УДК 519.8
DOI 10.53403/9785951505071_2022_217

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ОПТИМИЗАЦИИ НЕПРЕРЫВНЫХ ФУНКЦИЙ БОЛЬШОЙ РАЗМЕРНОСТИ НА ОСНОВЕ ГИБРИДИЗАЦИИ АЛГОРИТМА ОПТИМИЗАЦИИ «СЕРЫХ ВОЛКОВ»

О. В. Коваленко, Д. В. Ежов, И. А. Крючков

Российский федеральный ядерный центр –
Всероссийский НИИ экспериментальной физики, Саров

Биоинсперированный алгоритм «Серых волков» [1] хорошо себя показал на задачах оптимизации большой размерности [2]. В статье рассматривается улучшение точности версии алгоритма «серых волков» из [2] на овражных функциях Розенброка за счет добавления в рой серых волков дополнительной роли – движения части волков по сходящейся спирали, которая была позаимствована из другого биоинсперированной алгоритма «Желтой седловой барабульки» [4].

Ключевые слова: оптимизация, биоинспирированный алгоритм, алгоритм серых волков, функция Розенброка, экстремум, алгоритм желтой барабульки.

1. Описание проблемы

Алгоритм оптимизации «Серых волков» (GWO – Grey Wolf Optimization) [1] – это один из недавних биоинспированных алгоритмов оптимизации, основанный на имитации загонной охоты стаи серых волков. В среднем алгоритм очень хорошо зарекомендовал себя на различных типах функций, как по точности поиска экстремума, так и по скорости сходимости. Особенно поражает скорость сходимости алгоритма. Практически уже при 100 итерациях алгоритм находит приемлемое решение на всех тестовых функциях и на любой размерности задачи. Основные показатели эффективности алгоритма оптимизации «Серых волков» лучше, чем у алгоритма оптимизации роем частиц, который в классе биоинсперированых алгоритмом оптимизации является «классическим». При этом вычислительная сложность алгоритм оптимизации «Серых волков» [2] сравнима с алгоритмом оптимизации роем частиц. Ввиду многочисленных достоинств алгоритма оптимизации «Серых волков» за короткое время с момента первой публикации алгоритма [1] появилось много работ по его модификации. Особо привлекает модификация алгоритма, предложенная в работе [2] для поиска экстремума высокоразмерных функций с числом независимых переменных равным 10000. Единственным недостатком алгоритма, предложенного в работе [2], является сравнительно невысокая точность найденных координат минимума функции Розенброка. Причем невысокая точность найденного экстремума проявилась на всех размерностях функции Розенброка. Хотя функция Розенброка унимодальная (она имеет один минимум), но она является «овражной» функцией [3], на которой подавляющее большинство алгоритмов поиска экстремума (включая классические градиентные методы) многомерных функций дают низкую точность решения, которая практически не улучшается с увеличением числа итераций.

В варианте для N переменных функция Розенброка определяется следующим образом:

$$f(X) = \sum_{i=1}^{N-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2], \quad \forall x \in \mathbb{R}^N \qquad (1)$$

Эта функция имеет один глобальный минимум, равный 0, когда $x_i = 1, i \in [1, N]$, где N – размерность.

Предпринимались попытки улучшить качество решения данного алгоритма и избежать «застревания» в локальном экстремуме на мультимодальных функциях, например, были введены весовые коэффициенты при расчете координат каждого «волка» на следующем шаге, как описано в [2]. Тем не менее, эти попытки давали лишь процентное улучшение качества решения на различных классах тестовых функциях, и наиболее неточными выглядят решения для «овражной» функции.

2. Канонический вид алгоритма Серых волков

В каноническом алгоритме каждый волк в стае является текущей точкой пространства поиска решений, с вычисленным значением фитнес функции. По качеству решения вся стая делятся на 4 основных видов. По-одному волку (носители текущих наилучших трех решений) составляют категорию лидеров. Лидеры в оригинальном алгоритме называются $\alpha,\ \beta,\ \delta$. все остальные волки составляют категорию ω.

Математически идея загона жертвы статей волков выглядит следующим образом. Позиция каждого волка в стае на $t+1$ шаге алгоритма определяющего окружение жертвы можно записать как

$$X_i(t+1) = X_{p,i} - A_i \cdot \left| C_i \cdot X_{p,i} - X_i(t) \right|, i = 1, \dots, N \quad (2)$$

t – номер итерации, i – номер компоненты N мерного пространства, $\vec{X}(t)$ – вектор позиции рассматриваемого волка, $\overrightarrow{X_p}$ – вектор позиции рассматриваемой жертвы, которая символизирует оптимум фитнес функции и которая неизвестна.

$$A_i = 2a \cdot r_{1,i} - a \quad (3)$$
$$C_i = 2r_{2,i},$$

$r_{1,i}$ и $r_{2,i}$ – случайные величины, которые равномерно распределены на отрезке $[0,1]$, и разыгрываются независимо для каждой компоненты i N – мерного пространства, a некоторый коэффициент, который линейно уменьшается от 2 к 0 с ростом номера итерации по закону

$$a(t) = 2 - \frac{2t}{MaxIter}, \quad (4)$$

где $MaxIter$ - заданное максимальное число итераций в алгоритме.

Так как глобальный оптимум не известен, то организуются следующая сходящаяся процедура.

Пусть волки волков $\alpha,\ \beta,\ \delta$ характеризуют три лучшие текущие на шаге t позиции. Тогда новые позиции на шаге $t + 1$ для всех волков рассчитываются по следующим уравнениям.

Для каждого волка с ролью ω считаются свои вектора $\vec{X}_1, \vec{X}_2, \vec{X}_3$. Они вычисляются с учетом известных позиций волков $\alpha,\ \beta,\ \delta$ по следующим формулам

$$X_{1,i}^{(k)}(t+1) = X_i^{(\alpha)}(t) - A_i^{(\alpha)} \cdot \left| C_i^{(\alpha)} \cdot X_i^{(\alpha)}(t) - X_i^{(k)}(t) \right| \quad (5)$$

$$X_{2,i}^{(k)}(t+1) = X_i^{(\beta)}(t) - A_i^{(\beta)} \cdot \left| C_i^{(\beta)} \cdot X_i^{(\beta)}(t) - X_i^{(k)}(t) \right| \quad (6)$$

$$X_{3,i}^{(k)}(t+1) = X_i^{(\delta)}(t) - A_i^{(\delta)} \cdot \left| C_i^{(\delta)} \cdot X_i^{(\delta)}(t) - X_i^{(k)}(t) \right| \quad (7)$$

где i – номер компоненты N мерного пространства, k – номер волка с ролью ω, тогда итоговая позиция любого рассматриваемого волка определяется как

$$\vec{X}(t+1) = \frac{\vec{X}_1(t+1) + \vec{X}_2(t+1) + \vec{X}_3(t+1)}{3} \quad (8)$$

Алгоритм 1

Вход: Популяция размером *M*, общее количество итераций $MaxIter$, *N* – размерность пространства поиска.

1. Случайно инициализируется позиции *M* волков $\vec{X}_i(i = 1,2,\dots,M)$ в области поиска, инициализируется $t = 0$ – текущая итерация

2. while $(t < MaxIter)$

2.1. Рассчитывается фитнес функция для каждого волка $y_i = f(\vec{X}_i)\ (i = 1,2,\dots,M)$

2.2. Выбирается три наилучших решения из $\{y_i, i = 1,2,\dots N\}$ и определяются им соответствующие $\vec{X}^{(\alpha)}$, $\vec{X}^{(\beta)}$, $\vec{X}^{(\delta)}$

2.3. *for i* = 1:*M* % по каждому индивиду

for j = 1:*N* % по каждой компоненте размерности пространства

рассчитываются X_1, X_2, X_3 по (5)–(7),при новых r_1 и r_2,

рассчитываются компонента вектора новой позиции волка по (8)

end

end

2.4. $t = t + 1$

end while

3. return $\vec{X}^{(\alpha)}$

Заметим, что для того чтобы выполнялся на последней итерации шаг с $a(MaxIter) = 0$, в псевдокоде итераций ложно быть не $MaxIter$, а $MaxIter$ +1, т. е. в строчке 2 псевдокода необходимо написать *while* $(t < MaxIter + 1)$. Это как правило повышает точность окончательного решения на большинстве тестовых функциях.

Модификация ERGWO

В работе [2] оригинальный алгоритм из [1] был модифицирован и хорошо проявил себя на большеразмерных задачах. Модификации имели следующий вид. Вместо формулы (4) на итерациях параметр *a* менялся по закону

$$a(t) = a_{initial} - (a_{initial} - a_{final})\mu^{-t}, \tag{9}$$

где $a_{initial}$ – начальное значение параметра, которое принимается 2 как в оригинальном алгоритме, a_{final} – финальное значение параметра, которое принимается равным 0, $\mu \in [1.0001,1.005]$. $t \in \{0,1,2,\dots,MaxIter\}$.

В отличие от формулы (4), формула (9) выглядит неправильной, она дает значение a_{final} при $t = 0$ и далее стремится к значению $a_{initial}$, т. е. $\lim_{t\to\infty} a(t) = a_{initial}$. Это наводит на мысль, что в статье [2] была допущена опечатка. Чтобы сохранить убывающий тренд зависимости оригинальной статьи [1], скорее всего формула (9) должна была бы выглядеть как

$$a(t) = a_{final} + (a_{initial} - a_{final})\mu^{-t}, \tag{10}$$

Тогда, чтобы получить $a(MaxIter) = 0$, a_{final} должна быть отрицательной и вычисляться по формуле

$$a_{final} = -\frac{a_{initial}}{(\mu^{MaxIter} - 1)} \tag{11}$$

Проведенные тесты показали, что можно использовать более простую формулу для адаптации параметра *a* с ростом номера итерации

$$a(t) = 2\mu^{-t}, \tag{12}$$

При этом использование формул (4). (10) и, (11) дают примерно одинаковый результат, который зависит от общего числа итераций. В среднем предпочтительно использовать оригинальную формулу (4) или упрощенную (12).

Однако и следующие изменения оригинального алгоритма, сделанные в [2], выглядят как ошибка.

Вместо формулы (8), которая дает координату волка, как среднеарифметическое значение между координатами, посчитанными по направлением к волкам лидерам α, β, δ, в статье [2] предложено рассчитывать усреднение по формуле:

$$\vec{X}(t+1) = \frac{w_1\vec{X}_1(t+1) + w_2\vec{X}_2(t+1) + w_3\vec{X}_3(t+1)}{3}, \quad (13)$$

где

$$w_1 = \frac{|\vec{X}_1(t+1)|}{|\vec{X}_1(t+1)| + |\vec{X}_2(t+1)| + |\vec{X}_3(t+1)|}, \quad (14)$$

$$w_2 = \frac{|\vec{X}_2(t+1)|}{|\vec{X}_1(t+1)| + |\vec{X}_2(t+1)| + |\vec{X}_3(t+1)|}, \quad (15)$$

$$w_3 = \frac{|\vec{X}_3(t+1)|}{|\vec{X}_1(t+1)| + |\vec{X}_2(t+1)| + |\vec{X}_3(t+1)|}. \quad (16)$$

Формулы (13)–(16) выглядят как ошибка потому, что они зависят от выбора системы отсчета, а формула (13) является взвешенным усреднением, дополнительно деленным на 3.

Удивительным является, что так введенное «усреднение» улучшило точность оригинального алгоритма.

Нетрудно увидеть, что формула (13), по сути, корректирует делитель в формуле (8). Вместо делителя 3, выражения (13)–(16) соответствуют делителю 9 в формуле (8) на последних итерациях, где $w_1 \approx w_2 \approx w_3$, так как из-за сходимости в конце итерационного процесса $\vec{X}_1 \approx \vec{X}_2 \approx \vec{X}_3$.

Увеличение делителя в формуле (8) резко ускоряет сходимость на начальных итерациях, особенно при большеразмерных задачах. На средних размерностях опытным путем было получено, что в среднем замена делителя 3 на делитель 4 в формуле (8) улучшает результат.

Однако окончательная точность алгоритма на многих тестовых функциях остается хуже, чем у алгоритма PSO при увеличении числа итераций до 500. Очевидно, что агентам в алгоритме GWO и его модификациях явно не хватает исследовательской составляющей, особенно во второй половине итерационного процесса.

Идея алгоритма желтой седловой барабульки

В работе [4] новый алгоритм поиска глобального оптимума, инспирированный поведением желтой седловой барабульки (Yellow Saddle Goatfish). В алгоритме все особи разбиваются на заранее заданное количество групп. В каждой группе есть основные роли: 1) одна рыба преследователь (chaser fish) и 2) рыбы блокировщики (blocker fish).

В n-мерном пространстве поиска новая позиция рыбы блокировщика определяется по формуле

$$\Phi_l^{(i)}(t+1) = \Phi_l^{(i)}(t) + \gamma \cdot Levy_{t,l}^{(i)}(\lambda), \quad (17)$$

где l – номер группы, $Levy(\lambda)$ является случайным полетом Леви, который определяется по формуле

$$Levy_{t,l}^{(i)}(\lambda) = \frac{u_l}{|v_l|^{1/\lambda}}\left(\vec{\Phi}_l(t) - \vec{\Phi}_{best}(t)\right), \quad (18)$$

$\vec{\Phi}_{best}(t)$ – наилучшее на шаге t решение, u_l и v_l – нормально распределенные случайные величины, разыгрываемые для каждой пространственной компоненты i независимо

$$u = N(0, \sigma_u^2),$$
$$v = N(0, \sigma_v^2), \quad (19)$$

$$\sigma_u = \left(\frac{\Gamma(1+\lambda) sin\frac{\pi\lambda}{2}}{\Gamma\left(\frac{1+\lambda}{2}\right) \lambda 2^{(\lambda-1)/2}} \right)^{1/\lambda}, \quad \sigma_v = 1, \tag{20}$$

где $\Gamma(x)$ гамма функция.

Заметим, что соотношение (18) дает нулевое смещение для наилучшего решения. В статье [4] предложено наилучшее текущее решение подвергать случайному смещению, где каждая компонента вектора позиции независимо смещается как

$$S = \gamma \frac{u}{|v|^{1/\lambda}}. \tag{21}$$

Предложение спорное и нуждается в численной проверке. Скорее всего, оно играет дестабилизирующую роль, уводя наилучшего преследователя от глобального оптимума.

Нас интересует уравнение изменения координат для индивидов второй роли. Они двигаются по случайной логарифмической спирали вокруг рыбы преследователя в группе согласно уравнению

$$\varphi_i^{(g)}(t+1) = D_i^{(g)}(t) \cdot e^{b\rho_i} \cdot cos2\pi\rho_i + \Phi_i^{(i)}(t), \tag{22}$$

где i – номер пространственной компоненты, g – индекс рыбы блокировщика в рассматриваемой группе, l – номер группы, $\vec{\Phi}_i(t)$ – позиция рыбы преследователя в группе l, ρ случайная величина, равномерно распределенная на отрезке $[a, 1]$, который определяет, как близко блокировщик к рыбе-преследователю, рассматривая путь по спирали. Для усиления эксплуатационных свойств а линейно уменьшается от 1 до –2 по мере того, как номер итерации увеличивается. Позиция рыбы преследователя достигается, когда $\rho = -2$. Следовательно, самая близкая позиция к рыбе преследователю получается, когда $\rho \approx -2$, а самая дальняя, когда $\rho = 1$. Уравнение (22) применяется для каждой компоненты вектора позиции блокировщика. То есть для каждой компоненты разыгрывается случайная величина ρ и случайная величина r, входящая в формулу (23).

Параметр b в уравнении (22) константа, которая определяет форму и направление спирали, в этом подходе b = 1. Член Dg является расстоянием между текущим положением рыбы блокировщиком φ_g^l и рыбой преследователем $\boldsymbol{\Phi}_l$ в кластере $\boldsymbol{c}_l$, описанным как:

$$D_i^{(g)}(t) = \left| r_i \cdot \Phi_i^{(i)}(t) - \varphi_i^{(g)}(t) \right|, \tag{23}$$

где r_i случайное число, равномерно распределенное на отрезке $[-1,1]$.

На каждом шаге внутри каждой группы выбирается наилучшее решение. Рыбе с наилучшим решением в кластере назначается роль преследователь. Среди преследователей выбирается наилучшее глобальное решение.

Предложенная модификация алгоритма серых волков

В данной работе предложен параллельный алгоритм, основанный на гибридизации варианта алгоритма оптимизации «Серых волков», предложенного в работе [2], и алгоритма поведения Желтой Седловой Барабульки (Yellow Saddle Goatfish behavior) [4], за счет добавления новой роли в алгоритм серых волков, поведение которой соответствует поведению рыб блокировщиков из алгоритма [4].

Алгоритм «Серых волков» получил следующие нововведения:

1. В рой добавлена новая роль s – волки загоняющие жертву по сходящейся логарифмической спирали. Волки с это ролью начинают двигаться в многомерном пространстве (плоскость движения при этом получается как результат разигрывания случайных величин в формуле (21) и (22) вокруг предполагаемого (текущего) наилучшего решения, число (процент) волков, двигающихся по спирали, является динамической величиной;
2. Измерен алгоритм перераспределения ролей;
3. Экспериментально подобрано оптимальная доля волков, двигающихся по спирали
4. Число лидеров может быть переменной величиной.

Переменное число волков-лидеров

Вместо формул (5)–(8) для каждого индивида с ролью ω будем использовать следующие формулы

$$X_{\zeta,i}^{(k)}(t+1) = X_i^{(\zeta)}(t) - A_i^{(\zeta)} \cdot \left| C_i^{(\zeta)} \cdot X_i^{(\zeta)}(t) - X_i^{(k)}(t) \right|, \quad (24)$$

где ζ – номер лидерской роли, $\zeta \in \{\alpha, \beta, \delta, \dots\}$,

$$X_i^{(k)}(t+1) = \sum_{\zeta \in \{\alpha,\beta,\delta,\dots\}}^{L} w_\zeta X_{\zeta,i}^{(k)}(t+1), \quad (25)$$

L – количество лидеров в стае, w_ζ – вес лидерской роли с нормировкой в виде неравенства

$$\sum_{\zeta \in \{\alpha,\beta,\delta,\dots\}}^{L} w_\zeta \leq 1. \quad (26)$$

Веса подбираются опытным путем. Мы предлагаем использовать для трех ролей веса $w_\alpha = w_\beta = w_\delta = \frac{1}{4}$

Уравнение движения для волков с ролью s рассчитывается согласно уравнению

$$X_i^{(g)}(t+1) = D_i^{(g)}(t) \cdot e^{b\rho_i} \cdot cos 2\pi\rho_i + X_i^{(\alpha,\beta,\dots,L)}(t+1), \quad (27)$$

g номер волка с ролью s, $X_i^{(\alpha,\beta,\dots,L)}(t+1)$ – позиция жертвы, $D_i^{(g)}(t)$ – рассчитывается по формуле аналогичной формуле (23)

$$D_i^{(g)}(t) - \left| r_i \cdot \Phi_i^{(i)}(t) - X_i^{(\alpha,\beta,\dots,L)}(t+1) \right|. \quad (28)$$

Позиция жертвы рассчитывается по формуле

$$X_i^{(\alpha,\beta,\dots,L)}(t+1) = \sum_{\zeta \in \{\alpha,\beta,\delta,\dots\}}^{L} w_\zeta X_i^{(\zeta)}(t+1). \quad (29)$$

Сортировка волков

Алгоритм сортировки волков включает в себя следующее:

1. Определение координат предполагаемого наилучшего решения;
2. Определения координат всех волков, включая волков-лидеров;
3. Определение расстояния от всех волков до предполагаемого наилучшего решения;
4. Сортировка волков таким образом, что, волки, расположенные дальше всего от предполагаемого оптимального решения, принимают роль ω. Волки не вошедшие в число лидеров, и находящиеся ближе к оптимальному решению принимают роль s.

Данное улучшение позволяет динамически «перетасовывать» волков так, что самые дальние начинают подтягиваться к остальным, не оставаясь слишком далеко и, тем самым, не выпадая из числа волков, имеющих шанс найти лучшее решение, чем предполагаемое на данной итерации.

Определение доли волков двигающихся по спирали

Одним из нововведений стал внедренный алгоритм расчета оптимального числа волков, двигающихся по спирали.

В качестве исходной точки эмпирически был найден условно-оптимальный процент волков, двигающихся по спирали согласно уравнению (27), от общего числа – 0.9. Далее, в качестве изначального значения устанавливается оно. В процессе итераций, если значение фитнесс-функции не улучшается (меняется) в течение определенного числа итераций, процент волков с ролью s начинает

изменяться в диапазоне [0.85, 0.95]. Эксперименты показали, что выход за пределы этого диапазона с вероятностью p > 0.9 не приводит к улучшению качества решения.

При изменении значения фитнесс-функции (улучшении решения) процент волков, двигающихся по спирали, фиксируется на том значении, на котором произошло улучшение.

Алгоритм 2

Вход: Популяция размером *M*, общее количество итераций $MaxIter$, *N* –размерность пространства поиска.

1. Случайно инициализируются позиции *M* волков $\vec{X}_1(i = 1,2,\dots,M)$ в области поиска, инициализируется $t = 0$ – текущая итерация

2. while $(t < MaxIter)$

2.1. Распределение популяции по процессам

2.2. Вычисляется фитнес функция для каждого волка $y_t = f(\vec{X}_t)\ (i = 1,2,\dots,M)$.

2.3. Выбирается *L* наилучших решений из $\{y_t, i = 1,2,\dots N\}$ и определяются им соответствующие $\vec{X}^{(\zeta)}(t), \zeta \in \{\alpha, \beta, \delta, \dots, L\}$.

2.4. *If* (решение не улучшилось в течение определенного числа итераций):

Динамически меняем процент волков с ролью s (растет доля из не лидеров от 0.85 до 0.95 с шагом 0.01)

end if

2.5. *for i* = 1:*M* % по каждому индивиду

for j = 1:*N* % по каждой компоненте размерности пространства

If(*i* < *L* + 1)

по формуле (24) и (25) рассчитываются новые позиции лучших решений

новые

else If(*L* < *i* < *S*) % *S* – номер упорядоченных волков, отсекающий роль *s* от роли ω

рассчитываются $X_j^{(i)}(t+1)$ по формулам (28) и (27),

else

рассчитываются $X_{\alpha,j}^{(i)}, X_{\beta,j}^{(i)}, X_{\delta,j}^{(i)}, \dots$ по формуле (24), при новых r_1 и r_2

рассчитываются компонента вектора новой позиции волка по (26)

end If

end for

If(i==L+1)

По формуле (30) рассчитывается позиция жертвы

end If

end for

2.6. $t = t \mid 1$

end while

3. *return* $\vec{X}^{(\alpha)}$

3. Полученные результаты

В результате внесенных изменений в алгоритм, точность решения значительно повысилась на всех классах функций и на всех размерностях тестовых задач. Размерность задач варьировалась от 2 до 20000.

Рассмотрим улучшение качества решения на примере функции Розенброка. Результаты представлены в табл. 1 и на рис. 1 (для наглядности применена логарифмическая шкала).

Таблица 1

Результаты расчетов

Размерность	Без новой роли s	С ролью s	Коэффициент улучшения решения (разы)
2	0,467702	0,000003	155900,6667
4	2,996389	0,000018	166466,0556
40	38,974482	0,0077	5061,621039
400	398,99956	0,011349	35157,24381
1000	998,999865	0,007255	137698,1206
2000	1999	0,000888	2251126,126
3000	2998,999983	0,013721	218570,0738
4000	3998,975809	0,026884	148749,2862
5000	4998,989612	0,019844	251914,413
6000	5999	0,000036	166638888,9
7000	6998,991559	0,000184	38037997,6
8000	7998,979025	0,008828	906091,8696
9000	8998,982006	0,002281	3945191,585
10000	9998,98238	0,006632	1507687,331
20000	19998,962761	0,000616	32465848,64

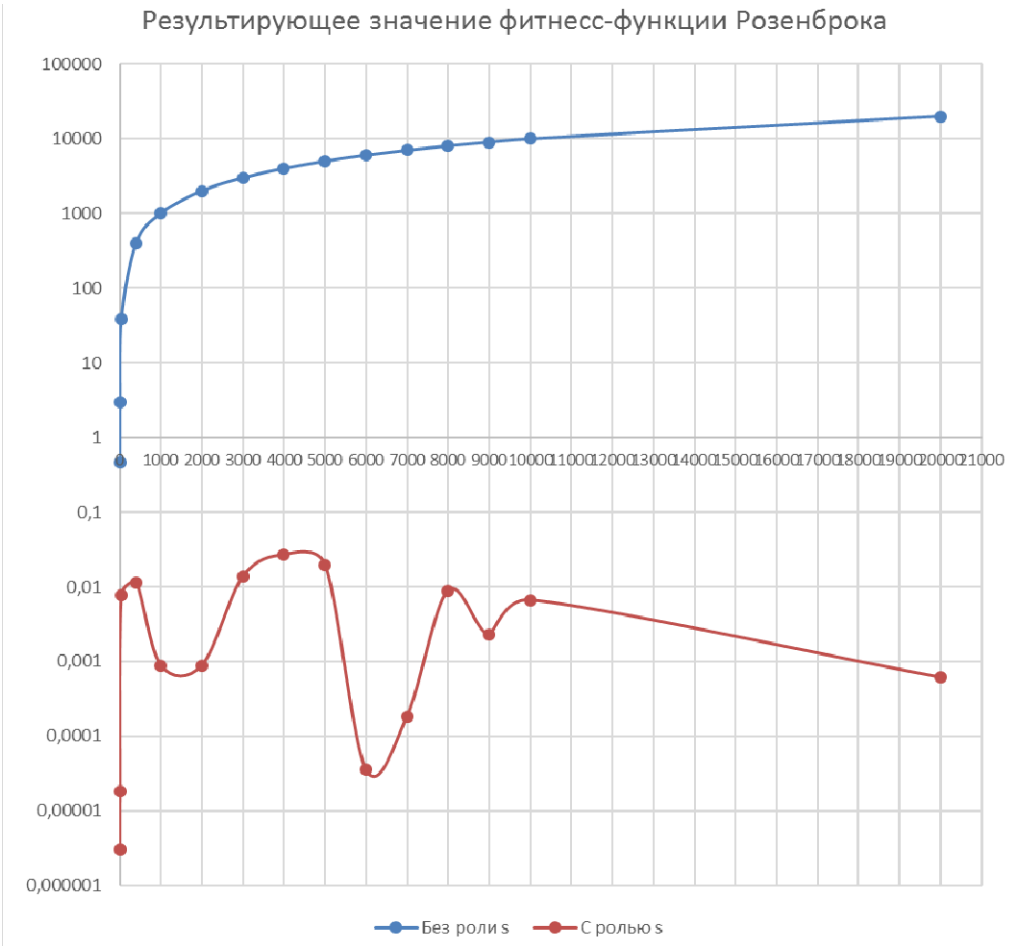

Рис. 1. График зависимости качества решения

На рис. 2 приведены графики зависимости счетного времени от размерности задачи для функции Розенброка.

На рис. 2 видно, что время расчета алгоритма без учета расчетов фитнес функции имеет очень пологий, почти линейный вид.

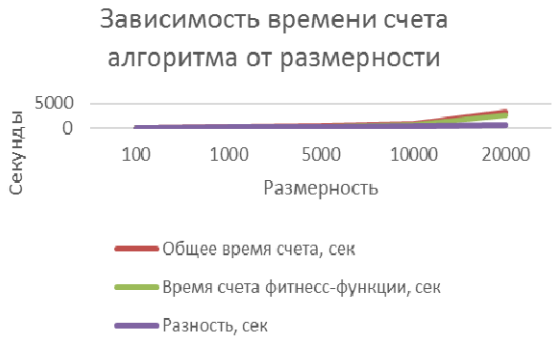

Рис. 2. Зависимость времени счета от размерности задачи для алгоритма в целом, для расчета фитнес функции и их разности

Заключение

В результате описанных выше модификаций удалось значительно улучшить качество конечного решения.

В дальнейшем планируется численно определить значимость s и ω ролей в точности алгоритма. Не до конца выяснены наилучшие динамические зависимости для констант алгоритма и числа лидеров.

Литература

1. Mirjalili S., Mirjalili S. M., Lewis A. «Grey wolf optimizer» // Advances in Engineering Software. 2014. Vol. 69. P. 46–61.

2. Long W., Cai S., Jiao J., Tang M. «An efficient and robust grey wolf optimizer algorithm for large-scale numerical optimization» // Soft Computing. – 2019. Vol. 3.

3. Rosenbrock H. H. «An automatic method for finding the greatest or least value of a function» // The Computer Journal. 1960. T. 3. P. 175–184.

4. Zaldivar D., Morales B., Rodriguez A., Valdivia-G A., Cuevas E., Pérez-Cisneros M. «A novel bio-inspired optimization model based on Yellow Saddle Goatfish behavior» // BioSystems. 2018. Vol. 174. P. 1–21.

PARALLEL ALGORITHM TO OPTIMIZE CONTINUOUS HIGH-DIMENSIONAL FUNCTIONS ON THE BASIS OF HYBRIDIZATION OF THE GREY WOLF OPTIMIZER

O. V. Kovalenko, D. V. Ezhov, I. A. Kryuchkov

Russian Federal Nuclear Center –
All-Russian Scientific Research Institute of Experimental Physics, Sarov

A bio-inspired Grey Wolf Optimization (GWO) algorithm [1] is well-reputed in problems of high dimensionality optimization [2]. The paper describes accuracy improvement of the Grey Wolf algorithm option from [2] on Rosenbrock «ravine» functions due to giving an additional role to the swarm of grey wolves – movement of a part of wolves along the converging spiral borrowed from another bio-inspired Yellow Saddle Goatfish algorithm [4].

Key words: optimization, bio-inspired algorithm, Grey Wolf algorithm, Rosenbrock function, extremum, Yellow Saddle Goatfish algorithm.