Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem

Anait Karapetyan [1], Eugene Fedorov [1], Oleksii Smirnov [2]

[1] *Cherkasy State Technological University, 460 Shevchenko ave, Cherkasy, 18006, Ukraine*
[2] *Central Ukrainian National Technical University, 8 Universytetskyi ave., Kropyvnytskyi, 25006, Ukraine*

Abstract

The problem under review is improving the efficiency of the solution search for the Traveling Salesman Problem. We propose a shortest path neural network training method that executes a calibration of winner neuron weights and his neighbours based on the calculation of the centre of gravity, which allows paralleling the learning process and also utilizes the effective dynamic width of the topological neighbourhood (based on simulated annealing) for the calculation of topological neighbourhood function. A proposed modification to the single solution human-based metaheuristic allows for potential integer solutions and utilizes a 2-opt permutation in local search neighbourhood creation and a 4-opt permutation in the case of global search. This will increase efficiency in the search for the optimal path by decreasing the computational complexity and increasing the accuracy of solutions.

Keywords 1

Travelling salesman problem, single solution human-based metaheuristics, self-organizing feature map, simulated annealing, dynamic effective width of the topological neighbourhood, rule of the centre of gravity calculation, 2-opt and 4-opt permutations.

1. Introduction

A harsh socio-economic environment in Ukraine is a consequence of the war that Russia started and the direct cause of the increase in the destruction of civil infrastructure and, as a result, the loss of life and damage to property. Human and material recourse shortages are the cause for the integration of new information technologies that make it possible to increase the efficiency of rescue units' performance. One of these tasks that requires using new models and methods is optimizing the time for the rescue units to travel to the location of destruction.

There is an urgent need to develop methods aimed at solving combinatorial optimization problems used in intelligent computational systems.

Optimization methods that find the exact solution have high computational complexity. Random search methods do not guarantee convergence. Optimization methods that find an approximate solution through local search have a high probability of hitting a local extremum. In this regard, there is a problem of insufficient efficiency of optimization methods, which needs to be addressed.

Metaheuristics (or modern heuristics) and artificial neural networks are used to speed up finding a quasi-optimal solution to optimization problems and reduce the probability of hitting a local extremum.

The object of inquiry. The process of finding solutions to optimization problems.

MoMLeT+DS 2022: 4th International Workshop on Modern Machine Learning Technologies and Data Science, November, 25-26, 2022, Lviv-Leiden, Ukraine
EMAIL: a.r.karapetyan@gmail.com (A. Karapetyan); fedorovee75@ukr.net (E. Fedorov); Dr.smirnovoa@gmail.com (O. Smirnov)

ORCID: 0000-0002-7412-3252 (A. Karapetyan); 0000-0003-3841-7373 (E. Fedorov); 0000-0001-9543-874X (O. Smirnov)

The subject of inquiry. Methods for finding a quasi-optimal solution based on metaheuristics and artificial neural networks.

The paper aims to increase the efficiency of a quasi-optimal solution search to the travelling salesman problem using neural networks and metaheuristic methods.

To achieve this goal, it is necessary to solve the following tasks:

Create an optimization method on an artificial neural network.

Create an optimization method based on single-solution human-based metaheuristics.

Conduct a quantitative study of the proposed optimization method.

2. Problem statement

The problem of increasing the efficiency of solution search to the travelling salesman problem based on single-solution human-based metaheuristics is presented as the problem of finding local and global search operators A^{local} and A^{global} respectively, its application provides a search for such a solution, under which $F(x^*) \to \min$ and $T \to \min$.

The problem of increasing the efficiency of searching for a solution to the travelling salesman problem based on a neural network is reduced to the problem of finding such a vector of parameters W that satisfies the criterion for the adequacy of the quasi-optimal solution search model $F = \frac{1}{P}\sum_{\mu=1}^{P}(f(x_\mu, W) - d_\mu)^2 \to \min_W$ i.e. they result in the minimum mean square error (the difference between the model output and the desired output), where P is the power of the test set x_μ.– μ-th - training input value, dd_μ.– μ-th - training output value.

3. Literature Review

The advantage of single-solution human-based metaheuristics is the ease of implementation, low computational complexity (no need to calculate a population of potential solutions), a small number of adjustable parameters and operators, and no requirement to model the behaviour of objects of different nature [1-3].

Existing metaheuristics suffer from one or more of the following disadvantages:

- there is only an abstract description of the method, or the description of the method is focused on solving only a specific problem [4-5];
- the convergence of the method is not guaranteed [7-6];
- the influence of the iteration number on the process of finding a solution is not taken into account [8-9];
- the procedure for determining the values of parameters is not automated; [11-10]
- there is no possibility of using non-binary potential solutions [13-12];
- insufficient accuracy of the method [14-15] ;
- there is no possibility of solving problems of conditional optimization [16-17];

This raises the problem of constructing efficient metaheuristic optimization methods.

Existing neural networks suffer from one or more of the following disadvantages:

- high probability of the learning and adaptation method hitting the local extremum[18-19];
- difficulty in determining the structure of the network since there are no algorithms for calculating the number of layers and neurons in each layer for specific applications [20-21];
- inaccessibility for human understanding of the knowledge accumulated by the network (it is impossible to represent the relationship between output and output in the form of rules) since they are distributed among all elements of the neural network and are presented in the form of its weight coefficients [22];
- the difficulty of forming a representative sample [23];

In this regard, the problem arises of constructing effective neural network optimization methods.

4. Self-organizing feature map

A one-dimensional self-organizing feature map (SOFM) [24-25] is a single-layer non-recurrent network. The neurons of the input layer correspond to the coordinates of the vertices. Before running the network, the neurons of the output layer correspond to the points of the multidimensional sphere (if the vertices are given on the plane, then these neurons correspond to the points of the ring), and after running the network, they correspond to the points in the obtained suboptimal path. In contrast to the classical learning method, in this paper, the weights of the winning neuron and its neighbours are adjusted not based on the Kohonen rule but based on calculation of the centre of gravity, which makes it possible to parallelize the learning process. [26-27]. In addition, unlike the classical learning method, this paper uses the effective dynamic width of the topological neighbourhood (based on simulated annealing) to calculate the topological neighbourhood function, which allows to explore the entire search space at the initial iterations and make the search directed at the final iterations, which ensures high search accuracy of this neural network.

The training method consists of the following steps:

1. Training iteration number $n = 1$, initialization of all network weights $w_{ij}(n)$, $i \in \overline{1,M}$,$j \in \overline{1,N^h}$ so that the point with coordinates $(w_{i1}(n),\dots,w_{iM}(n))$ is located on the is on the surface of an M -dimensional sphere of radius $R \in (0,1]$, where M – the number of input layer neurons (or the length of the vertex coordinate vector), N^h -the number of neurons in the output layer, usually $M \leq N^h \leq 3M$.

A set of vertex coordinates is specified $\{\boldsymbol{x}_\mu | \boldsymbol{x}_\mu \in R^M\}$, $\mu \in \overline{1,M}$, where $\boldsymbol{x}_\mu$ – μ - vertex coordinate vector.

Initial shortest distance $d(n)$ =0.

The maximum number of iterations N is set.

2. Calculation of the distance to all neurons of the network

The distance $d_{\mu j}$ from the μ vertex to each j neuron is determined by the formula: $d_{\mu j} = \sum_{i=1}^{M}(x_{\mu i} - w_{ij}(n))^2$, $\mu \in \overline{1,M}$, $j \in \overline{1,N^h}$

3. Calculation of the shortest distance and selection of the neuron with the shortest distance.

The smallest distance is calculated

$d_\mu = \min\limits_j d_{\mu j}$, $\mu \in \overline{1,M}$, $j \in \overline{1,N^{(1)}}$

and the winning neuron is selected j_μ^* for which the distance $d_{\mu j}$ is the shortest.

$j_\mu^* = arg\min\limits_j d_{\mu j}$, $\mu \in \overline{1,M}$, $j \in \overline{1,N^h}$.

4. Setting the weights of the winning neuron j_μ^* and its neighbours based on the calculation of the centre of mass of the j cluster.

$$w_{ij}(n+1) = \frac{\sum_{\mu=1}^{M} h_{j,j_\mu^*}(n) x_{\mu i}}{\sum_{\mu=1}^{M} h_{j,j_\mu^*}(n)},\ i \in \overline{1,M},\ j \in \overline{1,N^h},$$

where $h_{j,j^*}(n)$ - is the topological neighbourhood function, which is equal to 1 for $j = j^*$and decreases as the distance between the j and j^* neurons in the topological space increase. For example,

$$h_{j,j^*}(n) = \begin{cases} exp\left(-\dfrac{\rho^2(j,j^*)}{2\sigma^2(n)}\right), & \rho(j,j^*) < \sigma(n) \\ 0, & \rho(j,j^*) \geq \sigma(n) \end{cases}$$

$\rho(j,j^*) = min\{|j-j^*|, N^{(1)} - |j-j^*|\}$,

$\sigma(n)$ – the effective width of the topological neighbourhood (the "diameter" of the topological neighbourhood function) decreases with time. In this paper, it is calculated based on simulated annealing

$\sigma(n) = \sigma_0\, exp\left(-\frac{1}{T(n)}\right)$,

$T(n) = \beta^n T(0)$,

σ_0 – initial effective width of the topological neighbourhood,

$T(0)$ – start temperature,
β – cooling rate.
5. Checking the completion condition of training
$d(n+1) = \frac{1}{M}\sum_{\mu=1}^{M} d_\mu$,
If $n < N$, then $n = n + 1$, go to step 2.
6. Calculating the distance to all neurons in the network
$d_{\mu j} = \sum_{i=1}^{M}(x_{\mu i} - w_{ij}(n))^2$, $\mu \in \overline{1,M}$, $j \in \overline{1,N^h}$
7. Selecting the neuron with the shortest distance
$j_\mu^* = arg\min_j d_{\mu j}$, $\mu \in \overline{1,M}$, $j \in \overline{1,N^h}$
The result is a vector of top-rank pairs $((1,j_1^*),\dots,(M,j_M^*))$

5. Iterated local search

Iterated local search was proposed by Stutzle [28]. The algorithm consists of two phases - perturbation and local search. The use of a perturbation action makes it possible to avoid local optima in a local search. Too little perturbation makes the algorithm greedy. In this paper, the perturbation consists of the formation of a new solution by a permutation called "4-opt" (the current solution is randomly divided into 4 parts, which become in order 1,4,3,2), and the local search consists of the formation of a new solution by permutation called "2-opt" (the elements of the new solution, located between two randomly selected vertices, are rearranged in reverse order).

The method consists of the following steps:
1. Initialization
1.1. Setting the maximum number of iterations N_1; the maximum number of local search iterations N_2, where no new best solution is found; vertex vector lengths M.
1.2. An ordered set of vertices is specified $V = \{1,\dots,M\}$ and the edge weight matrix $[d_{ij}]$, $i,j \in \overline{1,M}$.
1.3. Specifying the Cost Function (Goal Function)
$F(x) = d_{x_M,x_1} + \sum_{i=1}^{M-1} d_{x_i,x_{i+1}} \to \min_x$,
where d_{x_M,x_1} – edge weight (x_i, x_{i+1}), $x_i, x_{i+1} \in V$,
x – vertex vector.
1.4. Set by randomly ordering the set V, the best vertex vector x^*.
1.5. Local search based on 2-opt
1.5.1. $m = 0$
1.5.2. Two vertices $c1$ и $c2$, are randomly selected from the vector x^* and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1$ is satisfied.
1.5.3. Based on the vector
$x^* = (x_1^*,\dots,x_{c1-1}^*,x_{c1}^*,\dots,x_{c2}^*,x_{c2+1}^*,\dots,x_M^*)$
The vector is created
$x = (x_1^*,\dots,x_{c1-1}^*,x_{c2}^*,\dots,x_{c1}^*,x_{c2+1}^*,\dots,x_M^*)$,

i.e. vertexes $x_{c1}^*,\dots,x_{c2}^*$ are transmitted in reverse order.
1.5.4. If $F(x) < F(x^*)$, then $x^* = x$, $m = 0$, otherwise $m = m + 1$
1.5.5. If $m < N_2$, then go to step 1.5.2.
2. Iteration number $n = 1$.
3. Performing a perturbation (generation of a solution $\hat{x}$ from a solution x^* based on 4-opt)
3.1. Three vertices are randomly selected from the x^* vector:
$c1 = 2 + (M/4) * U(0,1)$, $c2 = c1 + 1 + (M/4) * U(0,1)$,
$c3 = c2 + 1 + (M/4) * U(0,1)$,
where $U(0,1)$ – a function that returns a uniformly distributed random number in a range $[0,1]$
3.2. Based on the vector
$x^* = (x_1^*,\dots,x_{c1-1}^*,x_{c1}^*,\dots,x_{c2-1}^*,x_{c2}^*,\dots,x_{c3-1}^*,x_{c3}^*,\dots,x_M^*)$

The vector is created

$$\hat{x} = (x^*_1, \ldots, x^*_{c1-1}, x^*_{c3}, \ldots, x^*_{cM}, x^*_{c2}, \ldots, x^*_{c3-1}, x^*_{c1}, \ldots, x^*_{c2-1})$$

4. Local search based on 2-opt

4.1. $m = 0$

4.2. Two vertices $c1$ и $c2$, are randomly selected from the vector x^* and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1$ is satisfied.

4.3. Based on the vector

$\hat{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c1}, \ldots, \hat{x}_{c2}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$

The vector is created

$\check{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c2}, \ldots, \hat{x}_{c1}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$

i.e. the vertices $\hat{x}_{c1}, \ldots, \hat{x}_{c2}$ are rearranged in reverse order.

4.4. If $F(\check{x}) < F(\hat{x})$, then $\hat{x} = \check{x}$, $m = 0$, otherwise $m = m + 1$

4.5. If $m < N_2$, then go to step 4.2.

5. If $F(\hat{x}) < F(x^*)$, then $x^* = \hat{x}$

6. If $n < N_1$, then $n = n + 1$, go to step 3

The result is x^*.

6. Search with variable neighbourhood

Variable neighbourhood search was proposed by Mladenović and Hansen [28] and used a local search in an growing neighbourhood.

This algorithm is based on three principles:

- is a local minimum for one neighbourhood, possibly not a local minimum for another neighbourhood;
- the global minimum is a local minimum for all possible neighbourhoods;
- local minima are relatively close to global minima.

In this paper, the creation of a neighbourhood and a local search consists of the formation of a new solution by a permutation called "2-opt".

The method consists of the following steps:

An example of numbered list is as following.

1. Initialization

1.1. Setting the maximum number of iterations N_1; the maximum number of local search iterations N_2, where no new best solution is found; maximum neighbourhood size N_3; vertex vector lengths M.

1.2. An ordered set of vertices is specified $V = \{1, \ldots, M\}$ and the edge weight matrix $[d_{ij}]$, $i, j \in \overline{1, M}$.

1.3. Specifying the Cost Function (Goal Function)

$F(x) = d_{x_M, x_1} + \sum_{i=1}^{M-1} d_{x_i, x_{i+1}} \to \min\limits_{x}$,

where d_{x_M, x_1} – edge weight (x_i, x_{i+1}), $x_i, x_{i+1} \in V$, x – vertex vector.

1.4. Is set by randomly ordering the set V, the best vertex vectorx^*.

2. Iteration number $n = 0$.

3. Neighbourhood size$Z = 1$

4. Create a neighbourhood U_{x^*} solutions x^* based on 2-opt

4.1. $z = 1$

4.2. Two vertices $c1$ и $c2$, are randomly selected from the vector x^* and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1$ is satisfied.

4.3. Based on the vector

$x^* = (x^*_1, \ldots, x^*_{c1-1}, x^*_{c1}, \ldots, x^*_{c2}, x^*_{c2+1}, \ldots, x^*_M)$

the vector is created

$$x_z = (x^*_1, \ldots, x^*_{c1-1}, x^*_{c2}, \ldots, x^*_{c1}, x^*_{c2+1}, \ldots, x^*_M)$$

i.e. the vertices $x^*_{c1}, \ldots, x^*_{c2}$ are rearranged in reverse order.

4.4. If $x_z \notin U_{x^*}$, то $U_{x^*} = U_{x^*} \cup \{x_z\}$, $z = z + 1$

4.5. If $z \le Z$, then go to step 4.2.
5. Vector $\hat{x}$ is randomly selected from the neighbourhood U_{x^*}
6. Local search based on 2-opt
6.1. $m = 0$
6.2. Two vertices $c1$ и $c2$, are randomly selected from the vector $\hat{x}$ and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1$ is satisfied.
6.3. Based on the vector
$\hat{x} = (\hat{x}_1, \dots, \hat{x}_{c1-1}, \hat{x}_{c1}, \dots, \hat{x}_{c2}, \hat{x}_{c2+1}, \dots, \hat{x}_M)$
the vector is created
$\check{x} = (\hat{x}_1, \dots, \hat{x}_{c1-1}, \hat{x}_{c2}, \dots, \hat{x}_{c1}, \hat{x}_{c2+1}, \dots, \hat{x}_M)$
i.e. the vertices$\hat{x}_{c1}, \dots, \hat{x}_{c2}$ are rearranged in reverse order.
6.4. If $F(\check{x}) < F(\hat{x})$, then $\hat{x} = \check{x}$, $m = 0$, otherwise $m = m + 1$
6.5. If $m < N_2$, then go to step 6.2.
7. If $F(\hat{x}) < F(x^*)$, then $x^* = \hat{x}$, $n = 0$, go to step 3, otherwise $n = n + 1$
8. If $Z < N_3$, then $Z = Z + 1$, go to step 4
9. If $n < N_1$, then go to step 3
The result is x^*.

7. Greedy randomized adaptive search

Greedy randomized adaptive search was proposed by Feo and Resende [29]. The algorithm consists of two phases - a greedy randomized procedure and a local search. The goal of the algorithm is to repeatedly generate random greedy solutions and then use a local search to bring these solutions closer to local optima. In this paper, local search consists of the formation of a new solution by a permutation called "2-opt".

The method consists of the following steps:
1. Initialization
1.1. Setting the greediness parameter α or a greedy randomized procedure, where $\alpha \in [0,1]$ (where $\alpha = 0$ maximum greed)
1.2. Setting the maximum number of iterations N_1; the maximum number of local search iterations N_2, where no new best solution is found; vertex vector lengths M.
1.3. An ordered set of vertices is specified $V = \{1, \dots, M\}$ and the edge weight matrix $[d_{ij}]$, $i, j \in \overline{1, M}$.
1.4. Specifying the Cost Function (Goal Function)
1.5. $F(x) = d_{x_M, x_1} + \sum_{i=1}^{M-1} d_{x_i, x_{i+1}} \to \min\limits_{x}$,
where d_{x_M, x_1} – edge weight (x_i, x_{i+1}), $x_i, x_{i+1} \in V$, x – vertex vector.
Set by randomly ordering the set V, the best vertex vector x^*.
2. Iteration number $n = 1$.
3. Running a greedy randomized procedure
3.1. A vertex v^{cur} is randomly selected from the plurality of vertices.
From a set of vertices V, a vertex is randomly selected v^{cur}The initialization of the set of forbidden vertices is $V^{tabu} = \{v^{cur}\}$, number of forbidden vertices $l = 1$, $\hat{x}_1 = v^{cur}$
3.2. Creating a set of allowed vertices $\tilde{V} = V \backslash V^{tabu}$, initialization of the set of restricted candidates $V^{RCL} = \emptyset$, vertex number $j = 1$
3.3. $d^{min} = \min\limits_{s \in \overline{1, |\tilde{V}|}} d_{v^{cur}, \tilde{v}_s}$, $d^{max} = \max\limits_{s \in \overline{1, |\tilde{V}|}} d_{v^{cur}, \tilde{v}_s}$,
3.4. If $d_{v^{cur}, \tilde{v}_j} \le d^{min^{max^{min}}}$, then $V^{RCL} = V^{RCL} \cup \{\tilde{v}_j\}$
3.5. If $j < |\tilde{V}|$, then $j = j + 1$, go to step 3.3
3.6. From a set V^{RCL} a vertex is randomly selected v^{cur}, $V^{tabu} = V^{tabu} \cup \{v^{cur}\}$, $\hat{x}_{l+1} = v^{cur}$
3.7. If $l < M$, then $l = l + 1$, go to step 3.2
4. Local search based on 2-opt
4.1. $m = 0$

4.2. Two vertices $c1$ и $c2$, are randomly selected from the vector $\hat{x}$ and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1$ is satisfied.
4.3. Based on the vector
$\hat{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c1}, \ldots, \hat{x}_{c2}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$
the vector is created
$\breve{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c2}, \ldots, \hat{x}_{c1}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$
i.e. the vertices $\hat{x}_{c1}, \ldots, \hat{x}_{c2}$ are rearranged in reverse order.
4.4. If $F(\breve{x}) < F(\hat{x})$, then $\hat{x} = \breve{x}$, $m = 0$, otherwise $m = m + 1$
4.5. If $m < N_2$, then go to step4.2.
5. If $F(\hat{x}) < F(x^*)$, then $x^* = \hat{x}$
6. If $n < N_1$, then $n = n + 1$, go to step 3
The result is x^*.

8. Guided local search

Guided local search was proposed by Voudouris and Tsang [29]. The algorithm consists of two phases - local search and penalty modification. The use of a penalty function that increases the value of the goal function makes it possible to avoid local optima in local search. In this paper, local search consists of the formation of a new solution by a permutation called "2-opt".

The method consists of the following steps:
1. Initialization
1.1. Setting the scaling factor of the penalty function λ, and $\lambda > 0$ (for large λ the entire search space is explored, for small λ the search becomes directed)
1.2. Setting the maximum number of iterations N_1; the maximum number of local search iterations N_2, where no new best solution is found; vertex vector lengths M.
1.3. An ordered set of vertices is specified $V = \{1, \ldots, M\}$ and the edge weight matrix $[d_{ij}]$, $i, j \in \overline{1, M}$.
1.4. Specifying the Cost Function (Goal Function)

$$F(x) = d_{x_M, x_1} + \sum_{i=1}^{M-1} d_{x_i, x_{i+1}} \to \min_x,$$

where d_{x_M, x_1} – edge weight (x_i, x_{i+1}), $x_i, x_{i+1} \in V$,
x – vertex vector.
1.5. Specifying the penalty function

$$F_W(x, w) = w_{x_M, x_1} + \sum_{i=1}^{M-1} w_{x_i, x_{i+1}}$$

where $w_{x_i, x_{i+1}}$ – edge penalty (x_i, x_{i+1}), $x_i, x_{i+1} \in V$
1.6. Specifying an Extended Cost Function (Goal Function)

$$F_A(x, w) = F(x) + \lambda \cdot F_W(x, w) \to \min_x$$

1.7. Setting the utility function

$$f_U(x, w, i) = \begin{cases} \frac{d_{x_i, x_{i+1}}}{1 + w_{x_i, x_{i+1}}}, & 1 \le i < M \\ \frac{d_{x_M, x_1}}{1 + w_{x_M, x_1}}, & i = M \end{cases}, i \in \overline{1, M}$$

1.8. Set by randomly ordering the set V, the best vertex vector x^*.
1.9. The current solution is set $\hat{x}$, and $\hat{x} = x^*$.
1.10. Penalties are initialized
$w_{ij} = 0$, $i, j \in \overline{1, M}$
2. Iteration number $n = 1$.
3. Local search based on 2-opt
3.1. $m = 0$

3.2. Two vertices $c1$ и $c2$, are randomly selected from the vector $\hat{x}$ and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1$ is satisfied.
3.3. Based on the vector
$\hat{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c1}, \ldots, \hat{x}_{c2}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$
the vector is created
$\breve{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c2}, \ldots, \hat{x}_{c1}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$
i.e. the vertices $\hat{x}_{c1}, \ldots, \hat{x}_{c2}$ are rearranged in reverse order.
3.4. If $F_A(\breve{x}, w) < F_A(\hat{x}, w)$, then $\hat{x} = \breve{x}$, $m = 0$, otherwise $m = m + 1$
3.5. If $m < N_2$, then go to step3.2.
4. The penalties are modified
4.1. $i^* = arg \max_i f_U(\hat{x}, w, i)$
4.2. If $1 \le i^* < M$, then $w_{\hat{x}_{i^*}, \hat{x}_{i^*+1}} = w_{\hat{x}_{i^*}, \hat{x}_{i^*+1}} + 1$
If $i^* = M$, then $w_{\hat{x}_M, \hat{x}_1} = w_{\hat{x}_M, \hat{x}_1} + 1$
5. If $F(\hat{x}) < F(x^*)$, then $x^* = \hat{x}$.
6. If $n < N_1$, then $n = n + 1$, go to step 3
The result is x^*.

9. Partial optimization metaheuristic under special intensification conditions

Partial optimization metaheuristic under special intensification conditions was proposed by Taillard and Voss [30]. The algorithm consists of four phases - the choice of the number of the component (part) of the solution, the creation of a set of components closest to the selected one, the optimization procedure, and the analysis of the resulting solution. The larger the number of nearest components, the larger the neighbourhood. In the optimization procedure (for example, search with tabus), only the nearest components of the solution are changed (in the case of the problem of finding the optimal path, these are vertices). In this paper, the creation of a neighbourhood consists of forming a new solution by a permutation called "2-opt".The method consists of the following steps:

1. Initialization
1.1. Setting the maximum number of nearest components Q, the maximum number of iterations N_1; the maximum number of search iterations with tabus N_2; the size of neighbourhood Z; solution length M; maximum length of the tabu list L_{max}.
1.2. An ordered set of vertices is specified $V = \{1, \ldots, M\}$ and the edge weight matrix $[d_{ij}]$, $i, j \in \overline{1, M}$.
1.3. Specifying the Cost Function (Goal Function)
$F(x) = d_{x_M, x_1} + \sum_{i=1}^{M-1} d_{x_i, x_{i+1}} \to \min_x$,
where d_{x_M, x_1} – edge weight (x_i, x_{i+1}), $x_i, x_{i+1} \in V$,
x – vertex vector.
1.4. Set by randomly ordering the set V, the best vertex vector x^*.
1.5. The current solution is set $\hat{x}$, and $\hat{x} = x^*$.
1.6. Initialization of the set of tabu vertices $V^{tabu} = \emptyset$.
2. Iteration number $n = 1$.
3. The number of the solution component is randomly selected j, and $\hat{x}_j \notin V^{tabu}$
4. Creating a set of vertex numbers V^U
4.1. All vertices $\hat{x}_i$ are sorted by proximity to the vertex $\hat{x}_j$
4.2. Q of the top (closest) vertices are selected, from which a set V^U is created
5. Optimization procedure - use of search with prohibitions
5.1. Initializing the tabu list $T = \emptyset$
5.2. Search iteration number with tabus $m = 1$.
5.3. Creating a neighbourhood $U_{\hat{x}}$ solution $\hat{x}$ based on 2-opt
5.3.1. $z = 1$

5.3.2. Two vertices $c1$ и $c2$, are randomly selected from the vector $\hat{x}$ and the selection of these vertices continues until the condition $1 < c2 - c1 < M - 1 \wedge c1, c2 \in V^U$ is satisfied.
5.3.3. Based on the vector
$\hat{x} = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c1}, \ldots, \hat{x}_{c2}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$
the vector is created
$x_z = (\hat{x}_1, \ldots, \hat{x}_{c1-1}, \hat{x}_{c2}, \ldots, \hat{x}_{c1}, \hat{x}_{c2+1}, \ldots, \hat{x}_M)$
i.e. the vertices $\hat{x}_{c1}, \ldots, \hat{x}_{c2}$ are rearranged in reverse order.
5.3.4. A pair of tabu edges is created $E_z = ((c1 - 1, c1), (c2 - 1, c2))$ for the vertex vector x_z
5.3.5. $j = 1$
5.3.6. If $j < M \wedge (x_{zj}, x_{z,j+1}) \in T$ or $j = M \wedge (x_{zM}, x_{z1}) \in T$, then go to step 5.3.2
5.3.7. If $j < M$, then $j = j + 1$, go to step 5.3.6
5.3.8. If $x_z \notin U_{\hat{x}}$, then $U_{\hat{x}} = U_{\hat{x}} \cup \{x_z\}$, $z = z + 1$,
5.3.9. If $z \leq Z$, then go to step 5.3.3.
5.4. From neighbourhood $U_{\hat{x}}$ a solution with the lowest price is selected x_{z^*}, i.e. $z^* = arg\min_z F(x_z)$
5.5. If $F(x_{z^*}) \geq F(\hat{x})$, then go to step 5.9
5.6. $\hat{x} = x_{z^*}$.
5.7. A pair of edges is placed at the start of the tabu list .
5.8. If $|T| > L_{max}$, then a pair of edges that were tabued earlier is pushed from the end of the tabu list T.
5.9. If $m < N_2$, then $m = m + 1$, go to step 5.3
6. If $\hat{x} = x^*$, then $V^{tabu} = V^{tabu} \cup \{j\}$ (a stronger version is possible $V^{tabu} = V^{tabu} \cup V^U$), otherwise $x^* = \hat{x}$, $V^{tabu} = V^{tabu} \backslash V^U$ (a weaker version is possible $V^{tabu} = \emptyset$)
7. If $n < N_1$ и $|V^{tabu}| < M$, then $n = n + 1$, go to step 3
The result is x^*.

10. Quantitative study

The quantitative study of the proposed optimization methods was carried out using the Matlab package. For the traveling salesman problem, the search for a solution was carried out on the standard database berlin52. For this database, the optimal path length is 7542.

In this paper, for a self-organizing feature map, the annealing temperature decrease function is determined by the formula $T(n) = \beta^n T_0$ and is shown in Fig.1. In this case, the initial effective width of the topological neighbourhood is 1, the initial temperature is 106, and the cooling coefficient is 0.94.

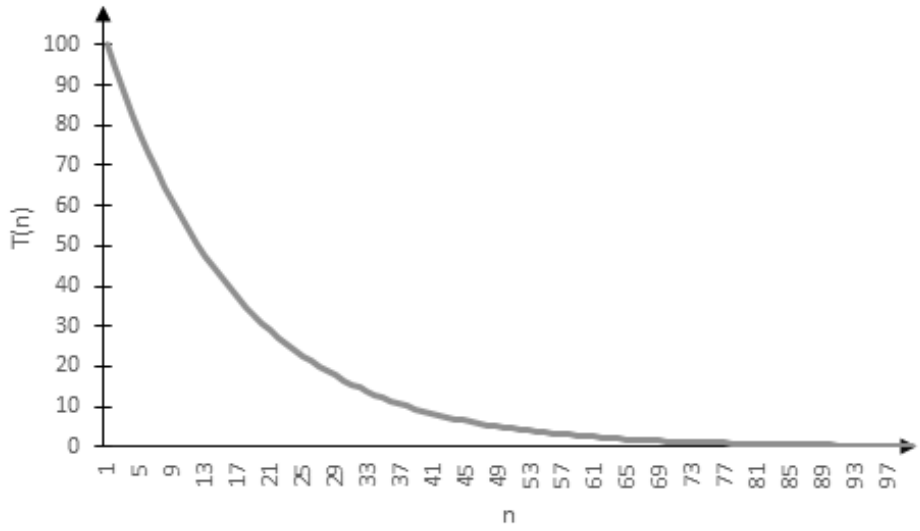

Figure 1: Annealing temperature decrease function

The dependence (Fig. 1) of the annealing temperature on the iteration number shows that the annealing temperature decreases with an increase in the iteration number.

In this case, the effective width of the topological neighbourhood according to the formula $\sigma(n) = \sigma_0 \, exp\left(-\frac{1}{T(n)}\right)$and is presented in Fig.2

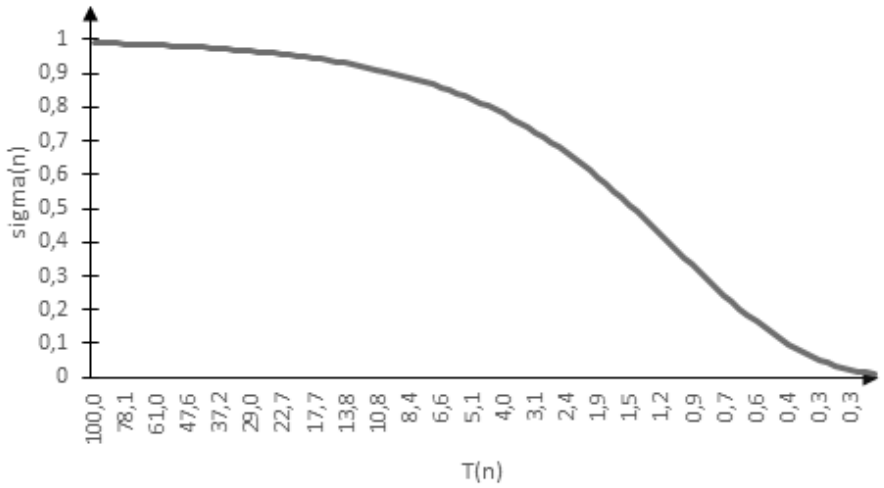

Figure 2. Effective Width of Topological Neighbourhood

The dependence (Fig. 2) of the effective width of the topological neighbourhood on the annealing temperature shows that the effective width of the topological neighbourhood decreases with decreasing temperature.

The results of comparing the proposed SOFM learning method with the existing one based on the criteria of time and path length are presented in Table 1, where M is the number of input layer neurons (or the length of the vertex coordinate vector), N^h is the number of output layer neurons, N is the maximum number of iterations.

Table 1

Comparison of the proposed SOFM training method with the existing one based on the criteria of time and path length

Criteria Head	Proposed method	Existing method
Time	$M \cdot N$ (in case of use of GPU)	$M \cdot N \cdot (M \cdot N^h)$
Path length	7919	8510

According to Table 1, the proposed teaching method gives a faster and more accurate result than the existing teaching method.

In this paper, for single-solution human-based metaheuristics, the maximum number of iterations is 100, and the neighbourhood size is 50. The penalty function scaling factor is 70, the greed parameter is 0.3, the maximum number of nearest components is 10, and the maximum length of the prohibition list is 3.

The results of the comparison of the proposed methods with the taboo search method based on the length of the path are presented in Table 2.

Table 2

Comparison of Proposed Single-Solution Human-Based Metaheuristics with Taboo Search Method

№ п/п	Metaheuristics	Best path length found
1	Iterative local search	8176
2	Search with variable neighborhood	8154
3	Greedy randomized adaptive search	8097

4	Guided local search	8034
5	Partial optimization metaheuristic under special intensification conditions	7919
6	Taboo Search	8296

According to Table 2, the proposed single-solution human-based metaheuristics give a more accurate result than the taboo search, and the partial optimization metaheuristic under special intensification conditions is the most accurate.

11. Conclusions

1. The urgent task of increasing the efficiency of methods for solving the traveling salesman problem was undertaken by creating methods based on artificial neural networks and single-solution human-based metaheuristics.
2. The proposed modification of the learning method for a self-organizing feature map uses:

- setting weights of the winning neuron and its neighbours based on the calculation of the centre of gravity, this allows for accelerating the training of this neural network due to parallelization (in case of GPU use);
- the effective dynamic width of the topological neighbourhood for calculating the topological neighbourhood function, which allows to explore the entire search space at the initial iterations and make the search directed at the final iterations, this ensures high accuracy of the search through this neural network.

3. A modification of single-solution human-based metaheuristics is proposed that allows potential integer solutions and uses a 2-opt permutation in the case of a local search neighbourhood creation, and a 4-opt permutation in the case of a global search, which allows adapting these metaheuristics to solve the traveling salesman problem.

The proposed methods make it possible to expand the scope of application of the self-organizing feature map and single-solution human-based metaheuristics, which is confirmed by their adaptation to the traveling salesman problem and contributes improve efficiency of intelligent computational systems.

Prospects for further research are the study of the implementation of the proposed method on a broad class of artificial intelligence problems.

12.References

[1] Rothlauf F. Design of modern heuristics: Principles and application / F. Rothlauf. – New York: Springer-Verlag, 2011. – 267 p.

[2] Nakib A. Metaheuristics for Medicine and Biology / Nakib A., Talbi El-G. – Berlin: Springer-Verlag, 2017. – 211 p.

[3] Engelbrecht A.P. Computational Intelligence: an introduction / A.P. Engelbrecht. – Chichester, West Sussex: Wiley & Sons, 2007. – 630 p.

[4] Yu X. Introduction to evolutionary algorithms / X. Yu, M. Gen. – London: Springer-Verlag, 2010. – 433 p.

[5] Subbotin S. Diagnostic Rule Mining Based on Artificial Immune System for a Case of Uneven Distribution of Classes in Sample / S. Subbotin, A. Oliinyk, V. Levashenko, E. Zaitseva // Communications. – Vol.3. – 2016. – P.3-11.

[6] Yang X.-S. Nature-inspired Algorithms and Applied Optimization / X.-S. Yang. – Charm: Springer, 2018. – 330 pp.

[7] Blum C. Hybrid Metaheuristics. Powerful Tools for Optimization / C. Blum, G. R. Raidl. – Charm: Springer, 2016. – 157 p.

[8] Martí R., Handbook of Heuristics / R. Martí, P.M. Pardalos, M.G.C. Resende. – Charm: Springer, 2018. – 1289 p.

[9] Yang X.-S. Optimization Techniques and Applications with Examples / X.-S. Yang. – Hoboken, New Jersey: Wiley & Sons, 2018. – 364 p.
[10] Gendreau M. Handbook of Metaheuristics / M. Gendreau, J.-Y. Potvin. – New York: Springer, 2010. – 640 p.
[11] Doerner K.F. Metaheuristics. Progress in Complex Systems Optimization / K.F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R.F. Hartl, M. Reimann. – New York: Springer, 2007. – 408 p.
[12] Chopard B. An Introduction to Metaheuristics for Optimization / B. Chopard, M. Tomassini. – New York: Springer, 2018. – 230 p.
[13] Bozorg-Haddad O. Meta-heuristic and Evolutionary Algorithms for Engineering Optimization / O. Bozorg-Haddad, M. Solgi, H. Loaiciga. – Hoboken, New Jersey: Wiley & Sons, 2017. – 293 p.
[14] Babayigit B. A clonal selection algorithm for array pattern nulling by controlling the positions of selected elements / B. Babayigit, K. Guney, A. Akdagli // Progress In Electromagnetics Research. – Vol. 6. – 2008. – P. 257-66.
[15] Radosavljević J. Metaheuristic Optimization in Power Engineering / J. Radosavljević. – New York: The Institution of Engineering and Technology, 2018. – 536 p.
[16] Du K.-L. Search and Optimization by Metaheuristics. Techniques and Algorithms Inspired by Nature / K.-L. Du, M.N.S Swamy. – Charm: Springer, 2016. – 434 p.
[17] Alba E., Nakib A., Siarry P. Metaheuristics for Dynamic Optimization. – Berlin: Springer-Verlag, 2013. – 398 p.
[18] Kobayashi M. Quaternionic Hopfield neural networks with twin-multistate activation function. Neurocomputing 267, 304–310 (2017)
[19] Haykin S. Neural Networks and Learning Machines. Upper Saddle River, New Jersey: Pearson Education. Inc., 2009. 906 p.
[20] K.-L. Du, K. M. S. Swamy, Neural Networks and Statistical Learning, Springer-Verlag, London, 2014.
[21] Russell S. Artificial Intelligence: a Modem Approach / S. Russell, P. Norvig. – Englewood Cliffs, NJ: Prentice Hall PTR, 2020. – 1136 p p.
[22] M. Tim Jones. Artificial Intelligence: A Systems Approach. Hingham, MA: INFINITY SCIENCE PRESS LLC, 2008. – 498 p.
[23] The method of intelligent image processing based on a three-channel purely convolutional neural network / Fedorov, E., Lukashenko, V., Patrushev, V., Rudakov, K., Mitsenko, S. // CEUR Workshop Proceedings, 2018, 2255, pp. 336–351.
[24] T Kohonen (2012) Self-organizing and associative memory, 3rd edn. Springer, New York
[25] T Kohonen. Essentials of the self-organizing map. Neural Networks Volume 37, January 2013, Pages 52-65 https://doi.org/10.1016/j.neunet.2012.09.018
[26] Parallel computational algorithms in thermal processes in metallurgy and mining / Shvachych, G.G., Ivaschenko, O.V., Busygin, V.V., Fedorov, Ye.Ye. // Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu, 2018, Vol. 4, pp. 129–137
[27] Automated control of temperature regimes of alloyed steel products based on multiprocessors computing systems / Shlomchak, G., Shvachych, G., Moroz, B., Fedorov, E., Kozenkov, D. // Metalurgija, 2019, 58(3-4), pp. 299–302
[28] Talbi El-G. Metaheuristics: from design to implementation / El-G. Talbi. – Hoboken, New Jersey: Wiley & Sons, 2009. – 618 p.
[29] Glover F. Handbook of Metaheuristics / F. Glover, G.A. Kochenberger. – Dordrecht: Kluwer Academic Publishers, 2003. – 570 p.
[30] Brownlee J. Clever Algorithms: Nature-Inspired Programming Recipes / J. Brownlee. – Melbourne: Brownlee, 2011. – 436 p.