С. И. Родзин
О. Н. Родзина

Институт
Компьютерных
Технологий и
Информационной
Безопасности

МАШИННОЕ ОБУЧЕНИЕ: МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ

С. И. Родзин
О. Н. Родзина

МАШИННОЕ ОБУЧЕНИЕ: МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ

Учебное пособие

Чебоксары
Издательский дом «Среда»
2024

УДК 004(075.8)
ББК 32.813я73
Р60

Авторы:

С.И. Родзин – канд. техн. наук, доцент, профессор ФГАОУ ВО «Южный федеральный университет» (ЮФУ), г. Таганрог;

О.Н. Родзина – старший преподаватель ФГАОУ ВО «Южный федеральный университет» (ЮФУ), г. Таганрог

Рецензенты:

д-р техн. наук, профессор, заместитель директора Ростовского филиала научно-исследовательского и проектно-конструкторского Института информатизации, автоматизации и связи на железнодорожном транспорте
С.М. Ковалев;
д-р технических наук, профессор, заместитель директора по науке Санкт-Петербургского филиала ФГБУН «Институт земного магнетизма, ионосферы и распространения радиоволн им. Н. В. Пушкова РАН
А.Г. Коробейников

Р60 **Родзин С.И.**
Машинное обучение: метаэвристики дифференциально-векторного движения: учебное пособие / С.И. Родзин, О.Н. Родзина. – Чебоксары: Среда, 2024. – 140 с.

ISBN 978-5-907830-17-2

Метаэвристические алгоритмы, инспирированные природой, являясь одним из направлений в искусственном интеллекте и машинном обучении, в последние десятилетия стали мощным инструментом оптимизации, широко используются при распознавании образов и компьютерном зрении, технической и медицинской диагностике, прогнозировании, биоинформатике, фильтрации спама, анализе фондового рынка, в поисковых системах, в аффективных вычислениях, компьютерных играх. В учебном пособии рассматривается современное состояние, проблемы и области применения метаэвристик. Представлены алгоритмы дифференциально-векторного движения: дифференциальной эволюции, синуса-косинуса, эгоистичного стада животных, летучих мышей, империалистической конкуренции, роя саранчи, гравитационного поиска, светлячковый, бабочки монарха, червей, COVID-19, бактериальный, межнейронного взаимодействия, оптимизации группового обучения.

Пособие адресовано бакалаврам и магистрам направлений «Информатика и вычислительная техника», «Информационные системы и технологии», «Программная инженерия», «Прикладная информатика», а также аспирантам специальности «Искусственный интеллект и машинное обучение».

ISBN 978-5-907830-17-2
DOI 10.31483/a-10610

ОГЛАВЛЕНИЕ

Предисловие

В основу учебного пособия положены материалы лекционных и практических занятий в рамках односеместровых курсов, которые Сергей Родзин в течение нескольких лет читает в Южном федеральном университете в Институте компьютерных технологий и информационной безопасности (https://ictis.sfedu.ru/) на кафедре математического обеспечения и применения ЭВМ (https://vk.com/sed_announce). Курсы под названием «Искусственный интеллект и анализ данных» и «Разработка приложений для систем искусственного интеллекта» задумывались для аспирантов и магистрантов, но оказались полезными также для бакалавров программ «Технологии искусственного интеллекта», «Методы и средства разработки программного обеспечения». В учебное пособие вошли также материалы некоторых научных статей авторов, используемых в проектной и исследовательской деятельности студентов.

О чем учебное пособие? В самых общих словах это учебное пособие о метаэвристических алгоритмах, инспирированных природой, как об одном из направлений в искусственном интеллекте и машинном обучении. Но акцент сделан на эффективных алгоритмах из класса дифференциально-векторного движения, использующих в процессе обучения для генерации новых решений как всю популяцию, так и репрезентативных агентов, субпопуляцию или агентов из некоторой окрестности. Авторы сознательно стремились уделять больше внимания содержательным аспектам алгоритмов, их особенностям и сравнительным оценкам. Кроме того, в книге принят подход, согласно которому многие практические задачи иллюстрируют и поясняют метаэвристические алгоритмы. Ниже перечислены основные рассматриваемые темы.

1. Современное состояние и проблемы метаэвристик, включая терминологию и классификацию, тестирование и области применения метаэвристик.

2. Метаэвристики дифференциально-векторного движения, использующие в процессе обучения и генерации новых решений репрезентативных агентов популяции, включая алгоритм дифференциальной эволюции, синус-косинусный алгоритм, алгоритм эгоистичного стада, алгоритм летучих мышей и алгоритм империалистической конкуренции.

3. Метаэвристики дифференциально-векторного движения, использующие в процессе обучения и генерации новых решений всю популяцию агентов, включая алгоритм роя саранчи, алгоритм колонии пауков, гибридный алгоритм роя саранчи и колонии пауков, алгоритм гравитационного поиска и светлячковый алгоритм.

4. Метаэвристики дифференциально-векторного движения, использующие в процессе обучения и генерации новых решений субпопуляцию агентов, включая алгоритм бабочки монарха, алгоритм червей и алгоритм Ковид-19.

5. Метаэвристики дифференциально-векторного движения, использующие в процессе обучения и генерации новых решений агентов из некоторой окрестности, включая бактериальный алгоритм, алгоритм межнейронного взаимодействия и алгоритм оптимизации группового обучения.

Требования к читателю.

Для полного понимания изложенного в учебном пособии материала мы рекомендуем следующее.

1. Прослушать курс по дискретной математике, теории вероятностей и математической статистике.

2. Иметь знания об информатике, структурах данных и алгоритмах в университетском объеме.

3. Иметь знания о программных системах, программной инженерии и языках программирования.

Благодарности.

Слова глубокого уважения и признательности авторы адресуют профессору ЮФУ В.М. Курейчику, которого считают своим наставником и другом, чья научная эрудиция и широта интересов всегда служили образцом для подражания.

Авторы выражают искреннюю благодарность рецензентам пособия – доктору технических наук, профессору С.М. Ковалеву, доктору технических наук, профессору А.Г. Коробейникову, а также своему коллеге по работе в Институте компьютерных технологий и информационной безопасности, доктору технических наук, профессору В.В. Курейчику, которые взяли на себя нелегкий труд ознакомиться с рукописью и высказали немало ценных советов и конструктивных замечаний, способствовавших устранению неточностей и улучшению содержания пособия.

ВВЕДЕНИЕ

Машинное обучение – это направление в искусственном интеллекте, которое исследует алгоритмы, способные обучаться и самостоятельно улучшаться по мере накопления опыта, извлекать закономерности из данных и делать прогнозы на их основе. Машинное обучение также тесно связано с математической статистикой и оптимизацией, с интеллектуальным анализом данных и распознаванием образов. Технологии и методы машинного обучения широко используются при распознавании образов и компьютерном зрении, технической и медицинской диагностике, прогнозировании, биоинформатике, фильтрации спама, обнаружении мошенничества, категоризации документов, кредитном скоринге и анализе фондового рынка, хемоинформатике, в поисковых системах, в аффективных вычислениях (сбор данных из выражений лица, голосовых сообщений и языка тела для измерения уровня человеческих эмоций), Интернет-рекламе, компьютерных играх. Причем сфера применений алгоритмов машинного обучения постоянно расширяется. Цифровизация приводит к накоплению огромных объёмов данных в науке, производстве, бизнесе, транспорте, здравоохранении. Возникающие при этом задачи анализа данных, прогнозирования, управления и поиска оптимальных решений часто сводятся к обучению по прецедентам. Раньше, когда таких данных не было, эти задачи либо вообще не ставились, либо решались совершенно другими методами.

В зависимости от характера обучающего «сигнала» или «обратной связи» задачи машинного обучения обычно подразделяются на обучение с учителем, обучение без учителя и обучение с подкреплением.

При обучении с учителем компьютеру предъявляются примеры входных данных и их желаемые выходные данные (обучающая выборка), заданные «учителем», и цель состоит в том, чтобы найти общее правило, которое сопоставляет входные данные с выходными.

При обучении без учителя алгоритму обучения не присваиваются метки, он должен самостоятельно найти структуру во входных данных или обнаружить скрытые закономерности в данных.

При обучении с подкреплением компьютерная программа взаимодействует со средой, в которой она должна выполнять определенную задачу, без явного указания учителя о том, приблизилась она к своей цели или нет. Откликом среды на принятые решения являются сигналы

подкрепления, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда или ее модель.

Другая классификация задач машинного обучения возникает в зависимости от выходных данных в результате машинного обучения. В частности, если при классификации входные данные разделяются на два или более классов, то необходимо построить модель, которая относит входные данные к одному или нескольким классам. Например, фильтрация спама является примером классификации, где входными данными являются сообщения электронной почты (или другие), а классами являются «спам» и «не спам». При решении регрессионной задачи выходные данные должны быть непрерывными, а не дискретными. При решении задачи кластеризации набор входных данных должен быть разделен на заранее неизвестное число групп.

Основная цель алгоритмов машинного обучения – обобщение своего опыта. Обобщение опыта – это способность обучающей машины правильно решать новые задачи после изучения набора обучающих данных. Поскольку набор обучающих данных является конечным, то для оценки эффективности алгоритмов машинного обучения обычно используется дисперсия в качестве способа количественной оценки ошибки обобщения. В дополнение к этому также используется временная оценка сложности алгоритмов машинного обучения (в теории машинного обучения алгоритм считается выполнимым, если он может быть выполнен за полиномиальное время).

Перечень современных алгоритмов машинного обучения включает:

• ***деревья принятия решений***, структура которых включает «листья» и «ветки». На ветках дерева решения записываются признаки, от которых зависит целевая функция, в листьях записываются значения целевой функции, а в остальных узлах – признаки, по которым различаются примеры. Чтобы классифицировать новый пример, надо спуститься по дереву до листа и выдать соответствующее значение. Метод деревьев принятия решений отличается простотой, не требует специальной подготовки данных, работает с категориальными и с интервальными переменными, получаемые результаты можно объяснить при помощи булевой логики (в отличие от нейронных сетей) и оценить при помощи статистических тестов, позволяет работать с большим объёмом информации. Однако следует иметь в виду, что проблема получения оптимального дерева решений является *NP*-полной задачей, при построении дерева

решений могут создаваться слишком сложные конструкции, которые недостаточно полно представляют данные;

• ***поиск ассоциативных правил*** для обнаружения интересующих нас связей между переменными в большой базе данных. Алгоритм генерирует также новые правила по мере анализа данных и создает возможность нахождения ассоциаций из новых неклассифицированных данных. Широко применяется при анализе рыночной корзины, в областях *Web mining*, а также для обнаружения вторжений, и биоинформатике. Одним из ограничений стандартного подхода к обнаружению ассоциаций является то, что при поиске в большом числе возможных ассоциаций множества объектов есть определенный риск нахождения случайных ассоциаций. Предложено много алгоритмов для генерации ассоциативных правил (*Apriori, Eclat, FP-Growth*), но они делают только половину работы, поскольку предназначены для отыскания часто встречающихся наборов объектов;

• ***обучение искусственной нейронной сети*** (ИНС). Некоторые ИНС обучаются без учителя (например, сети Хопфилда), они просматривают выборку только один раз. Другие (например, сети Кохонена), а также сети, обучающиеся с учителем, просматривают выборку множество раз (эпохи обучения). При обучении с учителем набор исходных данных делят на две части – собственно обучающую выборку и тестовые данные; принцип разделения может быть произвольным. Обучающие данные подаются сети для обучения, а проверочные используются для расчета ошибки сети (проверочные данные никогда для обучения сети не применяются). Таким образом, если на проверочных данных ошибка уменьшается, то сеть действительно выполняет обобщение. Если ошибка на обучающих данных продолжает уменьшаться, а ошибка на тестовых данных увеличивается, значит, сеть перестала выполнять обобщение и просто «запоминает» обучающие данные. Это явление называется переобучением сети (оверфиттинг). В таких случаях обучение обычно прекращают. В процессе обучения могут проявиться другие проблемы, такие как паралич или попадание сети в локальный минимум поверхности ошибок. Невозможно заранее предсказать проявление той или иной проблемы, равно как и дать однозначные рекомендации к их разрешению. Это относится только к итерационным алгоритмам поиска нейросетевых решений. Для них действительно нельзя ничего гарантировать и нельзя полностью автоматизировать обучение

нейронных сетей. Наряду с итерационными алгоритмами обучения, существуют алгоритмы, обладающие очень высокой устойчивостью и позволяющие полностью автоматизировать процесс обучения. В целом современные нейронные сети – это инструменты нелинейного статистического моделирования данных. Они используются для моделирования сложных взаимосвязей между входными и выходными данными, для поиска закономерностей в данных и других задач. Примерами использования нейросетей также являются нейроуправление, предсказание финансовых временных рядов, психодиагностика, хемоинформатика, экономика, роботы, беспилотные автомобили и дроны, распознавание образов и анализ текста, рекомендательные системы и нейрокомпьютеры;

• ***индуктивное логическое программирование*** (ИЛП) использует логику для представления примеров, фоновых знаний и гипотез. Алгоритм системы ИЛП состоит из двух частей: поиска гипотезы и выбора гипотезы. Получив описания уже известных фоновых знаний и множества примеров, представленных как логическая база фактов, ИЛП может породить логическую программу в форме гипотез, объясняющую все положительные примеры и ни одного отрицательного. Базовые знания излагаются в виде логики, обычно в форме предложений Хорна. Возникают вопросы о высокой вычислительной сложности (*NP*-полная задача), о полноте процедуры поиска гипотез в конкретной системе ИЛП. Индуктивно-логическое программирование особенно полезно в биоинформатике, обработке естественного языка, медицине и фармакологии. Обычно реализации ИЛП делаются на языке *Prolog*;

• ***машины опорных векторов*** (*SVM*) представляют собой набор связанных алгоритмов обучения с учителем, используемых для задач классификации и регрессии. Принадлежит семейству линейных классификаторов. Основная идея *SVM* – перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с наибольшим зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, создающая наибольшее расстояние до двух параллельных гиперплоскостей. Алгоритм основан на допущении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя

ошибка классификатора. Учитывая набор обучающих примеров, каждый из которых помечен как относящийся к одной из двух категорий, алгоритм *SVM* строит модель, которая предсказывает, попадает ли новый пример в ту или иную категорию. Однако использование алгоритма *SVM* требует полной маркировки входных данных, что применимо только для задач двух классов. *SVM* применяются для решения различных реальных задач: категоризация и семантический анализ текста, классификации и сегментации изображений, спутниковых данных, а также в биологических науках;

• ***кластерный анализ*** – это многомерная статистическая процедура, выполняющая сбор данных, содержащих информацию о выборке объектов, и затем упорядочивающая объекты в сравнительно однородные группы в соответствии с некоторыми заранее определенными критериями. Задача кластеризации относится к широкому классу задач обучения без учителя. Различные процедуры кластеризации делают разные предположения о структуре данных согласно некоторой метрики сходства оцениваемой, например, по внутренней компактности (сходству между данными одного и того же кластера) и разделению на различные кластеры;

• ***байесовская сеть*** – это вероятностная модель, представляющая собой множество переменных и их вероятностных зависимостей согласно теореме Байеса. Формально байесовская сеть представляет ориентированный ациклический граф, каждой вершине которого соответствует случайная переменная, а дуги графа кодируют отношения условной независимости между этими переменными. Вершины могут представлять переменные любых типов, быть взвешенными параметрами, скрытыми переменными или гипотезами. Существуют эффективные методы, которые используются для вычислений и обучения байесовских сетей. Байесовские сети, в которых могут присутствовать как дискретные переменные, так и непрерывные, называются гибридными байесовскими сетями. Байесовская сеть, в которой дуги помимо отношений условной независимости кодируют также отношения причинности, называют причинно-следственными байесовскими сетями. Например, байесовская сеть может представлять вероятностные взаимосвязи между болезнями и симптомами. Учитывая симптомы, сеть может быть использована для вычисления вероятностей наличия различных заболеваний. Байесовские сети ис-

пользуются для моделирования в биоинформатике, медицине, классификации документов, обработке данных и изображений, системах поддержки принятия решений;

• ***метаэвристики, инспирированные природой*** представляют собой математические преобразования, трансформирующие входной поток информации в выходной и основанные на правилах имитации биоинспирированных механизмов, на процедурах, содержащих элементы стохастической оптимизации со случайными переменными. Основными особенностями метаэвристик являются стратегии, которые направляют процесс поиска оптимальных решений, а также широкое варьирование алгоритмов от простых локальных процедур до сложных недетерминированных процедур. С помощью метаэвристик, инспирированных природой, исследуется пространство поиска, синтезируются решения, являющиеся точками этого пространства, запрашивается оценка их качества или «приспособленности», которая затем используется для осуществления «естественного отбора». Тем самым, метаэвристики обучаются тому, какие области пространства поиска содержат наиболее приспособленных особей. Метаэвристики обычно разрабатываются на основе наблюдений за природными процессами или паттернами поведения биологических организмов. Метаэвристики применяются для решения всех типов задач оптимизации, начиная от непрерывных задач и заканчивая комбинаторной оптимизацией. Многомерные задачи, включая большинство задач проектирования в инженерии, сталкиваются с проблемой проклятия размерности, что также делает их неосуществимыми для исчерпывающего поиска или аналитических методов. Метаэвристики также применяются к разнообразным задачам планирования, проектирования, логистики.

В учебном пособии изложены современное состояние метаэвристик, инспирированных природой, терминология, подходы к классификации и тестированию, а также основные области применения. Представлены эффективные алгоритмы из класса дифференциально-векторного движения, использующие как репрезентативных агентов популяции, так и всю популяции, субпопуляцию или агентов из некоторой окрестности. Эффективный метаэвристический алгоритм – это тот, который обеспечивает баланс между скоростью сходимости и диверсификацией пространства поиска решений; обеспечивает высокий уровень точности, сходится к оптимальному решению и способен

преодолевать локальные оптимумы; имеет стабильную производительность, при которой результаты существенно не отличаются от одного независимого запуска к другому; может применяться для решения разнообразных реальных задач оптимизации.

Пособие состоит из 5 глав.

В главе 1 приводится терминология и современная классификация метаэвристик, вопросы их тестирования, а также области практического применения.

Глава 2 посвящена метаэвристикам дифференциально-векторного движения, использующих репрезентативных агентов популяции: алгоритм дифференциальной эволюции, синус-косинусный алгоритм, алгоритм эгоистичного стада, летучих мышей, алгоритм империалистической конкуренции.

В главе 3 рассмотрены метаэвристики дифференциально-векторного движения, использующие для генерации решений всю популяцию агентов: алгоритм роя саранчи, алгоритм колонии пауков, гибридный алгоритм роя саранчи и колонии пауков, алгоритм гравитационного поиска, светлячковый алгоритм.

В главе 4 представлены метаэвристики дифференциально-векторного движения, использующие для генерации решений субпопуляции агентов: алгоритм бабочки-монарха, алгоритм червей, алгоритм Ковид-19.

Глава 5 посвящена метаэвристикам дифференциально-векторного движения, использующим для генерации решений агентов из некоторой окрестности: бактериальный алгоритм, алгоритм межнейронного взаимодействия, алгоритм группового обучения.

1. СОВРЕМЕННОЕ СОСТОЯНИЕ И ПРОБЛЕМЫ МЕТАЭВРИСТИК

В последние годы наблюдается стремительный рост числа метаэвристик и исследовательского интереса к ним. Согласно отечественным и зарубежным библиометрическим данным (*Web of Science – https://apps.webofknowledge.com/, Google Scholar – https://scholar.google/, Springer Link – https://link.springer.com/*, библиотека *IEEE Xplore – https://ieeexplore.ieee.org/Xplore/*, библиотека *eLibrary.Ru – https://elibrary.ru/*, КиберЛенинка – *https://cyberleninka.ru/*), среднегодовая норма цитирования статей в области искусственного интеллекта составляет около 5, в то время как 85 % статей, в которых предлагаются метаэвристики, цитируются в среднем около 20 раз.

1.1. Терминология и классификация метаэвристик

Рассмотрим существующие подходы к классификации и категоризации метаэвристик, а также терминологию, используемую в этой области машинного обучения.

Классификация обозначает разновидность деления объема понятия по определенному основанию (признаку, критерию), при котором объем родового понятия (класс, множество) делится на виды (подклассы, подмножества), а виды, в свою очередь делятся на подвиды и т. д. Классификация метаэвристик как процесс предполагает упорядоченное и систематическое отнесение каждой метаэвристики к одному и только одному классу в рамках системы взаимоисключающих и неперекрывающихся классов. Классификация сортирует метаэвристики в соответствии с их сходством, используя определенный набор признаков или критериев.

Термин «таксономия» изначально применялся только в биологии. Позже он стал использоваться для обозначения общей теории классификации и систематизации сложных систем как в биологии, так и в других областях знаний.

Категоризация обозначает деятельность по распознаванию общих черт или сходств между объектами и их отнесения к абстрактной группе (категории, классу или типу) на основе признаков сходства. Иными словами, категоризация подразумевает процесс разделения объектов на некоторые группы, объекты которых в некотором роде похожи друг на друга. В отличие от классификации, при

категоризации объект может быть частью более чем одной категории, в зависимости от контекста.

Основываясь на этих определениях, будем использовать термин «категоризация» для организации систем упорядочения с критериями, позволяющими сортировать метаэвристики в нескольких группах одновременно или несколько метаэвристик в одной группе. Наоборот, будем использовать термин «классификация», если каждая метаэвристика может быть четко упорядочена в отдельном классе по определенным критериям. Если при классификации остается группа метаэвристик, то внутри этой группы метаэвристики идентичны по критериям классификации.

Классификация позволяет объективно подходить к выбору метаэвристик, поскольку для каждой из них имеются конкретные задачи, с которыми она справляется особенно хорошо. Знать и понимать эти взаимосвязи важно для целенаправленного применения метаэвристики.

Одной из объективных проблем при классификации метаэвристик является бессистемность их обозначений. Большинство метаэвристик используют обозначения, основанные на их природной метафоре. Вместо использования общих терминов оптимизации. Это вызывает трудности при формулировании общих критериев классификации и извлечении необходимой информации из метаэвристики для применения этих критериев. Проблемой также является отсутствие руководящих принципов в отношении критериев классификации. Это приводит к множеству слабо детализированных схем классификации. Зачастую критерии классификации выбираются не по принципу их информативности и значимости, а с точки зрения простоты применения.

Существующая таксономия метаэвристик, основанная на природных метафорах, не позволяет однозначно различать метаэвристики. Этот подход направлен на категоризацию и предполагает, что метаэвристика может быть частью более чем одного класса.

Другая классификация предполагает разделение метаэвристик на группы биологических, физических, социальных, музыкальных, химических, спортивных, математических и роевых систем, а также критерии, разделяющие метаэвристики траекторного типа (в области поиска эволюционирует только одно решение задачи) и метаэвристики

популяционного типа (одновременно эволюционируют несколько вариантов решения задачи). Однако эти критерии также не позволяют провести однозначную классификацию метаэвристик.

Имеется обширный набор критериев категоризации метаэвристик в зависимости от ограничений на вид и число целевых функций оптимизации, значения переменных в задаче, детерминированный или стохастический характер переменных. Однако эти критерии затруднительно применять при классификации метаэвристик, потому что они сильно зависят от соответствующей реализации метаэвристического алгоритма.

В [1] была предложена категоризация, которая включает эволюционные и роевые метаэвристики, а также метаэвристики, основанные на физических процессах и на когнитивных процессах, связанных с деятельностью человека. Предлагаемое множество критериев включает такие характеристики, как скорость сходимости алгоритма, диверсификация пространства поиска решений, механизм селекции решений, вычислительная трудоемкость, требуемый объем памяти, настройка параметров алгоритма, трудности программной реализации.

Одной из важных особенностей метаэвристик является баланс между скоростью сходимости и диверсификацией пространства поиска оптимальных решений. Диверсификация пространства поиска решений относится к способности поисковых агентов исследовать новые области пространства поиска, в то время как сходимости алгоритма делает акцент на возможностях этих агентов уточнять уже найденные “хорошие” решения. Поиск баланса между ними зависит от используемого в метаэвристике механизма популяционного отбора, аттрактивности операторов поиска, настройки параметров и от числа итераций алгоритма.

Некоторые подходы к классификации метаэвристик основаны на использовании таких специфических характеристик как локальный поиск в окрестности, восхождение на холм, многократный запуск алгоритма, адаптивное программирование памяти. Метаэвристики роевого интеллекта иногда разделяются на инспирированные водной фауной, наземными животными, птицами, насекомыми и микроорганизмами.

Ясно, что современная классификация метаэвристик должна быть многоуровневой [2]. Уровни классификации и используемые критерии представлены в табл. 1.1.

Таблица 1.1

Многоуровневая классификация метаэвристик

№	Уровень	Критерии
1	Природная метафора	Роевые алгоритмы; алгоритмы, основанные на физических и химических процессах; алгоритмы, основанные на когнитивных процессах и деятельности человека; эволюционные алгоритмы; алгоритмы, основанные на особенностях организмов, способных к фотосинтезу; прочие алгоритмы
2	Структурный	Дискретные/траекторные; популяционные/одиночные; с памятью/без памяти
3	Поведенческий	Получение новых решений: создаются из уже имеющихся решений или путем дифференциально-векторного движения.
4	Поисковый	Оценка баланса: скорость сходимости метаэвристики/диверсификация пространства поиска
5	Компонентный	Локальный поиск; эволюционный механизм; механизм оптимизации роя частиц; механизм оптимизации колонии муравьев.
6	Специфические особенности	Механизмы расширения структуры фреймворков; вычислительная сложность метаэвристики
7	Оценочный	Тип/размерность задачи

Рассмотрим подробнее каждый уровень и соответствующие критерии классификации.

Уровень соответствия природной метафоре. Разнообразие природных метафор может привести к мелкозернистой классификации. Необходим компромисс, чтобы сохранить классификацию простой, но информативной. В результате более чем 400 из существующих метаэвристик разделяются по критерию соответствия природной метафоре на шесть классов: (1) роевые алгоритмы (48 %); (2) алгоритмы, основанные на физических и химических процессах (18 %); (3) алгоритмы, основанные на когнитивных процессах и деятельности человека (12 %); (4) эволюционные алгоритмы (7 %); (5) алгоритмы, основанные на особенностях многоклеточных организмов, способных к фотосинтезу (5 %); (6) прочие алгоритмы, которые

настолько различны, что их невозможно сгруппировать в представительные классы (4 или более метаэвристики, около 10 %). Для прояснения критериев классификации на этом уровне приведем описание основных характеристик каждого из шести классов.

Алгоритмы, основанные на роевом интеллекте, характеризуются коллективным поведением децентрализованных, самоорганизующихся агентов. Первоначально этот термин был предложен в контексте роботизированных систем, однако с годами получил широкое распространение для обозначения коллективного разума в группе агентов, управляемых простыми правилами поведения. Речь идет о моделях коллективного поведения таких сообществ как колонии насекомых или птичьи стаи, где коллективный интеллект роя позволяет эффективно решать задачи оптимизации.

Роевые метаэвристики разделяются по следующим критериям классификации: (1) алгоритмы, инспирированные полетом птиц и насекомых (17 %, наиболее известными являются оптимизация роя частиц (*Particle Swarm Optimization, PSO*) и искусственная пчелиная колония (*Artificial Bee Colony, ABC*); (2) алгоритмы, инспирированные механизмами добывания пищи и охоты наземных животных (14 %, наиболее известными являются алгоритм оптимизации колонии муравьев (*Ant Colony Optimization, ACO*), алгоритм оптимизации стаей серых волков (*Grey Wolf Optimizer, GWO*), львиный алгоритм оптимизации (*Lion Optimization Algorithm, LOA*), алгоритм оптимизации кузнечика (*Grasshopper Optimization Algorithm, GOA*); (3) алгоритмы, инспирированные водными животными (7 %, наиболее известными являются алгоритм стада криля (*Krill Herd, KH*), алгоритм оптимизации китов (*Whale Optimization Algorithm, WOA*); (4) алгоритмы, инспирированные микроорганизмами (4 %, наиболее известным является алгоритм поиска пищи бактериями (*Bacterial Foraging Algorithm, BFOA*); (5) другие роевые метаэвристики настолько различные, что их невозможно сгруппировать в представительные классы (6 %) [3].

Внутри каждой подкатегории также имеются определенные различия, например метафора связана с поисками роем пищи или же речь идет о модели движения роя.

Алгоритмы, основанные на физических (14 %) и химических (4 %) процессах, наиболее известные из которых алгоритм поиска гармонии (*Harmony Search, HS*), алгоритм интеллектуальных капель воды (*Intelligent Water Drops, IWD*), моделирования отжига

(*Simulated Annealing, SA*), характеризуются тем, что они имитируют, например, гравитационные силы, электромагнетизм, электрические заряды и движение воды, а также химические реакции и движение частиц газов [4].

Алгоритмы, основанные на когнитивных процессах и деятельности человека, инспирированы социальными и политическими концепциями, процессами принятия решений, конкуренцией идеологий внутри общества, спортивными соревнованиями, мозговым штурмом. Наиболее известными являются иммунный алгоритм (*Immune Algorithm, IA*), алгоритм империалистической конкуренции (*Imperialist Competitive Algorithm, ICA*), глобальный алгоритм оптимизации мозгового штурма (*Global Brainstorm Optimization Algorithm, GBSO*) [4].

Следующим классом метаэвристик по критерию соответствия природной метафоре являются эволюционные алгоритмы. Эволюционные алгоритмы основаны на принципах природной эволюции. Каждый агент в популяции представляет собой решение задачи и имеет соответствующее значение функции пригодности. В этих алгоритмах процесс воспроизводства и отбора повторяется в течение многих поколений, совокупность решений эволюционирует в направлении областей с более высокой пригодностью. Особенность селекции делает алгоритмы этого класса уникальными по сравнению с алгоритмами других категорий. Наиболее известными эволюционными алгоритмами являются генетический алгоритм (*Genetic Algorithm, GA*), алгоритм дифференциальной эволюции (*Differential Evolution, DE*), алгоритм эволюционной стратегии (*Evolutionary Strategy, ES*) [3].

Алгоритмы, основанные на особенностях многоклеточных организмов, способных к фотосинтезу, инспирированы растительным миром. В отличие от других метаэвристик, в них отсутствует связь между агентами. Одним из наиболее известных является алгоритм оптимизации лесов (*Forest Optimization Algorithm, FOA*), инспирированный процессом размножения растений.

В категорию прочих входят метаэвристики, которые не относятся ни к одной из перечисленных выше категорий. Они настолько различны, что их невозможно сгруппировать в представительные классы. Например, алгоритм Инь-Ян. Эта категория метаэвристик неоднородна, ее включение в классификацию, возможно, в будущем послужит основой для создания новых подкатегорий, развитию классификации и облегчит анализ достижений в этой области [5].

Структурный уровень. Он включает критерии, относящиеся к общей структуре метаэвристик. Вначале метаэвристики классифицируются на дискретные и траекторные. Далее, большинство дискретных метаэвристик являются популяционными, а траекторные обычно основаны на одном решении. На следующем шаге классификации проводится различие между метаэвристиками, основанными на локальном поиске и так называемыми метаэвристиками конструктивного поиска. На заключительном шаге структурной классификации используется критерий наличия/отсутствия памяти в процессе поиска. Структурный уровень классификации дает важную информацию о метаэвристике, однако недостаточную для однозначной классификации.

Поведенческий уровень. Согласно этому уровню классификации метаэвристики группируются по их поведенческим особенностям безотносительно от их природной метафоры. С этой целью необходим четкий критерий классификации. В качестве такого критерия используются механизмы для создания новых решений или для изменения существующих решений задачи оптимизации. Согласно этому критерию, вначале проводится различие между процессом получения новых решений: они создаются из уже имеющихся решений или путем дифференциально-векторного движения [6].

При дифференциально-векторном движении новые решения создаются путем сдвига или мутации предыдущего решения. Репрезентативными примерами этой категории метаэвристик являются *PSO* и *DE*.

При создании новых решений используются механизмы, которые либо рекомбинируют несколько решений, либо основаны на стигмергии путем спонтанного непрямого взаимодействия между индивидами. Репрезентативными примерами этой категории метаэвристик являются *GA* и *ACO*.

В результате более чем 400 метаэвристик вначале разделяются на два класса: (1) алгоритмы на основе дифференциально-векторного движения (66 %); (2) алгоритмы, основанные на создании новых решений (34 %).

В определении направления дифференциально-векторного движения может участвовать вся популяция решений (около 4 %), только определенные решения (около 55 %), только решения из некоторой окрестности или субпопуляции (около 7 %).

Среди алгоритмов, основанных на создании новых решений, около 32 % используют рекомбинацию нескольких решений, а 2 % – стигмергию.

Представим поведенческие характеристики метаэвристик из различных категорий.

Особенность категории метаэвристик дифференциально-векторного движения заключается в том, что вычисляется направление дифференциального вектора. Одним из возможных критериев является использование для этих целей всех агентов в популяции. В этих алгоритмах все агенты имеют определенную степень влияния на движение других решений. Такая степень обычно взвешивается в соответствии с разницей в функции пригодности или расстоянием между решениями. Примером здесь является светлячковый алгоритм (*Firefly Algorithms, FA*), в котором близкие к лучшему решению оказывают более сильное влияние, нежели более удаленные. Другим критерием класса метаэвристик с дифференциально-векторным движением является использование определенных репрезентативных решений. Чаще всего в этом качестве выбираются наилучшие решения, найденные алгоритмом. Примером является *PSO*. В этом оптимизаторе каждое решение или частица руководствуется глобальным текущим лучшим решением и лучшим решением, полученным этой частицей во время поиска. Другим примером в этой категории является семейство алгоритмов *DE*. В них наилучшее решение комбинируется с дифференциальным вектором для расширения пространства поиска решений. Еще одним критерием класса метаэвристик с дифференциально-векторным движением является использование решений из некоторой окрестности или субпопуляции. Примерами алгоритмов в этой категории являются алгоритм оптимизации бабочки монарха (*Monarch Butterfly Optimization, MBO*) и *BFOA.*

Особенность категории метаэвристик основанных на создании новых решений заключается в том, что они используют операторы рекомбинации или стигмергию. Наиболее распространенным вариантом является рекомбинация новых решений путем объединения некоторых из существующих решений с помощью оператора кроссинговера или путем комбинирования хороших решений. Самым популярным алгоритмом в этой категории являются эволюционный алгоритм *GA*, алгоритм *LOA*, алгоритм столкновения частиц (*Particle Collision Algorithm, PCA*). Наиболее распространенным вариантом создания

новых решений путем стигмергии является алгоритм *ACO*, моделирующий механизмом добывания пищи колонией муравьев [7].

Метаэвристики, имеющие схожую природную метафору, могут значительно отличаться друг от друга по поведенческому критерию. Примером являются алгоритмы эхолокации дельфина (*Dolphin Echolocation Algorithm, DEA*) и алгоритм дельфина (*Dolphin*). Оба инспирированы одним и тем же животным (дельфином) и его механизмом эхолокация для обнаружения рыбы, однако поведенческие механизмы у них разные: *DEA* создает новые решения путем комбинирования, в то время как *Dolphin* аналогичен *PSO*. Другим примером является класс метаэвристик с дифференциально-векторным движением, который содержит более половины рассмотренных алгоритмов (55%) и включает алгоритмы из всех различных классов по критерию соответствия природной метафоре: социальное поведение человека алгоритм оптимизации анархического общества (*Anarchic Society Optimization, ASO*), бактериальный алгоритм *BFOA*, алгоритм фейерверка (*Fireworks Algorithm, FWA*), алгоритм опыления цветов (*Flower Pollination Algorithm, FPA*).

На наш взгляд, поведенческий уровень классификации является наиболее информативным, а алгоритмы на основе дифференциально-векторного движения, входящие в этот класс, представляют более 60 % всех метаэвристик. По этой причине именно метаэвристики дифференциально-векторного движения будут подробно рассмотрены в последующих главах.

Поисковый уровень. Он связан со скоростью сходимости метаэвристики при поиске оптимального решения и диверсификацией пространства поиска решений. Важность поиска баланса между скоростью сходимости алгоритма и диверсификацией пространства поиска решений является фундаментальной задачей, однако пока отсутствуют инструменты измерения баланса. На баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска решений оказывают влияние механизм селекции, аттрактивность операторов поиска и число итераций алгоритма. Например, жадный механизм селекции приводит к преждевременной сходимости в точке локального оптимума. Аттрактивность операторов, используемых в метаэвристиках, означает дрейф имеющихся решений в направлении лучших решений, которые рассматриваются в качестве аттракторов. Этот механизм эффективен

для поддержания разнообразия популяции решений. Необходима метрика для оценки баланса между скоростью сходимости метаэвристики и диверсификацией пространства поиска решений.

Компонентный уровень. Он связан с особенностями базовых алгоритмических структур и механизмов. Метаэвристики, использующие механизм локального поиска, классифицируются по критериям поиска в окрестности, поиска восхождением к вершине, предотвращения преждевременной сходимости в локальных оптимумах, мультистарта и адаптивного программирования памяти. Метаэвристики, использующие эволюционный механизм, классифицируются по критериям использования популяционного поиска или поиск, при котором исследуется пространство между двумя или более наилучшими решениями. Метаэвристики, использующие механизм оптимизации роя частиц, классифицируются по критериям направленного поиска по перспективным направлениям (например, градиентным) и поиска в переменной окрестности. В настоящее время многие метаэвристики используют комбинацию сразу несколько указанных выше алгоритмических компонентов. Таким образом, компонентный классификационный уровень предоставляет информацию об их сходствах и различиях.

Специфический уровень. Он связан с особенностями и возможным расширением структуры метаэвристики. Поиск критериев, подходящих для данного уровня классификации, представляется несколько более сложным, нежели для предыдущих уровней, поскольку они должны быть связаны с возможностями фреймворка, т. е. программной платформы, определяющей структуру программного обеспечения, облегчающего разработку и объединение разных компонентов большого программного проекта. Некоторые метаэвристические фреймворки включают алгоритмы, в которых это расширение уже сделано. В других случаях расширение возможно, но не было включено или невозможно. Эта характеристика может быть использована в качестве критерия классификации. Также возможен поиск на этом уровне других критериев, основанных на особых или уникальных характеристиках метаэвристических фреймворков. Однако для этого требуются обширные знания о метаэвристиках, включая факторы, определяющие вычислительную сложность метаэвристики. Однако поиск критериев, подходящих для

этого уровня классификации, является серьезной проблемой и нуждается в дальнейшем исследовании. Однако при классификации метаэвристических алгоритмов этот уровень, на наш взгляд, важен для обеспечения детального различия.

Оценочный уровень. Он связан с эффективностью метаэвристик для решения конкретных задач оптимизации. Возможно, что подходящими классификационными критериями на этом уровне могут быть, например, структура задачи и ее размерность. Однако для этого необходимо знать эффективность метаэвристик для решения конкретных классов задач. Оценочный уровень классификации требует дальнейших исследований, чтобы стать применимым в общей системе классификации метаэвристик. Понятно, что знание о производительности метаэвристики для различных классов задач облегчает ее выбор для решения конкретной задачи, а также помогает определить, какие метаэвристики демонстрируют идентичные результаты на одних и тех же классах задач.

Проиллюстрируем применение многоуровневой классификации на примере трех метаэвристик в их базовых версиях: генетический алгоритм, эволюция $(1 + \lambda)$ и табуированный поиск [8], используя структурный, поведенческий, поисковый и компонентный уровни классификации.

Генетический алгоритм *GA* подходит для задач комбинаторной оптимизации, использует начальную популяцию в пространстве поиска решений, предполагает повторение следующих шагов до тех пор, пока не будет выполнен критерий остановки: (1) родители выбираются с использованием турнирного отбора; (2) потомство генерируется путем равномерного кроссинговера; (3) потомство мутирует, случайно инвертируя один бит в решении; (4) потомство оценивается и происходит замена поколений с учетом элитарного отбора. *GA* использует дискретную структуру, память, а также три различные стратегии поиска рекомбинацию, мутацию и отбор. На компонентном уровне оператор мутации использует поиск в окрестности, оператор кроссинговера использует промежуточный поиск на основе популяции, популяционный отбор, программируемую память и мультизапуск. Смена поколений в популяции происходит по схеме восхождения на вершину фитнес-ландшафта.

Эволюционная стратегия (1 + λ)-*ES* также, как и *GA*, относится к группе эволюционных алгоритмов, использует память и дискретную структуру, подходящую для комбинаторной оптимизации. В (1 + λ)-*ES* случайным образом инициализируется только один родительский элемент. Следующие шаги повторяются до тех пор, пока не будет выполнен критерий остановки: (1) λ потомков генерируются мутацией родителя; (2) потомство оценивается, и лучшая особь родителя и потомства выбирается в качестве родителя следующего поколения. В (1 + λ)-*ES* новые решения создаются путем адаптации ранее найденных решений, не используется оператор кроссинговера. На компонентном уровне популяция и оператор мутации используются для поиска в окрестности на основе популяции. Смена поколений в популяции происходит по схеме восхождения на вершину фитнес-ландшафта.

Табуированный поиск (*Tabu Search, TS*) подходит для решения задач комбинаторной оптимизации, использует кратковременную память при формировании списка запретов. Поиск начинается со случайно инициализированного решения, для которого создается список соседних решений. В табу список добавляется каждое просмотренное решение. *TS* предполагает повторение следующих шагов до тех пор, пока не будет выполнен критерий остановки: (1) на каждой итерации выбирается лучшее решение в окрестности текущего решения в качестве нового текущего решения, даже если это приводит к увеличению стоимости решения; (2) в кратковременной памяти, называемой списком табу, сохраняется недавно найденные решения, чтобы избежать зацикливания. Поиск прекращается после определенного числа итераций или если после ряда последовательных итераций не было достигнуто каких-либо улучшений в наилучшем известном решении. *TS* – это метаэвристика, основанная на траектории одного решения, которая использует локальный поиск и память. Его поведение основано на дифференциальном векторном движении в управляемой окрестности. На компонентном уровне список табу использует адаптивное программирование памяти, поиск восхождением на вершину в ландшафте фитнес-функции.

Если необходимо оценить скорость сходимости и возможности диверсификации пространства поиска указанных метаэвристик, например, при оптимизации мультимодальных целевых функций, то система классификации показывает, что *GA* предоставляет

больше возможностей для балансировки поиска, нежели *TS* и (1 + λ)-*ES*. При использовании соответствующего классификационного уровня эта оценка может быть улучшена.

1.2. Тестирование метаэвристик

Невозможно протестировать для оценки эффективности каждую метаэвристику на множестве конкретных задач. При этом следует иметь в виду *NFL*-теорему [9], согласно которой любая метаэвристика в среднем будет работать одинаково хорошо, как алгоритм случайного поиска, по всем возможным целевым функциям. Одна метаэвристика может оказаться предпочтительнее другой на конкретной задаче в зависимости от баланса между ее скоростью сходимости и возможностями диверсификации пространства поиска решений. Используя рандомизацию и факторный анализ, можно планировать эксперименты, выбирать комбинации параметров метаэвристик. При сравнении производительности метаэвристик следует использовать статистическую проверку гипотез на бенчмарках. В частности, нулевая гипотеза для похожих метаэвристик заключается в том, что они попадают в один и тот же класс. Если эта гипотеза не подтверждается, то метаэвристики разделяются на отдельные классы.

Для корректного тестирования метаэвристик необходимо единообразная нотация. Требуется стандартизированный язык, обеспечивающий их четкое и машиночитаемое описание. Между тем для большинства метаэвристик нет общедоступного исходного кода. Оценка и сравнение метаэвристик должны основываться на стандартизированной процедуре, включающей как теоретические принципы, так и эмпирические подходы. Теоретическое принципы помогают выделить отличительные особенности метаэвристики и определить ее новизну. Эмпирические подходы помогают стандартизировать соответствующие протоколы тестирования метаэвристик, их статистический анализ и выбор контрольных показателей. Разрабатываемые метаэвристики должны демонстрировать сокращение времени, затрат или сложности при решении оптимизационных задач, способность обрабатывать различные типы оптимизационных функций, демонстрировать сходимость, возможность распараллеливания.

Современные подходы к тестированию метаэвристик, чаще всего, основаны на эмпирических наблюдениях, собранных в результате моделирования, проведенного на тестовых задачах. Для

задач дискретной оптимизации используются задачи коммивояжера, о рюкзаке, об упаковке, транспортная задача, обучение параметров искусственной нейронной сети, планирование и составление расписаний, разработка игровых стратегий, задачи конструкторского проектирования микросхем [2].

Для задач непрерывной оптимизации чаще всего в качестве бенчмарок используются многомерные мультиэкстремальные математические функции, например:

- Гриванка:

$$F_{gri}(\boldsymbol{X}) = \frac{1}{40000}\sum_{i=1}^{n}(x_i - 100)^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i-100}{\sqrt{i}}\right) + 1, \quad (1.1)$$

- Растригина:

$$F_{rtg}(\boldsymbol{X}) = \sum_{i=1}^{n}\left[x_i^2 - 10cos(2\pi x_i) + 10\right], \quad (1.2)$$

- Розенброка:

$$F_{ros}(\boldsymbol{X}) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right], \quad (1.3)$$

- Швефеля:

$$F_{sch}(\boldsymbol{X}) = 418{,}9829n - \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right), \quad (1.4)$$

- Захарова:

$$f_{zach}(\mathbf{X}) = \sum_{i=1}^{n}(x_i)^2 + \left(\sum_{i=1}^{n} 0{,}5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0{,}5ix_i\right)^4, \quad (1.5)$$

- Саломона:

$$f_{sal}(\mathbf{X}) = -\cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0{,}1\sqrt{\sum_{i=1}^{n} x_i^2 + 1}, \quad (1.6)$$

- Цина:

$$f_{zin}(\mathbf{X}) = \sum_{i=1}^{n}(x_i^2 - i)^2. \quad (1.7)$$

Поиск глобального минимума каждой из этих тестовых функций является трудной задачей. В частности, «банановая» функция Розенброка имеет большое медленно убывающее плато, ее глобальный минимум функции находится внутри параболической сильно вытянутой поверхности. Функция Швефеля является мультиэкстремальной с «непредсказуемым» глобальным минимумом.

В качестве тестовых оптимизационных инженерных задач в современных исследованиях используются следующие три задачи оптимизации:

- растяжение/сжатие пружины с параметрами диаметр пружины, толщина пружины и длина пружины (рис. 1.1),

• проектирование сосуда высокого давления с параметрами толщина стенок, внутренний радиус и длина цилиндрического сечения (рис. 1.2),

• сварка балки с параметрами толщина сварного шва, длина, высота и толщина присоединяемой части стержня (рис. 1.3).

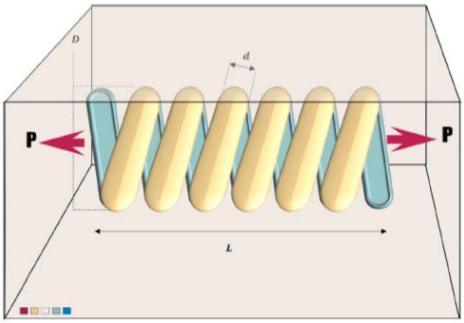

Рис. 1.1. Растяжение/сжатие пружины с параметрами: диаметр пружины (D), толщина пружины (d) и длина пружины (L)

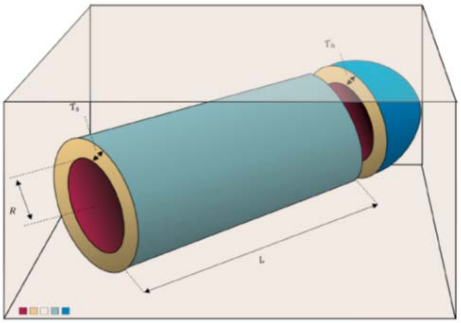

Рис. 1.2. Проектирование сосуда высокого давления с параметрами: толщина стенок сосуда (T_s), внутренний радиус (R) и длина цилиндрического сечения (L)

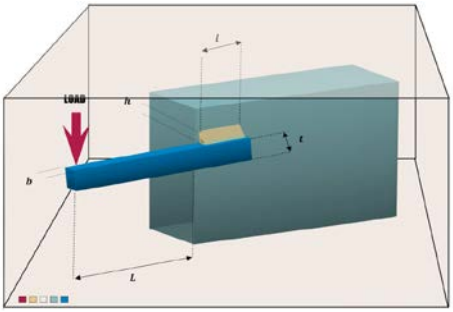

Рис. 1.3. Сварка балки с параметрами: толщина сварного шва (h), длина (L), высота (t) и толщина (l) присоединяемой части стержня

Сотни публикаций о новых метаэвристиках, их модификациях и приложениях появляются каждый год. Очевидно, что новые алгоритмы должны быть протестированы и валидированы с использованием различных задач. Однако большинство существующих бенчмарок являются функциональными тестами с известным оптимальным решением и регулярным пространством поиска в указанных границах переменных. Ограничения функциональных тестов, как правило, достаточно просты и существенно не меняют форму доменов поиска.

Полезность таких тестов нельзя преувеличивать, поскольку эти тестовые функции имеют мало общего с задачами оптимизации из реальных приложений. Однако тематические тесты, как правило, очень специализированы, а для правильной интерпретации результатов этих тестов необходимы специальные знания в предметной области. Имеет смысл попробовать некоторое подмножество задач оптимизации и попытаться сделать их практически независимыми от предметных областей.

Ниже представлены несколько тестовых задач со специальными свойствами, такими как шум, прерывистость, недифференцируемость, многослойные дискретные значения и другие.

В целом, специальные свойства тестовых функций можно разделить на следующие категории:

- гладкие тестовые функции, которых насчитывается более 200;
- составные функции, которых насчитывается более десятка, например функция Экли;

• неявно заданные функции, требующие использования пакетов анализа конечных элементов и вычислительной гидродинамики для оценки эффективности проектирования;

• функции с заданной структурой данных, например, в приложениях для молекулярной биологии;

• реальные задачи из практических областей.

Поэтому в дополнение к существующим тестовым функциям рекомендуется использовать более сложные функции для тестирования метаэвристик:

• зашумленные функции;

• недифференцируемые функции;

• функции с изолированными доменами;

• гиперболоидные функции;

• негладкие многослойные функции;

• бесконечномерные функции.

Зашумленные функции. Почти все существующие тестовые задачи являются детерминированными. Добавление шума к гладкой функции усложняет поиск ее оптимума. Одним из способов добавления шума, не влияя на местоположение оптимума, является умножение на случайную равномерно распределенную величину:

$$f(x) = \sum_{n=1}^{D} \varepsilon_n x_n^2, \quad (1.8)$$

где ε_n – величины, равномерно распределенные на интервале [0, 1], $-100 \le x_n \le 100$, $D \ge 1$ – размерность функции. Минимум функции $f_{min} = -1$ достигается при $x^* = (0, 0, \ldots, 0)$.

Недифференцируемые функции. Простейшей функцией с перегибом, вероятно, является $f(x) = |x|$, которая имеет глобальный минимум $f_{min} = 0$ при $x^* = 0$. В D-мерном пространстве функция имеет вид:

$$f_1(x) = \sum_{n=1}^{D} |x_n|. \quad (1.9)$$

Более сложным является вариант функции с несколькими перегибами:

$$f_2(x) = \left(\sum_{n=1}^{D} |x_n - n\pi|\right) \cdot \exp\left[-\sum_{n=1}^{D} |sin(x_n - n\pi|)|\right], \quad (1.10)$$

где $-D\pi \le x_n \le D\pi$. Минимум функции $f_{min} = 0$ достигается при $x^* = (\pi, 2\pi, \ldots, n\pi)$.

Функции с изолированными доменами. В большинстве тестовых функций области поиска варьируется в некотором интервале $x_n \in [a, b]$. Для задач оптимизации с ограничениями допустимые

области поиска могут иметь нерегулярную форму или даже изолированные домены. В качестве примера приводится функция с двумя изолированными доменами:

$$f(x) = x_1^2 + \sum_{n=1}^{D} |x_n^3| \quad (1.11)$$

при условиях $|x_1 - 2a| + \sum_{n=1}^{D} |x_n| \le a,\ \sum_{n=1}^{D} (x_n - 5a)^2 \le a^2$, $a \ge 1$ (1.12).

Минимум функции $f_{min} = a^2$ достигается при $x^* = (a, 0, \dots, 0)$.

Гиперболоидные функции. Сферическая функция может быть расширена до гиперболоидной функции:

$$f(x) = \sum_{n=1}^{D} x_n^2 \quad (1.13)$$

при условии:

$$\sum_{n=1}^{D-1} \frac{x_n^2}{a^2} \ge \frac{x_D^2}{b^2} + 1,\ \ a, b \ge 1. \quad (1.14)$$

Минимум функции $f_{min} = a^2$ достигается на (D – 1)-мерной гиперсфере $\sum_{n=1}^{D-1} x_n^2 = a^2$, что соответствует бесконечно большому числу решений.

Негладкие многослойные функции. К ним относятся функции с прерывистыми ландшафтами. Например, можно тестировать метаэвристики на сферической функции, переменные которой принимают целочисленные значения:

$$f(x) = \left\lfloor \sum_{n=1}^{D} x_n^2 \right\rfloor, \quad (1.15)$$

Минимум функции $f_{min} = 0$ достигается внутри гиперсферы $\sum_{n=1}^{D} x_n^2 = 1$. Любая точка внутри этой гиперсферы является оптимальным решением. Таким образом, эта функция может иметь бесконечно много оптимальных решений в пределах гиперобъёма, и все решения внутри этой области имеют одинаковое оптимальное значение $f_{min} = 0$. Кажется, что метаэвристике проще найти оптимальное решение; однако многие статистические показатели (математическое ожидание, стандартное отклонение, используемые для сравнения, не имеют особого смысла для этой функции. Тестовые бенчмарки подобного вида могут иметь несколько оптимальных изолированных областей.

Бесконечномерные функции. Все функциональные тесты, как правило, имеют определенное множество оптимальных решений. Однако, например, в вариационном исчислении оптимальными решениями могут быть кривые и поверхности. Такого рода задачи могут оказаться серьезной проблемой для метаэвристик.

Рассмотрим, например, задачу о кратчайшем пути. Пусть в двумерном пространстве (*x*, *y*) существует траектория или кривая *y*(*x*), которая минимизирует интеграл:

$$minQ = \int_0^1 \sqrt{1 + (\frac{dy}{dx})^2}dx. \quad (1.16)$$

Решение представляет собой прямую линию *y* = *x* из начала координат (0, 0) до точки (1, 1). Сложность задачи оптимизации заключается в том, что решением является не отдельная точка, а отрезок. Чтобы найти это решение (соответствующее бесконечному числу точек), необходима некоторая параметризация неизвестной кривой.

1.3. Области применения метаэвристик

На практике наиболее распространенный вопрос заключается в том, какая метаэвристика лучше всего подходит и как ее можно успешно применить для решения конкретной задачи? Ответ на этот вопрос – непростая задача. В зависимости от требований по точности может существовать несколько метаэвристик, дающих приемлемые решения. Проблема выбора метаэвристики включает в себя пространство задач *P*, пространство алгоритмов *A* и отображение *P*×*A* на модель эффективности *R*.

Известны рекомендации по выбору метаэвристик. Они включают выбор критериев и проверку результатов путем статистического анализа и визуализации, а также настройку параметров алгоритмов. С этой целью рекомендуются три указанных выше инженерные оптимизационные задачи: растяжение/сжатие пружины, проектирование сосуда высокого давления, сварка балки. Их использовали для определения наиболее эффективных метаэвристик.

Для задачи растяжения/сжатия пружины сравнивались метаэвристики: алгоритм обезьян (*Monkey Algorithm, MA*), алгоритм оптимизации африканского грифа (*African Vulture Optimization Algorithm, AVOA*), алгоритм оптимизации бабочек (*Butterfly Optimization Algorithm, BOA*), алгоритм оптимизации императорских пингвинов (*Emperor Penguin Optimization, EPO*), алгоритм оптимизации черной крачки (*Sooty Tern Optimization Algorithm, STOA*), алгоритм летучих мышей (*Bat Algorithm, BA*), алгоритм следопыта (*Pathfinder Algorithm, PFA*), алгоритм морского хищника (*Marine Predator Algorithm, MPA*), алгоритм роя хамелеонов (*Chameleon Swarm Algorithm, ChSA*), алгоритм оптимизации ястреба Харриса (*Harris Hawks Optimization, HHO*), алгоритм оптимизации серых

волков (*GWO*), алгоритм мотылька, летящего на свет (*Moth-Flame Optimization, MFO*), алгоритм оптимизации пятнистой гиены (*Spotted Hyena Optimizer, SHO*), алгоритм оптимизации китов (*WOA*), алгоритм роя сальп (*Salp Swarm Algorithm, SSA*).

Для задачи проектирования сосуда высокого давления сравнивались метаэвристики: алгоритм орлиной охоты (*Aquila Optimization, AO*), *AVOA, EPO, STOA, FA, MPA, ChSA*, алгоритм стрекозы, основанный на памяти (*Memory Based Dragonfly Algorithm, MHDA*), *HHO, GWO, MFO, SHO, WOA*, алгоритм стрекозы (*Dragonfly Algorithm, DA*).

Для задачи сварки балки сравнивались метаэвристики *AO, AVOA, BOA, MHDA*, алгоритм формирования шлама (*Slime Mould Algorithm, SMA*), *DA*, алгоритм роя оболочников (*Tunicate Swarm Algorithm, TSA*), алгоритм оптимизации чайки (*Seagull Optimization Algorithm, SOA*), *EPO, ABC, ChSA, MFO, MPA, SHO, GWO, WOA, HHO*, алгоритм оптимизации рыбы-паруса (*Sailfish Optimizer, SFO*).

Чтобы определить, какой алгоритм лучше всего подходит для этих инженерных задач, был применен подход, основанный на принципе Кондорсе для определения победителя при голосовании за кандидатов [10]. В этом контексте сравниваемые алгоритмы представляют кандидатов, а их решения для каждой из задач указывают на избирателей. Согласно принципу Кондорсе победителем на выборах объявляется кандидат, который превосходит остальных при парном сравнении. По результатам работы сравниваемых алгоритмов на трех инженерных задачах-бенчмарках были определены четыре лучших метаэвристики: алгоритм императорских пингвинов (*EPO*, 45 голосов), алгоритм охоты птицы-хищника (*AO*, 42 голоса), алгоритм хамелеона (*ChSA*, 34 голоса) и алгоритм африканских стервятников (*AVOA*, 32 голоса).

В последние годы метаэвристики стали активно применяться для решения широкого спектра задач. В интеллектуальных сетях метаэвристики используются для прогнозирования пиковой нагрузки, внезапных колебаний выходной мощности, обнаружения вторжений в сеть и оптимального планирования будущих требований клиентов. В сфере здравоохранения метаэвристики применяются в клинических исследованиях, персонализированной медицине, идентификации заболеваний и прогнозировании вирусных вспышек. В производстве, транспорте и промышленности применение метаэвристик в сочетании с другими методами машинного обучения выявило большие преимущества в управлении беспилотным транспортом, в использовании

виртуальных помощников, в обнаружении неисправностей технических устройств, в поддержке принятия решений и оперативном управлении. В торговле метаэвристические алгоритмы машинного обучения нашли применение для прогнозирования поведения клиентов, интеллектуального складирования и инвентаризации, прогнозирования спроса и оптимизации цен. Интернет-платформы социальных сетей в настоящее время используют метаэвристики в сочетании с другими алгоритмами машинного обучения для персонализированной рекламы, обнаружения мошеннических учетных записей. В образовании метаэвристики используется для анализа успеваемости обучающихся и моделирования их поведения.

В табл. 1.2 представлены современные области применения различных классов метаэвристик для решения конкретных оптимизационных задач.

Таблица 1.2

Области применения метаэвристик

Область применения	Метаэвристики	Задачи
1	*2*	*3*
Инженерное проектирование	*GSA, SA, GP, WOA*	Проектирование антенн; оптимизация авиационных конструкций; выбор параметров обработки деталей.
Обработка изображений и компьютерное зрение	*ACO, PSO, BA, ABC, BMO, CSA, EPO*	Сегментация изображений, обнаружение областей интереса на цифровых изображениях, цветовая экстракция.
Компьютерные сети и коммуникации	*GA, ABC, PSO, CS, EPO, MHSA, MBO, MMA*	Оптимальное распределение сети сенсорных датчиков, обнаружение сообществ в сети, обнаружение вредоносных *URL*-адресов и спама, криптоанализ, задача коммивояжера.
Энергетика и энергоменеджмент	*PSO, SSO, GWO, KHA, BBO, WOA, FA, CSA, HHO, MFO*	Системы хранения энергии, управление домашней энергоустановкой, регулирование нагрузки в энергосистеме, управление мощностью, диспетчеризация, реконфигурация, оптимальное размещение конденсаторов, ветрогенераторов.

Окончание таблицы 1.2

1	*2*	*3*
Анализ данных и машинное обучение	*WOA, GWO, SA, FA, PSO, GA, GSA, ABC, GOA, ALO, RIO*	Выбор информативных признаков для классификации объектов и ранжирования многомерных данных, кластеризация данных, обучение сверточных нейросетей, настройка параметров в *SVM*-методе машинного обучения.
Робототехника	*GSA, SA, GP, WOA*	Планирование и оптимизация траектории роботов, автономная навигация полета БПЛА, разработка контроллеров для роботизированных платформ.
Медицинская диагностика	*ACO, PSO, BA, ABC, BMO, CSA, EPO, PBA*	Обработка медицинских изображений, прогнозирование заболеваний путем анализа больших наборов данных, диагноз онкологии, болезни Паркинсона с использованием данных микрочипов.
Информатика и другие области	*BA, SHO, LA, ESA, MPA, BBO, BNSS, FOA, DFO*	Оценка подверженности наводнениям, планирование работы магазина, оптимизации каркасных конструкций, добычи природного газа, оценка токсичности лекарственного средства, регулирование орошения, подводная акустическая классификация, укрепление почвы, оптимизация протонообменных процессов в топливных элементах.

В качестве примера приведем краткое описание задач интеллектуального анализа данных, машинного обучения и энергетики, решенных в последние годы с использованием упомянутых метаэвристик.

Объект исследования – аккумуляторное оборудование, батареи и накопители тепловой энергии, имеющие большое значение при переключении пиковых нагрузок в энергетических системах. Для решения задачи оптимизации графиков работы энергетических систем, включающих батарею, воздушный тепловой насос, применяются метаэвристики, такие как генетический алгоритм (*GA*), опти-

мизация роя частиц (*PSO*) и поиск кукушки (*CS*). Показана эффективность метаэвристик при оптимизации режимов работы энергосистем (*DOI: 10.1016/j.apenergy.2015.04.029*).

Метаэвристика серых волков (*GWO*) применяется для оптимизации систем электроснабжения. Алгоритм тестируется на 23 тестовых функциях и применяется для подключенного к сети синхронного генератора, приводимому в действие ветровой турбиной. Результаты сравниваются с конкурирующими алгоритмами. Предлагаемый подход значительно выигрывает в производительности (*DOI: 10.1016/j.asoc.2018.05.006*).

Рассматривается задача оптимального распределения реактивной мощности как задача минимизации целевой функции, представляющей общие потери активной мощности в электрических сетях. Ограничения связаны с напряжением генератора, коэффициентами отводящих регулирующих трансформаторов и количеством реактивных шунтирующих компенсаторов. Цель исследования – найти наилучший вектор управляющих переменных, чтобы можно было реализовать снижение потерь мощности. Применяется биоэвристика охоты на горбатых китов с помощью пузырьковой сети (*WOA*). Приводится сравнение с алгоритмом роя частиц (*DOI: 10.1016/j.epsr.2017.09.001*).

Решается задача оптимального распределения реактивной мощности. Используется метаэвристика оптимизации пламени мотылька (*MFO*). Исследуется наилучшая комбинация управляющих переменных, включая напряжение генераторов, настройку отводов трансформаторов. Реактивные компенсаторы подбираются таким образом, чтобы обеспечить минимальные общие потери мощности и минимальное отклонение напряжения. Эффективность алгоритма *MFO* сравнивается с другими методами оптимизации. Статистический анализ показал конкурентоспособные результаты, обеспечивая меньшие потери мощности и меньшее отклонение напряжения, нежели конкурирующие подходы (*DOI: 10.1016/j.asoc.2017.05.057*).

Объект исследования – домашние системы управления энергопотреблением для минимизации затрат. Предлагается гибридная метаэвристика алгоритма дифференциальной эволюции (*DE*) и алгоритма оптимизации дождевого червя (*Earthworm Algorithm, EWA*). Моделирование показало, что *EWA* работает лучше с точки зрения снижения затрат, а *DE* – с точки зрения снижения пикового отношения к среднему (*DOI: 10.1007/978-3-319-93554-6_2*).

Применяется алгоритм охоты пауков (*SSA*) для повышения энергоэффективности поездов. Эксперименты показали, что предлагаемые алгоритмом планы энергосбережения на железной дороге могут помочь машинистам локомотивов в планировании движения поездов (*DOI: 10.1016/j.asoc.2015.02.014*).

Интеллектуальный анализ правил ассоциации, направленный на обнаружение правил ассоциации, которые удовлетворяют заранее определенной минимальной поддержке и достоверности из данной базы данных. Метаэвристика основана на оптимизации миграции животных для уменьшения количества ассоциативных правил: из данных удаляются правила, которые не имеют высокой поддержки и не нужны (*DOI: 10.1016/j.knosys.2018.04.038*).

SVM считается одним из самых мощных методов машинного обучения. Предлагается гибридный подход, основанный на алгоритме оптимизации кузнечиков (*GOA*). Цель подхода – оптимизировать параметры модели *SVM* и одновременно найти подмножество лучших функций. Результаты экспериментов показали, что предложенный подход превосходит конкурирующие алгоритмы с точки зрения точности классификации при минимизации количества признаков (*DOI: 10.1007/s12559-017-9542-9*).

Кластеризация текста – это процесс группировки значительных объемов текстовых документов в кластеры с релевантными документами. Для кластеризации текста используется алгоритм стада криля. Проверка алгоритма проводилась с использованием таких показателей как точность, полнота, *F*-мера и энтропия. (*DOI: 10.1007/s10489-018-1190-6*).

Кроме того, существует большое количество программных фреймворков для эволюционных и роевых алгоритмов на разных языках, таких как *C++, Java, Matlab* и *Python*. Например, фреймворк *Evolutional Computation Framework* (*http://ecf.zemris.fer.hr/*) и *ParadisEO* (*https://en.wikipedia.org/wiki/Paradiseo*) на *C++*; *jMetal* (*https://jmetal.sourceforge.net/)* на *Java, jMetalPy* на *Python* (*https://pypi.org/project/metal-python/*); *PlatEMO* в *Matlab* (*https://github.com/BIMK/PlatEMO*) и другие.

1.4. Резюме

Машинное обучение. Это одно из сложных и перспективных направлений в искусственном интеллекте, которое исследует алгоритмы, способные обучаться и самостоятельно улучшаться по мере накопления опыта, извлекать закономерности из данных и делать прогнозы на их основе.

Деревья принятия решений. Структура деревьев включает «листья» и «ветки». На ветках дерева решения записываются признаки, от которых зависит целевая функция, в листьях записываются значения целевой функции, а в остальных узлах – признаки, по которым различаются примеры. Чтобы классифицировать новый пример, надо спуститься по дереву до листа и выдать соответствующее значение. Метод деревьев принятия решений работает с категориальными и с интервальными переменными. Получения оптимального дерева решений является *NP*-полной задачей.

Поиск ассоциативных правил. Направлен на обнаружение интересующих нас связей между переменными при анализе данных. Алгоритмы находят ассоциации на новых неклассифицированных данных, часто встречающихся наборах объектов.

Обучение искусственной нейронной сети. Нейронная сеть является методом в области искусственного интеллекта, который учит компьютеры работать с данными так же, как человеческий мозг. Нейросеть – это не мыслящий объект, наделенный сознанием. Это сложнейшая база данных с огромным количеством формул, инструмент нелинейного статистического моделирования взаимосвязей между входными и выходными данными для поиска закономерностей в данных и решения других задач. Данные поступают в нейросеть, обрабатываются формулами, после чего пользователю выдается результат. Сложность заключается в том, как найти такие уравнения и алгоритмы, благодаря которым результат работы нейросети будет максимально полезным. Обучение нейросети как раз и заключается в выведении этих формул. Нейросети дают большой массив правильно решенных задач, после чего она сама пытается решить эти задачи, начиная угадывать, какой ответ от нее хотят получить. Алгоритм обучения подсказывает ей, правильно она справилась с решением или нет. Со временем нейросеть учится угадывать все лучше и лучше, формирует некие связи внутри своей структуры, которые обеспечивают полезный результат.

Индуктивное логическое программирование. Использует логику для представления примеров, фоновых знаний и гипотез. Получив описания уже известных фоновых знаний и множества примеров, представленных как логическая база фактов, ИЛП порождает логическую программу в форме гипотез, объясняющую все положительные примеры и ни одного отрицательного. Обычно реализации ИЛП делаются на языке *Prolog*.

Машина опорных векторов. Является линейным классификатором. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Чем больше расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора. Алгоритм строит модель, которая предсказывает, попадает ли новый пример в тот или иной класс.

Кластерный анализ. Это многомерная статистическая процедура, выполняющая сбор данных, содержащих информацию о выборке объектов, и затем упорядочивающая объекты в сравнительно однородные группы в соответствии с некоторыми заранее определенными критериями. Задача кластеризации относится к классу задач обучения без учителя. Алгоритмы кластеризации отличаются разными предположениями о структуре данных согласно некоторой метрики сходства между данными одного и того же кластера и разделению на различные кластеры.

Байесовская сеть. Это вероятностная модель, представляющая собой множество переменных и их вероятностных зависимостей согласно теореме Байеса. Представляет собой ориентированный ациклический граф, каждой вершине которого соответствует случайная переменная, а дуги графа кодируют отношения условной независимости между этими переменными. Вершины могут представлять переменные, параметры или гипотезы. Существуют эффективные методы, которые используются для вычислений и обучения байесовских сетей.

Метаэвристики, инспирированные природой. Так называются математические преобразования, трансформирующие входной поток информации в выходной и основанные на правилах имитации биоинспирированных механизмов, на процедурах, содержащих элементы стохастической оптимизации со случайными переменными. С помощью метаэвристик, инспирированных природой, ис-

следуется пространство поиска, синтезируются решения, являющиеся точками этого пространства, запрашивается оценка их качества или «приспособленности», которая затем используется для осуществления «естественного отбора». Тем самым метаэвристики обучаются тому, какие области пространства поиска содержат наиболее приспособленных особей. Метаэвристики обычно разрабатываются на основе наблюдений за природными процессами (например, процессы эволюции в природе, физические или химические процессы, когнитивные и общественные процессы) или паттернами поведения биологических организмов (роевой интеллект).

Классификация. Обозначает разновидность деления объема понятия по определенному основанию (признаку, критерию), при котором объем родового понятия (класс, множество) делится на виды (подклассы, подмножества), а виды, в свою очередь делятся на подвиды и т. д. Классификация как процесс предполагает упорядоченное и систематическое отнесение каждого понятия к одному и только одному классу в рамках системы взаимоисключающих и неперекрывающихся классов.

Категоризация. Обозначает деятельность по распознаванию общих черт или сходств между объектами и их отнесения к некоторой группе на основе признаков сходства. Категоризация подразумевает процесс разделения объектов на группы, объекты которых в некотором роде похожи друг на друга. В отличие от классификации, при категоризации объект может быть частью более чем одной категории, в зависимости от контекста.

Классификация метаэвристик по уровню природной метафоры. Этот уровень включает следующие метаэвристики: роевые алгоритмы; алгоритмы, основанные на физических и химических процессах; алгоритмы, основанные на когнитивных процессах и деятельности человека; эволюционные алгоритмы; алгоритмы, основанные на особенностях организмов, способных к фотосинтезу; прочие алгоритмы.

Классификация метаэвристик по структурному уровню. Он включает критерии, относящиеся к общей структуре метаэвристик: дискретные/траекторные, популяционные/одиночные и метаэвристики с памятью/без памяти.

Классификация метаэвристик по поведенческому уровню. Согласно этому уровню классификации метаэвристики группируются по

их поведенческим особенностям безотносительно от их природной метафоры. В качестве критерия классификации используются механизмы для создания новых решений. Этот уровень включает метаэвристики, в которых новые решения создаются из уже имеющихся решений или путем дифференциально-векторного движения.

Классификация метаэвристик по поисковому уровню. Этот уровень классификации связан со скоростью сходимости метаэвристики при поиске оптимального решения и диверсификацией пространства поиска решений. На баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска решений оказывают влияние механизм селекции, аттрактивность операторов поиска и число итераций алгоритма.

Классификация метаэвристик по компонентному уровню. Этот уровень классификации связан с особенностями используемых алгоритмических структур и механизмов: поиск в окрестности, поиск восхождением к вершине, предотвращение преждевременной сходимости в локальных оптимумах, мультистарт, адаптивное программирование памяти.

Классификация метаэвристик по специфическому уровню. Этот уровень классификации связан с особенностями и возможным расширением структуры метаэвристики, например, за счет фреймворков. Поиск критериев, подходящих для этого уровня классификации, нуждается в дальнейшем исследовании.

Классификация метаэвристик по оценочному уровню. Этот уровень классификации связан с эффективностью метаэвристик для решения конкретных задач оптимизации. Классификационными критериями на этом уровне могут быть, например, производительность метаэвристики, структура задачи, ее размерность.

Тестирование метаэвристик. Для задач дискретной оптимизации используются бенчмарки следующих задач: о коммивояжере, о рюкзаке, об упаковке, транспортная задача, обучение параметров искусственной нейронной сети, планирование и составление расписаний, разработка игровых стратегий, задачи конструкторского проектирования микросхем. Для задач непрерывной оптимизации в качестве бенчмарок используются многомерные математические функции, в том числе со специальными свойствами, такими как шум, прерывистость, недифференцируемость, с изолированными

доменами, многослойные и другие. В качестве тестовых оптимизационных инженерных задач в современных исследованиях используются следующие три задачи оптимизации: растяжение/сжатие пружины, проектирование сосуда высокого давления, сварка балки, присоединяемой к стержню.

Области применения метаэвристик. В инженерном проектировании для задач проектирования антенн, оптимизации авиационных конструкций, выбора параметров обработки деталей. При обработке изображений и в компьютерном зрении для задач сегментации изображений, обнаружения областей интереса на цифровых изображениях, цветовой экстракции. В компьютерных и телекоммуникационных сетях для задач оптимального распределения сенсорных датчиков, обнаружения сообществ в сети, обнаружения вредоносных URL-адресов и спама, криптоанализа, коммивояжера. В энергетике и энергоменеджменте для задач хранения энергии, управления домашней энергоустановкой, регулирования нагрузки в энергосистеме, управления мощностью, диспетчеризации, реконфигурации, оптимального размещения конденсаторов, ветрогенераторов. При анализе данных и машинном обучении в задачах выбора информативных признаков для классификации объектов и ранжирования многомерных данных, кластеризации данных, обучения сверточных нейросетей, настройке параметров в SVM-методе машинного обучения. В робототехнике для задач планирования и оптимизации траектории роботов, автономной навигации полета БПЛА, разработке контроллеров для роботизированных платформ. В медицинской диагностике для задач обработки медицинских изображений, прогнозирования заболеваний, диагнозе онкологии, болезни Паркинсона. В информатике и некоторых других областях для задач оценки наводнений, планирования работы магазина, оптимизации каркасных конструкций, добычи природного газа, оценки токсичности лекарственного средства, регулирования орошения, подводной акустической классификации, укрепления почвы, оптимизации протонообменных процессов в топливных элементах.

2. МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ, ИСПОЛЬЗУЮЩИЕ РЕПРЕЗЕНТАТИВНЫХ АГЕНТОВ ПОПУЛЯЦИИ

2.1. Алгоритм дифференциальной эволюции

Метаэвристика дифференциальной эволюции *DE*, предложенная Р. Сторном и К. Прайсом в 1996 г. [11], наряду с *GA*, является одной из самых популярных вычислительных моделей. В его базовом виде алгоритм можно описать следующим образом.

Вначале инициализируется множество случайных векторов (популяция), представляющих собой возможные решения задачи оптимизации. Число векторов в каждом поколении одно и то же и является одним из параметров настройки алгоритма.

На каждом шаге эволюционного процесса алгоритм генерирует новое поколение векторов, случайным образом комбинируя между собой векторы предыдущего поколения. На очередном *k*-шаге алгоритма *DE* применяются операторы мутации, кроссинговера и селекции, чтобы позволить популяции решений $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ "эволюционировать" к оптимальному решению.

При выполнении операции мутации новые решения (мутантные векторы) $\boldsymbol{m}_i^k = [m^k_{i,1}, m^k_{i,2}, ..., m^k_{i,\mathrm{d}}]$ генерируются для каждого отдельного индивидуума $\boldsymbol{x}_i$ по формуле:

$$\boldsymbol{m}_i^k = \boldsymbol{x}_{r3}^k + F(\boldsymbol{x}_{r1}^k - \boldsymbol{x}_{r2}^k), \tag{2.1}$$

где $r_1, r_2, r_3 \in \{1, 2, ..., N\}$ $(r_1 \neq r_2 \neq r_3 \neq i)$ – случайным образом выбранный индекс, $F \in [0, 2]$ – дифференциальный вес, используемый для управления разностью величин $\left(\boldsymbol{x}_{r1}^k - \boldsymbol{x}_{r2}^k\right)$. Иными словами, создание нового решения требует применения линейной операции к выбранным элементам с использованием параметра *F*.

При выполнении оператора кроссинговера формируется вектор пробного решения $\boldsymbol{u}_i^k = [u^k_{i,1}, u^k_{i,2}, ..., u^k_{i,\mathrm{d}}]$ для *i*-го индивида популяции решений. Компоненты $u^k_{i,j}$ в пробном векторе формируются из решения $\boldsymbol{x}^k_i$ и соответствующего ему решения $\boldsymbol{m}^k_i$ после выполнения операции мутации следующим образом:

$$u_{i,j}^k = \begin{cases} m_{i,j}^k, \text{если } (rand \leq CR) \text{ или } (n = n^*) \\ x_{i,j}^k, \text{ если } (rand > CR), \end{cases} \tag{2.2}$$

где $n^* \in \{1, 2, \ldots, d\}$ – случайно выбранный индекс, *rand* – случайное число из интервала [0, 1], $CR \in [0, 1]$, которое используется в алгоритме *DE* при выполнении кроссинговера для формирования элементов $u^k_{i,j}$ из элементов $x^k_{i,j}$ либо $m^k_{i,j}$. При выполнении селекции фитнесс-функция пробного решения $\boldsymbol{u}_i^k$ сравнивается с фитнесс-функцией решения-кандидата $\boldsymbol{x}_i^k$ с использованием жадного критерия.

На каждой шаге эволюционного процесса или с заданной периодичностью определяется лучший вектор в популяции с целью контроля скорости поиска оптимального решения. Условиями окончания алгоритма могут быть следующие:

- исчерпано заданное предельное количество итераций;
- исчерпано заданное предельное физическое расчётное время;
- значение критерия оптимизации лучшего вектора поколения не изменяется на протяжении заданного предельного количества итераций алгоритма;
- достигнуто удовлетворительное значение критерия оптимизации.

В большинстве случаев применения алгоритма дифференциальной эволюции *DE* для решения задачи многомерной оптимизации рекомендуется принимать число агентов в популяции приблизительно в 10 раз больше количества оптимизируемых переменных. Выбор параметров *F* и *CR* осуществляется эмпирически, поскольку во многом зависит от рельефа поверхности критерия оптимизации.

Алгоритм прост в реализации, содержит мало управляющих параметров, требующих подбора, легко распараллеливается. Известно, что поисковая система Яндекс использует алгоритм дифференциальной эволюции для улучшения своих алгоритмов ранжирования.

Однако успех *DE* в решении конкретной задачи в решающей степени зависит от правильного выбора стратегий генерации пробных векторов и связанных с ними значений управляющих параметров. Использование схемы проб и ошибок для поиска наиболее подходящей стратегии и связанных с ней настроек параметров требует больших вычислительных затрат. Более того, на разных этапах эволюции могут потребоваться разные стратегии в сочетании с разными настройками параметров для достижения наилучшей производительности. Таким образом при практическом применении алгоритма *DE* желательно постепенно адаптировать стратегии генерации пробных векторов и связанные с ними значения параметров управления.

В этом случае производительность алгоритма *DE* будет выгодно отличаться от базового *DE* и его современных модификаций.

2.2. Синус-косинусный алгоритм

Синус-косинус (*Sine Cosine Algorithm, SCA*) является относительно новым популяционным алгоритмом [12], основанным на вычислении направления дифференциального вектора. *SCA* генерирует различные исходные случайные решения и обновляет их в соответствии с уравнениями, используя математическую модель, основанную на функциях синуса и косинуса. В *SCA* сочетаются популяционный подход и стратегия локального поиска, простота реализации, гибкость и возможности применения для решения различных типов задач оптимизации, таких как планирование, проектирование сетей, диспетчирование электроэнергии, обработка изображений и др.

Сначала случайным образом инициализируется популяция агентов $X_i = (X_{i1}, X_{i2}, \dots, X_{id})$ с использованием следующего уравнения:

$$X_{ij} = X_{ij}^{\text{нг}} + rand() \times \left(X_{ij}^{\text{вг}} - X_{ij}^{\text{нг}}\right), j = 1..d, i = 1..N_P, \quad (2.3)$$

где X_{ij} представляет j-ю размерность i-го решения, $X_{ij}^{нг}$ и $X_{ij}^{вг}$ обозначают нижнюю границу и верхнюю границу i-го решения в j-м измерении пространства поиска соответственно, функция *rand ( )* генерирует равномерно распределенные случайные числа в диапазоне [0, 1], а N_p обозначает количество поисковых агентов в популяции, т. е. размер популяции.

Следующим шагом после инициализации совокупности в пространстве поиска является обновление позиции каждого агента для поиска оптимального решения. С этой целью положение агента оценивается с использованием целевой функции, и на основе критериев оптимизации каждому агенту присваивается значение пригодности или полезности. Поисковый агент с наибольшей пригодностью считается лучшим агентом, а его позиция обозначается как пункт назначения. После определения местоположения пункта назначения другие агенты поиска обновляют свое местоположение в пространстве поиска, используя пункт назначения в качестве ориентира.

Следующие уравнения являются уравнениями обновления положения:

$$X_{ij}^{t+1} = X_{ij}^{t} + r_1 \times \sin(r_2) \times |r_3 \times P_g^t - X_{ij}^t|, \quad (2.4)$$

$$X_{ij}^{t+1} = X_{ij}^{t} + r_1 \times \cos(r_2) \times |r_3 \times P_g^t - X_{ij}^t|, \quad (2.5)$$

где $j = 1..d$, $i = 1..N_p$, $X_i^t = (X_{i1}^t, X_{i2}^t, \dots, X_{id}^t)$ обозначает позицию i-го поискового агента на t-й итерации; $P_g^t = (P_{g1}^t, P_{g2}^t, \dots, P_{gd}^t)$ является g-м поисковым агентом, который имеет наилучшую пригодность и рассматривается в качестве точки назначения на t-й итерации; $|\,|$ – модуль числа.

Параметры r_1 r_2, r_3 определяются следующим образом:

$$r_1 = b - b \times (\frac{t}{t_{max}}), \quad (2.6)$$

$$r_2 = 2 \times \pi \times rand(), \quad (2.7)$$

$$r_3 = 2 \times rand(). \quad (2.8)$$

Здесь r_1 – некоторая функция итераций t, вычисляемая с использованием (2.6), где b – константа, а t_{max} обозначает максимальное количество итераций; r_2 и r_3 – равномерно распределенные случайные числа, сгенерированные с использованием (2.7) и (2.8) соответственно.

В алгоритме *SCA* управляющий параметр r_1 регулирует поддержание баланса между скоростью сходимости и диверсификацией пространства поиска решений, обеспечивая переход от фазы диверсификации к фазе сходимости во время поиска. Он является убывающей функцией счетчика итераций t, который линейно уменьшает значение константы b. Тригонометрические функции синус и косинус в (2.4) и (2.5) умножаются на r_1. Значение r_1 зависит от константы b, управляя которой, алгоритм *SCA* изменяет величину $r_1 \times \sin(r_2)$ и $\cdot r_1 \times \cos(r_2)$.

По сути, параметр r_1 работает как масштабирующий коэффициент для размера шага в (2.4) и (2.5). На ранних итерациях алгоритм *SCA* использует большие значения r_1 для расширения пространства перемещений поисковых агентов, а на более поздних итерациях значение r_1 уменьшается, диапазон перемещений поисковых агентов сокращается, обеспечивая сходимость к оптимуму. Таким образом, параметр r_1 в алгоритме *SCA* поддерживает определенный баланс между скоростью сходимости и диверсификацией пространства поиска решений.

Для повышения надежности алгоритма *SCA* используются два уравнения обновления положения. Чтобы определить, какое из уравнений (2.4) или (2.5) применять для обновления местоположения агентов поиска, используется вероятность переключения p ($p = 0{,}5$), которая зависит от случайного числа $r_4 \in [0, 1]$. Если $r_4 < p$, то для обновления положения агента используется уравнение (2.4). В противном случае используется уравнение (2.5):

$$X_{ij}^{t+1} = \begin{cases} X_{ij}^{t} + r_1 \times \sin(r_2) \times \left| r_3 \times P_{gj}^{t} - X_{ij}^{t} \right|, \text{если } r_4 < p \\ X_{ij}^{t} + r_1 \times cos(r_2) \times \left| r_3 \times P_{gj}^{t} - X_{ij}^{t} \right|, \text{если } r_4 \geq p \end{cases} \quad (2.9)$$

Это дает 50 %-ю вероятность выполнения каждого уравнения.

Значение константы b в алгоритме SCA принимается равным 2. Это означает, что две функции, зависящие от синуса и косинуса, будут изменяться в диапазоне [–2, 2]. Каждый агент поиска обновляет свое положение либо в направлении, противоположном точке назначения, либо в направлении к точке назначения. Если $r_1 < 1$, то X_i перемещается к точке назначения P_g (фаза сходимости к оптимуму), а когда $r_1 \geq 1$, то поисковый агент перемещается от точки назначения P_g (фаза расширения пространства поиска).

SCA останавливает процедуру оптимизации, когда количество итераций достигает заданного максимального числа. Существует вероятность найти с помощью *SCA* глобальный оптимум в задаче поскольку:

- алгоритм комбинирует стратегии направленного поиска и диверсификации пространства поиска, варьируя параметры в функциях синуса и косинуса;
- алгоритм в процессе поиска сохраняет наилучшее из полученных значений оптимума.

Причины эффективности алгоритма *SCA* заключаются в следующем: (он одновременно исследует несколько перспективных областей пространства решений; адаптивные значения параметров r_1, r_2, r_3, r_4 позволяют варьировать перспективные области между фазами направленного поиска и расширением поискового пространства; наилучшее решение никогда не теряется в процессе оптимизации. Доказательств сходимости алгоритма *SCA* нет, однако имеющиеся результаты говорят о том, что *SCA* эффективно конкурирует с другими метаэвристиками по точности и скорости сходимости. Одна из основных трудностей, связанных с *SCA*, заключается в управлении параметрами алгоритма. Четыре параметра *SCA*, которые необходимо настроить, являются важными целями для успешной работы этого алгоритма оптимизации. С учетом *NFL*-теоремы это означает, что *SCA* нуждается в гибридизации и настройке параметров при решении реальных проблем. Другим ограничением является целевая функция. Целевая функция требует модификации *SCA* для решения непрерывных, дискретных, бинарных, динамических или многокритериальных классов задач оптимизации.

В модификациях базового алгоритма *SCA* предлагается использовать полеты Леви для вычисления параметра r_1 и линейное уменьшении инерционного веса $w(t)$. Это способствует балансу между скоростью сходимости алгоритма и диверсификацией пространства поиска решений, однако временная сложность алгоритма при этом значительно увеличивается. Чтобы этого избежать, в уравнение для обновления положения поисковых агентов вводится инерционный вес $w(t)$:

$$X_{ij}^{t+1} = \begin{cases} w(t) \times X_{ij}^{t} + r_1 \times sin(r_2) \times \left| r_3 P_{ij}^{t} - X_{ij}^{t} \right|, r_4 < 0.5 \\ w(t) \times X_{ij}^{t} + r_1 \times co\,s(r_2) \times \left| r_3 P_{ij}^{t} - X_{ij}^{t} \right|, \ r_4 \geq 0.5 \end{cases}. \quad (2.10)$$

Согласно (2.10) большие значения w при небольших t активируют “глобальный поиск”, а меньшие значения w с возрастанием t облегчают “локальное уточнение”. Значение $w(t)$ уменьшается от начального значения w_{start} до конечного значения w_{end} в соответствии со следующим уравнением:

$$w(t) = (w_{start} - w_{end}) \times \frac{t_{max} - t}{t}. \quad (2.11)$$

Далее, в базовом алгоритме *SCA* используются параметры r_1, r_2, r_3, r_4. Параметр r_1 является наиболее важным для поддержания баланса между скоростью сходимости и диверсификацией пространства поиска решений. В базовом *SCA* значение r_1 линейно уменьшается от b до нуля с использованием (2.6). На начальных итерациях процесса оптимизации линейно убывающий параметр r_1 обеспечивает расширение пространства поиска при невысокой скорости сходимости; на более поздних итерациях скорость сходимости растет, однако увеличивается вероятность попадания в ловушку локальных оптимумов. Кроме того, поскольку процесс поиска *SCA* является нелинейным, то линейное уменьшение параметра r_1 неточно отражает фактический процесс поиска оптимума.

Поэтому предлагается модифицировать уравнение (2.6) для параметра преобразования r_1, следующим образом:

$$r_1(t) = (b_{start} - b_{end}) \times \exp\left(-\frac{t^2}{(k \times t_{max})^2}\right) + b_{end}. \quad (2.12)$$

Здесь используется стратегия нелинейного уменьшения параметра r_1, основанная на функции Гаусса, k – индекс нелинейной модуляции, b_{start} – начальное значение константы b, b_{end} – конечное значение константы b.

Приведем оценку вычислительной сложности модифицированного алгоритма *MSCA*.

Для инициализации популяции требуется $O(N_p)$ времени, где N_p обозначает размер популяции; для оценки пригодности каждого поискового агента требует $O(N_p)$ времени; вычисление управляющих параметров r_1, r_2, r_3, r_4 требует $O(N_p)$ времени; обновление местоположения поисковых агентов требует $O(N_p)$ времени. Таким образом, временная сложность *MSCA* $O(N_p)$ за одну итерацию $O(N_p)$. Следовательно, общая сложность *SCA* при максимальном числе итераций t_{max} равна $O(N_p \times t_{max})$.

Ниже приводятся примеры численных экспериментов и сравнительного анализа базового *SCA* и модифицированного *MSCA*.

Эксперименты проводились с использованием множества тестовых многомерных (с числом переменных *n* до 5000) функций и реального инженерного приложения из мирового набора бенчмарок для задач глобальной оптимизации (задача проектирования пружины растяжения/сжатия). В ходе экспериментов проводилось сравнение *MSCA* с базовым *SCA*, а также с наиболее известными конкурирующими метаэвристиками. Использовался язык программирования *C#*, тестирование проводилось на ЭВМ типа *IBM PC* с процессором *Core i7* с ОЗУ-8 Гбайт в среде *Windows 10*.

В качестве множества тестовых функций для оценки эффективности алгоритма *MSCA* были выбраны 7 известных функций с размерностями *n* = 100, 500, 1000 и 5000:

- функция Швефеля:
$$f_1(X) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|, x_i \in [-10; 10], f_{min} = 0;$$
- функция Розенброка:
$$f_2(X) = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2], x_i \in [-10; 10], f_{min} = 0;$$
- функция Растригина:
$$f_3(X) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10], x_i \in [-5{,}12; 5{,}12], f_{min} = 0;$$
- функция Гриванка:
$$f_4(X) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, x_i \in [-600; 600], f_{min} = 0;$$
- функция Захарова:
$$f_5(X) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0{,}5 i x_i\right)^2 + \left(\sum_{i=1}^{n} 0{,}5 i x_i\right)^4, x_i \in [-5; 10], f_{min} = 0;$$
- функция эллиптическая:
$$f_6(X) = \sum_{i=1}^{n} (10^6)^{(i-1)/(n-1)} x_i^2, x_i \in [-100; 100], f_{min} = 0;$$
- функция Саломона:
$$f_7(X) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0{,}1\sqrt{\sum_{i=1}^{n} x_i^2}, x_i \in [-100; 100], f_{min} = 0.$$

Здесь f_{min} является глобальным оптимальным значением.

Максимальное количество итераций t_{max} = 500 во всех запусках программ, в *MSCA* a_{start} = 0,1; a_{end} = 0; индекс нелинейной модуляции k = 15; и w_{start} = 2; w_{end} = 0.

В табл. 2.1 приведены средние результаты (*Mean*) и результаты стандартного отклонения (*St. dev*) для 30 независимых запусков 7 тестовых функций. Из табл. 2.1 видно, что за исключением функций $f_2(X)$, $f_5(X)$, $f_6(X)$ *MSCA* продемонстрировал отличную масштабируемость, поскольку его производительность почти не ухудшилась при увеличении размерности. При том, что оптимизация задач с размерностью n = 1000 и n = 5000 является весьма сложной задачей.

Таблица 2.1

Средние результаты и стандартное отклонение
для 30 независимых запусков семи тестовых функций
(n = 500, 1000 и 5000)

Функция	*MSCA*		
	Mean ± *St.dev* (n = 500)	*Mean* ± *St.dev* (n = 1000)	*Mean* ± *St.dev* (n = 5000)
$f_1(X)$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$
$f_2(X)$	$4{,}28\times10^{+02} \pm 2{,}09\times10^{-02}$	$9{,}74\times10^{+02} \pm 2{,}06\times10^{-02}$	$4{,}87\times10^{+03} \pm 6{,}22\times10^{-03}$
$f_3(X)$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$
$f_4(X)$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$
$f_5(X)$	$4{,}16\times10^{-93} \pm 7{,}18\times10^{-94}$	$8{,}48\times10^{-24} \pm 3{,}15\times10^{-25}$	$5{,}42\times10^{+02} \pm 2{,}68\times10^{+02}$
$f_6(X)$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$	$0{,}00\times10^{+00} \pm 0{,}00\times10^{+00}$
$f_7(X)$	$8{,}12\times10^{-77} \pm 2{,}54\times10^{-77}$	$5{,}11\times10^{-40} \pm 2{,}83\times10^{-40}$	$8{,}86\times10^{-03} \pm 3{,}50\times10^{-03}$

В табл. 2.2 приведены результаты среднего времени работы процессора (в сек.) для *MSCA* на 7 тестовых функциях (n = 100, 500, 1000 и 5000). Видно, что для 7 тестовых функций с n = 100, 500, 1000 и 5000 среднее процессорное время, затрачиваемое *MSCA*, растет незначительно.

Таблица 2.2

Среднее время работы процессора (в сек.) для *MSCA* на 7 тестовых функциях (n = 100, 500, 1000 и 5000)

Функция	*MSCA*			
	n = 100	n = 500	n = 1000	n = 5000
$f_1(X)$	0,9722	2,1567	3,9458	22,2772
$f_2(X)$	1,1287	3,0376	6,0482	24,0283
$f_3(X)$	0,8313	1,7633	4,2010	26,6843
$f_4(X)$	0,8678	2,0325	4,6021	26,4051
$f_5(X)$	0,8590	1,5285	4,3240	24,1148
$f_6(X)$	1,2658	3,9822	7,8033	33,6385
$f_7(X)$	1,0255	1,7931	5,6823	24,7228

Чтобы продемонстрировать превосходство алгоритма *MSCA* для решения многомерных задач глобальной оптимизации проведено его сравнение с четырьмя известными популяционными алгоритмами оптимизации: пчелиной колонии (*ABC*), роя частиц (*PSO*), дифференциальной эволюции (*DE*), оптимизатором китов (*WOA*). В качестве тестовых функций выбраны $f_2(X)$, $f_3(X)$, $f_4(X)$, $f_5(X)$, $f_6(X)$, $f_7(X)$. Размерность тестовых функций была установлена равной 1000. Каждый алгоритм независимо выполнялся 30 раз для каждой функции. Параметры алгоритмов *ABC, PSO, DE, WOA* и *MSCA* были установлены следующим образом: размер популяции был равен 100, 100, 100, 30 и 30, соответственно; максимальное число итераций равно 1000, 1000, 1000, 500 и 500 соответственно. В табл. 2.3 приведены средние значения (*Mean*) и стандартное отклонение (*St. dev*), полученные алгоритмами для указанных тестовых функций с n = 1000.

Таблица 2.3

Сравнение алгоритмов *ABC, PSO, DE, WOA* и *MSCA* для задачи глобальной оптимизации функций $f_2(X)$, $f_3(X)$, $f_4(X)$, $f_5(X)$, $f_6(X)$, $f_7(X)$ (размерность $n = 1000$) по показателям среднего значения (*Mean*) и стандартного отклонения (*St. dev*)

Функция	Индекс	*ABC*	*PSO*	*DE*	*WOA*	*MSCA*
$f_2(X)$	*Mean*	$1{,}46\times10^{+10}$	$7{,}00\times10^{+06}$	$3{,}18\times10^{+09}$	**$9{,}93\times10^{+02}$**	$9{,}99\times10^{+02}$
	St. dev	$1{,}64\times10^{+08}$	$9{,}97\times10^{+05}$	$1{,}78\times10^{+08}$	**$7{,}93\times10^{-01}$**	$2{,}03\times10^{-02}$
$f_3(X)$	*Mean*	$1{,}77\times10^{+04}$	$3{,}57\times10^{+03}$	$1{,}35\times10^{+04}$	$3{,}64\times10^{-13}$	**$0{,}00\times10^{+00}$**
	St. dev	$1{,}13\times10^{+02}$	$1{,}25\times10^{+02}$	$8{,}59\times10^{+01}$	$8{,}14\times10^{-14}$	**$0{,}00\times10^{+00}$**
$f_4(X)$	*Mean*	$2{,}77\times10^{+04}$	$2{,}89\times10^{+02}$	$6{,}44\times10^{+03}$	**$0{,}00\times10^{+00}$**	**$0{,}00\times10^{+00}$**
	St. dev	$6{,}32\times10^{+02}$	$2{,}95\times10^{+01}$	$1{,}33\times10^{+02}$	**$0{,}00\times10^{+00}$**	**$0{,}00\times10^{+00}$**
$f_5(X)$	*Mean*	$9{,}69\times10^{+06}$	$9{,}67\times10^{+03}$	$2{,}05\times10^{+04}$	$1{,}56\times10^{+04}$	**$9{,}08\times10^{-27}$**
	St. dev	$4{,}76\times10^{+06}$	$5{,}22\times10^{+03}$	$5{,}51\times10^{+02}$	$4{,}39\times10^{+02}$	**$2{,}03\times10^{-26}$**
$f_6(X)$	*Mean*	$2{,}05\times10^{+11}$	$2{,}84\times10^{+08}$	$1{,}12\times10^{+09}$	$4{,}17\times10^{-65}$	**$0{,}00\times10^{+00}$**
	St. dev	$5{,}94\times10^{+09}$	$2{,}83\times10^{+07}$	$4{,}76\times10^{+07}$	$8{,}23\times10^{-65}$	**$0{,}00\times10^{+00}$**
$f_7(X)$	*Mean*	$1{,}77\times10^{+02}$	$2{,}64\times10^{+01}$	$9{,}91\times10^{+01}$	$1{,}40\times10^{-01}$	**$1{,}37\times10^{-45}$**
	St. dev	$1{,}31\times10^{+00}$	$5{,}81\times10^{-01}$	$9{,}80\times10^{-01}$	$8{,}94\times10^{-02}$	**$3{,}06\times10^{-45}$**

В табл. 2.3 для наглядности наилучшие результаты выделены шрифтом. *MSCA* получил наилучшие результаты по сравнению с *ABC, PSO* и *DE* на тестовых функциях $f_2(X)$, $f_3(X)$, $f_4(X)$, $f_5(X)$, $f_6(X)$, $f_7(X)$ размерности $n = 1000$. Однако алгоритм *WOA* получил лучший результат для функции $f_2(X)$ и такой же результат, как *MSCA*, для функции $f_4(X)$. Была проведена проверка статистической значимости полученных результатов по тесту Фридмана и *T*-критерию Уилкоксона. *MSCA* получил первое место согласно теста Фридмана. Значения $p < 0{,}1$ и $p < 0{,}05$ с использованием p-критерия суммы

рангов Уилкоксона рассматривалось как адекватное доказательство против нулевой гипотезы, которая отвергается, поэтому *MSCA* превосходит конкурирующие алгоритмы.

Что касается среднего времени работы процессора, то оказалось, что *MSCA* по сравнению с алгоритмами *ABC, PSO, DE* и *WOA* требует меньшего процессорного времени выполнения для всех тестовых функций $f_1(X)$, $f_2(X)$, $f_3(X)$, $f_4(X)$, $f_5(X)$, $f_6(X)$, $f_7(X)$ размерности $n = 1000$.

Экспериментально была проведена оценка эффективности *MSCA* на тестовой инженерной задаче проектирования пружины растяжения/сжатия. Цель состоит в том, чтобы минимизировать вес пружины. Целевая функция задачи зависит от трех переменных: диаметра намотки (D), диаметр проволоки (d) и число витков (P) (рис. 2.1).

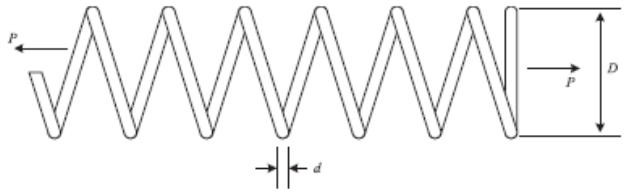

Рис. 2.1. Пружина растяжения/сжатия

Математическая формулировка задачи имеет следующий вид.

Заданы переменные $X = [x_1, x_2, x_3] = [d, D, P]$.

Необходимо минимизировать целевую функцию

$$f(X) = (x_3 + 2)x_2x_1^2$$

при следующих ограничениях:

$$g_1(X) = 1 - \frac{x_2^3 x_3}{71785x_1^4} \le 0,$$

$$g_2(X) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} \le 0,$$

$$g_3(X) = 1 - \frac{140{,}45x_1}{x_2^2x_3} \le 0,$$

$$g_4(X) = \frac{x_1 + x_2}{1{,}5} - 1 \le 0,$$

где $0{,}05 \le x_1 \le 2$; $0{,}25 \le x_2 \le 1{,}30$; $2{,}00 \le x_3 \le 15$.

Результаты работы *MSCA* для задачи проектирования пружины растяжения/сжатия сравнивались с известными результатами решения этой задачи конкурирующими алгоритмами *ACO, BA, CSA, ES, GA, MFO, SCA* и *WOA*. Результаты сравнения представлены в табл. 2.4.

Таблица 2.4

Результаты сравнения *MSCA* с алгоритмами *ACO, BA, CSA, ES, GA, MFO, SCA* и *WOA* для задачи проектирования пружины растяжения/сжатия

Алгоритм	Оптимальное значение переменных в задаче			Функция *f(X)*
	d	*D*	*P*	
ACO	0,051609	0,354714	11,41083	0,0126702
BA	0,051690	0,356730	11,28850	0,0126657
CSA	0,051689	0,356717	11,28901	0,0126652
ES	0,051643	0,355360	11,39793	0,0126978
GA	0,051989	0,363965	10,89052	0,0126810
MFO	0,051994	0,364109	10,86842	0,0126667
SCA	0,050780	0,334779	12,72269	0,0127096
WOA	0,051207	0,345215	12,54854	0,0131695
MSCA	0,052021	0,364768	10,83230	0,0126671

Из табл. 2.4 видно, что в сравнении с *ACO, ES, GA, SCA* и *WOA* алгоритм *MSCA* обеспечил лучший результат для целевой функции в задаче проектирования пружины растяжения/сжатия.

Рост интереса к *SCA* обусловлен его мощными возможностями оптимизации и простотой применения для решения сложных задач оптимизации в таких научных дисциплинах как электротехника, системы управления, теория графов, машинное обучение, робототехника, экономика, планирование поставок и др.

2.3. Алгоритм эгоистичного стада

Многие животные, живущие в коллективе, проявляют черты кооперативного поведения. Однако это не обязательно верно для каждой отдельной особи, живущей в социуме. В отличие от гипотезы о том, что социальное поведение основано на взаимной выгоде для всего коллектива, альтернативной представляется гипотеза из теории эгоистичного стада, предложенная В. Гамильтоном. Согласно этой теории, действия особи внутри коллектива (стада) отличаются определенной степенью эгоизма, особенно в ситуации, когда стадо подвергается опасности нападения хищников. Теория эгоистичного стада утверждает, что от решений, принимаемых особью стада, может выиграть как сама особь, так и стадо в целом. При

том, что эгоистичное поведение отдельных особей может приводить к негативным последствиям для другой особи стада.

Гамильтон предположил, что эгоистичное поведение заключается в том, что каждая особь стада стремится уменьшить свои шансы быть пойманной хищником. Поэтому доминантные и более сильные животные стремятся получить центральные позиции в агрегации, с низкой вероятностью стать жертвой нападения хищников. Во время нападения хищников риск возрастает среди особей, находящихся на периферии стада, и уменьшается ближе к центру стада. Подчиненных и более слабых животных принуждают занимать более рискованные позиции.

Хищники нападают на ближайшую добычу, которая, как правило, находится на периферии агрегации. Существуют несколько факторов, которые могут повлиять на выбранные правила движения: начальное пространственное положение особей стада, а также плотность популяции в стаде, стратегия нападения хищника и бдительность. В частности, в ситуации группового бегства, самой безопасной является позиция впереди стада. В этом случае наиболее вероятными целями для хищника являются более медленные члены стада.

В природе много примеров эгоистичного стадного поведения. Одним из наиболее широко изученных примеров являются стада антилоп гну и зебр, подвергающихся опасности нападения хищников.

Модель предполагает, что пространством поиска решений является открытая равнина с обитающим там стадом животных. Коэволюционный самонастраивающийся алгоритм (КОСА) моделирует два различных типа поисковых агентов: стадо травоядных особей и стая хищников, которые охотятся за добычей. Оба типа поисковых агентов индивидуально управляются набором различных эвристических операторов на основе поведенческих аспектов, наблюдаемых в отношениях «жертва-хищник». Рассмотрим основные шаги алгоритма [13].

Инициализация популяции. Первым шагом итерационного алгоритма является случайная инициализация популяции H из N травоядных и хищников:

$$H = \{h_1, h_2, \dots\ h_n\}$$

Каждая i-я особь представляется в виде n-мерного вектора $h_i = [h_{i,1}, h_{i,2}, \dots\ h_{i,n}]$, который является возможным решением оптими-

зационной задачи. Особи популяции инициализируются с помощью случайного равномерного распределение внутри пары заранее заданных границ пространства решений:

$$h_{i,j}^{0} = x_j^{low} - rand(0,1) * (x_j^{high} - x_j^{low}) \quad (2.13)$$

где $i = 1..N, j = 1..x_j^{low}$ и x_j^{high} – нижняя и верхняя границы пространства решений, $rand(0,\ 1)$ – случайное число из интервала [0, 1].

Вся популяция животных H делится на две группы: $P = \{p_1, p_2, \dots, p_{Np}\}$ – травоядные стада и $R = \{r_1, r_2, \dots, r_{Nr}\}$ – хищники стаи (N_p – число травоядных, N_r – число хищников, $N_p + N_r = N$). В природе число животных в стаде обычно превышает число животных в стае хищников. В алгоритме N_p выбирается случайным образом в диапазоне 70–90% от N:

$$Np = floor(N \cdot rand(0.7, 0.9)).$$

Оценка живучести. В биологической метафоре взаимодействия жертвы и хищника каждая особь в стаде травоядных или стае хищников, в зависимости от своих способностей к выживанию, имеет шанс пережить нападение или преуспеть в добыче, соответственно. В алгоритме каждой особи hi присваивается значение функции живучести SF_{hi}, которое представляет собой оценку его шансов к выживанию относительно его текущего положения в пространстве решений. Эта оценка зависит от расстояния относительно самой безопасной и самой рискованной позицией, которые известны в данный момент всем членам популяции и которые в контексте глобальной задачи оптимизации представлены текущими лучшим и худшим решениями, найденными к данной итерации алгоритма. Оценка живучести для каждой особи рассчитывается следующим образом:

$$SF_{hi} = \frac{f(h_i) - f_{best}}{f_{best} - f_{worst}} \quad (2.14)$$

где $f(h_i)$ – значение фитнесс-функции относительно позиции особи h_i, f_{best} и f_{worst} – лучшие и худшие значения фитнесс-функции, найденные алгоритмом, которые определяются следующим образом:

$$f_{best} = \max_{j \in \{0..k\}} \left(\left(\max_{i \in \{1..N\}} \left(f(h_i) \right) \right)_j \right), \quad (2.15)$$

$$f_{worst} = \min_{j \in \{0..k\}} \left(\left(\min_{i \in \{1..N\}} \left(f(h_i) \right) \right)_j \right), \quad (2.16)$$

где k – текущая итерация алгоритма.

Модель стада травоядных. Популяция травоядных особей включает вожака, группу особей, стремящихся занять позицию ближе к

центру стада, и особей, которые движутся относительно независимо от стада, что связано с текущей внутренней структурой и паттернами движения стада. Руководит стадом вожак – особь с наибольшими способностями к выживанию. Во время нападения хищника вожак стада выполняет важную задачу по выбору пути отступления или стратегии, которую должны использовать все остальные члены стада. С другой стороны, особи, двигающиеся вместе со стадом, пытаясь снизить риск хищничества, стремятся поместить других особей между собой и нападающими хищниками.

На каждой итерации k алгоритм определяет единственную особь p_L среди популяции стада как вожака эгоистичного стада (P) с учетом значений живучести:

$$p_L^k = \left(p_i^k \in P^k \middle| SF_{p_i^k} = \max_{j \in \{1..N_p\}} \left(SF_{p_j^k} \right) \right) \quad (2.17)$$

Вожак обладает наибольшим значением живучести.

Эгоистичные особи внутри стада стремятся увеличить свои шансы выжить в случае нападения хищника, стремясь занять более безопасное положение внутри стада по отношению к ближайшей соседней особи p^k_{ci}, которая определяется следующим образом:

$$p_{c_i}^k = (p_j^k \in P^k, p_j^k \neq [p_i^k, p_L^k] | SF_{p_j^k} > SF_{p_i^k}, r_{i,j} = \min_{j \in \{1..N_p\}} (\| p_i^k - p_j^k \|)) \quad (2.18)$$

где $r_{i,j}$ – евклидово расстояние между соседними особями стада на k-й итерации.

Наиболее интересным поведением, наблюдаемым в популяции, является решение отдельных особей либо следовать за движением стада, либо покинуть его и двигаться независимо от него. Критерии принятия решений, лежащие в основе такого поведения, тесно связаны со степенью безопасности каждой особи во время нападения хищников, а также с положением особи в стаде. В алгоритме применяются операторы принятия решений, которые учитывают индивидуальные возможности выживания каждой особи стада. В результате популяция $\boldsymbol{P}$ разделяется две подгруппы: подгруппа особей, остающихся в стаде $\boldsymbol{P_F}$ и подгруппа особей-дезертиров $\boldsymbol{P_D}$. В алгоритме эти подгруппы определяются на каждой итерации k в зависимости от текущего значения выживаемости следующим образом:

$$\boldsymbol{P}_F^k = \left\{ \boldsymbol{p}_i^k \neq \boldsymbol{p}_L^k \middle| SF_{\boldsymbol{p}_i^k} \geq rand(0,1) \right\}, \quad (2.19)$$

$$\boldsymbol{P}_D^k = \left\{ \boldsymbol{p}_i^k \neq \boldsymbol{p}_L^k \middle| SF_{\boldsymbol{p}_i^k} < rand(0,1) \right\}. \quad (2.20)$$

Особи с более высоким значением живучести имеет более высокие шансы следовать за стадом, в отличие от особей с низкой живучестью, вероятность дезертирства которых из стада увеличивается.

В большинстве случаев риск стать добычей увеличивается среди особей на периферии стада и уменьшается к центру такой агрегации. В алгоритме центр популяции стада определяется следующим образом:

$$\boldsymbol{p}_M^k = \frac{=\sum_{i=1}^{Np} SF_{\boldsymbol{p}_i^k} \cdot \boldsymbol{p}_i^k}{\sum_{j=1}^{Np} SF_{\boldsymbol{p}_j^k}} \tag{2.21}$$

Аналогично (2.1) определяется наилучшее положение атакующих хищников:

$$\boldsymbol{r}_M^k = \frac{\sum_{i=1}^{Nr} SF_{\boldsymbol{r}_i^k} \cdot \boldsymbol{r}_i^k}{\sum_{j=1}^{Nr} SF_{\boldsymbol{r}_j^k}} \tag{2.22}$$

Операторы движения стада травоядных. Для моделирования движения стада травоядных в алгоритме используются различные эвристические операторы: движение вожака и операторы движения остальных особей, включая особей-дезертиров. Особи в стаде стремятся повысить свои шансы выжить после нападения хищника, двигаясь таким образом, чтобы другая особь оказалась между ними и нападающими хищниками. Такое поведение моделируется следующим образом:

$$\xi_{\boldsymbol{p}_i,\boldsymbol{p}_j} = SF_{p_j} \cdot e^{-\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2}, \tag{2.23}$$

где $||pi - pj||$ – евклидово расстояние между особями $\boldsymbol{p}_i$ и $\boldsymbol{p}_j$. Величину (2.23) в общем виде можно вычислить для любой пары особей стада, однако в алгоритме используются несколько конкретных направлений движения:

- движение особи $\boldsymbol{p}_i$ к вожаку стада:

$$\xi_{\boldsymbol{p}_i,\boldsymbol{p}_L} = SF_{p_L} \cdot e^{-\|\boldsymbol{p}_i - \boldsymbol{p}_L\|^2}; \tag{2.24}$$

- движение особи $\boldsymbol{p}_i$ к ближайшему лучшему соседу:

$$\xi_{\boldsymbol{p}_i,\boldsymbol{p}_{c_i}} = SF_{p_{c_i}} \cdot e^{-\left\|\boldsymbol{p}_i - \boldsymbol{p}_{c_i}\right\|^2}; \tag{2.25}$$

- движение особи $\boldsymbol{p}_i$ к центру популяции стада $\boldsymbol{p}_M$:

$$\xi_{\boldsymbol{p}_i,\boldsymbol{p}_M} = SF_{p_M} \cdot e^{-\|\boldsymbol{p}_i - \boldsymbol{p}_M\|^2}; \tag{2.26}$$

- движение особи $\boldsymbol{p}_i$ к самой безопасной позиции, известной всей агрегацией:

$$\xi_{\boldsymbol{p}_i,\boldsymbol{p}_{best}} = SF_{p_{best}} \cdot e^{-\|\boldsymbol{p}_i - \boldsymbol{p}_{best}\|^2}, \tag{2.27}$$

и которая определяется как:

$$f(\boldsymbol{x}_{best}) = f_{best}, \tag{2.28}$$

где f_{best} вычисляется в соответствии с (2.15).

Кроме того, следует иметь в виду, что хищники представляют собой основной источник опасности для травоядных. Поэтому их присутствие влияет на движение особей стада. Этот фактор учитывается в алгоритме следующим образом:

$$\eta_{\boldsymbol{p}_i,\boldsymbol{r}_M} = -SF_{r_M} \cdot e^{-\|\boldsymbol{p}_i - \boldsymbol{r}_M\|^2}, \tag{2.29}$$

где SF_{r_M} – значение живучести согласно (2.22).

Величина в (2.29) определяется между особью $\boldsymbol{p}_i$ и центром стаи хищников $\boldsymbol{r}_M$, исходя из предположения, что любая особь стада всегда будет пытаться уйти как можно дальше от всех нападающих хищников.

Вожак стада занимает самую безопасную позицию в стаде, однако это не обязательно означает, что такая особь полностью защищена от хищников. При этом вожак способен проявить несколько различных типов лидерского поведения. Например, в алгоритме позиция вожака на следующей итерации обновляется следующим образом:

$$\boldsymbol{p}_L^{k+1} = \begin{cases} \boldsymbol{p}_K^k + \boldsymbol{c}^k, \text{если } SF_{\boldsymbol{p}_L^k} = 1 \\ \boldsymbol{p}_K^k + \boldsymbol{s}^k, \text{если } SF_{\boldsymbol{p}_L^k} < 1, \end{cases} \tag{2.30}$$

где $\boldsymbol{c}^k$ и $\boldsymbol{s}^k$ – вектора движения вожака.

Верхнее правило движения, выбранное вожаком стада, предполагает кооперативное лидерство, когда вожак стада находится в текущей лучшей позиции и направляет движение стада, чтобы оно оказалось потенциально полезным для стада. Из уравнения (2.14) следует, что если $f(p_L^k) = f_{best}$, то $SF_{p_L^k} = 1$.

Этим достигается удаление стаи от атакующих хищников. Вектор движения c^k определяется следующим образом:

$$\boldsymbol{c}^k = 2 \cdot \alpha \cdot \eta_{\boldsymbol{p}_L,\boldsymbol{r}_M}^k \cdot (\boldsymbol{r}_M^k - p_L^k), \tag{2.31}$$

где $\eta_{\boldsymbol{p}_i,\boldsymbol{r}_M}$ определяется согласно (2.29), α – случайное число в интервале [0, 1].

С другой стороны, если $SF_{\boldsymbol{p}_L^k} < 1$, то вожак стада выбирает эгоистичное направление движения, стремясь уменьшить риск и пытаясь двигаться к наиболее безопасному положению. Его вектор движения s^k определяется как

$$s^k = 2 \cdot \alpha \cdot \xi_{p_L,x_{best}}^k \cdot \left(x_{best}^k - p_L^k\right), \tag{2.32},$$

где $\xi_{p_L,x_{best}}^k$ определяется согласно (27).

Согласно (2.19) и (2.20) особи стада P разделяется на подгруппу остающихся в стаде P_F и подгруппу особей-дезертиров P_D в зависимости от текущего значения выживаемости. Учитывая это, на

каждой итерации k алгоритма обновляется позиция каждой особи следующим образом:

$$p_i^{k+1} = \begin{cases} p_i^k + f_i^k, \text{если } p_i^k \in P_F^k \\ p_i^k + d_i^k, \text{если } p_i^k \in P_D^k, \end{cases} \quad (2.33)$$

где f^k_i – вектор движения особей внутри стада; d^k_i – вектор движения особей-дезертиров.

$$f_i^k = \begin{cases} 2 \cdot (\beta \cdot \xi^k_{p_i,p_L} \cdot (p_L^k - p_i^k) + \gamma \cdot \xi^k_{p_i,p_{ci}} \cdot (p_{ci}^k - p_i^k), \text{если } p_i^k \in P_d^k \\ 2 \cdot \delta \cdot \xi^k_{p_i,p_M} \cdot (p_M^k - p_i^k), \text{если } p_i^k \in P_s^k \end{cases}, \quad (2.34)$$

где β, γ, δ – случайные числа из интервала [0, 1].

$$d_i^k = 2 \cdot (\beta \cdot \xi^k_{p_i,x_{best}} \cdot \left(x_{best}^k - p_i^k\right) + \gamma \cdot \left(1 - SF_{p_i^k}\right) \cdot \hat{r}, \quad (2.35)$$

где x_{best} определяется согласно (2.27), $\hat{r}$ – единичный вектор, указывающий случайное направление движения.

Операторы движения стаи хищников. Особи внутри стада находятся в большей безопасности, нежели одиночные особи. Это происходит потому, что движение стада затрудняет решение хищника атаковать конкретную особь. Однако этот уровень безопасности особей внутри стада зависит от их текущего положения. Положение хищника относительно стада также является важным фактором при принятии решения об атаке той или иной особи стада. Алгоритм моделирует движение стаи хищников R как с учетом живучести особи в зависимости от ее положения в стаде, так и от расстояния особи стада до хищников.

Сначала предполагается, что каждая особь p_j стада P может преследоваться хищником r_i с некоторой вероятностью:

$$\wp_{r_i,p_j} = \omega_{r_i,p_j} / \sum_{m=1}^{N_P} \omega_{r_i,p_m}. \quad (2.36)$$

В (2.36) величина ω_{r_i,p_j} обозначает привлекательность добычи p_j для хищника r_i. Эта величина учитывает положение особи в стаде, расстояние от особи стада до хищников и вычисляется по следующей формуле:

$$\omega_{r_i,p_j} = \left(1 - SF_{p_j}\right) \cdot e^{-\|r_i - p_j\|^2}, \quad (2.37)$$

где SF_{p_j} – значение живучести особи p_j, $\|r_i - p_j\|$ – евклидово расстояние между добычей и хищником. Это означает, что хищники предпочитают нападать на более слабые и близкие жертвы. С учетом этого на каждой итерации k алгоритма моделируется преследование хищником более слабой и близкой жертвы:

$$r_i^{k+1} = r_i^k + 2 \cdot \rho \cdot \left(p_u^k - r_i^k\right), \quad (2.38)$$

где ρ – случайное число из интервала [0, 1].

Оператор нападения хищника на жертву. Определим область опасности для особи стада в виде круга области конечного радиуса *RAD*:

$$RAD = \sum_{j=1}^{n} \left| x_j^{low} - x_j^{high} \right| / (2 \cdot n), \quad (2.39)$$

где n – число особей (в алгоритме для простоты предполагается, что радиус области опасности одинаков для всех особей).

До начала нападения множество жертв $Vic = \varnothing$.

Во время нападения предполагается, что расстояние между $\boldsymbol{r}_i$ и $\boldsymbol{p}_j$ меньше или равно радиусу *RAD*, а под угрозой может находиться более одной особи стада. Для каждого хищника r_i множество потенциальных жертв определяется как

$$Q_{r_i} = \left\{ p_j \in P \middle| SF_{p_j} < SF_{r_i} \right\} \& \left\| r_i - p_j \right\| \le RAD \;\&\; p_j \notin Vic, \quad (2.40)$$

где SF_{p_j} и SF_{r_i} – значения выживаемости жертвы и хищника соответственно, $\left\| r_i - p_j \right\|$ – евклидово расстояние между жертвой и хищником.

Как только для конкретного хищника r_i согласно (2.40) идентифицировано множество потенциальных жертв Q_{r_i}, то выбирается одна из них. В алгоритме такое решение принимается на основе следующей вероятности:

$$\mathcal{P}_{r_i,p_j} = \omega_{r_i,p_j} / \sum_{\left(p_m \in Q_{r_i}\right)} \omega_{r_i,p_m}, \quad (2.41)$$

где $p_j \in Q_{r_i}$, ω_{r_i,p_j} – привлекательность добычи согласно (2.37). Применяя метод выбора рулетки, согласно (2.41) выбирается одна из таких особей p_j, которая затем считается убитой атакующим хищником r_i. и помещается во множество *Vic*. Отметим, что если $Q_{r_i} = \emptyset$, то хищники не охотятся.

Решения, соответствующие убитой во время нападения особи стада, исключаются из данного пространства решений и замещаются новыми с помощью специального оператора кроссинговера.

Оператор кроссинговера. В природе размер популяций хищников и травоядных динамично изменяется во времени. В сбалансированных биосистемах такие изменения являются периодическими, и популяция восстанавливается. Поэтому алгоритм включает вычислительную процедуру, которая позволяет восстановить популяцию, используя оператор кроссинговера особей стада. Вначале формируется подмножество особей для кроссинговера:

$$C = \left\{ p_j \notin Vic \right\} \quad (2.42)$$

Вероятность участия в скрещивании зависит от живучести особи и определяется как:

$$\mathcal{C}_{p_j} = \frac{SF_{p_j}}{\sum_{p_m \in C_j} SF_{p_m}} \quad (2.43)$$

Особи с более высокими значениями живучести будут иметь более высокие шансы на генерацию новых решений-потомков. Чтобы сгенерировать новое решение-потомок, случайно, используя рулетку, выбираются *n* особей из множества *C*, с учетом вероятностей $\mathcal{C}_{p_j}$ согласно (2.43). Применяя эту процедуру, восстанавливаем популяцию стада после нападения хищников.

В отличие от конкурирующих биоэвристик, алгоритм КОСА моделирует различных поисковых агентов (хищники и жертвы) с различным индивидуальным поведением. Иначе говоря, КОСА представляет мультиалгоритмический подход и является коэволюционным алгоритмом. Кроме того, он включает самонастраивающиеся вычислительные механизмы, которые позволяют поддерживать баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска решений: эгоистичное стадное поведение, уникальную внутреннюю социальную структуру агентов, хищничество и механизмы восстановления популяции.

Алгоритм КОСА сравнивался с алгоритмами *PSO, ABC, FA, DE, GA, CSA, MFO* в задаче поиска глобального оптимума на тестовых многомерных функциях Розенброка, Швефеля, Захарова, Саломона и Цина (табл. 2.5).

Здесь *n* – размерность функции, B^n – интервал варьирования переменных x_i, $\boldsymbol{X}^*$ – оптимальное решение, $f_i(X^*)$ – минимальное значение функции. Для поиска решений алгоритмами КОСА, *PSO, ABC, FA, DE, GA, CSA, MFO* было выделено по $N = 50$ особей, а максимальное число итераций $T = 1000$.

Таблица 2.5

Тестовые функции для экспериментов

Имя функции	Тестовая функция $f_i(\boldsymbol{X})$	$x_i \in B^n$	n	Минимум
Розенброк	$f_1(\mathbf{X}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-5,10]^n$	30	$f_1(\mathbf{X}^*) = 0$, $\mathbf{X}^* = (1..1)$
Швефель	$f_2(\mathbf{X}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	$[-100,100]^n$	30	$f_2(\mathbf{X}^*) = 0$, $\mathbf{X}^* = (0..0)$
Захаров	$f_3(\mathbf{X}) = \sum_{i=1}^{n} (x_i)^2 + \left(\sum_{i=1}^{n} 0{,}5 i x_i\right)^2 + (\sum_{i=1}^{n} 0{,}5 i x_i)^4$	$[-5,10]^n$	30	$f_3(\mathbf{X}^*) = 0$, $\mathbf{X}^* = (0..0)$
Саломон	$f_4(\mathbf{X}) = -\cos\left(2\pi \sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0{,}1 \sqrt{\sum_{i=1}^{n} x_i^2 + 1}$	$[-100,100]^n$	30	$f_4(\mathbf{X}^*) = 0$, $\mathbf{X}^* = (0..0)$
Цин	$f_5(\mathbf{X}) = \sum_{i=1}^{n} (x_i^2 - i)^2$	$[-500,500]^n$	30	$f_5(\mathbf{X}^*) = 0$, $\mathbf{X}^* = (\mp\sqrt{1}) .. (\mp\sqrt{n})$

Алгоритм КОСА сравнивался с 7 конкурирующими алгоритмами по следующим показателям: среднее лучшее на данный момент решение, медианное лучшее на данный момент решение и стандартное отклонение от лучшего на данный момент решения. Результаты усреднялись по 30 отдельным запускам алгоритмов.

В большинстве случаев результаты, полученные КОСА на рассматриваемых тестовых функциях, превосходят результаты конкурирующих алгоритмов. Это связано с достигаемым балансом между скоростью сходимости алгоритма и диверсификацией пространства поиска решений.

Было проведено непараметрическое доказательство статистической значимости полученных результатов с использованием критерия суммы рангов Уилкоксона для независимых выборок, найденных каждым из сравниваемых алгоритмов на 30 тестовых запусках. Уро-

вень значимости 5 %. Значение $T < 0{,}05$ рассматривалось как адекватное доказательство против нулевой гипотезы, которая отвергается, а предложенный алгоритм превосходит конкурирующий.

Экспериментальные результаты позволяют утверждать, что результаты по алгоритму КОСА являются статистически значимыми, они не произошли случайно.

2.4. Алгоритм летучих мышей

Алгоритм летучих мышей (*BA*) был предложен Янгом на основе эхолокационного поведения летучих мышей. Эхолокация состоит из двух этапов: испускание громких частотно-модулированных звуковых импульсов и прием (прослушивание) эхо-звуков, которые отражаются от окружающих объектов. Летучие мыши используют свою гидролокационную систему для обнаружения добычи, уклонения от препятствий, определения местоположения ночлега.

В *BA* используются следующие параметры, значения которых корректируются в процессе поиска: частота, громкость и скорость импульсного излучения. Частота моделируется набором *d*-мерных векторов, каждый из которых связан с *i*-й мышью и которые случайным образом корректируются на *k*-й итерации по формуле:

$$\mathcal{F}_i^k = \mathcal{F}_{min} + \beta(\mathcal{F}_{max} - \mathcal{F}_{min}), \tag{2.44}$$

где параметр $\mathcal{F}_{min}$ и $\mathcal{F}_{max}$ обозначают минимальную и максимальную частоты соответственно; β – это вектор случайных чисел, каждое из которых находится в интервале [0, 1]. Параметры громкости A_i и скорости импульсного излучения r_i, начальные значения которых A_i^0 и r_i^0 определяются при инициализации алгоритма. По мере развития процесса поиска оптимального решения значения этих параметров изменяются в соответствии со следующим выражением:

$$A_i^{k+1} = \alpha A_i^k,\ r_i^{k+1} = r_i^0(1 - \exp(-\gamma k)), \tag{2.45}$$

где $\alpha < 1$ и $\gamma < 1$ – некоторые константы. При обновлении местоположения *i*-й летучей мыши на *k*-й итерации применяется следующий оператор:

$$\boldsymbol{x}_i^k = \boldsymbol{x}_i^{k-1} + \boldsymbol{v}_i^k, \tag{2.46}$$

где $\boldsymbol{x}_i^{k-1}$ представляет положение *i*-й летучей мыши на предыдущей итерации (k−1), в то время как $\boldsymbol{v}_i^k$ обозначает скорость *i*-й летучей мыши, которая, в свою очередь, вычисляется следующим образом:

$$\boldsymbol{v}_i^k = \boldsymbol{v}_i^{k-1} + \mathcal{F}_i^k \cdot \left(x_i^{k-1} - \boldsymbol{x}_{best}\right), \tag{2.47}$$

где $\boldsymbol{x}_{best}$ обозначает текущее глобально лучшее решение, найденное в процессе поиска, а $\mathcal{F}_i^k$ представляет частотный вектор, определяемый согласно (2.44).

Модификация процедуры *BA* включает локальную схему поиска, согласно которой на каждой итерации случайно выбранный индивид среди текущих лучших решений дополнительно уточняется путем выполнения случайного блуждания следующим образом:

$$\boldsymbol{x}_*^k = \begin{cases} \boldsymbol{x}_i^k + \varepsilon A_i^k, & \text{если } \left(rand > r_i^k\right), \\ \boldsymbol{x}_i^k, & \text{в противном случае,} \end{cases} \tag{2.48}$$

где *rand* – случайное число, полученное из равномерно распределенного интервала [0, 1]. Причем новое решение $\boldsymbol{x}_*{}^k$ принимается в качестве местоположения *i*-й летучей мыши только при соблюдении определенных условий, в частности:

$$\boldsymbol{x}_i^k = \begin{cases} \boldsymbol{x}_*^k, & \text{если } \left(rand < A_i^k \;\&\boldsymbol{x}_i^k < \boldsymbol{x}_*^k\right), \\ \boldsymbol{x}_i^k, & \text{в противном случае.} \end{cases} \tag{2.49}$$

В каждом поколении для модифицированной процедуры *BA* основным действием является обновление местоположения *i*-й летучей мыши на *k*-й итерации согласно формуле (2.46). В модифицированной процедуре *BA* применяется оператор скорости импульсного излучения r_i на основе *V*-образной передаточной функции согласно (2.45). *V*-образная передаточная функция в *BA* используется для установления положения летучей мыши согласно (2.46) следующим образом:

$$f = \arctan\left(\boldsymbol{x}_i^k\right). \tag{2.50}$$

В [14] был предложен гиперэвристический алгоритм *GEBA*, который является комбинацией метаэвристик летучих мышей и дифференциальной эволюции. Он включает вероятностный механизм их выбора в зависимости от текущего состояния решения и был разработан для решения задачи выбора значимых признаков классификации и оценки точности классификации. Выбор параметров настройки эвристик основан на экспериментальных данных. Если *n* – число признаков, характеризующих данные, то наилучшим будем считать решение, имеющее минимальное число признаков и высокую точность классификации.

Алгоритм начинается с генерации популяции случайных решений (подмножества классификационных признаков). Для оценки решений используется фитнесс-функция. Через $\boldsymbol{x}_{best}$ обозначается

лучшее решение в популяции. Основной цикл алгоритма повторяется многократно. В алгоритме *GEBA* эмпирически установлена вероятность выбора решения по метаэвристике летучих мышей, равной 0,6, по метаэвристике дифференциальной эволюции – 0,4.

В качестве данных использовался мировой репозиторий данных и модельных задач машинного обучения *UCI*. Репозиторий содержит реальные данные по прикладным задачам в области биологии, медицины, физики, техники, социологии и др. Наборы данных (*data set*) именно этого репозитория чаще всего используются исследовательским сообществом для эмпирического анализа алгоритмов машинного обучения. Для проверки производительности предложенного алгоритма использовались 10 эталонных наборов данных *UCI*. При выборе наборов данных из репозитория учитывалось разнообразие областей, а также различие в числе признаков и экземпляров. В табл. 2.6 приводится краткое описание наборов данных.

Таблица 2.6

Описание используемых наборов данных *UCI*

№ набора	Имя набора данных	Число признаков	Число экземпляров	Число классов	Область данных
S1	Банковский кредит	24	1000	2	Бизнес
S2	Гидролокатор	60	20	2	Физика
S3	Диабет	6	144	3	Медицина
S4	Диагностика онкологии	32	596	2	Медицина
S5	Зоопарк	18	101	2	Фауна
S6	Ионосфера	34	351	2	Физика
S7	Лимфография	18	148	4	Медицина
S8	Реабилитация	309	126	2	Жизнь
S9	Сердце	13	270	2	Медицина
S10	Шахматы	36	3196	2	Игра

Каждое решение оценивалось с помощью фитнесс-функции на каждой итерации алгоритма. Данные случайным образом делились на две части: обучающие и тестовые наборы. Первая часть предназначена для обучения классификатора, а вторая – для тестирования. В качестве критерия оценки классификации используется функция потерь (ошибки классификации) для линейных моделей.

Точность классификации определялась отношением правильно классифицированных экземпляров к их общему числу.

Использовались два популярных классификатора: *KNN* (*k* ближайших соседей) и *CART* (классификация и регрессия построением дерева решений). Согласно *KNN* для классификации каждого из объектов тестовой выборки необходимо последовательно выполнить следующие операции: вычислить расстояние до каждого из объектов обучающей выборки; отобрать *k* объектов обучающей выборки, расстояние до которых минимально; определить наиболее часто встречающийся класс среди *k* ближайших соседей в зависимости от расстояния между новым и обучающим экземплярами. Другим классификатором является *CART*, который строит бинарное дерево решений, где каждый узел дерева при разбиении имеет только двух потомков. На каждом шаге построения дерева правило, формируемое в узле, делит заданное множество наборов данных на часть, в которой выполняется правило, и часть, в которой правило не выполняется. Оценочная функция, используемая алгоритмом *CART*, базируется на идее уменьшения неопределённости в узле. В алгоритме *CART* идея неопределенности формализована в индексе *Gini*.

Цель любого классификатора – получить наибольшую точность, то есть лучшим решением является то, которое минимизирует ошибку классификации, минимизирует количество выбранных признаков классификации и максимизирует точность классификации. Процесс классификации по алгоритму *GEBA* останавливался при достижении максимального числа итераций. В целях сравнения с конкурирующими алгоритмами максимальное число итераций k_{max} во всех экспериментах устанавливалось равным 50, значение параметра размера популяции – 20. Эксперименты проводились с использованием *Matlab* на процессоре *Core i5* с ОЗУ 8 Гбайт.

Для сравнения алгоритма *GEBA* с современными конкурирующими алгоритмами использовались следующие критерии: ошибка классификации (сравниваются максимальные, средние и минимальные значения фитнесс-функции); средний размер выборки; *T*-критерий суммы рангов Уилкоксона для проверки различий между выборками по уровню количественного признака; критерий Фридмана – непараметрический статистический тест с 5%-м уровнем значимости.

В ходе экспериментов производительность *GEBA* для решения задачи выбора значимых признаков классификации сравнивалась с

алгоритмами *BA*, кукушки (*Cuckoo Search Algorithm, CSA), GWO* и гибридным алгоритмом роя частиц и гравитационного поиска (*Gravitational Search Algorithm, GSA*) *PSO-GSA*.

В качестве настроек конкурирующих алгоритмов использовались следующие параметры, указанные в оригинальных работах. Для *BA* скорости импульсного излучения r равна 0,1; громкость испускаемого летучей мышью звука A равна 0,25; параметр $\mathcal{F}_{min} = 0$, $\mathcal{F}_{max} = 2$. Для *CSA* параметр $fl = 2$. Для *PSO-GSA* начальное значение гравитационной постоянной $g_0 = 1$, а факторы $c_1' = -2\frac{t^3}{T^3} + 2, c_1' = -2\frac{t^3}{T^3}$, где T – максимальное число итераций, t-текущая итерация, коэффициент α равен 20. Для *GEBA* параметр $\mathcal{F}_{min} = 0{,}1$; $\mathcal{F}_{max} = 2$. Алгоритм *GWO* не предполагает настройки параметров.

В табл. 2.7 представлены лучшие, средние и худшие значения фитнесс-функции, полученные конкурирующими алгоритмами *BA, CSA, GWO, PSO-GSA*, и алгоритмом *GEBA* за 50 итераций.

Анализ результатов показывает следующее.

Для классификатора *KNN* алгоритм *GEBA* получает наилучшее среднее фитнесс-значение для 6 из 10 тестируемых наборов данных (*S3, S4, S5, S8 S9* и *S10*), в то время как *PSO-GSA* лидирует для 5 из 10 наборов данных (*S7, S1, S4, S6* и *S2*). Алгоритмы *BA, CSA* и *GWO* получили наилучшее среднее фитнесс-значение для 1 из 10 тестируемых наборов данных (*S10*). Для классификатора *KNN* алгоритм *GEBA* получает максимальные фитнесс-значения для 6 из 10 тестируемых наборов данных (*S3, S4, S5, S8, S9* и *S10*), как и алгоритмы *BA* и *PSO-GSA*. Алгоритм *CSA* на большинстве наборов данных дает наихудшие результаты.

Для классификатора *CART* алгоритм *GEBA* получает наилучшее среднее фитнесс-значение для 7 из 10 тестируемых наборов данных (*S1, S3, S4, S5, S7, S9* и *S10*), а лучший из конкурирующих алгоритмов *PSO-GSA* – на 3 из 10. Алгоритм *GEBA* получает максимальные фитнесс-значения для 6 из 10 тестируемых наборов данных, а наихудшие результаты на большинстве наборов данных дает *CSA*.

Таблица 2.7

Лучшие, средние и худшие значения фитнесс-функции, полученные алгоритмами *BA*, *CSA*, *GWO*, *PSO-GSA* и *GEBA*

№ набора	Классификатор *k* ближайших соседей					Классификатор *CART*				
	BA	*CSA*	*GWO*	*PSO-GSA*	*GEBA*	*BA*	*CSA*	*GWO*	*PSO-GSA*	*GEBA*
S1	**0,251**	0,278	0,259	0,255	0,266	0,237	0,246	0,236	0,229	**0,226**
	0,277	0,300	0,280	**0,271**	0,275	0,251	0,260	0,247	0,241	**0,238**
	0,299	**0,328**	0,299	0,293	0,291	0,268	**0,272**	0,263	0,266	0,256
S2	0,048	0,082	0,048	**0,034**	0,048	0,135	0,135	0,139	0,125	**0,120**
	0,075	0,098	0,075	**0,057**	0,063	0,163	0,163	0,175	**0,157**	0,165
	0,101	**0,120**	0,096	0,091	0,091	0,188	0,188	0,197	0,178	**0,202**
S3	**0,021**	0,028	**0,021**	0,035	**0,021**	**0,021**	**0,021**	**0,021**	0,028	**0,021**
	0,032	0,040	0,038	0,044	**0,030**	0,026	0,027	0,027	0,033	**0,025**
	0,049	0,056	0,056	**0,076**	0,049	0,063	0,035	0,035	**0,069**	0,028
S4	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,012**	0,013	**0,012**	0,013	0,013
	0,000	**0,000**	**0,000**	**0,000**	**0,000**	**0,015**	0,016	**0,015**	0,016	**0,015**
	0,001	0,000	0,000	**0,004**	0,000	0,021	0,020	**0,018**	0,020	0,018
S5	**0,000**	0,010	**0,000**	**0,000**	**0,000**	**0,020**	0,040	0,030	**0,020**	**0,020**
	0,007	0,022	0,012	0,009	**0,000**	0,029	0,054	0,044	0,035	**0,025**
	0,010	**0,040**	0,020	**0,040**	0,000	0,050	**0,069**	0,059	0,050	0,040
S6	0,046	0,074	0,063	**0,046**	0,054	**0,048**	0,060	**0,048**	**0,048**	0,051
	0,066	0,085	0,074	**0,059**	0,062	0,061	0,066	0,062	**0,057**	0,060
	0,080	**0,094**	0,083	0,074	0,074	0,071	0,071	**0,074**	0,071	0,071
S7	0,108	0,122	0,108	**0,101**	0,115	0,108	0,128	**0,101**	0,108	0,115
	0,127	0,162	0,128	**0,120**	0,123	0,139	0,148	0,139	0,133	**0,132**
	0,149	**0,196**	0,162	0,149	0,135	**0,176**	0,169	0,155	0,155	0,162
S8	0,294	0,317	0,333	0,230	**0,222**	**0,111**	0,119	0,127	**0,111**	0,119
	0,341	0,344	0,348	0,318	**0,310**	0,138	0,137	0,141	**0,133**	0,137
	0,357	**0,365**	**0,365**	**0,365**	**0,365**	**0,159**	0,151	**0,159**	**0,159**	0,151
S9	**0,156**	**0,156**	**0,156**	**0,156**	**0,156**	0,143	0,146	0,143	**0,139**	**0,139**
	0,169	0,198	0,178	0,172	**0,163**	0,159	0,166	0,157	0,157	**0,155**
	0,190	**0,231**	0,221	0,224	0,190	0,173	**0,187**	0,173	0,180	0,170
S10	**0,022**	0,052	**0,022**	0,023	**0,022**	**0,007**	0,018	**0,007**	**0,007**	**0,007**
	0,026	0,069	0,030	0,027	**0,024**	**0,008**	0,038	**0,008**	0,009	**0,008**
	0,041	**0,096**	0,043	0,032	0,027	0,014	**0,059**	0,009	0,014	0,009

T-критерий суммы рангов Уилкоксона и критерий Фридмана с доверительным уровнем 0,95 показывают, что алгоритм *GEBA* является статистически значимым по сравнению с *BA, CSA* и *GWO*. Алгоритм *GEBA* занимает лидирующие позиции, полученными за 50 независимых запусков для каждого из 10 наборов данных, в сравнении с *PSO-GSA*, как для классификатора *KNN*, так и для *CART*.

Табл. 2.8 иллюстрирует устойчивость сравниваемых алгоритмов.

Таблица 2.8

Стандартное отклонение для алгоритмов *BA, CSA, GWO, PSO-GSA, GEBA*

№ набора	Классификатор *k* ближайших соседей					Классификатор *CART*				
	BA	*CSA*	*GWO*	*PSO-GSA*	*GEBA*	*BA*	*CSA*	*GWO*	*PSO-GSA*	*GEBA*
S1	0,011	0,014	0,010	0,008	**0,006**	0,010	**0,007**	0,008	0,010	0,008
S2	0,014	**0,009**	0,014	0,014	0,011	0,016	**0,012**	0,014	0,013	0,019
S3	0,008	0,011	0,011	0,011	**0,005**	0,009	0,005	**0,003**	0,009	**0,003**
S4	**0,000**	**0,000**	**0,000**	0,001	**0,000**	0,002	0,002	**0,001**	0,002	**0,001**
S5	0,005	0,008	0,006	0,009	**0,000**	0,011	0,009	**0,008**	0,010	**0,008**
S6	0,011	0,007	**0,005**	0,007	0,006	0,005	**0,004**	0,006	0,007	0,005
S7	0,013	0,017	0,016	0,011	**0,007**	0,016	**0,012**	0,014	0,015	**0,012**
S8	0,016	0,012	**0,009**	0,046	0,050	0,013	0,009	**0,007**	0,012	0,009
S9	0,010	0,023	0,019	0,019	**0,008**	**0,008**	0,011	0,009	0,011	**0,008**
S10	0,005	0,011	0,005	0,003	**0,001**	0,001	0,011	0,001	0,002	**0,000**

Устойчивость *GEBA* выше, нежели у конкурирующих алгоритмов на 7 из 10 наборах данных (*S1, S3, S4, S5, S7, S9* и *S10*) при

использовании классификатора *KNN*, и на 6 из 10 наборах данных (*S3, S4, S5, S7, S9* и *S10*) при использовании классификатора *CART*. Главное преимущество алгоритма *GEBA* заключается в том, что у него меньше корректируемых переменных, что влияет на производительность алгоритма оптимизации.

Полученные экспериментальные результаты позволяют утверждать, что *GEBA* позволяет повысить производительность и устойчивость его работы при решении задачи выбора значимых классификационных признаков, в сравнении с конкурирующими метаэвристиками. Результаты, приведенные в табл. 2.7–2.8, демонстрируют наилучшую среднюю точность алгоритма *GEBA* для классификаторов *KNN* и *CART* в большинстве используемых наборов данных, в том числе для самого большого набора данных *S*16.

Сравнительный анализ скорости сходимости алгоритмов показывает, что *GEBA* быстрее сходится к глобальным оптимумам, нежели конкурирующие алгоритмы *BA*, *CSA, GWO* и *PSO-GSA*. Основной причиной здесь является комбинирование модифицированных процедур *BA* и *DE* с использованием механизма их выбора в зависимости от текущего состояния решения, при соблюдении баланса между скоростью сходимости алгоритма и диверсификацией пространства поиска решений. Самую низкую скорость сходимости демонстрирует алгоритм *CSA*. Это согласуется с результатами, представленными в табл. 2.8.

2.5. Алгоритм империалистической конкуренции

Алгоритм империалистической конкуренции (*ICA*), предложенный в [15], основан на действиях, предпринимаемых отдельными странами для расширения своей власти, как правило, путем колонизации других территорий.

На этапе инициализации *ICA* начинается со случайной генерации популяции из N агентов (страны), каждый из которых представляет решение $\boldsymbol{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]$ как совокупность социально-политических характеристик (культура, язык, экономика, религия и т. д.).

Затем множество решений в зависимости от фитнесс-функции классифицируются как империалистические страны N_{imp}: $(\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_{Nimp})$ и страны-колонии $N_{col} = N - N_{imp}$: $(\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_{Ncol})$. Каждая из N_{col} колоний пропорционально распределяется между N_{imp} странами для формирования империй. Число колоний в империи пропорционально значениям фитнесс-функций империалистической страны согласно:

$$power_i = \left| \frac{cost_i}{\sum_{j=1}^{N_{imp}} cost_j} \right|, \text{ где} \tag{2.51}$$

$$cost_i = \max_j \{f(\boldsymbol{i}_j)\} - f(\boldsymbol{i}_i). \tag{2.52}$$

Число колоний, закрепленных за каждой империалистической страной, определяется как:

$$NC_i = round(power_i \cdot N_{col}). \tag{2.53}$$

Итеративный процесс *ICA* включает этапы ассимиляции, революции и конкуренции.

Ассимиляция предполагает, что каждая колония, входящая в состав империи, находится под влиянием социально-политических факторов соответствующей империалистической страны, что заставляет колонии становиться ближе к империи. Указанное влияние представляется как движение колонии $\boldsymbol{c}_{ij}$ к $\boldsymbol{i}_j$ согласно:

$$\boldsymbol{c}_{ij}^* = \boldsymbol{c}_{ij} + rand(0, \beta) \cdot (\boldsymbol{i}_j - \boldsymbol{c}_{ij}), \tag{2.54}$$

где $\beta \in [1, 2]$.

Затем, на этапе революции, предполагается, что некоторые случайно выбранные колонии испытывают изменения в своих социально-политических характеристиках (аналогично оператору мутации в *GA*). Революция вызывает внезапные случайные изменения в положении некоторых стран в области поиска. Колония $\boldsymbol{c}_{ij}$ после некоторых изменений может превратиться в империалистическую, и наоборот. Во время ассимиляции и революции колония может достигнуть лучшего положения и имеет шанс взять на себя управление всей империи, заменить текущее империалистическое состояние империи.

Империалистическая конкуренция – другая часть этого алгоритма. Все империи пытаются выиграть эту игру и овладеть колониями других империй. В каждом шаге алгоритма, основанного на их власти, у всех империй есть шанс взять под свой контроль один или больше колоний самой слабой империи. На этапе конкуренции слабая колония внутри слабой империи может передаваться другой империи с вероятностью:

$$P_j = \left| \frac{NTC_j}{\sum_{i=1}^{N_{imp}} NTC_i} \right|, \tag{2.55}$$

где NTC_j – нормализованная общая стоимость j-й империи согласно:

$$NTC_j = \max_i \{TC_j\} - TC_j, \; TC_j = f(\boldsymbol{i}_j) + \xi \cdot mean\{f(c_{ij})\}, \tag{2.56}$$

где $\xi \in [0, 1]$ – параметр, отражающий влияние, которое оказывают власти колонии на власти империи. В результате конкуренции более

слабые империи постепенно теряют свои колонии, а более сильные империи овладевают новыми колониями. Более слабые империи со временем рушатся, пока не останется только одна самая сильная империя.

ICA с успехом применяется для решения различных задач оптимизации в разных областях разработки и науки. Например, при проектировании оптимальной аэрокосмической конструкции из многослойных композитных материалов, при диспетчеризации промышленных систем, проектирование интеллектуальных рекомендательных систем, оптимизации в системах связи, обучении и анализе искусственных нейронных сетей, а также при решении различных задач многокритериальной оптимизации.

2.6. Резюме

Алгоритм дифференциальной эволюции. Алгоритм *DE* представляет собой одну из «непрерывных» модификаций канонического генетического алгоритма с одной существенной особенностью – в качестве источника шума используется не внешний генератор случайных чисел, а внутренний, реализованный как разность между случайно выбранными векторами текущей популяции.

В качестве начальной популяции выбирается случайное множество из N векторов из пространства R^n. Обычно используется выборка из n-мерного равномерного или нормального распределения с заданными математическим ожиданием и дисперсией. На очередном k-ом шаге алгоритма производится скрещивание каждой особи X из исходной популяции со случайно выбранной особью, отличной от X. Координаты этих векторов рассматриваются как генетические признаки. Перед кроссинговером применяется специальный оператор мутации – в котором участвуют не исходные, а искаженные (мутантные) генетические признаки особи, согласно (2.1). В качестве шума, искажающего «генофонд» особи, используется не внешний источник энтропии, а внутренний – разность между случайно выбранными представителями популяции.

Кроссинговер производится следующим образом. Задается вероятность *rand*, с которой потомок наследует очередной (искаженный мутацией) генетический признак от родителя. Соответствующий признак от родителя X наследуется с вероятностью $(1 - rand)$, n раз разыгрывается бинарная случайная величина с математическим ожиданием *rand*, и для единичных ее значений производится

наследование (перенос) искаженного генетического признака от одного родителя, а для нулевых значений – от другого родителя.

Селекция производится на каждом шаге алгоритма. После формирования вектора-потомка производится сравнение его целевой функции и целевой функции его «прямого» родителя *X*. В новую популяцию отбирается тот вектор, у которого целевая функция достигает меньшего значения (для задачи минимизации). Это гарантирует постоянный размер популяции в процессе работы алгоритма. На сходимость алгоритма *DE* сильнее всего влияют размер популяции и сила мутации *F*.

Синус-косинусный алгоритм. Вначале случайным образом инициализируется популяция агентов. Следующим шагом является обновление позиции каждого агента для поиска оптимального решения. С этой целью положение агента оценивается с использованием целевой функции, и на основе критериев оптимизации каждому агенту присваивается значение пригодности или полезности. Поисковый агент с наибольшей пригодностью считается лучшим агентом, а его позиция обозначается как пункт назначения. После определения местоположения пункта назначения другие агенты поиска обновляют свое местоположение в пространстве поиска, используя пункт назначения в качестве ориентира. Для этой цели *SCA* использует тригонометрические функции синуса и косинуса (2.9). Параметр r_1 обусловливает сдвиг в обновлении решения в сторону наилучшего решения или за его пределы; параметр r_2 определяет, насколько велико смещение решения к месту назначения или за его пределы; параметр r_3 используется для присвоения веса пункту назначения, усиливая или подавляя влияние пункта назначения на процесс обновления других решений; параметр r_4 представляет собой переключение между функциями синуса и косинуса.

Модифицированный синус-косинусный алгоритм. В модификациях базового алгоритма *SCA* предлагается использовать полеты Леви для вычисления параметра r_1 и линейное уменьшении инерционного веса $w(t)$. Это способствует балансу между скоростью сходимости алгоритма и диверсификацией пространства поиска решений, однако временная сложность алгоритма при этом значительно увеличивается. Чтобы этого избежать, в уравнение для обновления положения поисковых агентов вводится инерционный вес $w(t)$ согласно (2.11), а параметр r_1 вычисляется согласно (2.12).

Тестирование *MSCA* на многомерных тестовых функциях и инженерной задаче проектирования пружины растяжения/сжатия показывает его преимущества перед известными конкурирующими метаэвристиками по точности и времени поиска решения.

Алгоритм эгоистичного стада. Коэволюционный самонастраивающийся алгоритм (КОСА) моделирует два различных типа поисковых агентов: стадо травоядных особей и стая хищников, которые охотятся за добычей. Оба типа поисковых агентов индивидуально управляются набором различных эвристических операторов на основе поведенческих аспектов, наблюдаемых в отношениях «жертва-хищник». Первым шагом алгоритма является случайная инициализация популяции H из $N = N_p + N_r$ травоядных и хищников. Каждой особи присваивается значение функции живучести, которая зависит от расстояния относительно самой безопасной и самой рискованной позиции согласно (2.14) – (2.16). Популяция травоядных особей включает вожака, группу особей, стремящихся занять позицию ближе к центру стада, и особей, которые движутся относительно независимо от стада, что связано с текущей внутренней структурой и паттернами движения стада. Руководит стадом вожак – особь с наибольшими способностями к выживанию согласно (2.17). Популяция разделяется на группу особей, остающихся в стаде, и группу особей-дезертиров согласно (2.19) и (2.20).

Для моделирования движения стада травоядных в алгоритме используются различные эвристические операторы: движение вожака и операторы движения остальных особей, с учетом положения хищников включая особей-дезертиров, согласно (2.23) – (2.35). Алгоритм моделирует также движение и нападение стаи хищников на травоядных согласно (2.36) – (2.41). Алгоритм включает вычислительную процедуру, которая позволяет восстановить популяцию, используя оператор кроссинговера особей стада согласно (2.42) – (2.43).

Тестирование алгоритма КОСА на многомерных тестовых функциях показывает его преимущества перед известными конкурирующими метаэвристиками по точности и времени поиска решения.

Алгоритм летучих мышей. Модель поведения летучих мышей включает испускание громких частотно-модулированных звуковых импульсов и прием эхо-звуков, которые отражаются от окружающих объектов. Алгоритм *BA* использует следующие параметры, значения

которых корректируются в процессе поиска: частота, громкость и скорость импульсного излучения согласно (2.44), (2.45). При обновлении местоположения летучей мыши применяется операторы (2.46) – (2.49).

Был предложен гиперэвристический алгоритм *GEBA*, который является комбинацией метаэвристик летучих мышей и дифференциальной эволюции. Он включает вероятностный механизм их выбора в зависимости от текущего состояния решения и был разработан для решения задачи выбора значимых признаков классификации и оценки точности классификации. В качестве данных использовался мировой репозиторий данных и модельных задач машинного обучения *UCI*. Репозиторий содержит реальные данные по прикладным задачам в области биологии, медицины, физики, техники, социологии и др. В качестве критерия оценки классификации используется функция потерь (ошибки классификации) для линейных моделей. Использовались два популярных классификатора: *KNN* (*k* ближайших соседей) и *CART* (классификация и регрессия построением дерева решений). Для сравнения алгоритма *GEBA* с современными конкурирующими алгоритмами использовались следующие критерии: ошибка классификации (сравниваются максимальные, средние и минимальные значения фитнесс-функции); средний размер выборки; *T*-критерий суммы рангов Уилкоксона для проверки различий между выборками по уровню количественного признака; критерий Фридмана – непараметрический статистический тест с 5%-м уровнем значимости. Полученные экспериментальные результаты позволяют утверждать, что *GEBA* позволяет повысить производительность и устойчивость его работы при решении задачи выбора значимых классификационных признаков, в сравнении с конкурирующими метаэвристиками.

Алгоритм империалистической конкуренции. Модель алгоритма основана на действиях, предпринимаемых отдельными странами для расширения своей власти, как правило, путем колонизации других территорий. Инициализации алгоритма *ICA* начинается со случайной генерации популяции из *N* агентов (страны), каждый из которых представляет решение как совокупность социально-политических характеристик (культура, язык, экономика, религия и т. д.). Затем множество решений в зависимости от фитнесс-функции классифицируются как империалистические страны и страны-колонии. Число колоний в империи пропорционально значениям

фитнесс-функций империалистической страны согласно (2.51) – (2.53). *ICA* включает этапы ассимиляции, революции и конкуренции согласно (2.54) – (2.56).

Алгоритм успешно применялся для проектирования оптимальной аэрокосмической конструкции, диспетчеризации промышленных систем, в интеллектуальных рекомендательных системах, а также при оптимизации систем связи, обучении искусственных нейронных сетей и при решении различных задач многокритериальной оптимизации.

3. МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ, ИСПОЛЬЗУЮЩИЕ ВСЮ ПОПУЛЯЦИЮ АГЕНТОВ

3.1. Алгоритм роя саранчи

Колонии насекомых предоставляют богатый набор метафор для разработки сбалансированных метаэвристик. Такие кооперативные образования представляют собой сложные системы, состоящие из агентов с различными задачами и специализированными паттернами поведения в зависимости от своего типа. В большинстве метаэвристик используются поисковые агенты с одинаковыми свойствами и паттернами поведения. В таких условиях операторы эти алгоритмов теряют аттрактивность, не позволяют улучшить разнообразие популяции и расширить пространство поиска оптимальных решений. Поэтому включение в алгоритм операторов, моделирующих индивидуальные поведенческие характеристики агентов популяции способствует появлению вычислительных механизмов, улучшающих баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска.

Роевой алгоритм и алгоритм дифференциальной эволюции являются наиболее популярными метаэвристиками для решения сложных оптимизационных задач. Однако они имеют определенные недостатки, связанные с преждевременной сходимостью и трудностями преодоления локальных оптимумов.

В частности, в роевом алгоритме проблемы связаны с операторами, которые изменяют местоположение агентов роя и обновляют положение каждого агента роя на очередной итерации, что приводит к их перемещению в направлении лучшей в данный момент особи. В алгоритме дифференциальной эволюции новое решение выбирается для дальнейшего поиска только если оно улучшает предыдущее решение. В результате вся популяция концентрируется вокруг лучшего решения или хаотично расходится в процессе поиска. Это способствует нарушению баланса между скоростью сходимости алгоритма и диверсификацией пространства поиска. Между тем, коллаборативное роевое поведение относительно простых агентов в популяции обеспечивает их выживание, используя ограниченную локальную информацию и несложные правила поведения.

Саранча является репрезентативным примером насекомых, которые могут сочетать роевое и индивидуальное поведение. Это различные паттерны поведения. Индивидуальное поведение предполагает, что саранча избегает контактов и, как следствие, рой распределяется по всему пространству поиска. Роевое поведение предполагает концентрацию саранчи вокруг особей, которым удалось найти источник пищи.

Используем для описания паттернов индивидуального и роевого поведения роя саранчи известную биологическая модель. Согласно этой модели, опишем вначале изменение положения отдельной саранчи при индивидуальном поведении. Пусть что $\boldsymbol{x}_i^k$ представляет собой текущее положение i-й саранчи в рое из N особей. Тогда новое положение $\boldsymbol{x}_i^{k+1}$ особи вычисляется по следующей формуле:

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \Delta \boldsymbol{x}_i \tag{3.1}$$

где $\Delta \boldsymbol{x}_i$ соответствует изменению положения i-й саранчи на (k+1)-й итерации в результате взаимодействия с другими особями роя.

Две саранчи при индивидуальном поведении не стремятся сблизиться, если между ними небольшое расстояние и, наоборот, сближаются, поддерживая сплоченность роя, если между ними значительное расстояние. Сила притяжения/отталкивания определяется как разность:

$$s(r) = kar \cdot e^{-\frac{r}{lim}} - e^{-r} \tag{3.2}$$

Здесь r – расстояние между парой особей, kar – коэффициент притяжения/отталкивания, lim – пограничное значение расстояния между особями. Если $kar < 1$ и $lim > 1$, то это означает, что между особями небольшое расстояние и отталкивание сильнее нежели притяжение. Тогда сила воздействия j-й саранчи на i-ю саранчу определяется как:

$$\boldsymbol{s}_{ij} = s(r_{ij}) \cdot \boldsymbol{d}_{ij} \tag{3.3}$$

где $r_{ij} = |\boldsymbol{x}_j - \boldsymbol{x}_i|$ – расстояние между j-й и i-й особями роя, а $\boldsymbol{d}_{ij} = (\boldsymbol{x}_j - \boldsymbol{x}_i)/ r_{ij}$ – единичный вектор. Тогда общая сила притяжения/отталкивания роя для i-й саранчи определяется как суперпозиция всех парных взаимодействий:

$$\boldsymbol{S}_i = \sum_{j=1,\ j\neq i}^{N} \boldsymbol{s}_{ij}. \tag{3.4}$$

Изменение положения i-й саранчи $\Delta \boldsymbol{x}_i$ соответствует (3.4):

$$\Delta \boldsymbol{x}_i = \boldsymbol{S}_i. \tag{3.5}$$

В отличие от индивидуального поведения при роевом поведении саранча стремительно концентрируется вокруг особей, которые нашли источники пищи. Для того чтобы смоделировать роевое поведение, вводится для каждой саранчи $\boldsymbol{x}_i$ индекс пищи f_i ($f_i \in [0, 1]$

Далее N особей популяции ранжируются по убыванию этого индекса, а затем среди них выбираются b особей ($b << N$) с наивысшими показателями пищи. Вокруг каждой из b особей в радиусе R_c случайным образом концентрируются подмножество саранчи.

Представим алгоритм роевого и индивидуального поведения саранчи (MAPC) [16].

Предположим, что все пространство поиска представляет собой плантацию, где саранча взаимодействует друг с другом. Каждое решение в пространстве поиска представляет положение саранчи на плантации и характеризуется значением функции пригодности, отражающей уровень пищевого индекса. Алгоритм реализует паттерны индивидуального и роевого поведения, которые управляются набором операторов, моделирующих эти поведенческие паттерны.

Популяция $\boldsymbol{L}^k(\{\boldsymbol{l}_1^k, \boldsymbol{l}_2^k,..., \boldsymbol{l}_N^k\})$ из N особей эволюционирует из начального положения ($k = 0$) к заданному числу поколений ($k = gen$). Каждая саранча $\boldsymbol{l}_i^k$ ($i = 1..N$) представляет собой n-мерный вектор $\{l_{i1}^k, l_{i2}^k,..., l_{in}^k\}$, в котором каждый элемент соответствует переменной решения задачи оптимизации. Множество переменных решений составляет допустимое пространство поиска $\boldsymbol{S} = \{\boldsymbol{l}_i^k \in \boldsymbol{R}^n \,|lb_d \le l_{id}^k \le ub_d\}$, где lb_d и ub_d соответствуют нижней и верхней границам размерности d соответственно. Уровень пищевого индекса, связанный с каждой саранчой, оценивается с помощью функции $f_i(\boldsymbol{l}_i^k)$.

В алгоритме MAPC на каждой итерации процесса эволюции применяются два оператора поведения: $\boldsymbol{A}$ – индивидуальный и $\boldsymbol{B}$ – роевой. Оператор $\boldsymbol{A}$ применяется для диверсификации пространства поиска решений, а оператор $\boldsymbol{B}$ – для уточнения решения в определенной области пространства.

Рассмотрим подробнее каждый из операторов.

Оператор $\boldsymbol{A}$, реализующий паттерн индивидуального поведения саранчи, изменяет текущее положение $\boldsymbol{l}_i^k$ i-й ($i = 1..N$) саранчи на величину $\Delta\boldsymbol{l}_i^k$: $\boldsymbol{p}_i = \boldsymbol{l}_i^k + \Delta\boldsymbol{l}_i^k$ с учетом значения функции пригодности и положения доминирующих особей роя. Сила притяжения/отталкивания между j-й и i-й особями, рассчитывается как

$$\boldsymbol{s}_{ij}^m = \rho\left(\boldsymbol{l}_i^k, \boldsymbol{l}_j^k\right) \cdot s\left(r_{ij}\right) \cdot \boldsymbol{d}_{ij} + rand(1,-1) \qquad (3.6)$$

где $s(r_{ij})$ определяется согласно (2); $\boldsymbol{d}_{ij} = (\boldsymbol{l}_j^k - \boldsymbol{l}_i^k)/\, r_{ij}$ – единичный вектор, направленный от $\boldsymbol{l}_i^k$ к $\boldsymbol{l}_j^k$; $rand(-1, 1)$ – случайное число из интервала (–1, 1); $\rho(\boldsymbol{l}_i^k, \boldsymbol{l}_j^k)$ – функция доминирования между j-й и i-й особями. Для определения ρ все особи популяции $\boldsymbol{L}^k(\{\boldsymbol{l}_1^k, \boldsymbol{l}_2^k,..., \boldsymbol{l}_N^k\})$

ранжируются по убыванию их функций пригодности. Лучшей особи присваивается ранг 0, худшая особь получает ранг N–1. Таким образом, функция $\rho(\boldsymbol{l}_i^k, \boldsymbol{l}_j^k)$ определяется следующим образом:

$$\rho(\boldsymbol{l}_i^k, \boldsymbol{l}_j^k) == \begin{cases} e^{-(5\cdot rank(l_i^k)/N)}, & \text{если } rank(l_i^k) < rank(l_j^k) \\ e^{-(5\cdot rank(l_j^k)/N)}, & \text{если } rank(l_i^k) > rank(l_j^k) \end{cases}, \quad (3.7)$$

где функция $rank(\alpha)$ указывает на ранг особи. Согласно (3.7) функция ρ принимает значения из интервала (0, 1), причем значение 1 достигается, когда одна из особей является лучшим элементом популяции, а значение близкое к 0 – когда обе особи обладают низкими показателями функции пригодности.

Наконец, общая сила притяжения/отталкивания, действующая на i-ю особь, вычисляется как суперпозиция всех парных взаимодействий:

$$\boldsymbol{S}_i^m = \sum_{j=1,\ j\neq i}^{N} \boldsymbol{s}_{ij}^m \quad (3.8)$$

Изменение положения $\Delta\boldsymbol{l}_i$ определяется следующим образом:

$$\Delta\boldsymbol{l}_i = \boldsymbol{S}_i^m. \quad (3.9)$$

После вычисления новых позиций $\boldsymbol{P}(\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\})$ особей популяции $\boldsymbol{L}^k$, необходимо изменить значения функций пригодности $\boldsymbol{F}(\{\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_N.\})$. При этом допускаются только те изменения, которые гарантируют улучшение результатов поиска. Иными словами, если $f_i(\boldsymbol{p}_i) > f_i(\boldsymbol{l}_i^k)$, то принимается новая позиция $\boldsymbol{p}_i$, иначе – сохраняется позиция $\boldsymbol{l}_i^k$:

$$f_i = \begin{cases} p_i, \text{если } f(p_i) < f(l_i^k), \\ l_i^k, \text{в противном случае.} \end{cases} \quad (3.10)$$

Оператор $\boldsymbol{B}$, реализующий паттерн роевого поведения саранчи, направлен на уточнение решения в определенной области пространства поиска. Для его выполнения вначале проводится сортировка функций пригодности особей по убыванию. Результаты сортировки сохраняются в множестве $\boldsymbol{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N\}$. Среди них выделяются g наилучших особей, имеющих наибольшее значение функции пригодности. Они образуют подмножество $\boldsymbol{E}$ наиболее перспективных решений. Вокруг каждой особи с $f_i \in \boldsymbol{E}$ создается подпространство C_j радиусом, который определяется следующим образом:

$$e_d = \frac{\sum_{q=1}^{n}(ub_q - lb_q)}{n} \cdot \beta, \quad (3.11)$$

где ub_q и lb_q – верхняя и нижняя границы в q-м измерении, n – размерность переменных оптимизационной задачи, $\beta \in [0, 1]$ – параметр алгоритма. Границы подпространства C_j моделируются следующим образом:

$$uss_j^q = b_{j,q} + e_d, lss_j^q = b_{j,q} - e_d\ , \quad (3.12)$$

где $uss^q{}_j$ и $lss^q{}_j$ – верхняя и нижняя границы в q-м измерении подпространства C_j соответственно. Внутри этого подпространства случайно генерируются h ($h < 4$) новых особей, среди которых выбирается особь с наилучшим значением функции пригодности.

Таким образом, алгоритм МАРС имеет 5 настраиваемых параметров: коэффициент притяжения/отталкивания *kar*, пограничное значение расстояния между особями *lim*, количество наилучших решений *g*, размер популяции *N* и число поколений *gen*.

Алгоритм включает шаги инициализации, выполнения индивидуального и роевого операторов.

При инициализации ($k = 0$) формируется начальная популяция $\boldsymbol{L}^0(\{\boldsymbol{l}_1{}^0, \boldsymbol{l}_2{}^0, ..., \boldsymbol{l}_N{}^0\})$. Значения $(\{l_{i1}{}^0, l_{i2}{}^0, ..., l_{in}{}^0\}$ каждого отдельного измерения d распределены случайным образом и равномерно между предварительно заданной нижней начальной границей параметра lb_d и верхней начальной границей параметра ub_d:

$$l_{ij}^0 = lb_d + rand \cdot (ub_d - lb_d), \quad (3.13)$$

где $i = 1..N$, $d = 1..n$. В процессе итеративного выполнения алгоритма индивидуальный оператор $\boldsymbol{A}$ и роевой оператор $\boldsymbol{B}$ исполняются до тех пор, пока не будет достигнуто число итераций $k = gen$. Псевдокод алгоритма МАРС имеет следующий вид:

1: ***Input***: *kar, lim, g, N, gen;*
2: Инициализация $\boldsymbol{L}^0(k{=}0)$;
3: ***until*** ($k{=}gen$);
4: $\boldsymbol{F} \leftarrow$ оператор$\boldsymbol{A}(\boldsymbol{L}^k)$;
5: $\boldsymbol{L}^{k+1} \leftarrow$ оператор$\boldsymbol{B}(\boldsymbol{L}^k, \boldsymbol{F})$;
6: $k = k + 1$;
7: ***end until***.

В отличие от многих и роевых алгоритмов, данная метаэвристика на основе паттернов поведения саранчи позволяет избежать концентрации особей на текущих наилучших позициях, снижает вероятность преждевременной сходимости, поддерживает баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска решений.

Алгоритм МАРС экспериментально проверялся на многомерных ($n = 30$) тестовых функциях Розенброка ($f_1(X)$ роз), сферической ($f_2(X)$ сф), Экли ($f_3(X)$ экл), Швефеля ($f_4(X)$ шв), квадратов ($f_5(X)$ ква), Растригина ($f_6(X)$ рас), Саломона ($f_7(X)$ сал):

- функция Розенброка:

$$f_1(X) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2];$$
$$x_i \in [-30, 30]^n,\ x^* = (1,\ldots,1),\ f_1(x^*) = 0;$$

- сферическая функция:

$$f_2(X) = \sum_{i=1}^{n} x_i^2,\ x_i \in [-100, 100]^n,$$
$$x^* = (0,\ldots,0),\ f_2(x^*) = 0;$$

- функция Экли:

$$f_3(X) = -20\exp\left(\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20;$$
$$x_i \in [-32, 32]^n,\ x^* = (0,\ldots,0),\ f_3(x^*) = 0;$$

- функция Швефеля:

$$f_4(X) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2,\ x_i \in [-100, 100]^n,$$
$$x^* = (0,\ldots,0),\ f_4(x^*) = 0;$$

- сумма квадратов:

$$f_5(X) = \sum_{i=1}^{n} i x_i^2,\ x_i \in [-10, 10]^n,$$
$$x^* = (0,\ldots,0),\ f_5(x^*) = 0;$$

- функция Растригина:

$$f_6(X) = \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i) + 10],$$
$$x_i \in [-5{,}12;\ 5{,}12]^n,\ x^* = (0,\ldots,0),\ f_6(x^*) = 0;$$

- функция Саломона:

$$f_7(X) = -\cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0{,}1\sqrt{\sum_{i=1}^{n} x_i^2 + 1},$$
$$x_i \in [-100, 100]^n,\ x^* = (0,\ldots,0),\ f_7(x^*) = 0.$$

Здесь n – размерность функции, x^* – оптимальное решение, $f_i(x^*)$ – минимальное значение функции.

Эксперименты проводились в программной среде на языке программирования *C#*. При отладке и тестировании использовался компьютер *IBM PC* с процессором *Core i7* с ОЗУ-8 Гб.

В экспериментах использовались следующие настройки параметров алгоритма МАРС: *kar* = 0,6, *lim* = 1, *N* = 50, g = 20, *gen* = 1000.

По каждой функции проводилось 30 прогонов. Затем полученные результаты усреднялись. Определялись среднее значение по лучшим решениям (*Mean best*), медианное лучшее решение (*Med. best*) и стандартное отклонение от лучшего решения (*St. dev*). Усредненные результаты алгоритма МАРС по 30 отдельным запускам, приведены в таблице 3.1.

Таблица 3.1

Результаты работы алгоритма MAPC

Функция	$f_1(X)$ роз	$f_2(X)$ сф	$f_3(X)$ экл	$f_4(X)$ шв	$f_5(X)$ ква	$f_6(X)$ рас	$f_7(X)$ сал
Mean best	$1{,}47 \cdot 10^{-02}$	$9{,}23 \cdot 10^{-06}$	$3{,}43 \cdot 10^{-05}$	$7{,}98 \cdot 10^{-02}$	$9{,}99 \cdot 10^{-04}$	$8{,}14 \cdot 10^{-02}$	$7{,}56 \cdot 10^{-02}$
Med. best	$1{,}86 \cdot 10^{-02}$	$7{,}95 \cdot 10^{-06}$	$2{,}65 \cdot 10^{-05}$	$6{,}62 \cdot 10^{-02}$	$5{,}17 \cdot 10^{-04}$	$6{,}96 \cdot 10^{-02}$	$6{,}28 \cdot 10^{-02}$
St. dev	$6{,}33 \cdot 10^{-03}$	$2{,}17 \cdot 10^{-06}$	$3{,}71 \cdot 10^{-06}$	$2{,}76\ 10^{-02}$	$1{,}76\ 10^{-04}$	$5{,}39\ 10^{-03}$	$2{,}24\ 10^{-03}$

3.2. Алгоритм колонии пауков

В [17] был представлен алгоритм оптимизации, моделирующий кооперативное поведение колонии социальных пауков, и биологическая модель их взаимодействия. Используем эту модель для представления паттернов поведения в распределенной самоорганизующейся сети и построения модифицированного алгоритма колонии пауков (МАКП) [18].

Основными функциями, характеризующими поведение пауков в колонии, являются строительство сетевой паутины, размножение и охота. При этом инструментом взаимодействия агрегации пауков является сеть. Сеть представляет собой канал связи через вибрации и постукивания. Вибрации используются пауками для синхронизации и декодирования своих действий, а также для авторизации особи, передающей сообщение. По интенсивности вибраций и постукиваний определяется вес паука и расстояние до него. Каждый шаг паука вызывает вибрацию. Поэтому, например, при охоте колония перестает двигаться, чтобы по сети почувствовать добычу и понять правильное направление движения. Колония использует

определенную тактику и командную работу для защиты своих ресурсов и потомства с дифференциацией ролей.

Определим согласно биологической модели, следующие паттерны поведения пауков: размножение и кооперация. Альфа-самцы пауков отличаются большим весом в сравнении с остальными самцами. При размножении они стремятся двигаться по сети к ближайшей самке. В отличие от альфа-самцов, недоминирующие самцы, в основном, сосредотачиваются в центре колонии, чтобы воспользоваться ресурсами альфа-самцов.

С учетом этого МАКП включает следующую последовательность шагов.

Шаг 1. Инициализация популяции пауков $S = B \cup M$ размером N. Поскольку согласно биологической модели, в колонии преобладают самки, то их число N_b определяется по формуле:

$$N_b = floor[(0{,}9 - rand \cdot 0{,}25) \cdot N]$$

где *rand* – случайное число на интервале [0, 1], *floor*(·) функция преобразования действительных чисел в целые числа. Общее число самцов $N_m = N - N_b$.

В целом колония социальных пауков S размером N состоит из подмножеств $B = \{b_1, b_2, \ldots, b_{Nb}\}$ и $M = \{m_1, m_2, \ldots, m_{Nm}\}$.

Шаг 2. Случайная инициализация позиций пауков. Позиции самцов b_i и самок m_j генерируются случайно и равномерно. Для этого задаются нижний начальный параметр $q_k^{\text{ниж}}$ и верхний начальный параметр $q_k^{\text{верх}}$:

$$b_{i,k}^0 = q_k^{\text{ниж}} + rand(0{,}1) \cdot (q_k^{\text{верх}} - q_k^{\text{ниж}}),$$
$$m_{j,k}^0 = q_k^{\text{ниж}} + rand(0{,}1) \cdot (q_k^{\text{верх}} - q_k^{\text{ниж}}),$$

где $i = 1..N_b$, $j = 1..N_m$, $k = 1..n$, функция *rand*(0, 1) генерирует случайное число в интервале от 0 до 1.

Шаг 3. Диапазон взаимодействия в колонии социальных пауков определяется размерами пространства поиска по следующей формуле:

$$r = \frac{\sum_{k=1}^{n}(q_k^{\text{верх}} - q_k^{\text{ниж}})}{2 \cdot n}.$$

Шаг 4. Вероятность участия в процессе размножения зависит от веса паука и вычисляется по методу рулетки:

$$P_{s_i} = \frac{w_i}{\sum_{k \in T^g} w_k},$$

где T^g – множество пауков в радиусе r. При этом сохраняется первоначальная пропорция между самками и самцами в популяции S.

Шаг 5. Позиция самки b_i на итерации алгоритма $(t+1)$ изменяется по формуле:

$$b_i^{t+1} = \begin{cases} b_i^t + \alpha \cdot Vib_{i,u} \cdot (m_u - b_i^t) + \beta \cdot Vib_{i,maxw} \cdot (m_{maxw} - b_i^t) + \delta \cdot \left(rand - \frac{1}{2}\right) \\ \text{с вероятностью } PF \\ b_i^t - \alpha \cdot Vib_{i,u} \cdot (m_u - b_i^t) - \beta \cdot Vib_{i,maxw} \cdot (m_{maxw} - b_i^t) + \delta \cdot \left(rand - \frac{1}{2}\right) \\ \text{с вероятностью } 1 - PF, \end{cases}$$

Здесь моделируются вибрации паутины. Вибрации зависят от веса и расстояния до паука, который их породил. Иными словами, особи рядом с пауком, создающим вибрации, улавливают их как более сильные, нежели особи, находящиеся вдали. Вибрация, которую i-й паук улавливает от паука j-го паука, определяется как $Vib_{i,j}$

$$Vib_{i,j} = w_j \cdot e^{-d_{i,j}^2},$$

где $d_{i,j} = \|s_i - s_j\|$ – евклидово расстояние между i-м и j-м пауками. Модификация алгоритма заключается в использовании в МАКП следующих вариантов вибраций:

• $Vibc_i = w_c \cdot e^{-d_{ci}^2}$. Это вибрации, улавливаемые i-м пауком от ближайшего к нему паука c, обладающим большим весом ($w_c > w_i$);

• $Vibb_{i,maxw} = w_{maxw} \cdot e^{-d_{bi,maxw}^2}$. Это вибрации, улавливаемые i-м пауком от паука $maxw$ с максимальным весом в колонии;

• $Vibf_i = w_f \cdot e^{-d_{fi}^2}$. Это вибрации, улавливаемые i-м пауком от ближайшей самки f.

Шаг 6. Выполняется оператор, моделирующий движение самца паука:

$$m_i^{t+1} = \begin{cases} m_i^t + \alpha \cdot Vib_{i,b} \cdot (s_b - m_i^t) + \delta \cdot \left(rand - \frac{1}{2}\right), \text{если } w_{N_{\text{ср}}} > w_i \\ m_i^t + \alpha \cdot \left(\frac{\sum_{h=1}^{N_m} m_h^t \cdot w_{N_b+h}}{\sum_{h=1}^{N_m} w_{N_b+h}} - m_i^t\right), \text{если } w_i > w_{N_{\text{ср}}} \end{cases},$$

где s_b – ближайшая к пауку самка, а величина $\left(\frac{\sum_{h=1}^{N_m} m_h^t \cdot w_{N_b+h}}{\sum_{h=1}^{N_m} w_{N_b+h}} - m_i^t\right)$ представляет средневзвешенный вес самцов M в колонии.

Оператор определяет различные поведенческие паттерны, что способствует диверсификации поиска оптимума. К тому же уменьшается влияние очень хороших и плохих решений на результаты поиска.

Шаг 7. Выполняется оператор размножения в определенном диапазоне r, который вычисляется как:

$$P_{s_i} = \frac{w_i}{\sum_{k \in T^g} w_k},$$

где T^g – множество пауков в радиусе r, которые участвуют в размножении.

Шаг 8. Останов алгоритма происходит при достижении заданного максимального числа итераций. Иначе – возврат к шагу 4 алгоритма.

Алгоритм МАРС экспериментально проверялся на многомерных (n = 30) тестовых функциях Розенброка ($f_1(X)$ роз), сферической ($f_2(X)$ сф), Экли ($f_3(X)$ экл), Швефеля ($f_4(X)$ шв), квадратов ($f_5(X)$ ква), Растригина ($f_6(X)$ рас), Саломона ($f_7(X)$ сал).

Эксперименты проводились в программной среде на языке программирования *C#*. При отладке и тестировании использовался компьютер *IBM PC* с процессором *Core i7* с ОЗУ – 8 Гб.

В экспериментах использовался следующий параметр настройки алгоритма МАКП: PF = 0,7.

По каждой функции проводилось 30 прогонов. Затем полученные результаты усреднялись. Определялись среднее значение по лучшим решениям (*Mean best*), медианное лучшее решение (*Med. best*) и стандартное отклонение от лучшего решения (*St. dev*). Усредненные результаты алгоритма МАКП по 30 отдельным запускам, приведены в таблице 3.2.

Таблица 3.2

Результаты работы алгоритма МАКП

Функция	$f_1(X)$ роз	$f_2(X)$ сф	$f_3(X)$ экл	$f_4(X)$ шв	$f_5(X)$ ква	$f_6(X)$ рас	$f_7(X)$ сал
Mean best	$8{,}77\cdot10^{-02}$	$5{,}12\cdot10^{-05}$	$4{,}22\cdot10^{-05}$	$8{,}77\cdot10^{-02}$	$8{,}98\cdot10^{-04}$	$7{,}88\cdot10^{-02}$	$8{,}24\cdot10^{-02}$
Med. best	$5{,}13\cdot10^{-02}$	$4{,}12\ 10^{-05}$	$1{,}52\cdot10^{-05}$	$5{,}13\cdot10^{-02}$	$6{,}62\cdot10^{-04}$	$6{,}74\cdot10^{-02}$	$5{,}79\cdot10^{-03}$
St. dev	$2{,}68\ 10^{-02}$	$8{,}82\ 10^{-06}$	$8{,}16\cdot10^{-06}$	$2{,}68\ 10^{-02}$	$1{,}76\ 10^{-04}$	$2{,}13\ 10^{-03}$	$3{,}20\ 10^{-03}$

3.3. Гибридный алгоритм модифицированных алгоритмов роя саранчи и колонии пауков

Комбинирование нескольких алгоритмов, инспирированных природой, при решении целого класса сложных задач глобальной оптимиза-

ции может оказаться более эффективным, нежели использование одного алгоритма. Перспективным направлением здесь является гибридизация различных алгоритмов глобальной оптимизации. Существуют различные способы гибридизации алгоритмов. Одним из них является последовательная комбинация алгоритмов (препроцессор/постпроцессор). Основная идея этого способа гибридизации заключается в том, чтобы на начальных этапах обеспечить широкий обзор всевозможных решений, а на последующих этапах сужать области поиска.

Сценарий предлагаемой гибридизации включает поиск решений с помощью алгоритма МАКП в качестве препроцессора, а затем проверку и выявление лучшего решения с помощью алгоритма МАРС. Триггером перехода с алгоритма препроцессора является отсутствие улучшения глобального оптимума. Алгоритм постпроцессора стартует, используя, в основном, решения, полученные на заключительном шаге алгоритма препроцессора с исключением решений, близких к области одного и того же экстремума.

Для оценки эффективности предлагаемого гибридного алгоритма глобальной оптимизации МАКП-МАРС, основанного на модифицированных алгоритмах роя саранчи и колонии пауков, были проведены эксперименты с использованием тестовых многомерных функций.

Эксперименты проводились в программной среде на языке программирования *C#*. При отладке и тестировании использовался компьютер *IBM PC* с процессором *Core i7* с ОЗУ– 8 Гб.

Множество тестовых функций составляли тестовая функция Розенброка ($f_1(X)$ роз), сферическая ($f_2(X)$ сф), Экли ($f_3(X)$ экл), Швефеля ($f_4(X)$ шв), квадратов ($f_5(X)$ ква), Растригина ($f_6(X)$ рас) и Саломона ($f_7(X)$ сал).

Результаты тестирования гибридного алгоритма МАКП-МАРС сопоставлялись с алгоритмами пчелиной колонии (*ABC*), дифференциальной эволюции (*DE*), роя частиц (*PSO*), МАРС и МАКП.

В экспериментах использовались следующие настройки параметров алгоритмов:

- *ABC* – параметр *limit* = 100;
- *DE* – параметр $F \in [0{,}4; 1{,}0]$, параметр $p_m \in [0{,}0; 1{,}0]$;
- *PSO* – коэффициенты обучения $c_1 = 2$ и $c_2 = 2$, вес w линейно уменьшается с 0,9 до 0,2 при увеличении числа итераций;
- МАКП – параметр $PF = 0{,}7$;
- МАРС – параметры $kar = 0{,}6$, $lim = 1$, $N = 50$, g = 20, $gen = 1000$.

По каждой функции и каждому алгоритму проводилось 30 прогонов. Затем полученные результаты усреднялись.

Показателями, по которым проводилось сравнение, являлись среднее значение по лучшим решениям, медианное лучшее решение и стандартное отклонение от лучшего решения. Усредненные результаты по 30 отдельным запускам, приведены в табл. 3.3.

Также оценивалась статистическая значимость полученных результатов. С этой целью применялся *T*-критерий суммы рангов Уилкоксона для независимых выборок, найденных каждым из сравниваемых алгоритмов на 30 тестовых запусках, при уровне значимости 5 %. Значение T < 0,05 рассматривалось как адекватное доказательство против нулевой гипотезы, которая отвергается.

Таблица 3.3

Результаты сравнения гибридного алгоритма МАКП-МАРС с алгоритмами *ABC, DE, PSO*, МАРС и МАКП

Функция	*ABC*	*DE*	*PSO*	МАРС	МАКП	МАКП-МАРС
$f_1(X)$ роз	$9{,}26\cdot10^{-02}$ $3{,}24\cdot10^{-01}$ $1{,}81\ 10^{-01}$	$2{,}27\cdot10^{-02}$ $2{,}23\cdot10^{-02}$ $5{,}03\cdot10^{-03}$	$2{,}73\cdot10^{-02}$ $2{,}61\cdot10^{-02}$ $\mathbf{5{,}83\cdot10^{-03}}$	$1{,}47\cdot10^{-02}$ $1{,}86\cdot10^{-02}$ $6{,}33\cdot10^{-03}$	$8{,}77\cdot10^{-02}$ $5{,}13\cdot10^{-02}$ $2{,}68\ 10^{-02}$	$\mathbf{1{,}02\cdot10^{-02}}$ $\mathbf{8{,}15\cdot10^{-03}}$ $7{,}17\cdot10^{-03}$
$f_2(X)$ сф	$7{,}08\cdot10^{-03}$ $5{,}16\cdot10^{-03}$ $1{,}86\ 10^{-03}$	$1{,}97\cdot10^{-05}$ $5{,}43\cdot10^{-05}$ $1{,}03\cdot10^{-05}$	$8{,}44\cdot10^{-03}$ $3{,}12\cdot10^{-02}$ $1{,}60\cdot10^{-03}$	$9{,}23\cdot10^{-06}$ $\mathbf{7{,}95\cdot10^{-06}}$ $2{,}17\cdot10^{-06}$	$5{,}12\cdot10^{-05}$ $4{,}12\ 10^{-05}$ $8{,}82\ 10^{-06}$	$\mathbf{3{,}38\cdot10^{-07}}$ $9{,}49\cdot10^{-06}$ $\mathbf{1{,}12\cdot10^{-06}}$
$f_3(X)$ экл	$6{,}26\cdot10^{-02}$ $5{,}95\cdot10^{-02}$ $1{,}33\ 10^{-03}$	$7{,}01\cdot10^{-04}$ $7{,}20\cdot10^{-04}$ $2{,}21\cdot10^{-04}$	$3{,}57\cdot10^{-02}$ $4{,}82\cdot10^{-02}$ $1{,}15\cdot10^{-03}$	$\mathbf{3{,}43\cdot10^{-05}}$ $2{,}65\cdot10^{-05}$ $3{,}71\cdot10^{-06}$	$4{,}22\cdot10^{-05}$ $1{,}52\cdot10^{-05}$ $8{,}16\cdot10^{-06}$	$4{,}05\cdot10^{-05}$ $\mathbf{2{,}02\cdot10^{-06}}$ $\mathbf{1{,}15\cdot10^{-06}}$
$f_4(X)$ шв	$2{,}34\cdot10^{-01}$ $6{,}77\cdot10^{-01}$ $2{,}72\ 10^{-01}$	$8{,}26\cdot10^{-01}$ $7{,}35\cdot10^{-01}$ $1{,}66\ 10^{-01}$	$8{,}62\cdot10^{-01}$ $5{,}25\cdot10^{-01}$ $1{,}12\ 10^{-01}$	$7{,}98\cdot10^{-02}$ $6{,}62\cdot10^{-02}$ $2{,}76\ 10^{-02}$	$8{,}77\cdot10^{-02}$ $5{,}13\cdot10^{-02}$ $2{,}68\ 10^{-02}$	$\mathbf{3{,}20\cdot10^{-03}}$ $\mathbf{7{,}57\cdot10^{-03}}$ $\mathbf{2{,}82\ 10^{-03}}$
$f_5(X)$ ква	$2{,}47\cdot10^{-03}$ $5{,}72\cdot10^{-03}$ $6{,}63\ 10^{-04}$	$2{,}47\cdot10^{-03}$ $5{,}70\cdot10^{-03}$ $\mathbf{1{,}58\ 10^{-04}}$	$6{,}96\cdot10^{-02}$ $5{,}49\cdot10^{-02}$ $3{,}33\ 10^{-02}$	$9{,}99\cdot10^{-04}$ $\mathbf{5{,}17\cdot10^{-04}}$ $1{,}76\ 10^{-04}$	$8{,}98\cdot10^{-04}$ $6{,}62\cdot10^{-04}$ $1{,}76\ 10^{-04}$	$\mathbf{1{,}46\cdot10^{-05}}$ $7{,}38\cdot10^{-04}$ $1{,}76\ 10^{-04}$
$f_6(X)$ рас	$9{,}58\cdot10^{-01}$ $9{,}18\cdot10^{-01}$ $2{,}19\ 10^{-01}$	$8{,}26\cdot10^{-01}$ $7{,}40\cdot10^{-01}$ $2{,}28\ 10^{-01}$	$8{,}48\cdot10^{-01}$ $8{,}28\cdot10^{-01}$ $1{,}16\ 10^{-01}$	$8{,}14\cdot10^{-02}$ $6{,}96\cdot10^{-02}$ $5{,}39\ 10^{-03}$	$7{,}88\cdot10^{-02}$ $6{,}74\cdot10^{-02}$ $2{,}13\ 10^{-03}$	$\mathbf{2{,}82\cdot10^{-03}}$ $\mathbf{8{,}44\cdot10^{-03}}$ $\mathbf{1{,}28\ 10^{-03}}$
$f_7(X)$ сал	$4{,}21\cdot10^{-02}$ $5{,}64\cdot10^{-01}$ $3{,}49\ 10^{-02}$	$6{,}52\cdot10^{-01}$ $7{,}46\cdot10^{-01}$ $3{,}35\ 10^{-01}$	$9{,}35\cdot10^{-01}$ $8{,}04\cdot10^{-01}$ $2{,}48\ 10^{-01}$	$7{,}56\cdot10^{-02}$ $6{,}28\cdot10^{-02}$ $2{,}24\ 10^{-03}$	$8{,}24\cdot10^{-02}$ $5{,}79\cdot10^{-03}$ $3{,}20\ 10^{-03}$	$\mathbf{4{,}81\cdot10^{-03}}$ $\mathbf{3{,}62\cdot10^{-03}}$ $\mathbf{2{,}03\ 10^{-04}}$

Гибридный алгоритм МАКП-МАРС превосходит конкурирующие, а экспериментальные результаты по алгоритму являются статистически значимыми.

3.4. Алгоритм гравитационного поиска

Алгоритм гравитационного поиска (*Gravitational Search Algorithm, GSA*) – это метаэвристика, основанная на некотором упрощении закона всемирного тяготения Ньютона и закона движения [19].

Алгоритм включает множество поисковых агентов, взаимодействующих между собой посредством силы гравитации. Агенты имеют массу, менее тяжелые агенты движутся к более тяжелым. Сила гравитации все время воздействует на всех агентов. Сила гравитации между двумя агентами прямо пропорциональна произведению их масс и обратно пропорциональна квадрату расстояния между ними:

$$F = G \cdot \frac{M_1 \cdot M_2}{R^2}, \quad (3.13)$$

Здесь, F – сила гравитационного притяжения, G – гравитационная постоянная, M_1 и M_2 – массы агентов, R – расстояние между агентами.

Второй закон Ньютона гласит, что если сила F воздействует на частицу, то ее ускорение a зависит только от этой силы и от своей массы:

$$a = \frac{F}{M}. \quad (3.14)$$

Отметим, что из-за эффекта уменьшения гравитации фактическое значение гравитационной постоянной G зависит от возраста вселенной. Поэтому гравитационная постоянная – это убывающая функция, зависящая от времени:

$$G(t) = G(t_0) \cdot \left(\frac{t_0}{t}\right)^{\beta}, \beta < 1.$$

Положение агента представляет собой решение задачи, причем самая тяжелая масса представляет собой оптимальное решение в пространстве поиска.

Алгоритм *GSA* включает следующие шаги:

1. Задание начальных параметров (N_{max} – максимальное число итераций алгоритма, N – количество агентов в системе, $G(t_0)$ – начальное значение гравитационной постоянной).

2. Инициализация агентов. Позиции N агентов инициализируются случайным образом путем равномерного распределения по всей области поиска:

$$X_i = \left(x_i^l, \dots, x_i^d, \dots, x_i^n\right), i = 1..N,$$

где x^d_i представляет позицию i-го агент в d-мерном пространстве поиска.

Инициализировать вектор скоростей $v^d_j(0) = 0,\ j = 1..n$.

3. Основной цикл. Для t = 1, … , N_{max}:

а) Вычислить значение фитнесс-функции для каждого агента $fit_i(t)$, найти худшее (минимальное) и лучшее (максимальное) из них:

$$w(t) = \min_{j=1..n} fit_j(t), b(t) = \max_{j=1..n} fit_j(t).$$

б) Вычислить массу каждого агента $M_i(t)$:

$$m_i(t) = \frac{fit_i(t) - w(t)}{b(t) - w(t)}, M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{n} m_i(t)}.$$

в) Вычислить силу гравитации, т. е. воздействия агента k на агента i:

$$F_{ik}^d = G(t) \cdot \frac{M_i M_k}{R_{ik} + \epsilon} \cdot \left(x_k^d(t) - x_i^d(t)\right),$$

где ε – некоторая малая константа, R_{ik} – евклидово расстояние между агентами i и k (установлено, что использование в знаменателе R вместо R^2 дает лучшие результаты работы алгоритма).

г) Вычислить суммарную силу притяжения, воздействующую на агента i:

$$F_i^d(t) = \sum_{k=1, k \neq i}^{N} \alpha_i F_{ik}^d(t),$$

где α_i равномерно распределенное число на отрезке [0, 1].

д) Вычислить ускорение агента:

$$a_i^d(t) = \frac{F_i^d(t)}{M_i^d(t)}.$$

е) Вычислить новые скорости:

$$v_i^d(t+1) = \alpha_i v_i^d(t) + a_i^d(t).$$

ж) Вычислить новую позицию каждого агента:

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1).$$

и) Вычислить гравитационную постоянную $G(t) = G(G_0, t)$.

Лучшее значение фитнесс-функции на последней итерации определяет положение соответствующего агента и является решением задачи.

Алгоритм *GSA* был значительно модифицирован с момента его появления в 2009 г. с целью повышения его производительности. Результаты сравнивались с исходным *GSA* и с другими известными алгоритмами, такими как *PSO* и *GA*. Практика применения *GSA* позволяет сделать вывод, что алгоритм позволяет получать более точные результаты и большую скорость сходимости, нежели алгоритмы *GA* и *PSO*. Однако эксперименты на тестовых многомерных мультимодальных функциях показывают, что *GSA* быстро сходиться к некоторому локальному оптимуму, из которого сложно выбраться, так как не предусмотрены процедуры, похожие на мутации в генетических алгоритмах.

Кроме того, известны многочисленные попытки гибридизации *GSA* с другими алгоритмами (*PSO, ABC, GA*), а также с моделями и методами машинного обучения (нейронные сети, *k-means*, нечеткая логика, *SVM*). Эти гибриды *GSA* продемонстрировали лучшие результаты в сравнении с конкурирующими алгоритмами.

В настоящее время *GSA,* его модификации и гибриды успешно применяются в различных областях и приложениях: бизнесе, биоинформатике, разработке программного обеспечения и в инженерном проектировании для решения разнообразных задач распознавания образов, кластеризации, классификации.

3.5. Светлячковый алгоритм

Светлячковый алгоритм (*Frefly Algorithm, FA*) – метаэвристика роевого интеллекта, ориентированная на оптимизацию функции и поиск ее глобального оптимума. В роли поисковых агентов выступают светлячки. В основе алгоритма лежит наблюдаемое в природе поведение светлячков, которые излучают свет для коммуникации между особями – с его помощью они привлекают особей противоположного пола, сообщают о приближении хищников и т.д.

Менее яркие светлячки перемещаются к более ярким; яркость одного светлячка, воспринимаемая другим, уменьшается при его удалении. Если светлячок не видит более яркого представителя роя, он перемещается хаотично. Каждый светлячок характеризуется яркостью и позицией. Первоначально задается положение каждой особи в определенном интервале случайным образом. Далее высчитывается яркость светлячка по формуле:

$$int = \frac{1}{f(x) + 1},$$

где $f(x)$ – значение фитнесс-функции (единица прибавляется, чтобы исключить деление на 0).

Если яркость i-го светлячка меньше яркости j-го светлячка, то перемещаем первого в направлении второго:

$$pos(i) = pos_0(i) + \alpha \cdot c + \beta \cdot \big(pos(i) - pos(j)\big), \beta = \beta_0 \cdot e^{-\gamma \cdot r^2},$$

где γ – коэффициент поглощения света среды, r – расстояние между агентами, $pos_0(i)$ – начальное положение i-го светлячка, $pos(i)$ – новое положение i-го светлячка, β – привлекательность j-го светлячка для i-го светлячка, β_0 – привлекательность светлячка, когда они находятся

вплотную друг к другу, *c* –- случайно выбранное число из промежутка [0; 1], α – свободный параметр рандомизации.

Если после перемещения светлячка его положение выходит за пределы определенного интервала, то задаем его случайным образом на данном промежутке. После перебора всех особей происходит сортировка светлячков по убыванию их яркости. Для удобства вводится переменная, численно равная лучшей позиции поискового агента *BestPos*. После заданного числа итераций происходит вывод наиболее оптимального решения.

Классический алгоритм *FA* прост в реализации. Сама идея алгоритма восхищает своей оригинальностью. Однако исследования показывают, что он медленно сходится и легко попадает в ловушку локального экстремума для мультимодальных задач оптимизации. Кроме того, в нем отсутствуют коэффициенты, параметрически зависимые от текущей итерации. Следовательно, модификация *FA* для повышения его производительности является актуальной задачей.

Известны модификации алгоритма, в основном, связанные с заменой случайной компоненты полетом Леви. Отличие в том, что в базовом алгоритме светлячок с наилучшей светимостью перемещается в случайном направлении от своего текущего положения, тогда как при использовании полетов Леви лучший светлячок будет двигаться от лучшего глобального положения, пытаясь улучшить не свое текущее положение, а решение в целом. К координатам лучшего решения прибавляется случайное число распределения полета Леви (распределение с тяжелыми хвостами) с тем же коэффициентом с учетом вектора перемещений. Это позволяет улучшить координаты глобального решения, уточняя окрестность поиска. При этом сохраняется высокая масштабируемость алгоритма, высокая скорость сходимости, устойчивость к застреванию в локальных экстремумах.

Модифицированные *FA* успешно применялись для решения следующих задач:

- оптимизации маршрутов в телекоммуникационных сетях, для нахождение подходящих маршрутов передачи данных, учитывая различные факторы, такие как пропускная способность, задержка и стоимость связи;
- оптимизация распределения ресурсов в системах энергоснабжения, например распределение производства энергии между раз-

личными источниками, такими как солнечные панели, ветрогенераторы и гидроэлектростанции, чтобы достичь баланса между производством и потреблением энергии;

- оптимизация параметров моделей, настройка гиперпараметров моделей, таких как коэффициенты регуляризации, скорость обучения и число скрытых слоев в нейронных сетях;
- оптимизация планирования ресурсов в производственных системах, определение оптимального распределения рабочих мест, оборудования и материалов на производственных линиях для повышения эффективности производства и снижения затрат;
- обучение нейронных сетей;
- оптимизация портфеля инвестиций между различными активами, учитывая риск и доходность каждого актива, с целью максимизации прибыли и управления риском.

3.6. Резюме

Алгоритм роя саранчи. Саранча является репрезентативным примером насекомых, которые могут сочетать роевое и индивидуальное поведение с разными паттернами поведения. Индивидуальное поведение предполагает, что саранча избегает контактов и, как следствие, рой распределяется по всему пространству поиска. Роевое поведение предполагает концентрацию саранчи вокруг особей, которым удалось найти источник пищи. Изменение положения отдельной саранчи при индивидуальном поведении определяется согласно (3.1), а сила притяжения/отталкивания – согласно (3.3–3.5). Алгоритм реализует паттерны индивидуального и роевого поведения, которые управляются набором операторов, моделирующих индивидуальный и роевой поведенческие паттерны согласно (3.6) – (3.12). Алгоритм позволяет избежать концентрации особей на текущих наилучших позициях, снижает вероятность преждевременной сходимости, поддерживает баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска решений.

Тестирование алгоритма на многомерных тестовых функциях показывает его преимущества перед известными конкурирующими метаэвристиками по точности и времени поиска решения.

Алгоритм колонии пауков. Основными функциями, характеризующими поведение пауков в колонии, являются строительство сетевой паутины, размножение и охота. При этом инструментом взаимодействия агрегации пауков является сеть. Сеть представляет

собой канал связи через вибрации и постукивания. Вибрации используются пауками для синхронизации и декодирования своих действий, а также для авторизации особи, передающей сообщение. По интенсивности вибраций и постукиваний определяется вес паука и расстояние до него. Колония использует определенную тактику и командную работу для защиты своих ресурсов и потомства с дифференциацией ролей. Согласно биологической модели, алгоритм реализует паттерны поведения пауков: размножение и кооперация. Альфа-самцы пауков отличаются большим весом в сравнении с остальными самцами. При размножении они стремятся двигаться по сети к ближайшей самке. В отличие от альфа-самцов, недоминирующие самцы, в основном, сосредотачиваются в центре колонии, чтобы воспользоваться ресурсами альфа-самцов.

Гибридный алгоритм алгоритмов роя саранчи и колонии пауков. Комбинирование нескольких алгоритмов, инспирированных природой, при решении целого класса сложных задач оптимизации может оказаться более эффективным, нежели использование одного алгоритма. Основная идея способа гибридизации путем последовательной комбинации алгоритмов (препроцессор/постпроцессор) заключается в том, чтобы на начальных этапах обеспечить широкий обзор всевозможных решений, а на последующих этапах сужать области поиска. Триггером перехода с алгоритма препроцессора является отсутствие улучшения глобального оптимума.

Тестирование алгоритма на многомерных тестовых функциях показывает его преимущества перед известными конкурирующими метаэвристиками по точности и времени поиска решения.

Алгоритм гравитационного поиска. Эта метаэвристика основана на некотором упрощении закона всемирного тяготения Ньютона и закона движения. Алгоритм включает множество поисковых агентов, взаимодействующих между собой посредством силы гравитации. Агенты имеют массу, менее тяжелые агенты движутся к более тяжелым. Гравитация воздействует на всех агентов. Сила гравитации между двумя агентами прямо пропорциональна произведению их масс и обратно пропорциональна квадрату расстояния между ними согласно (3.13). Ускорение агента (3.14) зависит от этой силы и от его массы. Положение агента представляет собой решение задачи, причем самая тяжелая масса представляет собой оптимальное решение в пространстве поиска. Основные шаги алгоритма включают инициализацию параметров и случайное равномерное распределение агентов

по всей области поиска. Основной цикл алгоритма заключается в вычислении фитнесс-функции и массы каждого агента, а также силы гравитации между агентами и силы притяжения, воздействующую на агента. После вычисления ускорения агента пересчитывается его скорость и новая позиция. Лучшее значение фитнесс-функции на заключительной итерации алгоритма определяет положение соответствующего агента и является решением задачи.

Практика применения *GSA* показала, что алгоритм позволяет получать более точные результаты и имеет большую скорость сходимости, нежели алгоритмы *GA* и *PSO*. Однако эксперименты на тестовых многомерных мультимодальных функциях показывают, что *GSA* быстро сходиться к некоторому локальному оптимуму, из которого сложно выбраться, так как не предусмотрены процедуры, похожие на мутации в генетических алгоритмах. В то же время известны многочисленные гибриды *GSA* с другими алгоритмами (*PSO, ABC, GA*), а также с моделями и методами машинного обучения (нейронные сети, *k-means*, нечеткая логика, *SVM*). Эти гибриды продемонстрировали лучшие результаты в сравнении с конкурирующими алгоритмами.

Светлячковый алгоритм. В роли поисковых агентов здесь выступают светлячки. В основе алгоритма лежит наблюдаемое в природе поведение светлячков, которые излучают свет для коммуникации между особями – с его помощью они привлекают особей противоположного пола, сообщают о приближении хищников и т. д. Менее яркие светлячки перемещаются к более ярким; яркость одного светлячка, воспринимаемая другим, уменьшается при его удалении. Если светлячок не видит более яркого представителя роя, он перемещается хаотично. Каждый светлячок характеризуется яркостью и позицией. Первоначально задается положение каждой особи в определенном интервале случайным образом. Далее высчитывается яркость светлячка и в соответствии с этим изменяется его позиция. Алгоритм прост, однако медленно сходится и легко попадает в ловушку локального экстремума для мультимодальных задач оптимизации. Известны многочисленные модификации алгоритма *FA*, позволяющие преодолевать указанные трудности и успешно решать задачи оптимизации маршрутов в телекоммуникационных сетях, распределения ресурсов в системах энергоснабжения, обучения нейросетей.

4. МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ, ИСПОЛЬЗУЮЩИЕ СУБПОПУЛЯЦИЮ АГЕНТОВ

4.1. Алгоритм бабочки-монарха

Алгоритм бабочки-монарха (*Monarch Butterfly Optimization, MBO*) – это метаэвристика, моделирующая миграционное поведение бабочек-монархов в природе [2]. Направление движения особей бабочки-монарха в алгоритме *MBO* в основном определяется оператором миграции и специальным корректирующим оператором. *MBO* подходит для параллельной обработки операторов и способен обеспечить баланс между скоростью сходимости алгоритма и диверсификацией пространства поиска решений. Во всяком случае об этом свидетельствуют результаты сравнения производительности *MBO* с многими конкурирующими метаэвристиками на тестовых бенчмарках.

Для решения различных оптимизационных задач используются следующие идеализированные правила поведения бабочек-монархов: (1) вся их популяция находится в двух областях их обитания, (2) особи размножаются и мигрируют из одной области в другую, (3) общая численность популяции поддерживается неизменной, (4) особи с наилучшей функцией пригодности автоматически переходят в следующее поколение.

Упрощая и идеализируя процесс миграции, предположим, что бабочки-монархи находятся в одной области с апреля по август (5 месяцев), а затем мигрируют в другую область с сентября по март (7 месяцев). Обозначим общее количество бабочек-монархов через *NP*, количество бабочек-монархов в области 1 через *NP1* = *ceil*(*p***NP*), количество бабочек-монархов в области 2 через *NP2* = *NP*–*NP1*. Здесь *ceil*(*x*) означает округление *x* до ближайшего целого числа, большего или равного *x*; *p* – доля бабочек-монархов в области 1. Бабочки-монархи в области 1 образуют субпопуляцию 1, в области 2 – субпопуляцию 2. Тогда процесс миграции бабочек-монархов можно выразить следующим образом:

$$x_{i,k}^{t+1} = x_{r1,k}^{t}, \tag{4.1}$$

где $x_{i,k}^{t+1}$ указывает положение *k*-й особи бабочки-монарха x_i в поколении $t + 1$. Аналогично, $x_{r1,k}^{t}$ указывает на случайно выбранную *k*-ю особь бабочки-монарха x_{r1} из субпопуляции 1 в поколении *t*. Если $r \le p$, то *k*-я особь новой бабочки-монарха генерируется с помощью уравнения (4.1).

Здесь

$$r = rand * peri \tag{4.2}$$

указывает на период миграции, равный 1,2 (12 месяцев в году), где *rand* – случайное число, полученное из равномерного распределения. Напротив, если $r > p$, то *k*-я особь новой бабочки-монарха генерируется следующим образом:

$$x_{i,k}^{t+1} = x_{r2,k}^{t}, \tag{4.3}$$

Здесь $x_{r2,k}^{t}$ указывает на случайно выбранную *k*-ю особь бабочки-монарха x_{r2} из субпопуляции 2 в поколении *t*.

В алгоритме *MBO* направление миграции зависит от параметра *p*. Если значение *p* велико, то больше бабочек-монархов выбирается из субпопуляции 1, иначе больше бабочек-монархов выбирается из субпопуляции 2. Обычно значение *p* выбирается равным 5/12 в соответствии с периодом миграции.

Псевдокод оператора миграции имеет следующий вид:

Begin
for *i*=1 *to NP1* ***do***
for *k*=1 *to D* ***do***
генерация *rand*;
if $r \le p$ ***then***
случайный выбор бабочки-монарха из субпопуляции 1 (скажем, *r1*);
генерация *k*-й особи x_i^{t+1} согласно (4.1).
else
случайный выбор бабочки-монарха из субпопуляции 2 (скажем, *r2*);
генерация *k*-й особи x_i^{t+1} согласно (4.3).
end if
end for *k*
end for *i*
End

Кроме оператора миграции, положение бабочки-монарха также может быть изменено специальным корректирующим оператором. Положение *k*-й особи бабочки-монарха x_j в поколении $t + 1$ определяется следующим образом:

$$x_{j,k}^{t+1} = x_{best,k}^{t}, \tag{4.4}$$

если случайно сгенерированное число $rand \le p$. Здесь $x_{best,k}^{t}$ является положением лучшей особи в субпопуляциях 1 и 2 текущего поколения.

Напротив, если *rand* > *p*, то положение *k*-й особи бабочки-монарха x_j в поколении $t + 1$ обновляется как

$$x_{j,k}^{t+1} = x_{r3,k}^{t}, \quad (4.5)$$

где $x_{r3,k}^{t}$ указывает на *k*-ю особь x_{r3}^{t}, случайно выбранную из субпопуляции 2. Здесь $r_3 \in \{1, 2, \dots, NP2\}$. При условии, что *rand* > *BAR*, положение *k*-й особи бабочки-монарха x_j в поколении $t + 1$ обновляется как:

$$x_{j,k}^{t+1} = x_{j,k}^{t+1} + \alpha * (dx_k - 0{,}5), \quad (4.6)$$

где *BAR* указывает на скорость корректировки, а dx_k – шаг корректировки, который рассчитывается согласно полетам Леви:

$$dx = Levy(x_j^t), \quad (4.7)$$

α – весовой коэффициент, который рассчитывается как

$$\alpha = S_{max}/t^2, \quad (4.8)$$

где S_{max} – максимальный шаг корректировки, t – текущее поколение.

Большее значение α приводит к диверсификации пространства поиска решения, а меньшие значения α – к увеличению скорости сходимости.

Псевдокод специального оператора корректировки имеет следующий вид:

Begin
for *j* = 1 *to NP2* ***do***
расчет *dx* согласно (4.7);
расчет весового коэффициента α согласно (4.8);
for *k*=1 *to D* ***do***
генерация *rand*;
if *r* ≤ *p* ***then***
случайный выбор бабочки-монарха из субпопуляции 2
(скажем, *r3*);
генерация *k*-й особи x_j^{t+1} согласно (4.5).
if *rand* > *BAR* ***then***
$x_{j,k}^{t+1} = x_{j,k}^{t+1} + \alpha * (dx_k - 0{,}5)$;
end if
end if
end for *k*
end for *j*
End

В целом алгоритм *MBO* включает следующие шаги.

Begin

1. Инициализация. Установить счетчик поколений $t = 1$; случайным образом инициализировать популяцию из *NP* особей бабочки-монарха; установить максимальное число поколений равным *MaxGen*, число бабочек-монархов *NP1* в субпопуляции 1, и число бабочек-монархов в субпопуляции 2 *NP2*, максимальный шаг корректировки, равным *Max*, скорость корректировки *BAR*, период миграции *peri* и коэффициент миграции *p*.

2. Вычислить функцию пригодности каждой бабочки-монарха в соответствии с ее положением.

3. ***While*** (лучшее решение не найдено ***or*** $t < MaxGen$) ***do***

Ранжировать особей бабочки-монарха в соответствии с их функцией пригодности;

Разделить особей на субпопуляцию 1 и субпопуляцию 2);

for $i = 1$ to $NP1$ **do**

Создать новую субпопуляцию 1 с помощью оператора миграции.

end for *i*

for $j = 1$ to $NP2$ **do**

Создать новую субпопуляцию 2 с помощью специального оператора корректировки.

end for *j*

Объединить две вновь созданные субпопуляции в одну популяцию;

Вычислить функцию пригодности для особей с обновленными позициями;

$t = t+1$

4. ***end while***

5. Вывод лучшего решения

В [21] представлены результаты сравнения производительности алгоритма *MBO* с пятью другими метаэвристиками (*ABC, ACO, BBO, DE, SGA*) с помощью 38 тестовых задач поиска оптимума для многомерных функций. Результаты показывают, что алгоритм *MBO* находит лучшие значения функций в большинстве задач в сравнении с конкурирующими алгоритмами.

Кроме того, алгоритм *MBO* прост и не содержит сложных вычислений и операторов. Это делает реализацию алгоритма *MBO*

простой и быстрой. Несмотря на преимущества *MBO* обращают на себя внимание следующие проблемы: необходимость настройки параметров, используемых в алгоритме с помощью теоретического анализа или экспериментов; анализ алгоритма в таких приложениях, как сегментация изображений, планирование, проектирование и транспортная логистика; алгоритм уступает конкурирующим алгоритмам по показателю средней производительности и стандартному отклонению, а также для низкоразмерных функций. Необходимо также теоретически проанализировать сходимость *MBO* с помощью динамических систем и цепей Маркова.

4.2. Алгоритм червей

Алгоритм оптимизации червей (*Worm Optimization, WO*), представленный в [22], основан на поведении червя нематоды, имеющего всего 302 нейрона. Тем не менее, это позволяет червям выполнять несколько сложных действий, включая выживание, смену индивидуального и коллективного стилей поиска пищи, избежание токсинов и другие. Алгоритм *WO* превзошел алгоритмы муравьиной колонии *ACO*, роя частиц *PSO* и генетического *GA* в *NP*-сложной задаче о коммивояжере.

Чтобы решить задачу оптимизации с помощью *WO*, ее необходимо представить в виде графа, содержащего вершины и дуги, в котором червь будет перемещаться от одной вершины к другой в процессе поиска оптимального решения.

WO начинается с отложения феромона. Первоначально определенное количество феромона τ_{ij} (обычно $\tau_{ij} = 0{,}01$) откладывается в каждой дуге графа. Черви перемещаются от узла к другому в соответствии с вероятностью, которая частично аналогична вероятности при оптимизации муравьиным алгоритмом [7]. Вероятность определяется тремя факторами: количеством феромона (τ), доступностью (η) и коэффициентом плохого решения (*ADF*). Вероятность перемещения из вершины *i* в вершину *j* для червя *k* вычисляется согласно:

$$P_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}ADF_{ij}}{\sum_{l\in\Psi}\tau_{il}^{\alpha}\eta_{il}^{\beta}ADF_{ij}}, \qquad (4.10)$$

где Ψ представляет собой множество еще не посещенных вершин, а величина η определяется в зависимости от того, является ли поведение червя коллективным или индивидуальным. Важными параметрами для направления поиска являются α (обычно $\alpha = 1{,}5$)

и β (обычно $\beta \in [0{,}5;\ 4{,}5]$), которые являются степенными показателями в (4.10) и определяют важность количества феромонов в сравнении с величиной доступности η. Затем величина феромона обновляется в соответствии со следующим уравнением:

$$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 + \rho), \qquad (4.11)$$

если дуга (i, j)) используется червем k. Здесь ρ (обычно $\rho = 0{,}01$) параметр обновления феромона (величина, на которую увеличивается количество феромона при обнаружении хорошего решения).

Рассмотрим подробнее основные операторы алгоритма *WO*.

Оператор, моделирующий паттерны добывания пищи. Параметр *RMG* указывает на индивидуальное поведение червя (если *RMG* = 1) или на коллективное поведение (если *RMG* = 0). Если *RMG* $\in [0{,}4;\ 0{,}7]$, то это не исключает ни того ни другого стиля поведения. После инициализации *WO*, в соответствии с *RMG*, каждый червь помечается как склонный к коллективному или индивидуальному поведению. Если червь склонен к коллективному поведению, он будет привлечен феромоном; в противном случае он с равной вероятностью переместится в другие вершины.

Оператор, позволяющий избежать токсинов. После перемещения червя в другую вершину, необходимо оценить данное решение. С этой целью формируется список *ADF*, в котором хранятся до h плохих решений, где $h = \lceil \sqrt{Worm} \rceil$. Здесь *Worm* определяет количество червей в алгоритме на этапе инициализации.

Оператор, моделирующий передвижение червя. В процессе моделирования передвижения червя вводится вероятность *AIY* (обычно *AIY* $\in [0{,}1;\ 0{,}5]$) проведения локального поиска. Чем она выше, тем выше вероятность локального поиска. В этом случае две вершины из *Worm* выбираются случайным образом и меняются местами. Если это приводит к лучшему решению, то оно сохраняется; в противном случае локальный поиск продолжается до тех пор, пока либо не будет найдено лучшее решение, либо не будет достигнуто заданное максимальное количество итераций.

Что касается концентрации червей в пространстве поиска, то ее лучше оценивать по их количеству. Для каждого червя количество итераций обновляется до тех пор, пока оно не достигнет своего максимума (обычно от 5000 до 15000), что указывает на высокую концентрацию червей.

Известны и другие примеры применения алгоритма *WO*, в частности для задачи минимизации времени параллельной обработки изделий на нескольких машинах, оптимального размещения изделий на складе.

4.3. Алгоритм Ковид-19

(*COVID-19 Optimizer Algorithm, CVA*)

В декабре 2019 года китайские власти уведомили мир о распространении нового вируса. Позже Всемирная организация здравоохранения объявила, что эту вспышку COVID-19 следует рассматривать как пандемию из-за ее быстрого распространения и того факта, что большинство инфицированных людей не имеют к ней иммунитета.

Первоначальный глобальный уровень инфицирования удваивался ежедневно в течение нескольких недель 2020 года. Многие страны прибегли к карантину, чтобы замедлить распространение короновируса, поскольку инфекции распространяются быстрее при больших скоплениях людей. Моделирование имеет важное значение для принятия решения о контроле за распространением вируса. Существует несколько математических моделей распространения COVID-19. В [23] представлен алгоритм оптимизации *COVID-19 Algorithm (CVA)*, который моделирует его распространение по странам мира в течение года, начиная с декабря 2019 г. Модель может использоваться для принятия решений при оптимизации карантинных мероприятий, чтобы свести к минимуму число стран, инфицированных COVID-19 и замедлить распространение эпидемии. Предложенный алгоритм *CVA* может быть применен в других областях исследований для решения оптимизационных задач. Рассмотрим алгоритм подробнее.

Предположим, что R представляет собой параметр, отражающий способность этого вируса заражать других людей в виде цепочки контагиозных инфекций (коэффициент повторного заражения). Каждый человек будет заражать группу людей в течение очень короткого периода времени, что затем приведет к вспышке. Если же вероятность заражения одного человека становится меньше, чем у другого, то распространение вируса будет снижаться.

С другой стороны, если взять простой коэффициент корреляции между значением R и скоростью передачи инфекции вирусом, то чем выше значение R, тем больше число инфицированных людей (в первоначальных оценках повторного заражения в среднем значение R от 2 до 3). Сезонный грипп может передаваться от одного

человека к другому в среднем за 1–4 часа, в то время как COVID-19 передается в среднем за 2–3 часа.

Насколько вероятно появление большого числа инфицированных людей? Это зависит от многих факторов: количества уязвимых людей в популяции, периода времени, в течение которого человек не имеет симптомов заболевания вирусом, вероятности социальных контактов каждого человека с другими людьми и возможностей передачи им вируса, а также количества людей, посещающих общественные места (торговые центры, аэропорты, железнодорожные вокзалы и т. д.). Принимая во внимание эти факторы, органы власти могут оптимально спланировать метод борьбы с распространением эпидемии, который в значительной степени связан с показателем *R*. Соответственно, можно лучше понять экосистему здравоохранения и спасти больше жизней.

Была исследована роль изоляции заболевших в ограничении распространения пандемии, а также отслеживались их контакты. Оказалось, что вспышку заболевания можно контролировать, если передача вируса прекращалась в течение первых 12 недель или при не более 5000 случаев заболевания.

С другой стороны, необходима эффективная система профилактики инфекций и контроля за ними, поскольку некоторые из введенных мер не имели научной основы и оказались неэффективными (вспышка быстро распространилась по другим странам).

Вероятность передачи вируса невелика при коротком личном контакте или начале заражения примерно через 14 дней после контакта с человеком, у которого положительный результат теста.

Коронавирус в природе начал быстро распространяться из-за его высокой способности к передаче и величине *R*-значения.

Первоначальное решение случайным образом устанавливается в области допустимого пространства, и начальная популяция генерируется на основе вспышки в этой области. Интенсивность экспорта вируса в другие регионы может отличаться для каждого региона в зависимости от поведения его населения и социальной активности. Для упрощения эта интенсивность рассматривается как постоянная величина p для всех стран ($0 < p < 1$). Пусть n — это количество решений в исходной популяции, поэтому $(1 - p)$ для n генерируются близко к исходному решению на основе следующего уравнения:

$$\|x_i - x_0\| < r, i = 1,2, \dots, (1 - p) \cdot n, \tag{4.12}$$

где x_0 − исходное решение, r − положительное постоянное число, которое ограничивает решения в определенной области. Когда сгенерированное число превысит свою верхнюю границу, происходит вспышка заболевания. *CVA* запускает вспышку, чтобы усилить поиск, интенсивно используя эту часть населения в поисках оптимального решения. Параметр p генерируется на основе следующего уравнения:

$$x_i = x_0 + \lambda \cdot (rand), i = (1-p) \cdot n + 1, (1-p) \cdot n + 2, \dots, n. \quad (4.13)$$

Здесь *rand* − действительное случайное число в диапазоне (−1, 1), λ − достаточно большое положительное число. Иными словами, существует меньшая вероятность того, что все люди подвергнутся воздействию вируса в первый момент. Таким образом, есть возможность контролировать случаи заражения и помещать их в карантин до того, как они могут привести к вспышке заболеваний. Фактически, первоначальная популяция *CVA* представляет собой первое поколение людей, подвергшихся воздействию вируса.

В процессе распространения коронавируса есть две категории людей: активные пациенты, у которых есть вирус и которые могут передавать его другим, и пациенты, которые либо умерли, либо выздоровели после COVID-19. Соответственно в алгоритме *CVA* после генерации исходной популяции имеются два вида решений: закрытые случаи, которые не могут передавать инфекцию в другие области допустимого пространства и которые удаляются из популяции, и активные случаи, когда вирус передается немедленно или через некоторое время (две итерации).

Количество активных решений равно

$$n_{act} = act \cdot n, \quad (4.14)$$

где n − количество решений в текущей популяции, act − доля активных решений в текущей популяции (обычно $act = 0{,}3$). В ходе моделирования все решения сортируются на основе их целевых функций, затем $n_{clc} = (n - n_{act})$ неактивных решений удаляются.

В алгоритме *CVA* наилучшее решение находится среди n_{act}. Для создания следующей популяции каждое активное решение может передавать вирус новым возможным решениям. Следовательно, число решений в следующей популяции будет равно

$$n_{next} = n_{act} + t \cdot n_{act} = (t+1) \cdot n_{act}, \quad (4.15)$$

где t – скорость передачи вируса.

Основные шаги алгоритма *CVA* включают следующие действия.

1. Исходная популяция решений создается случайным образом. Она подразумевает первоначальный регион или страну, где начинается распространение вируса. Параметрами алгоритма являются: количество решений n; произвольное малое положительное число r; количество итераций k (k: = 1 на первой итерации); интенсивность экспорта вируса p; процент выздоравливающих *rec*; доля активных решений в текущей популяции *act*; скорость передачи вируса t; положительное малое число ε, максимальное число итераций M_1. Параметры *CVA* устанавливаются эмпирическим путем.

2. Исходная совокупность с n решениями генерируется следующим образом: $(1 - p)$ процент решений генерируется очень близко в радиусе r, а доля решений p − вдали в допустимом пространстве.

3. Некоторые пациенты-решения выздоравливают или умирают, поэтому эти решения удаляются из популяции.

4. COVID-19, с одной стороны, распространяется в определенном регионе, а с другой стороны, через некоторое время передается в другие регионы и страны. Решения из исходной области передаются быстро, а остальные решения − после двух итераций.

5. В каждом поколении находится и сохраняется лучшее решение. Если $k > 1$, то переход к следующему шагу, в противном случае $k = k + 1$ и переход к шагу 3.

6. Если

$$d\left(F(x^{k-1}), F(x^k)\right) = \left(\sum_{i=1}^{n}\left(F\left(x_i^{k-1}\right) - F(x_i^k)\right)^2\right)^{\frac{1}{2}} < \varepsilon \vee k > M_1,$$

то алгоритм завершает свою работу. Решения (x^{k-1}) и (x^k) являются лучшими решениями в двух последовательных поколениях. В противном случае переход к шагу 2.

В [23] для демонстрации возможностей алгоритма *CVA* представлены результаты экспериментов на множестве задачи оптимизации известных мультиэкстремальных функций. Результаты по алгоритму *CVA* сравнивались с некоторыми другими метаэвристическими алгоритмами, такими как алгоритм извержения вулкана (*Volcano Eruption Algorithm, VEA*), алгоритм оптимизации серого волка (*Grey Wolf Optimizer, GWO*), *PSO* и *GA* и подтвердили его конкурентоспособность.

Рассмотрим возможности построения математической модели для описания процесса передачи инфекции с целью выработки мер по предотвращению распространения COVID-19 между странами.

Определим проблему распространения COVID-19 следующим образом. Во время распространения коронавируса существует N регионов (стран), которые могут находиться в следующих состояниях в k-й период времени:

1) страна в безопасном состоянии;

2) страна восприимчива к инфекции COVID-19;

3) страна инфицирована COVID-19, вирус может передаваться в другие страны;

4) страна инфицирована COVID-19, но приняты карантинные меры, чтобы вирус не передавался в другие страны.

Страна в состоянии 4) не нуждаются в помощи со стороны соседних регионов из-за введенного карантина. Страны в состоянии 2) и 3) не находятся в карантине. Стране в состоянии 2) рекомендуется поиск и карантин для людей, недавно прибывших из других стран, находящихся в состоянии 3). Стране в состоянии 3) рекомендуется вводить карантинные меры и тестировать людей, которые планируют ее покинуть.

Цель состоит в том, чтобы максимально увеличить число стран, находящихся в безопасном состоянии 1):

$$\max_{x_1} \sum \; x_{1_{t_k}}, \tag{4.16}$$

при условиях

$$x_{1_{t_k}} = x_{1_{t_{k-1}}} + \sum_{k=1}^{N} c_{i2_{t_{k-1}}} * h_{i_{t_{k-1}}} - \sum_{k=1}^{N} c_{i3_{t_{k-1}}} * S_{i_{t_{k-1}}}, \tag{4.17}$$

$$x_{2_{t_k}} = \sum_{k=1}^{N} c_{i3_{t_{k-1}}} * S_{i_{t_{k-1}}}, \tag{4.18}$$

$$x_{3_{t_k}} = x_{3_{t_{k-1}}} + \sum_{k=1}^{N} c_{i2_{t_{k-1}}} * (1 - h_{i_{t_{k-1}}}) - \sum_{k=1}^{N} c_{i3_{t_{k-1}}} * h_{i_{t_{k-1}}}, \tag{4.19}$$

$$x_{4_{t_k}} = x_{4t_{k-1}} \; + \sum_{k=1}^{N} c_{i3_{t_{k-1}}} * h_{i_{t_{k-1}}}, \tag{4.20}$$

$$x_{1_{t_1}} = N - 1, x_{2_{t_1}} = 0, x_{3_{t_1}} = 0, x_{4_{t_1}} = 0, \tag{4.21}$$

$$\sum_{j=1}^{4} x_{j_{t_k}} = N, k = 1,2, \dots m, \tag{4.22}$$

где m – периоды времени распространения инфекции, N – количество стран, $0 \le x_{j_{t_k}} \le N$, $c_{ij_{t_k}} \in \{0, 1\}$, i = 1, 2,..., N, j = 1, 2, 3, 4, k = 1, 2, ... , m. Переменная $x_{j_{t_k}}$ обозначает количество стран, находящихся в состоянии j в установленный период времени t_k, переменная $c_{ij_{t_k}}$ равна 1, если i-я страна находится в состоянии j период времени t_k, и равна 0 в противном случае. Переменная $h_{i_{t_k}}$ равна 1, если i-я страна в период времени t_k получает помощь, в противном

случае ее значение равно 0. $S_{i_{t_k}}$ обозначает это скорость распространения инфекции у i-й страны в период времени t_k. Значения $S_{1_{t_k}} = S_{2_{t_k}} = S_{4_{t_k}} = 0$ потому что только страны в состоянии 3) могут распространять COVID-19 в другие страны для всех k, тогда как значение $S_{3_{t_k}}$ является положительным целым числом.

Условия-ограничения в (4.17) – (4.22) справедливы для $k = 2, 3, \ldots m$ стран в состоянии 1 – период времени t_k, плюс страны в состоянии 2 – инфицированные страны несут ответственность за распространение вируса в другие страны.

Число стран в состоянии 2 сокращается в случае получения помощи на стадии (k–1), и, наоборот, увеличивается, если они не получают помощи.

Условие (4.19) указывает количество стран в состоянии 3) на (k – 1)-м этапе. Условие (4.20) выводится для стран в состоянии 4). Условие (4.21) показывает состояние стран на момент начала распространения коронавируса.

Для иллюстрации распространения вируса моделировались два сценария. Пусть количество стран $N = 50$, количество инфицированных агентов $n = 50$, количество периодов времени (итераций) $k = 5$, интенсивность экспорта вируса для каждой страны составляет 20 %.

В первом сценарии страны, отнесенные к состоянию 3), могут получить помощь и перейти в состояние 4). Фактически, страны, отнесенные к категории 3), могут экспортировать вирус в другие страны; следовательно, эффективная помощь им будет способствовать безопасности других стран. Например, путем тестирования пассажиров, выезжающих из страны, чтобы они не занесли вирус в страну назначения. Благодаря такого рода помощи страна будет переведена в режим карантина в состояние 4). Однако этот сценарий не позволяет контролировать распространение вируса. Потому что еще до того, как официальные лица страны узнают о начале ее заражения, инфекции, возможно, распространится в другие регионы.

Второй сценарий предполагает, что все страны в состоянии 2) распознаются заранее, их состояние изменяется на 3), а страны в состоянии 3) изменяются на состояние 4). Страны в состоянии 2) должны получить помощь и перейти в состояние 1).

Предположим, что 50 % стран, находящихся в состоянии 2), получат помощь до того, как они перейдут в состояние 3). Предлагается алгоритм, моделирующий этот сценарий.

Первый сценарий представляет собой стратегию, в рамках которой государственные органы по всему миру реализовали меры по защите своего населения. Благодаря этой стратегии COVID-19 получил широкое распространение и был объявлен пандемией в мире.

Более эффективным представляется использование второго сценария, который эффективно позволяет контролировать вирус, нежели традиционный сценарий.

Анализ сложности любого алгоритма позволит количественно оценить, насколько быстро он может работать, зависит от размера входных данных. Время выполнения алгоритма зависит от нескольких факторов: размер входных данных, а также используемое аппаратное и программное обеспечение. Здесь нас интересует скорость роста времени в соответствии с размер входных данных и используемый алгоритм. В частности, сложность предложенного алгоритма оценивается как:

$$O(\max((1-p), n, M, n3, n2, n*p))) = (1-p)\cdot n\cdot M[5+2\cdot n3+2\cdot times2\cdot n2+2\cdot(n\cdot p+1)]+\frac{17}{(1-p)\cdot n\cdot M}. \quad (4.23)$$

Что касается уравнения (4.23), то максимальная переменная равна n, что означает количество стран. В худшем случае сложность предлагаемого алгоритма равна $O(n^3)$.

4.4. Резюме

Алгоритм бабочки-монарха. Эта метаэвристика моделирует миграционное поведение бабочек-монархов в природе. Для решения различных оптимизационных задач используются следующие правила поведения бабочек-монархов: (1) вся их популяция находится в двух областях их обитания (субпопуляции), (2) особи размножаются и мигрируют из одной области в другую, (3) общая численность популяции поддерживается неизменной, (4) особи с наилучшей функцией пригодности автоматически переходят в следующее поколение. Процесс миграции определяется согласно (4.1) – (4.3). Кроме того, используется специальный корректирующий оператор согласно (4.4) – (4.8). Производительность алгоритма сравнивалась с конкурирующими алгоритмами на множестве многомерных

функций. Результаты показывают, что алгоритм MBO находит лучшие значения функций в большинстве задач в сравнении с конкурирующими алгоритмами. Однако алгоритм уступает конкурирующим алгоритмам по показателю средней производительности и стандартному отклонению для задач сегментация изображений, планирования и транспортной логистики.

Алгоритм червей. Основан на поведении червя нематоды, имеющего всего 302 нейрона, что, тем не менее, позволяет червям выполнять несколько сложных действий, включая выживание, смену индивидуального и коллективного стилей поиска пищи, избежание токсинов и другие. В *NP*-сложной задаче о коммивояжере алгоритм *WO* превзошел алгоритмы муравьиной колонии *ACO*, роя частиц *PSO* и генетического алгоритма *GA*. Задача оптимизации с помощью *WO* представляется в виде графа, в котором червь перемещается от одной вершины к другой в процессе поиска оптимального решения. *WO* начинается с отложения феромона. Первоначально определенное количество феромона откладывается в каждой дуге графа. Черви перемещаются от узла к другому в соответствии с вероятностью, которая частично аналогична вероятности при оптимизации муравьиным алгоритмом. Вероятность перемещения червя из одной вершины в другую вершину вычисляется согласно (4.10), а величина феромона обновляется в соответствии с (4.11). Основными операторами алгоритма являются оператор, моделирующий паттерны добывания пищи, оператор, позволяющий избежать токсинов, и оператор, моделирующий передвижение червя.

Известными примерами применения алгоритма *WO* являются задачи минимизации времени параллельной обработки изделий на нескольких машинах, а также оптимального размещения изделий на складе.

Алгоритм Ковид-19. Алгоритм *CVA* моделирует распространение Ковид-19 по странам мира. Модель может использоваться для принятия решений при оптимизации карантинных мероприятий, чтобы свести к минимуму число инфицированных стран и замедлить распространение эпидемии. Коронавирус сначала быстро распространяться из-за его высокой способности к передаче и коэффициенту повторного заражения. Интенсивность экспорта вируса в другие регионы может отличаться для каждого региона в зависи-

мости от поведения его населения и социальной активности. Первоначальная популяция *CVA* представляет собой первое поколение людей, подвергшихся воздействию вируса. В процессе распространения коронавируса есть две категории людей: активные пациенты, у которых есть вирус и которые могут передавать его другим, и пациенты, которые либо умерли, либо выздоровели. Соответственно в алгоритме *CVA* после генерации исходной популяции имеются два вида решений согласно (4.14). Для создания следующей популяции каждое активное решение может передавать вирус новым возможным решениям (4.15). В каждом поколении находится и сохраняется лучшее решение.

Алгоритм *CVA* тестировался на множестве задач оптимизации известных мультиэкстремальных функций. Результаты сравнивались с такими метаэвристиками как алгоритм извержения вулкана (VEA), алгоритм GWO, PSO и GA и подтвердили его конкурентоспособность *CVA*.

5. МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ, ИСПОЛЬЗУЮЩИЕ АГЕНТОВ ИЗ НЕКОТОРОЙ ОКРЕСТНОСТИ

5.1. Бактериальный алгоритм

Бактерии, такие как кишечная палочка или сальмонелла, являются одними из самых успешных организмов на планете. Эти подвижные бактерии имеют придатки, называемые жгутиками, с помощью которых они продвигают себя при помощи вращательных движений. Жгутики помогают бактерии кувыркаться или плавать, что является двумя основными операциями, выполняемыми бактерией во время поиска пищи. Когда они вращают жгутики по часовой стрелке, каждый жгутик толкает клетку. При вращении жгутиков в разные стороны бактерия разворачивается и совершает кувырок. Бактерия кувыркается с меньшим числом кувырков в благоприятной среде. В неблагоприятной среде она часто кувыркается, чтобы найти градиент питательных веществ. Движение жгутиков против часовой стрелки помогает бактериям плавать с очень высокой скоростью.

Такое поведение бактерий обусловлено механизмом, который называется бактериальным хемотаксисом и представляет собой двигательную реакцию этих микроорганизмов на химический раздражитель среды. В силу малого размера бактерии она не способна уловить разницу концентраций полезных и вредных веществ. Градиенты этих веществ бактерии определяют путем измерения изменений их концентраций при движении. Скорость этого движения может достигать нескольких десятков длин бактерии в секунду. Так, бактерия кишечной палочки обычно перемещается со скоростью 10÷20 своих длин в секунду. В целом бактериальный хемотаксис можно определить как сложное сочетание плавания и кувырков, позволяющее бактерии удерживаться в местах высокой концентрации питательных веществ и избегать неприемлемой концентрации вредных веществ.

В контексте задачи поисковой оптимизации, бактериальный хемотаксис можно также интерпретировать как механизм оптимизации использования бактерией известных пищевых ресурсов и поиска новых, потенциально более ценных областей. Популяция бактерий достаточной численности может формировать сложные про-

странственно-временные структуры – эффект структурообразования в популяциях бактерий. Этот эффект может быть вызван как хемотаксисом, так и некоторыми другими причинами. Как правило, бактерии перемещаются на большее расстояние в дружественной среде. Когда они получают достаточно пищи, они увеличиваются в длину и при наличии подходящей температуры ломаются посередине, превращаясь в точную копию себя.

Каноническая версия алгоритма бактериальной оптимизации (*Bacterial Foraging Optimization, BFO*), представленная в [24] включает в себя следующие основные этапы: (1) инициализация популяции бактерий, (2) хемотаксис, (3) роение, (4) воспроизведение, (5) ликвидация и устранение.

Инициализация популяции бактерий предполагает, что они могут выжить в среде, если изначально поместить их вместе в ее центр. В этом случае бактерии могут образовывать сложные устойчивые пространственно-временные паттерны в питательных веществах и смогут секретировать межклеточные сигналы, так что они будут группироваться и защищать друг друга.

Хемотаксис предполагает возможность определить характеристики движения бактерий в поисках пищи путем их совместного плавания и кувыркания. Бактерия «плавает», если она движется в нужном направлении, и «кувыркается», если движется в сторону ухудшения среды.

Роение предполагает возможность бактерии добраться до наиболее богатого пищей места, до которого ближе всего некоторая бактерия, имеющая оптимальное расположение, привлекающая другие бактерии, чтобы вместе они быстрее сходились в желаемом месте. Для этого к исходной фитнесс-функции добавляется штрафная функция, основанная на относительном расстоянии каждой бактерии от наиболее приспособленной бактерии. Когда все бактерии достигают точки оптимума, штрафная функция становится равной нулю. Эффект роения заключается в том, что бактерии собираются в группы и двигаются концентрическими паттернами с высокой плотностью бактерий.

Исходная популяция бактерий, пройдя несколько хемотаксических стадий, достигает стадии размножения и лучшее множество бактерий делится на две группы. Группа бактерий с меньшей способностью к поиску пищи заменяется более приспособленными

бактериями. Размер популяции бактерий является постоянной величиной в процессе эволюции.

В процессе эволюции может произойти внезапное непредвиденное событие, которое может резко изменить плавный процесс эволюции и вызвать элиминацию множества бактерий и/или рассеять их в новую среду. Существует вероятность того, что это неизвестное событие может поместить популяцию бактерий ближе к месту нахождения пищи. Иными словами, элиминация и рассредоточение являются частями поведения популяции на больших расстояниях. Применительно к оптимизации это помогает снизить вероятность преждевременной сходимости.

Формализация алгоритма представлена в [24] и многочисленных модификациях *BFO*. Сравнивая известные результаты тестирования *BFO* многомерных функций при решении задачи глобальной оптимизации с конкурирующими алгоритмами, например такими как *ACO, FA, BA, ABC, PSO, GWO*, можно сделать вывод о том, что результаты находятся в середине рейтинговой таблицы. Для алгоритма *BFO* настройка параметров имеет особое значение, поскольку параметров достаточно много. Настройка может дать улучшенную результативность, что является поводом для дополнительных экспериментов. *BFO* обладает рядом преимуществ, включая низкую чувствительность к начальным значениям координат при инициализации, неплохую надежность, простоту логики и реализации, возможность распараллеливания. Это делает его применимым для решения широкого круга задач оптимизации. Отметим медленную скорость сходимости *BFO*, невозможность в некоторых случаях выйти за пределы локальных оптимумов. Однако *BFO*, будучи метаэвристикой, может использоваться для разработки модификаций алгоритма.

5.2. Алгоритм межнейронного взаимодействия

Межнейронное взаимодействие осуществляется через дендриты и синапсы. Характеристики нейрона оказывают влияние на некоторое ограниченное число его соседей. Это сокращает время передачи данных и помогает повысить производительность системы. Благодаря межнейронному взаимодействию конечный результат зависит от характеристик всех нейронов.

В популяционных метаэвристиках для решения оптимизационных задач предполагается, что популяция включает множество независимых агентов с заданными входными данными и некоторой функцией

приспособленности. Если обеспечить эффективные взаимосвязи между агентами, то сходимость алгоритма к глобальному оптимуму может быть улучшена. В алгоритме *NC* роль агента выполняет нейрон, взаимодействующий с соседними нейронами. Как уже упоминалось, нейрон получает входные данные от соседей. После анализа входных данных нейроном результат может быть отправлен соседним нейронам. В отличие от биологических нейронов, нейроны в алгоритме *NC* не имеют фиксированного местоположения и могут перемещаться в процессе поиска лучшего местоположения. Критерием здесь является улучшение значения функции приспособленности.

Подобно биологическим нейронам, к каждому нейрону подключено некоторое множество соседей. Каждый нейрон может влиять на путь других нейронов, хотя они могут и не быть связаны напрямую. Рассмотрим основные компоненты алгоритма и численные примеры, чтобы показать эффективность и быстродействие алгоритма.

На первом этапе алгоритма межнейронного взаимодействия (*Neuronal Communication, NC*), представленном в [24], предполагается, что нейроны случайным образом распределены в пространстве нейросети и каждый из них связан с некоторым множеством соседних нейронов. Определяется значение функции приспособленности каждого нейрона. На очередной итерации алгоритма *NC* находится новое подходящее местоположение нейронов. Причем каждый нейрон информирует соседей о своем местоположении и значении своей функции приспособленности. Эта информация является руководством к движению нейронов: если значение функции приспособленности *j*-го соседа меньше, чем у соседнего *i*-го нейрона, то *i*-й нейрон будет стремиться приблизиться к *j*-му нейрону, и наоборот, если значение функции приспособленности *j*-го соседа больше, чем у *i*-го нейрона, то *i*-й нейрон будет стремиться отдалиться от *j*-го нейрона.

Алгоритм направляет нейрон в лучшее местоположение с учетом значения функций приспособленности соседей. Нормализованный результирующий вектор указывает траекторию движения *i*-го нейрона. Вследствие перемещения нейрона его соседи могут меняться.

Алгоритм *NC* предполагает на этапе инициализации, что соседи *i*-го нейрона выбираются случайным образом. Затем новое местоположение *i*-го нейрона вычисляется с использованием значения функции приспособленности его соседей.

Процедура работы алгоритма *NC* включает следующие шаги.

Шаг 1. Ввод параметров алгоритма и генерация случайной популяции. Параметры алгоритма включают в себя следующее: границы пространства поиска ($a \le X \le b$), максимальное число итераций (N_i), начальное и конечное число популяций (N_{ns}, N_{ne}), множество соседей на входах и выходах нейрона (N_{bs}, N_{be}), число соседей нейрона (m), количество случайно сгенерированных нейронов на каждой итерации (q) и постоянные коэффициенты (a_1, a_2, a_3, a_4).

Величины N_{ns}, N_{ne}, N_{bs}, N_{be} могут изменяться на каждом шаге алгоритма. На начальных итерациях *NC* число популяций является максимальным, а затем начинает уменьшаться, что приводит к снижению вычислительных затрат. Предлагаемые значения параметров:

$$0.1N_{ns} \le N_{ne} \le 0.8N_{ns}, \quad (5.1)$$

$$5 \le N_{bs} \le 0.3N_{ns}, \quad 3 \le N_{be} \le 0.1N_{ns}\,, \quad (5.2)$$

$$3 \le m \le 30, \quad 1 \le q \le 20\,. \quad (5.3)$$

Для первого поколения справедливо

$$\overrightarrow{X^{(1)}} = \vec{a} + (\vec{b} - \vec{a}) \cdot rand \quad , \quad (5.4)$$

где $\vec{a}$ и $\vec{b}$ – векторы, содержащие границы переменных, *rand* – генератор однородных случайных чисел, который генерирует действительные числа в диапазоне [0, 1]. $\overrightarrow{X^{(1)}}$ – вектор местоположения нейрона на первой итерации алгоритма. Верхний индекс показывает номер итерации.

Функции приспособленности *F* агентов популяции после завершения очередной итерации алгоритма сохраняются.

Шаг 2. На очередной итерации алгоритма используется множество из m нейронов, чтобы переместить часть из них (i_{ma}), имеющих наихудшие показатели в лучшие позиции. Их новые местоположения определяются с использованием среднего значения и стандартного отклонения:

$$\left.\begin{matrix}\mu = Mean[\overrightarrow{X^{(n)}}(I_m)] \\ \sigma = SD[\overrightarrow{x^{(n)}}(I_m)]\end{matrix}\right\} \rightarrow \overrightarrow{X^{(n)}}(i_{ma}) = a_1 \cdot (2 \cdot rand - 1) \cdot \sigma + \mu, \quad (5.5)$$

где $\overrightarrow{X^{(n)}}(I_m)$ показывает местоположение группы из m нейронов, *Mean* и *SD* – математическое ожидание и среднеквадратичное отклонение соответственно, i_{ma} – множество нейронов с наихудшими результатами. Рекомендуется выбирать $i_{ma} < 0{,}05N_{ns}$. Коэффициент a_1 определяет диапазон распределения новых нейронов ($a_1 \in [1, 5]$).

Шаг 3: Выбор соседей i-го нейрона осуществляется с помощью случайной селекции с кумулятивной взвешенной функцией (G) из

всей популяции нейронов (R), за исключением нейрона с наилучшим результатом (i_m), нейронов, (i_{ma}) и i-го нейрона.

Вначале находится вес j-го нейрона в диапазоне [0, 1] согласно

$$w_j = \frac{F_{max}-F_j}{F_{max}-F_{min}},\ \alpha_j = \frac{w_j}{\sum_{t=1}^{N_n} w_t},\ j,t \in R - \{i, i_m, i_{ma}\}, \quad (5.6)$$

где F_{max} и F_{min} – максимум и минимум функций приспособленности нейронов.

Далее, производится выбор соседей с помощью уравнений:

$$G_1 = \alpha_1, \quad (5.7)$$

$$G_j = G_{j-1} + \alpha_j, \quad j = 2,3, \ldots \in R - \{i, i_m, i_{ma}\}. \quad (5.8)$$

Для выбора каждого соседа i-го нейрона минимальное значение j-го нейрона должно быть таким, чтобы $rand \leq G_j$, где $j = 2,3, \ldots \in R - \{i, i_m, i_{ma}\}$ – равномерный генератор случайных чисел в диапазоне [0, 1]. Чтобы сформировать множество соседей i-го нейрона (I_b) процедуру следует повторить N_b раз.

Случайный выбор соседей уменьшает вероятность попадания в ловушку локального минимума.

Шаг 4: Шаг создания нового агента включает выполнение двух функций. Экспоненциальная весовая функция H повышает эффективность работы нейронов с улучшенным значением функции приспособленности для повышения сходимость алгоритма. Эта функция направляет локальный поиск алгоритма для нахождения глобального минимума. Более того, вектор $\overrightarrow{X_c}$ представляет собой устойчивое движение процедуры алгоритма. Этот вектор задается с использованием местоположения соседей на предыдущей итерации. Это помогает увеличить поисковые возможности метода. Таким образом, этот вектор играет роль глобального поиска алгоритма.

Экспоненциальная весовая функция H получается с помощью уравнения:

$$H_j = \frac{a_2^{w_j}}{\sum_{t=1}^{N_b} a_2^{w_t}}, \quad j,t \in I_b, \quad (5.9)$$

где a_2 является основанием экспоненциальной функции и находится в диапазоне [1, 5].

Притяжение или отталкивание, вызванное соседями, применяется вдоль линии между каждым соседом и i-м нейроном. Это направление может быть представлено единичным вектором $\vec{U}$ согласно:

$$\overrightarrow{U_J} = \frac{\overrightarrow{X_J^{(n)}} - \overrightarrow{X_i^{(n)}}}{\left\|\overrightarrow{X_J^{(n)}} - \overrightarrow{X_i^{(n)}}\right\|} \cdot sign(F_i - F_j),\ j \in I_b, \tag{5.10}$$

где $\overrightarrow{X_i^{(n)}}$ показывает местоположение *i*-го нейрона на *n*-й итерации, $\left\|\overrightarrow{X_i^{(n)}}\right\|$ - норма вектора местоположения *i*-го нейрона на *n*-й итерации. I_b показывает соседей *i*-го нейрона.

Вектор $\overrightarrow{X_c}$ представляет непрерывное движение в алгоритме, определяется согласно:

$$\overrightarrow{X_c} = \sum_{j=1}^{N_b} H_j \cdot \left\|\overrightarrow{X_i^{(n-1)}} - \overrightarrow{X_J^{(n-1)}}\right\| \cdot \overrightarrow{U_J},\ j \in I_b, \tag{5.11}$$

Здесь $\overrightarrow{X_i^{(n-1)}}$ - местоположение *i*-го нейрона на (*n*-1)-й итерации, N_b указывает количество соседей *i*-го нейрона.

Вектор направления $\overrightarrow{T_1}$ и длина шага T_2 определяются как

$$\overrightarrow{T_1} = C_1 \cdot \sum_{j=1}^{N_b} H_j \overrightarrow{U_J} + C_2 \cdot \frac{\overrightarrow{X_c}}{\left\|\overrightarrow{X_c}\right\|},\ j \in I_b, \tag{5.12}$$

$$T_2 = C_1 \cdot \sum_{j=1}^{N_b} H_j \cdot \left\|\overrightarrow{X_i^{(n)}} - \overrightarrow{X_J^{(n)}}\right\| + C_2 \cdot \left\|\overrightarrow{X_c}\right\|,\ j \in I_b. \tag{5.13}$$

Коэффициенты C_1 и C_2 вычисляются как

$$C_1 = a_2^{\frac{a_3 \cdot n}{N_i}}, \tag{5.14}$$

$$C_2 = a_2^{\frac{-a_4 \cdot n}{N_i}}, \tag{5.15}$$

где N_i - максимальное количество итераций, a_3 и a_4 – некоторые константы, которые, находятся в диапазоне [1, 5] и [1, 10] соответственно.

Здесь $\overrightarrow{T_1}$ представляет собой вектор направления, а T_2 – длина шага движения *i*-го нейрона. Коэффициенты C_1 и C_2 способствуют достижению баланса между скоростью сходимости алгоритма и диверсификацией пространства поиска глобального оптимума. В частности, C_1 способствует локализации поиск оптимума, C_2 – расширению пространства поиска.

Новое местоположение *i*-го нейрона вычисляется как:

$$\overrightarrow{X_i^{n+1}} = \overrightarrow{X_i^n} + rand \cdot \frac{\overrightarrow{T_1}}{\left\|\overrightarrow{T_1}\right\|} \cdot T_2, \tag{5.16}$$

где $\overrightarrow{X_i^{n+1}}$ и $\overrightarrow{X_i^n}$ – предполагаемое и текущее местоположение нейрона соответственно.

Местоположение $\overrightarrow{X_i^{n+1}}$ будет новым местоположением *i*-го нейрона только в том случае, если значение функции приспособ-

ленности в $\overrightarrow{X_i^{n+1}}$ будет не меньше, нежели в текущем местоположении $\overrightarrow{X_i^n}$. На каждой итерации процесс выбора соседей и нахождения нового местоположения каждого нейрона должен повторяться.

Шаг 5. Чтобы предотвратить попадание нейрона в ловушку локальных минимумов, используется стохастическая функция для генерации новых случайных нейронов. Эта функция заменяет нейроны с худшей функцией приспособленности (I_x) новыми случайными нейронами в соответствии с уравнением:

$$\overrightarrow{X^n}(I_x) = \vec{a} + (\vec{b} - \vec{a}) \cdot rand. \quad (5.17)$$

Здесь $\vec{a}$ и $\vec{b}$ – векторы, указывающие границы переменных в пространстве поиска.

Критериями останова алгоритма являются максимально допустимое число итераций и отсутствие улучшения значений наилучшей функции приспособленности.

Алгоритм сравнивался на бенчмарках с конкурирующими алгоритмами, в частности с *GA* и *CSA*. В качестве тестовых функций в экспериментах использовались функции Гриванка, де Йонга, Розенброка, Растригина и некоторые другие (всего 15 функций). Результаты свидетельствуют о преимуществах алгоритма *NC*.

5.3 Алгоритм оптимизации группового обучения

Алгоритм оптимизации группового обучения (*Group Teaching Optimization Algorithm, GTOA*), представленный в [28], имитирует механизм группового обучения студентов с учетом того, что у них может быть разная мотивация к обучению. *GTOA* для группового обучения делит студентов на сильно мотивированных и обычных студентов. Для разных студентов преподаватели используют разные методы обучения. Сильно мотивированные студенты проявляют способности к самообучению, подведению итогов. Обычным студентам не хватает мотивации к обучению, и они часто нуждаются в помощи других студентов или преподавателей.

Идея *GTOA* заключается в повышении уровня знаний всех студентов за счет моделирования механизма группового обучения. Алгоритм включает следующие основные этапы: группировка обучающихся, оценка их уровня обучения и распределение преподавателей. Рассмотрим их подробнее.

Чтобы лучше показать преимущества группового обучения, все учащиеся будут разделены на две группы в соответствии с уровнем

их знаний: группа мотивированных студентов с хорошей подготовкой и группа обычных студентов. В этом случае по мере преподавания предположительно разрыв в подготовке между будет только увеличиваться. Алгоритм *GTOA* предполагает возможность динамически изменять группировку после каждого цикла обучения.

В *GTOA* преподаватели используют два разных плана обучения для двух групп обучающихся.

В группе мотивированных студентов с хорошей подготовкой и навыками самостоятельной работы преподаватель больше внимания уделяет повышению общего среднего уровня знаний учащихся:

$$x_{teacher,i} = x_i + a \cdot (x_T - F \cdot (b \cdot M + c \cdot x_i)), \quad (5.18)$$

$$M = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad (5.19)$$

$$b + c = 1, \quad (5.20)$$

где N – количество студентов, x_i – i-й студент, x_T – преподаватель, M – средний уровень знаний студентов, $x_{teacher,i}$ – решение, полученное на обучения преподавателем, a, b, c – случайные числа на интервале [0, 1], F – коэффициент, который может принимать значение 1 или 2.

В группе обычных студентов они более склонны получать знания от преподавателей:

$$x_{teacher,i} = x_i + 2 \cdot d \cdot (x_T - x_i), \quad (5.21)$$

где d – случайное число между [0, 1]. Здесь нет гарантии, что они приобретут знания на этапе обучения преподавателем, поэтому назначается минимальное значение, отражающее уровень знаний учащихся после этапа обучения преподавателем

$$x_{teacher,i} = \begin{cases} x_{teacher,i}, \text{если} f(x_{teacher,i}) < f(x_i) \\ x_i, \text{если} f(x_{teacher,i}) \geq f(x_i) \end{cases} \quad (5.22)$$

Студенты могут приобретать знания посредством самостоятельного обучения или взаимодействия с другими студентами в свободное время:

$$x_{student,i} = \begin{cases} x_{teacher,i} + e \cdot (x_{teacher,i} - x_{teacher,j}) + g \cdot (x_{teacher,i} - x_i), \text{если} f(x_{teacher,i}) < f(x_{teacher,j}) \\ x_{teacher,i} - e \cdot (x_{teacher,i} - x_{teacher,j}) + g \cdot (x_{teacher,i} - x_i), \text{если} f(x_{teacher,i}) \geq f(x_{teacher,j}) \end{cases}, \quad (5.23)$$

где e и g – два случайных числа в диапазоне [0, 1], $x_{teacher,i}$ – это знания студента i, полученные на этапе обучения, $x_{teacher,j}$ – это знания студента j, полученные на этапе обучения с преподавателем. В (5.23) второе и третье слагаемые означают получение знаний от другого студента и самообучение соответственно.

Кроме того, назначается минимальное значение, отражающее уровень знаний обучающихся после этапа обучения:

$$x_i = \begin{cases} x_{teacher,i}, \text{если} f(x_{teacher,i}) < f(x_{student,i}) \\ \\ x_i, \text{если} f(x_{teacher,i}) \geq f(x_{student,i}) \end{cases}. \quad (5.24)$$

Создание эффективного механизма распределения преподавателей важно для повышения уровня знаний обучающихся. Например, в алгоритме серых волков *GWO* среднее значение получается путем выбора трех лучших волков и использования их для направления всех волков к добыче. По аналогии с охотничьим поведением в алгоритме *GWO*, распределение преподавателей в *GTOA* выражается следующей формулой:

$$x_T = \begin{cases} x_{first}, \text{если } (x_{first}) \leq f(\frac{x_{first}+x_{second}+x_{third}}{3}) \\ \frac{x_{first}+x_{second}+x_{third}}{3}, \text{ если } (x_{first}) > f(\frac{x_{first}+x_{second}+x_{third}}{3}) \end{cases}, \quad (5.25)$$

где, x_{first}, x_{second}, x_{third} – студенты с первым, вторым и третьим значениями уровней мотивированности соответственно. Чтобы ускорить сходимость алгоритма, к мотивированным и обычным студентам распределяют одного и того же преподавателя.

В целом, алгоритм *GTOA* включает следующие шаги.

Шаг 1. Инициализация параметров алгоритма: максимальное время оценки T_{max}, текущее время оценки t ($t = 0$), размер популяции N, верхняя и нижняя границы ub и lb переменных решения, размерность dim и фитнесс функцию f. Вначале инициализируется решение x. Каждая группа переменных представляет студентов:

$$x = [x_1, \dots, x_N]^{T_{max}} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,dim} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,dim} \end{bmatrix}, \quad (5.26)$$

$$x_i = lb + (ub - lb) \cdot k, \quad (5.27)$$

Шаг 2. Вычисление фитнесс функции студента, поиск оптимального решения G.

Шаг 3. Если текущий номер итерации t больше T_{max}, то остановка алгоритма.

Шаг 4. Отбираются первые три наиболее мотивированных студента. Распределение преподавателей производится по формуле (5.25).

Шаг 5. Студенты делятся на две группы в зависимости от уровня их мотивированности. Наиболее мотивированные студенты распределяются в элитную группу x_{good}, остальные – в обычную группу x_{ordinary}. У обеих групп общий учитель.

Шаг 6. Для группы x_{good} преподавание реализуется на основе уравнений (5.18)–(5.20) и (5.22), а затем согласно уравнениям (5.23) и (5.24).

Для группы x_{ordinary} преподавание реализуется на основе уравнений (5.21) и (5.22), а затем согласно (5.23) и (5.24).

Шаг 7: вычислить значение фитнесс функции студентов, выбрать текущее оптимальное решение *G*, обновить текущий номер итерации $t = t + 1$. Переход к шагу 3.

Предлагается следующая модификация алгоритма *MGTOA*.

1. Учебная мотивация.

Студенты могут получить знания не только от преподавателя, но и путем самостоятельного обучения или общения в своей группе. Мотивированные студенты с мотивацией *D* более склонны к самообучению:

$$D = \frac{1-i}{N} \cdot \sin(2\pi r), \tag{5.28}$$

$$x_{student,i} = x_{teacher,i} + D \cdot x_{teacher,i}, \tag{5.29}$$

где *r* – случайное число из интервала [0, 1].

Для обычных студентов это соотношение выглядит несколько иначе:

$$x_{student,i} = \begin{cases} x_{teacher,i} + e \cdot (x_{teacher,i} - x_{teacher,j}) + g \cdot (x_{teacher,i} - M), \\ \text{если } f(x_{teacher,i}) < f(x_{teacher,j}) \\ x_{teacher,i} - e \cdot (x_{teacher,i} - x_{teacher,j}) + g \cdot (x_{teacher,i} - M), \\ \text{если } f(x_{teacher,i}) \geq f(x_{teacher,j}) \end{cases} \tag{5.30}$$

2. Обучение на основе оппозиции.

Обучение на основе оппозиции (*OBL*) – это новая схема, которая в последние несколько лет успешно применяется в различных методах машинного обучения. Как известно, для популяционных метаэвристик рекомендуется инициировать популяцию решений случайным образом. Однако, зачастую, это приводит к поиску решений в бесперспективных областях. Основная идея обучения на основе оппозиции состоит в формировании исходной популяции из оппозиционных (симметричных, противоположных) решений, что, как показывают эксперименты, является более перспективным подходом, нежели генерация рандомизированных популяций [29]. Это расширяет область поиска, разнообразие популяции, благодаря чему алгоритм обладает более сильными исследовательскими возможностями и способностью к конвергенции. Конкретная формула выглядит следующим образом:

$$xnew_i = (ub + lb) - (x_T - t)/T_{max} \cdot rand \cdot x_i, \tag{5.31}$$

где $xnew_i$ представляет собой решение, полученное после обучения на основе оппозиции.

Определяется, является ли решение, полученное в результате обучения на основе оппозиции, лучшим, нежели исходное решение:

$$x_i = \begin{cases} x_i, \text{ если } f(x_i) \le f(xnew_i) \\ xnew_i, \text{ если } f(x_i) > f(xnew_i) \end{cases}. \quad (5.32)$$

3. Стратегия перезапуска.

Стратегия перезапуска позволяет преодолевать проблему попадания в локальный оптимум. Для этого устанавливается предел порога перезапуска согласно:

$$Limit = \ln(t). \quad (5.33)$$

Выясняется, получено ли лучшее решение на очередной итерации. Если превышен *Limit*, выполняется перезапуск. Стратегия перезапуска генерирует два новых решения T_1 и T_2 соответственно по уравнениям (18) и (19):

$$T_1 = lb + rand() \cdot (ub - lb), \quad (5.34)$$

$$T_1 = lb + rand() \cdot (ub - lb). \quad (5.35)$$

Затем выбирается лучшее решение для замены исходного решения. Формула стратегии перезапуска выглядит следующим образом:

$$T_2 = lb + rand(\ \ ) \cdot (ub - lb), \text{если } lb \ge T_2 \ge ub. \quad (5.36)$$

Временная сложность модифицированного алгоритма *MGTOA* зависит от количества студентов *N*, размерности данной задачи *dim*, количества итераций алгоритма *T* и сложности вычисления фитнесс функции *C*. Оценка временной сложности алгоритма *O(MGTOA)* включает оценку затрат на установку параметров *O(УП)*, на инициализацию популяции О(ИП), на вычисление фитнесс функции *O(ВФФ)* и на обновление популяции *O(ОП)*:

$$O(MGTOA) = O(\text{УП}) + O(\text{ИП}) + O(\text{ВФФ}) + O(\text{ОП}) \quad (5.37)$$

где временная сложность компонентов в (5.37) определяется следующим образом: (1) инициализация постановки задачи требует времени *O*(1); (2) инициализация популяции требует времени *O*(*N*×*dim*); (3) обновление популяции требует время *O*(2×*T*×*N*×*dim*); (4) время, необходимое для обучения на основе оппозиции *O*(*T*×*N*×*dim*); (5) время, необходимое для перезапуска стратегии *O*(2×*T*×*N*×*dim*/*Limit*); (6) стоимость времени фитнесс функции включает в себя стоимость времени работы алгоритма, а также стоимость времени обучения на основе оппозиции и стоимость времени расчета стратегии перезапуска.

Затраты времени на работу алгоритма составляют $O(T\times N\times C)$, стоимость времени расчета стратегии обучения на основе оппозиции $O(T\times N\times C)$, стоимость времени расчета стратегии перезапуска равна $O(T\times N\times C/Лимит)$. Тогда общая стоимость времени равна $O(2\times T\times N\times C+T\times N\times C/Limit)$.

Следовательно, временная сложность *MGTOA* выражается как:

$$O(MGTOA) = O\left(\begin{matrix}1 + N \cdot dim + T \cdot N \cdot C \cdot \left(2 + \frac{1}{Limit}\right)\\ +T \cdot N \cdot dim \cdot \left(3 + \frac{2}{Limit}\right)\end{matrix}\right). \quad (5.38)$$

Поскольку

$$1 \ll T \cdot N \cdot C, 1 \ll T \cdot N \cdot dim, N \cdot dim \ll T \cdot N \cdot C, N \cdot dim \ll T \cdot N \cdot dim,$$

то можно упростить:

$$O(MGTOA) = O\left(\begin{matrix}T \cdot N \cdot C \cdot \left(2 + \frac{1}{Limit}\right)\\ +T \cdot N \cdot dim \cdot \left(3 + \frac{2}{Limit}\right)\end{matrix}\right). \quad (5.39)$$

Псевдокод *MGTOA* имеет следующий вид.

1. Инициализация параметров алгоритма t , T_{max}, ub, lb, N, dim.
2. Инициализация популяции x согласно (5.26) и (5.27).
3. Вычисление фитнесс функции студентов и поиск оптимального решения G.
4. *while* $t < T_{max}$
5. Согласно (5.25) определяется распределение преподавателей.
6. Студенты делятся на две группы в зависимости от уровня их мотивированности. Наиболее мотивированные студенты распределяются в элитную группу x_{good}, остальные – в обычную группу x_{ordinary}. У обеих групп общий учитель. Число студентов в группе x_{good} равно N_{good}.
7. *for i = 1:N*
8. *if* $i < N_{good}$
9. Решение, полученное на этапе обучения преподавателем в группе мотивированных и обычных студентов в соответствии с (5.18)–(5.20).
10. *else*
11. Решение, полученное на этапе обучения преподавателем в группе мотивированных и обычных студентов в соответствии с (5.21), (5.22).
12. *end*
13. Назначить минимальное значение, отражающее уровень знаний обучающихся после этапа обучения согласно (5.24).
14. Определить средний уровень знаний мотивированных

студентов (M).

15. *if* $i < N_{good}$
16. *for j = 1:dim*
17. Мотивированные студенты получают знания не только от преподавателя, но и путем самостоятельного обучения или общения в своей группе согласно (5.28), (5.29) и (5.24).
18. *end*
19. *else*
20. Студенты из обычной группы получают знания согласно (5.24) и (5.30.
21. *end*
22. *end*
23. Определяется решение, полученное после обучения на основе оппозиции согласно (5.31), а позиция студента обновляется в соответствии с (5.32).
24. Вычислить предел порога перезапуска согласно (5.33).
25. *for i = 1:N*
26. сгенерировать два новых решения T_1 и T_2 согласно (5.34) и (5.35). Выбрать лучшее для замены исходного решения согласно (5.36).
27. *end*
28. *end*
29. $t = t + 1$
30. *end*

Для оценки производительности *MGTOA* проводились эксперименты с 23 тестовыми функциями, на которых он сравнивался с 9 метаэвристиками такими как *GA, SCA*, белоголового орла (*Bald Eagle Search, BES*), оптимизации Ремора (*Remora Optimization Algorithm, ROA*), арифметической оптимизации (*Arithmetic Optimization Algorithm, AOA*), *WOA* и некоторыми другими.

Тестовые функции включали 7 унимодальных, 6 мультимодальных и 10 мультимодальных функций фиксированной размерности. Все алгоритмы выполняются независимо 30 раз для получения оптимального значения, среднего значения приспособленности и стандартного отклонения тестируемой функции.

MGTOA показал более стабильные результаты при различных размерностях функций. Близкие результаты показали *ROA* и *WOA*. Остальные алгоритмы получили хорошие результаты только в некоторых функциях. *MGTOA* имеет лучшую производительность при решении многомерных задач.

По критерию суммы рангов Уилкоксона результаты *MGTOA* также продемонстрировал значимые результаты. Значения $p>0,05$ между *MGTOA* и *GTOA*, что указывает на их сильное отличие.

Представлены также результаты тестирования *MGTOA* на инженерной задаче проектирования конструкции сосуда под давлением с ограничениями (рис. 5.1).

Задача включает 4 переменные: толщина корпуса T_s, толщина головки T_h , внутренний радиус R и длина контейнера L.

Введем обозначения $\vec{X} = [x_1, x_2, x_3, x_4] = [T_s, T_h, R, L]$.

Необходимо минимизировать функцию

$$f(\vec{X}) = 0{,}6224x_1x_3x_4 + 1{,}7781x_2x_3^2 + 3{,}1661x_1^2x_4 + 19{,}84x_1^2x_3, \quad (5.40)$$

при условиях:

$$g_1(\vec{X}) = -x_1 + 0{,}0193x_3 \le 0, \quad (5.41)$$

$$g_2(\vec{X}) = -x_3 + 0{,}00954x_3 \le 0, \quad (5.42)$$

$$g_3(\vec{X}) = -\pi x_3^2 x_4 + \frac{4}{3}\pi x_3^3 + 1296000 \le 0, \quad (5.43)$$

$$g_4(\vec{X}) = x_4 - 240 \le 0, \quad (5.44)$$

$$0 \le x_1, x_2 \le 99, \ 10 \le x_3, x_4 \le 200, \quad (5.45)$$

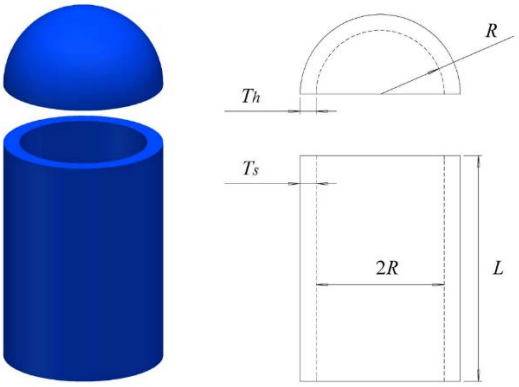

Рис. 5.1 Конструкция сосуда под давлением

Результаты решения задачи проектирования сосудов высокого давления с помощью *MGTOA*: $T_s = 0{,}754364$, $T_h = 0{,}366375$, $R = 40{,}42809$, $L = 198{,}5652$. Они оказались лучше, нежели у конкурирующих алгоритмов.

В последние годы был разработан целый спектр алгоритмов с подобного рода механизмом дифференциально-векторного движения, аналогичному *GTOA*. В частности, алгоритм следопыта (*Pathfinder algorithm, PA*, 2019), оптимизатор охоты на парусника (*Sailfish Optimizer, SO*, 2019), алгоритм оптимизации поденок (*Mayfly Optimization Algorithm, MOA*, 2020), медвежий алгоритм поиска по запаху (*Bear Smell Search Algorithm, BSSA*, 2020), алгоритм оптимизации Архимеда (*Archimedes Optimization Algorithm*, 2021), алгоритм оптимизации рыжей лисы (*Red Fox Optimization Algorithm, RFOA*, 2021), алгоритм оптимизации табуна лошадей (*Horse Herd Optimization Algorithm, HHOA*, 2021), змеиный оптимизатор (*Snake Optimizer*, 2022), алгоритм оптимизации Эболы (*Ebola Optimization Algorithm, EOA*, 2022).

5.4. Резюме

Бактериальный алгоритм. Паттерны поведения бактерий обусловлены механизмом, который называется бактериальным хемотаксисом и представляет собой двигательную реакцию этих микроорганизмов на химический раздражитель среды. Бактериальный хемотаксис представляет собой сложное сочетание плавания и кувырков, позволяющее бактерии удерживаться в местах высокой концентрации питательных веществ и избегать неприемлемой концентрации вредных веществ. В контексте задачи поисковой оптимизации бактериальный хемотаксис интерпретируется как механизм оптимизации использования бактерией пищевых ресурсов и поиска новых, потенциально более ценных областей. Популяция бактерий формирует сложные пространственно-временные структуры. Алгоритм BFO включает в себя следующие основные этапы: (1) инициализация популяции бактерий, (2) хемотаксис, (3) роение, (4) воспроизведение, (5) ликвидация и устранение. Группа бактерий с меньшей способностью к поиску пищи заменяется более приспособленными бактериями. Размер популяции бактерий является постоянной величиной в процессе эволюции. Сравнивая известные результаты тестирования *BFO* многомерных функций при решении задачи глобаль-

ной оптимизации с конкурирующими алгоритмами, например такими как *ACO, FA, BA, ABC, PSO, GWO*, можно сделать вывод о том, что результаты находятся в середине рейтинговой таблицы.

Алгоритм межнейронного взаимодействия. В алгоритме *NC* роль агента выполняет нейрон, взаимодействующий с соседними нейронами. Нейрон получает входные данные от соседей. После анализа входных данных нейроном результат может быть отправлен соседним нейронам. В отличие от биологических нейронов, нейроны в алгоритме *NC* не имеют фиксированного местоположения и могут перемещаться в процессе поиска лучшего местоположения. Критерием здесь является улучшение значения функции приспособленности. На первом этапе алгоритма *NC* нейроны случайным образом распределены в пространстве нейросети и каждый из них связан с некоторым множеством соседних нейронов. Определяется значение функции приспособленности каждого нейрона. На очередной итерации алгоритма находится новое местоположение нейронов. Алгоритм направляет нейрон в лучшее местоположение с учетом значения функций приспособленности соседей согласно (5.5) – (5.16.

Алгоритм сравнивался на бенчмарках с конкурирующими алгоритмами (*GA* и *CSA*). В качестве тестовых функций в экспериментах использовались функции Гриванка, де Йонга, Розенброка, Растригина и некоторые другие. Результаты свидетельствуют о преимуществах алгоритма *NC*.

Алгоритм оптимизации группового обучения. Алгоритм GTOA моделирует механизм группового обучения студентов с учетом того, что у них может быть разная мотивация к обучению, а преподаватели используют разные методы обучения. Идея GTOA заключается в повышении уровня знаний всех студентов за счет моделирования механизма группового обучения. Алгоритм включает следующие основные этапы: группировка обучающихся, оценка их уровня обучения и распределение преподавателей согласно (5.18) – (5.25). Алгоритм *GTOA* предполагает возможность динамически изменять группировку после каждого цикла обучения. Модификация алгоритма *MGTOA* предполагает, что студенты могут получать знания не только от преподавателя, но и путем самостоятельного обучения или общения в своей группе. В *MGTOA* также используется обучение на основе оппозиции согласно (5.31) – (5.32) и стратегия перезапуска согласно (5.33) – (5.36).

Для оценки производительности *MGTOA* проводились эксперименты на множестве тестовых функций и сравнение с алгоритмами *GA*, *SCA*, белоголового орла (*BES*), оптимизации Ремора (*ROA*), арифметической оптимизации (*AOA*), *WOA* и некоторыми другими. *MGTOA* показал более стабильные результаты при различных размерностях функций и лучшую производительность при решении многомерных задач. Результаты тестирования *MGTOA* на инженерной задаче проектирования конструкции сосуда под давлением оказались лучше, нежели у конкурирующих алгоритмов.

В последние годы был разработан целый спектр алгоритмов с подобного рода механизмом дифференциально-векторного движения, аналогичному *GTOA*: алгоритм следопыта (*Pathfinder algorithm, PA*, 2019), оптимизатор охоты на парусника (*Sailfish Optimizer, SO*, 2019), алгоритм оптимизации поденок (*Mayfly Optimization Algorithm, MOA*, 2020), медвежий алгоритм поиска пищи по запаху (*Bear Smell Search Algorithm, BSSA*, 2020), алгоритм оптимизации Архимеда (*Archimedes Optimization Algorithm*, 2021), алгоритм оптимизации рыжей лисы (*Red Fox Optimization Algorithm, RFOA*, 2021), алгоритм оптимизации табуна лошадей (*Horse Herd Optimization Algorithm, HHOA*, 2021), змеиный оптимизатор (*Snake Optimizer*, 2022), алгоритм оптимизации Эболы (*Ebola Optimization Algorithm, EOA*, 2022).

Заключение

В учебном пособии представлен анализ современного состояния исследований в области разработки метаэвристик, как одного из направлений машинного обучения, включая их классификацию, тестирование и области применения.

Около 60 % существующих метаэвристик относятся к классу дифференциально-векторного движения. Именно метаэвристики этого класса подробно рассматриваются в пособии. В определении направления дифференциально-векторного движения может участвовать вся популяция решений, только репрезентативные агенты популяции, либо только решения из некоторой окрестности или субпопуляции. Представлены эффективные алгоритмы из класса дифференциально-векторного движения, их особенности и сравнительные оценки. Также приводятся примеры практических задач оптимизации, которые иллюстрируют и поясняют метаэвристики.

Наметилась тенденция к гибридизации метаэвристик в одном оптимизаторе. Однако требуются убедительные доказательства, что результаты компенсируют увеличение сложности по сравнению с отдельными алгоритмами. Заметна также тенденция проводить сравнение производительности биоэвристик, используя статистическую проверку гипотез на бенчмарках.

Заслуживает внимания применение метаэвристик для решения следующих типов задач:

- динамической и стохастической оптимизации;
- многокритериальной оптимизации;
- мультимодальной оптимизации;
- многомерной оптимизации;
- оптимизации и адаптации настроек параметров метаэвристик для достижения баланса между скоростью сходимости и диверсификацией пространства поиска решений.

Перспективным направлением дальнейших исследований и разработок представляется использование идей меметики для практического применения к решению масштабных машинного обучения и поисковой оптимизации. Меметический автомата рассматривается как программный агент, способный к обучению, автономному поведению и взаимодействию с другими агентами через Интернет при поддержке облачной инфраструктуры. С алгоритмической точки

зрения мемы рассматриваются как строительные блоки знаний, выраженные в эвристических представлениях, которые могут быть извлечены из опыта и адаптивно переданы для повторного использования в различных задачах. Меметический подход предполагает, что обучение занимает центральное место в качестве фундаментального аспекта поиска оптимального решения. Предлагаются стратегии использования мемов знаний в процессе решения инженерных и оптимизационных задач [30, 31]. В случае последовательной передаче знаний от задачи к задаче предполагается, что при решении целевой задачи база знаний состоит из мемов задач, которые уже ранее решались. При одновременной передаче знаний сразу многим целевым задачам, в отличие от последовательной передачи мемов, где в данный момент оптимизируется одна целевая задача, сценарий многозадачности предполагает параллельное решение нескольких равноприоритетных задач. Предлагаются алгоритмы последовательной и параллельной передачи мемов знаний при решении целевой задачи. Дополняя базовый оптимизатор, модулем меметики становится возможным реализовать поиск «на лету» при поддержке таких технологий как облачные вычисления и Интернет вещей, которые предлагают крупномасштабное хранение данных и бесперебойные средства связи. Эффективность оптимизаторов экспериментально подтверждается на примере проектирования нейроэволюционного контроллера для робота-тележки с двумя флагштоками.

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

В учебном пособии используются следующие сокращения:

ИЛП – индуктивное логическое программирование.
ИНС – искусственная нейронная сеть.
ABC – Artificial Bee Colony.
ACO – Ant Colony Optimization.
AO – Aquila Optimization.
AOA – Arithmetic Optimization Algorithm.
ASO – Anarchic Society Optimization.
AVOA – African Vulture Optimization Algorithm.
BA – Bat Algorithm.
BES – Bald Eagle Search.
BFO – Bacterial Foraging Optimization.
BOA – Butterfly Optimization Algorithm.
BSSA – Bear Smell Search Algorithm.
CART – Classification and Regression Tree.
ChSA – Chameleon Swarm Algorithm.
CVA – COVID-19 Algorithm.
CSA – Cuckoo Search Algorithm.
DA – Dragonfly Algorithm.
DE – Differential Evolution.
DEA – Dolphin Echolocation Algorithm.
EOA – Ebola Optimization Algorithm.
EPO – Emperor Penguin Optimization.
ES – Evolutionary Strategy.
EWA – Earthworm Algorithm.
FA – Firefly Algorithms.
FOA – Forest Optimization Algorithm.
FPA – Flower Pollination Algorithm.
FWA – Fireworks Algorithm.
GA – Genetic Algorithm.
GBSO – Global Brainstorm Optimization Algorithm.
GOA – Grasshopper Optimization Algorithm.
GSA – Gravitational Search Algorithm.
GTOA – Group Teaching Optimization Algorith.
GWO – Grey Wolf Optimizer.
HHO – Harris Hawks Optimization.

HHOA – Horse Herd Optimization Algorithm.
HS – Harmony Search.
IA – Immune algorithm.
ICA – Imperialist Competitive Algorithm.
IWD – Intelligent Water Drops.
KH – Krill Herd.
KNN – K-Nearest Neighbors.
LOA – Lion Optimization Algorithm.
MA – Monkey Algorithm.
MBO – Monarch Butterfly Optimization.
MFO – Moth-Flame Optimization.
MHDA – Memory Based Dragonfly Algorithm.
MOA – Mayfly Optimization Algorithm.
MPA – Marine Predator Algorithm.
NC – Neuronal Communication.
NFL – No Free Lunch.
PA – Pathfinder algorithm.
PCA – Particle Collision Algorithm.
PFA – Pathfinder Algorithm.
PSO – Particle Swarm Optimization.
RFOA – Red Fox Optimization Algorithm.
ROA – Remora Optimization Algorithm.
SA – Simulated Annealing.
SCA – Sine Cosine Algorithm.
SFO – Sailfish Optimizer.
SHO – Spotted Hyena Optimizer.
SMA – Slime Mould Algorithm.
SO – Sailfish Optimizer.
SOA – Seagull Optimization Algorithm.
SSA – Salp Swarm Algorithm.
STOA – Sooty Tern Optimization Algorithm.
SVM – Support Vector Machine.
TS – Tabu Search.
TSA – Tunicate Swarm Algorithm.
UCI – University of California, Irvine.
VEA – Volcano Eruption Algorithm.
WO – Worm Optimization.
WOA – Whale Optimization Algorithm.

Список литературы

1. Fausto F. From Ants to Whales: Metaheuristics for All Tastes / F. Fausto // Artif. Intell. Rev. 2019. Vol. 53 (1). Pp. 753–810. DOI 10.1007/s10462-018-09676-2

2. Родзин С.И. Современное состояние биоэвристик: классификация, бенчмаркинг, области применения / С.И. Родзин // Известия ЮФУ. Технические науки. 2023. №2. С. 280–298 [Электронный ресурс]. – Режим доступа: https://izv-tn.tti.sfedu.ru/index.php/izv_tn/article/view/793/981

3. Курейчик В.В. Вычислительные модели эволюционных и роевых биоэвристик: обзор / В.В. Курейчик, С.И. Родзин // Информационные технологии. 2021. Т. 27. №10. С. 563–574 [Электронный ресурс]. – Режим доступа: http://novtex.ru/IT/it2021/number_10_annot.html#2

4. Курейчик В.В. Вычислительные модели биоэвристик, основанных на физических и когнитивных процессах: обзор / В.В. Курейчик, С.И. Родзин // Информационные технологии. 2021. Т. 27. № 11. С. 563–574 [Электронный ресурс]. – Режим доступа: http://novtex.ru/IT/eng/doi/it_27_563-574.html

5. Dragoi E. Review of Metaheuristics Inspired from the Animal Kingdom / E. Dragoi, V. Dafinescu // Mathematics. 2021. Vol. 9 (19). P. 2335. DOI 10.3390/math9182335

6. Molina D. Comprehensive Taxonomies of Nature- and Bio-Inspired Optimization: Inspiration versus Algorithmic Behavior, Critical Analysis, and Recommendations / D. Molina [Electronic resource]. – Access mode: https://arxiv.org/abs/2002.08136v3

7. Родзин С.И. Биоэвристики: теория, алгоритмы и приложения / С.И. Родзин, Ю.А. Скобцов, С.А. Эль-Хатиб. Чебоксары: Среда, 2019. 224 с. [Электронный ресурс]. – Режим доступа: https://phsreda.com/e-articles/54/Action54-22141.pdf

8. Курейчик В.В. Теория эволюционных вычислений / В.В. Курейчик, В.М. Курейчик, С.И. Родзин. М.: Физматлит, 2012. 260 с. [Электронный ресурс]. – Режим доступа: http://www.rfbr.ru/rffi/ru/books/o_1780977

9. Wolpert D. The no free lunch theorems for optimization / D. Wolpert, W. Macready // IEEE Trans. Evol. Comp. 1997. No. 1. Pp. 67–82. DOI 10.1109/4235.585893

10. Петровский А.Б. Групповой вербальный анализ решений / А.Б. Петровский. М.: Наука, 2019. 287 с.

11. Storn R. Differential evolution – a efficient heuristic for global optimization over continuous spaces / R. Storn, K. Price // Glob. Optim. 1997. No. 11 (4). Pp. 341–359 [Electronic resource]. – Access mode: https://www.researchgate.net/publication/227242104_Differential_Evolution_-_A_Simple_and_Efficient_Heuristic_for_Global_Optimization_over_Continuous_Spaces

12. Mirjalili S. SCA: a sine cosine algorithm for solving optimization problems / S. Mirjalili // Knowl. based syst. 2016. Vol. 96. Pp. 120–133 [Electronic resource]. – Access mode: http://zeus.inf.ucv.cl/~bcrawford/DiplomadoIA_2022/SCA_paper.pdf

13. Родзин С.И. Коэволюционный самонастраивающийся алгоритм оптимизации / С.И. Родзин // Вестник ВГУ. Системный анализ и информационные технологии. 2023. №1. С. 16–27 [Электронный ресурс]. – Режим доступа: https://journals.vsu.ru/sait/article/view/11112.

14. Родзин С.И. Биоинспирированная гиперэвристика для отбора значимых признаков в задачах классификации больших данных / С.И. Родзин // Вестник компьютерных и информационных технологий. 2021. Т. 18. №5. С. 35–44 [Электронный ресурс]. – Режим доступа: http://www.vkit.ru/index.php/archive-rus/1068-035-044.

15. Atashpaz-Gargari E. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition / E. Atashpaz-Gargari, C. Lucas // Proc. IEEE Congress on Evolutionary Computation (CEC07). 2007. Pp. 4661–4667 [Electronic resource]. – Access mode: https://www.academia.edu/699373/Imperialist_competitive_algorithm_an_algorithm_for_optimization_inspired_by_imperialistic_competition

16. Курейчик В.В. Модель коллаборативного поведения роя саранчи для оптимизации многомерных многоэкстремальных функций / В.В. Курейчик, С.И. Родзин // Известия вузов. Северо-Кавказский регион. Технические науки. 2023. №1. С. 10–16. DOI 10.17213/1560-3644-2023-1-10-16.

17. Cuevas E. A swarm optimization algorithm inspired in the behavior of the social spider / E. Cuevas // Jour. Expert Systems with Applications. 2013. No. 16. Pp. 6374–6384 [Electronic resource]. – Access mode: https://www.researchgate.net/publication/257405007_A_swarm_optimization_algorithm_inspired_in_the_behavior_of_the_social-spider

18. Rodzin S. Spider Colony Optimization Algorithm: A Bio-Heuristic for Global Optimization Problem / S. Rodzin, L. Rodzina // Lecture Notes in Networks and Systems book series. 2023. Vol. 722. Pp. 661-669. DOI 10.1007/978-3-031-35311-6_63.

19. Rashedi E. GSA: A Gravitational Search Algorithm / E. Rashedi, H. Nezamabadi-pour, S. Saryazdi // Information Sciences. 2009. Vol. 179. No. 13. Pp. 2232–2248 [Electronic resource]. – Access mode: https://www.academia.edu/830451/GSA_a_gravitational_search_algorithm

20. Карпенко А.П. Популяционные алгоритмы глобальной поисковой оптимизации. Обзор новых и малоизвестных алгоритмов / А.П. Карпенко // Информационные технологии. 2012. № 7. 32 с [Электронный ресурс]. – Режим доступа: http://novtex.ru/IT/it2012/number07_pril.html

21. Wang G.-G. Monarch butterfly optimization / G.-G. Wang, S. Deb, Z. Cui // Neural Comput. & Applic. 2019. Vol. 31. Pp. 1995–2014. DOI 10.1007/s00521-015-1923-y

22. Arnaout J.-P. Worm Optimization for the Traveling Salesman Problem / J.-P. Arnaout // Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling. 2016. DOI 10.1007/978-3-319-26024-2_11

23. Hosseini E. COVID-19 Optimizer Algorithm, Modeling and Controlling of Coronavirus Distribution Process / E. Hosseini // IEEE Journal of Biomedical and Health Informatics. 2020. Vol. 24. No. 10. Pp. 2765-2769. DOI 10.1109/JBHI.2020.3012487

24. Passino K.M. Biomimicry of bacterial foraging for distributed optimization and control / K.M. Passino // IEEE Control Systems Magazine. 2002. Vol. 22. No. 3. Pp. 52–67 [Electronic resource]. – Access mode: http://biomimetic.pbworks.com/f/Biomimicry+of+Bacterial+ForgaingPassino.pdf

25. Gharebaghi S.A. New meta-heuristic optimization algorithm using neuronal communication / S.A. Gharebaghi, M.A. Asl // Int. J. Optim. Civil Eng. 2017. Vol. 7 (3). Pp. 413–431 [Electronic resource]. – Access mode: https://www.dbpia.co.kr/Journal/articleDetail?nodeId=NODE10763568

26. Shan H. Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior / H. Shan, Q.H. Wu, J.R. Saunders // IEEE Trans. on Evolutionary Computation. 2009. Vol. 13 (5). Pp. 973–990 [Electronic resource]. – Access mode: https://www.researchgate.net/publication/224574306_Group_Search_Optimizer_An_Optimization_Algorithm_Inspired_by_Animal_Searching_Behavior

27. Rao H. A Modified Group Teaching Optimization Algorithm for Solving Constrained Engineering Optimization Problems / H. Rao // Mathematics. 2022. Vol. 10 (20). P. 3765. DOI 10.3390/math10203765

28. Родзин С.И. Вычислительная модель биогеографии и ее применение для задачи коммивояжера / С.И. Родзин, О.Н. Родзина // Известия ЮФУ. Технические науки. 2015. №6 (167). С. 210–222 [Электронный ресурс]. – Режим доступа: http://izv-tn.tti.sfedu.ru/wp-content/uploads/2015/6/19.pdf

29. Кравченко Ю.А. Интеллектуальные системы: эволюция моделей и методов приобретения, управления и передачи знаний / Ю.А. Кравченко, В.В. Курейчик, С.И. Родзин. Чебоксары: ИД Среда, 2023. – 192 с [Электронный ресурс]. – Режим доступа: https://phsreda.com/e-articles/10557/Action10557-108468.pdf

30. Rodzin S. Integration and Transfer of Information: Search Engine Optimizer Inspired by Memes / S. Rodzin, L. Rodzina // Proc. Int. Russian Automation Conf. (RusAutoCon). 2023. Publ. IEEE. Pp. 219–225. DOI 10.1109/RusAutoCon58002.2023.10272935

Для заметок

Для заметок

Учебное издание

Родзин Сергей Иванович
Родзина Ольга Николаевна

МАШИННОЕ ОБУЧЕНИЕ: МЕТАЭВРИСТИКИ ДИФФЕРЕНЦИАЛЬНО-ВЕКТОРНОГО ДВИЖЕНИЯ

Учебное пособие

Чебоксары, 2024 г.

Компьютерная верстка *К.Д. Каймакова*
Дизайн обложки *М.С. Федорова*

Подписано в печать 10.04.2024 г.
Дата выхода издания в свет 13.04.2024 г.
Формат 60×84/16. Бумага офсетная. Печать офсетная.
Гарнитура Times. Усл. печ. л. 8, 1375. Заказ К-1275. Тираж 500 экз.

Издательский дом «Среда»
428005, Чебоксары, Гражданская, 75, офис 12
+7 (8352) 655-731
info@phsreda.com
https://phsreda.com

Отпечатано в Студии печати «Максимум»
428005, Чебоксары, Гражданская, 75
+7 (8352) 655-047
info@maksimum21.ru
www.maksimum21.ru