

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

Факультет Информационных технологий и программирования

Направление (специальность) Прикладная математика и информатика

Квалификация (степень) Магистр прикладной математики и информатики

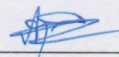
Кафедра Компьютерных технологий Группа 6538

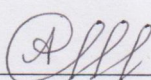
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

Разработка эффективной параллельной реализации

эволюционной стратегии с адаптацией ковариационной

Автор магистерской диссертации Архипов В. С. 
(Фамилия, И., О.)

Научный руководитель Фильченков А. А. 
(Фамилия, И., О.)

Руководитель магистерской программы Васильев В. Н.
(Фамилия, И., О.)

К защите допустить
Зав. кафедрой Васильев В. Н.
(Фамилия, И., О.)

“ ” 20 г.

Санкт-Петербург, 2015 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	6
1. Обзор предметной области	8
1.1. Эволюционные стратегии.....	8
1.1.1. Введение.....	8
1.1.2. Используемые обозначения	8
1.1.3. Основные принципы.....	9
1.2. Описание алгоритма для модификации.....	18
1.2.1. LM-CMA-ES	18
1.3. Функции тестирования эволюционных алгоритмов.....	23
1.3.1. Функция де Джонга («sphere»).....	23
1.3.2. Функция Розенброка	24
1.3.3. Функция Растригина.....	24
1.3.4. Функция гиперэллипсоида.....	25
2. Описание предлагаемого подхода.....	26
2.1. Основные идеи модификации алгоритма LM-CMA-ES	26
2.2. Псевдокод предлагаемого алгоритма	28
2.3. Реализация предлагаемого алгоритма.....	30
2.3.1. Пакет AlgorithmParameters	31
2.3.2. Пакет IndividualCalculation.....	32
2.3.3. Пакет Utility	34
2.3.4. Класс CMAEvolutionStrategy.....	35
2.3.5. Класс LMCMA.....	35
2.3.6. Класс Starter	35
Выводы по главе 2	35
3. Экспериментальное исследование	36
3.1. Методология проведения эксперимента	36
3.2. Конфигурация эксперимента	37
3.3. Результаты эксперимента	41
3.3.1. Функция де Джонга.....	41
3.3.2. Функция Растригина.....	42
3.3.3. Функция Розенброка	43
3.3.4. Гиперэллипсоид.....	43
Выводы по главе 3	44

ЗАКЛЮЧЕНИЕ.....	45
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	46
ПРИЛОЖЕНИЕ А. Результаты эксперимента	49

ВВЕДЕНИЕ

Эволюционные алгоритмы [1–5] — парадигма решения поисковой задачи, используемая для решения задач сложной оптимизации, для которых неизвестно других способов эффективного решения. Они относятся к семейству алгоритмов, решающих задачу по принципу «черного ящика» (рисунок 1), на непрерывном пространстве значений. Такие алгоритмы изучают проблему, применяя интересующую функцию, фитнес-функцию к различным решениям задачи, векторам размера n , на поисковом пространстве \mathbb{R}^n . В основе их действия лежат принципы биологической эволюции: решения задачи могут «мутировать» (немного измениться), «скрещиваться» (новые решения могут быть получены путем заимствования частей других решений), пройти отбор (только часть решений «выживает» — переходят на следующую итерацию).

Одна из самых главных черт эволюционных алгоритмов — возможность параллелизации. Большая часть эволюционных алгоритмов, как например, генетические алгоритмы, могут вычислять решения параллельно, что позволяет эффективно использовать их на многопроцессорных и распределенных системах. Однако большая часть алгоритмов использует принцип поколений: вычисляется группа особей, а затем происходят модификации. Это делает их параллельно неэффективными: быстро рассчитавшие свои решения потоки будут простаивать в ожидании других потоков этого поколения. Чтобы избежать этой проблемы, разработан тип алгоритмов, называемых стационарные. В своей основе они не имеют поколений, а потому могут теоретически быть модифицированы до полностью асинхронных.

Алгоритм CMA-ES — эволюционная стратегия с адаптацией ковариационной матрицы, как следует из названия, использует ковариационную матрицу для отслеживания зависимостей между переменными решений. Данный алго-

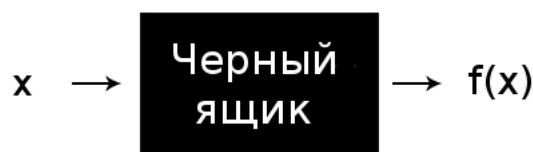


Рисунок 1 – Общий принцип черного ящика. В качестве входных данных — x -вектор. Известно только значение $f(x)$.

ритм показал очень хорошую способность к сходимости на большом количестве задач, включая неразрывные, мультимодальные и сильно зашумленные функции. Однако данный алгоритм имеет высокую вычислительную сложность $O(n^2)$ за одно вычисление функции, где n — размерность пространства, на котором изучается функция.

Были разработаны модификации этого алгоритма, например, sep-CMA-ES, который имеет сложность $O(n)$, но имеет производительность хуже на неразрывных функциях.

В данной работе предлагается steady-state модификация алгоритма CMA-ES, которая при достаточной сложности оптимизируемой функции приближается к полной асинхронности.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Эволюционные стратегии

В данном разделе приводится краткое введение в теорию эволюционных стратегий, рассматриваются их основные принципы построения.

1.1.1. Введение

Эволюционные стратегии [1–5] — способ решения задачи поиска, основанный на принципах биологической эволюции. Они относятся к семейству эволюционных алгоритмов, решающих задачу путем повторения небольших стохастических изменений, вслед за которым следует отбор: на каждой итерации новая особь генерируется из родителей (решения, которые уже рассматривались), вычисляется ее функция приспособленности и, при условии большего ее значения, особь выживает и переходит в следующую итерацию.

Эволюционные стратегии обычно используются на непрерывном пространстве решений — \mathbb{R}^n , где решением является вектор x размерности рассматриваемого пространства n . Исследуется функция $f : \mathbb{R}^n \rightarrow \mathbb{R}, x \rightarrow f(x)$, которую необходимо минимизировать.

Эволюционные стратегии обычно используются в случае неизвестного поведения f . Такая ситуация довольно часто встречается в практических задачах. Например, при проведении дорогостоящего эксперимента получают некоторые данные. Необходимо найти минимум некоторой функции, обобщающей эти данные. Такая ситуация, когда о функции известно только ее вычислимое значение, называется «черным ящиком». Подробное изображение приведено на рисунке 1.

1.1.2. Используемые обозначения

В дальнейшем используются следующие обозначения:

- $n \in \mathbb{N}$ — размер пространства поиска, является одним из задаваемых параметров;
- P — набор особей, популяция;
- $w_k \in \mathbb{R}$ — веса рекомбинаций;
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$ — функция для минимизации;
- $\rho \in \mathbb{N}$ — количество родителей для рекомбинации;
- $\sigma > 0$ — размера шага, среднеквадратичное отклонение;
- $\sigma \in \mathbb{R}_+^n$ — вектор шагов, среднеквадратичных отклонений;

- $x, x^{(t)}, x_k \in \mathbb{R}^n$ — решение, вектор родителя или вектор k -ой особи. Элемент поискового пространства \mathbb{R}^n , являющийся аргументом для функция приспособленности $f : \mathbb{R}^n \rightarrow \mathbb{R}$;
- $\lambda \in \mathbb{N}$ — количество особей в поколении;
- $\mu \in \mathbb{N}$ — количество особей, отбираемых для рекомбинации;
- $\mu_w = \frac{(\sum_{k=1}^{\mu} |w_k|)^2}{\sum_{k=1}^{\mu} w_k^2}$ — эффективная масса или эффективное количество родителей. Всегда выполнено $\mu_w \leq \mu$ или $\mu_w = \mu$, если все веса рекомбинаций w_k одинаковы по модулю.

1.1.3. Основные принципы

Эволюционные стратегии построены на основных принципах биологической эволюции. Рассмотрим популяцию P , состоящую из особей. Каждая особь состоит из решения-вектора $x \in \mathbb{R}^n$ и его значения приспособленности — $f(x)$. В некоторых случаях особь в популяции может быть всего одна. Каждая особь может использоваться как в качестве родителя, так и в качестве потомка, в зависимости от контекста.

Общий алгоритм состоит из трех основных процессов:

- **Скрещивание.** Один или несколько родителей выбираются из поколения и используются для генерации новой особи при помощи дублирования и рекомбинации этих родителей. Существует два основных сценария выбора:
 - Независимая от функции приспособленности рекомбинация. При отборе особей в родители значения функции приспособленности не учитываются, и потому он может быть или *детерминированным* или *стохастическим*.
 - Рекомбинация на основе функции приспособленности. При отборе особей в родители учитывается ранг их значений функции приспособленности среди других особей.
- **Мутация и контроль параметров.** На этапе мутации [6] особи подвергаются небольшим случайным *независимым* изменениям. Размер этих изменений напрямую зависит от *эндогенных параметров* алгоритма, которые также называются *параметрами контроля*. Это внутренние параметры алгоритма, изменяющиеся со временем. Примером такого параметра является *размер шага* — σ . Также существуют и экзогенные параметры, которые задаются в начале алгоритма. Например, *количество родите-*

лей — μ . Важно также отметить, что *независимость* мутаций и рекомбинаций является ключевым принципом эволюционных стратегий. Разнообразие, получаемое при рекомбинации или мутации, позволяет привнести в особь непредсказуемый набор информации. Отбор, наоборот, отдает предпочтение решениям с лучшими значениями функции приспособленности.

- **Отбор.** Размер популяции уменьшается при помощи использования принципа естественного отбора.

1.1.3.1. Нотация $(\mu/\rho^\dagger;\lambda)$ для описания скрещивания

Эволюционная стратегия — итеративный процесс. Для описания процесса скрещивания по итерациям существует специальная нотация. Алгоритм $(\mu/\rho^\dagger;\lambda)$ -ES, часто также обозначаемый как $(\mu^\dagger;\lambda)$ -ES. Данные нотации обозначают следующее:

- на каждой итерации отбираются μ родителей;
- на каждой итерации из μ родителей используются ρ . Данный выбор происходит на основе, например, веса;
- на каждой итерации генерируется λ особей;
- \dagger описывает, учитывается ли на стадии отбора *возраст* особей. Может использоваться или $+$ или $,$. $+$ — возраст не учитывается и μ лучших из $\mu + \lambda$ особей выбирается. Отбор элитический, то есть родители — это μ лучших особей за все время работы алгоритма. При $,$ отборе особи умирают после одной итерации и только новые особи переходят в следующее поколение. Таким образом, отбирается μ особей из λ ;
- иногда используется символ ρ для описания типа скрещивания — ρ_I для среднеквадратичного или ρ_W для взвешенного;

1.1.3.2. Скрещивание

На этапе скрещивания, происходит рекомбинация информации от нескольких родителей для генерации одной новой особи. Явление, когда для генерации используется больше двух родителей ($p > 2$), называется мультирекомбинацией.

Различают несколько основных типов рекомбинаций:

1. **Дискретная** или доминантная, обозначается $(\mu/\rho_D^\dagger;\lambda)$, также используется в генетических алгоритмах как равномерное скрещивание. Каждая

компонента x -вектора случайным образом наследуется от одного из ρ родителей. Для ρ родителей, различных в каждой компоненте, результат находится в равномерном распределении среди ρ^n различных векторов x .

2. **Промежуточный (Средний, Центральный)** вариант рекомбинации, обозначаемый как $(\mu/\rho_I^+; \lambda)$, считает среднее значение среди всех ρ родителей, т.е. центроид.
3. **Взвешенная [5, 7, 8]**, $(\mu/\rho_W^+; \lambda)$. В этом методе рекомбинации используется среднее взвешенное значение, среди ρ родителей. Это значение зависит от ранга значения функции приспособленности, среди других родителей — чем меньше значение функции приспособленности, тем больше вес. При равенстве весов превращается в центральную.

1.1.3.3. Мутация

Оператор мутации добавляет небольшое возмущение в результат рекомбинации. Это возмущение берется из нормального распределения $\mathcal{N}(0, C)$ с математическим ожиданием 0 и ковариационной матрицей $C \in \mathbb{R}^{n \times n}$. Тогда $x + \mathcal{N}(0, C) \sim \mathcal{N}(x, C)$, т.е. x определяет ожидаемое значение новой особи. Также $x + \mathcal{N}(0, C) \sim x + C^{\frac{1}{2}}\mathcal{N}(0, I)$, т.е. линейная трансформация $C^{\frac{1}{2}}$ генерирует необходимое распределение из вектора $\mathcal{N}(0, I)$. На рисунке 2 изображены различные распределения для пространства $n = 2$, на основе которых можно ввести понятия трех основных типов мутации.

1. **Сферическая/изотропическая.** График приведен на рисунке 2 левая секция. Ковариационная матрица пропорциональна единичной, т.е. распределение $\sigma\mathcal{N}(0, I)$ с шагом $\sigma > 0$. Распределение сферическое и независимо от поворотов вокруг ожидаемого значения. Например, алгоритм (1+1)-ES использует такую мутацию.
2. **Осевая.** График приведен на рисунке 2 средняя секция. Ковариационная матрица является диагональной, т.е. распределение $\mathcal{N}(0, \text{diag}(\sigma)^2)$, где σ — вектор по координатным стандартным отклонениям, а матрица $\text{diag}(\sigma)^2$ имеет собственные значения σ_i^2 , с собственными векторами e_i . Тогда основные оси эллипсоида параллельны осям координат (рис 2). Этот случай включает в себя предыдущий случай.
3. **Общий случай.** График приведен на рисунке 2 правая секция. Ковариационная матрица симметрична и положительно-определена

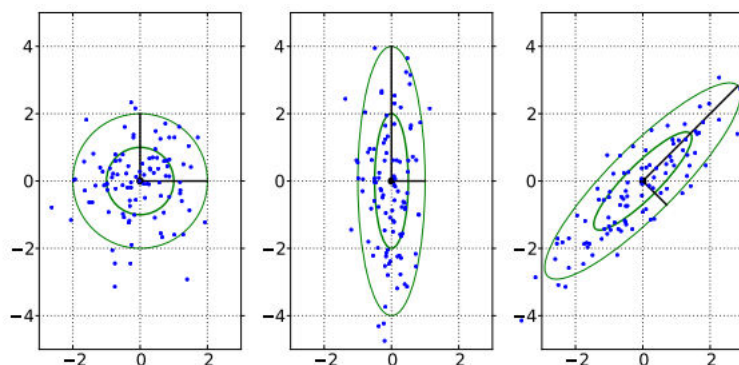


Рисунок 2 – Слева направо: ковариационная матрица — I (изотропическое распределение), диагональная матрица $\begin{pmatrix} 1/4 & 0 \\ 0 & 4 \end{pmatrix}$ (осевое) и $\begin{pmatrix} 2.125 & 1.875 \\ 1.875 & 2.125 \end{pmatrix}$ с такими же собственными значениями как диагональная матрица.

($\forall x \neq 0 : x^T C x > 0$), в общем случае — не диагональна и имеет $(n^2 + n)/2$ степеней свободы. Общий случай включает в себя все, рассмотренные ранее.

В первом и во втором случаях, вариации переменных независимы друг от друга, они некоррелированы. Это ограничение делает невозможным их использование в практических задачах.

Операторы мутации независимы одновременно в нескольких направлениях, являются симметричными и потому имеют математическое ожидание, равное нулю. Поэтому при использовании только мутации достижение лучшего значения функции приспособленности практически невозможно. Сферический оператор мутации имеет одинаковое распределение во всех направлениях. Общий оператор независим от выбора системы координат, пока неизвестно C , соответственно, независим от представления решения x .

1.1.3.4. Два основных шаблона

Чаще всего эволюционные стратегии использует один из двух основных шаблонов построения алгоритма.

Рассмотрим каждый из них подробнее.

1. **Шаблон 1.** См. листинг 1. Изначально на входе алгоритма известна популяция, состоящая, как минимум, из μ особей вида $(x_k, s_k, f(x_k))$, где $k = 1, \dots, \mu$. Вектор $x_k \in \mathbb{R}^n$ — вектор-решение, s_k хранит в себе внутренние эндогенные параметры алгоритма, например, количество успешных особей или размер шага, который в основном отвечает за мутацию x (строка 6). Значение s_k может быть одинаковым для всех k . В каждом по-

Листинг 1 – Шаблон 1

Require: $n, \rho, \mu, \lambda \in \mathbb{N}^+$

Инициализация: $P = \{(x_k, s_k, f(x_k)) | 1 \leq k \leq \mu\}$

while Пока не достигли стоп условия **do**

for $k \in \{1, \dots, \lambda\}$ **do**

$(x_s, s_k) = \text{recombine}(\text{select_mates}(/rho, P))$

$s_k \leftarrow \text{mutate}_s(s_k)$

$x_k \leftarrow \text{mutate}_x(s_k, x_k) \in \mathbb{R}^n$

end for

$P \leftarrow P \cup \{(x_k, s_k, f(x_k)) | 1 \leq k \leq \lambda\}$

$P \leftarrow \text{select_by_age}(P)$ //Шаг для ‘+’

$P \leftarrow \text{select_}\mu\text{_age}(\mu, P)$ // ранжирование по ФП

end while

колении, генерируются первые λ особей (строки 3-6) с использованием $\rho \leq \mu$ особей из P (строка 4). Затем производится мутация s (строка 5) и x (строка 6). Новые особи добавляются в P (строка 7). Старые особи (каждая особь имеет «срок жизни»: количество поколений, прошедших после ее генерации) удаляются из поколения (строка 8). После всех этих действий в P остается μ лучших особей.

2. **Шаблон 2.** См. листинг 2. Изначально на входе алгоритма только один родительский центроид. Мутация использует родительский центроид для получения новой особи, а рекомбинация теперь происходит в конце цикла (строка 8) и генерирует новый родительский центроид.

Листинг 2 – Шаблон 2

Require: $n, \lambda \in \mathbb{N}^+$

Инициализация: $x \in \mathbb{R}^n, s, P = \{\}$

while Пока не достигли стоп условия **do**

for $k \in \{1, \dots, \lambda\}$ **do**

$(x_s, s_k) = \text{recombine}(\text{select_mates}(/rho, P))$

$s_k \leftarrow \text{mutate}_s(s_k)$

$x_k \leftarrow \text{mutate}_x(s_k, x_k) \in \mathbb{R}^n$

$P \leftarrow P \cup \{(x_k, s_k, f(x_k))\}$

end for

$P \leftarrow \text{select_by_age}(P)$ //Шаг для ‘+’

$(x, s) \leftarrow \text{recombine}(P, x, s)$

end while

1.1.3.5. Контроль параметров

Одной из важнейших частей эволюционной стратегии является выбор метода контроля мутации [5]. Например, можно рассмотреть рисунке 2, левая секция. На нем λ контролирует скорость сходимости, потому эффективный метод контроля должен увеличивать шаг, при условии, что это приведет к большей скорости сходимости, и уменьшать в противоположном случае.

Цель контроля параметров — это свести внутренние (динамические) параметры алгоритма к их оптимальным значениям. Эти оптимальные значения могут сильно изменяться в зависимости от времени и текущей позиции в поисковом пространстве. В самом общем случае оператор мутации имеет (n^2+n) степеней свободы.

Существует несколько различных техник контроля параметров. Основные техники перечислены ниже.

1. **Правило 1/5 (The 1/5th Success Rule).** Этот метод контроля является одной из самых первых методик [1, 9], но, тем не менее, достаточно эффективной. Рассмотрим функцию $f : x \rightarrow x_1$ или $f : x \rightarrow \sum_i x_i$. В этом случае любой симметричный мутационный оператор имеет вероятность успеха $\frac{1}{2}$: в половине случаев решение улучшится, в половине — станет хуже. Если рассмотреть гладкую функцию с использованием формулы Тейлора, то можно заметить, что она становится все более линейной, чем меньше количество соседних точек. Поэтому для $\sigma \rightarrow 0$ вероятность успеха становится $\frac{1}{2}$. На большей части нелинейных функций вероятность успеха — монотонная функция, которая уменьшается до 0 при $\sigma \rightarrow \infty$. Именно из этого возникла идея увеличивать размер шага для большой вероятности успеха и уменьшать для малой.

Реченбер [1] вывел зависимость между двумя простыми различными функции:

$$f : x \rightarrow \begin{cases} x_1, & \text{if } |x_i| \leq 1 \text{ for } i = 2, \dots, n \\ \infty, & \text{в противном случае} \end{cases}$$

и функцию де Джонга («sphere»):

$$f : x \rightarrow \sum x_i^2.$$

Он обнаружил, что оптимальные значения для успешной изотропической мутации для $(1 + 1)$ -ES составляют $\approx 0.184 > \frac{1}{6}$ и $\approx 0.270 > \frac{1}{3}$ (для $n \rightarrow \infty$), что составляет приблизительно $1/5$.

Пример использования данной стратегии представлен на листинге 3 [10].

2. **Самоадаптация (Self-Adaptation)**. В данном способе вводятся новые параметры контроля, которые генерируются методом, аналогичным генерации новых x -векторов рекомбинацией и мутацией. Пример алгоритма, использующего данный метод представлен на листинге 4.

В строках 4–6 вычисляется набор случайных элементов. В строке 7 вектор размера шага подвергается мутации, общей для всех компонент, а также независимым покомпонентным мутациям. В строке 8 применяет-

Листинг 3 – $(1 + 1)$ -ES с правилом $1/5$

Require: $n \in \mathbb{N}_+, d \approx \sqrt{n + 1}$

- 1: **Инициализация:** $x \in \mathbb{R}^n, \sigma > 0$
- 2: **while** Не достигли условия останова **do**
- 3: $x_1 = x + \sigma \times \mathcal{N}(0, I)$
- 4: $\sigma \leftarrow \sigma \times \exp^{\frac{1}{d}}(\mathbf{1}_{f(x_1) \leq f(x)} - 1/5)$
- 5: **if** $f(x_1) \leq f(x)$ **then**
- 6: $x = x_1$
- 7: **end if**
- 8: **end while**

Листинг 4 – $(\mu/\mu, \lambda) - \sigma SA - ES$

Require: $n \in \mathbb{N}_+, \lambda \geq 5n, \mu \approx \frac{\lambda}{4} \in \mathbb{N}, \tau \approx \frac{1}{\sqrt{n}}, \tau_i \approx \frac{1}{n^{\frac{1}{4}}}$

- 1: **Инициализация:** $x \in \mathbb{R}^n, \sigma \in \mathbb{R}_+^n$
- 2: **while** Не достигли условия останова **do**
- 3: **for** $k \in \{1, \dots, \lambda\}$ **do**
- 4: $\xi_k = \tau \mathcal{N}(0, 1)$
- 5: $\zeta_k = \tau_i \mathcal{N}(0, I)$
- 6: $z_k = \mathcal{N}(0, I)$
- 7: $\sigma_k = \sigma \circ \exp(\zeta_k) \times \exp(\xi_k)$
- 8: $x_k = x + \sigma_k \circ z_k$
- 9: **end for**
- 10: $P = sel_{\mu_best}(\{(x_k, \sigma_k, f(x_k)) | 1 \leq k \leq \lambda\})$
- 11: $\sigma = \frac{1}{\mu} \sum_{\sigma_k \in P} \sigma_k$
- 12: $x = \frac{1}{\mu} \sum_{x_k \in P} x_k$
- 13: **end while**

ся усредненная рекомбинация для вычисления x и σ для следующего поколения. Однако применение и мутации, и рекомбинации к σ создает слабую зависимость, из-за которой σ имеет тенденцию к возрастанию при нейтральном отборе [11].

Для того, чтобы достичь стабильности поведения σ , число родителей μ должно быть достаточно большим.

3. **Неслучайная самоадаптация (Derandomized Self-Adaptation)**. Данный метод [12] призван справляться с проблемой несоответствия параметров при отборе. Так очень хорошая особь, может быть сгенерирована с плохими внутренними динамическими параметрами и наоборот. Данная проблема случается достаточно часто и имеет две основные причины:

- Маленькая или большая компонента в $|\sigma_k \circ z_k|$ (строка 8 листинга 4) не обязательно влияет на величину компоненты σ_k .
- При отборе маленькой или большой компоненты $|\sigma_k \circ z_k|$ ее знак может иметь большее значение, чем размер компоненты, или даже влияние всех остальных отобранных компонент.

Из-за проблемы несоответствия параметров, очень часто попадают плохие значения σ , а также наблюдаются стохастические флуктуации. Такие флуктуации способствуют отбрасыванию больших и распространению небольших значений. Размер этих флуктуаций контролируется количеством родительских особей (строка 11 листинга 4), по причине того, что рекомбинация по медиане эффективно уменьшает размер колебаний изменений σ , тем самым направляя $\log \sigma$ к большим значениям.

Пример алгоритма $(1, \lambda)$ - σ SA-ES представлен на листинге 5.

Вместо ввода переменной разброса для σ в строке 7 используются вариации, использовавшиеся для z_k .

При использовании же данного метода в случае $(\mu/\mu, \lambda)$ - σ -self-adaptation-ES, получаемые шаги оказываются слишком маленькими, что сильно замедляет скорость алгоритма. Для решения этой проблемы используется нелокальная информация для контроля шага.

4. **Неслучайный контроль размера шага по нелокальной информации (Non-Local Derandomized Step-Size Control (CSA))**. При использовании самоадаптации размеры шага привязаны к особи и выбираются на основе значений функции приспособленности каждой из особей. Однако при

Листинг 5 – Derandomized $(1, \lambda)$ - σ SA-ES

Require: $n \in \mathbb{N}_+$, $\lambda \approx 10$, $\tau \approx \frac{1}{3}$, $d \approx \sqrt{n}$, $d_i \approx n$

```

1: Инициализация:  $x \in \mathbb{R}^n$ ,  $\sigma \in \mathbb{R}_+^n$ 
2: while Не достигли условия останова do
3:   for  $k \in \{1, \dots, \lambda\}$  do
4:      $\xi_k = \tau \mathcal{N}(0, 1)$ 
5:      $z_k = \mathcal{N}(0, I)$ 
6:      $x_k = x + \exp(\xi_k) \times \sigma \circ z_k$ 
7:      $\sigma_k = \sigma \circ \exp^{\frac{1}{d_i}} \left( \frac{|z_k|}{\mathbb{E}|\mathcal{N}(0,1)|} - 1 \right)$ 
8:   end for
9:    $(x_1, \sigma_1, f(x_1)) \leftarrow \text{select\_sigma\_best}((x_k, \sigma_k, f(x_k)) | 1 \leq k \leq \lambda)$ 
10:   $\sigma = \sigma_1$ 
11:   $x = x_1$ 
12: end while

```

таком отборе, размеры шага способствуют прогрессу конкретной особи, а не всей популяции или родительскому центруиду x . С другой точки зрения, размер шага μ -той лучшей особи обычно даже меньше, чем размер шага лучшей особи. Таким образом, алгоритм постепенно использует все меньший и меньший шаг. Для того, чтобы этого избежать, необходимо использовать нелокальную информацию об эволюции популяции. Вместо использования одного мутационного шага (локального) z , нужно использовать экспоненциально убывающую запись s_σ об уже прошедших шагах. Эта запись называется поисковый или эволюционный путь. Поисковый путь несет в себе информацию о взаимосвязи между отдельными шагами, и эта информация может значительно улучшить процесс адаптации и поиска.

Пример алгоритма $(\mu/\mu, \lambda)$ -ES с сохранением пути поиска представлен на листинге 6. В этом алгоритме используется контроль суммарной длины пути, а также нелокальная индивидуальная адаптация длины шага особи [13, 14]. В сравнении с алгоритмом, приведенном на листинге 5, изменение глобального размера шага ξ_k исчезло. Обновление σ происходит после окончания цикла *for* по генерации x_k . Изменение глобального размера шага теперь происходит в зависимости от длины пути $\|s_\sigma\|$, а не случайной вариации. В то время как изменение индивидуального размера шага $|z_k|$ заменено на $|s_\sigma|$.

Листинг 6 – $(\mu/\mu, \lambda)$ -ES с сохранением пути поиска

Require: $n \in \mathbb{N}_+$, $\lambda \in \mathbb{N}$, $\mu \approx \frac{\lambda}{4} \in \mathbb{N}$, $c_\sigma \approx \sqrt{\frac{\mu}{(n+\mu)}}$, $d \approx 1 + \sqrt{\frac{\mu}{n}}$, $d_i \approx 3n$

```
1: Инициализация:  $x \in \mathbb{R}^n$ ,  $\sigma \in \mathbb{R}_+^n$ ,  $s_\sigma = 0$ 
2: while Не достигли условия останова do
3:   for  $k \in \{1, \dots, \lambda\}$  do
4:      $z_k = \mathcal{N}(0, I)$ 
5:      $x_k = x + \sigma \circ z_k$ 
6:   end for
7:    $P \leftarrow sel\_mu\_best(\{(x_k, z_k, f(x_k)) \mid 1 \leq k \leq \lambda\})$ 
8:    $s_\sigma \leftarrow (1 - c_\sigma)s_\sigma + \sqrt{c_\sigma(2 - c_\sigma)} \frac{\sqrt{\mu}}{\mu} \sum_{z_k \in P} z_k$ 
9:    $\sigma \leftarrow \sigma \circ \exp^{\frac{1}{d_i}} \left( \frac{\|s_\sigma\|}{\mathbb{E}\|\mathcal{N}(0, I)\|} - 1 \right) \times \exp^{\frac{c_\sigma}{d}} \left( \frac{\|s_\sigma\|}{\mathbb{E}\|\mathcal{N}(0, I)\|} - 1 \right)$ 
10:   $x = \frac{1}{\mu} \sum_{x_k \in P} x_k$ 
11: end while
```

1.1.3.6. Зависимости между переменными

Рассмотренные до сих пор примеры эволюционных стратегий имеют независимые распределения по каждой из компонент в заданной координатной системе. Линии одинаковой плотности являются либо сферическими, либо эллипсоидальными, параллельными осям координат. Это условие сильно ограничивает возможность решения задач — эффективно будут решаться только те, решения которых пролегают вдоль осей координат. Для вычисления других случаев, используются или ограниченная ковариационная матрица, или полная ковариационная матрица. В первом случае, адаптация происходит для недиагональной ковариационной матрицы специального типа [14–17] (Данный тип алгоритмов не будет подробнее рассмотрен в рамках этой работы). Второй случай — адаптация полной ковариационной матрицы.

1.2. Описание алгоритма для модификации

1.2.1. LM-CMA-ES

LM-CMA-ES является наследником $(\mu/\mu_w, \lambda)$ -Cholesky-CMA-ES и использует некоторые его особенности.

1.2.1.1. Разложение Холецкого в CMA

Алгоритм LM-CMA-ES унаследовал одно из самых важных нововведений в $(\mu/\mu_w, \lambda)$ -Cholesky-CMA-ES: использование разложения Холецкого и хранение векторов вместо цельной ковариационной матрицы.

Разложение Холецкого — представление матрицы A , симметрично-положительной, в виде $A = LL^T$. L — нижняя треугольная матрица, имеющая строго положительные элементы на всей диагонали. Разложение всегда существует для любой положительно-определенной матрицы. Тогда, элементы можно вычислить как:

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2},$$

$$L_{ij} = \frac{1}{L_{jj}}(A_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk}), \text{ при } j < i$$

Функция реконструкции $x = A^t z$ из m эволюционных путей и их обратных значений v , используемая в LM-CMA-ES, представлена на листинге 7.

На каждой итерации реконструкции $x = A^t z$ (строки 2–3) x обновляется суммой себя, веса a , и путем эволюции p_c^t , веса b^t , масштабированного относительно скалярного произведения v^t и x . Нотация индекса $j(t)$ вводится для удобства ссылок на матрицы P и V , которые содержат в себе соответственно вектора p_c^t и v^t .

Аналогичный подход выполняется для построения $x = A^{t-1} z$. Пример представлен на листинге 8. Алгоритм использует $c = \frac{1}{1-c_1}$ и $d^t = \frac{1}{\sqrt{q-c_1\|v^t\|^2}} \times (1 - \frac{1}{\sqrt{1+\frac{c_1}{1-c_1}\|v^t\|^2}})$. Вычислительная сложность обеих процедур — $O(mn)$

1.2.1.2. Отбор и хранение векторов направления

В изначальной версии CMA [18–25] ковариационная матрица хранит информацию о n направлениях. Для больших λ и n , где $\lambda \ll n$, вектора эволюционного пути p_c^t за последние m итераций будут схожими или идентичными, потому они будут хранить небольшое количество локальной информации.

В алгоритме LM-CMA-ES используется другой метод: последние хранимые m векторов специально удерживаются на одинаковом расстоянии друг от друга между итерациями, но на наибольшем расстоянии N_{steps} в общем

Листинг 7 – Az() — Холецкий

Require: $z \in \mathbb{R}^n, m \in \mathbb{Z}_+, j \in \mathbb{Z}_+^m, P \in \mathbb{R}^{m \times n}, V \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, a \in [0, 1]$

- 1: **Инициализация:** $x \leftarrow z$
- 2: **for** $t \in \{1, \dots, \min(m, |j|)\}$ **do**
- 3: $k \leftarrow b_{j_t} V(j_t) \cdot x$
- 4: $x \leftarrow ax + kP_{j_t}$
- 5: **end for**
- 6: **return** x

Листинг 8 – Ainvz() – Холецкий inverse

Require: $z \in \mathbb{R}^n, m \in \mathbb{Z}_+, j \in \mathbb{Z}^m, V \in \mathbb{R}^{m \times n}, d \in \mathbb{R}^m, c \in [0, 1]$

```

1: Инициализация:  $x \leftarrow z$ 
2: for  $t \in \{1, \dots, \min(m, |j|)\}$  do
3:    $k \leftarrow d_{j_t} V(j_t) \cdot x$ 
4:    $x \leftarrow cx - kV_{j_t}$ 
5: end for
6: return  $x$ 

```

понимании, при условии, что m -й вектор — это вектор из последнего шага. Процедура отбора представлена на листинге 9.

Листинг 9 – UpdateSet()

Require: $m \in \mathbb{R}^+, j \in \mathbb{Z}_+^m, l \in \mathbb{Z}_+^m, t \in \mathbb{Z}_+, N_{steps} \in \mathbb{Z}_+$

```

1: if  $t < m$  then
2:    $j_t \leftarrow t$ 
3: else
4:    $i_{\min} \leftarrow l + \arg \min_i (l_{j_{i+1}} - l_{j_i}), |1 \leq i \leq (m - 1)|$ 
5:   if  $l_{j_{i_{\min}}} - l_{j_{i_{\min}-1}} \geq N_{steps}$  then
6:      $i_{\min} \leftarrow 1$ 
7:   end if
8:   if  $i_{\min} \neq m$  then
9:      $j_{temp} \leftarrow j_{i_{\min}}$ 
10:    for  $i = \{i_{\min}, \dots, m - 1\}$  do
11:       $j_i \leftarrow j_{i+1}$ 
12:    end for
13:     $j_m \leftarrow j_{min(t+1, m)}$ 
14:  end if
15:  end if
16:   $j_{cur} \leftarrow j_{min(t+1, m)}$ 
17:   $j_t \leftarrow t$  return  $j_{curr}, j, l$ 

```

Процедура возвращает массив указателей j , таких, что j_1 указывает на ряд в матрицах P и V , хранящих информацию о самых старых векторах p_c и v , которые будут использованы на этапе реконструкции. Чем больше индекс i у j_i , тем новее указываемый вектор. Индекс j_{cur} указывает на самый старый вектор, который будет заменен новым, в той же итерации, в которой вызвана процедура.

Правило отбора вектора для удаления следующее:

- найдем пару последовательно сохраненных векторов с наименьшей дистанцией между ними (в понимании итераций, сохраненных в l) (строка 4);
- если расстояние между ними меньше, чем N_{steps} , тогда самый новый вектор будет удален (строка 5), путем $j_{cur} \leftarrow i_{min}$, иначе самый старый из m векторов будет удален (строка 8).

1.2.1.3. Правило отбора

LM-CMA-ES использует правило успеха, предложенное в [26]. Оно используется для неэлитических эволюционных стратегий. В этом правиле медиана значения функции приспособленности популяции сравнивается с медианой значения ФП предыдущей итерации. Их разность должна достигать $1/2$. Для реализации необходимо посчитать количество K_{succ} особей в текущей популяции, лучших, чем j -я лучшая особь их предыдущей популяции, где j зависит от n и λ , но может быть просто 0.3λ [26]. Тогда нормированная мера

$$z \leftarrow \frac{2}{\lambda} \left(K_{succ} - \frac{\lambda + 1}{2} \right)$$

может быть посчитана. Если особь прошла отбор, то $z \geq 0$.

Шаг алгоритма обновляется следующим образом:

$$\sigma \leftarrow \sigma \exp\left(\frac{s}{d_\sigma}\right)$$

где $s \leftarrow (1 - c_{sigma})s + c_\sigma z$ и $d_s = 2(n - 1)/n$

Однако этот метод имеет недостаток, который можно продемонстрировать на примере. Предположим, что имеется следующий набор значений функций приспособленности из предыдущей популяции: $f_{t-1} = [2.1, 3.1, 4.1, 5.1, 6.1, 7.1, 8.1]$, а значения для текущей популяции: $f_t = [1, 2, 3, 4, 5, 6, 7]$. Тогда, используя правило отбора по медиане, если j выбран как 3, то количество успешных особей: 4 (со значением функции приспособленности больше чем $f_{t-1}(3) = 4.1$, так как $f_t(1) = 1$, $f_t(2) = 2$, $f_t(3) = 3$, $f_t(4) = 4$). Тогда значение $K_{succ} = 4$ будет использоваться для адаптации шага алгоритма. Однако такой расчет не принимает во внимание то, что значения $f_{t-1}(i)$ для $1 \leq i < j$, даже если все эти значения f_{t-1} лучше,

чем лучшее значение $f_t(1)$. Несмотря на этот недостаток, это правило отбора хорошо показывает, насколько новое поколение лучше предыдущего.

Используется это правило следующим образом:

1. значения функций приспособленности из текущей популяции и предыдущей смешиваются в одно целое — $f_{\min} \leftarrow f_{t-1} \cup f_t$;
2. в данном объединении проставляются ранги;
3. создается два набора рангов: r_t и r_{t-1} , содержащие для, соответственно, текущей популяции и предыдущей, ранги особей в объединении;
4. вычисляется нормализация для медианного правила отбора:

$$z_{PSR} \leftarrow \frac{\sum_{i=1}^{\lambda} r^t(i) - r^{t-1}(i)}{\lambda^2} - z^*,$$

где z^* — задаваемый необходимый порог для отбора.

1.2.1.4. LM-CMA-ES алгоритм

На листинге 10 представлен $(\mu/\mu_w, \lambda)$ -LM-CMA-ES.

В строках 8 и 12 используется восстановление векторов и разложение векторов Холецкого. В строках 18–21 происходит адаптация шага алгоритма.

Листинг 10 – $(\mu/\mu_w, \lambda)$ -LM-CMA-ES

Require:

- 1: $n \in \mathbb{N}_+, \lambda = 4 + \lfloor 3 \ln n \rfloor, \mu = \lfloor \lambda/2 \rfloor, w_i = \frac{\ln(\mu+1) - \ln(i)}{\mu \ln(\mu+1) - \sum_{j=1}^{\mu} \ln(j)}, i = 1 \dots \mu,$
- 2: $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}, c_\sigma = 0.3, d_\sigma = 1, m = 4 + \lfloor 3 \ln n \rfloor, N_{steps} = m, c_c = \frac{1}{\mu},$
- 3: $c_1 = \frac{1}{10 \ln(n+1)}$
- 4: **Инициализация:** $m^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, p_c^{t=0} = 0, s \leftarrow 0, t \leftarrow 0$
- 5: **while** не выполнены условия останова **do**
- 6: **for** $k = \{1, \dots, \lambda\}$ **do**
- 7: $z_k = \mathbb{N}(0, 1)$
- 8: $x_k = m^t + \sigma^t A z(z_k)$
- 9: $f_k = f(x_k)$
- 10: **end for**
- 11: $m_{t+1} \leftarrow \sum_{i=1}^{\mu} w_i x_{i\lambda}$
- 12: $p_c^{t+1} \leftarrow (1 - c_c) p_c^t + \sqrt{c_c(2 - c_c)} \sqrt{\mu_w} (m^{t+1} - m^t) / \sigma$
- 13: $v \leftarrow Ainvz(p_c^{t+1})$
- 14: $j_{cur} \leftarrow UpdateSet()$
- 15: $V_{j_{curr}} \leftarrow v; P_{j_{curr}} \leftarrow p_c^{t+1}$
- 16: $b_{j_{curr}} \leftarrow \frac{\sqrt{1-c_1}}{\|v^t\|^2} (\sqrt{1 + \frac{c_1}{1-c_1} \|v^t\|^2} - 1)$
- 17: $d_{j_{curr}} \leftarrow \frac{1}{\sqrt{1-c_1} \|v^t\|^2} (1 - \frac{1}{\sqrt{1 + \frac{c_1}{1-c_1} \|v^t\|^2} - 1})$
- 18: $r^t, r^{t-1} \leftarrow \text{Ranks of } f^t \text{ and } f^{t-1} \text{ in } f^t \cup f^{t-1}$
- 19: $z_{PSR} \leftarrow \frac{\sum_{i=1}^{\lambda} r^t(i) - r^{t-1}(i)}{\lambda^2} - z^*$
- 20: $s \leftarrow (1 - c_\sigma) s + c_\sigma z_{PSR}$
- 21: $\sigma^{t+1} \leftarrow \sigma^t \exp(s/d_\sigma)$
- 22: $t \leftarrow t + 1$
- 23: **end while**

1.3. Функции тестирования эволюционных алгоритмов

В процессе изучения эволюционных алгоритмов и вопросов глобальной оптимизации, был сформирован набор специальных функции, из-за некоторых особенностей которых, они оказались трудно для оптимизации различным наборам алгоритмов. [27–33]

Ниже представлен набор функций, использующийся в данной работе.

1.3.1. Функция де Джонга («sphere»)

Наиболее простая функция для тестирования — функция де Джонга. Также известная как сферическая модель. Является непрерывной, выпуклой и унимодальной. Изображение функции представлено на рисунке 3.

Определение: $f(x) = \sum_{i=1}^n x_i^2$, где $i \in [1; n]$, $-5.12 \leq x_i \leq 5.12$.

Глобальный минимум: $f(x) = 0, x_i = 0, i \in [1; n]$.

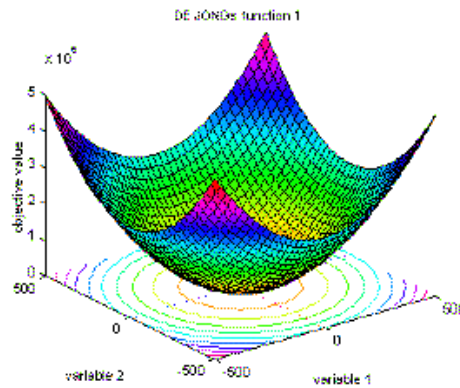


Рисунок 3 – Пример графика сферической функции

1.3.2. Функция Розенброка

Функция известная так же как «Банановая функция». Глобальный минимум находится внутри длинной, узкой, параболической формы плоскости. Изображение функции представлено на рисунке 4.

Определение: $f(x) = \sum_{i=1}^{n-1} 100 * (x_{i+1} - x_i^2)^2 + (1 - x_i)^2$, где $i \in [1; n - 1]; -2.048 \leq x_i \leq 2.048$.

Глобальный минимум: $f(x) = 0; x_i = 1, i \in [1; n]$.

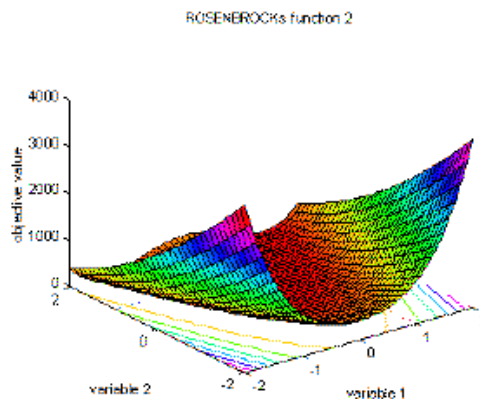


Рисунок 4 – Пример графика функции Розенброка

1.3.3. Функция Растригина

Функция, построенная на основе функции де Джонга («sphere»), с применением модуля по функции *cos* для получения множества глобальных минимумов. Минимумы равномерно распределены. Изображение функции представлено на рисунке 5.

Определение: $f(x) = 10n + \sum_{i=1}^n x_i^2 - 10 \cos(2\pi x_i)$, где $i \in [1; n]; -5.12 \leq x_i \leq 5.12..$

Глобальный минимум: $f(x) = 0; x_i = 0, i \in [1; n]..$

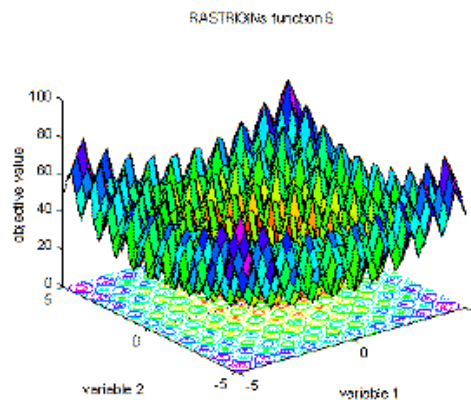


Рисунок 5 – Пример графика функции Растригина

1.3.4. Функция гиперэллипсоида

Расширение стандартной эллипсоидной функции, которое по отношению к координатным осям генерирует гиперэллипсоиды. Является непрерывной, выпуклой и унимодальной. Изображение функции представлено на рисунке 6.

Определение : $f(x) = \sum_{i=1}^n (\sum_{j=1}^i x_j)^2$, где $j \in [1; i], i \in [1; n], -65.536 \leq x_i \leq 65.536..$

Глобальный минимум : $f(x) = 0; x_i = 0, i \in [1; n]..$

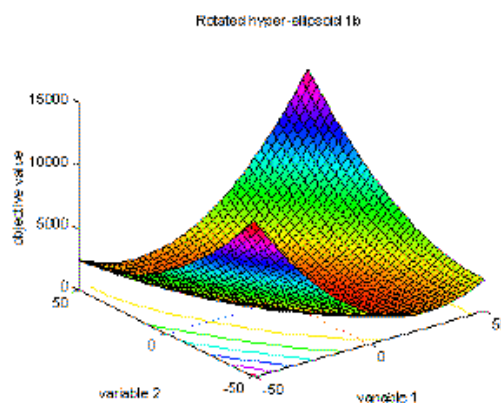


Рисунок 6 – Пример графика функции гиперэллипсоид

ГЛАВА 2. ОПИСАНИЕ ПРЕДЛАГАЕМОГО ПОДХОДА

2.1. Основные идеи модификации алгоритма LM-CMA-ES

Алгоритм семейства CMA-ES состоит, как правило, из нескольких основных частей: генерации особей на основании векторов направления, мутации особей и модификации внутренних параметров алгоритма. При подборе алгоритма для модификации учитывались несколько параметров:

- эффективность алгоритма в целом на общем наборе функций;
- эффективность алгоритма с точки зрения использования вычислительной мощности;
- эффективность алгоритма по расходу памяти.

Алгоритм LM-CMA-ES является эффективным по вычислениям и по затратам памяти, использующим последние наработки в области эволюционных стратегий. Обратим внимание на строки 6–10. Это первая часть алгоритма, где происходит генерация λ особей при помощи векторов из разложения Холецкого для ковариационной матрицы. Несложно заметить, что сложность этой части алгоритма зависит в небольшой степени от размерности, и в большой — от сложности вычисляемой функции. Часть 11–21 зависит только от размера пространства. Так как при использовании сложных функций для вычислений процессорное время первой части будет крайне быстро расти, была необходима параллелизация этой части и выделение части модификации динамических параметров как критической.

Изначальный алгоритм можно разделить на четыре условные части:

1. **Генерация поколений** представлена в строках 6–10. При использовании данного оптимального метода модификации возникает проблема зависимостей внутри поколений: динамические параметры алгоритма изменяются по окончании вычисления именно поколения. Для преодоления этой трудности используется *steady-state* подход: убирается понятие поколений и генерация особей выделяется в отдельный независимый поток. Тогда при достаточной сложности вычисляемой функции потоки, генерирующие особь и считающие значение ее функции приспособленности будут приходить в критическую секцию в различное время, чем приблизят алгоритм к асинхронному. В свою очередь, для генерации внутри потока динамические параметры будут браться в виде «слепка» с состояния алгоритма на момент начала генерации.

Тогда для каждой пары потоков возможны два состояния на момент генерации:

- (a) Два потока приходят за данными одновременно, т.е. между считыванием данных в потоке 1 и в потоке 2 обновления переменных не производилось. Такое поведение является схожим с поведением начального алгоритма. Если предположить, что все μ потоков начали работу одновременно, то они все составляют псевдопоколение размера μ .
- (b) Поток считывает динамические данные, после этого данные обновляются одним из предыдущих потоков, второй поток получает на вход обновленные данные. Такое поведение будет встречаться тем чаще, чем асинхронно работает алгоритм.

Следует отметить, что поведение будет изменяться в течение работы алгоритма. Сразу после запуска будет чаще встречаться случай 1, что позволит получить максимальное разнообразие особей на старте и затем, при случае 2, более «детально» разрабатывать каждый из них. Минус данного подхода — сильная зависимость от метода обновления σ , так как только при обновлении шага будет вноситься сильное разнообразие в набор особей, в отличие от LM-CMA-ES, где разнообразие вносится также регенерацией $\lambda - \mu$ особей на каждом шаге.

2. **Вычисление среднего значения среди поколения** представлено в строке 11. На данном шаге алгоритма происходит два действия: вычисление веса и вычисление среднего значения в поколении. Вес — величина, зависящая от ранга значения функции приспособленности особи в поколении, т.е. на каждом шаге (поколении) оригинального алгоритма LM-CMA-ES происходит сортировка весов, и лучшие μ особей используются в дальнейшем для вычисления динамических величин. Для данного процесса необходимо взаимодействовать с другими особями, поэтому именно с этого шага начинается критическая секция. Так как понятия поколения не существует, нужно было изменить способ подсчета среднего значения в поколении. Выбрана была следующая стратегия: из всех прошедших особей хранить μ лучших. Тогда подсчет веса, ранга и среднего значения в поколении становятся инкрементальными, и появляется возможность замены обычной сортировки ($O(N \log(N))$) и обновления

среднего значения ($O(\log(\mu))$) на динамическую структуру, способную поддержать вставку и удаление элемента за небольшое время, например $O(\log(mu))$.

3. **Обновление векторов направлений и эволюционного пути** представлено в строках 12–17. Данный шаг оставлен без изменений. Обновление находится в критической секции и потому метод его работы аналогичен работе в LM-CMA-ES.
4. **Обновление значения шага алгоритма** представлено в строках 18–21. При использовании выбранного метода модификации данный шаг должен был быть изменен. Действительно, при отсутствии понятия поколения, понятие расстояния между поколениями тоже теряет смысл. Расстояние между предыдущим набором лучших особей и текущим, при добавлении в него новой, уже учитывается при подсчете среднего значения.

Для разработанного алгоритма выбран метод, называемый 1/5 success rule. Он заключается в увеличении шага, если вероятность удачного поколения превышает 1/5, в ином случае — уменьшении. Данный метод выбран по нескольким причинам, которые приводятся далее.

- (a) Является базовым для использования в CMA-ES, но все же достаточно эффективным.
- (b) Не зависит ни от каких мешающих параллелизации элементов.
- (c) Очень быстр по времени вычисления, что уменьшает необходимый набор работ в критической секции и, как следствие, ускоряет алгоритм.
- (d) Для сравнения модификации и начального алгоритма в честных условиях, использование слишком «сильных» методов изменения шага было бы недопустимым. Данный метод дает средние результаты по скорости работы.

Главным минусом использования этого метода, является его склонность к схождению к локальному минимуму, что может негативно сказаться на комплексных функциях с большим количеством локальных минимумов.

2.2. Псевдокод предлагаемого алгоритма

На листинге 11 представлен псевдокод разработанного метода.

Схема работы алгоритма следующая:

Листинг 11 – Modified $(\mu/\mu_w, \lambda)$ -LM-CMA-ES

Require: $numberOfThreads \in \mathbb{N}$

- 1: $n \in \mathbb{N}_+, \lambda = 4 + \lfloor 3 \ln n \rfloor, \mu = \lfloor \lambda/2 \rfloor, w_i = \frac{\ln(\mu+1) - \ln(i)}{\mu \ln(\mu+1) - \sum_{j=1}^{\mu} \ln(j)}, i = 1 \dots \mu,$
- 2: $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}, c_\sigma = 0.3, d_\sigma = 1, m = 4 + \lfloor 3 \ln n \rfloor, N_{steps} = m, c_c = \frac{1}{\mu},$
- 3: $c_1 = \frac{1}{10 \ln(n+1)}, pTargetSucc = \frac{2}{11}, d = 1 + \frac{n}{2}, c_p = \frac{1}{12}, pSucc = 0$
- 4: **Инициализация:** $m^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, p_c^{t=0} = 0, s \leftarrow 0, t \leftarrow 0, numberOfThreads$
- 5: **for all** $k = \{1, \dots, numberOfThreads\}$ **do in parallel**
- 6: **while** не выполнены условия останова **do**
- 7: $z = \mathbb{N}(0, 1)$
- 8: $x = m^t + \sigma^t Az(z)$
- 9: $f = f(x)$
- 10: UPDATE DYNAMIC PARAMETERS(x, f)
- 11: **end while**
- 12: **end for**
- 13:
- 14: **procedure** UPDATE DYNAMIC PARAMETERS($x, fitnessValue$)
- 15: INSERT TO MUBEST($x, fitnessValue$)
- 16:
- 17: $p_c^{t+1} \leftarrow (1 - c_c)p_c^t + \sqrt{c_c(2 - c_c)}\sqrt{\mu_w}(m^{t+1} - m^t)/\sigma$
- 18: $v \leftarrow Ainvz(p_c^{t+1})$
- 19: $j_{cur} \leftarrow UpdateSet()$
- 20: $V_{j_{curr}} \leftarrow v; P_{j_{curr}} \leftarrow p_c^{t+1}$
- 21: $b_{j_{curr}} \leftarrow \frac{\sqrt{1-c_1}}{\|v^t\|^2} (\sqrt{1 + \frac{c_1}{1-c_1}\|v^t\|^2} - 1)$
- 22: $d_{j_{curr}} \leftarrow \frac{1}{\sqrt{1-c_1}\|v^t\|^2} (1 - \frac{1}{\sqrt{1 + \frac{c_1}{1-c_1}\|v^t\|^2} - 1})$
- 23: $pSucc^{t+1} \leftarrow pSucc(1 - c_p) + (wasSuccessfulFlag?c_p : 0)$
- 24: $\sigma^{t+1} \leftarrow \sigma^t \exp(\frac{1}{d}(pSucc - (\frac{pTargetSucc}{1-pTargetSucc})(1 - pSucc)))$
- 25: $t \leftarrow t + 1$
- 26: **end procedure**

1. Основная часть программы (строки 0–12) занимается конфигурацией параметров (статистических, таких как, например, различные коэффициенты), запуском рабочих потоков, их мониторингом. После достижения условия останова, производится завершение всей программы. Каждый рабочий поток (строка 5) производит генерацию особей с использованием текущих векторов направления, нормального распределения, σ и среднего значения вектора на текущий момент. Важно отметить, что условие

останова может выполняться только в критической секции (строка 10), после чего поток успешно останавливается.

2. Критическая секция (строки 14–26). На вход подается значение x сгенерированного вектора и `fitnessValue` — вычисленное значение функции приспособленности для этой особи. На входе в секцию установлен `lock`, что исключает одновременный вход в эту секцию нескольких потоков. Важно также отметить вызов процедуры `InsertToMuBest` в строке 14. Для инкрементального подсчета веса и среднего значения по просчитанным поколениям было выбрано AVL-дерево, предоставляющее возможность вставки, удаления и пересчета элементов за максимум сложности $O(n \log(\mu))$.

2.3. Реализация предлагаемого алгоритма

Предлагаемый алгоритм реализован на языке программирования *Java*, что облегчило работу с потоками и измерение времени, проведенного каждым потоком в различных участках кода. С целью сокращения объема самостоятельно написанного кода были использованы следующие библиотеки:

- `org.apache.logging.log4j/log4j-core`, версия 2.2 — для асинхронного логирования данных;
- `com.google.guava/guava`, версия 18 — для удобства использования потоков;
- `com.lmax/disruptor`, версия 3.3.2 — для поддержки `log4j` асинхронного логирования.

В разделах 2.3.1–2.3.6 приведены интерфейсы и описание функциональности *Java*-классов, составляющих реализацию предлагаемого метода. С целью сравнения предлагаемого метода с алгоритмом LM-CMA-ES, последний также был реализован. При этом максимальный объем кода был разделен между сравниваемыми алгоритмами, что позволило уменьшить общий объем работы, снизить число ошибок и обеспечить равенство времени работы общих фрагментов кода.

Исходный код доступен в сети Интернет на сайте bitbucket.org¹.

¹<https://github.com/killerforfun/Masters>

2.3.1. Пакет `AlgorithmParameters`

2.3.1.1. Класс `AlgorithmStaticParameters`

Данный класс содержит в себе статически задаваемые параметры алгоритма, не изменяющиеся в процессе его работы. Значения большинства данных параметров наследовано из [34]. Параметры:

- Инициализируемые пользователем
 - `dimension` — размерность пространства для вычисления функции;
 - `target_f` — значение функции, цель. Задается как $10e - 10$;
 - `target_sigma` — максимальное значение параметра размера шага σ ;
- Неизменяемые без recompilation
 - `target_evals` — максимальное число вычислений функции. Задается как постоянное значение $1e6$;
 - `lambda` — число генерируемых особей. Задается как $(4 + \text{Math.floor}(3 * \text{Math.log}(\text{dimension})))$;
 - `mu` — число отбираемых родителей. Задается как $\text{lambda}/2$;
 - `ccov` — параметр обучаемости для обновление ковариационной матрицы. Задается как $1/(10 * \text{log}(\text{dimension} + 1.0))$;
 - `xmin`, `xmax` — минимальное и максимальное значение области поиска. Задаются как постоянные значения -5 и 5 соответственно;
 - `cc` — временная константа для обновления ковариационной матрицы. Задается как $1.0/\text{lambda}$;
 - `sigma` — начальное значение `sigma`. Задается как $0.5 * (\text{xmax} - \text{xmin})$;
 - `c_s` — параметр обучаемости для `sigma`. Имеет постоянное значение 0.3 ;
 - `nvectors` — размер вектора, регулирующего перемещения. Равен `lambda`, для изменений по всем компонентам;
 - `maxsteps` — максимальное количество шагов – `lambda`;
 - `weights` — постоянный взвешенный набор весов. Задается как $\text{Math.log}(\text{mu} + 0.5) - \text{Math.log}(1 + i)$;
 - `mu_eff` — Эффективная по вариации сумма. Задается как $1.0/\text{lambda}$;
 - `K` — вспомогательная форма `ccov`, $1/\text{Math.sqrt}(1 - \text{ccov})$;

– M – вспомогательная форма $ccov$, $Math.sqrt(1 - ccov)$.

Процесс инициализации происходит при запуске алгоритма. Для модифицированного алгоритма – в классе `CMAEvolutionStrategy`, для LM-CMA – в классе `Starter`.

2.3.1.2. Класс `AlgorithmDynamicParameters`

Основной класс для модифицированной версии алгоритма, рассматриваемого в данной работе. В нем происходит инициализация и обновление динамических параметров алгоритма.

Основным методом является `phase2`, на вход которому передаются параметры `arx`, `fitnessValue` и `currentTimeCome`, где `arx` – сгенерированная особь, `fitnessValue` – значение функции приспособленности для этой особи, `currentTimeCome` – время вызова данного метода из процедуры генерации особи в отдельном потоке. Основные этапы работы метода представлены далее.

- При входе в метод, осуществляется блокировка. Выполняемые дальше обновления являются критическими и, потому к данной секции предоставляется уникальный, одиночный доступ;
- при условии, что новая особь входит в μ лучших, происходит обновление значений `xmean` и рангов для всех лучших особей;
- происходит обновление `pc`, векторов движения, а также всех исторических данных, связанных с ними [34];
- обновляется `sigma` с использованием правила $\frac{1}{2}$;
- происходит разблокирование критической секции.

2.3.2. Пакет `IndividualCalculation`

2.3.2.1. Пакет `ObjectiveFunction`

Класс `IndividualCalculationFactory`. Фабрика для генерации потоков. Основным методом является `getIndividualCalculation`, который, в зависимости от инициализационных параметров фабрики, возвращает класс для расчета определенной функции с заданной дополнительной сложностью. В данном классе также происходит генерация матрицы для расчета алгоритмом Беллмана-Форда.

Интерфейс `ObjectiveFunctionToCalculation`. Интерфейс функции для вычислений. Содержит два метода – `valueOf` и `valueOfLMCMA`, которые соответ-

ственно рассчитывают значения функции для переданного массива целиком или только для его части.

Класс `EllipsoidObjectiveFunction`. Реализация интерфейса `ObjectiveFunctionToCalculation` для вычисления функции гиперболлипсоида.

Класс `RastriginObjectiveFunction`. Реализация интерфейса `ObjectiveFunctionToCalculation` для вычисления функции Растригина.

Класс `RosenbrockObjectiveFunction`. Реализация интерфейса `ObjectiveFunctionToCalculation` для вычисления функции Розенброка.

Класс `SphereObjectiveFunction`. Реализация интерфейса `ObjectiveFunctionToCalculation` для вычисления функции де Джонга («sphere»).

2.3.2.2. Класс `DynamicParametersAnswer`

Используется для логирования в разработанном алгоритме. Класс-обертка над ответом, получаемым от критической секции потоком, для минимизации задержек.

2.3.2.3. Класс `IndividualCalculationController`

Класс, являющийся контроллером для вычислений в разработанном алгоритме. При инициализации создает экземпляр класса `IndividualCalculationFactory`, выполняющего вычисления, и запускает его.

2.3.2.4. Класс `IndividualCalculation`

Класс представляет собой `java Runnable`, предполагающий запуск в отдельном потоке. Используется для параллельной генерации особи и вычисления ее функции приспособленности в разработанном алгоритме.

2.3.2.5. Класс IndividualGenerationLMCMA

Класс представляет собой java Callable, предполагающий запуск в отдельном потоке. Используется для параллельной генерации особи и вычисления ее функции приспособленности в алгоритме LM-CMA.

2.3.2.6. Класс IndividualLog

Используется для передачи параметров в LM-CMA. Класс-обертка над ответом, получаемым от потока основным алгоритмом, для минимизации задержек на генерации.

2.3.3. Пакет Utility

Содержит небольшие вспомогательные классы.

2.3.3.1. Пакет AdditionalWeight

Содержит классы, относящиеся к генерации матриц и вычисления алгоритма Беллмана-Форда на них.

Класс AdjMatrix. Класс является реализацией представления матрицы смежности. Содержит методы для добавления, удаления ребер, а также методы для итерации по определенному набору ребер.

Класс DirectedEdge. Класс является реализацией представления направленного ребра. В качестве параметров имеет начало, конец, а также вес.

Класс BellmanFord. Класс является реализацией алгоритма Беллмана-Форда.

2.3.3.2. Пакет FitnessTree

Пакет, содержащий классы для работы с деревом, содержащим лучших особей.

Класс FitnessTreeElement. Класс является реализацией элемента дерева - пары особь-значение функции приспособленности.

Класс FitnessTree. Класс, реализующий AVL дерево для хранения и обновления параметров лучших особей.

2.3.3.3. Класс TestCaseException

Класс, представляющий из себя java Exception. Используется при получении ошибок при запуске алгоритмов, например, при некорректно заданных параметрах.

2.3.4. Класс CMAEvolutionStrategy

Класс разработанного метода. В нем происходит инициализация динамических параметров, создание контроллера и запуск вычислений.

2.3.5. Класс LMCMA

Основной класс LM-CMA алгоритма. Представляет из себя адаптированный исходный cpp метод². Единственным изменением в сравнении с оригиналом является наличие параллельности для генерации особей и вычисления функции.

2.3.6. Класс Starter

Основной класс для запуска алгоритмов. Производит обработку параметров алгоритма и запуск тестов, описание которых производится в .properties файле.

Выводы по главе 2

В данной главе было представлено описание первоначального алгоритма, описание идей модификаций. Обоснованы внесенные изменения, представлен псевдокод и описания модифицированного метода.

²<https://sites.google.com/site/lmcmases/lmcm.cpp?attredirects=0&d=1>

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

В данной главе приводится описание экспериментального исследования, целью которого является сравнение разработанного алгоритма с его непосредственным предшественником LM-CMA-ES. Сравнение осуществляется по двум основным показателям — качеству получаемого решения и астрономическому времени, затрачиваемому на получение решения заданного качества. Сравнение производится при различных условиях, включающих в себя оптимизируемую функцию, моделируемую вычислительную сложность оптимизируемой функции, размерность задачи и число доступных вычислительных ядер.

3.1. Методология проведения эксперимента

Основной задачей эксперимента является сравнение разработанного алгоритма с алгоритмом LM-CMA-ES по следующим показателям:

1. Качество наилучшего получаемого решения. Основной целью данного показателя является определение того, насколько хорошо тот или иной алгоритм обучается особенностям оптимизируемых функций, и до каких по качеству решений он в принципе способен добраться. Для измерения данного показателя исследуемый алгоритм запускается до тех пор, пока он не достигнет оптимума с заданной точностью (например, 10^{-10}), либо пока он не истратит определенный вычислительный ресурс (например, 10^6 вычислений функции приспособленности), либо пока не будет определено, что алгоритм не в состоянии найти лучшее решение за разумное время.
2. Астрономическое время, затрачиваемое на получение решения заданного качества. Как в теории, так и в практике решения задач оптимизации существует две меры эффективности алгоритма оптимизации: наилучшее значение приспособленности, которое алгоритм может достигнуть, используя заданный объем вычислительных ресурсов, и объем вычислительных ресурсов, требуемый для достижения заданного значения приспособленности. Первая из этих мер является более «практической», так как вычислительные ресурсы обычно ограничены, однако при ее применении становится сложным сравнивать различные алгоритмы при решении различных задач — так, разница в значениях приспособленности, равная 0.1, может быть очень большой для одних задач и пренебрежимо

малой для других. По указанной причине, в настоящей работе используется вторая из описанных мер.

Указанные показатели напрямую зависят от таких параметров задачи, как вид оптимизируемой функции и размерность задачи (число аргументов функции). В дополнение к этому, по причине работы с многопоточными вычислительными машинами, а также по причине асинхронности разработанного алгоритма, указанные показатели также зависят от числа доступных вычислительных ядер, а также от соотношения вычислительных сложностей функции приспособленности и фрагментов алгоритма, работающих в критических секциях.

Для упрощения сравнения, для каждой оптимизируемой функции рассматриваются также варианты этой функции, в которой при вычислении значения в каждой точке, помимо полезной работы, также производится дополнительное вычисление определенного объема, которое моделирует большую вычислительную сложность оптимизируемой функции. Таким образом, мы получаем более «тяжеловесную» оптимизируемую функцию с таким же поведением, как и у более «легкого» аналога, что позволит наблюдать эффекты, зависящие только от вычислительной сложности функции, но не от ее вида.

Для вычисления второго показателя необходимы пороговые значения качества решений. В свете того, что сравниваемые алгоритмы могут вести себя на некоторых функциях различным образом (например, один из алгоритмов может достигнуть оптимума с точностью 10^{-10} , а другой не может), представляется разумным рассмотреть для каждой задачи несколько различных пороговых значений и вычислить астрономическое время достижения рассматриваемыми алгоритмами каждого из этих пороговых значений.

3.2. Конфигурация эксперимента

Сравнение алгоритмов производится на конфигурациях, состоящих из следующих компонент:

1. Вид оптимизируемой функции. В качестве оптимизируемых функций используются функция де Джонга («sphere»), функция Расстригина, функция Розенброка и гиперэллипсоид. Описание указанных функций приведены в разделе 1.3.

2. Размерность задачи. В свете того, что оба рассматриваемых алгоритма, в отличие от классической CMA-ES, поддерживают достаточно большие размерности задачи, выбраны размерности 100, 300, 1 000, 3 000.
3. Вычислительная сложность функции. Для внесения дополнительной вычислительной сложности был выбран алгоритм Беллмана-Форда [35, 36]. При этом фиксируется число ребер, равное 20 000, а число вершин выбирается равным 0 (отсутствие дополнительной вычислительной сложности), 200, 400 и 600. Это обеспечивает линейное возрастание дополнительной вычислительной сложности, которая при этом сравнима с или превосходит вычислительную сложность самой оптимизируемой функции.
4. Число используемых ядер процессора. Тестирование проводилось при использовании двух, четырех, шести и восьми ядер.

Критериями останова для рассматриваемых алгоритмов были выбраны следующие:

- достижение оптимума функции с точностью до 10^{-10} (оптимумы рассматриваемых функций известны и равны нулю);
- достижение числа вычислений функции значения 10^6 ;
- для разработанного алгоритма — достижение параметром σ значения 10^{-20} .

Каждый эксперимент проводился по 50 раз при наличии дополнительной вычислительной сложности и по 100 раз в противном случае.

Для определения пороговых значений функций приспособленности были проведены пробные запуски разработанного алгоритма на каждом виде оптимизируемой функции на размерности 100, без дополнительной вычислительной сложности для различного числа ядер. Графики медианных значений функции приспособленности приведены на рисунках 7–10 для функций де Джонга, Растригина, Розенброка и гиперэллипсоида соответственно.

На рисунках 7 и 8 можно видеть, что оптимизация функций де Джонга и Растригина не представляет существенной проблемы для разработанного алгоритма — независимо от числа используемых ядер, во всех случаях значение 10^{-10} достигается. Также можно отметить практически линейный (в логарифмическом масштабе значения приспособленности) характер сходимости алгоритма на указанных функциях (в случае функции Растригина имеется также

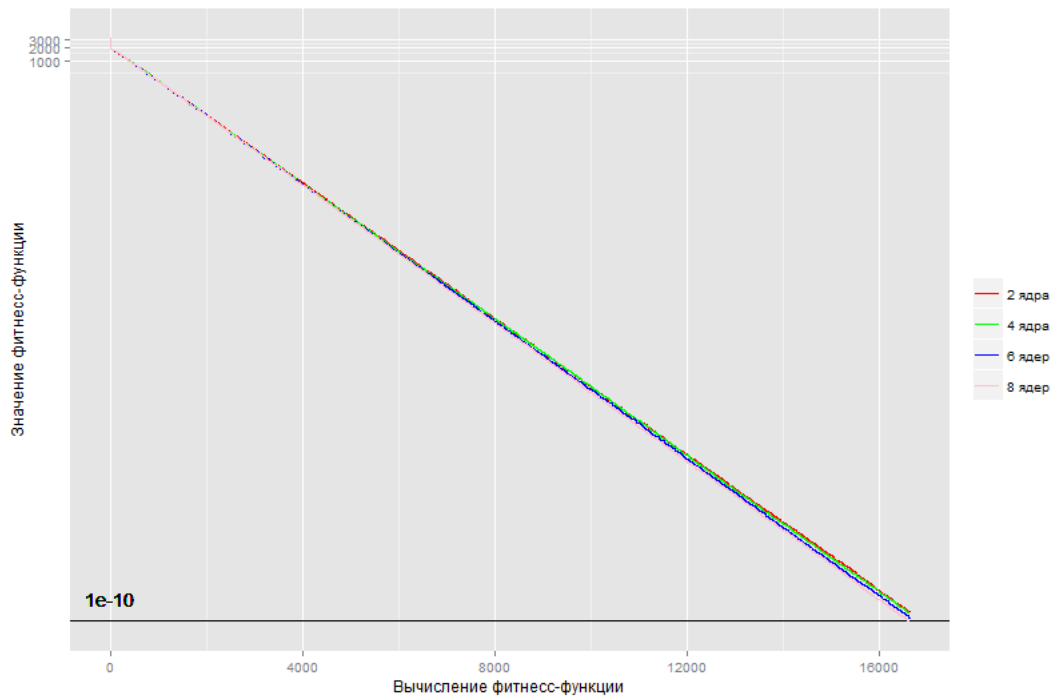


Рисунок 7 – Графики медианных значений для функции де Джонга

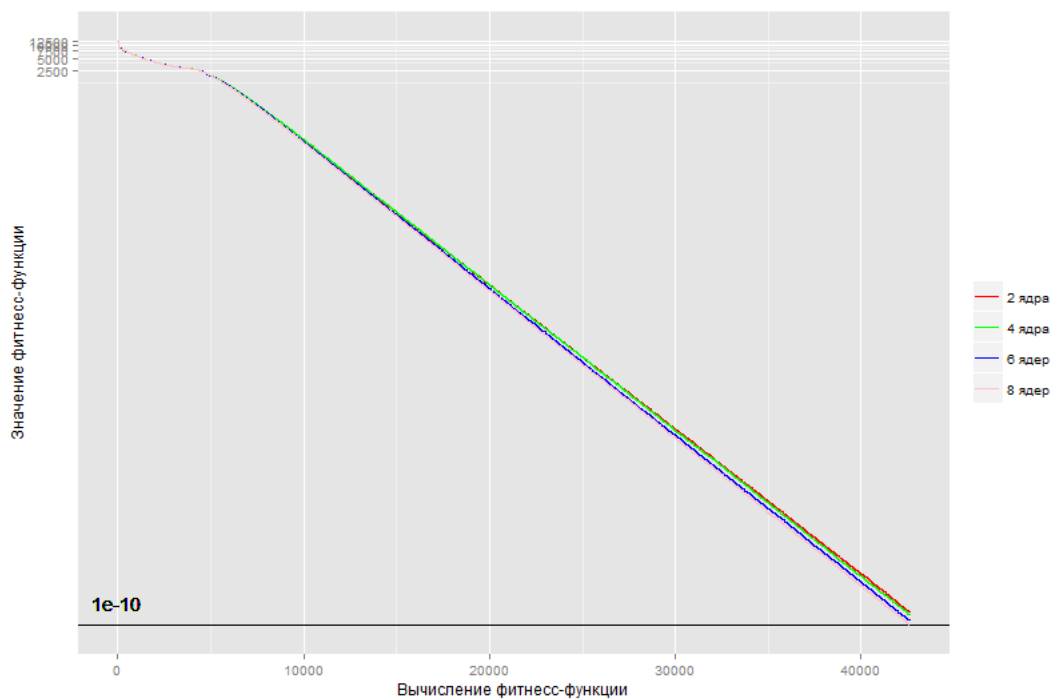


Рисунок 8 – Графики медианных значений для функции Растргина

практически линейный, но более горизонтальный участок в начале процесса оптимизации, что, по-видимому, связано с поиском окрестности глобального оптимума). Следовательно, для данных функций можно предложить равноот-

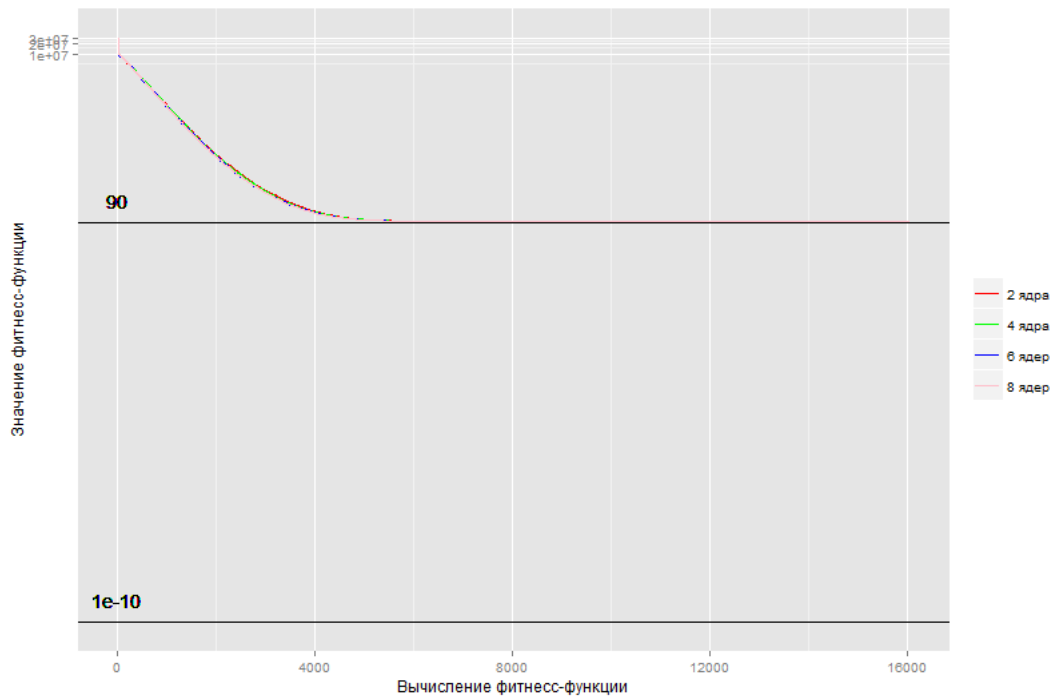


Рисунок 9 – Графики медианных значений для функции Розенброка

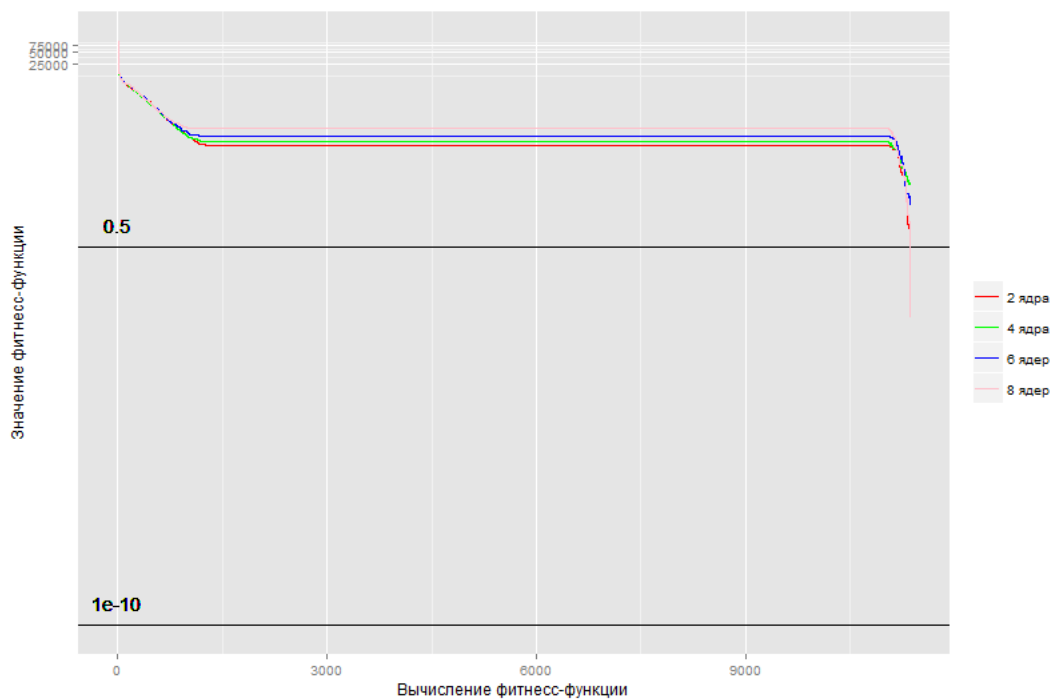


Рисунок 10 – Графики медианных значений для гиперэллипсоида

стоящие (в логарифмическом масштабе) пороговые значения функции приспособленности. Были выбраны значения 1 , $3 \cdot 10^{-2}$, 10^{-5} , $3 \cdot 10^{-7}$ и 10^{-10} .

Рисунок 9 демонстрирует иной характер сходимости на функции Розенброка. Данная функция представляет собой большую трудность для разрабо-

танного алгоритма. На рисунке 9 видно, что все запуски доходят до значения, примерно равного 90, и дальше не демонстрируют никакого прогресса. Столь единообразное поведение, по-видимому, вызвано характерной особенностью функции Розенброка — при ее оптимизации алгоритмы сначала достигают произвольного места «ложбины», а затем пытаются найти в этой ложбине оптимум, что затруднительно по причине крайне малого градиента. По указанной причине в множество пороговых значений разумно включить значение 100, как близкое к наблюдаемому значению стагнации. Также было решено включить значение 3000, как значение, которое легко достижимо даже слабыми оптимизаторами, значение 10^{-10} как используемое в качестве критерия останова, а также два промежуточных значения 10^{-2} и 10^{-6} .

Наконец, рисунок 10 демонстрирует еще одну разновидность сходимости разработанного алгоритма на гиперэллипсоиде. В случае гиперэллипсоида алгоритм также сравнительно быстро достигает некоторого порогового значения, а затем в течение длительного времени не демонстрирует никакого прогресса. Однако спустя некоторое время (порядка десяти тысяч вычислений функции приспособленности для размерности 100) алгоритм «переучивается» на новые значения градиента и доводит оптимизацию до конца. Такой характер оптимизации говорит о том, что используемое в разработанном алгоритме «правило одной пятой» для обновления параметров является недостаточно эффективным для решения указанной задачи. Схожесть характера оптимизации данной функции с функцией Розенброка мотивирует аналогичный выбор пороговых значений: 3000, 100, 10^{-2} , 10^{-6} и 10^{-10} .

3.3. Результаты эксперимента

Результаты эксперимента приведены в таблицах 1–4, размещенных в приложении А. В разделах 3.3.1–3.3.4 произведено сравнение рассматриваемых алгоритмов для каждого вида оптимизируемой функции по отдельности.

3.3.1. Функция де Джонга

В таблице 1 приведены результаты эксперимента для функции де Джонга. В ячейках таблицы через наклонную черту даны медианное время достижения порогового значения (в секундах) и средняя загрузка процессоров.

Из таблицы 1 видно, что в целом алгоритм LM-CMA-ES кажется более эффективным для рассмотренных конфигураций. Наибольшие проблемы у

разработанного алгоритма возникают, когда число ядер велико, а вычислительная сложность — мала. В этом случае доминируют вычисления в критической секции, а общая загрузка системы стремится к 1.0 (в каждый момент времени работает только один процессор). Из представленных в таблице результатов худший случай наблюдается, когда число ядер равно восьми, а дополнительная сложность равна нулю — в этом случае разработанный алгоритм достигает пороговых значений в четыре раза медленнее, чем LM-CMA-ES.

Однако при возрастании вычислительной сложности функции эффективность разработанного алгоритма быстро растет. Так, для двух ядер загрузка достигает 1.9 уже при минимальной ненулевой «нагрузке», а для восьми ядер, дополнительной сложности 600 и размерности 100 загрузка уже равна 7.8 против 6.8 у LM-CMA-ES. При достаточно большом числе ядер (начиная с четырех) и при дополнительной сложности, равной 600, разработанный алгоритм достигает пороговых значений быстрее, чем LM-CMA-ES.

Таким образом, при большом числе используемых ядер процессоров и при большой вычислительной сложности функции использование разработанного алгоритма становится предпочтительнее, так как, несмотря на более слабое правило обновления вычислительного состояния, алгоритм тем не менее достигает заданных значений приспособленности быстрее благодаря более эффективному использованию ресурсов многопроцессорной вычислительной машины.

3.3.2. Функция Растригина

В таблице 2 приведены результаты эксперимента для функции Растригина. Можно наблюдать, что разработанный алгоритм всегда достигает все рассматриваемые пороговые значения (в частности, находит оптимум с точностью до 10^{-10}), в то время как алгоритм LM-CMA-ES оказался неспособным найти не только оптимум, но даже и точку со значением функции, меньшим единицы.

Данный результат оказался неожиданным для автора диссертации, так как в разработанном алгоритме использовалось более простое правило обновления состояния, которое признается менее эффективным, чем правило, используемое в LM-CMA-ES.

3.3.3. Функция Розенброка

В таблице 3 приведены результаты эксперимента для функции Розенброка. Данная функция является сложной для обоих рассматриваемых алгоритмов — в частности, для размерности 1000 оба алгоритма нашли решения с значением функции, меньшим 3 000, но ни один из алгоритмов не оказался способен найти значение, меньшее 100.

Для размерности 100 алгоритм LM-CMA-ES обычно способен найти глобальный оптимум, однако для прохождения регионов приспособленности с 3 000 до 100 и со 100 до 0.01 ему обычно требуется значительное время. Тот факт, что для размерности 300 в таблице отображены успешные попытки нахождения глобального оптимума лишь для двух ядер, а во всех остальных случаях минимальным найденным значением остается 100, говорит о том, что в этом случае число вычислений функции приспособленности, требуемое для нахождения глобального оптимума, слишком близко к пороговому значению 10^6 , и в случае большого числа ядер общего числа итераций уже не хватает.

Разработанный алгоритм достигает еще меньшего числа пороговых значений, но в случае большой вычислительной сложности он делает это быстрее, в некоторых случаях — в два и более раза. Это может объясняться более эффективным использованием ресурсов (большая загрузка многопроцессорной системы), но полным объяснением это быть не может. Вероятно, более простое правило обновления состояния быстрее приводит к регионам с меньшим значением приспособленности, которые при этом находятся дальше от глобального оптимума.

3.3.4. Гиперэллипсоид

В таблице 4 приведены результаты эксперимента для гиперэллипсоида. Данная функция является примером того, как «продвинутое» правило обновления состояния, используемое в LM-CMA-ES, помогает вести оптимизацию в условиях быстроменяющихся градиентов, в то время как правило «одной пятой» не позволяет сделать этого достаточно быстро. Несмотря на то, что в предварительных экспериментах разработанный алгоритм был способен найти оптимум этой функции, он делает это в менее, чем 50 % случаев, что подтверждается отсутствующими медианами в соответствующих столбцах.

Однако и в случае гиперэллипсоида проявляется эффект «жадного» правила, используемого в разработанном алгоритме — достигнутые пороговые

значения он достигает быстрее, чем LM-CMA-ES, и это ускорение нельзя объяснить одним лишь ускорением от эффективного использования многопроцессорной системы.

Выводы по главе 3

В данной главе описана методология проведения эксперимента, конфигурации эксперимента, а также приведен анализ его результатов. По итогам анализа можно сделать общий вывод: разработанный алгоритм оказывается эффективнее, чем LM-CMA-ES, когда используется большое число процессорных ядер, а вычислительная сложность функции приспособленности высока. Кроме того, спецификой использования правила «одной пятой», используемого в разработанном алгоритме, является то, что алгоритм быстрее достигает локальных оптимумов, что может оказаться полезным при его использовании для решения практических задач. Сюрпризом послужило то, что разработанный алгоритм легко находит оптимум функции Растригина, в то время как алгоритм LM-CMA-ES не способен этого сделать.

ЗАКЛЮЧЕНИЕ

В данной работе предложен метод эффективной параллельной реализации эволюционной стратегии с адаптацией ковариационной матрицы на основе алгоритма LM-CMA.

Метод успешно исследован на нескольких стандартных задачах оптимизации, проведено подробное сравнение с изначальным алгоритмом LM-CMA, описаны необходимые изменения.

Исследование показало эффективность предлагаемого подхода при вычислении функций с высокой вычислительной сложностью в случае использовании большого числа процессорных ядер.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 *Rechenberg I.* Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. — FrommannHolzboog Verlag, 1973.
- 2 *Rechenberg I.* Evolutionsstrategie '94. — Frommann-Holzboog Verlag, 1994.
- 3 *Schwefel H.-P.* Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. — Birkhäuser, 1977.
- 4 *Schwefel H.-P.* Evolution and Optimum Seeking. — Wiley, 1995.
- 5 *Hansen N., Ostermeier A.* Completely derandomized self-adaptation in evolution strategies // *Evolutionary Computation*. — 2001. — Т. 9, № 2. — С. 159—195.
- 6 *Hansen N.* An Analysis of Mutative σ -Self-Adaptation on Linear Fitness Functions // *Evolutionary Computation*. — 2006. — Т. 3, № 14. — С. 255—275.
- 7 *Rudolph G.* Convergence Properties of Evolutionary Algorithms. — Dr. Kovač, 1997.
- 8 *Arnold D. V.* Weighted multirecombination evolution strategies // *Theoretical Computer Science*. — 2006. — Т. 361, № 1. — С. 18—37.
- 9 *Schumer M., Steiglitz K.* Adaptive step size random search // *IEEE Transactions on Automatic Control*. — 1968. — Т. 13, № 3. — С. 270—276.
- 10 Learning probability distributions in continuous evolutionary algorithms — A comparative review / S. Kern [и др.] // *Natural Computing*. — 2004. — Т. 3, № 1. — С. 77—112.
- 11 *Hansen N.* An analysis of mutative σ -self-adaptation on linear fitness functions // *Evolutionary Computation*. — 2006. — Т. 14, № 3. — С. 255—275.
- 12 *Ostermeier A., Gawelczyk A., Hansen N.* A derandomized approach to self-adaptation of evolution strategies // *Evolutionary Computation*. — 1994. — Т. 2, № 4. — С. 369—380.
- 13 *Ostermeier A., Gawelczyk A., Hansen N.* Step-size adaptation based on non-local use of selection information // *Parallel Problem Solving from Nature (PPSN III)*. — 1994. — С. 189—198.

- 14 *Hansen N., Ostermeier A., Gawelczyk A.* On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation // International Conference on Genetic Algorithms. — 1995.
- 15 *Ostermeier A.* An evolution strategy with momentum adaptation of the random number distribution // Parallel Problem Solving from Nature (PPSN II). — 1992. — C. 199–208.
- 16 *Poland J., Zell A.* Main vector adaptation: A CMA variant with linear time and space complexity // Genetic and Evolutionary Computation Conference (GECCO 2001). — 2001.
- 17 *Knight J. N., Lunacek M.* Reducing the space-time complexity of the CMA-ES // Genetic and Evolutionary Computation Conference (GECCO 2007). — 2007.
- 18 Impacts of Invariance in Search: When CMA-ES and PSO Face Ill-Conditioned and Non-Separable Problems / N. Hansen [и др.] // Applied Soft Computing. — № 11. — C. 5755–5769.
- 19 *Suttorp T., Hansen N., Igel C.* Efficient Covariance Matrix Update for Variable Metric Evolution Strategies // Machine Learning. — № 75. — C. 167–197.
- 20 *Hansen N., Ostermeier A.* Completely Derandomized Self-Adaptation in Evolution Strategies // Evolutionary Computation. — 2001. — T. 2, № 9. — C. 159–195.
- 21 *Hansen N., Müller S., Koumoutsakos P.* Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES) // Evolutionary Computation. — 2003. — T. 1, № 11. — C. 1–18.
- 22 *Hansen N., Kern S.* Evaluating the CMA evolution strategy on multimodal test functions // Parallel Problem Solving from Nature - PPSN VIII. — 2004. — № 8. — C. 282–291.
- 23 *Igel C., Hansen N., Roth S.* Covariance Matrix Adaptation for Multi-objective Optimization // Evolutionary Computation. — 2007. — T. 1, № 15. — C. 1–28.
- 24 *Hansen N.* CMA-ES Source Code. — URL: https://www.lri.fr/~hansen/cmaes_inmatlab.html.
- 25 *Loschilov I.* LM-CMA-ES Source Code. — URL: <https://sites.google.com/site/lmcmases/>.

- 26 *Elhara O. A., Auger A., Hansen N.* A median success rule for non-elitist evolution strategies: Study of feasibility // Genetic and Evolutionary Computation Conference. — 2013.
- 27 GEATbx: Example Functions (single and multi-objective functions). — URL: <http://www.geatbx.com/docu/fcnindex-01.html>.
- 28 *Yen, Y. J.* An algorithm for finding shortest routes from all source nodes to a given destination in general networks // Quarterly of Applied Mathematics. — 1970. — № 27. — С. 526—530.
- 29 *Binh T.* A multiobjective evolutionary algorithm. The study cases. Тех. отч. / Institute for Automation ; Communication. — 1999.
- 30 *Bäck T.* Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms. — Birkhäuser, 1977.
- 31 *Binh T., Korn U.* A Multiobjective Evolution Strategy for Constrained Optimization Problems // Third International Conference on Genetic Algorithms. — 1997. — С. 176—182.
- 32 *Oldenhuis R.* Many test functions for global optimizers // Mathworks. — 2012.
- 33 Test functions and datasets. — URL: <http://www.geatbx.com/docu/fcnindex-01.html>.
- 34 *Loshchilov I.* A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization // Genetic and Evolutionary Computation Conference (ACM 2014). — 2014.
- 35 *Bellman, Richard* On a routing problem // Quarterly of Applied Mathematics. — 1958. — № 16. — С. 87—90.
- 36 *Ford Jr L., Lester R.* Network Flow Theory // P-923. — 1959.

ПРИЛОЖЕНИЕ А. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА

Таблица 1 – Результаты эксперимента для функции де Джонга. В ячейках таблицы через наклонную черту даны медианное время достижения порогового значения (в секундах) и медианное значение загрузки процессоров

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
2 ядра, дополнительная сложность — 0						
100	Proposed	0.17/1.6	0.25/1.6	0.42/1.6	0.49/1.6	0.65/1.6
	LM-CMA	0.11/1.7	0.15/1.7	0.24/1.7	0.28/1.7	0.38/1.7
300	Proposed	1.18/1.7	1.63/1.7	2.67/1.7	3.13/1.7	4.15/1.7
	LM-CMA	0.55/1.7	0.76/1.8	1.22/1.8	1.42/1.8	1.91/1.8
1000	Proposed	10.38/1.6	14.04/1.6	22.22/1.6	25.75/1.6	33.84/1.6
	LM-CMA	5.45/1.8	7.42/1.8	11.87/1.8	13.88/1.8	18.34/1.8
2 ядра, дополнительная сложность — 200						
100	Proposed	0.46/1.9	0.62/1.9	1.07/1.9	1.28/1.9	1.69/1.9
	LM-CMA	0.26/1.9	0.36/1.9	0.6/1.9	0.71/1.9	0.96/1.9
300	Proposed	1.72/1.8	2.39/1.8	3.86/1.8	4.52/1.8	6.03/1.8
	LM-CMA	0.97/1.8	1.32/1.8	2.15/1.8	2.5/1.8	3.33/1.8
1000	Proposed	10.46/1.7	13.98/1.7	22.11/1.7	25.67/1.7	33.75/1.7
	LM-CMA	6.2/1.8	8.56/1.8	13.88/1.8	16.43/1.8	22.08/1.8
2 ядра, дополнительная сложность — 400						
100	Proposed	1.59/1.9	2.23/1.9	3.74/1.9	4.4/1.9	5.94/1.9
	LM-CMA	1.45/1.9	2.56/1.9	2.97/1.9	4.04/1.9	1.01/1.9
300	Proposed	3.87/1.9	5.4/1.9	8.79/1.9	10.31/1.9	13.82/1.9
	LM-CMA	3.02/1.9	4.26/1.9	7.02/1.9	8.18/1.9	10.91/1.9
1000	Proposed	17.3/1.8	23.46/1.8	36.71/1.8	42.81/1.8	56.0/1.8
	LM-CMA	14.73/1.8	20.14/1.8	32.7/1.8	38.11/1.8	52.13/1.8
2 ядра, дополнительная сложность — 600						
100	Proposed	3.23/1.9	4.78/1.9	8.19/1.9	9.49/1.9	12.96/1.9
	LM-CMA	2.29/1.9	3.39/1.9	5.66/1.9	6.8/1.9	9.39/1.9
300	Proposed	9.27/1.9	12.99/1.9	21.32/1.9	25.49/1.9	34.05/1.9
	LM-CMA	6.31/1.9	8.79/1.9	14.45/1.9	16.84/1.9	22.68/1.9
1000	Proposed	30.73/1.9	41.52/1.9	65.3/1.9	76.12/1.9	100.51/1.9

Продолжение таблицы 1

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
	LM-CMA	23.79/1.9	32.4/1.9	51.3/1.9	60.22/1.9	79.74/1.9
4 ядра, дополнительная сложность — 0						
100	Proposed	0.27/1.1	0.39/1.2	0.66/1.1	0.76/1.1	1.03/1.1
	LM-CMA	0.07/2.5	0.1/2.5	0.17/2.5	0.2/2.5	0.27/2.5
300	Proposed	1.42/1.7	1.96/1.7	3.2/1.7	3.74/1.7	4.99/1.6
	LM-CMA	0.43/2.6	0.59/2.6	0.95/2.6	1.11/2.6	1.48/2.6
1000	Proposed	10.31/1.9	13.83/1.9	22.03/1.9	25.52/1.9	33.62/1.9
	LM-CMA	3.77/2.8	5.13/2.8	8.2/2.8	9.52/2.8	12.57/2.8
4 ядра, дополнительная сложность — 200						
100	Proposed	0.22/3.5	0.31/3.5	0.54/3.4	0.64/3.4	0.86/3.4
	LM-CMA	0.16/3.4	0.23/3.4	0.39/3.5	0.46/3.5	0.66/3.5
300	Proposed	1.09/3.0	1.53/3.0	2.53/3.0	2.98/3.0	3.97/3.0
	LM-CMA	0.65/3.4	0.93/3.4	1.49/3.4	1.75/3.4	2.33/3.4
1000	Proposed	7.93/2.5	10.69/2.5	16.98/2.5	19.75/2.5	25.95/2.5
	LM-CMA	4.4/3.0	5.96/3.0	9.58/3.0	11.28/3.0	14.83/3.0
4 ядра, дополнительная сложность — 400						
100	Proposed	0.66/3.9	0.94/3.9	1.59/3.9	1.88/3.9	2.5/3.9
	LM-CMA	0.73/3.6	1.06/3.6	1.76/3.6	2.13/3.6	2.84/3.
300	Proposed	2.04/3.7	2.85/3.7	4.7/3.7	5.5/3.7	7.27/3.7
	LM-CMA	2.08/3.6	2.88/3.6	4.9/3.6	5.69/3.6	7.49/3.6
1000	Proposed	9.9/3.0	13.41/3.0	21.28/3.0	24.84/3.0	32.64/3.0
	LM-CMA	8.82/3.2	12.03/3.2	19.97/3.2	23.54/3.2	31.36/3.2
4 ядра, дополнительная сложность — 600						
100	Proposed	1.31/3.9	1.92/3.9	3.27/3.9	3.81/3.9	5.2/3.9
	LM-CMA	1.51/3.7	2.14/3.7	3.64/3.7	4.36/3.7	5.67/3.7
300	Proposed	4.32/3.9	6.03/3.9	10.03/3.9	11.89/3.9	15.92/3.9
	LM-CMA	4.58/3.7	6.56/3.7	10.88/3.7	12.89/3.7	17.51/3.7
1000	Proposed	16.55/3.6	22.38/3.6	34.93/3.6	40.52/3.6	53.01/3.6
	LM-CMA	15.15/3.5	20.28/3.5	33.64/3.5	39.07/3.5	52.54/3.5
6 ядер, дополнительная сложность — 0						
100	Proposed	0.29/1.0	0.42/1.0	0.7/0.9	0.84/0.9	1.12/0.9

Продолжение таблицы 1

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
	LM-CMA	0.08/2.3	0.11/2.4	0.17/2.3	0.21/2.3	0.28/2.3
300	Proposed	1.6/1.3	2.24/1.3	3.65/1.3	4.29/1.3	5.7/1.3
	LM-CMA	0.44/2.5	0.6/2.5	0.99/2.5	1.15/2.5	1.54/2.5
1000	Proposed	11.08/1.7	14.95/1.7	23.6/1.7	27.49/1.7	36.26/1.7
	LM-CMA	3.55/3.0	4.86/3.1	7.76/3.1	9.02/3.1	11.89/3.1
6 ядер, дополнительная сложность — 200						
100	Proposed	0.28/4.4	0.37/4.4	0.61/4.4	0.72/4.4	0.93/4.3
	LM-CMA	0.16/4.7	0.23/4.8	0.37/4.8	0.43/4.8	0.57/4.8
300	Proposed	1.22/3.3	1.69/3.3	2.77/3.4	3.2/3.4	4.27/3.4
	LM-CMA	0.51/4.6	0.69/4.7	1.16/4.8	1.36/4.7	1.82/4.8
1000	Proposed	8.83/2.5	11.9/2.5	18.87/2.5	21.96/2.5	29.04/2.5
	LM-CMA	4.34/3.4	5.96/3.5	9.45/3.5	10.96/3.5	14.52/3.4
6 ядер, дополнительная сложность — 400						
100	Proposed	0.48/5.7	0.7/5.8	1.14/5.8	1.36/5.8	1.88/5.8
	LM-CMA	0.63/5.2	0.86/5.2	1.43/5.2	1.67/5.2	2.2/5.1
300	Proposed	1.57/5.1	2.23/5.1	3.67/5.1	4.33/5.1	5.81/5.1
	LM-CMA	1.49/5.1	2.12/5.1	3.55/5.1	4.13/5.1	5.52/5.1
1000	Proposed	9.38/4.0	12.56/4.0	19.8/4.0	23.04/4.0	30.46/4.0
	LM-CMA	8.03/4.0	11.44/4.0	18.44/3.9	21.55/3.9	29.44/3.9
6 ядер, дополнительная сложность — 600						
100	Proposed	0.99/5.9	1.46/5.9	2.47/5.9	2.94/5.9	4.02/5.9
	LM-CMA	1.42/5.3	2.05/5.3	3.2/5.3	3.73/5.3	4.94/5.2
300	Proposed	3.3/5.7	4.52/5.7	7.49/5.7	8.82/5.7	11.87/5.7
	LM-CMA	3.52/5.3	4.83/5.3	7.85/5.2	9.28/5.2	12.45/5.2
1000	Proposed	12.66/4.9	16.99/4.9	26.84/4.9	31.11/4.9	40.95/4.9
	LM-CMA	13.02/4.8	17.75/4.8	29.68/4.8	34.6/4.8	46.78/4.8
8 ядер, дополнительная сложность — 0						
100	Proposed	0.3/1.0	0.43/1.0	0.73/1.0	0.86/1.0	1.16/1.0
	LM-CMA	0.08/2.3	0.11/2.3	0.18/2.3	0.21/2.3	0.28/2.3
300	Proposed	1.64/1.4	2.27/1.4	3.73/1.4	4.37/1.3	5.83/1.3
	LM-CMA	0.46/2.4	0.64/2.4	1.03/2.4	1.2/2.4	1.6/2.4

Продолжение таблицы 1

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
1000	Proposed	11.18/1.7	14.99/1.7	23.83/1.7	27.69/1.7	36.45/1.7
	LM-CMA	3.63/3.0	4.94/3.0	7.91/3.0	9.22/3.0	12.15/3.0
8 ядер, дополнительная сложность — 200						
100	Proposed	0.32/4.1	0.46/4.1	0.76/4.1	0.9/4.2	1.18/4.2
	LM-CMA	0.13/5.3	0.2/5.3	0.35/5.2	0.4/5.3	0.51/5.3
300	Proposed	1.39/3.2	1.97/3.2	3.16/3.3	3.71/3.2	4.95/3.2
	LM-CMA	0.42/5.6	0.58/5.7	0.97/5.7	1.13/5.7	1.49/5.7
1000	Proposed	9.22/2.5	12.56/2.5	19.9/2.5	23.13/2.5	30.47/2.5
	LM-CMA	4.72/3.6	6.47/3.7	10.77/3.6	12.28/3.6	16.05/3.6
8 ядер, дополнительная сложность — 400						
100	Proposed	0.39/7.3	0.55/7.4	0.93/7.4	1.09/7.4	1.46/7.3
	LM-CMA	0.49/6.6	0.68/6.5	1.13/6.5	1.32/6.5	1.77/6.5
300	Proposed	1.54/6.6	2.12/6.6	3.53/6.5	4.08/6.5	5.39/6.6
	LM-CMA	1.23/6.5	1.79/6.6	3.02/6.6	3.51/6.6	4.79/6.6
1000	Proposed	9.41/4.4	12.65/4.4	20.17/4.4	23.59/4.4	31.05/4.4
	LM-CMA	8.47/4.4	11.6/4.4	18.91/4.4	22.32/4.3	29.63/4.2
8 ядер, дополнительная сложность — 600						
100	Proposed	1.19/7.8	1.96/7.8	2.3/7.8	3.12/7.8	0.85/7.8
	LM-CMA	1.28/6.8	1.73/6.8	2.96/6.8	3.47/6.8	4.52/6.8
300	Proposed	2.69/7.4	3.74/7.5	6.25/7.5	7.46/7.5	10.07/7.5
	LM-CMA	2.87/6.8	3.87/6.8	6.34/6.8	7.35/6.8	9.93/6.8
1000	Proposed	11.86/6.1	16.07/6.1	25.53/6.1	29.48/6.1	39.06/6.1
	LM-CMA	12.41/5.9	17.39/5.8	28.12/5.8	32.75/5.8	43.58/5.8

Таблица 2 – Результаты эксперимента для функции Растригина. В ячейках таблицы через наклонную черту даны медианное время достижения порогового значения (в секундах) и медианное значение загрузки процессоров

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
2 ядра, дополнительная сложность — 0						
100	Proposed	0.29/1.6	0.37/1.6	0.56/1.6	0.64/1.6	0.83/1.6
	LM-CMA	—	—	—	—	—
300	Proposed	1.97/1.7	2.51/1.7	3.72/1.7	4.24/1.7	5.41/1.7
	LM-CMA	—	—	—	—	—
1000	Proposed	17.54/1.6	21.67/1.6	31.17/1.6	35.24/1.6	44.13/1.6
	LM-CMA	—	—	—	—	—
2 ядра, дополнительная сложность — 200						
100	Proposed	0.68/1.9	0.91/1.9	1.41/1.9	1.63/1.9	2.16/1.9
	LM-CMA	—	—	—	—	—
300	Proposed	2.53/1.8	3.18/1.8	4.74/1.8	5.45/1.8	6.93/1.8
	LM-CMA	—	—	—	—	—
1000	Proposed	18.83/1.7	23.31/1.7	33.68/1.7	38.37/1.7	48.24/1.7
	LM-CMA	—	—	—	—	—
2 ядра, дополнительная сложность — 400						
100	Proposed	2.26/1.9	3.0/1.9	4.72/1.9	5.53/1.9	7.12/1.9
	LM-CMA	—	—	—	—	—
300	Proposed	6.76/1.9	8.6/1.9	12.74/1.9	14.62/1.9	18.89/1.9
	LM-CMA	—	—	—	—	—
1000	Proposed	32.42/1.8	40.52/1.8	58.98/1.8	67.34/1.8	85.16/1.8
	LM-CMA	—	—	—	—	—
2 ядра, дополнительная сложность — 600						
100	Proposed	5.07/1.9	6.67/1.9	10.4/1.9	12.11/1.9	15.81/1.9
	LM-CMA	—	—	—	—	—
300	Proposed	12.32/1.9	15.85/1.9	23.79/1.9	27.34/1.9	35.07/1.9
	LM-CMA	—	—	—	—	—
1000	Proposed	52.96/1.9	66.23/1.9	95.9/1.9	109.09/1.9	136.96/1.9
	LM-CMA	—	—	—	—	—
4 ядра, дополнительная сложность — 0						

Продолжение таблицы 2

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
100	Proposed	0.38/1.9	0.49/1.8	0.74/1.7	0.85/1.7	1.12/1.7
	LM-CMA	—	—	—	—	—
300	Proposed	1.84/2.3	2.35/2.3	3.5/2.2	4.0/2.2	5.14/2.2
	LM-CMA	—	—	—	—	—
1000	Proposed	14.14/2.3	17.59/2.3	25.5/2.3	28.93/2.3	36.26/2.3
	LM-CMA	—	—	—	—	—
4 ядра, дополнительная сложность — 200						
100	Proposed	0.44/3.4	0.59/3.3	0.8/3.2	0.9/3.2	1.17/3.2
	LM-CMA	—	—	—	—	—
300	Proposed	1.54/3.0	2.0/3.0	3.06/3.0	3.49/3.0	4.43/3.0
	LM-CMA	—	—	—	—	—
1000	Proposed	11.6/2.6	14.46/2.6	21.18/2.6	24.16/2.6	30.27/2.6
	LM-CMA	—	—	—	—	—
4 ядра, дополнительная сложность — 400						
100	Proposed	1.0/3.8	1.3/3.8	1.96/3.8	2.23/3.8	2.85/3.8
	LM-CMA	—	—	—	—	—
300	Proposed	2.88/3.7	3.7/3.6	5.46/3.6	6.31/3.6	8.04/3.7
	LM-CMA	—	—	—	—	—
1000	Proposed	14.71/3.1	18.36/3.1	27.13/3.0	30.76/3.0	38.65/3.0
	LM-CMA	—	—	—	—	—
4 ядра, дополнительная сложность — 600						
100	Proposed	1.98/3.9	2.53/3.9	3.77/3.8	4.31/3.9	5.49/3.9
	LM-CMA	—	—	—	—	—
300	Proposed	5.79/3.8	7.37/3.8	11.14/3.8	12.79/3.8	16.53/3.8
	LM-CMA	—	—	—	—	—
1000	Proposed	22.88/3.5	28.49/3.5	40.98/3.5	46.64/3.5	58.23/3.5
	LM-CMA	—	—	—	—	—
6 ядер, дополнительная сложность — 0						
100	Proposed	0.48/1.1	0.63/1.1	0.96/1.0	1.11/1.0	1.43/1.0
	LM-CMA	—	—	—	—	—
300	Proposed	2.47/1.6	3.17/1.6	4.72/1.5	5.4/1.5	6.93/1.5
	LM-CMA	—	—	—	—	—

Продолжение таблицы 2

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
	LM-CMA	—	—	—	—	—
1000	Proposed	15.9/2.1	19.9/2.1	28.98/2.0	32.92/2.0	41.43/2.0
	LM-CMA	—	—	—	—	—
6 ядер, дополнительная сложность — 200						
100	Proposed	0.43/3.1	0.55/3.1	0.83/3.2	0.98/3.1	1.32/3.0
	LM-CMA	—	—	—	—	—
300	Proposed	1.59/3.8	2.04/3.7	3.1/3.6	3.54/3.6	4.54/3.7
	LM-CMA	—	—	—	—	—
1000	Proposed	13.12/2.7	16.49/2.6	24.01/2.6	27.29/2.6	34.32/2.6
	LM-CMA	—	—	—	—	—
6 ядер, дополнительная сложность — 400						
100	Proposed	0.7/5.6	0.92/5.6	1.33/5.6	1.55/5.6	2.04/5.6
	LM-CMA	—	—	—	—	—
300	Proposed	2.22/5.0	2.9/5.1	4.3/5.1	4.93/5.1	6.35/5.2
	LM-CMA	—	—	—	—	—
1000	Proposed	13.72/3.8	17.22/3.8	25.13/3.8	28.71/3.8	36.15/3.8
	LM-CMA	—	—	—	—	—
6 ядер, дополнительная сложность — 600						
100	Proposed	1.34/5.8	1.76/5.8	2.62/5.7	3.09/5.7	4.24/5.7
	LM-CMA	—	—	—	—	—
300	Proposed	4.16/5.6	5.38/5.6	8.12/5.6	9.36/5.6	12.1/5.6
	LM-CMA	—	—	—	—	—
1000	Proposed	17.74/4.8	22.19/4.7	32.23/4.7	36.81/4.7	46.23/4.7
	LM-CMA	—	—	—	—	—
8 ядер, дополнительная сложность — 0						
100	Proposed	0.47/1.0	0.61/1.0	0.94/0.9	1.09/0.9	1.41/0.9
	LM-CMA	—	—	—	—	—
300	Proposed	2.58/1.4	3.26/1.4	4.85/1.4	5.57/1.4	7.21/1.3
	LM-CMA	—	—	—	—	—
1000	Proposed	16.35/2.0	20.47/1.9	29.7/1.9	33.83/1.9	42.39/1.9
	LM-CMA	—	—	—	—	—

Продолжение таблицы 2

Размерность пространства	Алгоритм	Пороговые значения				
		1	$3 \cdot 10^{-2}$	10^{-5}	$3 \cdot 10^{-7}$	10^{-10}
8 ядер, дополнительная сложность — 200						
100	Proposed	0.6/2.3	0.81/2.3	1.23/2.3	1.4/2.3	1.78/2.5
	LM-CMA	—	—	—	—	—
300	Proposed	2.15/2.9	2.74/2.8	4.14/2.7	4.74/2.7	6.17/2.7
	LM-CMA	—	—	—	—	—
1000	Proposed	14.83/2.3	18.53/2.3	27.27/2.3	30.8/2.3	38.83/2.3
	LM-CMA	—	—	—	—	—
8 ядер, дополнительная сложность — 400						
100	Proposed	0.7/5.4	0.92/5.5	1.4/5.5	1.61/5.6	2.11/5.8
	LM-CMA	—	—	—	—	—
300	Proposed	2.25/5.3	2.87/5.3	4.35/5.3	4.97/5.3	6.4/5.2
	LM-CMA	—	—	—	—	—
1000	Proposed	14.23/4.2	17.73/4.1	25.76/4.1	29.27/4.1	36.93/4.1
	LM-CMA	—	—	—	—	—
8 ядер, дополнительная сложность — 600						
100	Proposed	1.12/7.5	1.5/7.4	2.38/7.4	2.75/7.5	3.56/7.4
	LM-CMA	—	—	—	—	—
300	Proposed	4.01/7.1	5.07/7.1	7.37/7.0	8.41/7.1	10.88/7.1
	LM-CMA	—	—	—	—	—
1000	Proposed	18.1/5.7	22.67/5.7	33.06/5.7	37.62/5.7	46.82/5.6
	LM-CMA	—	—	—	—	—

Таблица 3 – Результаты эксперимента для функции Розенброка. В ячейках таблицы через наклонную черту даны медианное время достижения порогового значения (в секундах) и медианное значение загрузки процессоров

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
2 ядра, дополнительная сложность — 0						
100	Proposed	0.09/1.6	0.23/1.6	—	—	—
	LM-CMA	0.05/1.7	1.32/1.7	6.41/1.7	6.55/1.7	6.72/1.7
300	Proposed	0.71/1.6	—	—	—	—
	LM-CMA	0.43/1.8	47.04/1.7	53.22/1.7	54.03/1.7	54.7/1.7
1000	Proposed	7.04/1.6	—	—	—	—
	LM-CMA	6.08/1.8	—	—	—	—
2 ядра, дополнительная сложность — 200						
100	Proposed	0.29/1.9	0.71/1.9	—	—	—
	LM-CMA	0.16/1.8	1.9/1.9	20.5/1.8	21.07/1.8	21.61/1.8
300	Proposed	1.08/1.8	—	—	—	—
	LM-CMA	0.72/1.8	69.03/1.8	23.27/0.9	24.21/0.9	25.05/0.9
1000	Proposed	7.15/1.7	—	—	—	—
	LM-CMA	6.9/1.8	—	—	—	—
2 ядра, дополнительная сложность — 400						
100	Proposed	0.92/1.9	2.44/1.9	—	—	—
	LM-CMA	0.59/1.9	2.9/1.9	78.27/1.9	80.25/1.9	82.35/1.9
300	Proposed	2.81/1.9	—	—	—	—
	LM-CMA	2.92/1.8	136.75/1.7	—	—	—
1000	Proposed	12.46/1.8	—	—	—	—
	LM-CMA	9.5/1.8	—	—	—	—
2 ядра, дополнительная сложность — 600						
100	Proposed	1.9/1.9	4.81/1.9	—	—	—
	LM-CMA	1.51/1.9	6.76/1.9	179.06/1.9	183.73/1.9	188.64/1.9
300	Proposed	5.84/1.9	—	—	—	—
	LM-CMA	5.82/1.9	459.66/1.8	252.88/1.7	180.88/0.9	187.57/0.9
1000	Proposed	19.8/1.9	—	—	—	—
	LM-CMA	28.82/1.9	—	—	—	—
4 ядра, дополнительная сложность — 0						

Продолжение таблицы 3

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
100	Proposed	0.13/1.8	0.31/1.8	—	—	—
	LM-CMA	0.04/2.6	1.23/2.5	5.12/2.3	5.24/2.3	5.38/2.3
300	Proposed	0.75/2.1	—	—	—	—
	LM-CMA	0.34/2.6	33.76/2.6	16.01/1.2	16.38/1.2	—
1000	Proposed	6.51/2.1	—	—	—	—
	LM-CMA	4.2/2.8	—	—	—	—
4 ядра, дополнительная сложность — 200						
100	Proposed	0.14/3.4	0.34/3.4	—	—	—
	LM-CMA	0.1/3.4	0.38/3.5	12.53/3.5	12.85/3.5	13.16/3.5
300	Proposed	0.66/3.0	—	—	—	—
	LM-CMA	0.51/3.4	36.05/3.2	—	—	—
1000	Proposed	5.59/2.5	—	—	—	—
	LM-CMA	4.63/3.1	—	—	—	—
4 ядра, дополнительная сложность — 400						
100	Proposed	0.37/3.9	0.88/3.9	—	—	—
	LM-CMA	0.32/3.6	2.93/3.6	40.15/3.6	41.13/3.6	42.09/3.6
300	Proposed	1.21/3.7	—	—	—	—
	LM-CMA	2.03/3.5	86.35/2.9	—	—	—
1000	Proposed	6.83/3.0	—	—	—	—
	LM-CMA	10.33/3.4	—	—	—	—
4 ядра, дополнительная сложность — 600						
100	Proposed	0.82/3.9	2.15/3.9	—	—	—
	LM-CMA	0.77/3.6	3.03/3.7	90.25/3.6	92.21/3.6	94.42/3.6
300	Proposed	2.29/3.9	—	—	—	—
	LM-CMA	3.47/3.6	310.07/3.6	—	—	—
1000	Proposed	9.83/3.5	—	—	—	—
	LM-CMA	18.41/3.5	—	—	—	—
6 ядер, дополнительная сложность — 0						
100	Proposed	0.16/1.5	0.4/1.4	—	—	—
	LM-CMA	0.04/3.1	0.44/3.0	5.33/2.4	5.44/2.4	5.57/2.3
300	Proposed	0.9/1.7	—	—	—	—

Продолжение таблицы 3

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
	LM-CMA	0.34/2.6	34.74/2.6	39.48/2.5	40.67/2.5	19.86/1.2
1000	Proposed	6.98/2.0	—	—	—	—
	LM-CMA	3.78/3.2	—	—	—	—
6 ядер, дополнительная сложность — 200						
100	Proposed	0.16/4.0	0.37/4.2	—	—	—
	LM-CMA	0.08/4.8	0.79/4.9	11.45/4.9	11.77/4.9	12.06/4.9
300	Proposed	0.75/3.5	—	—	—	—
	LM-CMA	0.33/4.5	26.4/4.6	29.54/4.4	30.36/4.4	30.95/4.4
1000	Proposed	6.5/2.4	—	—	—	—
	LM-CMA	4.17/3.8	—	—	—	—
6 ядер, дополнительная сложность — 400						
100	Proposed	0.29/5.7	0.73/5.8	—	—	—
	LM-CMA	0.3/5.1	2.44/5.1	38.03/5.1	39.0/5.1	39.91/5.1
300	Proposed	0.96/5.1	—	—	—	—
	LM-CMA	0.96/5.1	79.83/5.1	37.55/4.8	39.06/4.8	40.67/4.8
1000	Proposed	6.21/4.0	—	—	—	—
	LM-CMA	7.73/4.2	—	—	—	—
6 ядер, дополнительная сложность — 600						
100	Proposed	0.62/5.9	1.68/5.9	—	—	—
	LM-CMA	0.76/5.3	6.53/5.3	86.33/5.2	88.47/5.2	90.83/5.2
300	Proposed	1.93/5.7	—	—	—	—
	LM-CMA	2.1/5.3	150.7/5.1	35.79/4.4	36.47/4.4	—
1000	Proposed	7.87/4.9	—	—	—	—
	LM-CMA	14.87/4.7	—	—	—	—
8 ядер, дополнительная сложность — 0						
100	Proposed	0.17/1.3	0.42/1.2	—	—	—
	LM-CMA	0.05/2.3	1.24/2.2	5.58/2.1	5.69/2.1	5.83/2.1
300	Proposed	0.94/1.7	—	—	—	—
	LM-CMA	0.34/2.5	44.85/2.4	41.63/2.4	40.6/2.4	34.76/2.4
1000	Proposed	7.22/2.0	—	—	—	—
	LM-CMA	3.92/3.2	—	—	—	—

Продолжение таблицы 3

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
8 ядер, дополнительная сложность — 200						
100	Proposed	0.2/3.6	0.47/3.6	—	—	—
	LM-CMA	0.08/5.8	1.04/5.8	9.28/5.9	9.47/5.9	9.7/5.9
300	Proposed	0.87/2.8	—	—	—	—
	LM-CMA	0.34/5.5	29.96/5.6	26.72/5.2	24.77/5.2	12.68/2.6
1000	Proposed	6.74/2.3	—	—	—	—
	LM-CMA	4.54/4.1	—	—	—	—
8 ядер, дополнительная сложность — 400						
100	Proposed	0.22/7.1	0.55/7.2	—	—	—
	LM-CMA	0.24/6.6	1.23/6.7	32.18/6.6	32.95/6.6	33.67/6.6
300	Proposed	0.88/6.1	—	—	—	—
	LM-CMA	0.89/6.4	66.77/6.2	36.36/5.5	37.58/5.4	38.56/5.4
1000	Proposed	6.35/4.7	—	—	—	—
	LM-CMA	7.4/5.4	—	—	—	—
8 ядер, дополнительная сложность — 600						
100	Proposed	0.49/7.7	1.17/7.8	—	—	—
	LM-CMA	0.71/6.9	9.63/6.9	87.36/6.8	88.94/6.8	91.39/6.8
300	Proposed	1.3/7.3	—	—	—	—
	LM-CMA	2.43/6.8	233.21/6.5	—	—	—
1000	Proposed	6.8/5.9	—	—	—	—
	LM-CMA	15.54/5.8	—	—	—	—

Таблица 4 – Результаты эксперимента для гиперэллипсоида. В ячейках таблицы через наклонную черту даны медианное время достижения порогового значения (в секундах) и медианное значение загрузки процессоров

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
2 ядра, дополнительная сложность — 0						
100	Proposed	0.02/1.6	0.06/1.6	—	—	—
	LM-CMA	0.02/1.7	0.37/1.7	1.89/1.7	3.4/1.7	4.92/1.7
300	Proposed	0.32/1.7	0.58/1.7	—	—	—
	LM-CMA	1.43/1.7	13.11/1.8	48.54/1.8	84.02/1.8	119.75/1.8
1000	Proposed	7.3/1.8	5.66/0.9	—	—	—
	LM-CMA	181.06/1.9	660.1/1.9	—	—	—
2 ядра, дополнительная сложность — 200						
100	Proposed	0.04/1.8	0.1/1.8	—	—	—
	LM-CMA	0.05/1.8	0.93/1.9	4.57/1.9	8.54/1.8	12.49/1.8
300	Proposed	0.36/1.8	0.52/1.7	—	—	—
	LM-CMA	1.54/1.8	16.88/1.8	63.1/1.8	110.26/1.8	157.16/1.8
1000	Proposed	6.95/1.8	5.12/0.9	—	—	—
	LM-CMA	185.05/1.9	660.87/1.9	—	—	—
2 ядра, дополнительная сложность — 400						
100	Proposed	0.14/1.9	0.39/1.9	—	—	—
	LM-CMA	0.19/1.9	3.23/1.9	17.41/1.9	32.68/1.9	47.76/1.9
300	Proposed	0.67/1.9	0.72/1.9	—	—	—
	LM-CMA	4.07/1.9	41.38/1.9	158.47/1.9	276.14/1.9	387.09/1.9
1000	Proposed	7.78/1.9	—	—	—	—
	LM-CMA	244.14/1.9	893.87/1.9	—	—	—
2 ядра, дополнительная сложность — 600						
100	Proposed	0.29/1.9	0.9/1.9	—	—	—
	LM-CMA	0.6/1.9	7.49/1.9	40.08/1.9	73.33/1.9	108.08/1.9
300	Proposed	1.44/1.9	2.0/1.9	—	—	—
	LM-CMA	8.48/1.9	90.54/1.9	344.16/1.9	603.95/1.9	844.17/1.8
1000	Proposed	9.72/1.9	—	—	—	—
	LM-CMA	319.63/1.9	201.08/1.9	—	—	—
4 ядра, дополнительная сложность — 0						

Продолжение таблицы 4

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
100	Proposed	0.03/1.7	0.08/1.4	—	—	—
	LM-СМА	0.02/2.8	0.28/2.7	1.43/2.6	2.61/2.5	3.74/2.5
300	Proposed	0.26/2.6	0.46/2.5	—	—	—
	LM-СМА	0.9/2.9	8.29/2.9	30.65/2.9	53.38/2.9	76.15/2.9
1000	Proposed	4.35/3.3	6.81/3.3	—	—	—
	LM-СМА	97.71/3.7	357.67/3.7	—	—	—
4 ядра дополнительная сложность — 200						
100	Proposed	0.04/3.2	0.07/2.9	—	—	—
	LM-СМА	0.04/3.5	0.64/3.6	3.36/3.5	6.1/3.5	8.82/3.5
300	Proposed	0.24/3.0	0.4/2.9	—	—	—
	LM-СМА	0.86/3.5	8.99/3.5	34.29/3.5	59.95/3.5	86.08/3.5
1000	Proposed	4.24/3.2	6.27/3.1	—	—	—
	LM-СМА	104.96/3.4	395.21/3.3	—	—	—
4 ядра, дополнительная сложность — 400						
100	Proposed	0.07/3.8	0.18/3.8	—	—	—
	LM-СМА	0.13/3.6	2.02/3.6	10.49/3.6	19.14/3.6	27.64/3.6
300	Proposed	0.41/3.7	0.63/3.5	—	—	—
	LM-СМА	2.23/3.6	24.57/3.6	93.33/3.6	161.2/3.6	228.31/3.6
1000	Proposed	5.0/3.4	—	—	—	—
	LM-СМА	134.38/3.4	495.07/3.4	—	—	—
4 ядра, дополнительная сложность — 600						
100	Proposed	0.2/3.9	0.37/3.9	—	—	—
	LM-СМА	0.36/3.7	4.45/3.6	24.19/3.6	43.76/3.6	62.9/3.6
300	Proposed	0.73/3.9	—	—	—	—
	LM-СМА	6.11/3.4	58.09/3.4	234.18/3.3	400.76/3.4	570.06/3.3
1000	Proposed	5.54/3.5	—	—	—	—
	LM-СМА	187.61/3.6	681.65/3.6	—	—	—
6 ядер дополнительная сложность — 0						
100	Proposed	0.04/1.3	0.1/1.1	—	—	—
	LM-СМА	0.02/3.5	0.3/3.1	1.45/2.7	2.61/2.8	3.77/2.8
300	Proposed	0.34/2.3	0.6/2.1	—	—	—

Продолжение таблицы 4

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
	LM-СМА	0.88/3.2	7.98/3.2	29.82/3.2	51.71/3.2	73.57/3.2
1000	Proposed	3.9/4.0	—	—	—	—
	LM-СМА	89.57/4.5	322.84/4.5	—	—	—
6 ядер, дополнительная сложность — 200						
100	Proposed	0.04/3.5	0.08/2.4	—	—	—
	LM-СМА	0.03/4.7	0.46/4.9	2.74/4.9	5.0/4.9	7.26/4.9
300	Proposed	0.3/3.3	0.2/1.1	—	—	—
	LM-СМА	0.93/4.5	8.04/4.5	31.01/4.4	53.49/4.4	76.14/4.4
1000	Proposed	3.64/4.1	—	—	—	—
	LM-СМА	79.45/4.9	289.77/4.8	—	—	—
6 ядер, дополнительная сложность — 400						
100	Proposed	0.08/5.3	0.15/5.1	—	—	—
	LM-СМА	0.11/5.1	1.42/5.2	7.73/5.1	14.54/5.1	21.31/5.1
300	Proposed	0.36/4.7	0.57/4.4	—	—	—
	LM-СМА	2.26/5.1	20.84/5.0	80.58/5.1	138.97/5.0	198.07/5.0
1000	Proposed	4.16/4.5	2.74/2.0	—	—	—
	LM-СМА	97.49/5.0	359.01/5.0	—	—	—
6 ядер, дополнительная сложность — 600						
100	Proposed	0.08/5.3	0.15/5.1	—	—	—
	LM-СМА	0.24/5.3	3.36/5.3	17.92/5.2	33.2/5.2	48.78/5.2
300	Proposed	0.36/4.7	0.57/4.4	—	—	—
	LM-СМА	4.16/5.3	39.42/5.3	146.97/5.3	251.66/5.3	361.75/5.3
1000	Proposed	4.16/4.5	2.74/2.0	—	—	—
	LM-СМА	161.67/4.6	613.68/4.6	—	—	—
8 ядер, дополнительная сложность — 0						
100	Proposed	0.04/1.3	0.1/1.0	—	—	—
	LM-СМА	0.02/2.8	0.31/2.7	1.56/2.6	2.77/2.5	3.98/2.5
300	Proposed	0.37/1.9	0.64/1.8	—	—	—
	LM-СМА	0.97/3.1	9.13/3.0	35.07/3.0	60.18/3.0	86.06/3.0
1000	Proposed	3.95/3.9	—	—	—	—
	LM-СМА	85.91/5.1	319.85/5.1	—	—	—

Продолжение таблицы 4

Размерность пространства	Алгоритм	Пороговые значения				
		3000	100	10^{-2}	10^{-6}	10^{-10}
8 ядер, дополнительная сложность — 200						
100	Proposed	0.05/3.2	0.11/2.6	—	—	—
	LM-CMA	0.03/6.0	0.41/6.0	2.13/6.0	3.89/6.0	5.64/6.0
300	Proposed	0.33/2.7	0.54/2.3	—	—	—
	LM-CMA	0.9/5.3	7.91/5.0	29.44/5.0	51.42/4.9	72.73/4.9
1000	Proposed	3.95/4.2	—	—	—	—
	LM-CMA	72.46/5.6	269.39/5.6	—	—	—
8 ядер, дополнительная сложность — 400						
100	Proposed	0.07/6.6	0.15/5.2	—	—	—
	LM-CMA	0.1/6.6	1.14/6.7	6.84/6.6	12.26/6.5	17.53/6.5
300	Proposed	0.36/5.0	0.57/3.4	—	—	—
	LM-CMA	1.65/6.1	15.71/6.2	60.05/6.2	105.71/6.2	150.66/6.2
1000	Proposed	3.72/5.0	—	—	—	—
	LM-CMA	82.97/6.0	316.75/5.9	—	—	—
8 ядер, дополнительная сложность — 600						
100	Proposed	0.14/7.3	0.22/6.9	—	—	—
	LM-CMA	0.22/7.0	3.75/6.9	18.11/6.8	33.17/6.8	48.28/6.8
300	Proposed	0.54/7.1	0.6/5.4	—	—	—
	LM-CMA	3.26/6.6	28.07/6.7	109.14/6.7	193.33/6.7	277.51/6.7
1000	Proposed	4.28/6.1	—	—	—	—
	LM-CMA	141.83/5.4	546.15/5.3	—	—	—