https://doi.org/10.25559/SITITO.021.202501.113-126
УДК 004.42, 519.6

Оригинальная статья

Обзор программных средств для работы с эволюционными и роевыми методами оптимизации

А. Г. Николашкин*, **Н. М. Ершов**
ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова», г. Москва, Российская Федерация
Адрес: 119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1
* nagvv97@mail.ru

Аннотация
Настоящая статья посвящена обзору программных средств, позволяющих применять, разрабатывать и исследовать эволюционные и роевые методы оптимизации для решения сложных задач дискретной и непрерывной оптимизации. В статье рассматриваются различные виды и типы оптимизационных задач, возникающих в прикладных задачах, включая задачи многокритериальной оптимизации. Формализуется понятие популяционного алгоритма оптимизации и рассматриваются основные классы алгоритмов данного типа, в том числе эволюционные алгоритмы, роевые алгоритмы и многочастичные алгоритмы. Приводятся результаты детального анализа тринадцати современных наиболее популярных фреймворков для работы с эволюционными и роевыми алгоритмами оптимизации. Основной целью анализа является исследование предоставляемых данными программными средствами возможностей по созданию, настройке и применению популяционных алгоритмов оптимизации для решения прикладных оптимизационных задач. В том числе анализируются типы поддерживаемых рассматриваемыми программными средствами задач оптимизации, а также наличие встроенных средств тестирования и наборов тестовых задач оптимизации. Отдельный пункт исследования посвящен анализу поддержки рассматриваемыми фреймворками параллельных вычислений, так как известно, что применение популяционных алгоритмов с одной стороны является вычислительно затратным, а с другой стороны такие алгоритмы обладают значительным потенциалом к распараллеливанию. На основании результатов проведенного обзора даются рекомендации по применению рассмотренных программных средств в различных сценариях их практического использования.

Ключевые слова: эволюционные алгоритмы, роевые алгоритмы, оптимизация, дискретная оптимизация, пакеты программ, фреймворки

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Николашкин А. Г., Ершов Н. М. Обзор программных средств для работы с эволюционными и роевыми методами оптимизации // Современные информационные технологии и ИТ-образование. 2025. Т. 21, № 1. С. 113-126. https://doi.org/10.25559/SITITO.021.202501.113-126

Original article

Review of Software Tools for Working with Evolutionary and Swarm Optimization Methods

A. G. Nikolashkin*, N. M. Ershov
Lomonosov Moscow State University, Moscow, Russian Federation
Address: 1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation
* nagvv97@mail.ru

Abstract
The article is devoted to a review of software tools that allow applying, developing and investigating evolutionary and swarm optimization methods for solving complex discrete and continuous optimization problems. The article considers various types and kinds of optimization problems arising in applied problems, including multi-objective optimization problems. The concept of a population optimization algorithm is formalized and the main classes of algorithms of this type are considered, including evolutionary algorithms, swarm algorithms and multiparticle algorithms. The results of a detailed analysis of thirteen modern most popular frameworks for working with evolutionary and swarm optimization algorithms are presented. The main goal of the analysis is to study the capabilities provided by these software tools for creating, customizing and using population optimization algorithms for solving applied optimization problems. In particular, the types of optimization problems supported by the considered software tools, as well as the presence of built-in testing tools and sets of test optimization problems are analyzed. A special section of the study is devoted to the analysis of support of parallel computing by frameworks under consideration, since it is known that the use of population algorithms, on the one hand, is computationally expensive, and on the other hand, such algorithms have significant potential for parallelization. Based on the results of the review, recommendations are given on the use of the considered software tools in various scenarios of their practical use.

Keywords: evolutionary algorithms, swarm algorithms, optimization, discrete optimization, software packages, frameworks

Conflict of interests: The authors declare no conflict of interest.

For citation: Nikolashkin A.G., Ershov N.M. Review of Software Tools for Working with Evolutionary and Swarm Optimization Methods. *Modern Information Technologies and IT-Education.* 2025;21(1):113-126. https://doi.org/10.25559/SITITO.021.202501.113-126

Введение

Целью настоящей работы является проведение обзора и анализа существующих программных средств (систем, библиотек, пакетов и фреймворков), позволяющих применять методы популяционной оптимизации для решения различных задач дискретной и непрерывной многомерной оптимизации. Под методами популяционной оптимизации понимаются, прежде всего, два основных класса оптимизационных алгоритмов, основанных на биологических принципах и аналогиях: эволюционные алгоритмы (генетические алгоритмы, методы дифференциальной эволюции [1]) и роевые алгоритмы (метод роя частиц [2], алгоритмы бактериального поиска [3] и т. д.). К популяционным алгоритмам могут быть отнесены и так называемые многочастичные методы оптимизации, например метод имитации отжига [4], которые построены на аналогии уже с физическими, а не биологическими системами. Объединяет все эти методы общая идея *коллективного поиска* глобального экстремума заданной целевой функции с помощью популяции искусственных организмов, частиц и т. д.

Преимуществом такого коллективного подхода к решению оптимизационных задач является то, что спектр решаемых с его помощью задач оказывается практически неограниченным. Например, популяционные алгоритмы с успехом применяются для решения задач дискретной оптимизации [5-7], задач непрерывной оптимизации в условиях недифференцируемой [8, 9] или зашумленной целевой функции [7], [10, 11], задач условной [12, 13] и многокритериальной оптимизации [14] и т.д. Для каждой конкретной задачи во многих случаях, скорее всего, можно придумать более эффективный приближенный метод ее решения, однако, популяционные алгоритмы предлагают готовую схему решения, в которой единственное, что остается сделать пользователю, это настроить числовые параметры выбранного алгоритма и, в особых случаях, адаптировать его операторы под решаемую задачу.

В настоящее время разработаны и применяются на практике десятки алгоритмов популяционной оптимизации и сотни их различных вариаций[1] [15]. С этой точки зрения интересным оказывается разработка программного комплекса, позволяющего упорядочить, стандартизировать и унифицировать работу с такими алгоритмами, чтобы пользователь мог, в частности, легко и безболезненно заменять один алгоритм на другой, выполнять настройку алгоритма, в том числе и автоматическую, для более эффективного решения заданной задачи оптимизации, выполнять автоматическое тестирование и сравнение алгоритмов на решении тестовых задач и т.д.

Ключевым вопросом при использовании популяционных алгоритмов оптимизации для решения задач высокой размерности является применение параллельной обработки данных, что обусловлено высокой вычислительной сложностью данного класса алгоритмов. С другой стороны, в силу особенности устройства подобных алгоритмов (*коллективный* поиск решения) они в большинстве своем обладают значительным потенциалом к распараллеливанию с использованием практически всех технологий параллельного программирования [16, 17]. Поэтому вопрос поддержки параллельного исполнения алгоритмов является важным пунктом для сравнения существующих программных систем в рассматриваемой области.

Стоит упомянуть, что обзоры программных средств производились и ранее. В работе [18] приводится подробный сравнительный анализ 10 различных фреймворков. Однако только менее половины рассмотренных в этой статье фреймворков поддерживаются и являются актуальными в настоящее время. Также существует обзор [19], в котором программные средства рассматриваются исключительно с точки зрения решения задач многокритериальной оптимизации. В отличие от указанных работ, настоящая статья рассматривает более широкий набор актуальных программных средств, не ограничиваясь одним определенным типом задач оптимизации.

Настоящая статья устроена следующим образом. Во второй ее секции определяется формальная постановка задачи оптимизации, рассматриваются различные виды и типы оптимизационных задач. В третьей секции формализуется понятие популяционного алгоритма оптимизации и рассматриваются основные классы алгоритмов данного типа. Четвертая секция посвящена собственно обзору программных средств по работе с популяционными алгоритмами оптимизации, отдельно рассматриваются возможности данных программных систем по настройке алгоритмов, типы поддерживаемых ими задач оптимизации, наличие средств тестирования, поддержка параллельных вычислений. В последней секции формулируются результаты проведенного обзора.

1. Задачи оптимизации

Базовая задача оптимизации в своей наиболее общей постановке формулируется следующим образом: 1) задана *целевая функция* $f: A \to \mathbb{R}$, ставящая в соответствие каждому элементу x некоторого множества A (пространство решений) вещественное число $f(x)$; 2) требуется найти такой элемент $x_0 \in A$, чтобы для $\forall x \in A$ выполнялось условие $f(x_0) \le f(x)$ (задача минимизации) или $f(x_0) \ge f(x)$ (задача максимизации). Заметим, что задача оптимизации может иметь несколько решений (глобальных экстремумов), соответственно, целью оптимизации может быть либо один (произвольный) глобальный экстремум, либо сразу все такие экстремумы. Исходя из этой общей постановки возникают различные вариации оптимизационных задач (рис. 1).

Если множество A является подмножеством конечномерного евклидова пространства $\mathbb{R}^n$, то задача оптимизации называется *непрерывной*. Если целевая функция дифференцируема на A, то для поиска глобального экстремума можно использовать различные методы типа градиентного спуска. Однако, в случае недифференцируемой или даже разрывной целевой функции методы данного класса оказываются, как правило, неприменимыми.

Если множество A дискретно (например, множество двоичных строк заданной длины, множество графов заданного размера), то задача оптимизации на таком множестве называется задачей *дискретной оптимизации*. Традиционно, задачи дис-

[1] Campelo F., Aranha C. EC Bestiary: A bestiary of evolutionary, swarm and other metaphor-based algorithms [Электронный ресурс] // Zenodo. June 20, 2018. https://doi.org/10.5281/zenodo.1293352

кретной оптимизации являются существенно более сложными, чем задачи непрерывной оптимизации. Многие из них, являющиеся практически важными (задача коммивояжера, задача об укладке рюкзака), принадлежат классу NP-сложных задач, для решения которых в настоящее время не существует точных алгоритмов с полиномиальным временем работы. В прикладных задачах пространство решений может быть и *смешанным* [20-22], в этом случае часть решения описывается дискретными переменными, часть – непрерывными.

Р и с. 1. Типы и классы оптимизационных задач

F i g. 1. Types and classes of optimization problems

Источник: здесь и далее в статье все таблицы и рисунки составлены авторами.

Source: Hereinafter in this article all tables and figures were made by the authors.

Важным параметром задачи оптимизации является ее *размерность*, под которой понимается число параметров, описывающих каждое решение. В общем случае сложность решения задачи резко возрастает с ростом ее размерности – проклятие высокой размерности[2]. В простейшем случае размерность задачи фиксирована. В более сложных задачах размерность может быть переменной, например, в задаче поиска кратчайшего пути в графе G размера n пространством решений являются все простые (без самопересечений) пути в графе G между двумя заданными вершинами, длины которых могут потенциально меняться от 1 до $n - 1$.

Во многих задачах оптимизации пространство решений, являющееся подмножеством некоторого «регулярного» множества (например, множества $\mathbb{R}^n$), не может быть задано явным образом, а определяется через дополнительные условия (ограничения). Такие задачи называются задачами *условной оптимизации*. В некоторых случаях задачи условной оптимизации могут быть сведены к стандартной постановке модификацией целевой функции, к которой добавляются штрафные слагаемые, существенно ухудшающие ее значения при нарушении имеющихся в задаче ограничений.

В практических приложениях часто приходится решать задачи оптимизации, в которых целевая функция является зашумленной [23-25], например, если ее вычисление связано с измерением каких-то физических параметров. Такие задачи называются задачами *стохастической оптимизации*. Очевидно, что применяемые для решения такого рода задач алгоритмы должны быть устойчивы к наличию шума в целевой функции.

Еще один важный класс задач оптимизации – задачи *динамической оптимизации*, в которых целевая функция зависит, некоторым образом, от времени t: $f(x, t)$. Если данная задача является задачей реального времени, то на ее решение обычно накладываются жесткие временные ограничения. То есть мы не можем решать для каждого t соответствующую задачу оптимизации «с нуля», необходимо как-то учитывать решения, найденные в предыдущие моменты времени, модифицируя их под изменяющуюся целевую функцию. Популяционные алгоритмы, как правило, хорошо справляются с такими задачами.

Естественным обобщением стандартной задачи оптимизации являются задачи *многокритериальной оптимизации* [26]. В таких задачах нужно оптимизировать не одну целевую функцию, а сразу несколько таких функций. Очевидно, что критерии оптимальности, определяемые данными функциями, в общем случае могут противоречить друг другу, что означает отсутствие единого решения рассматриваемой задачи, которое бы давало экстремум по всем целевым функциям сразу. В таких ситуациях либо заменяют набор критериев их взвешенной суммой, либо говорят об оптимальности по Парето. В последнем случае решений оказывается бесконечно много (множество Парето), нахождение такого множества является непростой задачей, особенно в случае дискретной оптимизации.

2. Популяционные методы оптимизации

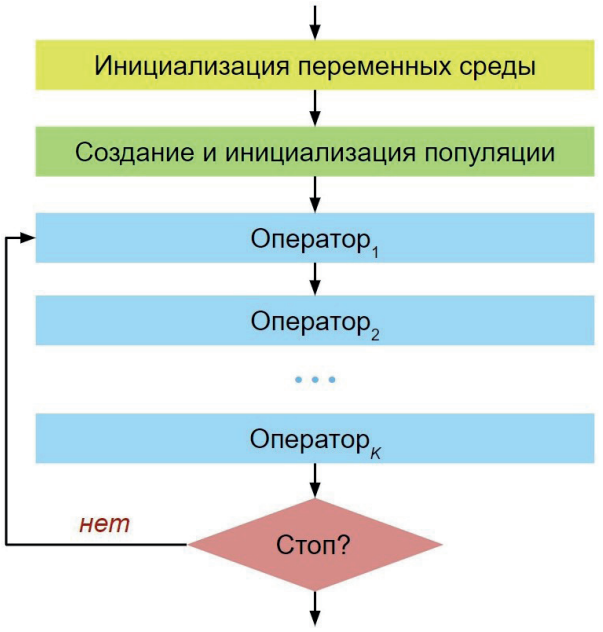
Популяционные методы оптимизации относятся к классу приближенных эвристических алгоритмов, работа которых заключается в коллективном поиске глобального экстремума заданной целевой функции. Практически все алгоритмы данного типа основаны на моделировании коллективного поведения различных природных, прежде всего биологических, систем по одной и той же достаточно общей схеме (рис. 2):

1. создается *популяция* пробных решений рассматриваемой задачи, эти решения мы далее иногда будем называть *особями* популяции;
2. инициализируются глобальные переменные, которые будем называть *переменными среды*;
3. организуется *основной цикл*, итерации которого соответствуют поколениям алгоритма (эволюционные алгоритмы) или дискретному времени внутри данной популяции (роевые алгоритмы, многочастичные алгоритмы);
4. на каждой итерации основного цикла к популяции последовательно применяются несколько *операторов*, изменяющих состояния особей и значения переменных среды;
5. основной цикл завершается при выполнении некоторого заданного условия останова: достижение максимального числа итераций, достижение максимального числа вычислений целевой функции, достижение решения с заданной точностью.

Упомянутая во введении классификация популяционных алгоритмов на три основных класса – эволюционные, роевые и многочастичные алгоритмы, обусловлена прежде всего «происхождением» каждого конкретного алгоритма, а не его внутренним устройством. Более того, некоторые алгоритмы занимают промежуточное положение в этой классификации, например алгоритм бактериального поиска несет черты как роевого, так и эволюционного алгоритма.

[2] Bellman R. E. Adaptive Control Processes: A Guided Tour. Princeton, New Jersey: Princeton University Press, 1961. 255 p.

Эволюционные алгоритмы построены на идее естественного отбора – выживание в популяции только наиболее приспособленных особей. Критерием приспособленности выступает целевая функция, чем она лучше, тем больше приспособленность данной особи. Выжившие при отборе особи размножаются, передавая свои положительные свойства потомкам. Итерации эволюционного алгоритма соответствуют поколениям популяции. Примерами эволюционных алгоритмов являются: генетические алгоритмы, метод дифференциальной эволюции [1], алгоритм клональной селекции [27].



Р и с. 2. Структура популяционного алгоритма
F i g. 2. Structure of the population algorithm

Роевые алгоритмы моделируют коллективное поведение колонии некоторых живых организмов в течение одного жизненного цикла популяции, то есть обычно в процессе выполнения алгоритмов данного типа особи не умирают и не рождаются. Среди операторов роевого алгоритма оптимизации должен быть хотя бы один оператор, моделирующий взаимодействие особей, при выполнении которого особи обмениваются информацией о своих состояниях и координируют свои дальнейшие действия. В основе большинства таких операторов лежит принцип выравнивания Рейнолдса [28], который в обобщенной форме звучит так: делай то, что делают твои соседи. Применительно к роевым алгоритмам это правило модифицируется: делай то, что делают твои соседи, обладающие лучшей целевой функцией, чем у тебя (и соответственно, не делай того, что делают твои соседи, которые хуже тебя). Примерами роевых алгоритмов являются: метод роя частиц [2], муравьиные алгоритмы [29], алгоритм бактериального поиска [3], имеющий черты эволюционного алгоритма из-за применения оператора отбора, алгоритм пчелиного поиска [30], светлячковый алгоритм [31], конкурентный роевой оптимизатор [32].

Многочастичные алгоритмы, как правило, моделируют поведение физических систем, состоящих из большого числа частиц. В качестве целевой функции в этом случае может выступать, например, потенциальная энергия рассматриваемой системы частиц. Примерами многочастичных алгоритмов являются: метод имитации отжига [4], алгоритм гравитационного поиска [33], алгоритм формирования рек [34]. Заметим, что метод имитации отжига, по-видимому, является единственным популяционным алгоритмом, в котором в принципе отсутствует какое-либо взаимодействие между особями популяции.

Разные алгоритмы отличаются друг от друга тем, что применяют разные наборы операторов для обновления популяции. При этом число принципиально разных типов операторов относительно невелико. Например, стандартный генетический оператор мутации широко применяется под разными названиями во многих других популяционных алгоритмах.

3. Обзор программных систем

3.1. Цели обзора

Целью настоящей работы является обзор актуальных программных средств метаэвристической оптимизации, которые далее мы обобщенно будем называть *фреймворками*, хотя многие из них частично несут также свойства и библиотек. Четыре важных для пользователя аспекта, которые он должен учитывать при выборе подходящего фреймворка для решения своего класса задач и которые будут проанализированы ниже – это:

- алгоритмическая наполненность, в том числе, возможность глубокой модернизации поддерживаемых фреймворков алгоритмов вплоть до создания абсолютно новых алгоритмов с использованием предоставляемой фреймворком инфраструктуры;
- универсальность, т. е. способность решать оптимизационные задачи в широком диапазоне их типов и параметров;
- наличие средств автоматического тестирования, которые могут использоваться, в том числе и для подбора или настройки наиболее эффективных алгоритмов решения пользовательских задач.
- возможность ускорения вычислений за счет их распараллеливания;

Заметим, что за последние десятилетия было представлено множество фреймворков метаэвристической оптимизации, многие из которых в настоящее время уже не поддерживаются разработчиками или сообществом пользователей. Если проект не поддерживается, то со временем теряется доступ к документации и другим материалам, он может стать несовместимым с современными инструментами разработки, бинарные сборки могут перестать работать в новых операционных системах, не будут исправляться найденные ошибки в коде, а также исчезает сообщество пользователей, от которой можно получить различную поддержку и ответы на вопросы. Кроме того, устаревший проект скорее всего будет неактуальным для потенциального пользователя. В результате было принято решение исключить из обзора проекты, у которых не было обновлений в последние пять лет.

Другими критериями для включения программных систем в настоящий обзор были следующие:

- использование стандартного языка программирования: использование таких языков разрешает большую гибкость и

позволяет выполнять интеграцию в конечное пользовательское приложение;

- открытость кода: для обзора подбирались непроприетарные фреймворки, которые доступны всем для получения и использования;
- поддержка широкого набора оптимизационных техник: исключались проекты, которые поддерживают только один тип метаэвристических алгоритмов, например, были исключены из рассмотрения фреймворки Jenetics[3] и pyswarms [35], которые поддерживают только генетический алгоритм и метод роя частиц соответственно;
- наличие возможности ускорения вычислений за счет их распараллеливания, что является важным критерием практической значимости заданного фреймворка.

В итоге в настоящий обзор были включены 13 фреймворков, общая информация о которых приведена в таблице 1. Если в поле «Язык» указаны два языка, то первый – это язык, на котором написан фреймворк, а второй – язык для работы пользователя с этим фреймворком. В поле «Даты» указаны даты выхода фреймворка и последнего его обновления.

Таблица 1. **Перечень фреймворков, вошедших в обзор**

Table 1. **List of frameworks included in the review**

Фреймворк	Язык	Даты	Версия	Лицензия
Pagmo2 [36]	C++	2010/2023	2.19.0	GPL3/LGPL3
DEAP [37]	Python	2010/2023	1.4.1	LGPL3
Pymoo [38]	Python	2018/2023	0.6.0.1	Apache2
Jmetal [39]	Java	2006/2023	6.2.2	MIT
Inspyred[4]	Python	2012/2023	1.0.2	MIT
MOEA Framework[5]	Java	2011/2024	3.10	LGPL3
platypus[6]	Python	2018/2024	1.2.0	GPL3
LEAP [40]	Python	2020/2023	0.8.1	AFL3
Paradise [41]	C++	1999/2023	3.0.0	LGPL v2.1
EASEA [42]	C++/ez (C-like)	2000/2022	3.1.1	AGPL3
HeuristicLab [43]	C#	2014/2022	3.3.16	GPL3
Opt4j [44]	Java	2011/2021	3.3.0	MIT
ECJ[7]	Java	1999/2019	27	AFL3

3.2. Поддержка алгоритмов

Важными свойствами, которые потенциально могут влиять на выбор пользователем фреймворка для решения поставленной им оптимизационной задачи (класса задач), являются полнота и расширяемость. Под *полнотой* фреймворка мы будем понимать, прежде всего, число поддерживаемых им метаэвристик, таких как генетические алгоритмы (всех видов), метод роя частиц и т.д. Другим аспектом полноты, важным для пользователей, не являющихся профессионалами в области программирования или методов оптимизации, является количество реализованных во фреймворке алгоритмов, которые можно применять для решения пользовательских задач напрямую, без какой-либо их настройки.

Под *расширяемостью* фреймворка будем понимать предоставляемые им возможности по глубокой настройке и модернизации поддерживаемых алгоритмов, вплоть до разработки средствами фреймворка абсолютно новых метаэвристик. Данное свойство может быть важным для пользователей, решающих сложные нестандартные задачи оптимизации, либо для исследователей, разрабатывающих новые подходы к решению оптимизационных задач.

Косвенно указанные свойства определяются используемыми фреймворками моделями построения алгоритмов. Обобщенно эти модели можно разделить на два больших класса, которые условно назовем монолитными и конструктивными. *Монолитная* модель подразумевает встроенную во фреймворк реализацию некоторого набора алгоритмов, изменение которых пользователем возможно только частично: пользователь может настраивать числовые параметры алгоритма, перегружать операторы собственными функциями, добавлять/удалять операторы алгоритма.

В *конструктивной* модели алгоритмы собираются пользователем по некоторому шаблону из предоставляемых фреймворком или самостоятельно написанных пользователем элементов (функций, классов). В частности, так устроены и все алгоритмы, поддерживаемые фреймворками данного типа. У пользователя, соответственно, в данном случае имеется полная свобода в конструировании любых алгоритмов, как стандартных для фреймворка, так и абсолютно новых.

[3] Wilhelmstötter F. Jenetics: Java Genetic Algorithm Library. 2025 [Электронный ресурс]. URL: https://jenetics.io (дата обращения: 17.01.2025).

[4] Garrett A. Inspyred [Электронный ресурс] // GitHub, 2025. URL: https://github.com/aarongarrett/inspyred (дата обращения: 17.01.2025).

[5] Hadka D. MOEA Framework A Free and Open Source Java Framework for Multiobjective Optimization (Version 5.1). 2025 [Электронный ресурс]. URL: https://moea-framework.org/ (дата обращения: 17.01.2025).

[6] Hadka D. Platypus [Электронный ресурс] // GitHub, 2025. URL: https://github.com/Project-Platypus/Platypus (дата обращения: 17.01.2025).

[7] Luke S. ECJ evolutionary computation library [Электронный ресурс]. URL: https://cs.gmu.edu/~eclab/projects/ecj/ (дата обращения: 17.01.2025).

Таблица 2. **Поддержка фреймворками алгоритмов оптимизации**
Table 2. **Framework support for optimization algorithms**

Фреймворк	Эвол.	Роев.	Многочаст.	Модель
Pagmo2	8	5	1	М--+
DEAP	7	1	0	К
Pymoo	8	1	0	М-+-
Jmetal	4	2	1	М+-+
Inspyred	4	2	1	К
MOEA Framework	5	0	1	М+++
Platypus	2	0	0	М---
LEAP	2	0	0	К
Paradise	6	1	1	К
EASEA	6	0	0	К
HeuristicLab	11	1	1	М--+
Opt4j	2	0	1	М+-+
ECJ	9	2	2	К

В таблице 2 приведены результаты анализа фреймворков из списка, представленного в таблице 1, с точки зрения поддержки ими алгоритмов оптимизации. В первых трех столбцах таблицы указаны количество поддерживаемых фреймворком алгоритмов в каждом из трех классов метаэвристик: эволюционные алгоритмы, роевые алгоритмы и многочастичные алгоритмы.

В последнем столбце таблицы указана используемая фреймворком модель построения алгоритмов – монолитная (М) или конструктивная (К). Если модель монолитная, то плюсами и минусами показаны предоставляемые фреймворком возможности пользователя по модернизации алгоритмов, плюс – соответствующая возможность поддерживается, минус – не поддерживается: а) перегрузка операторов; б) модификация структуры алгоритма; в) построение новых алгоритмов с использованием встроенных средств фреймворка.

Отметим, что параметрическая настройка алгоритмов доступна во всех рассматриваемых фреймворках.

3.3. Задачи оптимизации

Был проведен анализ фреймворков с точки зрения поддержки ими классов задач оптимизации, которые были разбиты на четыре категории: классическая однокритериальная оптимизация (поддерживается всеми фреймворками); условная оптимизация; многоэкстремальная оптимизация (цель которой – поиск сразу нескольких решений); многокритериальная оптимизация (также поддерживаются всеми рассматриваемыми фреймворками). Кроме того, исследовалось еще одно важное свойство – типы пространств решений, поддерживаемых хотя бы одним алгоритмом заданного фреймворка.

Таблица 3. **Поддержка фреймворками пространств решений**
Table 3. **Framework support for solution spaces**

Фреймворк	Вект.	Комб.	Деревья	Смеш.	Пользов.	Адапт.
Pagmo2	IF	–	–	–	–	–
DEAP	BIF	P	●	–	●	–
Pymoo	BIF	P	–	●	●	–
Jmetal	BIF	P	–	●	●	–
Inspyred	BIF	P	–	–	●	–
MOEA Framework	BIF	PS	●	●	●	–
Platypus	BIF	PS	–	–	●	–
LEAP	BIF	–	–	–	●	●
Paradise	BF	P	●	–	●	–
EASEA	F	P	●	–	●	–
HeuristicLab	BIF	P	●	–	●	–
Opt4j	BIF	P	–	●	●	●
ECJ	BIF	P	●	–	●	–

Информация о поддерживаемых фреймворками пространствах решений показана в таблице 3. Первый столбец соответствует векторным пространствам фиксированной (векторы) или переменной (списки) длины, символы B, I и F означают двоичные, целочисленные и вещественные пространства. Второй столбец соответствует комбинаторным пространствам, символ P означает перестановки, символ S – подмножества. В следующих трех столбцах отражена поддержка фреймворками пространств деревьев, смешанных (композитных) и пользовательских пространств решений. Последний столбец таблицы показывает возможность применения адаптеров (или декодеров) задач, позволяющих прозрачным для пользователя способом перекодировать решение из внутреннего пространства решений, применяемого заданным алгоритмом (*генотип* в терминах генетических алгоритмах), в пользовательское (*фенотип* в терминах тех же генетических алгоритмов). Применение адаптеров позволяет, например, использовать алгоритмы непрерывной оптимизации, такие как метод роя частиц, для решения задач дискретной оптимизации.

В таблице 4 приведены результаты анализа рассматриваемых фреймворков на предмет их поддержки разных типов задач оптимизации. Классическая безусловная однокритериальная оптимизация поддерживается всеми фреймворками, поэтому в таблицу этот тип задач не включен. Первый столбец показывает возможности фреймворка по работе с задачами условной оптимизации: одиночный плюс означает, что фреймворк позволяет задавать ограничения функционального вида (например, $g(x) > 0$); двойной плюс означает, что у пользователя имеется возможность вводить штрафные функции; тройной плюс означает наличие дополнительного функционала при работе с условной оптимизацией.

Таблица 4. **Поддержка типов задач оптимизации**
Table 4. **Support for optimization task types**

Фреймворк	Условная	Многоэкстр.	Многокритер.
Pagmo2	+++	–	4
DEAP	++	–	4
Pymoo	+++	–	13
Jmetal	+	–	29
Inspyred	–	●	2
MOEA Framework	+	–	39
Platypus	+	–	13
LEAP	–	–	1
Paradise	–	●	8
EASEA	–	–	8
HeuristicLab	–	–	2
Opt4j	+	–	5
ECJ	–	–	3

Второй столбец таблицы показывает, поддерживается ли фреймворком решение задачи многоэкстремальной оптимизации, то есть одновременный поиск сразу нескольких решений с использованием нишевых стратегий [45]. Из рассмотренных фреймворком данным свойством обладают только фреймворки *Inspyred* и *paradiseo*, первый из них использует стратегию *crowding*, второй – *fitness sharing*.

В последнем столбце таблицы 4 показан уровень поддержки многокритериальных задач оптимизации. Числа в этом столбце соответствуют количеству реализованных во фреймворке алгоритмов (или их вариаций), предназначенных для решения задач многокритериальной оптимизации.

3.4. Средства тестирования

Тестирование популяционных алгоритмов с целью оценивания их корректности и эффективности решения различных оптимизационных задач является необходимым этапом их разработки и применения. С этой точки зрения важными для пользователя могут оказаться два свойства фреймворка: 1) поддержка им готовых тестовых задач (или тестовых наборов – бенчмарков), оформленных в соответствии со стандартами данного фреймворка; 2) наличие средств автоматического тестирования заданного набора алгоритмов на заданном наборе задач с последующим оформлением результатов тестирования, в том числе и их визуализацией. Результаты анализа рассматриваемых фреймворков на предмет поддержки ими двух указанных свойств приведены в таблице 5.

В первом столбце таблицы показано количество реализованных во фреймворке тестовых задач (однокритериальных / многокритериальных). Во втором столбце таблицы показано количество поддерживаемых фреймворков стандартных тестовых наборов (однокритериальной / многокритериальной оптимизации), таких как CEC2018[8] [46], ZDT [47], DTLZ [48] и т.п. Два фреймворка из рассматриваемого списка имеют встро-

[8] Benchmark Problems for CEC2018 Competition on Dynamic Multiobjective Optimisation / S. Jiang [et al.] // 2018 IEEE Congress on Evolutionary Computation (CEC 2018). Rio de Janeiro, Brazil, July 8-13, 2018 [Электронный ресурс]. URL: http://homepages.cs.ncl.ac.uk/shouyong.jiang/cec2018/CEC2018_Tech_Rep_DMOP.pdf (дата обращения: 17.01.2025).

енные средства для работы с внешними бенчмарками (третий столбец): фреймворк *pymoo* может работать с тестовыми задачами, реализованными в пакете SciPy [49], фреймворк *paradiseo* – с тестирующей системой IOHexperimenter [50]. Наконец, наличие тестирующей подсистемы отражено в последнем столбце таблицы. Как видно, только четыре фреймворка поддерживают подсистемы автоматического тестирования алгоритмов, в остальных фреймворках тестирование и постобработка его результатов ложится на плечи пользователя.

Таблица 5. **Средства тестирования**
Table 5. **Testing tools**

Фреймворк	Задачи	Наборы	Внешние наборы	Тест. Система
Pagmo2	11 / 0	3 / 4	–	–
DEAP	30 / 12	0 / 0	–	–
Pymoo	11 / 8	2 / 8	●	–
Jmetal	9 / 18	1 / 15	–	●
Inspyred	8 / 1	0 / 1	–	–
MOEA Framework	9 / 31	0 / 9	–	●
Platypus	0 / 0	0 / 0	–	●
LEAP	22 / 1	0 / 1	–	–
Paradise	11 / 0	0 / 1	●	–
EASEA	10 / 2	1 / 4	–	–
HeuristicLab	39 / 6	0 / 3	–	●
Opt4j	2 / 1	0 / 3	–	–
ECJ	38 / 8	1 / 1	–	–

3.5. Поддержка параллельных вычислений

Имеются как минимум две стандартные схемы распараллеливания, применимые ко всем популяционным алгоритмам без исключения: схема с параллельным вычислением целевой функции и островная схема. Первая схема актуальна для задач с «тяжелыми» целевыми функциями, когда практически все время работы алгоритма тратится на вычисления значений этих функций. Примерами таких задач являются задачи машинного обучения [51], в которых даже для скромных по размеру искусственных нейронных сетей вычисление функции потерь оказывается весьма значительным, и задачи биоинформатики [52], такие как задача предсказания трехмерной структуры белковых молекул, в которой в качестве целевой функции оптимизации выступает, например, потенциальная энергия межатомных связей. Распараллеливание такого рода задач сводится к параллельному вычислению значений целевой функции в точках, соответствующих особям популяции. Стандартным способом такого распараллеливания является схема *master-slave*, когда один главный процесс *(master)* отвечает за логику выполнения алгоритма, а все остальные *(slaves)* заняты только вычислениями целевой функции для решений, предоставляемых им главным процессом. Теоретически, ускорение в такой схеме выполнения может быть близким к размеру популяции.

Островная схема по сути является стандартным методом преобразования заданного популяционного алгоритма в так называемую островную модель [53]: вся популяция делится на несколько субпопуляций (островов), для каждой субпопуляции запускается свой алгоритм оптимизации, с некоторой периодичностью субпопуляции обмениваются своими решениями, используя оператор миграции. Хотя эта схема и была придумана изначально в качестве схемы именно распараллеливания, оказалось, что она позволяет выполнять и более качественный поиск решения за счет замедления процесса потери популяционного разнообразия, свойственного практически всем эволюционным и роевым алгоритмам оптимизации. При параллельной реализации островной модели с низкой частотой миграции теоретическое ускорение может быть сколь угодно близким к числу островов.

Информация о поддержке фреймворками указанных двух схем параллельного выполнения приведена в таблице 6: столбец ЦФ соответствует схеме параллельного вычисления целевой функции; столбец ОС – островной схеме. Знак минус означает отсутствие поддержки соответствующей схемы, незакрашенный кружок – наличие такой поддержки, закрашенный кружок – наличие поддержки и возможность использования пользовательской параллельной реализации данной схемы. Как видно, все рассматриваемые фреймворки поддерживают параллельное вычисление целевой функции, часть фреймворков – параллельную реализацию островной модели. Заметим, что островная модель, но без параллельного ее выполнения, поддерживается также фреймворком LEAP, а во фреймворке HeuristicLab имеется реализация островной модели генетического алгоритма как отдельной метаэвристики.

В последнем столбце таблицы 6 указаны технологии параллельного программирования, применяемые в рассматриваемых фреймворках для реализации двух описанных выше схем. Эти технологии можно разбить на две группы по их возможности запускаться на нескольких машинах. В первую группу, позволяющую параллельную работу фреймворка на нескольких машинах, входят SCOOP, Dask, Parallel Python, Apache Ignite и MPI. Первые три являются пакетами для языка Python, позволяющие распределять вычисления по рабочим процессам, Apache Ignite – это СУБД, в которой реализован интерфейс Java ExecutorService, MPI является стандартным средством парал-

лелизации по процессам, взаимодействующих через передачу сообщений. Во вторую группу входят Intel TBB, multiprocessing, ForkJoinPool, ThreadPoolExecutor, std::thread, OpenMP, .NET TPL и CUDA. Все перечисленные, кроме *multiprocessing* и CUDA, являются средствами для *многопоточной* параллелизации, *multiprocessing* – это стандартный пакет языка Python для паралеллизации по процессам, CUDA – технология для ускорения вычислений на графических процессорах NVidia. Отдельного упоминания стоят не перечисленные в этих группах HiveEngine, processes via socket и ThreadPool. Они не являются независимыми средствами распараллеливания вычислений, а представлены только в рамках соответствующих фреймворков. Под processes via socket представлены случаи, когда в фреймворке напрямую используются сокеты для коммуникации между процессами, HiveEngine – это интегрированная в HeuristicLab средство для распределения вычислений по рабочим процессам, ThreadPool – это внутренняя реализация пула потоков во фреймворке ECJ.

Таблица 6. **Поддержка параллельных вычислений**
Table 6. **Parallel computing support**

Фреймворк	ЦФ	ОС	Технология
Pagmo2	●	●	Intel TBB[9]
DEAP	●	●	SCOOP [54], multiprocessing
Pymoo	●	–	Dask [55], multiprocessing
Jmetal	●	–	ForkJoinPool
Inspyred	●	●	Parallel Python[10], multiprocessing
MOEA Framework	●	●	ThreadPoolExecutor, Apache Ignite[11]
Platypus	●	–	mpi4py [56], multiprocessing
LEAP	○	–	Dask
Paradise	○	○	std::thread, OpenMP, MPI
EASEA	○	○	OpenMP, CUDA, processes via socket
HeuristicLab	○	–	.NET TPL, HiveEngine
Opt4j	●	–	ThreadPoolExecutor
ECJ	○	○	ThreadPool, processes via socket

Заключение

Анализ представленных выше таблиц показывает, что ни один из рассмотренных фреймворков не является универсальным инструментом, каждый из них обладает как плюсами, так и минусами. С этой точки зрения более полезным будет рассмотреть соответствие данных фреймворков частным сценариям их применения. Традиционно выделяется три таких сценария: индустриальный, исследовательский и образовательный.

Индустриальный сценарий ориентирован на решение прикладных оптимизационных задач, этот сценарий можно разбить на два более конкретных – простой и продвинутый. Простой сценарий подразумевает либо многократное решение простых задач оптимизации, для которых не требуется использования продвинутых алгоритмов, либо однократное решение умеренно сложных задач. В обоих случаях для пользователя достаточно будет, чтобы фреймворк поддерживал какое-то количество готовых к использованию алгоритмов с возможностью их простой (параметрической) настройки, а также мог бы работать с различными типами оптимизационных задач. Продвинутый индустриальный сценарий соответствует решению сложных задач оптимизации, для чего пользователю необходимо более глубоко работать с реализованными во фреймворке алгоритмами и иметь доступ к тестирующей подсистеме. Критически важным для данного сценария оказывается поддержка параллельных вычислений.

Исследовательский сценарий в большей степени ориентирован на работу с алгоритмами, чем с задачами, его целью является разработка новых алгоритмов оптимизации или глубокая модернизация существующих алгоритмов. Для этого сценария важно иметь возможность автоматического тестирования с доступом к различным тестовым задачам и их наборам.

Образовательный сценарий является промежуточным, его цель – изучение рассматриваемого класса алгоритмов при решении классических задач оптимизации с визуализацией процесса решения. Предпочтительным языком фреймворка для этого сценария является, скорее всего, Python, так как этот язык широко применяется в прикладных исследованиях пользователями, не являющимися специалистами в программировании.

Соответствие фреймворкам четырем описанным сценариям отражено в таблице 7. Заметим, что в отличие от предыдущих таблиц, содержимое данной таблицы носит несколько субъективный характер, так как не существует формальных критериев соответствия фреймворка и сценария его использования. Поэтому в каждом столбце таблицы выделены фреймворки, наиболее приспособленные к заданному сценарию. Это значит, что результаты сравнения фреймворков, показанные в данной таблице, имеют относительный характер, а не абсолютный.

[9] Intel TBB / OneTBB [Электронный ресурс] // GitHub, 2025. URL: https://github.com/oneapi-src/oneTBB (дата обращения: 17.01.2025).

[10] Vanovschi V. Parallel Python Software [Электронный ресурс]. URL: http://www.parallelpython.com (дата обращения: 17.01.2025).

[11] Apache Ignite : сайт [Электронный ресурс] // The Apache Software Foundation, 2025. URL: https://ignite.apache.org (дата обращения: 17.01.2025).

Таблица 7. **Соответствие сценариям использования**
Table. 7. **Compliance with usage scenarios**

Фреймворк	Прост. И.	Продв. И.	Исслед.	Образов.
Pagmo2	●	●	○	○
DEAP	○	●	●	●
Pymoo	●	○	○	●
Jmetal	●	●	●	○
Inspyred	○	●	○	○
MOEA Framework	●	○	●	○
Platypus	●	○	○	○
LEAP	○	●	●	●
Paradise	○	●	○	○
EASEA	●	○	○	○
HeuristicLab	●	○	●	●
Opt4j	●	○	○	○
ECJ	○	●	●	○

Результаты, показанные в таблице 7, показывают, что наиболее универсальными фреймворками для работы с популяционными алгоритмами оптимизации являются фреймворки DEAP, jmetal, LEAP и HeuristicLab. С другой стороны, эти же данные, а также результаты общего анализа, представленные в первых шести таблицах, говорят о том, что универсального фреймворка, полноценно работающего во всех четырех сценариях, нет. Каждый из рассмотренных в обзоре фреймворков обладает какими-то уникальными свойствами, не представленными в других пакетах. Это значит, в частности, что тема разработки программного средства, с которым можно будет эффективно работать на всех указанных уровнях, все еще остается актуальной и практически важной задачей.

References

[1] Storn R., Price K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*. 1997;11:341-359. https://doi.org/10.1023/A:1008202821328
[2] Kennedy J., Eberhart R. Particle swarm optimization. In: Proceedings of the International Conference on Neural Networks (ICNN'95). Perth, WA, Australia: IEEE Press; 1995. Vol. 4. p. 1942-1948. https://doi.org/10.1109/ICNN.1995.488968
[3] Passino K.M. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE control systems magazine.* 2002;22(3):52-67. https://doi.org/10.1109/MCS.2002.1004010
[4] Corana A., et al. Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm. *ACM Transactions on Mathematical Software*. 1987;13(3):262-280. https://doi.org/10.1145/29380.29864
[5] Chen W.N., Tan D.Z. Set-based discrete particle swarm optimization and its applications: a survey. *Frontiers of Computer Science*. 2018;12:203-216. https://doi.org/10.1007/s11704-018-7155-4
[6] Khuri S., Bäck T., Heitkötter J. An Evolutionary Approach to Combinatorial Optimization Problems. *ACM Conference on Computer Science*. 1994:66-73. https://doi.org/10.1145/197530.197558
[7] Bianchi L., et al. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing.* 2009;8:239-287. https://doi.org/10.1007/s11047-008-9098-4
[8] Janecek A., Tan Y. Using Population Based Algorithms for Initializing Nonnegative Matrix Factorization. In: Tan Y., Shi Y., Chai Y., Wang G. (eds.) Advances in Swarm Intelligence. ICSI 2011. *Lecture Notes in Computer Science*. Vol. 6729. Berlin, Heidelberg: 2011. p. 307-316. https://doi.org/10.1007/978-3-642-21524-7_37
[9] Chen X.Y., Chau K.W., Busari A.O. A comparative study of population-based optimization algorithms for downstream river flow forecasting by a hybrid neural network model. *Engineering Applications of Artificial Intelligence*. 2015;46:258-268. https://doi.org/10.1016/j.engappai.2015.09.010
[10] Arnold D.V., Beyer H.G. Noisy Optimization with Evolution Strategies. Universität Dortmund; 2002. 20 p. http://dx.doi.org/10.17877/DE290R-14966
[11] Patle B.K., et al. Path planning in uncertain environment by using firefly algorithm. *Defence technology.* 2018;14(6):691-701. https://doi.org/10.1016/j.dt.2018.06.004
[12] Askarzadeh A. A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. *Computers & structures*. 2016;169:1-12. https://doi.org/10.1016/j.compstruc.2016.03.001
[13] Kaboli S.H.A., Selvaraj J., Rahim N.A. Rain-fall optimization algorithm: A population based algorithm for solving constrained optimization problems. *Journal of Computational Science*. 2017;19:31-42. https://doi.org/10.1016/j.jocs.2016.12.010

[14] Deb K. Multi-objective optimisation using evolutionary algorithms: an introduction // Multi-objective evolutionary optimisation for product design and manufacturing. London: Springer London, 2011. https://doi.org/10.1007/978-0-85729-652-8_1
[15] Rajpurohit J., et al. Glossary of metaheuristic algorithms. *International Journal of Computer Information Systems and Industrial Management Applications*. 2017;9:25-25. Available at: https://cspub-ijcisim.org/index.php/ijcisim/article/view/353 (accessed 17.01.2025).
[16] Cantu-Paz E., Goldberg D.E. On the scalability of parallel genetic algorithms. *Evolutionary computation*. 1999;7(4):429-449. https://doi.org/10.1162/evco.1999.7.4.429
[17] Essaid M., et al. GPU parallelization strategies for metaheuristics: a survey. *International Journal of Parallel, Emergent and Distributed Systems.* 2019;34(5):497-522. https://doi.org/10.1080/17445760.2018.1428969
[18] Parejo J.A., Ruiz-Cortés A., Lozano S., Fernandez P. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing.* 2012;16(3):527-561. https://doi.org/10.1007/s00500-011-0754-8
[19] Ramírez A., Barbudo R., Romero J.R. An experimental comparison of metaheuristic frameworks for multi-objective optimization. *Expert Systems*. 2023;40(4):e12672. https://doi.org/10.1111/exsy.12672
[20] Talbi E. G. Automated design of deep neural networks: A survey and unified taxonomy. *ACM Computing Surveys*. 2021;54(2):1-37. https://doi.org/10.1145/3439730
[21] Huang H.Z., Zhang X. Design optimization with discrete and continuous variables of aleatory and epistemic uncertainties. *Journal of Mechanical Design*. 2009;131(3):031006. https://doi.org/10.1115/1.3066712
[22] Kokkolaras M., Audet C., Dennis J. E. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*. 2001;2:5-29. https://doi.org/10.1023/A:1011860702585
[23] Gutjahr W.J., Strauss C., Wagner E. A stochastic branch-and-bound approach to activity crashing in project management. *INFORMS journal on computing*. 2000;12(2):125-135. https://doi.org/10.1287/ijoc.12.2.125.11894
[24] Patle B.K., et al. Path planning in uncertain environment by using firefly algorithm. *Defence technology*. 2018;14(6):691-701. https://doi.org/10.1016/j.dt.2018.06.004
[25] Mahanta B.K., Chakraborti N. Tri-objective optimization of noisy dataset in blast furnace iron-making process using evolutionary algorithms. *Materials and Manufacturing Processes*. 2020;35(6):677-686. https://doi.org/10.1080/10426914.2019.1643472
[26] Emmerich M.T.M., Deutz A.H. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*. 2018.17:585-609. https://doi.org/10.1007/s11047-018-9685-y
[27] De Castro L.N., Von Zuben F.J. Learning and optimization using the clonal selection principle. *IEEE transactions on evolutionary computation.* 2002;6(3):239-251. https://doi.org/10.1109/TEVC.2002.1011539
[28] Reynolds C.W. Flocks, herds and schools: A distributed behavioral model. In: Proceedings of the 14th annual conference on Computer graphics and interactive techniques. (SIGGRAPH '87). New York, NY, USA: Association for Computing Machinery; 1987. p. 25-34. https://doi.org/10.1145/37401.37406
[29] Dorigo M., Birattari M., Stutzle T. Ant colony optimization. *IEEE computational intelligence magazine*. 2006;1(4):28-39. https://doi.org/10.1109/MCI.2006.329691
[30] Pham D.T., et al. The Bees Algorithm – A Novel Tool for Complex Optimisation Problems. Intelligent production machines and systems. 2nd I*PROMS Virtual International Conference 3-14 July 2006. Elsevier Science Ltd; 2006. p. 454-459. https://doi.org/10.1016/B978-008045157-2/50081-X
[31] Krishnanand K.N., Ghose D. Glowworm swarm optimisation: a new method for optimising multi-modal functions. *International Journal of Computational Intelligence Studies*. 2009;1(1):93-119. https://doi.org/10.1504/IJCISTUDIES.2009.025340
[32] Cheng R., Jin Y. A competitive swarm optimizer for large scale optimization. *IEEE transactions on cybernetics*. 2014;45(2):191-204. https://doi.org/10.1109/TCYB.2014.2322602
[33] Rashedi E., Nezamabadi-Pour H., Saryazdi S. GSA: a gravitational search algorithm. *Information sciences*. 2009;179(13):2232-2248. https://doi.org/10.1016/j.ins.2009.03.004
[34] Rabanal P., Rodríguez I., Rubio F. Using River Formation Dynamics to Design Heuristic Algorithms. In: Akl S.G., Calude C.S., Dinneen M.J., Rozenberg G., Wareham H.T. (eds.) Unconventional Computation. UC 2007. *Lecture Notes in Computer Science*. Vol. 4618. Berlin, Heidelberg: Springer; 2007. p. 163-177. https://doi.org/10.1007/978-3-540-73554-0_16
[35] Miranda L.J. PySwarms: a research toolkit for Particle Swarm Optimization in Python. *Journal of Open Source Software.* 2018;3(21):433. https://doi.org/10.21105/joss.00433
[36] Biscani F., Izzo D. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software.* 2020;5(53):2338. https://doi.org/10.21105/joss.02338
[37] Fortin F.A., et al. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research.* 2012;13(1):2171-2175. Available at: https://dl.acm.org/doi/10.5555/2503308.2503311 (accessed 17.01.2025).
[38] Blank J., Deb K. Pymoo: Multi-objective optimization in python. *IEEE Access*. 2020;8:89497-89509. https://doi.org/10.1109/ACCESS.2020.2990567
[39] Durillo J.J., Nebro A.J. jMetal: A Java framework for multi-objective optimization. *Advances in engineering software.* 2011;42(10):760-771. https://doi.org/10.1016/j.advengsoft.2011.05.014
[40] Coletti M.A., Scott E.O., Bassett J.K. Library for evolutionary algorithms in Python (LEAP). In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20). New York, NY, USA: Association for Computing Machinery; 2020. p. 1571-1579. https://doi.org/10.1145/3377929.3398147

[41] Dreo J., et al. Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of Paradiseo. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '21). New York, NY, USA: Association for Computing Machinery; 2021. p. 1522-1530. https://doi.org/10.1145/3449726.3463276
[42] Collet P., Lutton E., Schoenauer M., Louchet J. Take It EASEA. In: Schoenauer, M., et al. Parallel Problem Solving from Nature PPSN VI. PPSN 2000. Lecture Notes in Computer Science. Vol. 1917. Berlin, Heidelberg: Springer; 2000. p. 891-901. https://doi.org/10.1007/3-540-45356-3_87
[43] Wagner S., et al. Architecture and Design of the HeuristicLab Optimization Environment. In: Klempous R., Nikodem J., Jacak W., Chaczko Z. (eds.) Advanced Methods and Applications in Computational Intelligence. *Topics in Intelligent Engineering and Informatics*. Vol. 6. Heidelberg: Springer; 2014. p. 197-261. https://doi.org/10.1007/978-3-319-01436-4_10
[44] Lukasiewycz M., Glaß M., Reimann F., Teich J. Opt4J: a modular framework for meta-heuristic optimization. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO '11). New York, NY, USA: Association for Computing Machinery; 2011. p. 1723-1730. https://doi.org/10.1145/2001576.2001808
[45] Navarro R., Nápoles G. Classical niching methods for multimodal optimization: a brief review. *Revista Cubana de Ciencias Informáticas*. 2013;7(2):137-153. Available at: https://rcci.uci.cu/index.php/RCCI/article/view/430 (accessed 17.01.2025). (In Spain., abstract in Eng.)
[46] Jiang S., Kaiser M., Wan S., Guo J., Yang S., Krasnogor N. An Empirical Study of Dynamic Triobjective Optimisation Problems. In: 2018 IEEE Congress on Evolutionary Computation (CEC). Rio de Janeiro, Brazil: IEEE Press; 2018. p. 1-8. https://doi.org/10.1109/CEC.2018.8477667
[47] Zitzler E., Deb K., Thiele L. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*. 2000;8(2):173-195. https://doi.org/10.1162/106365600568202
[48] Deb K., Thiele L., Laumanns M., Zitzler E. Scalable Test Problems for Evolutionary Multiobjective Optimization. In: Abraham A., Jain L., Goldberg R. (eds.) Evolutionary Multiobjective Optimization. Advanced Information and Knowledge Processing. London: Springer; 2005. p. 105-145. https://doi.org/10.1007/1-84628-137-7_6
[49] Virtanen P., et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*. 2020;17(3):261-272. https://doi.org/10.1038/s41592-019-0686-2
[50] de Nobel J., Ye F., Vermetten D., Wang H., Doerr C., Bäck T. IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics. *Evolutionary Computation*. 2024;32(3):205-210. https://doi.org/10.1162/evco_a_00342
[51] Telikani A., et al. Evolutionary machine learning: A survey. *ACM Computing Surveys*. 2021;54(8):1-35. https://doi.org/10.1145/3467477
[52] Zhang Y., et al. Evolutionary Computation in bioinformatics: A survey. *Neurocomputing*. 2024;591:127758. https://doi.org/10.1016/j.neucom.2024.127758
[53] Cantú-Paz E. Topologies, migration rates, and multi-population parallel genetic algorithms. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation – Vol. 1 (GECCO'99). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1999. p. 91-98. Available at: https://dl.acm.org/doi/10.5555/2933923.2933933 (accessed 17.01.2025).
[54] Hold-Geoffroy Y., Gagnon O., Parizeau M. Once you SCOOP, no need to fork. In: Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment (XSEDE '14). New York, NY, USA: Association for Computing Machinery; 2014. Article number: 60. https://doi.org/10.1145/2616498.2616565
[55] Rocklin M. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In: Proceedings of the 14th Python in Science Conference (SciPy 2015). Austin, Texas July 6-12, 2015. p. 126-132. https://doi.org/10.25080/MAJORA-7B98E3ED-013
[56] Dalcin L., Fang Y.L.L. mpi4py: Status Update After 12 Years of Development. *Computing in Science & Engineering*. 2021;23(4):47-54. https://doi.org/10.1109/MCSE.2021.3083216

Поступила 17.01.2025; одобрена после рецензирования 02.03.2025; принята к публикации 26.03.2025.
Submitted 17.01.2025; approved after reviewing 02.03.2025; accepted for publication 26.03.2025.

Об авторах:

Николашкин Алексей Герасимович, аспирант кафедры суперкомпьютеров и квантовой информатики факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М.В. Ломоносова» (119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1), **ORCID: https://orcid.org/0009-0005-5089-3446,** nagvv97@mail.ru

Ершов Николай Михайлович, старший научный сотрудник кафедры автоматизации научных исследований факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М.В. Ломоносова» (119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1), кандидат физико-математических наук, **ORCID: https://orcid.org/0000-0001-5963-0419**, ershov@cs.msu.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

About the authors:

Aleksei G. Nikolashkin, Postgraduate Student of the Department of Supercomputers and Quantum Informatics, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation), **ORCID: https://orcid.org/0009-0005-5089-3446,** nagvv97@mail.ru

Nikolay M. Ershov, Senior Researcher of the Department of Automation for Scientific Research, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation), Cand. Sci. (Phys.-Math.), **ORCID: https://orcid.org/0000-0001-5963-0419**, ershov@cs.msu.ru

All authors have read and approved the final manuscript.