Azerbaijan Journal of High Performance Computing, Vol 1, Issue 1, 2018, pp.113-125
https://doi.org/10.32010/26166127.2018.1.1.113.125

Controlling Optimization Software Packages with the Application of Parallel Computing

Samir Z. Guliyev

High Performance Computing Research Advance Center, Department of General and Applied Mathematics, Azerbaijan State Oil and Industry University, Baku, Azerbaijan, azcopal@gmail.com

*Correspondence: Samir Guliyev, High Performance Computing Research Advance Center, Department of General and Applied Mathematics, Azerbaijan State Oil and Industry University, Azerbaijan, Baku, azcopal@gmail.com

Abstract

The paper is devoted to the analysis of techniques and algorithms of controlling computational process of solution to complex optimization problems with the use of multiprocessor and/or multicore computer systems. We have developed automatic and dialog systems of control of an optimization process. The paper contains the protocols and results of computer-based experiments for the class of unconstrained optimization problems on the basis of the developed software package.

Keywords: optimization methods, parallel computations, multiprocessor and multicore systems, dialog systems

1. Introduction

It is known that in spite of a large number of methods for numerical solution to various classes of problems, the choice of the most efficient method for solving a particular problem under specific values of its parameters requires a large number of comparative experiments. As a rule, the end users tend to have difficulty both in carrying out such experiments, which requires the knowledge of the domain of applicability of various numerical methods, and in proper conducting of the comparative analysis of the results, which is quite time consuming.

In the paper, for the class of problems of multivariate unconstrained optimization, we propose two approaches for facilitating the use of available applied software packages using modern multi-processor (multicore) computer systems. One of the approaches involves active work of the user with the optimization program package in a dialogue mode. The other approach involves the packet control by means of a specially developed control program in automatic mode.

Let's note that beginning from the 70s of the last century, in different schools, they have been carrying out works on creating intellectual algorithms for controlling software packages. For example, Yu.G.Evtushenko's school have developed controlling dialog systems of unconstrained optimization ("DISO") and of optimal control ("DISOPT") [1,2]. In the works [3,4], they developed controlling systems of solving vector optimization problems ("DIVO") and global optimization problems ("GLOPT"). These systems are constantly improved and modified in connection with the development of information, computer, and software technologies. The results of the given paper are direct development of ideas employed in the above-mentioned systems, taking into account new possibilities provided by modern computing technologies.

Drawing on the existing experience in the development of systems of this type, we formulated some basic requirements for the dialog optimization system being developed:

- the ability to specify initial values and variants of initial configurations in various ways, as well as to change flexibly the parameters of steps, criteria, etc.;
- the possibility of manual management of configuration, dynamic change of the problem dimension in the optimization process;
- the use of visual aids to represent the results in the form of mapping the optimization trajectory;
- step-by-step fixing of results; display of optimization results with the ability to change configuration parameters at any step; storing a configuration tree with the ability to roll back to any step.
- the possibility of development; creation of a system with the ability to easily implement new algorithms, rules, criteria, new types of tasks, etc.;
- software implementation; the system ought to be developed taking into account modern requirements for algorithmic software, opportunities for further development, for example, creating a module to display graphically the result of optimization, to integrate with office applications to facilitate the procedure for data entry and output of reporting data. The main way to implement the dialog system is object-oriented programming.

2. *Problem Statement*

Let $P = \{p_i(x) : i \in N\}$ be the class of optimization problems (tasks). Here N is a given set defining individual problems of the class; $x \in D_i \subset R^n$ are the arguments of each individual problem, which can take on values from some given admissible set D_i, defined by each specific optimization problem individually. It is assumed that for every problem $p_i(x)$ there exists a goal subset of extrema $D_i^* \subset D_i$ such that $D_i^* \neq \emptyset$. The problem $p_i(x)$ consists in finding at least one point $x^* \in D_i^*$. The set D_i^* is called a set of solutions to the problem $p_i(x)$.

To solve all the problems of the class P, there is usually a corresponding family of methods $M = \{M_j : j \in J\}$, each of which solves the problems $p_i(x)$ of the given class, i.e. they find a point $x^* \in D_i$. Moreover, each method M_j, $j \in J$, has different efficiency (in terms of time used, the accuracy of the solution, etc.) when solving the problem $p_i(x)$.

As the optimization techniques we use methods of direct search (zero-order methods), gradient-based methods (first-order methods), and Newton-type methods (second-order methods) [5-9]. These methods have a large number of options settings, thus providing the ability to adapt the system to any process quickly. Furthermore, the combination of direct search methods, gradient-based methods, and Newton-type methods allow us to find an optimal solution for a smaller number of steps and/or calculations of the objective function, which is important in terms of the cost of optimization process.

The paper sets out the possible principles of management of optimization software package when solving a particular applied problem $p_i(x) \in P$, allowing us to increase the overall efficiency of solving the problem by combining the optimization methods in the process of solving the problem with the use of a multiprocessor (multicore) computer system.

2.1. The principle of sequential implementation on a single-core architecture

This principle is of important significance in its own right, and can be considered as the basic unit for the implementation on multiprocessor or multicore architectures. Let us describe one of the principles of the possible schemes of implementation of the algorithm for solving optimization problems on such architectures.

Let $M_1, M_2, \dots, M_k$ be a list of optimization methods, composed of algorithms in the software package of unconstrained optimization. It is reasonable to include in the list diverse methods, if the structure of the objective function is, generally speaking, not known.

The process for solving the problem is carried on in stages, each of which consists of training and working steps. The first of these steps is intended to identify the locally efficient algorithm from the available list of algorithms. After that, the working step is carried out, which consists in solving the problem using only the algorithm that has proven to be the most efficient in the first step. Both the training and working steps are carried out within a certain time slice. One can use two variants of the training step:

1. To determine the local efficiency of the methods, the optimization process starts from the same point x^0. In this case there is somewhat wasteful consumption of machine time, and the training step is only used to identify a locally efficient algorithm;
2. The training step is used not only to find an efficient algorithm, but also to advance to an extreme point, because instead of the original point we use the current point to train each of the following algorithm.

At the training step all the algorithms of the initial list $M_1, M_2, \dots, M_k$ have the opportunity to work within the given initial time slice (quantum) τ, with the exception of only those methods that have been the least efficient for two consecutive training steps. These methods are not allocated any time slice and are temporarily excluded from the list.

The value of the initial time quantum depends on the type of function being minimized, more precisely, on the time spent by the computing system on one computation of the objective function, and on the number of its variables, i.e. $\tau = \tau(n, \chi)$, where χ is the time spent on one computation of the function (in milliseconds). For example, the time quantum can be defined by the formula $\tau = (5n(n+1) + 20)\chi$. Here the number of computations of the function on one iteration by the Powell's conjugate directions method is assumed as a basis of determining the coefficient of χ. Powell's method requires $n(n+1)/2$ one-dimensional minimizations (minimizations in given directions), each of which uses on average 10 function computations.

As a result, the most effective algorithm is identified at the learning stage, which then is used during the working step. The duration of the working step $T_i = T_i(n, \chi, \tau)$ is selected as follows:

$$T_0 = \alpha\tau, \quad T_i = \delta_i T_{i-1} + T_0, \quad \alpha > 1. \qquad (1)$$

The quantity α depends on the complexity and dimension of the objective function, and is chosen arbitrarily. It is desirable to carry out tests at different values of α. The quantity δ_i is taken equal to 1 if the same method turned out to be the most effective at two consecutive stages. Otherwise, we take $\delta_i = 0$, i.e. the duration of the working step may increase if any method turned out to be the most effective at several successive stages. This means that in order to minimize the given objective function, we have found out the method that reaches an optimum point within a minimum time, and therefore for this case (in general, ideal) it makes no sense to conduct further training.

To calculate the values of the local efficiencies of the methods, we make use of the following formula:

$$E_i = \frac{|f(x^{k+1}) - f(x^k)|}{|f(x^k)| + \varepsilon} + \frac{\|x^{k+1} - x^k\|}{\|x^k\| + \varepsilon}.$$

Here E_i is the local efficiency of the ith algorithm; x^{k+1}, x^k are the final and initial points obtained using the ith algorithm; $f(x^{k+1}), f(x^k)$ are the values of the objective function at these points; $\|.\|$ is the Euclidean norm; ε is a small positive number.

If the value of the function within the time quantum does not decrease, the local efficiency of such an algorithm is considered equal to 0. If, at the training stage, all the methods from the list have displayed zero efficiency, further search stops and the procedure terminates. This situation is possible when the list of methods is not focused on solving the given problem (for example, the function is ravine-like, and the list consists of only coordinate-wise descent and gradient methods). Thus, the inclusion of different methods in the list will help to avoid such situations.

Note that the local zero efficiency of the method can appear in the method also in the case when an arithmetic interrupt occurs during its operation. Implementing exception handling in the software of the system, in this case it is necessary to provide a transition from the ith method to the following, $(i + 1)$th method from the list. And the ith method does not fully work for the time quantum allocated to it and obtains the value of local efficiency $E_i = 0$.

The cycle criterion of the proposed procedure is the fulfillment of the exit criteria for all methods. In conclusion, the user receives the accumulated information on the search process, which includes the optimal chain of methods that worked at the working steps; the total time of search for solutions; the values of the objective function, of the coordinates, and of local efficiencies of the methods obtained during the training step.

The schematic block diagram of the proposed procedure for solving the optimization problem on a single-core system is shown in figure 1.

2.2. The principle of parallel implementation on a multicore architecture

The simplest implementation of a multi-threaded version of the solution to the given optimization problem seems to be an approach that involves several threads independently performing operations of the sequential algorithm described above.

The solution to an unconstrained optimization problem is carried out in stages. At each stage, the following steps are implemented:

1. At the initial step, from the list of all available algorithms $M_1, M_2, \ldots, M_k$ of unconstrained optimization, we randomly select several algorithms $M_{s_1}, M_{s_2}, \ldots, M_{s_l}$, the

number l of which is equal to the number of CPU cores in the computer system (we can also take l as a multiple of the number of CPU cores installed in the system, e.g. $2N$, $3N$, etc.).

2. At the working step, we identify the most efficient algorithms. The duration T_i of the working step may increase if any method has proven to be the most efficient for several consecutive stages.

3. The current values of the local efficiencies E_i of the methods are calculated. From the list of working algorithms, we exclude a half of those who have exhibited the lowest efficiency.

4. To the list of working algorithms we then add as many other algorithms as were excluded in the previous step, and repeat steps 2 through 4 again.

In the procedure described above, all methods from the list are forcibly interrupted after the specified time slice; while for any method(s) the next started iteration might not complete until the end. A possible modification of this procedure is to enable all the methods to complete the iterations that have been started or to perform a whole number of iterations in the neighborhood of the given time interval. In the latter case, it becomes necessary to slightly change the formula for calculating the local efficiencies of methods.

The schematic block diagram of the proposed procedure for solving the optimization problem on a multicore system is shown in figure 2.

2.3. The principle of parallel implementation in a distributed computing environment

Unlike parallel systems, distributed systems have a hierarchical organization: they consist of heterogeneous nodes, each of which can, in turn, be a parallel system, i.e. one of the systems discussed above. It is natural to assume that the maximum efficiency is achieved when the computational process is organized in accordance with this hierarchy. With this approach, at each node, the computations are carried out according to the scheme most suitable for the given node. In fact, each node of the distributed system runs a separate application – a "solver", performing the operations of the selected optimization algorithm. Interaction between several applications (nodes) is organized at the next level of the hierarchy through a dedicated central control process – a "supervisor".

The first stage of solving an optimization problem in a distributed computing environment is the synthesis of the computational space that is formed by instances of solvers. With a large number of nodes, running applications manually can be quite time-consuming. Therefore, it is necessary to provide the ability to automate launching solvers by the supervisor. In this case, remote task launching tools can be used, which are provided by a specific system, e.g. SSH, grid services, etc.

The created computing space can be used to solve the problem. In the solution process, it is necessary to distribute the computations between solvers in order to maximize the performance of the application in a distributed environment. Data exchange between the solvers and supervisor can be implemented by means provided for interaction with a particular node. If it is possible to establish a direct network connection, the exchange methods based on TCP / IP protocols, such as the socket interface, are used. In some cases, data exchange with applications is possible only through the transfer of files using the middleware “grid”. Load balancing occurs at two levels: at the upper level, the supervisor distributes the computational load between the solvers, and at the lower level (within a computing node), the solver distributes the work between methods designed for a particular type of computing node.

Both approaches of the search algorithm proposed above enable automatic selection of an efficient fast-acting optimization method from the existing list to solve a particular problem due to self-learning of the methods used.

When working with the automatic and dialogue systems, the user, in accordance with the standard requirements, formulates an optimization problem in any programming language in the form of a module (dynamic link library), and then enters it into the system by specifying the full path to the created library file; using the directives (instructions), the user runs the most appropriate (in his/her opinion) algorithms of the library of modules, and tunes their various settings. The control program will then organize the interaction of the modules from the package; manages the input of the initial and current information; interpret the user's directives (instructions); load optimization modules into the computer memory dynamically; output the results of computations on the display (at the same time you can get results on a printer) in a prescribed form.

Analyzing the results of the computations, the user decides on the further calculations, thus obtaining the possibility to monitor the progress of solving the problem, to intervene promptly in the computation process, to choose the working methods, and to adjust, if necessary, their parameters. The user determines how often and in what form the results should be displayed on the screen, and then, using a predefined set of directives carries out calculations.

The developed optimization systems were created on a modular basis, taking into account the further expansion of their capabilities. The developed systems are equipped with an extensive library of optimization programs; the interactive service provided in the dialog system enables the user to manage the process of solving problems on a computer system, and depending on the current calculation results, to choose the most rational sequence of methods used (correcting the parameters of methods, if necessary), as well as to make changes in the formulation of the problems to be solved.

During the operation of the system, the user and system activity modes alternate. At the time of interruption of computations, the user performs any of the following:

- analysis and correction of the current state;
- computation of the values of the objective function, constraints, their gradients at the current point;
- the choice of solution method(s);
- adjustment of control parameters of the method(s), specification of stopping criteria and information delivery forms;
- launching method(s);
- fixing the current state in text (or binary) files.

When working with the automatic and dialog systems, the user, in accordance with standard requirements, prepares the optimization task in any programming language in the form of a module (library file with the .dll extension); enters it into the system by specifying the full path to the file of the created library; with the help of special directives (commands) calls from the library of modules the most suitable (from his point of view) algorithms and selects their parameters. The control program organizes the interaction of modules from the package, manages the input of the source and current information, interprets the user directives, dynamically loads optimization modules into the memory of the computer system, prints the results of computations in a user-specified form on the display screen (he can simultaneously output results to a disk file). Analyzing the results of computations, the user makes decisions on further computations, thus getting the opportunity to follow the progress of the problem solution, to promptly intervene in the computation process, to select methods, and to correct their parameters if necessary. The dialog system is a system with a directive input language. The user determines how often and in what form the results should be printed on the display screen, and then, using a predetermined set of directives, conducts computations.

3. Numerical Results

The results of computations depend to a great extent on the values of the control parameters; by carefully selecting them, one can significantly affect the course of computations and obtain some “record” results. Below we present the results of computation records implemented in the dialog mode. Computations were mainly conducted using the parameters of the methods provided for the “default” operation mode; only in a few cases we conducted two or three additional computations with other parameters and the best result is presented in the table. The data given in the tables are thus far from being “record”. To simplify the description of numerical experiments, the simplest test problems were taken. Therefore, the results of computations say little about the effectiveness of the methods used.

Comparative analysis of the methods should be carried out on the basis of solution to various and more complex problems. In order to bring the computations closer to reality to some extent, numerical differentiation of the functions to be optimized is used everywhere. The complexity of computations is determined basically by the number N, equal to the number of calls to the function (the calls necessary for the computation of derivatives are also taken into account).

Test 1. Consider the minimization of the function

$$F(x) = \sum_{i=1}^{49} \{100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\} \tag{2}$$

from different initial points. When minimizing this objective function, the following values of the optimization parameters were used: the accuracy of solution to the multidimensional minimization problem is $\varepsilon = 10^{-4}$; the accuracy of solution to the one-dimensional minimization problem is $\delta = 10^{-5}$; 0.5 seconds were allocated for the work of each algorithm in the list; to calculate the gradient and Hessian of the objective functional, we used a finite-difference formula with the approximation step $\Delta = 10^{-5}$.

When the function (2) is minimized from the initial point $\mathrm{x}^0 = (\mathrm{x}_\mathrm{i}^0 = 10.0,\ \mathrm{i} = 1,2,\dots,50)$ (the initial value of the function is $\mathrm{f}(\mathrm{x}^0) \approx 396903969$), we obtained the minimization process protocol given in Table 1, where all the implemented optimization algorithms were used.

TABLE 1. Result of algorithms used for optimization for function 1

Cycle	Method Function Value; Function Evaluation Count; Function Argument
1	Conjugate Gradient Polak-Ribiere (first-order method) 0.605243133940574; 3901; ($\{1.0002\}_{i=\overline{1,13}}$; $\{0.9999\}_{i=\overline{14,34}}$; 1.0007; 1.0018; 1.0011; 1.0014; 1.0017; 1.0017; 1.0031; 1.0049; 1.0084; 1.0169; 1.0332; 1.0672; 1.1376; 1.2895; 1.6638; 2.7669)
2	Newton Modification (second-order method) $1.66582504899541\times10^{-18}$; 80027; ($\{1.0000\}_{i=1,2,\dots,50}$)
3	Newton (second-order method) $2.51745236378655\times10^{-27}$; 10006; ($\{1.0000\}_{i=1,2,\dots,50}$)

When the function (2) is minimized from the initial point $x^0 = (x_i^0 = -26.0 + i,\ i = 1,2,\dots,50)$ (the initial value of the function is $f(x^0) \approx 3633898996$) we obtained the minimization process protocol given in Table 2, where all the implemented optimization algorithms were used.

TABLE 2. Result of algorithms used for optimization for function 2

Cycle	Method Function Value; Function Evaluation Count; Function Argument
1	Conjugate Gradient Polak-Ribiere (first-order method) 44.5426166226669; 4633; (0.9390; 0.8786; 0.7714; 0.5892; 0.3408; 0.1087; 0.0132; 0.0013; $\{\approx 0.0010\}_{i=\overline{9,49}}$; 0.0000)
2	Quasi-Newton Powell (second-order method) 32.0565470941584; 6411; ($\{\approx 1.0000\}_{i=\overline{1,10}}$; 0.9897; 0.9790; 0.9560; 0.9127; 0.8329; 0.6892; 0.4703; 0.2174; 0.0427; 0.0020; 0.0013; $\{\approx 0.0000\}_{i=\overline{22,50}}$)
3	Parallel Tangents (first-order method) 26.3052568352726; 6468; ($\{\approx 1.0000\}_{i=\overline{1,16}}$; 0.9859; 0.9712; 0.9423; 0.8867; 0.7847; 0.6139; 0.3748; 0.1387; 0.0180; $\{\approx 0.0000\}_{i=\overline{26,50}}$)
4	Powell (zero-order method) 17.588626941515; 66370; ($\{\approx 1.0000\}_{i=\overline{1,25}}$; 0.9832; 0.9671; 0.9366; 0.8751; 0.7685; 0.5883; 0.3477; 0.1145; 0.0018; $\{\approx 0.0010\}_{i=\overline{35,50}}$)

5	Parallel Tangents (first-order method) 7.61196779857186; 8500; ($\{\approx 1.0000\}_{i=\overline{1,34}}$; 0.9894; 0.9805; 0.9632; 0.9295; 0.8654; 0.7497; 0.5611; 0.3119; 0.0930; 0.0072; $\{\approx 0.0010\}_{i=\overline{45,49}}$; 0.0000)
6	Newton Modification (second-order method) 0.00817103023157141; 220064; ($\{\approx 1.0000\}_{i=\overline{1,46}}$; 0.9799; 0.9603; 0.9221; 0.8503)
7	Newton (second-order method) $1.7558039070914\times10^{-15}$; 40015; ($\{\approx 1.0000\}_{i=\overline{1,50}}$)

Test 2. Consider the minimization of the function

$$F(x) = \sum_{i=1}^{29}\{e^i(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\} \tag{3}$$

from different initial points. When minimizing this objective function, the following values of the optimization parameters were used: the accuracy of solution to the multidimensional minimization problem is $\varepsilon = 10^{-9}$; the accuracy of solution to the one-dimensional minimization problem is $\delta = 10^{-10}$; 6 seconds were allocated for the work of each algorithm in the list; to calculate the gradient and Hessian of the objective functional, we used a finite-difference formula with the approximation step $\Delta = 10^{-5}$.

When the function (3) is minimized from the initial point $x^0 = (x_i^0 = 10.0,\ i = 1,2,\ldots,30)$ (the initial value of the function is $f(x^0) \approx 5.03\times10^{16}$) we obtained the minimization process protocol given in Table 3, where all the implemented optimization algorithms were used.

TABLE 3. Result of algorithms used for optimization for function 3

Cycle	Method Function Value; Function Evaluation Count; Function Argument
1	Hooke-Jeeves (zero-order method) 10.4011697177792; 17181; ($\{\approx 1.0000\}_{i=\overline{1,21}}$; 1.0109; 1.0220; 1.0444; 1.0909; 1.1902; 1.4166; 2.0068; 4.0276; 16.2219)
2	Newton (second-order method) 0.203044664702809; 4914490; ($\{\approx 1.0000\}_{i=\overline{1,23}}$; 1.0105; 1.0212; 1.0428; 1.0876; 1.1829; 1.3993; 1.9581)
3	Newton (second-order method) 0.0267062905278361; 5343247; ($\{\approx 1.0000\}_{i=\overline{1,25}}$; 1.0166; 1.0336; 1.0684; 1.1415; 1.3030)

4	Newton (second-order method) 0.00668677461243312; 6510619; $(\{\approx 1.0000\}_{i=\overline{1,26}}$; 1.0173; 1.0349; 1.0710; 1.1471)
5	Newton-Raphson (second-order method) 0.00095164753562332; 6023236; $(\{\approx 1.0000\}_{i=\overline{1,27}}$; 1.0132; 1.0267; 1.0542)
6	Newton-Raphson (second-order method) $2.01752516177751\times10^{-5}$; 5957248; $(\{\approx 1.0000\}_{i=\overline{1,27}}$; 1.0019; 1.0038; 1.0078)
7	Newton-Raphson (second-order method) $4.22645477506483\times10^{-10}$; 10004512; $(\{\approx 1.0000\}_{i=\overline{1,30}})$

When the function (3) is minimized from the initial point $x^0 = (x_i^0 = -16.0 + i,\ i = 1,2,\dots,30)$ (the initial value of the function is $f(x^0) \approx 1.27\times10^{17}$) we obtained the minimization process protocol given in Table 4, where all the implemented optimization algorithms were used.

TABLE 4. Result of algorithms used for optimization for function 3 in the initial point x^0

Cycle	Method Function Value; Function Evaluation Count; Function Argument
1	Newton Modification (second-order method) 0.386688225037339; 6625915; $(\{\approx 1.0000\}_{i=\overline{1,22}}$; 0.9889; 0.9779; 0.9564; 0.9147; 0.8368; 0.7002; 0.4903; 0.2404)
2	Newton Modification (second-order method) 0.0926635390824508; 6017008; $(\{\approx 1.0000\}_{i=\overline{1,24}}$; 0.9815; 0.9634; 0.9281; 0.8614; 0.7421; 0.5507)
3	Newton Modification (second-order method) 0.00226517847677732; 7411369; $(\{\approx 1.0000\}_{i=\overline{1,26}}$; 0.9895; 0.9792; 0.9589; 0.9195)
4	Newton Modification (second-order method) 0.000282974896353102; 7984246; $(\{\approx 1.0000\}_{i=\overline{1,28}}$; 0.9854; 0.9711)
5	Newton-Raphson (second-order method) $7.35668971593544\times10^{-6}$; 4413862; $(\{\approx 1.0000\}_{i=\overline{1,30}})$
6	Newton-Raphson (second-order method) $4.45892579296002\times10^{-8}$; 4952764; $(\{\approx 1.0000\}_{i=\overline{1,30}})$
7	Newton-Raphson (second-order method) $1.95412491898793\times10^{-10}$; 5429344; $(\{\approx 1.0000\}_{i=\overline{1,30}})$

4. Conclusion

In the paper, we proposed an approach to control of computational process for solving complex applied problems by an example of multivariate unconstrained optimization problems using appropriate software packages on multi-processor (multi-core) computer systems. The proposed approaches essentially facilitate the end-users' work of using existing standard software packages. They require a different level of users' knowledge of methods implemented in the software packages.

References

[1]. Evtushenko, Yu.G. (1982) *Methods of solving extreme problems and their application in optimization systems.* Moscow: Nauka.

[2]. Evtushenko, Yu.G., Mazurik V.P. (1989) *Optimization systems software.* Moscow: Znanie.

[3]. Aidazade, K.R., Sidorenko, N.S. (1982) An approach to the construction of combined optimization algorithms. *Technical Cybernetics,* 6, 87-93.

[4]. Aidazade, K.R., Novruzbekov, I.G. (1987) Dialog system of multicriteria optimization. *Proceedings of the Academy of Sciences of Azerbaijan SSR, series of Physical-Technical and Mathematical Sciences, 2.*

[5]. Vasilyev, F.P. (2011) *Optimization methods, 1, 2.* Moscow: MTsNMO.

[6]. Polyak, B.T. (2014) *Introduction to optimization.* Moscow: Lenand, 2014.

[7]. Nocedal, J., Wright S. (Ed.) (2003) *Numerical optimization.* NY: Springer.

[8]. Baldick, R. (2009) *Applied optimization: formulation and algorithms for engineering systems.* Cambridge University Press.

[9]. Larichev, O.I., Horvitz G.G. (1990) *Methods of searching for a local extremum of ravine functions.* Moscow: Nauka, 1990.

[10]. Dongarra, J., Foster I., Fox G.C. (2003) *The Sourcebook of Parallel Computing.* San Francisco: Morgan Kaufmann Publishers.

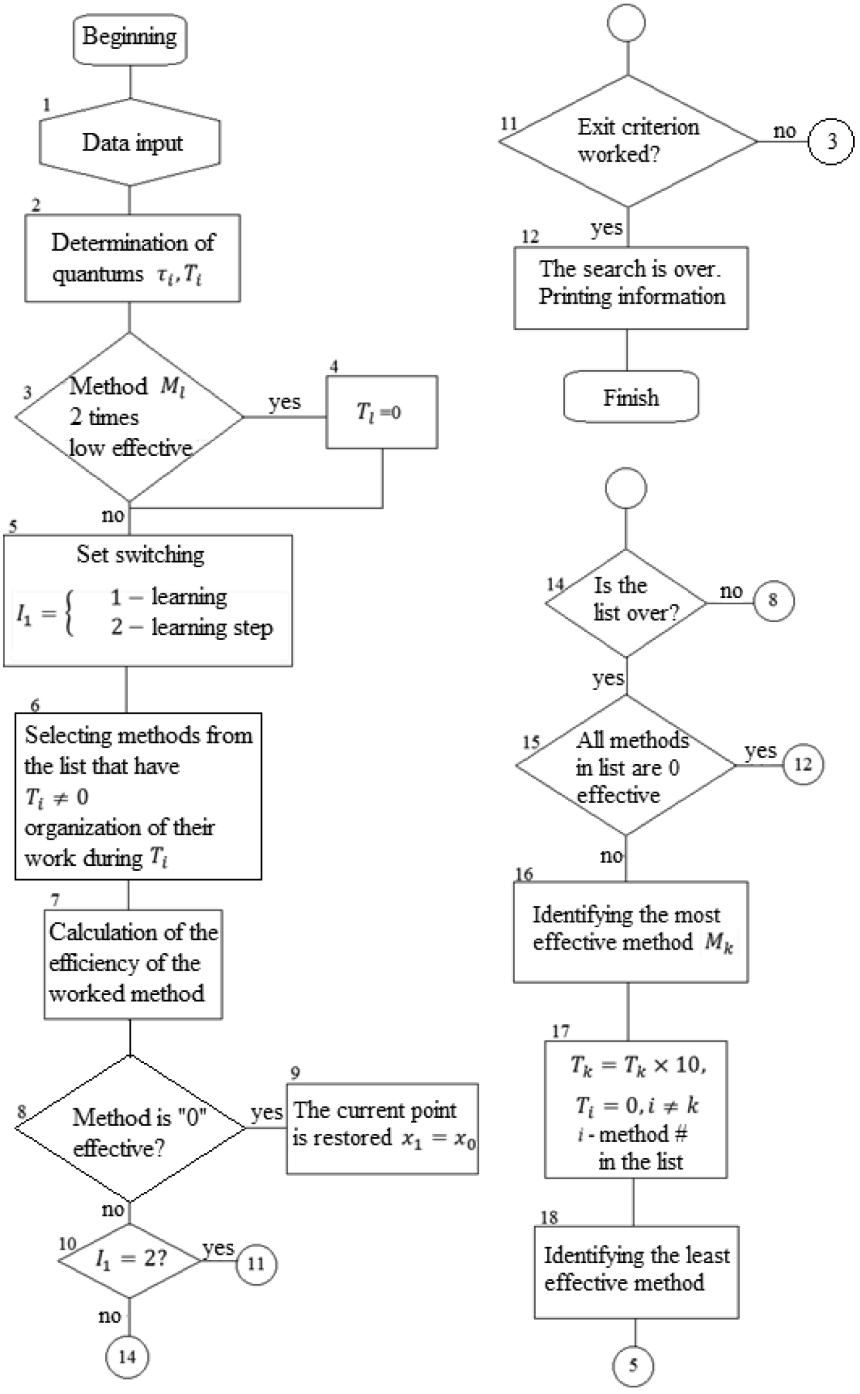
Beginning
1 Data input
2 Determination of quantums τ_i, T_i
3 Method M_l 2 times low effective
yes
4 $T_l = 0$
no
5 Set switching $I_1 = \begin{cases} 1 - \text{learning} \\ 2 - \text{learning step} \end{cases}$
6 Selecting methods from the list that have $T_i \neq 0$ organization of their work during T_i
7 Calculation of the efficiency of the worked method
8 Method is "0" effective?
yes
9 The current point is restored $x_1 = x_0$
no
10 $I_1 = 2?$
yes
11
no
14
11 Exit criterion worked?
no
3
yes
12 The search is over. Printing information
Finish
14 Is the list over?
no
8
yes
15 All methods in list are 0 effective
yes
12
no
16 Identifying the most effective method M_k
17 $T_k = T_k \times 10$, $T_i = 0, i \neq k$ i - method # in the list
18 Identifying the least effective method
5

Figure 1. Block diagram of the procedure for finding the optimum point

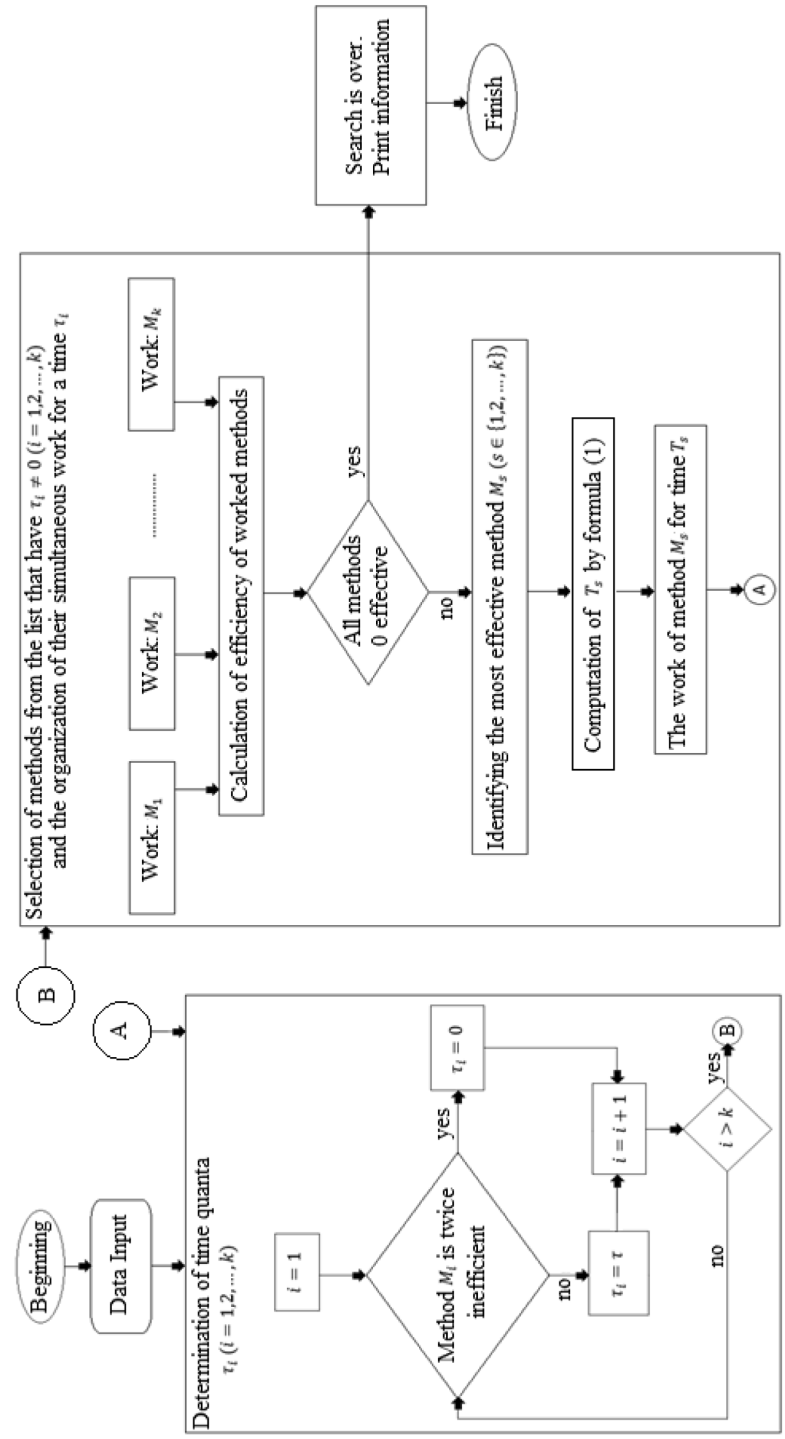
Beginning
Data Input
Determination of time quanta τ_i $(i = 1,2,\ldots,k)$
$i = 1$
Method M_i is twice inefficient
yes
no
$\tau_i = 0$
$\tau_i = \tau$
$i = i + 1$
$i > k$
yes
no
A
B
Selection of methods from the list that have $\tau_i \neq 0$ $(i = 1,2,\ldots,k)$ and the organization of their simultaneous work for a time τ_i
Work: M_1
Work: M_2
Work: M_k
Calculation of efficiency of worked methods
All methods 0 effective
yes
no
Identifying the most effective method M_s $(s \in \{1,2,\ldots,k\})$
Computation of T_s by formula (1)
The work of method M_s for time T_s
Search is over. Print information
Finish

Figure 2. Block diagram of the procedure for finding the optimum point

Submitted 21.02.2018
Accepted 27.05.2018