Comparison of several sequential and parallel derivative-free global optimization algorithms

Vladislav Sovrasov(✉) and Semen Bevzuk

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

Аннотация 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 the 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

Нелинейная глобальная оптимизация невыпуклых функций традиционно считается одной из самых трудных задач математического программирования. Отыскание глобального минимума функции от нескольких переменных зачастую оказывается сложнее, чем локальная оптимизация в тысячемерном пространстве. Для последней может оказаться достаточно применения простейшего метода градиентного спуска, в то время как чтобы *гаранти-рованно* отыскать глобальный оптимум методам оптимизации приходится накапливать информацию о поведении целевой функции во всей области поиска [3, 12, 20, 32]. В последнее время стали популярны различные стохастические алгоритмы глобально оптимизации, прежде всего эволюционные [14, 24, 28]. Они имеют довольно простую структуру, позволяют решать задачи большой размерности, однако обеспечивают глобальную сходимость только в вероятностном смысле.

В данной работе рассмотрены open-source реализации девяти различных методов глобальной оптимизации. Все алгоритмы были протестированы на

наборе из 900 существенно многоэкстремальных функций, который был сгенерирован с помощью специализированных генераторов задач [5,9]. Помимо сравнения последовательных алгоритмов, на подмножестве из 200 тестовых задач произведено сравнение солвера MIDACO [23] и программной системы Globalizer [8, 30] в условиях работы на суперкомпьютере Лобачевский.

2 Related Work

Ранее в литературе рассмартивалось как сравнение между собой стохастических алгоритмов глобальной оптимизации [1, 19], так и детерминированных [15, 16, 21]. В них большинство современных методов изучены довольно детально, но упор в основном делается на последовательные алгоритмы, а основными критериями эффективности метода оптимизации является его надёжность и скорость сходимости. В большинстве работ в качестве набора тестовых функций берётся набор известных тестовых задач (например, функция Растригина, Ackley function и др.). Размер такого набора обычно не превышает 100 различных функций, некоторые из которых могут быть одноэкстремальны (как функция Розенброка).

В работе [2] сформулированы некие общие принципы, которых, по мнению авторов, следует придерживаться при сравнении методов оптимизации. В частности, авторы говорят о преимуществах генераторов задач, позволяющих создать большие наборы задач, сводя случайные эффекты при сравнении к минимуму. В то же время, использование одного генератора может оказаться недостаточно для исчерпывающего сравнения методов. Чтобы частично преодолеть эту проблему, авторы [2] советуют использовать несколько генераторов различной природы и создавать наборы задач различной сложности.

Учитывая опыт предыдущих работ в области сравнения методов оптимизации, в данной работе будут использованы два генератора тестовых задач разной природы, с помощью которых сгенерировано 9 наборов по 100 задач разной сложности размерности от 2 до 5. Помимо сравнения последовательных методов, также в работе приводится сравнение эффективности двух параллельных алгоритмов.

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:

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

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 Evolvents 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 descision 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

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

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 $x^{k_s} \in [0,1], 1 \leqslant s \leqslant S$, 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) [23] 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 [24].

The MIDACO solver utilizes reverse communication architecture [23]. 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-based Algorithm of Global Search (AGSl)** [18]. 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 minimas.
- **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-based 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 minimas, 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** [22]. 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** [17]. 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 (except of AGSl which implemented in the Globalizer system) are available in source codes as parts of wide-spread optimization packages. AGS, DIRECT, DIRECTl, CRS, MLSL and StoGO are part of the NLOpt libray [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

Использование сгенерированных некоторыми случайными механизмами наборов тестовых задач с известными решениями является одним из общепринятых подходов к сравнению алгоритмов оптимизации [2]. В данной работе будем использовать два генератора тестовых задач, порождающих задачи различной природы [5, 9].

Обозначим набор задач, полученных с помощью первого генератора из [9] как F_{GR}. Механизм построения задач F_{GR}, не предусматривает контроль за сложностью задач и количеством локальных оптимумов, однако известно, что порождаемые функции являются существенно многоэкстремальными. Кроме того, задачи, порождаемые F_{GR} двухмерные. В данной работе будем использовать 100 случайно сгенерированных функций из класса F_{GR}.

Генератор GKLS [5] позволяет получать задачи заданной размерности и с заданным количеством экстремумов. Кроме того GKLS позволяет регулировать сложность задач, уменьшая или увеличивая размер области притяжения глобального минимума. В работе [25] указаны параметры генератора, позволяющие породить наборы по 100 задач двух уровней сложности (Simple и Hard) размерностей 2, 3, 4 и 5. Следуя примеру авторов генератора GKLS, будем использовать предложенные ими параметры и, таким образом, добавим в тестовый ещё 800 задач различной размерности и сложности.

Будем считать, что тестовая задача решена, если метод оптимизации провёл очередное испытание y^k в δ-окрестности глобального минимума y^*, т.е. $\left\|y^k - y^*\right\| \leqslant \delta = 0.01 \left\|b - a\right\|$, где a и b — левая и правая границы гиперкуба из (1). Если указанное соотношение не выполнено до истечения лимита на количество испытаний, то задача считается нерешённой. Максимальный лимит на количество испытаний установлен для каждого класса задач, в соответствии с размерностью и сложностью (см. таблицу 1).

Таблица 1. Trials limit for the each of test problems 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

В качестве характеристик метода оптимизации на каждом из классов будем рассматривать среднее число испытаний, затраченное для решения одной задачи, и количество решённых задач. Чем меньше число испытаний, тем быстрее метод сходится к решению, а значит и меньше обращается к потенциально трудоёмкой процедуре вычислений целевой функции. Количество решённых задач говорит о надёжности метода с заданными параметрами на решаемом классе тестовых задач. Чтобы сделать величины, характеризующие надёжность и скорость сходимости, независимыми при подсчёте среднего количества испытаний учитывались только решённые задачи.

6 Results of Numerical Experiments

6.1 Results of Sequential Algorithms

Результаты тех или иных алгоритмов на различных классах задач напрямую зависят от настроек алгоритмов. В большинстве случаев авторы программных реализаций ориентируются на задачи средней сложности и чтобы получить удовлетворительный результат при решении существенно многоэкстремальных задач, требуется корректировка некоторых параметров. При проведении сравнения были заданы следующие параметры для методов:

- в методе AGSl параметр чередования глобального и локального правил вычисления характеристик был задан равным 5:1;
- в методах DIRECT и DIRECTl параметр $\epsilon = 10^{-4}$;
- в методе SDA параметр $visit = 2.72$.

Остальные параметры варьировались в зависимости от класса задач (см. таблицу 2).

Таблица 2. Class-specific parameters of 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

В таблицах 3, 4 приведены результаты запуска методов оптимизации на рассматриваемых классах задач. Методы DIRECT и AGSl показали наилучшую скорость сходимости на всех классах, причём AGSl уступает DIRECT

на задачах из классов Simple и обходит его на задачах классов Hard. Как видно из таблицы 4, самыми надёжными методами являются детерминированные: AGS, AGSl, DIRECT и DIRECTl. Из стохастических методов самую высокую надёжность демонстрируют MLSL и SDA.

Таблица 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

Таблица 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 Results of Parallel Algorithms

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

Сравнение двух параллельных алгоритмов, реализованных в системах Globalizer и MIDACO было проведено на четырёхмерных задачах GKLS,

т.к. задачи большей размерности не поддерживаются бесплатной версией MIDACO. Как и в случае с последовательными алгоритмами, для кадого класса задач были изменены параметры методов. Для MIDACO параметр $focus = -1$. В случае с Globalizer параметр r был выбран следуя таблице 2. С удвоением количества задействованных развёрток этот параметр уменьшался на 0.3, поскольку увеличение количества развёрток увеличивает и надёжность метода из секции 4.2 (подробнее см. [27]).

Если считать, что затраты на параллелизм пренебрежимо малы по сравнению с затратами на вычисление целевых функций в задачах оптимизации, то ускорение по времени от использования параллельного метода будет равно ускорению по итерациям. Однако, в действительности это предположение справедливо, тлолько если вычисление значения целевой функции достаточно трудоёмко. Во всех численных экспериментах время вычисления целевой функции занимает примерно 2×10^{-3}с.

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. В скобках также указано количество решённых задач в каждом классе. Globalizer решает практически все задачи в обоих классах, в то время как MIDACO не решает многие задачи из класса GKLS 4d Hard. Стоит ещё отметить, что с ростом числа узлов надёжность MIDACO падает на обоих классах задач, в то время как надёжность Globalizer остаётся стабильной.

Таблица 5. Averaged numbers of iterations executed by the parallel algorithm 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)

Среднее ускорение по итерациям и среднее ускорение по времени (в скобках) приведены в таблице 6. В первой строке таблицы приведено среднее количество итераций и среднее время решения задачи в последовательном режиме (в скобках). MIDACO для при параллельных вычислениях использует

меньшее количество коммуникаций, поэтому показатели ускорения монотонно возрастают с ростом количества вычислительных узлов. В Globalizer каждый узел должен переслать результаты проведения итерации всем другим узлам и с увеличением количества узлов и количества рабочих потоков на каждом узле эффективность распараллеливания падает. Это можно видеть в таблице 6, когда при переходе от 4х узлов к 8 ускорение по времени при использовании 16 потоков упало. В случае одного потока на узел ускорение обоих методов оптимизации примерно одинаковое. Однако стоит заметить, что Globalizer выполняет меньше испытаний и при $p = 1$ тратит гораздо меньше времени на решение всех задач. С ростом количества узлов и потоков MIDACO догоняет Globalizer по среднему времени решения задачи, однако надёжность при этом падает.

Аномальное сверхлинейное ускорение по времени MIDACO на классе GKLS 2d Hard при использовании 8 и 12 узлов можно объяснить тем, что, согласно таблице 5, MIDACO решил больше задач в параллельном режиме, чем в последовательном, а значит, на меньшем количестве задач работал до исчерпания лимита испытаний.

Таблица 6. Speedup of parallel computations executed by the parallel algorithm

	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)

7 Conclusions

В статье были рассмотрены несколько последовательных и параллельных алгоритмов глобальной оптимизации. Было проведено сравнение эффективности их работы на множестве тестовых задач, по результатам которого можно сделать следующие выводы:

- методы AGSl, реализованный в системе Globalizer показал скорость сходимости и надёжность на уровне $DIRECT$ и превосходит многие другие алгоритмы, реализации которых есть в открытом доступе;
- стохастические методы оптимизации проигрывают детерминированным по скорости сходимости и надёжности. Особенно сильно это проявляется на более сложных многоэкстремальных задачах;
- параллельная версия системы Globalizer демонстрирует хорошие показатели ускорения по времени при работе на нескольких узлах, на каждом из которых вычисляется одно значение целевой за итерацию. При решении задач с быстро вычислимой целевой функцией и использовании многопоточности на узлах показатели ускорения системы Globalizer деградируют с увеличением количества узлов;
- система MIDACO больше всего подходит для несложных глобальных задач, целевые функции в которых могут быть быстро вычислимы. В таком случае MIDACO достаточно надёжна и обеспечивает линейное ускорение с ростом количества узлов и параллельно выполняемых потоков на них.

Acknowledgements

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

Список литературы

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. Journal of Global Optimization **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. Applied Mathematics and Computation **318**, 245 – 259 (2018)
16. Liberti, L., Kucherenko, S.: Comparison of deterministic and stochastic approaches to global optimization. International Transactions in Operational Research **12**, 263 – 285 (2005)
17. Madsen, K., Zertchaninov, S.: A new branch-and-bound method for global optimization (1998)
18. 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. Computational Mathematics and Mathematical Physics **27(1)**, 33–39 (1987)
19. Mullen, K.: Continuous global optimization in r. Journal of Statistical Software, Articles **60**(6), 1–45 (2014)
20. Paulavivcius, R., Zilinskas, J., Grothey, A.: Parallel branch and bound for global optimization with combination of lipschitz bounds. Optim. Methods Softw. **26**(3), 487–498 (1997)
21. Pošík, P., Huyer, W., Pál, L.: A comparison of global search algorithms for continuous black box optimization. Evolutionary Computation **20**(4), 509–541 (2012)
22. Price, W.L.: Global optimization by controlled random search. Journal of Optimization Theory and Applications **40**(3), 333–348 (1983)
23. Schlueter, Martin: Nonlinear mixed integer based optimization technique for space applications. Ph.D. thesis, The University of Birmingham (2012)
24. 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)
25. Sergeyev, Y., Kvasov, D.: Global search based on efficient diagonal partitions and a set of lipschitz constants. SIAM Journal on Optimization **16**(3), 910–937 (2006)
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)