Parallelization of the Method of Simulated Annealing when Solving Multicriteria Optimization Problems

Lesia Mochurad [1[0000-0002-4957-1512]], Nataliya Boyko[1[0000-0002-6962-9363]], Vasyl Sheketa[2[0000-0002-1318-4895]]

[1] Department of Artificial Intelligent Systems, Lviv Polytechnic National University, 12 S. Bandery str., Lviv, 79000, Ukraine
[2] Ivano-Frankivsk National Technical University of Oil and Gas

lesia.i.mochurad@lpnu.ua, nataliya.i.boyko@lpnu.ua, vasylsheketa@gmail.com

Abstract. In the analysis of methods of multicriteria optimization. The detailed implementation of the parallel algorithm of the simulated annealing method is reproduced by the example of the extension of a large-scale travelling salesman problems. For this purpose are used such properties as multithreading and multicore of modern computer systems. An application software system was developed. We conducted a number of experimental studies. Adhering to the results that indicate that more computational process optimization is available that is at the optimal gap of the multicriteria optimization problem, the large rate for probable variations are parallel threads and computer cores.

Keywords: Travelling Salesman Problem, Parallel Algorithm, Multithreading, Multicore, Efficiency Factor.

1 Introduction

In the process of designing intelligent control systems, there is often the task of determining the best values for the parameters or structure of objects [1, 2]. This task is called optimization. Today, optimization problems and decision-making problems are modeled and solved in various fields of engineering [3-7]. The skills of mathematical justification for decision making include the skills of mathematical modeling of optimization problems, the choice of adequate mathematical support (method, algorithm, software system) with the necessary justification, the analysis of the obtained results and their interpretation in terms of the subject area.

For example, the task of the travelling salesman problem is widely used in computer-aided design, transportation systems, PCB (printed circuit boards) manufacturing, protein structure studies, X-ray crystallography, and other fields. An important feature of these problems is their large dimension and the inability to obtain real-time solutions [8, 9].

Purpose and task. The purpose of this work is to parallel the simulated annealing method and to develop software for large-scale multicriteria optimization. To achieve this goal, you must solve the following problems:

- To analyze the existing methods of multicriteria optimization and to outline the advantages of using the simulated annealing method;
- Develop an application software system to parallelize large-scale problem solving, based on the simulated annealing method;
- Without reducing the generality, as an example of solving a travelling salesman problem, to evaluate the effectiveness of the proposed algorithm of parallel calculations in terms of quality and timing of the task based on the experimental studies and comparative analysis of the results obtained.

The object of study – Is the process of paralleling the annealing method when solving large-scale multicriteria optimization problems.

The subject of the study – Is methods, algorithms and software for solving large-scale multicriteria optimization problems.

Research methods. The simulated annealing method to solve large-scale multicriteria optimization problems was analyzed in this work. For the parallelization, it is suggested to use the many thread properties and Java programming tools. Algorithm theory, object-oriented programming methods were used in the development of the software.

2 Related Works

The optimization problem as a whole is reduced to finding the extremum (minimum or maximum) of the objective function with given constraints [10]. Its mathematical formulation is as follows:

The values of the vector variables must be defined: $x = (x_1, x_2, ..., x_m)$, which satisfy the limitation of appearance $g_i = (x_1, x_2, ..., x_m) \leq b_i$, for every $i = 1, ..., k$ and at which the maximum or minimum of the objective function is reached $f(x) = (x_1, x_2, ..., x_m)$: $f(x) = (x_1, x_2, ..., x_m) \rightarrow (\max, \min)$. The final solution to the problem is a pair that consists of the optimal solution and the optimal value of the objective function.

The methods of mathematical programming give a great variety of algorithms for solving this problem [11, 12]. In general, search algorithms implement methods of descent to the extremum, in which the values of the objective function are consistently improved until reaching the extremum. Depending on the possibility of algorithm of finding a local or global extremum, they are divided into local and global search algorithms.

Local extremum search algorithms are designed to determine one of the local extrema on the set of endless solutions in which the objective function takes the maximum or minimum value. In their construction, both deterministic descent in the extremum and random search can be used. The deterministic methods distinguish between zero-order and gradient (1st and 2nd order) methods. The first ones are based on calculations only of the values of the optimized function. The second use partial derivatives of the second order. Methods of stochastic programming or neural networks are used to find extremum in cases where the type of optimized function is not fully known or its structure is too complex. The effectiveness of the optimal search procedure - the ability to find the solution and the convergence to the solution on the speed depend on the type of function and the method applied to it.

Of the direct methods of multicriteria optimization in the search for local extremum the most famous are [13-15]:

- coordinate descent – alternate parameter optimization along axes by one of the known one-dimensional methods;
- spiral coordinate descent;
- rotating coordinates (Rosenbrock method);
- simplex search;
- Hook-Jeeves sample search and more.

The task of finding the global extremum of a function on a valid set X is to find the point $x_* \in X$, for which is executed: $f(x_*) \le f(x),\ f(x_*) \ge f(x)$ for each $x \in X$. Limitations related to computational error and other factors often do not allow finding the exact solution to the problem. In this case, there is a search for an approximate solution, that is, the point of many $\varepsilon -$optimal solutions $X_*^\varepsilon = \{x \in X : f(x) \le f(x_*) + \varepsilon\}$. Finding the exact solution can be considered as a case of searching for a close solution with $\varepsilon = 0$.

The global extremum search algorithms are divided into deterministic, statistical and combined [16]. Often, the task of global optimization is reduced to the task of finding local extrema and finding among them the global optimum, thus, using the methods of finding local extrema.

Determined methods for finding a global solution include the Gomori algorithm or the cutting-plane method. The latter is an alternative basis for the construction of both accurate and approximate algorithms for solving integer programming problems. Currently, their high efficiency in combination with the branch and bound method [17] is proven. Such hybrid computing schemes are commonly referred to as branch and clipping method. All of these methods implement a common computational strategy that consists in solving the sequence of relaxed linear programming sub-tasks. In the clipping method, relaxed subtasks gradually improve the approximation of a given integer problem, reducing the neighborhood of the optimal solution.

In severe cases, optimality may not be obtained or proved by an acceptable number of steps, however, even in this case, the clipping methods allow us to find an approximate integer solution with a given error (in the previously known neighborhood of the optimal solution).

Cutting-plane method algorithms can be effectively used to solve both partial integer programming problems, including the traveling salesman problem, the maximum cut problem, the knapsack problem, and the overall integer linear programming problem.

Compared to branch and bound methods, cutting-plane methods are more technological in programming because they do not require extra amount of RAM to store the decision tree, but at the same time less versatile because they are not able to work with relaxed subtasks that are convex programming tasks. The first cutting-plane algorithm was proposed by R.E. Gomori in 1958. Currently, these methods are effectively used to solve various tasks, including the travelling salesman problem.

Recent studies [17] have also shown that the cuts proposed by Gomori are quite actual at present. The advantage of this method is that any linearity in the original statement of the problem remains. This algorithm is quite effective for solving a certain class of problems – geometric programming. Its main disadvantage is the requirement of the convexity of the valid area and the increasing dimension of the linear programming problem from iteration to iteration.

Genetic algorithms are a class of optimization methods based on imitations of processes occurring in nature, in particular natural selection – a concept voiced from the evolutionary theory of Charles Darwin [16]. In accordance with Darwin's theory in the natural environment, preference for survival and reproduction is given to individuals most adapted to the conditions of a particular habitat. The main material for natural selection is natural gene mutations and their combinations obtained by reproduction. In the course of natural selection, individuals with the greatest fitness function – a numerical characteristic that is determined according to a specific task - survive. The most fit individuals get the opportunity to cross (crossover) and give offspring. The resulting population is then affected by random mutations.

Reformulate the optimization problem as the problem of finding the maximum of a function $f(x_1, x_2, ..., x_n)$, called the fitness function. It is necessary that $f(x_1, x_2, ..., x_n) \geq 0$ in a limited area of definition, continuity and differentiation are absolutely not required. Each parameter of the fitness function is encoded by a row of bits. The individual will be called a row, which is a concatenation of rows of an ordered set of parameters.

The versatility of genetic algorithms is that only parameters such as adaptability and decision coding depend on a particular task. The rest of the steps for all tasks are the same. With the function of fitness among all individuals, the population distinguishes:

- the most adapted, which get the opportunity to cross and give offspring;
- the worst (bad decisions) that are removed from the population.

In addition to the global extremum search methods described above, there is also the method of simulated annealing that was selected in the article for further research.

The algorithm of simulated annealing is based on the simulation of a physical process that occurs during crystallization of a substance from a liquid state into a solid, including when annealing of metal. The final state after crystallization corresponds to the minimum energy of the lattice configuration.

Before applying any optimization algorithm, the data on which the model will be formed are selected according to certain requirements, namely:

1. Volume and representativeness, that is, the data should be as fresh as possible, and the sample should be large depending on the number of parameters being optimized.
2. The minimum of rules and parameters, because with the increase of parameters that are optimized, the probability of fitting the model to historical data increases, rather than reflecting the real patterns of market behavior.
3. The sample should be divided into two parts. One part of the data to create a model, the other – to test the created model.

3 Setting the Task

Without reducing the generality, let us look at the main aspects of parallelizing the annealing simulation method, as an example of solving a traveling salesman problem. It is known that the traveling salesman problem has a wide application [8]. However, an important feature of these tasks is their large dimension, sometimes over one million points. The traveling salesman problem belongs to the class NP because it has factorial computational complexity. This, in turn, does not allow accurate resolutions to be made for large dimension problems in an acceptable time. Therefore, there is a need to analyze the possibility of parallelizing the solution of the problem and developing an appropriate application software system [19]. An algorithm based on an annealing simulation method and a property such as multithreading [20] are proposed.

The traveling salesman problem can be represented as a model on a graph, that is, using vertices and edges between them. Thus, the vertices of the graph correspond to the cities and the edges $(i,\ j)$ between vertices i i j - ways of communication between these cities. To each rib $(i,\ j)$ it is possible to set the criterion of profitability of the route $C_{i,j} > 0$, which can be understood as, for example, distance between cities, time or cost of travel. Thus, the solution to the traveling salesman problem is to find the Hamilton cycle of the minimum weight cycle in a fully weighted graph.

4 Research Methods and Tools

4.1. An annealing simulation method

The following describes the algorithm of the annealing method. This algorithm is proposed in the paper to be used in solving large-scale multicriteria optimization problems.

The main elements of the annealing method are:

1. A limited set S.
2. A valid target function J, that is defined at the set S. Let us mark $S^* \subset S$„ the set of global minimum of the function J.
3. For each $i \in S$„ call the set $S(i) \subset S - \{i\}$ as the set of adjacent nodes of i.

4. For each i set of positive coefficients $q_{ij}, j \in S(i)$, such that $\sum_{j \in S(i)} q_{ij} = 1$.
   Suppose that $j \in S(i)$ if and only if $i \in S(i)$.
5. Non-growing function $T : N \to (0, \infty)$, called the freezing schedule and $T(t) -$ temperature of the moment t.
6. Initial conditions $x(0) \in S$.

The algorithm is based on a discrete-time inhomogeneous Markov chain $x(t)$, whose development is as follows:

1. If the current state $x(t)$, equivalent to i, select the next node j for i randomly, the probability of choosing an individual $j \in S(i)$ is q_{ij}.
2. Then the next state $x(t+1)$ is defined as follows:
   2.1. If $J(j) \le J(i)$, then $x(t+1) = j$.
   2.2. If $J(j) \ge J(i)$, then $x(t+1) = j$ with the probability of $\exp\left[-(J(j) - J(i) / T(t)\right]$, else $x(t+1) = i$.

   Formally, $P\left[x(t+1) = j \middle| x(t) = i\right] = q_{ij} \cdot \exp\left[-\frac{1}{T(t)} \max J(j) - J(i)\right]$, if $j \ne i,\ j \in S(i)$.

   If $j \ne i,\ j \notin S(i)$, then $P\left[x(t+1) = j \middle| x(t) = i\right] = 0$.

You can justify the described method by considering the inhomogeneous Markov chain $x_T(t)$, for which temperature $T(t)$ is kept constant level of T. Suppose that the Markov chain is irreducible and aperiodic and $q_{ij} = q_{ji}$ for every $i,\ j$. Then $x_T(t)$ - is a Markov reversible chain with its invariant probability of distribution $\pi_T(i) = \frac{1}{Z_T} \exp\left[-\frac{J(i)}{T}\right],\ i \in S,$ where Z_T - is the normalizing constant. At $T \to 0$, probability distribution π_T concentrates on the plural S^* the global minimum. The conditions of convergence of the algorithm are formulated by Hayek [21].

4.2. Multithreading in Java

Java implements integrated multithreaded programming support. A multithreaded program contains two or more parts that can be executed simultaneously.

Each part of such a program is called a thread, and each thread specifies a separate execution path. In other words, multithreading is a specialized form of multitasking.

In a multitasking environment, the smallest element of managed code is thread. This means that one program can perform two or more tasks simultaneously. For example, a text editor can format text at the same time as it is printed, as long as the two actions are performed by two separate threads.

Multithreading allows you to write effective programs that make the most of the available power of the processor of the system. Another advantage of multithreading is that it minimizes downtime. This is especially important for interactive Java-based networking environments, as they have idle time and downtime.

For example, the speed of data transmission over a network is much lower than the speed at which a computer can process it. Even reading and writing local file system resources is much slower than the processing speed of the processor. And, of course, the user enters the keyboard much slower than the computer can process.

In single-threaded environments, your application is forced to wait for such tasks to complete before moving on to the next one, even if the program is idle most of the time, waiting for input. Multithreading helps reduce downtime because other threads can run while one waits.

Java assigns to each thread a priority that determines the behavior of that thread relative to others. Thread priorities are given by integers that determine the relative priority of one thread over the other.

The priority value itself is irrelevant – the higher priority of thread is not executed faster than the lower priority when it is the only executable thread at this time.

Instead, the thread priority is used to make the decision when switching from one running thread to another. This is called context switching. The rules that determine when context switching should take place are quite simple.

The thread may voluntarily give way to management. To do this, you can either explicitly concede the execution queue, suspend the thread, or block the I/O wait time. In this scenario, all other threads are checked, and the CPU resources are transferred to the thread with the highest priority ready to execute.

The thread may be interrupted by another, more priority thread.

In this case, the low priority thread, which is not occupied by the processor, is simply terminated by the high priority thread, no matter what it does.

5 Results

The main steps of the program:

- Set the initial temperature and random initial solution;
- We get into a cycle that is active until the condition is reached.
- Now we choose a neighbor making small changes in the decision.
- We then decide whether a new decision is worth considering.
- We reduce the temperature and make a new iteration of the cycle.

Initialization of the temperature

For better optimization when initiating a temperature variable, you should choose a temperature that will initially allow practically any movement against the current solution. This gives the algorithm a better idea of exploring the entire search space before cooling and settling in a more focused area.

Visualization of the problem and its results

For a better understanding of the algorithm, we decided to visualize the results. In Fig. 1 is shown the initial map of the cities to be crawled.

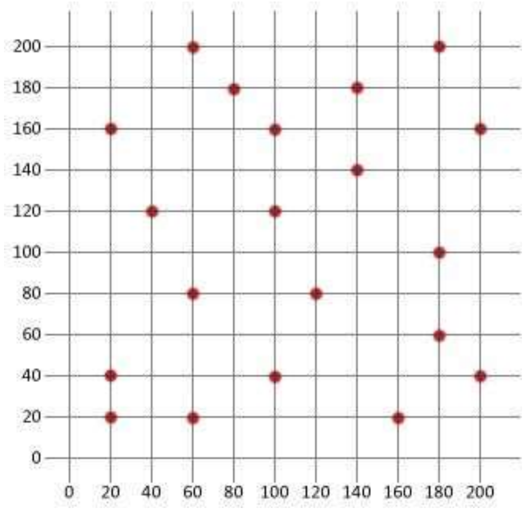
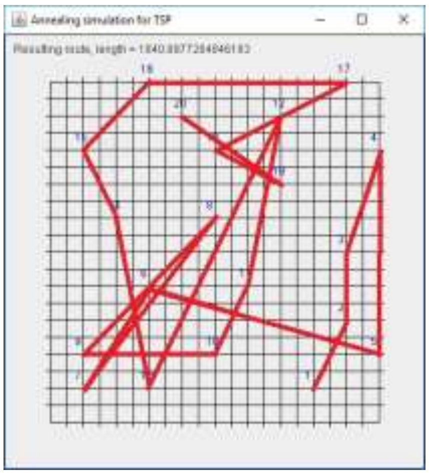
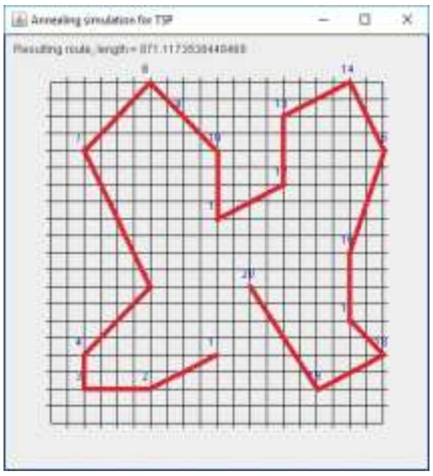


Fig. 1. Initial City Map (Quantity: 20)

In Fig. 2 is shown the operation of the algorithm for the number of cities: 20 and cooling rate 0.1. And in Fig. 3 at the same number of cities and cooling rate – 0.0001.



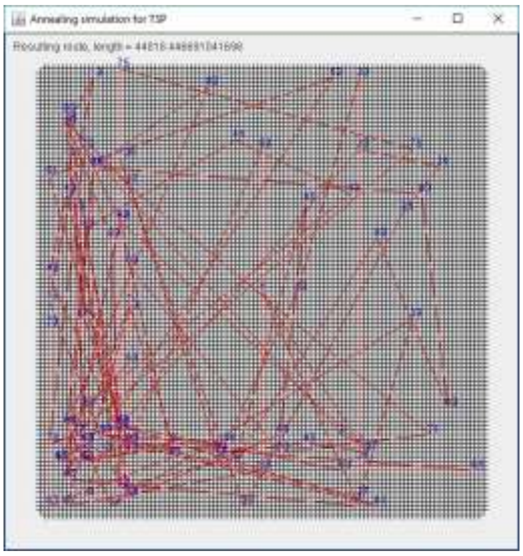
Resulting route, length = 1840.8977284846183



Resulting route, length = 871.1173538440488

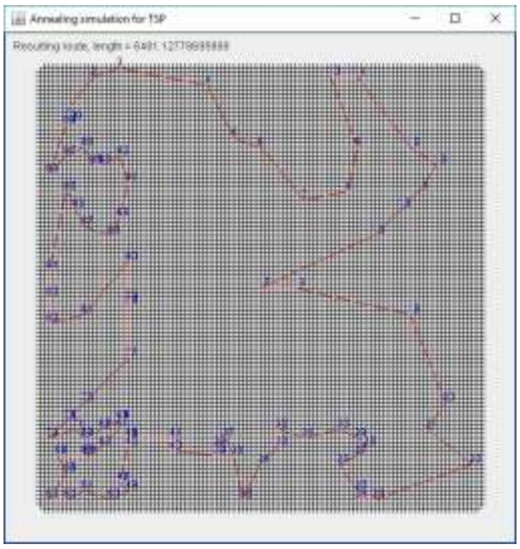
Fig. 2. Coolingspeed – 0.1(20) **Fig. 3.** Coolingspeed – 0.0001(20)

In Fig. 4 is shown the operation of the algorithm at a coolingrate of 0.1 and the number of cities: 100. And in Fig. 5 – at a coolingrate of 0.0001 and the number of cities: 100.


Annealing simulation for TSP

Resulting route, length = 44818.448881041698

Fig. 4. Coolingspeed – 0.1(100)


Annealing simulation for TSP

Resulting route, length = 6481.12778699888

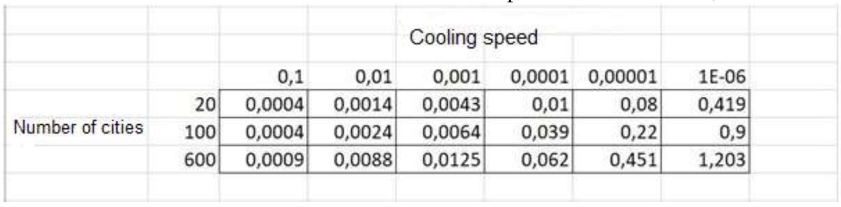
Fig. 5. Coolingspeed – 0.00001(100)

From the results of testing the algorithm at different cooling rates, we can conclude that the lower the cooling temperature, the better the algorithm works to find a solution to the travelling salesman problem.

For the sake of clarity and objective assessment of parallelization performance, tests were conducted on different cities and PCs with different amount of cores.

In the Tabl. 1 is shows the results of calculations performed on two-core PC without parallelization.

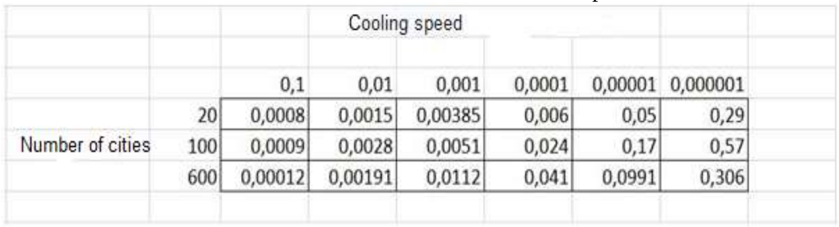
Tabl. 1. Solution time of theTSP without parallel on the 2-core, s



		Cooling speed					
		0,1	0,01	0,001	0,0001	0,00001	1E-06
Number of cities	20	0,0004	0,0014	0,0043	0,01	0,08	0,419
	100	0,0004	0,0024	0,0064	0,039	0,22	0,9
	600	0,0009	0,0088	0,0125	0,062	0,451	1,203

In the Tabl. 2 is shown the results of the calculations performed on two parallel nuclear PCs.

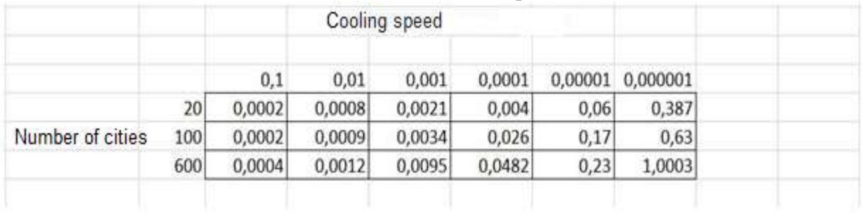
Tabl. 2. Solution time of theTSP with 2-core parallel, s



		Cooling speed					
		0,1	0,01	0,001	0,0001	0,00001	0,000001
Number of cities	20	0,0008	0,0015	0,00385	0,006	0,05	0,29
	100	0,0009	0,0028	0,0051	0,024	0,17	0,57
	600	0,00012	0,00191	0,0112	0,041	0,0991	0,306

In the Tabl. 3 is shown the results of the calculations performed on four-core non-parallel PCs.

Tabl. 3. Solution time of theTSP without parallel on the 4-core, s



		Cooling speed					
		0,1	0,01	0,001	0,0001	0,00001	0,000001
Number of cities	20	0,0002	0,0008	0,0021	0,004	0,06	0,387
	100	0,0002	0,0009	0,0034	0,026	0,17	0,63
	600	0,0004	0,0012	0,0095	0,0482	0,23	1,0003

In Fig. 6 the dependence of the program execution time on the number of flows on two and four nuclear architectures with/without the use of parallel simulated annealing method for the number of cities – 20 is presented.

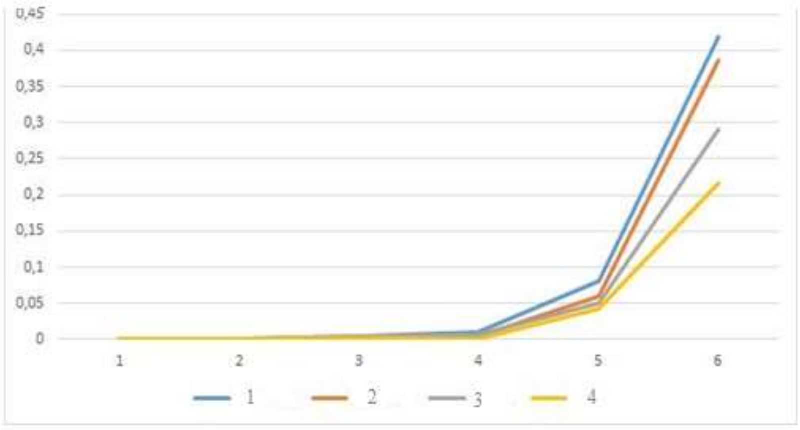


Fig. 6. Graph of algorithm results for the number of cities: 20

Here 1 – is dependency without parallelization on a dual-core processor; 2 – without parallelization on quad-core; 3 – with parallelization on a dual-core processor; 4 – with a quad-core parallel. These designations are preserved when the following dependencies are displayed (see Fig. 7-8).

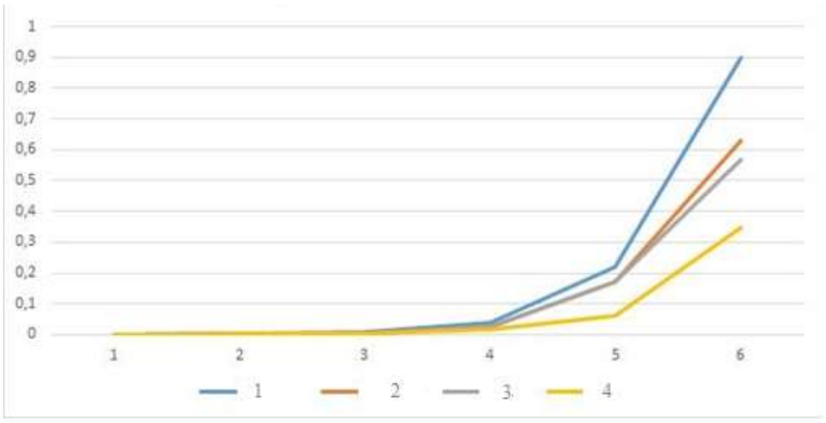


Fig. 7. Graph of algorithm results for the number of cities: 100

In Fig. 7 the dependence of the program execution time on the number of threads on two- and four-core architectures with/without the use of parallel simulated annealing method for the number of cities – 100 is presented.

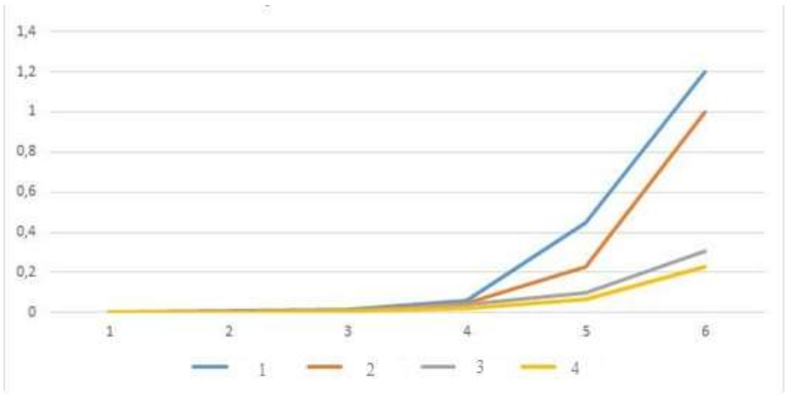


Fig. 8. Graph of algorithm results for the number of cities: 600

In Fig. 8 shows the dependence of the program execution time on the number of threads on two- and four-core architectures with/without the use of parallel simulated annealing method for the number of cities – 600.

In the process carrying out numerical experiments, it was found that for a small number of cities, the execution time of the program at parallel is almost indistinguishable from the execution time of the program in a single thread. But the larger the number of cities, the better results at parallelization. Also, the program works more efficiently with more cores of PC.

6 Conclusion

In order to organize efficient calculations suitable for large-dimensional problems, in this work was proposed a parallel algorithm for finding the global extremum – a method of simulated annealing. An application software system has been developed to give the opportunity to solve a large-scale traveling salesman problem. The program is written in Java using a property such as multithreading. On the basis of a number of numerical experiments, the advantages of the proposed approach were analyzed: graphs of dependencies of the implementation time of the sequential and parallel algorithm depending on the dimension of the processed data and the multi-core architecture of the corresponding computing system were constructed. On the basis of coefficients of efficiency and acceleration the prospects of further optimization of the computational process due to the modern development of multi-core systems are investigated.

References

1. Baklan, I.V., Bidiuk, P.I., Nesterenko O.V.: Designing Intelligent Decision Making Systems, K. NAU. 196 p. (2010).
2. Gozhiy, A.P.: Basic Aspects of Application of Information Technologies in Scenario Planning Problems. Scientific Works of the ChDU of Petro Mohyla: Mykolaiv, series: Computer Technologies, № 148, T.160, 158-167 (2012).
3. Trius, Y.V., Manko, M.O.: Web-oriented consulting expert system on optimization methods. Bulletin of the Cherkasy University. Series: Applied Mathematics. Computer Science, № 18. 99-114 (2014).
4. Litvin, V.V.: Problems of optimizing the structure and content of ontologies and methods for solving them. Visn. Nat. University of Lviv Polytechnic. "Information systems and networks", № 715. 189–200 (2011).
5. Prokudin, G.S., Belous, S.O.: One of the approaches to solving the network transport problem. Traffic safety at the crossroads of Ukraine, K.: OOO "Magazine" Rainbow", №. 1, 2 (15). 52-56 (2003).
6. Syerov, Y., Shakhovska, N., Fedushko, S.: Method of the Data Adequacy Determination of Personal Medical Profiles. Proceedings of the International Conference of Artificial Intelligence, Medical Engineering, Education (AIMEE2018). Advances in Artificial Systems for Medicine and Education II. Volume 902, 2019. pp. 333-343. https://doi.org/10.1007/978-3-030-12082-5_31
7. Mastykash, O., Peleshchyshyn, A., Fedushko, S., Trach O. and Syerov, Y.: Internet Social Environmental Platforms Data Representation, 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, pp. 199-202. (2018) doi: 10.1109/STC-CSIT.2018.8526586
8. Bazylevych, R., Kutelmakh, R., Kuz, B.: Methods of clustering of a set of points in a salesman problem with constraints. Visnyk of Lviv National University, № 672: Computer Science and Information Technology. 207-212 (2010).
9. Bazylevych, R., Kutelmakh, R., Prasad, B., Bazylevych, L.: Decomposition and scanning optimization algorithms for TSP. Proceeding of the International Conference on Theoretical and Mathematical Foundations of Computer Science, Orlando, USA. 110-116 (2008).
10. Vitlinsky, V.V., Tereshchenko, T.O., Savina, S.S.: Economic-mathematical methods

and models: optimization: textbook, K.: KNEU. 303 p. (2016).
11. Samoilenko, M.I., Skokov, B.G.: Operations Research (Mathematical Programming. Queuing Theory): Educ. Manual, Kharkiv: HNAMG. 176 p. (2005).
12. Dunaevskaya, O.I., Akhiezer, E.B.: Essence of mathematical methods and models for solving economic problems. International conferences: Research and optimization of economic processes "Optimum": Kharkov. 128 – 134 (2014).
13. Klimov, A.S.: Numerical methods for solving the problem of optimal design of complex technical systems. Vis. Nat. aviation. un-ty, № 1. 133-139 (2006).
14. Zakharova, E.M., Minashina, I.K.: A review of multidimensional optimization methods. Information processes, T. 14, № 3. 256-274 (2014).
15. Sorin Mihai Grad: Vector Optimization and Monotone Operators via Convex Duality. Recent Advances (Springer). 269 p. (2014).
16. Larionov, Y.I., Levikin, V.M., Khazhmuradov, M.A.: Research of operations in information systems. Kharkov: SMIT Company. 364 p. (2005).
17. Solovyova, T. A., Khristoforova, E.I.: The Gomori method for solving integer programming problems. Youth and Science: Proceedings of the VIII All-Russian Scientific and Technical Conference of Students, Post-Graduate Students and Young Scientists, dedicated to 155-the anniversary of the birth of K. E. Tsiolkovsky [Electronic resource], Krasnoyarsk: Siberian Federal University. Access mode: http://conf.sfu-kras.ru/sites/mn2012/section11.html (2012).
18. Boyko, N., Shakhovska, K.: Information system of catering selection by using clustering analysis : 2018 IEEE Ukraine Student, Young Professional and Women in Engineering Congress (UKRSYW), October 2 – 6, Kyiv, Ukraine, pp.7-13 (2018).
19. Mochurad, L., Boyko, N.: Solving Systems of Nonlinear Equations on Multi-core Processors. DOI: 10.1007/978-3-030-33695-0_8, 17 p. (2020).
20. Schildt, Herbert: Java. The Complete Guide, 8-th ed. M .: ID Williams LLC. 1104 p. (2012).
21. Bertsimas, D., Tsitsiklis, J.: Simulated Annealing. Statistical Science, Vol. 8, № 1. 10-15 (1993).
22. Mochurad, L., Shakhovska, Kh., Montenegro, S.: Parallel Solving of Fredholm Integral Equations of the First Kind by Tikhonov Regularization Method Using OpenMP Technology. Advances in Intelligent Systems and Computing IV. DOI: 10.1007/978-3-030-33695-0_3, 11 p. (2020).