Intelligent algorithms of construction of public transport routes

Mirxalil Ismailov[1], *Davron* Ziyadullaev[2*], *Dilnoz* Muhamediyeva[1], *Rano* Gazieva[1], *Aksulu* Dzholdasbaeva[2], and *Sharofiddin* Aynaqulov[1]

[1]"Tashkent Institute of Irrigation and Agricultural Mechanization Engineers" National Research University, Tashkent, 100000, Uzbekistan
[2]Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Tashkent, 100200, Uzbekistan

Abstract. Today, in public transport planning systems, it is relevant to a search for a possible route with a minimum time. The aim of the work is the development of intelligent algorithms for constructing public transport routes, the development of programs, and the conduct of a computational experiment. Research methods are the theory of neural networks. The paper considers Hopfield neural networks and proposed recurrent neural networks. However, in Hopfield neural networks, the chances of solving this optimization problem decrease as the matrix size increases. A recurrent neural network is proposed, represented by a differential equation to solve this problem. As a result, the number of iterative computations can be reduced by n^2 times than in the Hopfield network.

1 Introduction

Algorithms for finding the shortest path are used to build public transport routes. For those who want to use public transport, many online services help plan the route. However, the problem is a bit more complicated because the routes have to be short; for example, the next train cannot arrive at the station before the departure of the current train [1-7]. Besides, there may be several mandatory criteria. A short train/air transfer can be very important in reducing the overall travel time. Price may also be taken into account. In airline companies, sites such as http://orbitz.com or http://expedia.comgive a long list of possible routes for users.

There are more sophisticated train journey planning tools in Europe, such as the HAFAS planning system. One of these approaches is that a network that expands over time can be wide. An alternative approach is to define a "time-dependent" model in which one node is available for each station [1]. If a modified version of Dijkstra's algorithm finds the optimum path between two stations, the first problem that comes to mind is whether it can calculate the shortest path from one node to another. For example, the time-based approach can be used to combine the modified Dijkstra algorithm with the A* search algorithm and

*Corresponding author: dziyadullaev@inbox.ru

use the Nachtigal algorithm [2], which calculates the shortest path between nodes, and computes results for the German railways, consisting of 26 routes and 37 stations.

Pyrga et al. [3] studied a two-criteria version of time-varying approaches to minimize transport time and the number of transfers. Their largest examples were networks with over 30,000 ends and 90,000 edges for a time-dependent approach.

According to the analysis, the development of hardware, computing tools, and software to solve the problem of traffic regulation is growing. At the same time, the number of addresses, ends, and edges of traffic is growing rapidly, moving from one parameter to more parameters and from static parameters to dynamic ones [4-7].

The study presents an analytical table of the level of complexity of algorithms for solving problems of finding the shortest path by static parameters (Table 1).

Table 1. Algorithms for finding the shortest path based on static parameters

Algorithm name	Complexity level	Author
Ford algorithm	O(V2E)	Ford1956
Bellman and Ford algorithms	O(VE)	Bellman 1958, Moore 1957
Denzig algorithms	O(V2 log V)	Denzig1958, Denzig 1960
Dijkstra algorithm	O(V2)	Leyzorek 1957, Dijkstra 1959
Dijkstra and Fibanacci algorithm	O(E + V log V)	Friedman & Tarjan 1984, Friedman & Tarjan 1987
Johnson algorithms	O(E log log L)	Johnson 1982, Karlsson & Poblete 1983
Gabov algorithm	O(E logE/VL)	Gabov 1983, Gabov 1985
Ahuja algorithm	O(E + V√log L)	Ahuja 1990

The concepts of choosing a route and the shortest path are very close to each other, and the problem needs to be solved. However, the shortest path problem is a special case of the routing problem, and routing problems can be used to find ideal solutions to this problem. Finding the shortest paths is integral to many routing problems [4-7].

2 Methods

To solve this routing problem, we construct the objective function of the Hopfield neural network. We use two indices to represent each neuron. In this case, the routing index is the node number, and the second index is the node access sequence number at the time of the route. For example, the expression $Y_{xi} = 1$ means the input of the x-bit node as the i-th node in the route.

The objective function must satisfy two conditions: first, each row and each column of the direction matrix should have the minimum (unique)value; second, the length of the entire path in the chosen direction should be minimal.

The fulfillment of the first condition of the objective function can be checked based on the following expression [8]:

$$E_{1,2,3} = \frac{A}{2}\sum_{X}\sum_{i}\sum_{X}Y_{Xi}Y_{Xj} + \frac{B}{2}\sum_{i}\sum_{X}\sum_{T\neq X}Y_{Xi}Y_{Ti} + \frac{C}{2}\left[\left(\sum_{X}\sum_{i}Y_{Xi}\right) - V\right]^2 \qquad (1)$$

where A, B and C are constant numbers. To achieve this aim, the following conditions must be met:

1. If there is no more than one unit in each row, the sum of the first three terms is zero.

2. If there is no more than one unit in each column (the sequence number of entry into the node), the sum of the second triple is zero.

3. If there are V units in the matrix, the third sum is zero.

The objective function is focused on finding the minimum path based on the fulfillment of the second condition –introducing an additional element to the objective function:

$$E_4 = \frac{D}{4}\sum_X \sum_{T \neq X} \sum_i d_{XT} Y_{Xi} \left(Y_{T,i+1} + Y_{T,i-1}\right).$$

Sufficiently large values for parameters A, B, and C indicate that the route has the least cost, while a large value of D ensures that the shortest path is selected.

Opening the brackets of expression (1) and introducing additional variables, we create a matrix of weights for the following neural network connections:

$$w_{Xi,Ti} = -A\delta_{XT}\left(1-\delta_{ij}\right) - B\delta_{ij}\left(1-\delta_{XT}\right) - C - D \cdot d_{XT} \cdot \left(\delta_{j,i+1} + \delta_{j,i-1}\right)$$

where δ_{ij} is the Kronecker parameter, which takes the value of 1 if condition $i = j$ is satisfied, otherwise, it takes the value of 0.

We proposed to choose the following function as the activation function of the Fneuron:

$$F = \frac{1}{2}\left[1 + \frac{e^{(\beta u_0)} - 1}{e^{(\beta u_0)} + 1}\right]$$

where u_0 is the threshold value of neural network connections, e is the exponential function, β is the constant number.

Based on experiments, it was observed that effective results could be achieved with this function in many cases, so we have proposed the same function to solve this problem.

Then, the initial arbitrary values of the weight coefficients of the neural network were obtained, and in the next steps, the parameters of the neural network were selected, which can solve the problem as a result of their evolutionary change.

The Hopfield network setting process continues iteratively until the network's position remains unchanged, and the function reaches its minimum value. After the termination of the computational process, the neural network output is taken as the most optimal direction.

To solve this problem, the Hopfield network algorithm can be expressed in the following steps:

Step 1. Initialization:

Const: A, B, C, D, u_0, tao=1.

CityXY is node coordinates, N is number of nodes.

Step 2. Calculation of distances between nodes:

$$d(i, j).$$

Step 3. Initialization of the initial values of the neural network weight matrix:

X=rand();

$$U = a\tanh(2 * X - 1) * u_0.$$

Step 4. Calculation of the optimization function:

The first part of the optimization function is (there is only one unit in each row of the matrix)

$$E_1 = \frac{A}{2}\sum_i \sum_j \sum_{k \neq j} X_{ij} X_{kj} \cdot$$

The second part of the optimization function is (there is only one unit in each column of the matrix)

$$E_2 = \frac{B}{2}\sum_i \sum_j \sum_{k \neq i} X_{ki} X_{ji} \cdot$$

The third part of the optimization function is (one direction only)

$$E_3 = \frac{C}{2}\left[\left(\sum_i \sum_j X_{ij}\right) - N\right]^2 \cdot$$

Minimizing the length (cost) of the sought-for path, we obtain

$$E_4 = \frac{D}{4}\sum_i \sum_{j \neq i} \sum_k d[i,j] X_{ik}\left(X_{j,k+1} + X_{j,k-1}\right) \cdot$$
$$Udao = -U + E_1 + E_2 + E_3 + E_4.$$

Step 5. Recalculation of weights of neural connections:

U=U+lamda*Udao;

We calculate the output value of the neuron

$$F = \frac{1}{2}\left[1 + \frac{e^{(U/u_0)} - 1}{e^{(U/u_0)} + 1}\right].$$

Recalculation of neural output based on threshold function

$$\begin{cases} X = 0, \text{ if } F < 0.3; \\ X = 1, \text{ if } F > 0.7. \end{cases}$$

Step 6. Testing:

1. If there is no more than one unit in each raw, then the condition $\sum_{i=1}^{N}\sum_{j=1}^{N-1}\sum_{k=j+1}^{N} X_{ij} X_{ik} = 0$ is satisfied.

2. If there is no more than one unit in each column (the sequence number of entry into the node), then the condition $\sum_{i=1}^{N}\sum_{j=1}^{N-1}\sum_{k=j+1}^{N} X_{ji} X_{ki} = 0$ is satisfied.

3. If there are N units in the matrix, then the condition $\sum_i \sum_j X = N$ is satisfied.

Step 7. Termination.

If all three check conditions are met simultaneously, the iteration stops, and the sequence of found nodes is printed; otherwise, the algorithm continues from step 3.

In many cases, the solution to combinatorial optimization problems may require a lot of computational processes and computational time. In such cases, to increase the program's speed and efficiency, there are cases of inappropriate actions, such as purchasing a number of additional devices and software and attracting additional services. However, intuitive assessment does not always lead to effective success. When calculating according to the Amdahl law based on ideal parallelism to processor *p*, the calculating speed can be expressed as follows [10]

$$S_n = \frac{1}{(1-\eta)+\eta/n} \le \frac{1}{1-\eta}. \qquad (2)$$

Based on this expression, 95 percent parallel computing speed of 20 processors gives a numerical value of 1/(0.05 + 0.95/20) = 10.26. In practice, the efficiency of parallel computing may be lower. Observations have shown that in a modern dual-core computer with two cores, the solution to a problem divided into four mutually independent processes is 1.5 times faster than the speed of a simple, sequential solution, as long as it is possible. It should also be noted that the speed and quality of parallelization also depend on the communication parameters.

Gustafson and Barsis, in 1988, found that the execution time of the non-parallel part of the program takes less time than its parallel parts [10, 11]. This is especially true for the non-parallel preparation of sections and results processing. In this case, it is convenient to load a non-parallel task as a parameter of a parallel task:

$$\sigma_n = \frac{1-\eta}{\eta/n+1-\eta}. \qquad (3)$$

Gustafson and Barsis combined (2) with formula (3) to get the following expression:

$$S_n \le n + (1-n)\sigma_n \Rightarrow \sigma_n \le \frac{n-S_n}{n-1}. \qquad (4)$$

The resulting formula (4) allows us to estimate the percentage of the sequential part of the program required to achieve a given speed. For example, to increase the performance of a program by 19 times on 21 computers, the proportion of sequentially running programs on each computer should not exceed (21-19)/(21-1)=10%.

The main conclusion of the Gustafson-Barsis law is that parallelization is effective for large tasks when the task requires a lot of time for parallel computing. Parallelization across multiple processors is effective for very large tasks.

It is known that when solving this problem in the Hopfield neural network, neurons tend to solve the problem on the "one after another" principle. This dramatically increases the computational processes. If there is a matrix dimension, then based on this matrix, iterative calculations can be performed to solve this optimization problem using the Hopfield grid. It follows that as the size of the matrix increases, the chances of solving this optimization problem decrease.

To solve the problem (1), a recurrent neural network is proposed, represented by the following differential equation [12]:

$$\frac{\partial u_{ij}(t)}{\partial t} = -\eta\left(\sum_{k=1}^{n} x_{ik}(t) + \sum_{l=1}^{n} x_{lj}(t) - 2\right) + \lambda r_{ij} \exp\left(-\frac{t}{\tau}\right).$$

Here $x_{ij} = \varphi(u_{ij}(t)),\ \varphi(u) = \dfrac{1}{1+\exp(-\beta u)}$.

The finite difference version of the above equation for solving the proposed neural network can be expressed as:

$$u_{ij}^{t+1} = u_{ij}^{t} - \Delta t \cdot \left[\eta\left(\sum_{k=1}^{n} x_{ik}(t) + \sum_{l=1}^{n} x_{lj} - 2\right) - \lambda r_{ij} \exp\left(-\frac{t}{\tau}\right)\right], \qquad (5)$$

where Δt is the time step, and the value of this parameter can take values in the range [0,1]. Parameters $\eta, \lambda, \tau, \beta$ are chosen based on experiments and have a significant impact on the speed of solving the problem and the quality of this solution. Observations show that the chances of obtaining an effective result $\eta = \dfrac{1}{t}$ increase. Here t is the number of iteration steps.

To speed up the solution of the above system of equations (5), it was proposed to use the "Winner takes all" principle [19]. Accordingly, the solution to this optimization problem is performed in the following sequence:

1. Matrix $\left\|x_{ij}\right\|$ of random values $x_{ij}^{0} \in [0,1]$ is generated.

2. Iteration(7) continues until the following inequality condition is met:

$$\sum_{k=1}^{n} x_{ik}(t) + \sum_{l=1}^{n} x_{lj}(t) - 2 \le \varepsilon,$$

here ε is a very small positive number.

3. Processing of the generated elements $\left\|x_{ij}\right\|$ of the matrix:

3.1. From the i-th row of matrix $x_{i,j_{\max}}$, the element with the maximum value is obtained, where $j_{\max}$ is the column number of the element with the maximum value in the row.

3.2. Replacement $x_{i,j_{\max}} = 1$ is performed by the element obtained. The values of the row and all other elements in the column where this element is located are replaced with zeros:

$$x_{i,j} = 0,\ j \ne j_{\max},$$
$$x_{k,j_{\max}} = 0,\ k \ne i.$$

The next step is to go back to row $j_{\max}$.

Steps 3.1 and 3.2 continue until the switching processes return to the first row. This condition indicates that the iteration is complete.

4. If the transition to the first row is done before *n* elements $\left\| x_{ij} \right\|$ of the matrix take the value of 1, than this means that the number of iterations performed was less than *n*. In this case, steps 1 and 3 are repeated.

3 Results and Discussion

When transmitting vector data coming to the input of the neural network, the initial states of the neurons are determined. Since the neural network has the feedback property, in the next steps, the outputs of the neurons again come to their inputs in the form of a new vector based on the feedback, and the state of the neurons changes again. It is known that recurrent neural networks are directly related to the concept of neuronal stasis [12–14]. According to the Lyapunov criterion, such a neural network is considered stable if, after a finite number of iterations, the state of the neurons takes a state in which the topologies do not change. The output signals of neurons are formed due to transferring the vector to the input of fixed recurrent networks. They again enter the input as unwanted signals and form a new precedent vector. However, as the number of iterations increases, the number of changes in node states decreases until the final state of the network is established. It is known that open-loop networks are always stable because when a single unwanted vector is fed into the input, the nodes of the network can change their position only once due to the continuity of neuron inputs.

One of the most important features of artificial neural networks is the reliability of neural network models and systems. This feature allows the implementation of practical neural network systems in various sectors of the economy and solutions to problems that require high reliability.

Another important feature of neural networks is their ability to learn. Taking advantage of this ability, adequate adaptive neural network models were built to solve problems (Fig. 1,2).

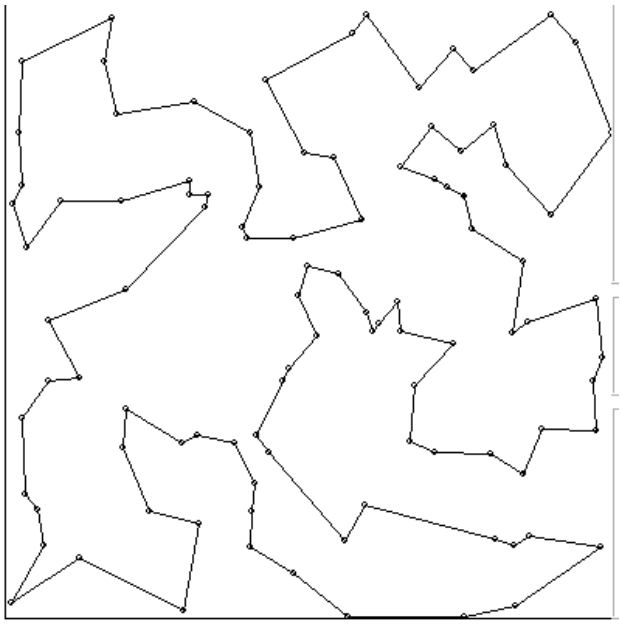

Fig. 1. Nearest distance between 100 nodes obtained from neural network models

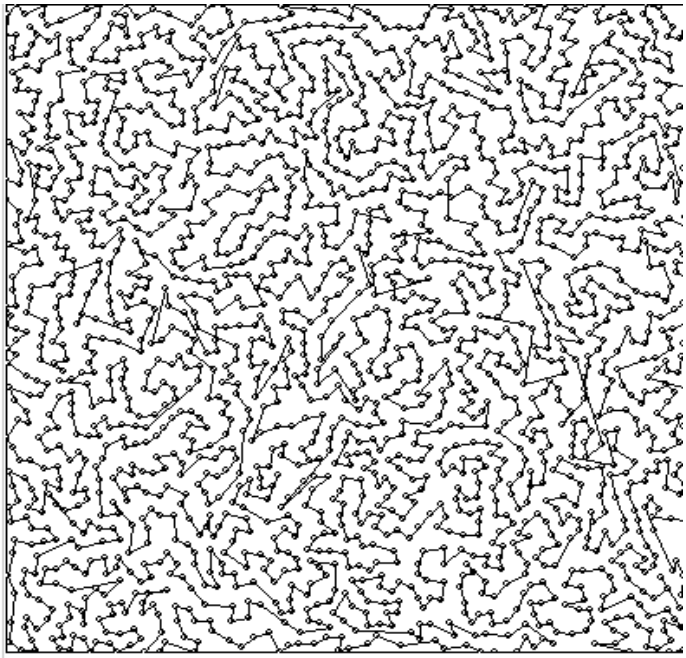

Fig. 2. Nearest distance between 2000 nodes obtained from neural network models

The objective function of the given optimization problem must satisfy two conditions: first, the resulting solution matrix is unique in each row and in each column; second, the total time obtained on the selected objects should be minimal.
To solve this problem, it was proposed to choose the following function as a neuron activation function:

$$f(u)=\frac{1}{2}\left[1+\frac{e^{(\beta u_0)}-1}{e^{(\beta u_0)}+1}\right],$$

where u_0 is the threshold value of neural network connections, e is the exponential function, β is the constant number.

The results of numerous studies have shown that, in many cases, effective results can be achieved with this function, and for this task, it was proposed to choose this particular activation function.

It can be seen that in the proposed neural network $n \times n$ dimensional matrix is formed, as in the Hopfield network. But in this case, neurons interact not "one after another" but in rows and columns. As a result, the number of iterative calculations can be reduced by n^2 times compared with the Hopfield network [16–20].

4 Conclusions

1. The work considers the Hopfield neural network for finding the optimal route by public transport. A computational experiment was carried out to find the optimal route for different numbers of nodes.

2. When the number of nodes is more than 30, the time to find the optimal route based on the Hopfield neural network has increased dramatically. To overcome this problem, a recurrent neural network is presented, represented by a differential equation.

3. The methods and algorithms used in the software package designed to find the optimal route for vehicles with different numbers of nodes have been tested. The method, algorithm, and modules of the software package presented in the research work are based on the effectiveness of model tests. The time to find the optimal route with the number of 2000 nodes is equally decreased by times than in the Hopfield network.

References

1. M. Muller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis, Algorithmic methods for railway optimization. Lecture Notes in Computer Science, vol. 4359, Timetable Information: Models and Algorithms, pp. 67-90, Springer Berlin / Heidelberg, September 2007.
2. K. Nachtigal, Time depending shortest-path problems with applications to railway networks, European Journal of Operations Research 83 (1995), 154-166.
3. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, Experimental comparison of shortest path approaches for timetable information. Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM, 2004, pp. 88- 99.
4. U. Zwick, All pairs shortest paths using bridging sets and rectangular matrix multiplication. Journal of the ACM 49 (2002). no. 3, 289-317.
5. T. Hagerup, Improved. Shortest Paths in the Word RAM, 27th Int. Colloq. on Automata, Languages and Programming, Geneva, Switzerland, 2000, pp. 61-72.
6. U. Meyer, Single-Source Shortest Paths on Arbitrary Directed Graphs in Linear Average Time, 12th Symp. on Discr. Alg., 2001, pp. 797-806.
7. M. Thorup, Integer priority queues with decrease key in constant time and the single source shortest paths problem, J. Comput. Syst. Sci. 69 (2004), no. 3, 330-353.
8. Komashensky V.I., Smirnov D.A. Neural networks and their application in control and communication systems. - M.: Hotline - Telecom, 2003.
9. Falfushinsky V.V. Parallel data processing in multicomponent observation systems. // Cybernetics and system analysis. International scientific and theoretical journal. - Ukraine. No. 2, 2002.
10. Antonov A. Under the Amdahl law // Computerra. - 11.02.2002. — No. 430(in Russian)
11. Quinn M.J Parallel Programming in C with MPI and OpenMP. — New York: NY: McGraw-Hill, 2004.
12. Kruglov V.V., Dli M.I., Golunov R.Yu. Fuzzy logic and artificial neural networks. – M.: Fizmatlit. 2001. - 224 p.
13. A. P. Rotshtein, Fuzzy multicriteria choice of alternatives: worst case method, News of RAN. Theory and control systems. 2009. - No. 3. - P. 51-55
14. Rotstein A.P. Intelligent identification technologies: fuzzy logic, genetic algorithms, neural networks. -Vinnitsa: UNIVERSUM-Vinnitsa. 1999. - 320 p.

15. Rutkovskaya D., Pilinsky M., Rutkovsky L. Neural networks, genetic algorithms and fuzzy systems: Translated from Polish. I.D. Rudinsky. -M.: Hotline-Telecom, 2004. - 452 p.
16. Mukhamediyeva D K 2020 Study parabolic type diffusion equations with double nonlinearity. IOP Conf. Series: Journal of Physics: Conference Series1441 012151
17. Z Abdullaev, G E Ziyodullaeva, D T Muhamedieva. Introduction of modern information and communication technologies in transport flow management // IOP Conf. Series:Journal of Physics: Conference Series, 2176 (2022) 012017.
18. Z Abdullaev, D Sh Ziyadullaev, D T Muhamediyeva. The task of assessing the risk in the operation of a complex free formal system // IOP Conf. Series:Journal of Physics: Conference Series, 2176 (2022) 012071.
19. D T Muhamediyeva. Building and training a fuzzy neural model of data mining tasks // IOP Conf. Series:Journal of Physics: Conference Series, 2182 (2022) 012024.
20. D T Mukhamedieva, A X Mirzaakhmedova and U U Khasanov. Development of a model for determining theoptimal number of urban passenger transport // IOP Conf. Series:Journal of Physics: Conference Series, 2182 (2022) 012025