ComparisonOfSeveralSequentialAndParallelDerivative-freeGlobalOptimizationAlgorithms

Vladislav Sovrasov(✉) and Semen Bevzuk

Lobachevsky State University of Nizhni Novgorod, Russia
sovrasov.vlad@gmail.com, semen.bevzuk@gmail.com

Abstract. This work considers several stochastic and deterministic derivative-free global optimization algorithms. In the first part of the paper popular sequential open-source solvers are compared against Globalizer solver, which is developed at the Lobachevsky State University. The Globalizer is designed to solve problems with black-box objectives satisfying the Lipschitz condition and shows competitive performance with other similar solvers. The comparison is done on several sets of challenging multi-extremal benchmark functions. The second part of this work is devoted to comparison between the Globalizer and MIDACO solvers on systems with shared and distributed memory. MIDACO is a state-of-the-art global solver included to the TOMLAB optimization environment for MATLAB. Results of the benchmark show advantages of the Globalizer on small-dimensional, but sufficiently multi-extremal benchmark functions.

Keywords: deterministic global optimization · stochastic global optimization · parallel numerical methods · derivative-free algorithms · black-box optimization

1 Introduction

Nonlinear global optimization of non-convex functions is considered to be one of the most difficult problems of mathematical programming. The finding of the global minimum of a function of several variables often appears to be more difficult than local optimization in a thousand-dimensional space. In local optimization applying of the simplest gradient descent method may appear to be enough, whereas in order to *guarantee* finding the global optimum, an optimization method has to accumulate information about behavior of the objective function within the whole search domain [3, 12, 21, 32]. Nowadays, various stochastic global optimization algorithms have attracted much attention, first of all, the evolution ones [14, 25, 28]. They have rather simple structure and allow solving the problems of large dimensionality. However, the evolution algorithms ensure the global convergence in the probabilistic sense only.

In the present paper, the open-source implementations of nine different global optimization methods were considered. All the algorithms were tested using a

set of 900 essentially multiextremal functions, which has been generated using generators of test problems [5, 9]. Besides the comparison of the sequential algorithms, a comparison of the MIDACO solver [24] with the Globalizer software system [8, 30] on a subset of 200 test problems has been performed in the Lobachevsky supercomputer environment.

2 Related Work

Earlier, the comparison of the stochastic global optimization algorithms [1, 20] as well as of the deterministic ones [15, 17, 22] between each other has been considered in the literature. In these works, most of modern methods have been studied in details, but the emphasis was made mainly on the sequential algorithms while the reliability and the convergence speed were the main criteria of efficiency of the optimization methods. In the majority of works, the sets of well-known test problems (for example, the Rastrigin function, Ackley function, etc.) were taken as the sets of test functions. The sizes of such sets don't exceed 100 different functions usually, some of which can be the single-extremal ones (such as the Rosenbrock function).

In Ref. [2], some general principles were formulated, which, in the author's opinion, should be obeyed when comparing the optimization methods. In particular, the authors say about the advantages of the problem generators allowing generating the large sets of problems thus minimizing the random effects when comparing the methods. At the same time, the use of a single generator can appear to be not enough for a comprehensive comparison of the methods. In order to overcome this problem in part, the authors of Ref. [2] advise to use several generators of various nature and to create the sets of problems of various complexity.

Taking into account the experience of the preceding works in the field of comparison of the optimization methods, two generators of the test problems of different nature will be used in the present work. Using these ones, 9 sets of 100 problems of various complexity with the dimensionality varying from 2 to 5 were generated. Besides the comparison of the sequential methods, a comparison of the efficiencies of two parallel algorithms is presented in the work as well.

3 Statement of Multidimensional Global Optimization Problem

In this paper, the core class of optimization problems, which can be solved using global optimization methods, is formulated. This class involves the multidimensional global optimization problems without constraints, which can be defined in the following way:

$$\begin{gathered} \varphi(y^*) = \min\{\varphi(y) : y \in D\}, \\ D = \{y \in \mathbb{R}^N : a_i \le y_i \le b_i, 1 \le i \le N\} \end{gathered} \tag{1}$$

with the given boundary vectors a and b. It is supposed, that the objective function $\varphi(y)$ satisfies the Lipschitz condition

$$|\varphi(y_1) - \varphi(y_2)| \leq L\|y_1 - y_2\|, y_1, y_2 \in D, \tag{2}$$

where $L > 0$ is the Lipschitz constant, and $||\cdot||$ denotes the norm in $\mathbb{R}^N$ space.

Usually, the objective function $\varphi(y)$ is defined as a computational procedure, according to which the value $\varphi(y)$ can be calculated for any vector $y \in D$ (let us further call such a calculation *a trial*). It is supposed that this procedure is time-consuming.

4 Review of Considered Optimization Methods

4.1 Parallel Algorithm of Global Search

Dimension Reduction with Space-Filling Curves Within the framework of the information-statistical global optimization theory, the Peano space-filling curves (or *evolvents*) $y(x)$ mapping the interval $[0, 1]$ onto an N-dimensional hypercube D unambiguously are used for the dimensionality reduction [26,29,32].

As a result of the reduction, the initial multidimensional global optimization problem (1) is reduced to the following one-dimensional problem:

$$\varphi(y(x^*)) = \min\{\varphi(y(x)) : x \in [0, 1]\}. \tag{3}$$

It is important to note that this dimensionality reduction scheme transforms the Lipschitzian function from (1) to the corresponding one-dimensional function $\varphi(y(x))$, which satisfies the uniform Hölder condition, i. e.

$$|\varphi(y(x_1)) - \varphi(y(x_2))| \leq H|x_1 - x_2|^{\frac{1}{N}}, x_1, x_2 \in [0, 1], \tag{4}$$

where the constant H is defined by the relation $H = 2L\sqrt{N+3}$, L is the Lipschitz constant from (2), and N is the dimensionality of the optimization problem (1).

The algorithms for the numerical construction of the Peano curve approximations are given in [32].

The computational scheme obtained as a result of the dimensionality reduction consists of the following:

- The optimization algorithm performs the minimization of the reduced one-dimensional function $\varphi(y(x))$ from (3),
- After determining the next trial point x, a multidimensional image y is calculated by using the mapping $y(x)$,
- The value of the initial multidimensional function $\varphi(y)$ is calculated at the point $y \in D$,
- The calculated value $z = \varphi(y)$ is used further as the value of the reduced one- dimensional function $\varphi(y(x))$ at the point x.

Algorithm of Global Search on Shared Memory Parallel optimization methods applied in Globalizer to solve the reduced problem (3) are based on the MAGS method, which can be presented as follows — see [29], [32].

The initial iteration of the algorithm is performed at an arbitrary point $x^1 \in (0,1)$. Then, let us suppose that k, $k \geq 1$, optimization iterations have been completed already. The selection of the trial point x^{k+1} for the next iteration is performed according to the following rules.

Rule 1. Renumber the points of the preceding trials by the lower indices in order of increasing value of coordinates $0 = x_0 < x_1 < ... < x_{k+1} = 1$.

Rule 2. Compute the characteristics $R(i)$ for each interval $(x_{i-1}, x_i), 1 \leq i \leq k+1$.

Rule 3. Determine the p intervals with the maximum characteristics $R(t_j) = \max_{1 \leq i \leq k+1} R(i)$, $j = \overline{1,p}$.

Rule 4. Execute new trials at points x^{k+j}, $j = \overline{1,p}$ located within intervals with maximum characteristics from the previous step $x^{k+j} = d(x_{t_j})$, $j = \overline{1,p}$.

The stopping condition, which terminated the trials, is defined by the inequality $\rho_{t_j} < \varepsilon$, $j = \overline{1,p}$ for the intervals with maximum characteristics from Step 3 and $\varepsilon > 0$ is the predefined accuracy of the optimization problem solution. If the stopping condition is not satisfied, the index k is incremented by p, and the new global optimization iteration is executed.

This method is employed in Globalizer to organize parallel computations on shared memory: each of p trials can be carried out on one of p local computation units.

The convergence conditions and exact formulas for decision rules $R(i)$ and $d(x)$ of the described algorithm are given, for example, in [32].

4.2 Parallel Algorithm on Distributed Memory Exploiting a Set of Evolvents

Rotated Evolvents One of the possible ways to overcome the negative effects of using a numerical approximation of evolvent (it destroys the information about the neighbor points in $\mathbb{R}^N$ space) consists in using the multiple mappings

$$Y_L(x) = \{y^0(x),\ y^1(x), ...,\ y^L(x)\} \tag{5}$$

instead of single Peano curve $y(x)$ (see [32]). The building of a set of Peano curves by rotation of the evolvents around the coordinate origin is a distinctive feature was proposed in [31]. Taking into account the initial mapping, one can conclude that current implementation of the method allows to build up to $N(N-1)+1$ evolvents for mapping the N-dimensional domain onto the corresponding one-dimensional intervals. This method for building a set of mappings can be "scaled" easily to obtain more evolvents (up to 2^N) if necessary.

Using the multiple mapping allows solving initial problem (1) by parallel solving the problems

$$\min\{\varphi(y^s(x)) : x \in [0,1]\}, 1 \leq s \leq S,$$

on a set of intervals $[0,1]$ by the index method. Each one-dimensional problem is solved on a separate processor. The trial results at the point x^k obtained for the problem being solved by particular processor are interpreted as the results of the trials in the rest problems (in the corresponding points $x^{k_1},\ldots,x^{k_S}$). In this approach, a trial at the point $x^k \in [0,1]$ executed in the framework of the s^{th} problem, consists in the following sequence of operations:

1. Determine the image $y^k = y^s(x^k)$ for the evolvent $y^s(x)$.

2. Inform the rest of processors about the start of the trial execution at the point y^k (the blocking of the point y^k).

3. Determine the preimages $x^{k_s} \in [0,1], 1 \leq s \leq S$, of the point y^k and interpret the trial executed at the point $y^k \in D$ as the execution of the trials in the S points $x^{k_1},\ldots,x^{k_s}$

4. Inform the rest of processors about the trial results at the point y^k.

The decision rules for the mentioned parallel algorithm, in general, are the same as the rules of the sequential algorithm (except the method of the trial execution). Each processor has its own copy of the software realizing the computations of the problem functions and the decision rule of the index algorithm. For the organization of the interactions among the processors, the queues are created on each processor, where the processors store the information on the executed iterations in the form of the tuples: the processor number s, the trial point x^{k_s}. The modifications of the method taking into account multiply criteria are given in [6,7].

The mentioned parallelization scheme was proposed in [31] and implemented in the Globalizer system with the use of MPI technology. Main features of implementation consist in the following:

- A separate MPI-process is created for each of S one-dimensional problems being solved, usually, one process per one processor employed.
- Each process can use p threads to parallel execute p trials, usually one thread per an accessible core.

4.3 MIDACO Parallel Solver

MIDACO (Mixed Integer Distributed Ant Colony Optimization) [24] is a solver which implements the evolutionary algorithm the ant colony optimization metaheuristic for continuous search domains. ACO was originally proposed to solve the TSP, but later has been successfully adopted to solve mixed integer nonlinear programming problems [25].

The MIDACO solver utilizes reverse communication architecture [24]. Reverse communication means that the call of the objective function happens outside and independently of the MIDACO source code. Relying on this feature the following distributed parallelization scheme was implemented:

1. MIDACO runs at the master node and generates $n \times p$ new trial points at each iteration.
2. Each of n distributed computational nodes gets p points from the master and performs p parallel trials on local computing devices.

3. Each of n distributed computational nodes sends p values of the objective function to the master.

In this scheme all distributed communications were implemented using the MPI library. Parallelism within a single node is powered by the OpenMP standard.

4.4 Sequential Methods

- **Algorithm of Global Search**. Sequential version of the method described in Section 4.1.
- **Locally-biased Algorithm of Global Search (AGSl)** [19]. It's a modification of the original AGS which make it more locally oriented by alternately using of two types of characteristics in the Rule 2 from the Section 4.1.
- **Multi Level Single Linkage** [13]. MLSL is an improved multistart algorithm. It samples low-discrepancy starting points and does local optimizations from them. In contrast to the dummy multistart schemes MLSL uses some clustering heuristics to avoid multiple local descents to already explored local minima.
- **DIRECT** [12]. The algorithm is deterministic and recursively divides the search space and forms a tree of hyper-rectangles (boxes). DIRECT uses the objective function values and the Lipschitz condition (2) to estimate promising boxes.
- **Locally-biased DIRECT (DIRECTl)** [4]. It's a variation of DIRECT which pays less attention to non-promising boxes and therefore has less exploration power: it can converge faster on problems with few local minima, but lost the global one in complicated cases.
- **Dual Simulated Annealing** [33]. This stochastic method is a combination of the Classical Simulated Annealing and the Fast Simulated Annealing coupled to a strategy for applying a local search on accepted locations. It converges much faster than both parent algorithms, CSA and FSA.
- **Differential Evolution** [28]. DE is an adaptation of the original genetic algorithm to the continuous search domain.
- **Controlled Random Search** [23]. The CRS starts with a set of random points and then defines the next trial point in relation to a simplex chosen randomly from a stored configuration of points. CRS in not an evolutional algorithm, although stores something like population and performs transformation resembling a mutation.
- **StoGO** [18]. StoGO is dividing the search space into smaller hyper- rectangles via a branch-and-bound approach, and searching them by a local-search algorithm, optionally including some randomness.

All the mentioned algorithms are available in source codes as parts of widespread optimization packages (except of AGS and AGSl, which are implemented in the Globalizer system). DIRECT, DIRECTl, CRS, MLSL and StoGO are part of the NLOpt library [10]. Differential Evolution and DSA can be found in the latest version of the SciPy [11] package for Python.

5 Tools for Comparison of Global Optimization Algorithms

The use of the sets of test problems with known solutions generated by some random mechanisms is one of commonly accepted approaches to comparing the optimization algorithms [2]. In the present work, we will use two generators of test problems generating the problems of different nature [5, 9].

Let us denote the problem set obtained with the use of the first generator from [9] as F_{GR}. The mechanism of generation of the problems F_{GR} doesn't provide the control of the problem complexity and of the number of local optima. However, the generated functions are known to be the multiextremal ones essentially. Besides, the problems generated by F_{GR} are the two-dimensional ones. In the present work, we will use 100 functions from the class F_{GR} generated randomly.

The GKLS generator [5] allows obtaining the problems of given dimensionality with given number of extrema. Moreover, GKLS allows adjusting the complexity of the problems by decreasing or increasing the size of the global minimum attractor. In [16] the parameters of the generator allowing generating the sets of 100 problems each of two levels of complexity (Simple and Hard) of the dimensionality equal to 2, 3, 4, and 5 are given. Following the authors of the GKLS generator, we will use the parameters proposed by them and, this way, add 800 more problems of various dimensionalities and complexity into the test problem set.

Let us suppose a test problem to be solved if the optimization method executes the scheduled trial y^k in a δ-vicinity of the global minimum y^*, i.e. $\left\|y^k - y^*\right\| \leq \delta = 0.01 \left\|b - a\right\|$, where a and b are the left and the right boundaries of the hypercube from (1). If this relation is not fulfilled before the expiration of the limit of the number of trials, the problem was considered to be unsolved. The limit of the number of trials was set for each problem class according to the dimensionality and complexity (see Table 1).

Table 1: Trials limits for the test problem classes

Problems class	Trials limit
F_{GR}	5000
GKLS 2d Simple	8000
GKLS 2d Hard	9000
GKLS 3d Simple	15000
GKLS 3d Hard	25000
GKLS 4d Simple	150000
GKLS 4d Hard	250000
GKLS 5d Simple	350000
GKLS 5d Hard	600000

Let us consider the averaged number of trials executed to solve a single problem and the number of solved problems as the characteristics of the optimization method on each class. The less the number of trials, the faster the method converges to a solution, hence the less times it turns to a potentially computation-costly procedure of computing the objective function. The number of solved problems evidences the reliability of the method at given parameters on the class of test problems being solved. In order to make independent the quantities featuring the reliability and the speed of convergence, averaged number of trials always was calculated taking into account solved problems only.

6 Results of Numerical Experiments

6.1 Sequential Algorithms

The results of various algorithms on different problem classes depend on the adjustments of algorithms directly. In most cases, the authors of software implementations are oriented onto the problems of medium difficulty. In order to obtain a satisfactory result when solving the essentially multiextremal problems, a correction of some parameters is required. When conducting the comparison, the following parameters for the methods were employed:

- in the AGSl method, the parameter of alternation the global and local iterations was set to be equal to 5:1;
- in the DIRECT and DIRECTl methods, the parameter $\epsilon = 10^{-4}$;
- in the SDA method, the parameter $visit = 2.72$.

The rest parameters were varied subject to the problem class (see Table 2).

Table 2: Class-specific parameters of the optimization algorithms

	AGS, AGSl	CRS	DE
F_{GR}	$r = 3$	popsize=150	mutation=(1.1,1.9), popsize=60
GKLS 2d Simple	$r = 4.6$	popsize=200	mutation=(1.1,1.9), popsize=60
GKLS 2d Hard	$r = 6.5$	popsize=400	mutation=(1.1,1.9), popsize=60
GKLS 3d Simple	$r = 3.7$	popsize=1000	mutation=(1.1,1.9), popsize=70
GKLS 3d Hard	$r = 4.4$	popsize=2000	mutation=(1.1,1.9), popsize=80
GKLS 4d Simple	$r = 4.7$	popsize=8000	mutation=(1.1,1.9), popsize=90
GKLS 4d Hard	$r = 4.9$	popsize=16000	mutation=(1.1,1.9), popsize=100
GKLS 5d Simple	$r = 4$	popsize=25000	mutation=(1.1,1.9), popsize=120
GKLS 5d Hard	$r = 4$	popsize=30000	mutation=(1.1,1.9), popsize=140

The results of running the optimization methods on the considered problem classes are presented in Tables 3, 4. The DIRECT and AGSl methods have demonstrated the best convergence speed on all classes, at that AGSl inferior

to DIRECT on the problems from the Simple classes and has an advantage on the problems of the Hard classes. As one can see from Table 4, the deterministic methods (AGS, AGSl, DIRECT, and DIRECTl) were the most reliable. Among the stochastic methods, MLSL and SDA have demonstrated the highest reliability.

Table 3: Averaged number of trials executed by sequential methods for solving the test optimization problems

	AGS	AGSl	CRS	DIRECT	DIRECTl	MLSL	SDA	DE	StoGO
F_{GR}	193.1	**158.3**	400.3	182.3	214.9	947.2	691.2	1257.3	1336.8
GKLS 2d Simple	254.9	217.6	510.6	**189.0**	255.2	556.8	356.3	952.2	1251.5
GKLS 2d Hard	728.7	**488.0**	844.7	985.4	1126.7	1042.5	1637.9	1041.1	2532.2
GKLS 3d Simple	1372.1	1195.3	4145.8	**973.6**	1477.8	4609.2	2706.5	5956.94	3856.1
GKLS 3d Hard	3636.1	**1930.5**	6787.0	2298.7	3553.3	5640.1	4708.4	6914.3	7843.2
GKLS 4d Simple	26654.1	11095.7	37436.8	**7824.3**	15994.1	41514.3	21417.9	19157.7	59895.4
GKLS 4d Hard	54536.8	**23167.8**	73779.3	23204.4	54489.9	80247.2	68815.5	27466.1	109328.1
GKLS 5d Simple	29810.0	11529.0	143575.0	**7166.5**	13970.5	52647.6	34255.3	73074.5	91580.4
GKLS 5d Hard	113129.1	67652.7	165192.8	**66327.4**	164390.6	138766.2	116973.1	105496.9	155123.8

Table 4: Number of test optimization problems solved by sequential methods

	AGS	AGSl	CRS	DIRECT	DIRECTl	MLSL	SDA	DE	StoGO
F_{GR}	100	100	76	100	100	97	96	96	67
GKLS 2d Simple	100	100	85	100	100	100	100	98	90
GKLS 2d Hard	100	100	74	100	100	100	93	85	77
GKLS 3d Simple	100	97	75	100	100	100	89	86	44
GKLS 3d Hard	100	99	72	100	99	100	88	77	43
GKLS 4d Simple	100	100	46	100	100	94	78	59	16
GKLS 4d Hard	100	100	47	99	97	94	72	32	10
GKLS 5d Simple	100	100	68	100	100	98	100	77	9
GKLS 5d Hard	97	99	42	100	90	79	84	48	8

6.2 Parallel Algorithms

Computational experiments with parallel algorithms have been carried out on the Lobachevsky supercomputer at State University of Nizhni Novgorod. A computational node includes 2 Intel Sandy Bridge E5-2660 2.2 GHz processors, 64 GB RAM. Each CPU has 8 cores (i. e. total 16 cores were available per a node).

All the computational nodes are working under CentOS Linux 7.2. All the considered algorithms are implemented using C++ and compiled by the GCC 4.8.5.

The comparison of two parallel algorithms implemented in the Globalizer and MIDACO systems was conducted on four-dimensional GKLS problems since the problems of larger dimensionality are not supported by the free version of MIDACO. As in the case of the sequential algorithms, the parameters of methods were determined for each problem class. For MIDACO, the parameter $focus = -1$. In the case of Globalizer, the parameter r was selected following Table 2. This parameter decreased by 0.3 with doubling the number of employed evolvents since the increasing of the number of evolvents enhances the reliability of the method from Section 4.2 as well (see [27] for details).

If one considers the costs of parallelism to be negligible as compared to the costs of computing the objective functions in the optimization problems, the speedup in time due to the use of the parallel method would be equal to the speedup with respect to the number of iterations. However, actually this assumption is true if the computing of the objective function is well computation costly only. In all numerical experiments, the time of computing the objective function was approximately 2.5×10^{-3} s.

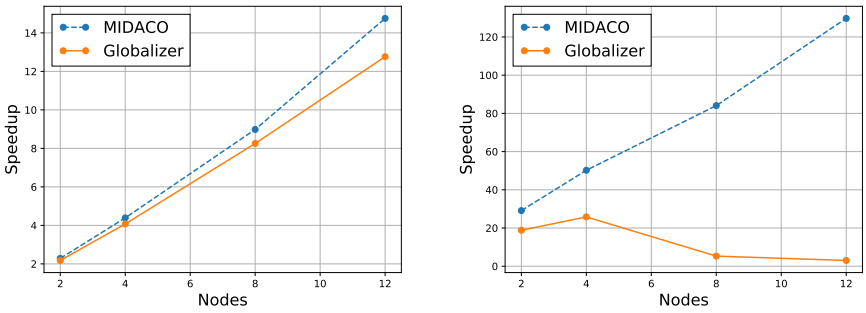
In Table 5, an averaged number of iterations when solving 100 problems from each considered class is presented. The number of iterations is reduced considerably with increasing the number of nodes and the number of threads on each node. Also, the number of solved problems in each class is given in brackets. Globalizer solves almost all problems in both classes whereas MIDACO hasn't solved a considerable number of problems from the GKLS 4d Hard class. Also, it is worth noting that the reliability of MIDACO decreased with increasing number of nodes on both problem classes whereas the reliability of Globalizer remains stable.

The averaged speedup in the number of iterations and the one in time (in brackets) are presented in Table 6. In the first row of Table 6, the averaged number of iterations and the averaged time of solving the problems in the sequential mode (in brackets) are presented. MIDACO uses a smaller number of communications in the parallel computations than the Globalizer. Therefore, the indicators of speedup increased monotonously with increasing number of computational nodes employed. In Globalizer, each node should transfer the results of the executed iterations to all other nodes. Therefore, the efficiency of parallelization decreased with increasing number of nodes and the number of active threads on each node. This is evident from Table 6 and Picture 1b: the speedup in time decreased at the transition from 4 nodes to 8 ones when using 16 threads. In the case of using 1 thread per a node, the speedups of both optimization methods were approximately the same (see Picture 1b). However, it should be noted that Globalizer executed less number of trials, and spends much less time for solving all problems when $p = 1$. With increasing number of nodes and threads, MIDACO approaches Globalizer in the averaged time of problem solving (see Pictures 2a, 2b), however, its reliability decreases at the same time.

Table 5: Averaged numbers of iterations executed by the parallel algorithms for solving the test optimization problems

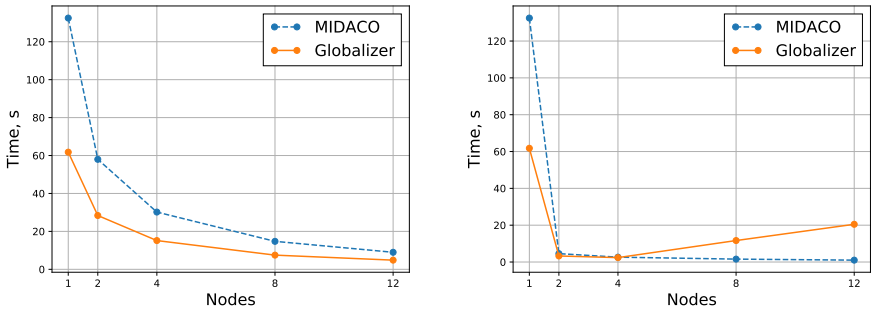
		p	Globalizer		MIDACO	
			Simple	*Hard*	*Simple*	*Hard*
I	**1 cluster node**	*1*	25270 (100)	55180 (99)	27645 (98)	72068 (71)
		16	1765 (100)	3714 (100)	1640 (97)	4304 (70)
II	**2 cluster nodes**	*1*	13056 (100)	22938 (99)	10558 (89)	27128 (73)
		16	732 (100)	1759 (100)	1130 (92)	2254 (73)
III	**4 cluster nodes**	*1*	5016 (100)	12703 (100)	5777 (94)	15980 (75)
		16	367 (100)	776 (100)	529 (88)	1264 (66)
VI	**8 cluster nodes**	*1*	2103 (100)	5063 (100)	2847 (97)	9853 (83)
		16	145 (100)	310 (100)	272 (83)	774 (57)
V	**12 cluster nodes**	*1*	1155 (100)	2399 (100)	2233 (98)	6022 (86)
		16	76 (100)	159 (100)	168 (87)	393 (53)

An anomalous superlinear speedup of MIDACO in time on the GKLS 2d Hard class when employing 8 and 12 nodes can be explained by the fact that, according to Table 5, MIDACO has solved a greater number of problems in the parallel mode than in the sequential one, hence, has worked until the limit of trials is expired on a less number of problems.



(a) $p = 1$ (b) $p = 16$

Fig. 1: Speedup demonstrated by the parallel algorithms when solving problems from the GKLS 4d Hard class



(a) $p = 1$ (b) $p = 16$

Fig. 2: Averaged execution time of the parallel algorithms when solving problems from the GKLS 4d Hard class

7 Conclusions

In the present paper, several sequential and parallel global optimization algorithms were considered. A comparison of efficiencies of these ones has been done on a set of test problems. The results allow making the following conclusions:

- AGSl method implemented in the Globalizer system has demonstrated the convergence speed and reliability at the level of DIRECT and exceeds many other algorithms, the open-source implementations of which are available;
- the stochastic optimization methods inferior to the deterministic ones in the convergence speed and in reliability. It is manifested especially strongly on more complex multiextremal problems;
- the parallel version of the Globalizer system demonstrates good speedup when running on several nodes, on each of which a single objective function value per iterations is computed. When solving the problems with fast computable objective functions and using multiple threads on the nodes, the speedup for the Globalizer system degrade with increasing number of nodes;
- the MIDACO system is the most suitable for simple global optimization problems with the fast-computable objective functions. In this case, MIDACO is reliable enough and provides a linear speedup with increasing number of nodes and threads executed on these ones in parallel.

Acknowledgements

The study was supported by the Russian Science Foundation, project No 16-11-10150.

Table 6: Speedup of parallel computations executed by the parallel algorithms

		p	Globalizer		MIDACO	
			Simple	*Hard*	*Simple*	*Hard*
I	**1 cluster node**	*1*	25270 (27.3s)	55180 (61.8s)	27645 (32.2s)	72068 (132.5s)
		16	14.3(11.7)	14.9(12.6)	16.9(14.4)	16.7(14.4)
II	**2 cluster nodes**	*1*	1.9(1.9)	2.4(2.2)	2.6(1.7)	2.7(2.3)
		16	34.5(20.4)	31.4(18.8)	24.5(18.8)	32.0(29.1)
III	**4 cluster nodes**	*1*	5.0(4.6)	4.3(4.1)	4.8(3.9)	4.5(4.4)
		16	68.8(23.7)	71.2(25.8)	52.3(32.9)	57.0(50.2)
VI	**8 cluster nodes**	*1*	12.0(8.7)	10.9(8.3)	9.7(8.2)	7.3(9.0)
		16	174.1(4.7)	177.8(5.3)	101.6(51.6)	93.1(84.1)
V	**12 cluster nodes**	*1*	21.9(11.4)	23.0(12.8)	12.4(11.1)	12.0(14.8)
		16	333.5(2.7)	347.9(3.0)	164.5(82.0)	183.2(129.7)

References

1. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization **31**(4), 635–672 (2005)
2. Beiranvand, V., Hare, W., Lucet, Y.: Best practices for comparing optimization algorithms. Optimization and Engineering **18**(4), 815–848 (2017)
3. Evtushenko, Y., Posypkin, M.: A deterministic approach to global box-constrained optimization. Optim. Lett. **7**, 819–829 (2013)
4. Gablonsky, J.M., Kelley, C.T.: A locally-biased form of the direct algorithm. J. Glob. Optim. **21**(1), 27–37 (2001)
5. Gaviano, M., Kvasov, D.E., Lera, D., Sergeev, Ya.D.: Software for generation of classes of test functions with known local and global minima for global optimization. ACM Transactions on Mathematical Software **29**(4), 469–480 (2003)
6. Gergel, V., Kozinov, E.: Accelerating parallel multicriterial optimization methods based on intensive using of search information. Procedia Computer Science **108**, 1463 – 1472 (2017). International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland
7. Gergel, V., Kozinov, E.: Efficient multicriterial optimization based on intensive reuse of search information. J. Glob. Optim. **71**(1), 73–90 (2018)
8. Gergel V.P., Barkalov K.A., and Sysoyev A.V.: A novel supercomputer software system for solving time-consuming global optimization problems. Numerical Algebra, Control & Optimization **8**(1), 47–62 (2018)
9. Grishagin, V.: Operating characteristics of some global search algorithms. Problems of Statistical Optimization **7**, 198–206 (1978 (In Russian))
10. Johnson, S.G.: The nlopt nonlinear-optimization package. URL http://ab-initio.mit.edu/nlopt. [Online; accessed ¡24.12.2018¿]
11. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–). URL http://www.scipy.org/. [Online; accessed ¡24.12.2018¿]

12. Jones, D.R.: The direct global optimization algorithm. In: The Encyclopedia of Optimization, pp. 725–735. Springer, Heidelberg (2009)
13. Kan, A.H.G.R., Timmer, G.T.: Stochastic global optimization methods part ii: Multi level methods. Math. Program. **39**, 57–78 (1987)
14. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942–1948 vol.4 (1995)
15. Kvasov, D.E., Mukhametzhanov, M.S.: Metaheuristic vs. deterministic global optimization algorithms: The univariate case. Appl. Math. Comput. **318**, 245 – 259 (2018)
16. Lera, D., Sergeyev, Y.: An information global minimization algorithm using the local improvement technique. J. Glob. Optim. **48**(1), 99–112 (2010)
17. Liberti, L., Kucherenko, S.: Comparison of deterministic and stochastic approaches to global optimization. International Transactions in Operational Research **12**, 263 – 285 (2005)
18. Madsen, K., Zertchaninov, S.: A new branch-and-bound method for global optimization (1998)
19. Markin, D.L., Strongin, R.G.: A method for solving multi-extremal problems with non-convex constraints, that uses a priori information about estimates of the optimum. Comp. Math. Math. Phys. **27(1)**, 33–39 (1987)
20. Mullen, K.: Continuous global optimization in r. Journal of Statistical Software, Articles **60**(6), 1–45 (2014)
21. Paulavivcius, R., Zilinskas, J., Grothey, A.: Parallel branch and bound for global optimization with combination of lipschitz bounds. Optim. Method. Softw. **26**(3), 487–498 (1997)
22. Pok, P., Huyer, W., Pl, L.: A comparison of global search algorithms for continuous black box optimization. Evolutionary Computation **20**(4), 509–541 (2012)
23. Price, W.L.: Global optimization by controlled random search. Journal of Optimization Theory and Applications **40**(3), 333–348 (1983)
24. Schlueter, Martin: Nonlinear mixed integer based optimization technique for space applications. Ph.D. thesis, The University of Birmingham (2012)
25. Schluter, M., Egea, J.A., Banga, J.R.: Extended ant colony optimization for non-convex mixed integer nonlinear programming. Computers & Operations Research **36**(7), 2217 – 2229 (2009)
26. Sergeyev, Y.D., Strongin, R.G., Lera, D.: Introduction to global optimization exploiting space-filling curves. Springer Briefs in Optimization, Springer, New York (2013)
27. Sovrasov, V.: Comparison of dimensionality reduction schemes for derivative-free global optimization algorithms. Procedia Computer Science **136**, 136 – 143 (2018). 7th International Young Scientists Conference on Computational Science, YSC2018, 02-06 July2018, Heraklion, Greece
28. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997)
29. Strongin, R.: Numerical Methods in Multiextremal Problems (Information-StatisticalAlgorithms). Moscow: Nauka (In Russian) (1978)
30. Strongin, R.G., Gergel, V.P., Barkalov, K.A., Sysoyev, A.V.: Generalized parallel computational schemes for time-consuming global optimization. Lobachevskii Journal of Mathematics **39**(4), 576–586 (2018)

31. Strongin, R.G., Gergel, V.P., Barkalov, K.A.: Parallel methods for global optimization problem solving. Journal of instrument engineering **52**, 25–33 (2009 (In Russian))
32. Strongin R.G., Sergeyev Ya.D.: Global optimization with non-convex constraints. Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht (2000)
33. Xiang, Y., Sun, D., Fan, W., Gong, X.: Generalized simulated annealing algorithm and its application to the thomson model. Physics Letters A **233**(3), 216 – 220 (1997)