УДК 519.853.4; 517.972.8
DOI: 10.25559/SITITO.15.201902.340-350

Параллельный метод липшицевой глобальной оптимизации

А. С. Горбунов[1*], **М. А. Посыпкин**[2]
[1] Московский государственный университет имени М.В. Ломоносова, г. Москва, Россия
119991, Россия, г. Москва, Ленинские горы, д. 1
[*] aleksey.gorbunov.97@gmail.com
[2] Федеральный исследовательский центр «Информатика и управление» Российской академии наук, г. Москва, Россия
119333, Россия, г. Москва, ул. Вавилова, д. 44-2

Аннотация

В статье представлен подход к решению задач липшицевой глобальной оптимизации, основной целью которого является ускорение нахождения глобального минимума липшицевой функции за счет использования нескольких параллельных потоков обработки. Алгоритм состоит из двух основных этапов: глобального поиска, во время которого определяются гиперинтервалы (многомерные параллелепипеды), на которых может содержаться глобальный минимум, и локального поиска, где происходит уточнение значения минимума для рассматриваемого гиперинтервала. На этапе глобального поиска используется метод ветвей и границ. Получение нижней оценки минимума функции достигается с помощью оценивания константы Липшица. В статье приведено подробное описание разработанного алгоритма, даны необходимые пояснения, показана корректность разработанного алгоритма. Метод может быть применен к многомерным многоэкстремальным функциям, в том числе, не имеющим аналитического представления, т.е. заданным в форме «черного ящика». Простота представленного алгоритма позволяет выполнять оценку константы Липшица параллельно, что, при использовании современных вычислительных систем с многоядерными процессорами может существенно уменьшить время работы программы. Достигаемое благодаря параллельному выполнению ускорение подтверждено серией экспериментов. Методика проведения экспериментов представлена в данной статье, так же, как и пояснения по выбору параметров работы алгоритма. Проведено сравнительное исследование скорости работы алгоритма в последовательном и параллельном режимах, проведен анализ полученных результатов и сделаны выводы об эффективности предложенного подхода.

Ключевые слова: глобальная оптимизация, константа Липшица, параллельное программирование.

Для цитирования: *Горбунов А. С., Посыпкин М. А.* Параллельный метод липшицевой глобальной оптимизации // Современные информационные технологии и ИТ-образование. 2019. Т. 15, № 2. С. 340-350. DOI: 10.25559/SITITO.15.201902.340-350

Parallel Lipschitz Global Optimization Method

A. S. Gorbunov[a*], **M. A. Posypkin**[b]
[a] Lomonosov Moscow State University, Moscow, Russia
1 Leninskie Gory, GSP-1, Moscow 119991, Russia
[*] aleksey.gorbunov.97@gmail.com
[b] Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia
44-2 Vavilov St., Moscow 119333, Russia

Abstract

The article is devoted to the description of the developed approach for solving Lipschitz global optimization problems. The main goal of the proposed approach is to accelerate the search for a global minimum of the Lipschitz function through the use of several parallel processing threads. The algorithm consists of two phases: the global search phase, which specifies the hyperintervals (multidimensional parallelepipeds), which may contain a global minimum, and the local search phase, where the minimum value for the hyperinterval is refined. The global search phase uses the branch and bound method. The calculation of the lower estimate of the minimum of a function is performed by estimating the Lipschitz constant. The article describes the developed algorithm, gives the necessary explanations, and shows the correctness of the developed algorithm. The method can be applied to multidimensional multiextremal functions, including those that do not have an analytical representation, i.e. specified in the form of a "black box". The simplicity of the developed algorithm makes it possible to estimate the Lipschitz constant in parallel, which, in case of using modern computing systems with multi-core processors, can significantly speed up the execution of the program. The acceleration achieved due to parallel execution is confirmed by a series of experiments. The methodology of the experiments provided in this article, as well as explanations for the choice of the parameters of the algorithm. A comparative study of the speed of the algorithm in sequential and parallel modes was carried out, obtained results were analyzed and conclusions were made about the effectiveness of the proposed approach.

Keywords: global optimization, Lipschitz constant, parallel programming.

For citation: Gorbunov A.S., Posypkin M.A. Parallel Lipschitz Global Optimization Method. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2019; 15(2):340-350. DOI: 10.25559/SITITO.15.201902.340-350

Введение

Задачи глобальной оптимизации возникают во многих сферах человеческой деятельности, и зачастую представляются в форме многомерных многоэкстремальных функций, нахождение глобального минимума аналитически для которых либо не представляется возможным, либо слишком сложно. Зачастую удается решить задачу глобальной оптимизации, допуская условие Липшицевости целевой функции:

$$\left|f(x)-f(y)\right| \le L\left\|x-y\right\|, x, y \in R^n, f: R^n \to R. \quad (1)$$

Если выполнено условие (1), то следующая функция будет нижней оценкой для $f(x)$ $\phi(x)=\phi(c)-L\left\|x-c\right\|, \quad (2)$

где c некоторая точка из R^n При этом, могут рассматриваться различные варианты нормы. Существует много методов Липшицевой глобальной оптимизации, использующих это свойство [1-7]. Однако, ввиду сложности этих методов, в последовательном варианте они работают неприемлемо долго. В то же время, современная высокопроизводительная техника располагает значительными вычислительными мощностями, предоставляя широкие возможности для эффективного распараллеливания и ускорения алгоритмов. Поэтому, разработка алгоритма глобальной оптимизации, использующего возможности современных многоядерных вычислительных систем, является важной задачей. Разработке таких подходов посвящено большое число статей [15-23].

В данной работе предложен алгоритм, использующий поиск на равномерной сетке с локальной оценкой константы Липшица. Полученная оценка константы Липшица, в свою очередь, применяется для оценки нижней границы целевой функции. Предложенный метод работает в соответствии со стандартной схемой «ветвей и границ»: допустимое множество поиска разбивается на два и поиск повторяется, соответственно, с меньшим шагом сетки до достижения условия окончания итераций. Допустимые множества в подзадачах, получаемых в процессе работы метода ветвей и границ, являются n-мерными параллелепипедами (гиперинтервалами).

Оценка константы Липшица выполняется на равномерной сетке на каждом гиперинтервале. Для ускорения вычисления оценки нижней границы для гиперинтервала применяется параллельное программирование с выполнением несколькими потоками на CPU, при помощи библиотеки OpenMP. Структура предложенного алгоритма позволяет применить возможности параллельного программирования и добиться значительного сокращения времени выполнения в случае запуска на устройствах с несколькими процессорными ядрами. Далее в разделе 1 приводится постановка задачи, в разделе 2 дано теоретическое описание алгоритма. Результаты вычислительного эксперимента на наборе из нескольких известных тестовых многомерных функций приведены в разделе 3.

1. Постановка задачи

Многоядерные системы с общей памятью являются наиболее распространенной платформой для высокопроизводительных расчетов. Приложение, ориентированное на такие системы, состоит из параллельно выполняемых потоков [8-10]. Цель данной работы – разработка алгоритма глобальной оптимизации, который поддерживал бы многопоточное выполнение, тем самым обеспечивая ускорение вычислений.

Рассматривается задача глобальной оптимизации с интервальными ограничениями:

$$f(x) \to min, x_i \in [a_i, b_i], i = 1, \ldots, n. \quad (3)$$

Ограничения задачи (1) задают гиперинтервал n (-рный параллелепипед) $B=[a_1,b_1]\times\ldots\times[a_n,b_n]$Оптимальным решением задачи (1) называется точка $x_* \in B$ ля которой выполнено соотношение

$$f(x_*) \le f(x) для\ всех\ x \in B$$

За исключением совсем простых случаев, найти оптимальное решение задачи (3) не представляется возможным. Поэтому обычно требуется найти -оптимальное решение данной задачи, т.е. такую точку $x_\varepsilon \in B$ что выполнено неравенство $f(x_\varepsilon) \le f(x_*)+\varepsilon$ Для решения данной задачи применяется метод неравномерных покрытий [7, 11-14]. Предполагается, что функция удовлетворяет условию Липшица (1) на гиперинтервале $B=[a_1,b_1]\times\ldots\times[a_n,b_n]$Базовая схема метода неравномерных покрытий состоит в том, что исходный параллелепипед делится на параллелепипеды меньшего размера. Параллелепипеды, для которых выполнено следующее условие, называемое *условием отсева*, исключаются из дальнейшего рассмотрения и добавляются к покрытию:

$$f(c)-L\frac{d}{2}+\varepsilon \ge f_r, \quad (4)$$

где A центр рассматриваемого параллелепипеда, d его диаметр (расстояние между двумя самыми удаленными вершинами), а f_r рекордное значение, т.е. наилучшее найденное на данном шаге значение целевой функции в допустимой точке x_r сходного множества B называемой *рекордным решением*. Действительно, в силу (2) из неравенства (4) следует, что найденный рекорд является -решением для задачи (3) на рассматриваемом гиперинтервале. Рекордное значение ищется как минимальное значение функции в центрах рассматриваемых параллелепипедов.

Существует два основных подхода к нахождению константы Липшица. Первый [11-14] состоит в применении методов интервального анализа для получения верхней оценки на норму градиента функции. Такая оценка, очевидно, будет константой Липшица. Достоинством такого подхода является гарантия ε – тимальности найденного решения. Для его применения необходимо аналитическое представление функции, что не позволяет применять его к задачам оптимизации «черного ящика», в которых такое представление недоступно. Второй подход основан на приближенных оценках константы Липшица, получаемых в процессе расчета. В настоящее время разработано много алгоритмов для получения таких оценок [1-6]. В работе мы также следуем данному подходу – константа Липшица оценивается на равномерной сетке в процессе расчетов. Предложенный метод не оптимален с точки зрения количества вычислений функции, в сравнении с другими известными методами, которые минимизируют это количество. Но использование равномерной сетки позволяет эффективно применить методы параллельных вычислений и, тем самым, ускорить работу программы в несколько раз. Несмотря на большой объем вычислений применение данного метода оправданно, поскольку ускорение, полученное применением параллельных вычислений может быть весьма значительным.

2. Теоретическое описание алгоритма

2.1 Описание последовательного алгоритма

В основе предложенного подхода лежит метод неравномерных покрытий, работающий по схеме «ветвей и границ»:

Псевдокод

```
Функция search(dim, a[], b[], x[], func)
1. Hyperintervals H, H1;
2. UpperBound = max.value;
3. ДОБАВИТЬ исходный h(a,b) гиперинтверал в H
4. ПОКА H не пуст
5.          ДЛЯ всех гиперинтервалов h из H
6.                  Gridsearch(h.a, h.b, func, h.x, &h.UpperBound,
&h.LowerBound);
7.                  Update(UpperBound, x[]);
8.          ДЛЯ всех гиперинтервалов h из H
9.
```

ЕСЛИ

$$(h.LowerBound < UpperBound - \varepsilon)$$

```
ТО
10.                 Divide(h.a, h.b,&a1, &b1,&a2,&b2);
11.                 ДОБАВИТЬ h1(a1,b1),h2(a2,b2) в H1
12.         H.clear()
13.         H = H1;
14.         H1.clear();
```

Пояснение

Функция принимает на вход размерность решаемой задачи dim, два массива a[] и b[], содержащие соответственно левую и правую границы гиперинтервала, на котором требуется осуществить поиск, массив x[], в который будут записаны координаты найденного минимума, а также указатель на функцию, возвращающую значение исследуемой функции в заданной точке.

Алгоритм начинает работу с объявления двух массивов гиперинтервалов H и H1. Для каждого гиперинтервала h в массиве возможно хранить его левую и правую границы h.a и h.b соответственно, верхнюю h.UpperBound и нижнюю h.LowerBound оценки минимума, а также координаты h.x, в которых была найдена верхняя оценка минимума.

В переменной UpperBound будем хранить текущий «рекорд» - наименьшее значение верхней оценки минимума. Изначально инициализируем ее максимально возможной величиной для рассматриваемого типа значений. Вначале добавим в список гиперинтервалов H исходный гиперинтервал с границами a и b, переданными в качестве аргументов функции поиска.

В цикле, пока список гиперинтервалов не пуст, к каждому из гиперинтервалов h из списка H применим функцию поиска на сетке Gridsearch. Результатом ее работы будут являться значения верхней и нижней оценок минимума для данного гиперинтервала, а также координаты, в которых было получено минимальное значение. Обновим, если необходимо, глобальный рекорд, сравнив его текущее значение и полученное на предыдущем шаге значение локального минимума, и соответствующие ему координаты Update(UpperBound,x[]). После того, как были получены локальные значения оценок для всех гиперинтервалов из списка, проверим каждый гиперинтервал на соответствие условию отсева. Если условие отсева не выполнено, то есть, если найденная локальная нижняя оценка минимума меньше, чем текущий рекорд, продолжаем уточнение значения минимума на нем. Для этого разобьем текущий гиперинтервал на два вдоль большей стороны. Добавим два новых получившихся гиперинтервала в список H1. Если же условие отсева выполнено, никаких новых гиперинтервалов не создается, гиперинтервал будет исключен из рассмотрения.

Очистим список рассматриваемых гиперинтервалов и присвоим ему указатель на новый список, состоящий из более мелких, которые потенциально могут содержать минимум. На следующей итерации цикла будут рассматриваться эти гиперинтервалы. Когда для всех гиперинтервалов выполнено условие отсева, никаких новых гиперинтервалов для рассмотрения сгенерировано не будет, цикл завершится. В результате получим наименьшее значение функции в заданном гиперинтервале с заданной точностью.

2.2 Оценка константы Липшица

Зная константу Липшица для рассматриваемой функции, мы можем оценить нижнюю границу для функции на рассматриваемом гиперинтервале и применить формулу (4) для принятия решения об отсеве параллелепипеда. Проблема заключается в том, что точное значение константы Липшица, как правило, неизвестно. Точность оценки константы Липшица играет ключевую роль в успешности нахождения глобального минимума. В случае, если константа Липшица будет недооценена, мы получим завышенное значение нижней границы функции, а значит можем не получить -оптимального решения. Напротив, слишком большая величина константны Липшица приведет к занижению оценки нижней границы функции, что ведет к значительному увеличению времени поиска глобального минимума, и, как следствие, к росту объема производимых алгоритмом вычислений.

Обозначим через L_* минимальную» константу Липшица на гиперинтервале P определяемую формулой:

$$L_*(P) = \max_{x',x'' \in P} \frac{|f(x') - f(x'')|}{\|x' - x''\|} \quad (5)$$

Непосредственное применение данной формулы представляется затруднительным, т.к. решение задачи (5) не менее сложно, чем исходной задачи. В данной работе предлагается оценивать константу Липшица на равномерной сетке. Пусть, N множество узлов прямоугольной равномерной сетки, построенной на рассматриваемом гиперинтервале. Гранулярность разбиения задается числом узлов вдоль одного измерения. Положим

$$\delta_i = \frac{b_i - a_i}{N-1}, \delta = \sum_{i=1}^{\dim} \delta_i \quad (6)$$

Для оценки константы Липшица используется следующее соотношение:

$$L(P) = \kappa(\delta) \max_{x',x'' \in N} \frac{|f(x') - f(x'')|}{\|x' - x''\|}, \quad (7)$$

где δ диаметр ячейки сетки, определяемый (6), а $\kappa(\delta)$ коэффициент надежности вычисленной оценки. Очевидно, что чем больше элементов в сетке, тем точнее оценка константы Липшица. Поэтому, коэффициент надежности является функцией от диаметра ячейки сетки и обладает следующими свойствами:

$1)\kappa(\delta)\to 1 \ при \delta\to 0,$
$2)\kappa(\delta')\geq\kappa(\delta'') при \delta'\geq\delta''.$

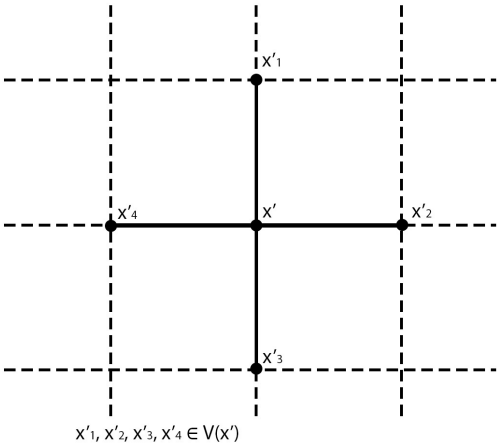
рвое свойство выражает тот факт, что при измельчении сетки оценка стремится к точному значению константы Липшица. Второе свойство означает монотонное убывание введенного коэффициента с уменьшением гранулярности сетки. Этим свойствам удовлетворяет функция $\kappa(\delta)=e^{\delta}$ Результаты экспериментов подтвердили эффективность применения этой функции.

Вычисление оценки $L(P)$ о формуле (7) является, вообще говоря, весьма трудоемким, т.к. требует в общем случае $O(|N|^2)$ пераций. Сложность может быть уменьшена до $O(|N|)$ ля «манхеттенской» нормы $\|x\|_1=\sum_{i=1}^{n}|x_i|$

Утверждение 1. Пусть задана равномерная сетка N а некотором гиперинтервале. Тогда

$$\max_{x',x''\in N}\frac{|f(x')-f(x'')|}{\|x'-x''\|_1}=\max_{x'\in N}\max_{x''\in V(x')}\frac{|f(x')-f(x'')|}{\|x'-x''\|_1}, \quad (8)$$

где $V(x')$ ножество соседних с x'очек сетки: $V(x)=\{y\in N:|x_i-y_i|=\delta_i \ для \ некоторого \ i, \ x_j=y_j \ при \ i\neq j\}$ Рис. 1).



Р и с. 1.
F i g. 1.

Доказательство.
Возьмём произвольные, не соседние точки, путь между которыми состоит из $k+1$ебер сетки, и обозначим их соответственно x_1 x_k Рис. 2). Можем записать:

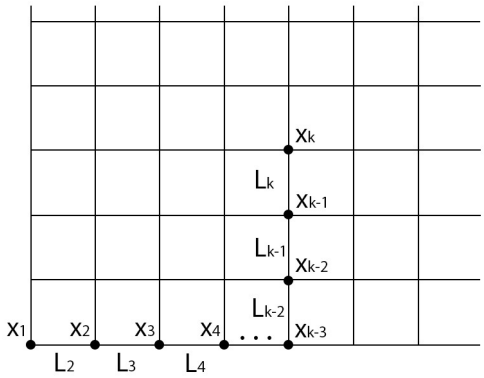
$$f(x_k)-f(x_1)=f(x_k)-f(x_{k-1})+f(x_{k-1})-f(x_{k-2})+\cdot$$
$$=f(x_{k-2})-\cdots-f(x_2)+f(x_2)-f(x_1)$$

итывая $|\sum x_i|\leq\sum|x_i|$ свойство липшицевости функции, получим:

$$|f(x_k)-f(x_1)|\leq|f(x_k)-f(x_{k-1})|+|f(x_{k-1})-f(x_{k-2})|+$$
$$+\ldots+|f(x_2)-f(x_1)|\leq L_k x_k-x_{k-1}+L_{k-1}x_{k-1}-x_{k-2}+$$
$$+\ldots+L_2x_2-x_1\leq L'\sum_{i=2}^{k}x_i-x_{i-1}=L'x_k-x_1$$

лучаем: $|f(x_k)-f(x_1)|\leq L'x_k-x_1$ где $L'=\max_{i=2\ldots k}L_i$

При обработке очередного гиперинтервала P троится сетка N с помощью которой вычисляются верхняя $f_u(P)$ нижняя $f_l(P)$ енки минимума $f_*(P)$ ункции $f(x)$ Следующее утверждение обосновывает формулы вычисления этих оценок.



Р и с. 2.
F i g. 2.

Утверждение 2. Если функция $f(x)$ довлетворяет условию Липшица с константой $L(P)$ то для величин $f_u(P)$ $f_l(P)$ вычисленных по формулам:

$$f_u(P)=\min_{x\in N}f(x), f_l(P)=f_u(P)-\frac{\delta}{2}L(P) \quad (9)$$

справедливо неравенство

$$f_u(P)\geq f_*(P)\geq f_l(P) \quad (10)$$

Поскольку на практике $L(P)$ ценивается приближенно, то второе неравенство в (9) выполнено приближенно с точностью до корректности сделанных предположений.

2.3 Нахождение оценок целевой функции на гиперинтервале

Последовательный алгоритм
Рассмотрим алгоритм функции, находящей верхнюю и нижнюю оценки минимума целевой функции на гиперинтервале по формулам (9).
Псевдокод
Для работы функции предварительно созданы вспомогательные массивы:
$F_{nodes}[gridnodes], x[dim]$, где $gridnodes=nodes^{dim}$
Функция $GridEvaluator(dim, a[], b[], xmin[], f_u, f_l, func, nodes)$.
1. $L=min.value; \delta=min.value$
2. $f_u=max.value; L=min.value$
3. ДЛЯ ВСЕХ $i=1\ldots dim$
4. $Gridstep[i]=\frac{|b[i]-a[i]|}{nodes}$
5. ЕСЛИ $(\delta<Gridstep[i]), 0 \ \delta=Gridstep[i]$
6. $\kappa=e^{0.5*dim*\delta}$
7. ДЛЯ ВСЕХ $i=1\ldots gridnodes$
8. $x=indtocoord(i)$
9. $F_{nodes}[i]=func(x)0$
10. ЕСЛИ $(f_u>F_{nodes}[i]), 0$
11. $f_u=F_{nodes}[i]$

12. $node = i$
13. ДЛЯ ВСЕХ $i = 1 \ldots gridnodes$
14. ДЛЯ ВСЕХ $k = 1 \ldots dim$
15. $x_{neighbour} = getNeighbour(i,k)$
16. $L_{loc} = \dfrac{|F_{nodes}[i] - F_{nodes}[x_{neighbour}]|}{Gridstep[dim-1-k]}$
17. ЕСЛИ $(L_{loc} > L)$, О
18. $L = L_{loc}$
19. $xmin = indtocoord(node)$
20. $f_l = f_u - L * \kappa * \delta$

Пояснение

Для работы функции необходимо несколько вспомогательных массивов. Массив $Gridstep$ предназначен для хранения длины шагов сетки вдоль каждого из измерений. Всего в сетке будет содержаться $gridnodes = nodes^{dim}$ узлов. Массив F_{nodes} служит для хранения значений функции во всех узлах сетки. Массив x служит для хранения координат узла, в котором вычисляется значение целевой функции.

Функция принимает на вход размерность решаемой задачи dim, границы рассматриваемого гиперинтервала $a[]$ $b[]$, указатель на функцию, для которой выполняется поиск минимума $func$, а также, количество узлов в сетке вдоль одного измерения. Функция заполняет массив $xmin[]$ кординатами наименьшего найденного значения, а также вычисляет верхнюю f_u и нижнюю f_l оценки минимума.

В начале работы функции инициализируются переменные, которые будут содержать оценку константы Липшица L и наибольшую сторону ячейки сетки. Так как необходимо отыскать наибольшее из значений, целесообразно инициализировать эти переменные минимально возможной величиной. Поскольку необходимо найти минимальное значение функции, инициализируем f_u максимально возможной величиной, а оценка константы Липшица, наоборот, должна быть максимальной из всех вычисленных, соответственно, инициализируем L минимально возможным значением. Вычислим шаг сетки вдоль всех измерений, и найдем среди них наибольшее значение δ. Это значение используем, чтобы вычислить коэффициент надежности для данной сетки κ.

В цикле по всем узлам сетки производится расчет значений функции в узлах сетки. При этом изначально имеется лишь индекс узла в сетке, но, зная шаг сетки вдоль каждого из измерений и границы рассматриваемого гиперинтервала, можно от индекса перейти к координатам узла. Для этого необходимо поделить индекс на число узлов в сетке вдоль одного измерения, взять остаток от деления и, умножив его на соответствующий шаг сетки, прибавить к левой границе гиперинтервала. Повторив процедуру деления и получения координаты для соответствующего измерения dim раз, получим все координаты рассматриваемого узла. Полученное значение функции сравнивается с текущим рекордом, если оно меньше, рекорд и соответствующий ему индекс узла обновляются.

Когда значения функции во всех узлах сетки получены, начинается оценивание константы Липшица. Для всех узлов сетки в цикле находятся dim соседних узлов, которые отличаются от текущего узла на +1 шаг сетки по одной из координат. Соседи, отличающиеся на -1 шаг сетки не рассматриваются, чтобы избежать повторных вычислений для одних и тех же пар узлов, а также исключаются из рассмотрения соседние узлы, которые оказываются за границами рассматриваемого гиперинтервала. Вычисленное для каждой пары соседних узлов, принадлежащих рассматриваемому гиперинтервалу, значение оценки константы Липшица сравнивается с текущим рекордом, и, если оно больше, рекорд обновляется. Во избежание зависимостей по данным здесь применяется тот же подход, что и при вычислении значений функции во всех узлах сетки.

Зная индекс узла с минимальным значением функции в сетке, найдем координаты этого узла. Затем вычислим по формуле оценку константы Липшица для рассматриваемого гиперинтервала (строка 20). Работа функции завершена.

Параллельный алгоритм

Функция, находящая верхнюю и нижнюю оценки минимума для каждого гиперинтервала была распараллелена, поскольку именно она обладает наибольшей вычислительной сложностью и её выполнение составляет наибольшую часть от времени работы программы.

Псевдокод

Для работы функции предварительно созданы вспомогательные массивы:

$Gridstep[dim], f_u ARRAY[numprocs],$

$L_{values} ARRAY[numprocs], X_{min} ARRAY[numprocs],$

где $gridnodes = nodes^{dim}$

Функция $GridEvaluator(dim, a[], b[], xmin[], f_u, f_l, func, nodes)$

1. $L = min.value; \delta = min.value$
2. ДЛЯ ВСЕХ $i = 1 \ldots numthreads$
3. $f_u ARRAY[i] = max.value; L_{values} ARRAY[i] = min.value$
4. ДЛЯ ВСЕХ $i = 1 \ldots dim$
5. $Gridstep[i] = \dfrac{|b[i] - a[i]|}{nodes}$
6. ЕСЛИ $(\delta < Gridstep[i])$, ТО $\delta = Gridstep[i]$
7. $\kappa = e^{0.5 * dim * \delta}$

#ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ $numthreads$ (потоков)

8. ДЛЯ ВСЕХ $i = 1 \ldots gridnodes$ (итерации распределены по потокам)
9. $nt = get\_thread\_num()$
10. $x + nt * dim = indtocoord(i)$
11. $F_{nodes}[i] = func(x + nt * dim)$
12. ЕСЛИ $(f_u ARRAY[nt] > F_{nodes}[i])$, ТО
13. $f_u ARRAY[nt] = F_{nodes}[i]$
14. $X_{min} ARRAY[nt] = i$

#КОНЕЦ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ

#ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ $numthreads$ (потоков)

15. ДЛЯ ВСЕХ $i = 1 \ldots gridnodes$ итерации распределены по потокам)
16. ДЛЯ ВСЕХ $k = 1 \ldots dim$
17. $x_{neighbour} = getNeighbour(i,k)$
18. $L_{loc} = \dfrac{|F_{nodes}[i] - F_{nodes}[x_{neighbour}]|}{Gridstep[dim-1-k]}$
19. $nt = get\_thread\_num()$
20. ЕСЛИ $(L_{loc} > L_{values} ARRAY[nt])$, ТО
21. $L_{values} ARRAY[nt] = L_{loc}$

#КОНЕЦ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ

22. ДЛЯ ВСЕХ $i = 1 \ldots numthreads$
23. ЕСЛИ $(f_u ARRAY[i] < f_u)$, О
24. $f_u = f_u ARRAY[i]$

25. $node = X_{min} ARRAY[i]$
26. ЕСЛИ $\left(L_{values}[i] > L\right)$,
27. $L = L_{values}[i]$
28. $xmin = indtocoord(node)$
29. $f_l = f_u - L * \kappa * \delta$

Пояснение

Для работы функции, кроме массивов, использующихся в последовательном алгоритме, необходимо ещё несколько вспомогательных массивов. $f_u ARRAY, L_{values} ARRAY$ $X_{min} ARRAY$ сужат для хранения вычисленных верхней оценки минимума, оценки константы Липшица и индекса точки, в которой был достигнут минимум для каждого потока. Массив x служит для хранения координат узлов для каждого из потоков, для вычисления в них значений исследуемой функции.

Чтобы использовать возможности вычислительной системы максимально, алгоритм на параллельном этапе работы использует столько потоков, сколько максимально может обрабатывать процессор $numthreads$. Использование такого числа потоков позволяет наиболее эффективно использовать вычислительные ресурсы.

Параметры функции совпадают с таковыми у функции из описания последовательного алгоритма. Инициализация переменных L, δ аналогично последовательному алгоритму, так же как и расчет коэффициента надежности κ.

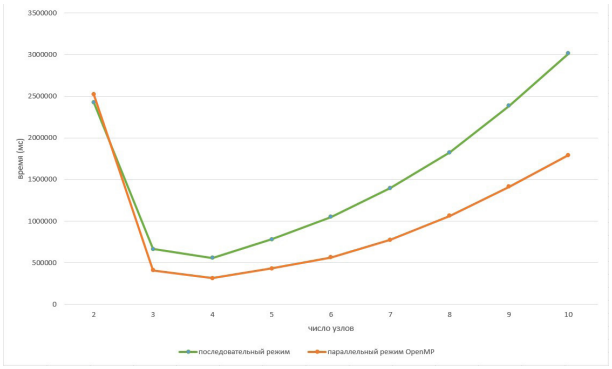
В цикле, итерации которого выполняются параллельно на $numthreads$ потоках, производится расчет значений функции в узлах сетки. Получаемые значения функции сравниваются с текущим рекордом на данном потоке, и, если текущее значение меньше, рекорд и соответствующий ему индекс узла обновляются. Каждый поток использует для хранения рекорда свою ячейку в массиве, благодаря чему удается избежать зависимости по данным.

Когда значения функции во всех узлах сетки получены, начинается оценивание константы Липшица. Для всех узлов сетки в цикле, итерации которого выполняются параллельно, находятся dim соседних узлов, которые отличаются от текущего узла на +1 шаг сетки по одной из координат. Логика вычисления оценки константы Липшица в данном случае совпадает с таковой в последовательном алгоритме. Вычисленное для каждой пары соседних узлов, принадлежащих рассматриваемому гиперинтервалу, значение оценки константы Липшица сравнивается с текущим рекордом для данного потока, и, если оно больше, рекорд обновляется. Во избежание зависимостей по данным здесь применяется тот же подход, что и при вычислении значений функции во всех узлах сетки.

3. Вычислительный эксперимент

3.1 Методика эксперимента

Целью эксперимента является сравнительное исследование предложенного алгоритма в последовательном и параллельном режимах. Основными параметрами, которые влияют на работу алгоритма, являются число узлов вдоль одного измерения, задающее общее число узлов в сетке, и точность, с которой необходимо найти глобальный минимум, а также, для параллельного режима работы алгоритма, число используемых вычислительных потоков. При проведении эксперимента было использовано значение числа узлов вдоль одного измерения равное 4.



Р и с. 3.
F i g. 3.

Очевидно, что чем больше узлов в сети, тем меньше шаг сетки, а также быстрее достигается необходимая точность вычисления. Соответственно методу ветвей и границ необходимо сделать меньше итераций. Увеличение числа узлов вдоль одного измерения с одной стороны уменьшает число итераций метода ветвей и границ, а с другой – увеличивает число вычислений функции, притом настолько, что рост не компенсируются за счет уменьшения итераций метода ветвей и границ, что, очевидно, ведет к росту времени работы алгоритма. Другим негативным фактором, сопровождающим увеличение числа узлов сетки, является рост объема используемой памяти. Так как алгоритм хранит все вычисленные значения для одного гиперинтервала, требуется много памяти для хранения всех вычисленных значений. Например, в случае задачи размерностью 6 и числе узлов вдоль одного измерения 20, потребуется хранить в памяти 20^6 узлов сети, что для данных двойной точности (тип double языка C ++) составляет 2,71 Гб. Следовательно, меньшее число узлов в сетке не только обеспечивает лучшую производительность, но и более эффективно расходует память.

Для определения оптимального значения числа узлов было проведено исследование, в котором проводилось вычисление одного и того же набора тестовых задач из библиотеки GKLS [24] с фиксированными параметрами работы алгоритма, но с различным числом узлов в сети. По полученным результатам был построен график зависимости времени работы от числа узлов. (Рис. 3)

3.2 Результаты эксперимента

Эксперименты были проведены на персональном компьютере с процессором Intel Core i5 3230M и 6Гб оперативной памяти. Указанный процессор обладает двумя вычислительными ядрами с частотой 2.5 ГГц, а также поддерживает технологию HyperThreading, позволяющий осуществлять одновременную обработку сразу 4 потоков. При проведении экспериментов использовалась значение требуемой точности, равное 0.01. По умолчанию, алгоритм, работающий в параллельном режиме использует число потоков, равное числу логических процессоров в системе. В таком случае вычислительные ресурсы процессора используются максимально эффективно.

Для оценки производительности алгоритма было использовано два набора тестовых функций. В таблице 1 представлены результаты работы алгоритма с набором тестовых функций

GKLS [24]. Данный программный комплекс позволяет генерировать тестовые функции с заданными параметрами, в том числе с различной размерностью. Для проведения данного эксперимента для каждой из размерностей от 2 до 5 было сгенерировано 100 тестовых функций со следующими характеристиками: количество локальных минимумов: 10; значение глобального минимума: –1; радиус области притяжения глобального минимума: 1/3; расстояние от глобального минимума до вершины квадратичной функции: 2/3. При этом замерялось время, за которое алгоритм найдет оценку минимума для всех 100 тестовых функций. Во всех случаях найденное приближенное значение глобального минимума оказалось отличающимся от значения реального минимума не более чем на заданную величину точности вычисления ε Для данных функций осуществлялся поиск минимального значения на отрезке [-3; 3] по каждому из измерений. Тестирование проводилось для последовательного и параллельного вариантов работы алгоритма. Как видно из результатов, представленных в Таблице 1, для функций размерности 2 применение параллельного метода решения оказывается нецелесообразным, поскольку в этом случае число узлов в сетке достаточно мало, и использование нескольких потоков для расчета в силу накладных расходов на поддержание многопоточного режима оказывается неэффективным, обычный последовательный метод расчета решает задачу быстрее. Так как при большем числе измерений число узлов в сетке достаточно велико, то использование многопоточной модели оправдывает себя и, как видно из таблицы, позволяет достичь ускорения.

Т а б л и ц а 1. **Время работы последовательного и параллельного алгоритмов на наборе GKLS**

T a b l e 1. **Serial and parallel algorithms running time on the GKLS set**

Размерность / Режим работы	2	3	4	5
Последовательный, мс	50	2918	105886	3725719
Параллельный, мс	196	2638	68318	2078421
Ускорение, раз	0.255	1.106	1.550	1.793

Таблица 1

В таблице 2 представлены результаты вычислительных экспериментов, проведенных для функций из тестового набора [25]. Были выбраны 10 тестовых функций, для которых произведен поиска глобального минимума в последовательном и параллельном режиме работы алгоритма. Задано значение точности $\varepsilon = 0.01$ Для всех функций была найдена оценка глобального минимума, отличающаяся от значения реального глобального минимума не более чем на заданное значение точности вычисления. Как видно из результатов, представленных в таблице, параллельный режим позволяет ускорить процесс нахождения приближенного решения для всех исследованных функций.

Т а б л и ц а 2. **Время работы последовательного и параллельного алгоритмов на стандартных тестах**

T a b l e 2. **Serial and parallel algorithms running time on standard tests**

Тестовая функция (размерность)	Последовательный режим, мс	Параллельный режим, мс	Ускорение, раз
Ackley 3 (2)	1656	1220	1.357
Rosenbrock (3)	308171	176579	1.745
Beale (2)	1097	660	1.662
Goldstein Price (2)	1074	697	1.541
Booth (2)	99	63	1.571
Matyas (2)	85	53	1.604
Himmelblau (2)	173	111	1.559
Sphere (3)	167	88	1.898
Egg Holder (2)	12586	8616	1.461
Styblinski-Tang (2)	64	42	1.524

Заключение

В работе предложен алгоритм решения задач липшицевой глобальной оптимизации, основанный на принципе поиска на равномерной сетке с получением локальной оценки константы Липшица для рассматриваемой функции. Полученная оценка используется для расчета оценки нижней границы целевой функции на рассматриваемом гиперинтервале. Создание гиперинтервалов для обработки производится с использованием метода ветвей и границ, в результате применения которого исходный гиперинтервал разбивается на более мелкие гиперинтервалы с меньшим шагом сетки, до тех пор, пока не будет достигнута необходимая точность вычисления. В работе приведены основные формулы и утверждения, использованные при построении алгоритма, представлен псевдокод как для последовательной, так и для параллельной модификаций алгоритма.

Примененный подход к решению задачи позволяет использовать возможности параллельного программирования для вычисления значений функции в узлах сетки, а также для вычисления локальных оценок константы Липшица. Возможность эффективного параллельного выполнения на данный момент особенно актуальна, поскольку позволяет решать задачи значительно быстрее. Вместе с тем, многоядерные процессоры, необходимые для эффективной работы параллельных алгоритмов, получают широкое распространение. Алгоритм разработан в двух модификациях: последовательной и параллельной, что позволяет экспериментально оценить эффективность использования параллельной обработки.

При проведении вычислительных экспериментов были использованы два набора тестовых задач глобальной оптимизации. Проведенные вычислительные эксперименты показали корректность работы предложенного алгоритма, поскольку во всех случаях найденные приближенные значения глобального минимума отличаются от значения реального глобального минимума не более чем на заданную величину точности вычислений ε Сравнение результатов, полученных при использовании последовательной и параллельной модификаций алгоритма, показало эффективность применения параллельного алгоритма при решении задач размерности 3 и выше. Это обусловлено меняющимся при изменении размерности числом узлов в накладываемый сетке: при достаточно большом числе узлов распараллеливание процесса вычисления оправданно и позволяет значительно ускорить нахождение решения.

Полученные в работе результаты могут быть применены при решении широкого круга прикладных задач, формализация которых основана на оптимизационных моделях [26-28].

Список использованных источников

[1] *Pintér J. D.* Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications // Nonconvex Optimization and Its Applications. Vol. 6. Springer, Boston, MA, 2013. DOI: 10.1007/978-1-4757-2502-5

[2] *Strongin R. G., Sergeyev Y. D.* Global optimization with non-convex constraints: Sequential and parallel algorithms // Nonconvex Optimization and Its Applications. Vol. 45. Springer, Boston, MA, 2013. DOI: 10.1007/978-1-4615-4677-1

[3] *Sergeyev Y. D., Kvasov D. E.* Global search based on efficient diagonal partitions and a set of Lipschitz constants // SIAM Journal on Optimization. 2006. Vol. 16, Issue 3. Pp. 910-937. DOI: 0.1137/040621132

[4] *Sergeyev Y. D., Strongin R. G., Lera D.* Introduction to Global Optimization Exploiting Space-Filling Curves // SpringerBriefs in Optimization. Springer, New York, NY, 2013. DOI: 10.1007/978-1-4614-8042-6

[5] Hansen P., Jaumard B. Lipschitz optimization //Handbook of global optimization. Springer, Boston, MA, 1995. Pp. 407-493. DOI: 10.1007/978-1-4615-2025-2

[6] *Paulavičius R., Žilinskas J.* Simplicial Lipschitz optimization without Lipschitz constant // Simplicial Global Optimization / R. Paulavičius, J. Žilinskas (eds). Springer, New York, NY, 2014. Pp. 61-86. DOI: 10.1007/978-1-4614-9093-7

[7] *Евтушенко Ю. Г.* Численный метод поиска глобального экстремума функций (перебор на неравномерной сетке) // Журнал вычислительной математики и математической физики. 1971. Т. 11, № 6, С. 1390-1403. URL: http://www.mathnet.ru/links/d88634ac10c83c27d8703f0bbc54941c/zvmmf6781.pdf (дата обращения: 21.04.2019).

[8] *Chandra R.* et al. Parallel programming in OpenMP. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[9] *Sato M.* OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors // 15th International Symposium on System Synthesis, 2002. Kyoto, Japan, 2002. Pp. 109-111. DOI: 10.1145/581199.581224

[10] *Williams A.* C++ concurrency in action. Manning Publications Co., 2019.

[11] *Evtushenko Y., Posypkin M.* A deterministic approach to global box-constrained optimization // Optimization Letters. 2013. Vol. 7, Issue 4. Pp. 819-829. DOI: 10.1007/s11590-012-0452-1

[12] *Evtushenko Y. G., Posypkin M. A.* A deterministic algorithm for global multi-objective optimization // Optimization Methods and Software. 2014. Vol. 29, Issue 5. Pp. 1005-1019. DOI: 10.1080/10556788.2013.854357

[13] *Evtushenko Y. G., Posypkin M. A.* An application of the nonuniform covering method to global optimization of mixed integer nonlinear problems // Computational Mathematics and Mathematical Physics. 2011. Vol. 51, Issue 8. Pp. 1286. DOI: 10.1134/S0965542511080082

[14] *Evtushenko Y. G., Posypkin M. A.* Nonuniform covering method as applied to multicriteria optimization problems with guaranteed accuracy // Computational Mathematics and Mathematical Physics. 2013. Vol. 53, Issue 2. Pp. 144-157. DOI: 10.1134/S0965542513020061

[15] *Evtushenko Y., Posypkin M., Sigal I.* A framework for parallel large-scale global optimization // Computer Science-Research and Development. 2009. Vol. 23, Issue 3-4. Pp. 211-215. DOI: 10.1007/s00450-009-0083-7

[16] *Barkalov K., Gergel V.* Parallel global optimization on GPU // Journal of Global Optimization. 2016. Vol. 66, Issue 1. Pp. 3-20. DOI: 10.1007/s10898-016-0411-y

[17] *Haftka R. T., Villanueva D., Chaudhuri A.* Parallel surrogate-assisted global optimization with expensive functions–a survey // Structural and Multidisciplinary Optimization. 2016. Vol. 54, Issue 1. Pp. 3-13. DOI: 10.1007/s00158-016-1432-3

[18] *Barkalov K., Gergel V., Lebedev I.* Use of xeon phi coprocessor for solving global optimization problems // International Conference on Parallel Computing Technologies. Springer, Cham, 2015. Pp. 307-318. DOI: 10.1007/978-3-319-21909-7_31

[19] *Ferreiro A. M.* et al. Parallel two-phase methods for global optimization on GPU // Mathematics and Computers in Simulation. 2019. Vol. 156. Pp. 67-90. DOI: 10.1016/j.matcom.2018.06.005

[20] *Strongin R. G.* et al. Generalized Parallel Computational Schemes for Time-Consuming Global Optimization // Lobachevskii Journal of Mathematics. 2018. Vol. 39, Issue 4. Pp. 576-586. DOI: 10.1134/S1995080218040133

[21] *Zhang G. W.* et al. Parallel particle swarm optimization using message passing interface // Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Vol. 1. Springer, Cham, 2015. Pp. 55-64. DOI: 10.1007/978-3-319-13359-1_5

[22] *Попов М. В., Посыпкин М. А.* эффективная реализация точных алгоритмов решения задач дискретной оптимизации на графических ускорителях // Современные информационные технологии и ИТ-образование. 2018. Т. 14, №. 2. С. 408-418. DOI: 10.25559/SITITO.14.201802.408-418

[23] *Горчаков А. Ю., Посыпкин М. А.* Сравнение вариантов многопоточной реализации метода ветвей и границ для многоядерных систем // Современные информационные технологии и ИТ-образование. 2018. Т. 14, №. 1. DOI: 10.25559/SITITO.14.201801.138-148

[24] *Gaviano M., Kvasov D. E., Lera D., Sergeyev Y. D.* Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization // ACM Transactions on Mathematical Software. 2003. Vol. 29, Issue 4. Pp. 469-480. DOI: 10.1145/962437.962444

[25] *Posypkin M., Usov A.* Implementation and verification of global optimization benchmark problems // Open Engineering. 2017. Vol. 7, Issue 1. Pp. 470-478. DOI: 10.1515/eng-2017-0050

[26] *Горчаков А. Ю., Посыпкин М. А.* Эффективность методов локального поиска в задаче минимизации энергии плоского кристалла // Современные информационные технологии и ИТ-образование. 2017. Т. 13, №. 2. DOI: 10.25559/SITITO.2017.2.247

[27] *Евтушенко Ю. Г., Лурье С. А., Посыпкин М. А., Соляев*

Ю. О. Применение методов оптимизации для поиска равновесных состояний двумерных кристаллов // Журнал вычислительной математики и математической физики. 2016. Т. 56, №. 12. С. 2032-2041. DOI: 10.7868/S0044466916120097

[28] *Posypkin M.* Automated Robot's Workspace Approximation // Journal of Physics: Conference Series. 2019. Vol. 1163, № 1. Pp. 012050. DOI: 10.1088/1742-6596/1163/1/012050

Поступила 21.04.2019; принята к публикации 20.05.2019; опубликована онлайн 25.07.2019.

Об авторах:

Посыпкин Михаил Анатольевич, главный научный сотрудник Вычислительного центра имени А.А. Дородницына РАН, Федеральный исследовательский центр «Информатика и управление» Российской академии наук (119333, Россия, г. Москва, ул. Вавилова, д. 44, кор. 2), доктор физико-математических наук, доцент, ORCID: http://orcid.org/0000-0002-4143-4353, mposypkin@gmail.com

Горбунов Алексей Степанович, магистрант, факультет вычислительной математики и кибернетики, Московский государственный университет имени М.В. Ломоносова (119991, Россия, г. Москва, Ленинские горы, д. 1), ORCID: http://orcid.org/0000-0001-6033-5978, aleksey.gorbunov.97@gmail.com

Все авторы прочитали и одобрили окончательный вариант рукописи.

References

[1] Pintér J.D. Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications. *Nonconvex Optimization and Its Applications.* Springer, Boston, MA. 2013; 6. (In Eng.) DOI: 10.1007/978-1-4757-2502-5

[2] Strongin R.G., Sergeyev Y.D. Global optimization with non-convex constraints: Sequential and parallel algorithms. *Nonconvex Optimization and Its Applications.* Springer, Boston, MA. 2013; 45. (In Eng.) DOI: 10.1007/978-1-4615-4677-1

[3] Sergeyev Y.D., Kvasov D.E. Global search based on efficient diagonal partitions and a set of Lipschitz constants. *SIAM Journal on Optimization.* 2006; 16(3):910-937. (In Eng.) DOI: 0.1137/040621132

[4] Sergeyev Y.D., Strongin R.G., Lera D. Introduction to Global Optimization Exploiting Space-Filling Curves. *SpringerBriefs in Optimization.* Springer, New York, NY, 2013. (In Eng.) DOI: 10.1007/978-1-4614-8042-6

[5] Hansen P., Jaumard B. Lipschitz optimization. *Handbook of global optimization.* Springer, Boston, MA. 1995; 407-493. (In Eng.) DOI: 10.1007/978-1-4615-2025-2

[6] Paulavičius R., Žilinskas J. Simplicial Lipschitz optimization without Lipschitz constant. In: Paulavičius R., Žilinskas J. (eds). *Simplicial Global Optimization*. Springer, New York, NY. 2014; 61-86. (In Eng.) DOI: 10.1007/978-1-4614-9093-7

[7] Evtushenko Yu.G. Numerical methods for finding global extrema (Case of a non-uniform mesh). *USSR Computational Mathematics and Mathematical Physics.* 1971; 11(6):38-54. (In Eng.) DOI: 10.1016/0041-5553(71)90065-6

[8] Chandra R. et al. Parallel programming in OpenMP. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. (In Eng.)

[9] Sato M. OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors. *15th International Symposium on System Synthesis*. Kyoto, Japan. 2002; 109-111. (In Eng.) DOI: 10.1145/581199.581224

[10] Williams A. C++ concurrency in action. Manning Publications Co., 2019. (In Eng.)

[11] Evtushenko Y., Posypkin M. A deterministic approach to global box-constrained optimization. *Optimization Letters.* 2013; 7(4):819-829. (In Eng.) DOI: 10.1007/s11590-012-0452-1

[12] Evtushenko Y.G., Posypkin M.A. A deterministic algorithm for global multi-objective optimization. *Optimization Methods and Software.* 2014; 29(5):1005-1019. (In Eng.) DOI: 10.1080/10556788.2013.854357

[13] Evtushenko Y.G., Posypkin M.A. An application of the nonuniform covering method to global optimization of mixed integer nonlinear problems. *Computational Mathematics and Mathematical Physics.* 2011; 51(8):1286. (In Eng.) DOI: 10.1134/S0965542511080082

[14] Evtushenko Y.G., Posypkin M.A. Nonuniform covering method as applied to multicriteria optimization problems with guaranteed accuracy. *Computational Mathematics and Mathematical Physics.* 2013; 53(2):144-157. (In Eng.) DOI: 10.1134/S0965542513020061

[15] Evtushenko Y., Posypkin M., Sigal I. A framework for parallel large-scale global optimization. *Computer Science-Research and Development.* 2009; 23(3-4):211-215. (In Eng.) DOI: 10.1007/s00450-009-0083-7

[16] Barkalov K., Gergel V. Parallel global optimization on GPU. *Journal of Global Optimization.* 2016; 66(1):3-20. (In Eng.) DOI: 10.1007/s10898-016-0411-y

[17] Haftka R.T., Villanueva D., Chaudhuri A. Parallel surrogate-assisted global optimization with expensive functions–a survey. *Structural and Multidisciplinary Optimization.* 2016; 54(1):3-13. (In Eng.) DOI: 10.1007/s00158-016-1432-3

[18] Barkalov K., Gergel V., Lebedev I. Use of xeon phi coprocessor for solving global optimization problems. *International Conference on Parallel Computing Technologies.* Springer, Cham. 2015; 307-318. (In Eng.) DOI: 10.1007/978-3-319-21909-7_31

[19] Ferreiro A. M. et al. Parallel two-phase methods for global optimization on GPU. *Mathematics and Computers in Simulation.* 2019; 156:67-90. (In Eng.) DOI: 10.1016/j.matcom.2018.06.005

[20] Strongin R.G. et al. Generalized Parallel Computational Schemes for Time-Consuming Global Optimization. *Lobachevskii Journal of Mathematics.* 2018; 39(4):576-586. (In Eng.) DOI: 10.1134/S1995080218040133

[21] Zhang G.W. et al. Parallel particle swarm optimization using message passing interface. *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems*. Springer, Cham. 2015; 1:55-64. (In Eng.) DOI: 10.1007/978-3-319-13359-1_5

[22] Popov M.V., Posypkin M.A. Effective Realization of Exact Algorithms for Solving Discrete Optimization Problems on

Graphic Accelerators. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2018; 14(2):408-418. (In Russ., abstract in Eng.) DOI: 10.25559/SITITO.14.201802.408-418

[23] Gorchakov A.Y. Posypkin M.A. Comparison of variants of multithreading realization of method of branches and borders for multi-core systems. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2018; 14(1):138-148. (In Russ., abstract in Eng.) DOI: 10.25559/SITITO.14.201801.138-148

[24] Gaviano M., Kvasov D.E., Lera D., Sergeyev Y.D. Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Transactions on Mathematical Software.* 2003; 29(4):469-480. (In Eng.) DOI: 10.1145/962437.962444

[25] Posypkin M., Usov A. Implementation and verification of global optimization benchmark problems. *Open Engineering.* 2017; 7(1):470-478. (In Eng.) DOI: 10.1515/eng-2017-0050

[26] Gorchakov A.Y. Posypkin M.A. The effectiveness of local search methods in the problem of finding the minimum energy of a 2-d crystal. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2017; 13(2):97-102. (In Russ., abstract in Eng.) DOI: 10.25559/SITITO.2017.2.247

[27] Evtushenko Y.G., Lurie S.A., Posypkin M.A., Solyaev Yu.O. Application of optimization methods for finding equilibrium states of two-dimensional crystals. *Computational Mathematics and Mathematical Physics.* 2016; 56(12):2001–2010. (In Eng.) DOI: 10.1134/S0965542516120083

[28] Posypkin M. Automated Robot's Workspace Approximation. *Journal of Physics: Conference Series.* 2019; 1163(1):012050. (In Eng.) DOI: 10.1088/1742-6596/1163/1/012050

Submitted 21.04.2019; revised 20.05.2019;
published online 25.07.2019.

About the authors:

Mikhail A. Posypkin, Chief Research Officer, Dorodnicyn Computing Centre of RAS, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences (44-2 Vavilov St., Moscow 119333, Russia), Dr. Sci. (Phys.-Math.), Associate Professor, ORCID: http://orcid.org/0000-0002-4143-4353, mposypkin@gmail.com

Alexey S. Gorbunov, student, Lomonosov Moscow State University (1 Leninskie Gory, GSP-1, Moscow 119991, Russia), ORCID: http://orcid.org/0000-0001-6033-5978, aleksey.gorbunov.97@gmail.com

All authors have read and approved the final manuscript.