Kohonen model has a number of limitations. The quality of the result depends on the initial arrangement of weights, the number of neurons, the learning rate, the neighborhood radius, the attenuation coefficients, and the number of iterations. In addition, learning is stochastic, so different runs with the same parameters may yield slightly different routes. Finally, as the number of points and neurons increases, the computational cost of training increases, since distances must be calculated and weights updated at each iteration. This is why in the following SOM it is considered in conjunction with the hyperparameter selection procedure and GPU acceleration.

3.2 Application of Self-Organizing Maps to the Traveling Salesman Problem

The application of Self-Organizing Maps to the traveling salesman problem is based on a transition from a direct combinatorial search over permutations to a process of geometric adaptation of a ring neural structure [2], [6], [9]. In the classical TSP formulation, the order of traversal points must be determined. In the SOM approach, instead of directly enumerating routes, a closed ring of neurons is constructed, which gradually adapts to the location of the problem vertices. After training is complete, the order of neurons on the ring is used to reconstruct the order of traversal of cities. The schematic relationship between the Kohonen map and the traveling salesman problem is shown in Figure 3, [2].

Problem vertices are matched with neurons in the map, and the topological structure of the network is used to reconstruct the traversal order.

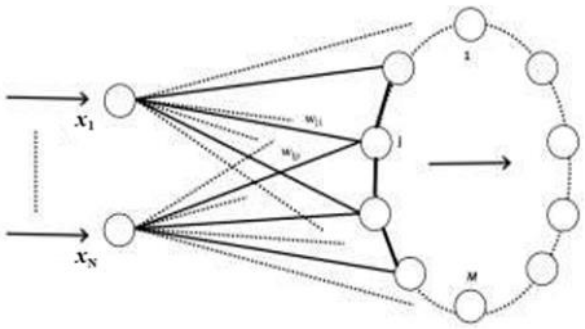


Fig. 3: SOM for the traveling salesman problem
Source: created by the authors

Suppose a set of points
$X = \{x_1, x_2, \ldots\ldots, x_N\}, x_i \in R^2$,
corresponds to cities or technological points.

The Kohonen map contains M neurons with weight vectors $W_j \in R^2$, where $M \geq N$ is usually chosen. Using a larger number of neurons compared to the number of cities allows the ring to more flexibly adapt to the geometry of the point set and reduces the risk of too rough an approximation of the route [6], [9]. At the initial stage, neurons are arranged in the form of a closed contour. For the Euclidean problem, ring initialization around the center of the point set (6) is often used. The training process follows the general scheme of the SOM, but receives a special interpretation for the TSP. At each iteration, an input point x_i is selected, after which the winning neuron closest to it is determined (2). Then the weights of all neurons are adjusted according to rule (3).

The winning neuron can be considered the current reference point of the route for the selected city, and its neighboring neurons can be considered a local fragment of the future traversal. Therefore, updating not only the winner but also its neighborhood helps maintain the smoothness and consistency of the ring. For a ring map, the neighborhood function must take into account the closed nature of the structure. The topological distance between the winning neuron and neuron j is defined by formula (5). After training is complete, the neuron positions must be converted into a discrete route. For each point x_i, the closest neuron is determined:

$$b(i) = \arg\min_{1 \le j \le M} \|x_i - W_j\| .$$

Then the cities are ordered by increasing index $b(i)$ along the ring. If multiple points correspond to the same neuron, their relative order can be determined by an additional local rule, such as distance to neighboring points or local post-processing. The result is a permutation of cities (1), which is used to calculate the route length:

$$L(\pi) = \sum_{k=1}^{N} \|x_{\pi_k} + 1 - x_{\pi_k}\| + \|x_{\pi_1} - x_{\pi_N}\|$$

The algorithm for applying the SOM to the traveling salesman problem can be represented as the following steps:

(a) preparing the point coordinates;

(b) choosing the number of neurons and initial hyperparameters;

(c) initializing the ring;

(d) repeatedly presenting the network points;

(e) updating the weights based on the neighborhood function (4);

(f) reconstructing the city order based on the nearest neurons;

(g) calculating the route length.

This scheme does not require explicit enumeration of permutations and therefore differs from classical combinatorial algorithms.

However, SOM does not guarantee finding a global optimum. The final route depends on the initial configuration, the order of point presentation, the number of neurons, the learning rate, the neighborhood radius, and their decay laws [6], [9], [10]. Furthermore, as the problem dimension increases, the computational cost of training increases: at each iteration, the winning neuron must be found and the map weights updated. Therefore, the practical application of SOM to the TSP requires not only the selection of the model itself but also careful tuning of hyperparameters, statistical testing of the result's robustness, and an efficient software implementation. The limitations of the Kohonen map method are related to the lack of a strict guarantee of optimality and its sensitivity to training parameters. Therefore, further research requires consideration of Bayesian hyperparameter optimization and a GPU implementation of the algorithm.

3.3 Selecting SOM Hyperparameters

Applying self-organizing Kohonen maps to the traveling salesman problem requires selecting a set of hyperparameters that significantly affect the route quality and training time. These parameters include the number of neurons M, the initial learning rate α_0, the initial neighborhood radius η_0, the attenuation coefficients λ_α and λ_η, and the number of iterations T. An unsuccessful configuration can lead to too slow adaptation of the map, premature stabilization, the appearance of inactive neurons, or obtaining a route with a large length. Therefore, tuning the hyperparameters is not an auxiliary step, but an important part of applying SOM to the TSP [2], [10].

Suppose the hyperparameter vector is denoted by

$$\theta = (M, \alpha_0, \eta_0, \lambda_\alpha, \lambda_\eta, T).$$

For each configuration θ, the SOM training is launched, after which a route is constructed and its length calculated. If we denote the result of one run by $D_{loc}(\theta)$, then the hyperparameter selection problem can be written as the problem of minimizing the objective function

$$\theta^* = \arg\min_{\theta \in \Theta} f(\theta) ,$$

where Θ is the space of admissible hyperparameter values, and $f(\theta)$ is the configuration quality indicator. In the simplest case, $f(\theta)$ can coincide with the route length, but for comparing different runs, it is more convenient to use the relative deviation from the best available value. Let D_b be the best known or best-found route length for the problem under consideration. Then, the relative deviation of the route obtained by the SOM with parameters θ can be defined as

$$\delta(\theta) = \frac{|D_{loc}(\theta) - D_b|}{D_b}.$$

The smaller the value of $\delta(\theta)$, the closer the found route is to the reference value. This form of the objective function is convenient because it allows comparing results for problems of different dimensions and interpreting the quality as a percentage. Since SOM training is stochastic, a single run does not always reliably characterize the quality of the selected hyperparameters. For the same configuration θ, different runs may yield different route lengths. Therefore, it is advisable to evaluate the configuration over a series of S runs. If $D_{loc}^s(\theta)$ is the route length in the s-th run, then the relative deviation for this run is δ_s:

$$\delta_s(\theta) = \frac{|D_{loc}^s(\theta) - D_b|}{D_b}, s = 1, 2, \ldots, S.$$

The final objective function can be defined as the mean deviation $\overline{\delta}$

$$\overline{\delta}(\theta) = \frac{1}{S} \sum_{s=1}^{S} \delta_s(\theta) \to min.$$

This approach takes into account not only the quality of the best individual route, but also the algorithm's robustness to randomness. Direct enumeration of hyperparameters over the grid quickly becomes ineffective. If q possible values are specified for each of the p parameters, then the total number of combinations is $N_{grid} = q^p$.

As the number of parameters increases, the grid size grows exponentially, and each SOM run requires separate model training. Random search partially reduces the grid stiffness but does not use information about previously tested configurations. Therefore, with an expensive objective function, a more rational method is a sequential method that takes into account the accumulated results.

3.4 Bayesian Optimization

Bayesian optimization is a method for global optimization of functions whose values are expensive to compute, [9]. Its idea is to construct a probabilistic surrogate model of the objective function. After several initial trials, the algorithm uses the already obtained pairs:

$$\mathcal{D}t = \{(\theta_1, f(\theta_1)), (\theta_2, f(\theta_2)), \dots, (\theta_t, f(\theta_t))\}$$

to select the next configuration θ_{t+1}. Thus, the optimization is performed sequentially: each new run is selected taking into account the results of previous runs. The choice of the next point is carried out using an acquisition function, which specifies a trade-off between exploring new areas of the parameter space and refining already promising areas. In general, this step can be written as:

$$\theta_{t+1} = \arg\max_{\theta \in \Theta} a_t(\theta),$$

where $a_t(\theta)$ is the acquisition function at step t. If the algorithm explores new regions too aggressively, it may waste computational budget on weak configurations. If it concentrates too early around the current best solution, there is a risk of missing a more successful region of the parameter space. Bayesian optimization is valuable precisely because it formalizes this balance. Modern software tools for selecting hyperparameters often use variants of sequential model searches. One such tool is Optuna, designed for automated hyperparameter optimization and using effective strategies for selecting new trials, [11], [12], [13]. In the context of SOM for the TSP, each trial corresponds to a single configuration θ: the number of neurons, learning rate, neighborhood radius, attenuation coefficients, and the number of iterations is specified, after which the model is trained and returns the objective function value. For self-organizing maps, it is especially important to correctly define the search space. Too narrow ranges may not include good configurations, while too wide ones lead to inefficient use of the optimization budget. Therefore, the bounds for M, $\alpha_0, \eta_0, \lambda_\alpha, \lambda_\eta$, and T should be chosen taking into account the theoretical properties of SOM and preliminary experiments [10], [14]. For example, the number of neurons should be sufficient to provide a flexible route representation, but excessively large M increases the computational cost of training. Similarly, the damping of parameters should provide a gradual transition from global adaptation to local refinement. An important advantage of Bayesian optimization is that it allows SOM tuning to be viewed as a reproducible computational procedure rather than as a manual parameter selection.

The problem of selecting SOM hyperparameters is multidimensional and multi-extremal: the objective function $f(\theta)$ can have several local minima, and direct enumeration or random search becomes ineffective due to the high cost of one calculation (the full cycle of training the Kohonen map). To solve such problems, a multi-step dimensionality reduction scheme is used, which reduces the multidimensional optimization to a sequence of one-dimensional subproblems [15].

The hyperparameter vector is denoted as

$$\theta = (\omega_1, \dots, \omega_6), \quad \text{where}$$
$$\omega_1 = M, \omega_2 = \alpha_0, \omega_3 = \eta_0, \omega_4 = \lambda_\alpha,$$
$$\omega_5 = \lambda_\eta, \omega_6 = T.$$

The domain is defined by the hyperparallelepiped:

$$\Theta = [A_1, B_1] \times \dots \times [A_6, B_6], \quad A_i, B_i - \text{real}.$$

Then the classical reduction is written as

$$\min_{\theta \in \Theta} f(\theta) =$$
$$= \min_{\omega_1 \in [A_1, B_1]} \min_{\omega_2 \in [A_2, B_2]} \dots \min_{\omega_6 \in [A_6, B_6]} f(\omega_1, \dots, \omega_6)$$

Recursive functions φ_i are introduced, which represent the search result for all parameters starting from the i-th:

$$\varphi_6(\omega_1, \dots, \omega_6) = f(\omega_1, \dots, \omega_6),$$
$$\varphi_i(\omega_1, \dots, \omega_i) =$$
$$= \min_{\omega_{i+1} \in [A_{i+1}, B_{i+1}]} \varphi_{i+1}(\omega_1, \dots, \omega_i, \omega_{i+1}),$$
$$i = 5, \dots, 1.$$

Then the original problem is equivalent to $\min_{\omega_1} \varphi_1(\omega_1)$.

This formulation of the classical reduction and the recursive functions φ_i follows the adaptive nested optimization scheme proposed in paper [15].

The efficiency of the method is increased due to parallel computing.

Denote:

$$\tilde{\omega}_j = \begin{cases} \omega_j & \text{when } i = 1, \\ \omega_1 & \text{for } j = 6 \text{ and } i = 1 \\ \omega_j^{rand} \sim U[A_j, B_j] & \text{in other cases.} \end{cases}$$

At each recursion i, a uniform grid of $\mathrm{Q_{points}}$ of values ω_i is generated; for each configuration, the estimate ψ_i

$$\psi^i(u_{i-1}, \omega_i) = f(\omega_1, \dots, \omega_i, \tilde{\omega}_{i+1}, \dots, \tilde{\omega}_6)$$

is calculated in parallel, after which the best value ω_i^* is selected and the search area is narrowed:

$$[A_i^{k+1}, B_i^{k+1}] = \\ = [\max(A_i, \omega_i^* - \gamma\Delta_i), \min(B_i, \omega_i^* + \gamma\Delta_i)], \\ \Delta_i = B_i - A_i,$$

where γ is the narrowing coefficient. The process is repeated iteratively until the stopping criterion is reached.

To rigorously formalize the computational limits of the multi-step dimensionality reduction framework, we examine its algorithmic complexity, memory upper bounds, and operational scalability. Let $p = 6$ be the spatial dimension of the hyperparameter vector $\theta = (\omega_1, \dots, \omega_6) \in \Theta$, Q_{points} be the number of uniform discretization nodes per recursion level, and S be the number of independent stochastic evaluation runs per trial.

Unlike a full grid search whose time complexity scales exponentially as $\mathcal{O}\left(Q_{\text{points}}^p\right)$, the sequential-recursive reduction scheme decomposes the search space into p ordered one-dimensional subproblems. The total number of model evaluations is strictly bounded by:

$$N_{\text{eval}} = \sum_{i=1}^{p} Q_{\text{points}} = p \cdot Q_{\text{points}}$$

Thus, the sequential time complexity is linear with respect to the parameter dimension, $\mathcal{O}(p \cdot Q_{\text{points}} \cdot S \cdot t_e)$, where t_e represents the execution time of a single SOM training cycle. The domain bound contraction at level i follows the recurrence equation:

$$\Delta_i^{k+1} = 2\gamma\Delta_i^k = (2\gamma)^k\Delta_i^0$$

where $\Delta_i^0 = B_i - A_i$. For the narrowing coefficient $\gamma = 0.2$, the search volume contracts strictly geometrically by a factor of $0.4^k < 1$ per iteration step k, guaranteeing a linear convergence rate in space volume reduction.

Given the parallelization vector $\rho = (\rho_1, \dots, \rho_p)$ specifying the worker thread allocation per recursion level i, the theoretical speedup ratio $U(\rho)$ is derived as:

$$U(\rho) = \frac{T_{\text{sequential}}}{T_{\text{parallel}}} = \frac{\sum_{i=1}^{p} Q_{\text{points}} \cdot S \cdot t_e}{\sum_{i=1}^{p} \left\lceil \frac{Q_{\text{points}}}{\rho_i} \right\rceil \cdot S \cdot t_e}$$

Under ideal hardware capacity where $\rho_i = Q_{\text{points}}$, the speedup reaches its absolute upper bound $U_{\max} = Q_{\text{points}}$, compressing execution time to $\mathcal{O}(p \cdot S \cdot t_e)$. Furthermore, the spatial memory complexity is bounded by $\mathcal{O}\left(\max_i \rho_i \cdot M\right)$, where M is the number of neurons, ensuring negligible Random Access Memory (RAM)/ Video Random Access Memory (VRAM) footprint overhead during execution on multi-core CPU and GPU systems.

The proposed adaptive multi-step dimension reduction scheme with an explicit parallelization vector allows us to effectively solve the problem of tuning SOM hyperparameters, combining global search, resistance to multiextremality, and scalability on multiprocessor systems.

Bayesian optimization based on sequential trial selection was used to select the parameters. A new SOM configuration was defined for each trial, after which the map was trained, and the objective function value was calculated. This approach differs from exhaustive searches in that subsequent trials are selected based on the results already obtained. Therefore, the computational budget is spent primarily on regions of the parameter space where a lower objective function value is expected. For the Asia_SOM problem set with 100, 200, and 500 vertices, 50, 75, and 100 trials were used, respectively. Figure 4 shows the routes for the Asia_SOM problem set using the selected hyperparameters.

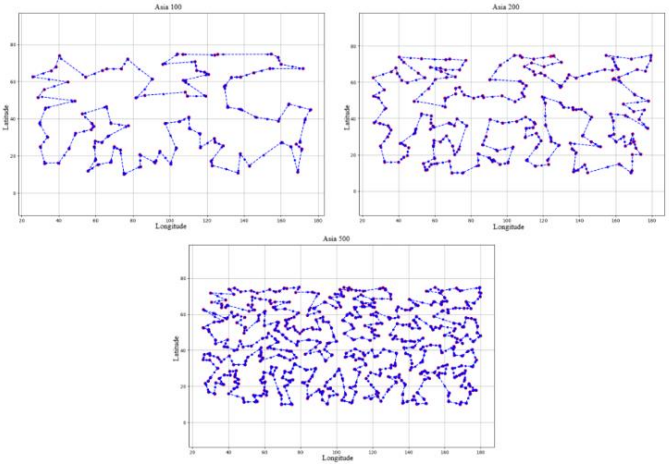


Fig. 4: SOM routes for problems of 100, 200, and 500 dimensions after Bayesian optimization

Source: created by the authors

Multidimensional multi-extremal optimization based on an adaptive multi-step dimensionality reduction scheme was used to select the SOM hyperparameters. Unlike Bayesian optimization, where trials are selected sequentially based on a surrogate model, the multi-step scheme performs a deterministic search at each recursion followed by narrowing the search space. This allows for fully parallelized computations within the recursion and yields reproducible results. Parameter search was performed on the same Asia_SOM datasets with 100, 200, and 500 cities. For each level, Q_{points} = 12 grid points were used, and adaptive narrowing of the intervals with a factor of $\gamma = 0.2$ was performed after each level, except for the attenuation parameters, for which the bounds remained unchanged.

The best configurations found by the multi-step scheme are presented in Table 1.

Table 1. The best configurations

Problems	M	α_0	λ_α	λ_η	η_0	T	$\delta(\theta)$
Asia 100	398	0.6988	0.999967	0.999900	6.0951	22485	0.025
Asia 200	527	0.6782	0.999999	0.999909	2.1818	25000	0.019
Asia 500	1150	0.9326	0.999965	0.999900	7.5977	49916	0.004

Source: created by the authors

Figure 5 shows the routes for the Asia_SOM problem set using the selected hyperparameters.

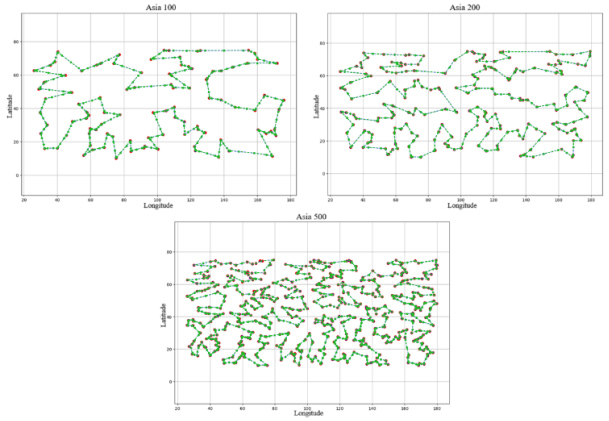


Fig. 5: SOM routes for problems of dimensions 100, 200, and 500 after applying a multi-step dimensionality reduction scheme
Source: created by the authors

For visual comparison, the routes obtained by both approaches are combined in Figure 6. The blue line denotes the route after Bayesian optimization (Figure 4) and the green line denotes the route after the multi-step dimensionality reduction scheme (Figure 5). The main changes for the 100-dimensional case (Figure 6a) are localized within the coordinate ranges [longitude 80–100, latitude 10–20] and [longitude 140–170, latitude 10–30], while for problems of dimensions 200 and 500 (Figure 6b and Figure 6c) the route changes cover almost the entire considered coordinate domain.

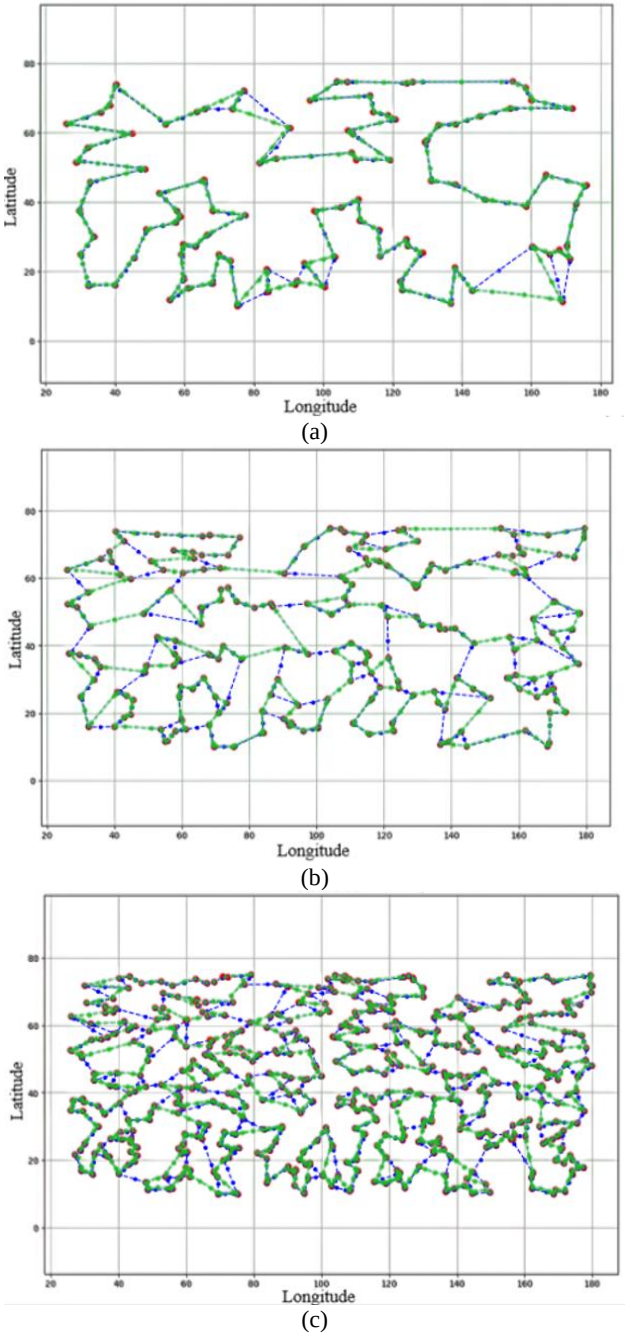


Fig. 6: Combined view of SOM routes from Figure 4 (Bayesian optimization) and Figure 5 (multi-step dimensionality reduction scheme) for 100 (a), 200 (b), and 500 (c) dimensions
Source: created by the authors

To accelerate the training of a self-organizing map for solving the traveling salesman problem, this paper utilizes CUDA parallel computing

technology, implemented using the PyTorch library. The main computational load in the SOM algorithm falls on large-scale, similar operations: calculating Euclidean distances from the current point to all neurons, finding the winning neuron (2), calculating the neighborhood function (4), and then updating the weights (3). All these steps can be performed independently for each neuron, making them ideal for parallel processing. In the GPU implementation, all data (city coordinates and neuron weight vectors) are stored as tensors on the CUDA device, and subtraction, squaring, summation, exponentiation, and weight updating operations are performed as single vectorized array operations. The computational pipeline involves transferring data to the GPU, launching CUDA kernels to calculate distances and updates, and then returning the results to the CPU only after training is complete, minimizing data transfer overhead [8]. This approach eliminates explicit loops based on the number of neurons at the high-level code level and replaces them with tensor operations executed in parallel across thousands of GPU cores. The efficiency and scalability of this approach are supported by modern research [8] and official NVIDIA documentation [16]. In experiments, the use of GPU acceleration provided a significant performance boost compared to a sequential CPU implementation.

The plot in Figure 7 clearly demonstrates that, as the dimensionality increases, the difference between CPU and GPU implementations increases.

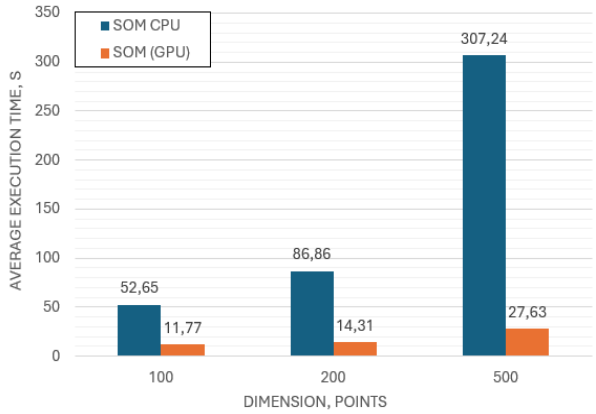


Fig. 7: The difference between CPU and GPU implementations increases
Source: created by the authors

Figure 8 presents the same results graphically. It is evident that for all three dimensions, the shortest route length in the available data was obtained using SOM.

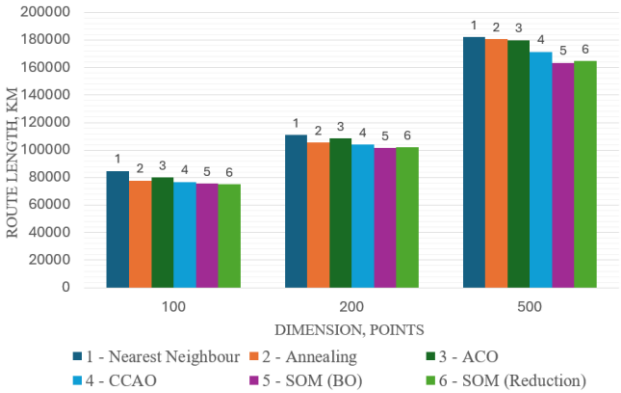


Fig. 8: Comparison of route lengths on the Asia_SOM demo datasets
Source: created by the authors

To evaluate the quality of SOM, the results are compared with other heuristic methods for solving the traveling salesman problem. Nearest Neighbor (NN), Ant Colony Optimization (ACO), simulated annealing, and the Consistent Adjustment of the Climatology to Actual Observations (CCAO) method are considered as alternatives [9]. Summary results for the Asia_SOM problem sets are presented in Table 2 and Table 3.

Table 2. Comparison of NN, ACO, and simulated annealing (route length in km)

Points	NN	ACO	Annealing
100	84733.5	80223.4	77771.4
200	111060.8	108482.3	105625.8
500	182087.9	179953.6	180552.7

Source: created by the authors

Table 3. Comparison of CCAO, SOM (BO), and SOM (Reduction) (route length in km)

Points	CCAO	SOM (BO)	SOM (Reduction)
100	76916.6	75667.3	75011.7
200	103951.0	101409.3	101929.2
500	171405.0	163523.0	164673.0

Source: created by the authors

4 Conclusion

Thus, the main objectives of this study were the following: CPU and GPU versions of the algorithm were developed, a Bayesian optimization methodology and a multi-step dimensionality reduction scheme for hyperparameter search were described, experiments were conducted on datasets, and route quality and execution time were compared. The obtained results confirm that Kohonen self-organizing maps can be considered an effective heuristic tool for solving the traveling salesman problem, especially when combined with

CUDA acceleration and automated parameter selection.

However, this study has several limitations. SOM does not guarantee finding a global optimum and remains sensitive to the initial map state, the order of point presentation, and the chosen hyperparameters. Therefore, for a more rigorous quality assessment, it is necessary to conduct a series of independent runs and analyze not only the best but also the average route length, the scatter of results, and the stability of found configurations. Furthermore, promising directions include integrating SOM with local route improvers, expanding the library of sample instances for the TSP (TSPLIB) test problem set, conducting a more detailed analysis of GPU memory consumption, and developing adaptive schemes for changing the learning parameters during the self-organization process.

Further work will improve the parameter selection methodology.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, the author(s) utilized artificial intelligence (AI) technologies solely for the purpose of linguistic refinement and grammatical correction. Following the application of these tools, the author(s) comprehensively reviewed, verified, and edited the resulting text. The author(s) maintain full accountability and responsibility for the accuracy, integrity, and final content of the publication.

References:

[1] M.R.Hossen, H.J.Koushik, M.F.Uddin, M.A.Rahman, M.A. Hye, Optimal algorithm of the traveling salesman problem for tourist local selection, *International Journal of Simulation and Multidisciplinary Design Optimization*, 2025, Vol. 16, Article 11, pp.1-16, https://doi.org/10.1051/smdo/2025011.

[2] S.Sarikyriakidis, K.Goulianas, A. I. Margaris, Using Self-Organizing Maps to Solve the Travelling Salesman Problem: A Review, *WSEAS Transactions on Systems*, 2023, pp.131-159, https://doi.org/10.37394/23202.2023.22.14.

[3] I.Urbanik, P.Gajewski, SatSOM: Saturation Self-Organizing Maps for Continual Learning, *Journal of Artificial Intelligence and Soft Computing Research*, Vol. 16, No 3, 2026, pp.293-310, https://doi.org/10.2478/jaiscr-2026-0015.

[4] T.Kohonen, *Self-Organizing Maps*, 3rd ed. Berlin: Springer, 2001, pp.1-501, https://doi.org/10.1007/978-3-642-56927-2.

[5] D.De Cicco, Y.Maruccia, M.Paolillo, S.Cavuoti, G.Longo, M.Brescia, C.Tortora, G.D'Angelo, Navigating AGN variability with self-organizing maps, *Astronomy & Astrophysics*, Vol. 699, 2025, Article 303, pp.1-15, https://doi.org/10.1051/0004-6361/202553866.

[6] H.Sharma, S.Kamat, V.Daroch, Self-Organizing Maps and Optimality Gaps for Solving the Traveling Salesman Problem, *Term Paper for course DBA5103 at NUS Business School*. 2024, pp.1-9, [Online]. https://shlok-kamat.github.io/assets/DBA5103_Paper.pdf (Accessed Date: January 15, 2026).

[7] M.Nilashi, R.A.Abumalloh, S.Alyami, A.Alghamdi, M.Alrizq, Parkinson's Disease Diagnosis Using Laplacian Score, Gaussian Process Regression and Self-Organizing Maps, *Brain Sciences*, Vol. 13, no. 4, 2023, Article 543,pp.1-17, https://doi.org/10.3390/brainsci13040543.

[8] J.Kaminski, O.Koch, χ-SOM: Emergent self-organising Maps for large chemical space visualization, *ChemRxiv*. 2026, pp.1-24, https://doi.org/10.17879/54cfz-bqf24.

[9] D.Maquin, The Traveling Salesman Problem–TSP2024 Matlab App, *HAL open science*. 2024, hal-04909391, pp.1-112, https://hal.science/hal-04909391.

[10] H.N.Nguyen, T.H.Hoang, V.L.Cao, N.N.Tran, Tuning Hyperparameters of Self-Organizing Maps in combination with K-Nearest Neighbors for IoT Malware Detection, *Journal of Science and Technique*, Vol. 12, No. 1, 2023, pp. 1–15, https://doi.org/10.56651/lqdtu.jst.v12.n1.654.ict.

[11] S.Watanabe, F.Hutter, c-TPE: Tree-Structured Parzen Estimator with Inequality Constraints for Expensive Hyperparameter Optimization, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence* (IJCAI-23), 2023, pp.1-9, https://doi.org/10.24963/ijcai.2023/486.

[12] O.Teke, Hyperparameter Optimization with Optuna in Wind Energy Prediction Models Towards to More Accuracy, *7th International Conference on Scientific and Innovative Studies*, 2026, pp.1-5.

[13] T.Vaiyapuri, An Optuna-Based Metaheuristic Optimization Framework for Biomedical

Image Analysis, *Engineering, Technology & Applied Science Research*, Vol. 15, No. 4, 2025, pp.1-8, https://doi.org/10.48084/etasr.11234.

[14] J.Guérin et al., Hyperparameter Tuning for Self-Organizing Maps, *IEEE International Conference on AI, Blockchain, and IoT*. 2024, pp.228-235, https://doi.org/10.1109/aixset62544.2024.00044.

[15] V.Gergel, V.Grishagin, R.Israfilov, Multiextremal Optimization in Feasible Regions with Computable Boundaries on the Base of the Adaptive Nested Scheme, *Lecture Notes in Computer Science, Springe*r, Vol. 11890, 2020, pp. 112–125, https://doi.org/10.1007/978-3-030-40616-5_9.

[16] NVIDIA Corporation. CUDA C++ Programming Guide [Term paper], [Online]. https://docs.nvidia.com/cuda/cuda-c-programming-guide (Accessed Date: January 15, 2026).

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

- Filipp Maksimov contributed the theoretical results and has organized and executed the experiments,
- Vera V. Bukhvalova: conceptually formulating the initial research problem and the technical control of the mathematical style of the paper,
- Irina G. Burova provided consultations on optimization algorithms.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0 https://creativecommons.org/licenses/by/4.0/deed.en_US