Методы стохастической оптимизации

Book · March 2016

CITATIONS
0

READS
2,910

3 authors, including:

Pavel V Matrenin
Novosibirsk State Technical University
28 PUBLICATIONS **38** CITATIONS
SEE PROFILE

Vg Sekaev
5 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

П.В. МАТРЕНИН, М.Г. ГРИФ, В.Г. СЕКАЕВ

МЕТОДЫ СТОХАСТИЧЕСКОЙ ОПТИМИЗАЦИИ

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

Новосибирск
2016

УДК 519.856(075.8)
М 346

Рецензенты:
канд. техн. наук, доц. *А.В. Гаврилов*;
канд. техн. наук, доц. *В.С. Поздняков*

Работа подготовлена на кафедре
автоматизированных систем управления
для студентов дневного отделения по курсу взаимосвязанных
дисциплин «Интеллектуальные системы» и «Интеллектуальные
системы и технологии» (ООП по направлениям 09.04.01
«Информатика и вычислительная техника» (магистерская программа
«Компьютерное моделирование систем», квалификация «магистр»),
09.03.01 «Информатика и вычислительная техника»
(квалификация «бакалавр»)

Матренин П.В.
М 346 Методы стохастической оптимизации: учебное пособие / П.В. Матренин, М.Г. Гриф, В.Г. Секаев. – Новосибирск: Изд-во НГТУ, 2016. – 67 с.

ISBN 978-5-7782-2861-0

Учебное пособие раскрывает принципы, модели и методы стохастической оптимизации, рекомендации по их реализации и применению на практике, дает примеры использования. Кроме того, пособие содержит описание практической работы по данной теме.

Адресовано студентам и специалистам, изучающим методы оптимизации и системы искусственного интеллекта.

УДК 519.856(075.8)

ISBN 978-5-7782-2861-0

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Детерминированные эвристические (жадные) методы основаны на идее локально оптимальных выборов на каждом шаге. Принцип жадного выбора может дать оптимальное решение, если последовательность таких выборов дает глобально оптимальное решение [15], т. е. на каждом шаге алгоритм делает выбор такого варианта, который кажется наилучшим на данном шаге. Выбор, сделанный в жадном алгоритме, может зависеть от сделанных ранее выборов, но он никак не зависит от выборов на последующих шагах или от решений последующих подзадач, в отличие от метода динамического программирования. Такие методы позволяют получать решения очень быстро, так как формируют лишь один вариант решения задачи на основе некоторых правил, но полученное таким образом решение может оказаться как наилучшим, так и очень далеким от наилучшего в зависимости от экземпляра задачи.

Например, для задачи календарного планирования, в которой нужно определить наилучшую последовательность выполнения этапов, реализация жадного алгоритма может быть, например, такой: «выбирать на каждом шаге этап, выполнение которого завершится как можно раньше». Существуют более сложные жадные эвристические алгоритмы, основанные на назначении требованиям приоритетов в зависимости от многих факторов, а также на использовании нескольких эвристических правил для совершения выбора на каждом шаге. Однако даже сложные эвристические правила могут быть эффективными только в сочетании с другими более мощными оптимизационными алгоритмами. Во многих задачах оптимизации эффективность жадных алгоритмов очень сильно зависит от условий конкретной задачи.

Стохастические (рандомизированные) эвристические методы также не гарантируют получение точного решения, но они, как правило, позволяют находить достаточно близкие для практического использования решения за приемлемое время. При этом стохастическая природа

большинства методов делает их применение нетривиальной задачей, поскольку для каждой алгоритмической реализации и для каждого класса задач оптимизации эффективность, быстродействие, сходимость, влияние условий задачи и параметров алгоритма требуют тщательного исследования. Сказанное здесь относится только к стохастическим методам, основанным на некоторых эвристиках. Кроме них существуют и очень простые стохастические методы, такие как случайный поиск.

Рассматриваемые далее алгоритмы роевого интеллекта, эволюционные алгоритмы и алгоритм имитации отжига имеют одну основу, а именно конечные цепи Маркова. Однако рассмотренные алгоритмы появились не в результате изучения цепей Маркова или их применения для решения задач оптимизации. Они стали результатом моделирования, например, поведения птиц в стае (алгоритм роя частиц) или изучения принципов поведения муравьев (алгоритм «колонии муравьев» или «муравьиный алгоритм»). Уже после распространения генетического алгоритма и других к ним были применены различные математические модели, в том числе и цепи Маркова. Использование цепей Маркова позволяет доказать сходимость рассматриваемых алгоритмов к глобальному оптимуму только теоретически при устремлении времени работы алгоритма к бесконечности. Однако такой подход не объясняет показанную в многочисленных экспериментах высокую эффективность рассматриваемых стохастических алгоритмов оптимизации для решения практических задач с ограничениями по времени. Рассматриваемые алгоритмы называют алгоритмами дискретной оптимизации, поскольку они работают пошагово, по итерациям, а не непрерывно. Область же применения включает в себя и задачи дискретной оптимизации (комбинаторные), и задачи непрерывной оптимизации, и гибридные задачи.

1. ОПИСАНИЯ АЛГОРИТМОВ

1.1. Термины и обозначения

Среди стохастических методов оптимизации особенно хорошо зарекомендовали себя на практике методы, использующие закономерности и принципы, заимствованные у самой природы, такие как методы эволюционной оптимизации, методы роевого интеллекта, алгоритмы имитации отжига. Первые две группы относятся к так называемым популяционным методам, поскольку используют системы, состоящие из агентов (популяций агентов).

Как правило, под агентом понимается некоторая точка в пространстве поиска решений задачи, а процесс оптимизации заключается в перемещении агентов в этом пространстве. При этом методы эволюционной оптимизации предполагают создание на каждом шаге новых популяций агентов с учетом опыта, полученного предыдущими популяциями. Методы роевого интеллекта используют некоторые правила, задающие косвенный обмен информацией между агентами. Алгоритмы роевого интеллекта принципиально отличаются от эволюционных, поскольку не требуют создания на каждом шаге новых популяций путем отбора и скрещивания агентов предыдущей популяции, а используют коллективные децентрализованные перемещения агентов одной популяции, без процедур отбора, уничтожения старых и порождения новых агентов. Способ определения, относится ли тот или иной популяционный алгоритм к роевому интеллекту, предложен в [11]: в формулах, задающих поведение (перемещения, миграцию) агентов роя, должен присутствовать объект, обеспечивающий косвенный обмен информацией между агентами.

К эволюционным алгоритмам относятся:

– алгоритм растущих деревьев;

– алгоритм эволюции разума;

– бактериальная оптимизация;

– гармонический поиск;
– генетический алгоритм;
– культурный алгоритм;
– кукушкин поиск;
– меметические алгоритмы;
– сорняковый алгоритм.

Роевой интеллект включает в себя такие алгоритмы:
– алгоритм гравитационного поиска;
– алгоритм динамики формирования рек;
– алгоритм колонии муравьев;
– алгоритм летучих мышей;
– алгоритм роения бактерий;
– алгоритм роя пчел;
– алгоритм роя частиц;
– алгоритм светлячков;
– алгоритм стаи волков;
– гармонический поиск;
– интеллектуальные капли воды;
– обезьяний алгоритм;
– поиск косяком рыб;
– стохастический диффузионный поиск;
– тасующий алгоритм прыгающих лягушек;
– улучшенный алгоритм поиска кукушки;
– электромагнитный поиск.

Введем следующие обозначения [11]:

$f(X)$ – скалярная целевая функция (фитнесс-функция), для которой требуется найти экстремальное значение;

X – вектор варьируемых параметров;

D – область допустимых значений X, $D \in \mathbf{R}|X|$;

$G(X)$ – функция, задающая ограничения на X;

$|S|$ – количество агентов популяции;

$S = \{s_1, s_2, \ldots, s_{|S|}\}$ – множество всех агентов;

X_{ij} – вектор варьируемых параметров i-го агента на j-й итерации алгоритма;

X^{opt} – наилучшее значение вектора варьируемых параметров;

f^{opt} – наилучшее значение целевой функции;

$\varphi(X) = f(X)$ – значение фитнесс-функции в положении X (фитнесс-агента).

Рассматривается задача максимизации, которая легко сводится к задаче минимизации:

$$f^{\text{opt}} = f(X^{\text{opt}}) = \max_{X \in D} f(X). \quad (1)$$

Генетический алгоритм и алгоритм имитации отжига хорошо описаны в литературе, а в описании алгоритмов роевого интеллекта существуют некоторые трудности из-за отсутствия четкой общепринятой классификации и путаницы в терминах, поэтому эти алгоритмы описаны подробнее.

На основании проведенного анализа [11] предлагается единая схема обобщенного описания алгоритмов роевого интеллекта, предполагающая представление алгоритма в следующем виде:

$$SI = \{S, M, A, P, I, O\}, \quad (2)$$

где S – множество агентов роя; M – объект для обмена опытом между агентами S; A – правила работы роевого алгоритма; P – параметры, используемые в правилах A; I и O – порты (входы и выход) роевого алгоритма, посредством которых он взаимодействует с окружающей средой и управляющей системой.

Любой алгоритм роевого интеллекта включает в себя следующие этапы, определяемые правилами A:

1) инициализация начальных состояний агентов S;

2) вычисление фитнесс-функции для каждого агента;

3) миграция (перемещение) агентов S с учетом правил A, вектора параметров P и объекта обмена опытом между агентами M.

По этой схеме и будут даны описания алгоритмов роя пчел, роя частиц, колонии муравьев и алгоритма поиска косяком рыб.

Ниже кратко рассмотрены семь распространенных стохастических алгоритма дискретной оптимизации:

– случайный поиск и поиск с возвратом;
– алгоритм имитации отжига (Simulated Annealing),
– генетический алгоритм (Genetic Algorithm),
– алгоритм роя части (Particle Swarm Optimization),
– алгоритм роя пчел (Art Bee Colony),
– алгоритм колонии муравьев,
– алгоритм поиска косяком рыб.

Каждый из этих алгоритмов имеет большое количество модификаций, поэтому даются базовые описания.

1.2. Случайный поиск, поиск с возвратом

Самым простым методом поиска является произвольная выборка решений. Случайным образом генерируются и оцениваются произвольные решения X до тех пор, пока не будет найдено достаточно хорошее решение или не пройдет заданное время или число итераций алгоритма. В качестве результата выбирается наилучшее решение из всех, которые были сформированы в процессе выборки. Основным достоинством метода является простота, хотя выбор произвольных решений с одинаковой вероятностью может быть нетривиальной задачей. Существуют некоторые условия, при выполнении которых метод случайного поиска можно успешно применять [15]:

– пространство решений содержит высокую долю удовлетворительных решений;

– пространство решений не является однородным. Иными словами, нельзя определить признаки приближения к оптимальному решению.

Простые числа длинами в несколько сотен цифр часто получают путем случайной генерации чисел и проверки их на простоту, так как такие числа встречаются относительно часто и сложно выделить признаки близости известного произвольного числа к ближайшему неизвестному простому числу.

Случайный поиск неэффективен для задач, пространство решений которых не является однородным. Можно устранить этот недостаток, если на каждой итерации алгоритма учитывать качество предыдущего решения (поиск с возвратом). Для этого в пространстве оптимизируемых параметров делается шаг в случайном направлении. Если значение критерия в новом состоянии не лучше, чем ее значение на предыдущем шаге, то поиск возвращается в предыдущее состояние, после чего снова делается шаг в случайном направлении. Если же шаг удачен, т. е. новое состояние лучше предыдущего, то последующий шаг делается уже из нового состояния. Эффективность метода низкая, так как при попадании в локальный экстремум длина шага может быть недостаточной для выхода из него.

Два рассмотренных метода имеют очень ограниченную область эффективного использования по сравнению с рассмотренными далее методами, однако приемы генерации случайного решения, выполнения шага для перехода в новое состояние и возврата так или иначе используются во всех стохастических методах.

1.3. Алгоритм имитации отжига

Алгоритм возник в середине 1980-х годов, его основатели: Скотт Киркпатрик, Даниель Желатт, Марио Вечи и Владо Черни [27]. Алгоритм основан на аналогии с процессом кристаллизации вещества, например, при отжиге металла. В ходе этого процесса температура вещества понижается, оно отвердевает, при этом замедляется скорость движения частиц вещества.

Кристаллическую решетку можно представить как систему частиц, а ее энергетическое состояние – совокупностью состояний частиц. Частицы переходят из одного энергетического состояния в другое произвольным образом, но вероятность переходов зависит от температуры системы. Вероятность перехода из высокоэнергетического состояния в низкоэнергетическое велика при любой температуре, также существует отличная от нуля вероятность перехода в состояние с более высоким значением энергии. Эта вероятность тем выше, чем меньше разница между состояниями и чем выше температура системы.

Если в роли физической системы представить задачу оптимизации, в роли энергии системы – значение целевой функции $f(X)$, а в роли частиц – управляющие переменные X, то можно решать задачу оптимизации функции $f(X)$, используя механизмы и законы, которые определяют процесс отвердевания. Необходимо задать закон, по которому будет меняться температура системы, закон, по которому будет случайным образом изменяться значение координат в пространстве решения, и правило определения перехода частицы в точку с новыми координатами. Кроме того, нужно выбрать начальную и конечную температуру (T_0 и T_f).

Ниже приведены шаги алгоритма.

1. Генерация случайного начального состояния.

2. Сравнение значения функции с наилучшим найденным, если текущее (X) лучше, то оно принимается в качестве лучшего.

3. Вычисление случайного нового состояния (X_{new}) и определение значения функции для него. Закон распределения может быть любым, например, экспоненциальным:

$$x = x + r_1 M_x \ln(r_2),$$

где x – одна из координат состояния X; r_1 – случайное число (–1 или +1); r_2 – случайное число, равномерно распределенное от 0 до 1;

M_x – параметр алгоритма, от которого зависит, как далеко от текущей точки может переместиться процесс поиска за один шаг.

4. Если новое состояние лучше текущего, система переходит в новое состояние ($X = X_{new}$), иначе случайным образом принимается решение о переходе или не переходе в новое состояние. При этом в случае максимизации может быть использовано следующее условие перехода:

$$\frac{\varphi(X_{new}) - \varphi(X)}{T} > r,$$

где T – текущая температура; r – случайное число, равномерно распределенное от 0 до 1.

5. Если температура достигла конечной, алгоритм завершается, иначе температура понижается и происходит переход на шаг 2. Новое значение температуры на i-м шаге может быть вычислено как

$$T = \frac{T_0}{e^{ivt}},$$

где vt – коэффициент, влияющий на скорость снижения температуры.

При завершении алгоритма сохраненное в памяти наилучшее решение и будет результатом работы алгоритма.

Алгоритм имитации отжига прост в реализации и показывает высокую эффективность в решении самых разных задач оптимизации [15]. Однако для получения высокой эффективности может потребоваться тщательная настройка формулы понижения температуры как численных параметров, так и самого вида зависимости, кроме того, часто температуру меняют не на каждой итерации алгоритма, а через определенное число итераций.

1.4. Генетический алгоритм

Впервые компьютерным моделированием эволюционного отбора занялся Нильс Баричелли в 1954 году. Признание генетического алгоритма как метода решения оптимизационных задача произошло в 1960–70 годах в результате работ Инго Рехенберга и Джона Голланда

[24]. Генетический алгоритм (ГА) относится к стохастическим методам и основан на принципе естественного эволюционного отбора. Идея генетических алгоритмов заимствована у живой природы и состоит в моделировании эволюционного процесса, конечной целью которого является получение оптимального решения сложной комбинаторной задачи. В описании метода используются упрощенные биологические термины.

Основной идеей ГА является борьба за существование между решениями задачи [24]. Каждое решение записывается в виде некоторого вектора значений (аллелей), который называется хромосомой, или особью. Совокупность решений называют популяцией. Каждая особь в популяции оценивается значением целевой функции, рассчитанной на основе значений из хромосомы. Более перспективные решения проходят в следующую стадию и оказывают влияние на «потомство», т. е. на вновь генерируемые решения.

Необходимо представить задачу таким образом, чтобы ее решение можно было бы записать в виде генотипа, т. е. вектора значений (генов). Например, для поиска максимума функции десяти аргументов $y(x_1, x_2,\ldots, x_{10})$ генотип будет состоять из десяти чисел (значений x_1, $x_2,\ldots, x_{10}$), а значение функции y от этих чисел характеризует приспособленность генотипа, поэтому оптимизируемая функция также называется функцией приспособленности, или фитнесс-функцией.

Стандартный ГА начинает свою работу с формирования начальной популяции как конечного набора допустимых решений задачи (хромосом). Эти решения могут быть выбраны как случайным образом, так и с помощью приближенных алгоритмов. На каждом шаге эволюции формируется новая популяция с помощью вероятностного оператора селекции. Для каждой новой особи популяции выбираются два решения, или «родители» (в общем случае родителей может быть и больше двух). Вероятность выбора решения в качестве родительского прямо пропорциональна его качеству. В ходе скрещивания двух особей они по какому-либо правилу обмениваются элементами хромосом (такой обмен называют кроссинговером). Кроссинговер (употребляется также название кроссовер, или скрещивание) создает из родительских хромосом одну или несколько новых хромосом. В простейшем случае кроссинговер в генетическом алгоритме реализуется путем разрезания вектора хромосом родителей в случайной позиции и обмене частями векторов (рис. 1).

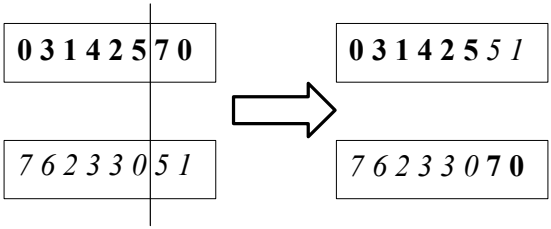

Рис. 1. Пример кроссинговера

Оператор скрещивания по этим решениям строит новое решение, которое затем подвергается небольшим случайным модификациям (мутациям). Мутация – преобразование хромосомы, случайно изменяющее одну или несколько ее позиций (генов). Наиболее распространенный вид мутаций – случайное изменение только одного из генов хромосомы. Мутациям подвергаются не все хромосомы, а только выбранные с определенной вероятностью. В результате получается новая популяция, а старая полностью или частично уничтожается. Популяция следующего поколения в большинстве реализаций генетических алгоритмов содержит столько же особей, сколько и начальная, но в силу отбора приспособленность в ней в среднем выше. Теперь описанные процессы отбора, скрещивания и мутации повторяются уже для этой популяции и т. д.

В каждом следующем поколении наблюдается возникновение новых хороших и плохих решений рассматриваемой задачи. Благодаря тому что более хорошие решения отбираются с большей вероятностью, их число увеличивается. Иногда в ГА сохраняют несколько лучших хромосом текущего поколения (элитные хромосомы). Схема работы ГА может быть представлена следующим образом.

1. Генерация начальной популяции.
2. Вычисление целевой функции по каждой особи (хромосоме).
3. Выбор особей и их скрещивание (кроссинговер).
4. Случайные изменения особей (мутация).
5. Если условие завершения выполнено, то конец работы, иначе перейти к п. 2.

1.5. Алгоритм роя частиц

Специалист в области компьютерной графики Крейг Рейнольдс создал в 1986 году компьютерную модель для визуальной имитации поведения стаи птиц, основанную на том, что движение каждой птицы описывается рядом простых правил, связанных с ориентацией на остальных птиц стаи. Модель заинтересовала ученых и в 1995 году Джеймс Кеннеди и Рассел Эберхарт предложили метод для оптимизации непрерывных функций, названный ими алгоритмом роя частиц [26].

Стая птиц всегда действует скоординированно, каждая птица действует согласно простым правилам, следит за другими птицами и согласует свое движение с ними. Найдя источник пищи, птица сообщает о нем всей стае. Именно этот факт создает коллективное поведение и роевой интеллект. Источники пищи обычно расположены случайным образом, и одной птице очень сложно быстро найти их. Только в том случае, если птицы будут обмениваться информацией, вся стая сможет выжить.

При переходе к модели слово «птица» заменено на слово «частица» и вводятся следующие положения:

– частицы существуют в мире, где время дискретно;

– частицы оценивают свое положение с помощью фитнесс-функции;

– каждая частица знает позицию в пространстве, в которой она нашла наибольшее количество пищи (своя наилучшая позиция);

– каждая частица знает позицию в пространстве, в которой найдено наибольшее количество пищи среди всех позиций, в которых были все частицы (общая наилучшая позиция);

– частицы имеют тенденцию стремиться к лучшим позициям, в которых были сами, и к общей наилучшей позиции;

– частицы случайным образом меняют свою скорость, так что описанная тенденция определяет лишь усредненное движение частиц;

– частицы обладают инерцией, поэтому их скорость в каждый момент времени зависит от скорости в предыдущий момент;

– частицы не могут покинуть заданную область поиска.

Основная идея метода заключается в перемещении частиц в пространстве решений. Пусть решается задача нахождения минимума (максимума) функции вида $f(X)$, где X – вектор варьируемых параметров, которые могут принимать значения из некоторой области D. Тогда каждая частица в каждый момент времени характеризуется значением параметров X из области D (координатами точки в пространстве решений) и значением оптимизируемой функции $f(X)$ (привлекательностью

данной точки). При этом частица «помнит» наилучшую точку в пространстве решений, в которой была, и стремится в нее вернуться, но подчиняется также закону инерции и имеет склонность к небольшому стохастическому изменению направления движения. Однако этих правил недостаточно для перехода к системе, так как не заданы связи между элементами. В качестве связи используется так называемая общая память, благодаря которой каждая частица знает координаты наилучшей точки среди всех, в которых была любая частица роя. В итоге на движение частицы влияют стремление к своему наилучшему положению, стремление к наилучшему среди всех частиц положению, инерционность и случайные отклонения.

Алгоритм завершается при достижении заданного числа итераций либо при достижении удовлетворительного решения, либо после истечения отведенного на работу времени. Таким образом, алгоритм можно записать следующим образом.

1. Случайно распределить частицы в области решения, назначить нулевые начальные скорости.

2. Значения оптимизируемой функции по каждой частице с обновлением при необходимости локальных и глобальных лучших решений.

3. Вычислить новые значения скоростей по каждой частице.

4. Вычислить новые координаты частиц.

5. Если выполнено условие завершения, закончить алгоритм, иначе перейти на шаг 2.

Результатом работы алгоритма является глобальное лучшее решение.

Согласно формуле (2) алгоритм роя частиц $PSO = \{S, M, A, P, I, O\}$.

1. Множество агентов (частиц) $S = \{s_1, s_2, \ldots, s_{|S|}\}$, $|S|$ – количество частиц. На j-й итерации i-я частица характеризуется состоянием $s_{ij} = \left\{X_{ij}, V_{ij}, X_{ij}^{best}\right\}$, где $X_{ij} = \left\{x_{ij}^1, x_{ij}^2, ..., x_{ij}^{\ell}\right\}$ – вектор варьируемых параметров (положение частицы); $V_{ij} = \left\{v_{ij}^1, v_{ij}^2, ..., v_{ij}^{\ell}\right\}$ – вектор скоростей частицы; $X_{ij}^{best} = \left\{b_{ij}^1, b_{ij}^2, ..., b_{ij}^{\ell}\right\}$ – наилучше по значению фитнесс-функции положение частицы среди всех положений, которые она занимала в процессе работы алгоритма от 1-й до j-й итераций; ℓ – количество варьируемых параметров.

2. Вектор $M = X_j^{best}$ – наилучшее значение вектора варьируемых параметров, которое было получено среди всех частиц от 1-й до

j-й итерации алгоритма. Этот вектор обеспечивает косвенный обмен опытом между частицами.

3. Алгоритм A описывает механизмы функционирования роя частиц. Существуют различные модификации этого алгоритма. Далее представлено описание базового алгоритма.

3.1. Генерация начальных положений и скоростей ($j = 1$):

$$X_{i1} = rand(G(X)),\ i = 1,\ldots,|S|,$$

где $rand(G(X))$ – вектор равномерно распределенных случайных величин, отвечающих ограничениям на область поиска;

$$V_{i1} = rand(V_{\min},\ V_{\max}),\ i = 1,\ldots,|S|,$$

где $rand(V_{\min},\ V_{\max})$ – вектор равномерно распределенных случайных величин в диапазоне $(V_{\min},\ V_{\max})$.

$$X_{i1}^{best} = X_{ij},\ i = 1,\ldots,|S|.$$

Произвольно выбирается наилучшая позиция (при вычислении фитнесс-функций будет определена действительно наилучшая позиция):

$$X_1^{best} = X_{11}.$$

3.2. Вычисление фитнесс-функций и определение наилучшего положения:

$$X_{ij}^{best} = X_{ij}, \varphi(X_{ij}^{best}) < \varphi(X_{ij}),\ i = 1,\ldots,|S|,$$

$$X_j^{best} = X_{ij}, \varphi(X_j^{best}) < \varphi(X_{ij}),\ i = 1,\ldots,|S|.$$

Вычисление $\varphi(X) = f(X)$ происходит во внешней среде с помощью обмена данными по обратной связи $(I_{\text{ос}},\ O_{\text{ос}})$.

3.3. Перемещения частиц:

$$V_{ij+1} = V_{ij}\omega + \alpha_1\left(X_{ij}^{best} - X_{ij}\right)rnd_1 + \alpha_2(M - X_{ij})rnd_2,\ i = 1,\ldots,|S|,$$

$$V_{ij+1} = \begin{cases} V_{ij+1}, V_{\min} \le V_{ij+1} \le V_{\max}, \\ V_{\min}, V_{ij+1} \le V_{\min}, \\ V_{\max}, V_{ij+1} \ge V_{\max}; \end{cases} \quad i = 1,\ldots,|S|,$$

$$X_{ij+1} = \begin{cases} X_{ij} + V_{ij+1}, G(X_{ij} + V_{ij+1}) = 1, \\ X_{ij}, G(X_{ij} + V_{ij+1}) = 0, \end{cases} \quad i = 1,\ldots,|S|,$$

где rnd_1 и rnd_2 – случайные числа, равномерно распределенные в интервале [0,1); $G(X)$ здесь используется как предикат, который показывает, принадлежит ли X области допустимых значений D.

3.4. Если на j-й итерации выполнено условие остановки, то значение $X_{final}^{best} = X_j^{best}$ подается на выход O_1. Иначе происходит переход к итерации 2.

4. Вектор $P = \{\alpha_1, \alpha_2, \omega\}$ – коэффициенты алгоритма A, которые используются в формуле и влияют на перемещения частиц в пространстве поиска. Коэффициенты α_1 и α_2 определяют соответственно степень учета индивидуального и группового опыта агентов. Коэффициент ω характеризует инерционные свойства частиц.

5. Идентификаторы I и O – описанные выше вход и выход роя, которые не зависят от реализации алгоритмов роевого интеллекта.

1.6. Алгоритм роя пчел

Алгоритм роя пчел (Artificial Bee Colony Algorithm или Bees Algorithm) разработан в 2005 году несколькими авторами [28]. Метод основан на симуляции поведения пчел при поиске нектара. Рой пчел отправляет несколько разведчиков в случайных направлениях для поиска нектара. Вернувшись, разведчики сообщают о найденных на поле участках с цветами, содержащими нектар, и на них вылетают остальные пчелы. При этом чем больше на участке нектара, тем больше пчел к нему устремляется. Однако при этом пчелы могут случайным образом отклоняться от выбранного направления. После возвращения всех пчел в улей вновь происходит обмен информацией и отправка пчел-разведчиков и пчел-рабочих. Фактически разведчики действуют по алгоритму случайного поиска.

Для перехода к формальному описанию алгоритма необходимо представить поле с цветами как пространство поиска решения, а количество нектара как критерий задачи оптимизации, т. е. целевую функцию. На каждом шаге работы алгоритма среди всех агентов выбирается n^b лучших по значению целевой функции. Среди прочих выбирается еще n^g лучших, так называемых «выбранных», или «перспективных» (здесь верхние индексы не являются показателем степени). В некоторых вариантах алгоритма требуется, чтобы расстояния между каждой парой позиций в объединенном множестве лучших и выбранных позиций не превышали определенной величины. Иными словами, если есть две близкие позиции, то худшая из них по значению целевой функции отбрасывается, вместо нее берется позиция другого агента, подходящая под условия.

Определенные таким образом позиции (множество лучших N^b, и множество выбранных N^g) запоминаются, и на следующем шаге в окрестность каждой лучшей позиции высылается c^b пчел, а каждой выбранной c^g. Пчела, посылаемая в окрестность участка, попадает в случайную точку внутри окрестности, например, в двумерном пространстве окрестность позиции с центром в точке (x, y) представляет область $([x - rx; x+rx], [y - ry; y+ry])$, где rx и ry – параметры алгоритма. Возможно использование одного коэффициента rx по всем измерениям или вектора RX. Пчелы-разведчики в количестве n^s высылаются в случайные позиции по всему пространству поиска на каждой итерации.

Алгоритм завершается при достижении заданного числа итераций либо при достижении удовлетворительного решения, либо после истечения отведенного на работу времени. Ниже приведены шаги алгоритма.

1. Послать n^s разведчиков в случайные точки и вычислить значения оптимизируемой функции в них.

2. Отобрать из полученных участков N^b лучших и N^g выбранных с учетом пересечений. Может оказаться, что участков меньше $N^b + N^g$, тогда выбрать все имеющиеся с учетом пересечений.

3. Послать в окрестность каждого лучшего участка c^b пчел и в окрестность каждого выбранного участка c^g пчел с вычислением целевой функции.

4. Послать n^s разведчиков.

5. Если выполнено условие окончания алгоритма, то завершение алгоритма, иначе переход на шаг 2.

Лучшее из всех полученных на всех итерациях решение и будет результатом работы алгоритма.

Согласно формуле (2) для получения обобщенного описания алгоритм роя пчел необходимо представить в виде

$$ABCO = \{S, M, A, P, I, O\}$$

и описать каждый элемент.

1. Множество агентов (пчел) $S = \{s_1, s_2, \ldots, s_{|S|}\}$, $|S|$ – количество агентов. На j-й итерации i-й агент характеризуется состоянием $s_{ij} = \{X_{ij}\}$, где $X_{ij} = \{x^1_{ij}, x^2_{ij}, \ldots, x^{\ell}_{ij},\}$ – вектор варьируемых параметров (положение агента); ℓ – размерность пространства поиска решений.

2. Средством косвенного обмена M является список лучших и перспективных позиций, найденных на j-й итерации. Нужно отметить, что в отличие от эволюционных алгоритмов отбираются не агенты, а только соответствующие позиции в пространстве поиска, $M = \{N_{ij}^{b}, N_{kj}^{g}\}$, $i = 1, \ldots, n^b$, $k = 1, \ldots, n^g$.

3. Алгоритм согласно формуле (1), реализующий правила A, описывает механизмы функционирования роя пчел, в общем виде может быть записан следующим образом.

3.1. Инициализация начальных положений ($j = 1$) выполняется только для разведчиков:

$$X_i = rand(G(X)),\ i = 1, \ldots,\ n^s,$$

где n^s – количество пчел-разведчиков.

3.2. Вычисление фитнесс-функций каждого агента на текущей j-й итерации:

$$\varphi(X_{ij}) = f(X_{ij}) i = 1, \ldots,\ |S|.$$

3.3. Миграция агентов. После формирования списков лучших и перспективных, найденных на $(j-1)$-й итерации (N_{ij}^{b}, N_{kj}^{g}), в окрестности позиций отправляются пчелы-рабочие. В окрестность каждой лучшей позиции отправляется c^b пчел, в окрестность каждой перспективной c^g пчел. В некоторых вариантах алгоритма число отправляющихся в окрестность участка пчел зависит от его качества с точки зрения целевой функции, но в данном описании это не используется. Таким образом, позиции всех пчел-рабочих определяются следующим образом:

$$X_{(i-1)c^b + k\ j} = N^b_{ij-1} + Rnd \cdot rad,\ i = 1, \ldots,\ n^b,\ k = 1, \ldots,\ c^b, \qquad (3)$$

$$X_{n^b c^b + (i-1)c^b + k\ j} = N^g_{ij-1} + Rnd \cdot rad,\ i = 1, ...,\ n^g,\ k = 1, ...,\ c^g, \qquad (4)$$

где *Rnd* – вектор, состоящий из *l* равномерно распределенных случайных величин в интервале от –1 до 1.

Пчелы-разведчики в этом случае отправляются в случайные позиции, координаты которых являются случайными величинами, равномерно распределенными на всем допустимом диапазоне значений:

$$X_{n^b c^b + n^g c^g + i\ j} = rand(G(X)),\ i = 1, ...,\ n^s.$$

4. Коэффициенты (параметры) алгоритма, используемые в данном описании и формулах (3) и (4) образуют вектор коэффициентов алгоритма $P = \{n^s, n^b, n^g, c^b, c^g, rad, rx\}$ – коэффициенты алгоритма, который согласно формуле (2) использует правила *A*. Коэффициент *rad* определяет рассеяние агентов при отправлении на лучшие и перспективные позиции, коэффициент *rx* задает минимально возможные расстояния между этими позициями. Значение выражения $n^s + n^b c^b + n^g c^g$ равно общему количеству агентов роя $|S|$. От выбора параметров алгоритма значительно зависит качество получаемых решений, поэтому для повышения эффективности алгоритма необходимо адаптировать значения параметров.

5. Идентификаторы *I* и *O* – вход и выход роя, которые не зависят от реализации алгоритма роевого интеллекта.

1.7. Алгоритм поиска косяком рыб

Алгоритм поиска косяком рыб (Fish School Search, FSS) создан на основании моделирования движения косяка рыб и представлен Б. Фило и Л. Нето в 2008 году [6]. В косяке рыбы двигаются в одном направлении с близкими скоростями, поддерживая согласованное движение и приблизительно одинаковое расстояние между соседями. Такой способ коллективного движения помогает эффективнее перемещаться на большие расстояние, находить пищу и защищаться от хищников. Как и для алгоритма роя частиц, агенты (рыбы) перемещаются в пространстве поиска решений, но правила перемещения и средство косвенного обмена информацией значительно отличаются.

Каждый агент выполняет несколько видов движений: во-первых, движения на основании только своего собственного опыта, во-вторых,

на основании опыта всего косяка. Второй вид движения разделяется на две фазы, описанные ниже. В отличие от алгоритма роя частиц косяк рыб «помнит» только результаты предыдущей итерации, а не наилучшие найденные результаты за весь процесс. Для применения алгоритма требуется, чтобы целевая функция была неотрицательна на всем пространстве поиска: $f(X) \geq 0, X \in D$.

Согласно формуле (1) для получения обобщенного описания алгоритма поиска косяком рыб его необходимо представить в виде $FSS = \{S, M, A, P, I, O\}$.

Затем требуется поочередно описать каждый элемент *FSS* и действия на отдельных этапах работы алгоритма: инициализацию, вычисление фитнесс-функций и миграцию.

1. Множество агентов (рыб) $S = \{s_1, s_2, \ldots, s_{|S|}\}$, $|S|$ – количество агентов. На j-й итерации i-й агент характеризуется состоянием $s_{ij} = \{X_{ij}, V_{ij}, w_{ij}\}$, где $X_{ij} = \{x^1_{ij}, x^2_{ij}, \ldots, x^{\ell}_{ij},\}$ – вектор варьируемых параметров (положение агента); ℓ – размерность пространства поиска решений; $V_{ij} = \{v^1_{ij}, v^2_{ij}, \ldots, v^{\ell}_{ij}\}$ – вектор скоростей агента; w_{ij} – вес i-го агента на j-й итерации.

2. Средством косвенного обмена M является вектор из двух элементов. Первый из них является скаляром и определяет взвешенную сумму индивидуальных перемещений рыб, а второй – вектором длиной l и представляет взвешенный центр тяжести всего косяка, $M = \{m^S_j, C_j\}$.

3. Реализация правил A определяет механизм функционирования косяка рыб и согласно используемой схеме описания должна включать в себя инициализацию, вычисление фитнесс-функций и миграцию. Существуют различные вариации алгоритма поиска косяком рыб. Далее представлена схема алгоритма в соответствии с описанием.

3.1. Инициализация начальных положений ($j = 1$):

$$X_{i1} = rand(G(X)),\ i = 1,\ \ldots,\ |S|,$$

где $rand(G(X))$ – вектор равномерно распределенных случайных величин, отвечающих ограничениям на область поиска;

$$w_{i1} = \frac{w_{\max}}{2},\ i = 1, \ldots,\ \left|S\right|,$$

где $w_{\max}$ – один из параметров алгоритма.

3.2. Вычисление фитнесс-функций каждого агента на текущей j-й итерации:

$$\varphi(X_{ij}) = f(X_{ij}) i = 1, ..., |S|.$$

3.3. Миграция, т. е. перемещения агентов в рамках одной итерации, включает в себя несколько стадий, в данное описание вводится промежуточная между итерациями j и $j + 1$ итерация с индексом $j + 0.5$. Этот индекс введен для упрощения формул и показывает, что соответствующие значения скорости i-й частицы $V_{ij+0.5}$, ее положения $X_{ij+0.5}$ и инерции $w_{ij+0.5}$ являются промежуточным в расчетах.

3.3.1. Индивидуальные перемещения агентов между итерациями j и $j + 1$ выполняются независимо для всех агентов и состоят из трех шагов. На первом шаге задается случайное значение скорости:

$$V_{ij+0.5} = rnd \cdot V_{\max}, \; i = 1, ..., |S|,$$

где rnd – случайное число, равномерно распределенное в интервале [0,1]; $V_{\max}$ – вектор максимальных скоростей по каждому измерению области поиска $V_{\max} = \{v^1_{\max}, v^2_{\max}, \ldots, v^{\ell}_{\max}\}$. Вектор может быть заменен скалярной величиной $v_{\max}$ в случае равенства всех его элементов.

На втором шаге выполняется перемещение с найденной скоростью в пределах области допустимых значений:

$$X_{ij+0.5} = \begin{cases} X_{ij} + V_{ij+0.5}, G(X_{ij} + V_{ij+0.5}) = 1, \\ X_{ij}, G(X_{ij} + V_{ij+0.5}) = 0, \end{cases} \quad i = 1, ..., |S|, \qquad (5)$$

где $G(X)$ здесь используется как предикат, который показывает, принадлежит ли X области допустимых значений D.

Последний третий шаг возвращает агента на предыдущую позицию, если значение целевой функции в новой позиции оказалось хуже:

$$X_{ij+0.5} = \begin{cases} X_{ij+0.5}, \varphi(X_{ij+0.5}) \geq \varphi(X_{ij}), \\ X_{ij}, \varphi(X_{ij+0.5}) < \varphi(X_{ij}), \end{cases} \quad i = 1, ..., |S|.$$

3.3.2. Инстинктивно-коллективное плавание выполняется всеми рыбами в одном направлении и с одинаковой скоростью. На этом этапе используется объект косвенного взаимодействия m^S_j:

$$m_j^S = \frac{\sum_i (V_{ij+0.5}(\varphi(X_{ij+0.5}) - \varphi(X_{ij})))}{\sum_i (\varphi(X_{ij+0.5}) - \varphi(X_{ij}))}$$

Каждая рыба после этого перемещается на данную величину:

$$X_{ij+0.5} = X_{ij+0.5} + m_j^S,\ i = 1,\ ...,\ |S|.$$

Если какой-либо элемент $X_{ij+0.5}$ выходит за границу области допустимых значений, то он заменяется на значение соответствующей границы.

3.3.3. Последний этап перемещений называется коллективно-волевым плаванием. Предварительно необходимо вычислить веса агентов; вес агента *i* на шаге *j* вычисляется по формуле

$$w_{ij+0.5} = w_{ij} + \frac{\varphi(X_{ij+0.5}) - \varphi(X_{ij})}{\max(\varphi(X_{ij+0.5}),\ \varphi(X_{ij}))} i = 1,\ ...,\ |S|,$$

с учетом ограничения

$$1 \le w_{ij+0.5} \le w_{\max}.$$

Если в результате индивидуального и инстинктивно-коллективного плавания положение всего роя (косяка) в целом стало лучше, то область поиска сужается для более тщательного исследования текущей занятой области. В противном случае эта область расширяется для поиска новых решений и выхода из потенциального локального экстремума. Используется так называемый центр тяжести роя:

$$C_j = \frac{\sum_i w_{ij+0.5} X_{ij+0.5}}{\sum_i w_{ij+0.5}} i = 1,\ ...,\ |S|. \qquad (6)$$

Перемещения при коллективно-волевом плавании выполняется по следующему правилу:

$$X_{ij+1} = \begin{cases} X_{ij+0.5} + vol(X_{ij+0.5} - C_j),\ ws_{j+0.5} > ws_{j-0.5}, \\ X_{ij+0.5} - vol(X_{ij+0.5} - C_j),\ ws_{j+0.5} \le ws_{j-0.5}, \end{cases} \quad i = 1,\ ...,|S|,$$

где $ws_{j+0.5}$ – сумма весов всех агентов на текущей стадии (знаменатель в формуле (6)), $ws_{j-0.5}$ является аналогичной суммой на предыдущей итерации, а *vol* определяет величину шага перемещений и вычисляется как

$$vol = rnd \cdot vol_{\max},$$

где $vol_{\max}$ – максимально возможный размер шага; *rnd* – случайная величина, равномерно распределенная в интервале [0, 1].

Чтобы получить окончательные позиции агентов на итерации $j + 1$, нужно учесть границы области допустимых значений аналогично формуле (5).

4. Коэффициенты (параметры) алгоритма образуют вектор $P = \{v_{\max}, w_{\max}, vol_{\max}\}$. Эти коэффициенты определяют поведение роя, выбор их значений является особой задачей, которая решается с помощью различных методов адаптации.

5. Идентификаторы *I* и *O* – вход и выход роя, которые не зависят от реализации алгоритма роевого интеллекта.

1.8. Алгоритм колонии муравьев

Алгоритм основан на модели, симулирующей процесс поиска муравьями кратчайших путей от муравейника до источников пищи, и разработан М. Дорино в 1990-е годы [20, 21]. Муравьи решают задачи поиска путей с помощью химической регуляции. Каждый муравей оставляет за собой на земле дорожку особых веществ – феромонов. Другой муравей, почуяв след на земле, устремляется по нему. Чем больше по одному пути прошло муравьев, тем заметнее для них след, а чем более заметен след, тем большее желание пойти в ту же сторону возникает у муравьев. Поскольку муравьи, нашедшие самый короткий путь к «кормушке», тратят меньше времени на путь туда и обратно, их след быстро становится самым заметным. Он привлекает большее число муравьев, таким образом, процесс поиска более короткого пути быстро завершается. Остальные пути – менее используемые – постепенно пропадают. Можно сформулировать основные принципы взаимодействия муравьев: случайность, многократность, положительная обратная связь.

Так как каждый муравей выполняет примитивные действия, то и алгоритм получается очень простым и сводится к многократному об-

ходу некоторого графа, дуги которого имеют не только вес, но и дополнительную динамически меняющуюся количественную характеристику, называемую количеством феромона или просто феромоном.

Итерационный алгоритм включает в себя построение решения всеми муравьями, улучшение решения методом локального поиска, обновление феромона. Построение решения начинается с пустого частичного решения, которое расширяется путем добавления к нему новой допустимой компоненты решения. В отличие от алгоритма роя частиц, который описывается как алгоритм для нахождения экстремумов непрерывных функций, муравьиный алгоритм в классической формулировке решает комбинаторные задачи, например, задачу коммивояжера. Поэтому вектором варьируемых параметров в муравьином алгоритме обычно является последовательность узлов в графе, оптимальный способ обхода которого нужно найти (в задаче коммивояжера – последовательность городов в искомом маршруте). Тем не менее представленная в формуле (2) структура полностью сохраняется.

Согласно формуле (2) алгоритм колонии муравьев $ACO = \{S, M, A, P, I, O\}$. Для удобства будем считать, что решается задача минимизации (так как муравьи ищут наикратчайший путь).

Множество агентов (муравьев) $S = \{s_1, s_2,\ldots,s_{|S|}\}$, $|S|$ – количество муравьев. На j-й итерации i-й муравей характеризуется состоянием $s_{ij} = \{X_{ij}, T_{ij}\}$, где $X_{ij} = \{x^1_{ij}, x^2_{ij},\ldots, x^\ell_{ij}\}$ – вектор варьируемых параметров (последовательность узлов графа), $T_{ij} = \{t^1_{ij}, t^2_{ij},\ldots, t^\ell_{ij}\}$ – вектор булевых переменных, которые показывают, был ли ℓ-й узел посещен i-м муравьем на j-й итерации (в начале каждой итерации все его компоненты равны 0).

Граф $M = \{F, R\}$ – граф, каждая дуга которого имеет два веса: переменный (количество феромона) и постоянный (характеризующий задачу, например, в задаче коммивояжера – это расстояние между городами). Поэтому граф M можно представить в виде двух графов с одинаковыми структурами, но разными весами дуг (F и R):

$$F = \begin{pmatrix} \tau_{11} & \cdots & \tau_{1l} \\ \vdots & \ddots & \vdots \\ \tau_{l1} & \cdots & \tau_{ll} \end{pmatrix}, \; R = \begin{pmatrix} r_{11} & \cdots & r_{1l} \\ \vdots & \ddots & \vdots \\ r_{l1} & \cdots & r_{ll} \end{pmatrix},$$

где τ_{k1k2} – количество феромона на дуге, соединяющей k_1-й и k_2-й узлы графа F, r_{k1k2} – вес (длина) дуги, соединяющей k_1-й и k_2-й узлы графа R, при этом в общем случае $\tau_{k1k2} \neq \tau_{k2k1}$ и $r_{k1k2} \neq r_{k2k1}$.

Алгоритм *A* описывает механизмы функционирования колонии муравьев.

1. Генерация начальных положений. В зависимости от задачи каждый муравей может быть создан в случайном узле графа или в заданном. Если муравей помещен в узел *k*, то

$$x_{i1}^{1} = k, \;\; t_{i1}^{k} = 1.$$

На каждую дугу наносится некоторое ненулевое количество феромона $\tau_{ij} = \tau_{\min}$.

2. В алгоритме колонии муравьев на первой итерации ($j = 1$) второй шаг пропускается, так как для вычисления фитнесс-функций необходимо выполнить перемещения агентов.

Для остальных шагов выполняются следующие действия:

– вычисление целевых функций:

$$\varphi(X_{ij}) = f(X_{ij}), \; i = 1, \; |S|,$$

после каждого вычисления целевой функции происходит сравнение ее значения с $\varphi(X_j^{best})$ аналогично формуле $X_j^{best} = X_{ij}, \varphi(X_j^{best}) < \varphi(X_{ij})$, $i = 1, \ldots, |S|$;

– определение количества феромона, которое нужно нанести на дугу, соединяющую узлы k_1 и k_2 (феромон для *i*-го муравья наносится только на те дуги, которые вошли в маршрут X_{ij}):

$$\Delta\tau_{ij}^{k_1 k_2} = \begin{cases} \dfrac{\gamma}{\varphi(X_{ij})}, \text{в } X_{ij} \; k_1 \text{ следует сразу за } k_2, \\ 0, \text{ если не следует сразу за } k_2, \end{cases} \quad i = 1, \ldots |S|,$$

$$k_1 = 1, \ldots, l, \; k_2 = 1, \ldots, l;$$

– пересчет количества феромона на всем графе с учетом испарения и ограничений:

$$\theta = \rho\tau_j^{k_1 k_2} + \Delta\tau_{ij}^{k_1 k_2}, \;\; k_1 = 1, \ldots, l, \; k_2 = 1, \ldots, l \;,$$

$$\tau_j^{k_1 k_2} = \begin{cases} \theta, \theta \geq \tau_{\min}, \\ \tau_{\min}, \theta < \tau_{\min}. \end{cases}$$

3. Перемещения агентов. В каждом узле k муравей выбирает, в какой из еще не посещенных узлов перейти. При этом вероятность перехода в m-й узел равна (индексы i и j, определяющие агента и итерацию, здесь опущены):

$$P_m = \begin{cases} \dfrac{(\tau_{km})^{\alpha}(\eta(r_{km}))^{\beta}}{\sum_{z=1, t_z=0}^{l}\left((\tau_{kz})^{\alpha}(\eta(r_{kz}))^{\beta}\right)}, & t_m = 0, \\ 0, & t_m = 1. \end{cases}$$

Здесь $\eta(r_{km})$ – некоторая функция от веса дуги, в простейшем случае $\eta(r_{km}) = r_{km}$.

После вычисления вероятностей с помощью розыгрыша по жребию происходит определение, в какой узел m должна переместиться частица. При этом $t_m = 1$, номер узла добавляется в маршрут частицы. После окончания обхода вектор T обнуляется.

4. Если на j-й итерации выполнено условие остановки, то значение $X_{final}^{best} = X_j^{best}$ подается на выход O_1. Иначе происходит переход к итерации 2.

Вектор $P = \{\alpha, \beta, \gamma, \rho\}$ – коэффициенты алгоритма A. Коэффициент α определяет степень влияния количества феромона на дуге на вероятность того, что муравей выберет эту дугу. Коэффициент β определяет степень влияния веса дуги графа на вероятность ее выбора. Коэффициент γ – коэффициент интенсивности выделения феромона. Коэффициент ρ влияет на испаряемость феромона, принимает значения от 0 (нет испарения) до 1 (испаряется до минимального уровня после каждой итерации).

Идентификаторы I и O – вход и выход роя, которые не зависят от реализации алгоритмов роевого интеллекта.

2. ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ АЛГОРИТМОВ

Раздел посвящен программной реализации алгоритмов стохастической оптимизации и их применению. Особое внимание уделено использованию алгоритмов в такой области искусственного интеллекта, как машинное обучение.

2.1. Области применения

Благодаря своей гибкости и универсальности стохастические методы оптимизации широко применяются во многих областях, некоторые из которых перечислены ниже.

1. Биоинформатика. Стохастическая оптимизация используется в диагностике некоторых болезней, таких как рак, болезнь Паркинсона, в разработке медицинских препаратов, в задачах анализа генетических последовательностей.

2. Построение телекоммуникационных сетей: выбор наиболее эффективных схем различных видов сетей, оптимизация управления потоками данных в сети, разработка структур сетей GPS. Особенно эффективны в этой области алгоритмы роевого интеллекта.

3. Решение комбинаторных задач в планировании производственных и логистических процессов, таких как задача коммивояжера, транспортная задача, задача о назначениях, задачи календарного планирования, задачи раскроя и упаковки, раскраски графов, задача о клике и многие другие.

4. Управление и контроль в автоматических и автоматизированных системах: разработка контроллеров, включая ПИД-контроллеры, управление транспортными потоками, контроль работы двигателей.

5. Поиск наилучших параметров и конфигураций двигателей, летательных аппаратов, электрических сетей, сплавов и т. д.

6. Анализ изображений и видео. Распознавание лиц, идентификация людей по изображению глаза, сегментация изображений, поиск изображений, обнаружение движений, слияние изображений, распознавание символов, повышение контрастности, MPEG-оптимизация.

Среди других областей применения можно выделить энергетику, компьютерную графику и визуализацию, робототехнику, обработку

сигналов. Часто стохастические методы оптимизации применяются в системах искусственного интеллекта вместе с методами машинного обучения, о которых речь пойдет ниже.

2.2. Машинное обучение

В настоящем разделе дается очень краткое введение в теорию обучения машин и приводится пример использования стохастических методов оптимизации в качестве инструмента для машинного обучения.

Машинное обучение (Machine Learning, теория обучения машин) – один из важнейших разделов искусственного интеллекта на современном этапе развития науки. Сфера применений машинного обучения постоянно расширяется из-за распространения информационных технологий и накопления огромных объёмов данных. Наиболее тесно теория машинного обучения связана с интеллектуальным анализом данных (Data Mining).

Теория машинного обучения возникла в конце 1950 годов и развивается на стыке прикладной статистики, численных методов оптимизации, комбинаторики, дискретного анализа, вычислительной математики [2]. Сейчас машинное обучение является самостоятельным и быстроразвивающимся разделом математики и имеет большую практическую ценность. Ни один серьезный проект по распознаванию речи, изображений или видео, по созданию интеллектуальных систем управления, ни одна крупная социальная сеть или интернет-магазин не обходятся без применения машинного обучения. Так, например, появление электронной коммерции привело к возможности собирать сведения о клиентах, чтобы затем извлекать из полученных данных новые знания о предпочтениях людей. Поэтому сегодня специалисты в области машинного обучения востребованы во всех крупных IT-компаниях и во многих научных институтах.

2.2.1. Задача машинного обучения

Как правило, машинное обучение направлено на решение задачи следующего вида [2].

1. Пусть V – множество объектов, W – множество допустимых ответов и существует некоторая неизвестная зависимость между объектами и ответами w: $V \to W$, т. е. w ставит каждому объекту из V в соответствие некоторый ответ из W (например, если V – множество изоб-

ражений отдельных цифр, а W – множество {0, 1, 2, …, 9}, то w – некоторый черный ящик, определяющий цифру по изображению).

2. Для некоторого подмножества $V^* = \{v_1, v_2, \ldots, v_k\}$ известны ответы $w_i = w(v_i^*)$. Пары (v_i, w_i) называют прецедентами, а все множество пар (v_i, w_i), $i = 1, \ldots, k$ называют обучающей выборкой (training sample).

3. Необходимо построить такое отображение (функцию, алгоритм, модель) a: $V \to W$, которое было бы как можно ближе к неизвестной зависимости $w(v)$ на всем множестве объектов V (не только на обучающей выборке). При этом отображение a должно быть реализуемо на компьютере (машине) и работать автоматически, без участия человека.

2.2.2. Признаки

Каждый элемент v из множества V можно назвать объектом. Объект v имеет ряд характеристик (свойств), называемых признаками. Например, у человека есть такие признаки, как возраст, рост, вес, пол. Для изображения в формате BMP признаком будет является матрица пикселей, для слова признаком является количество букв, какая буква стоит в каждой позиции слова – тоже признак.

Выделяют следующие виды признаков:

– бинарные (0 или 1);

– номинальные (признак принимает значения из некоторого конечного множества);

– количественные (вещественные значения).

Таким образом, у объекта имеется вектор признаков $\{p_1, p_2, \ldots, p_n\}$. Тогда множество объектов V^* можно представить матрицей

$$V^* = \begin{pmatrix} p_1(v_1) & \cdots & p_n(v_1) \\ \vdots & \ddots & \vdots \\ p_1(v_k) & \cdots & p_n(v_k) \end{pmatrix}.$$

Например, ставшая классической в литературе по машинному обучению задача ирисов Фишера заключается в построении алгоритма, определяющего вид ириса (Iris setosa, Iris virginica или Iris versicolor) по четырем признакам:

– длине наружной доли околоцветника (sepal length);

– ширине наружной доли околоцветника (sepal width);

– длине внутренней доли околоцветника (petal length);

– ширине внутренней доли околоцветника (petal width).

Пример матрицы для этой задачи легко найти по запросу «классификация ирисов», или «ирисы Фишера».

Для упрощения формул часто объект и вектор его параметров отождествляют, так что $v_i = \{p_1(v_i), p_2(v_i),\ldots, p_n(v_i)\}$, в этом случае под записью $f(v_i)$ понимают функцию от всех признаков $f(p_1(v_i), p_2(v_i),\ldots, p_n(v_i))$.

2.2.3. Виды задач и примеры

Множество ответов W может иметь различный вид в зависимости от вида задачи машинного обучения.

1. Классификация непересекающихся классов. Множество $W = \{1, \ldots, m\}$, каждый объект v принадлежит одному и только одному из классов. Такие задачи называют задачами распознавания образов. К ним относятся определение символов на изображениях, распознавание жестов, определение надежности клиентов и компаний, классификация документов (в частности, фильтрация спама), определение заболевания по данным анализов.

2. Классификация пересекающихся классов. Упомянутая задача определения заболевания (диагностики) может быть отнесена и к классификации пересекающихся классов, поскольку одновременно возможно наличие нескольких болезней, в этом случае $W = \{0,1\}^m$. Задача часто возникает при диагностике оборудования, поиске причин событий, разделении документов на пересекающиеся группы.

3. Восстановление регрессии. Множество W принадлежит множеству вещественных чисел. Например, по известным параметрам дома и статистике продаж (какие дома за какие суммы были куплены) нужно определить наиболее подходящую цену данного дома. Особенно часто такая задача возникает при прогнозировании. Например, задача прогнозирования потребительского спроса на определенный товар, предсказание процесса распространения эпидемий.

Задачи классификации можно также разделить на задачи собственно классификации и задачи кластеризации. В задачах классификации заранее известны возможные классы и множество прецедентов (v_i, w_i), $i = 1, \ldots, k$. В задачах кластеризации ни по одному из объектов неизвестно, к какому классу он принадлежит, т. е. нет множества прецедентов и, как правило, неизвестно и количество классов. Кластеризация помогает выявлять социологические группы, разбивать изображения на отдельные объекты, выполнять поиск групп семантически близких документов или сообщений. Задача кластеризации стала особенно

актуальной с развитием электронной коммерции, поскольку разделение клиентов на группы позволяет создавать для каждой группы специфические стратегии продвижения товаров и услуг.

2.2.4. Методы машинного обучения и оценка качества

Существует множество методов решения задач классификации, кластеризации и восстановления регрессии. Ниже перечислены наиболее эффективные и распространенные:

– деревья решений (Decision Tree);

– метод опорных векторов (Support Vector Machine, SVM);

– метод *k* ближайших соседей (k-Nearest Neighbor, kNN);

– нейронные сети (искусственные нейронные сети, Artificial Neural Networks);

– глубокое обучение (Deep Learning), которое обычно используется для нейронных сетей;

– наивный Байесовский классификатор (Naive Bayes classifier);

– EM-алгоритм (Expectation-maximization);

– построение решающих правил (Decision rule algorithms);

– для повышения качества методов часто используют техники бустинга (boosting) и баггинга (bagging), позволяющие объединять наборы алгоритмов *a* в так называемые комитеты или ансамбли.

Чтобы построить хорошо работающий алгоритм $a: V \to W$, необходимо уметь определять качество алгоритма. Часто используют следующий функционал качества алгоритма a на выборке V^*:

$$Q(a,V^*) = -\frac{1}{k}\sum_{i=1}^{k} q(a,v_i).$$

Функция $q(a, v)$ показывает ошибку распознавания алгоритма a при анализе объекта v. Это может быть бинарная функция, принимающая значение 0, если $q(a, v) = w_i$, может быть модуль разности или квадрат разности $q(a, v)$ и w_i. Чем меньше ошибок допустит алгоритм a, тем выше его качество $Q(a, V^*)$.

2.2.5. Обучение как задача оптимизации

Алгоритм a имеет некоторую основу и ряд параметров, значения которых нужно подобрать, чтобы обеспечить наилучшее качество решения задачи. Подбор параметров и называют обучением.

В качестве простого примера можно взять задачу классификации на два непересекающихся класса, $W = \{-1, +1\}$. Рассмотрим метод линейной классификации, при котором алгоритм a можно представить в следующем виде:

$$a(v, X) = \operatorname{sign}\left(\sum_{j=1}^{n} x_j p_j(v) + x_0 \right),$$

где $X = \{x_0, x_2, \ldots, x_n\}$ – параметры алгоритма (веса признаков $p_1(v)$, $p_2(v)$, …). Таким образом, каждый признак объекта v умножается на вес признака, затем полученные произведения суммируются, и к ним прибавляется параметр x_0. Если полученная сумма больше 0, то объект v будет отнесен к классу –1, иначе +1. Пример такого классификатора показан на рис. 2.

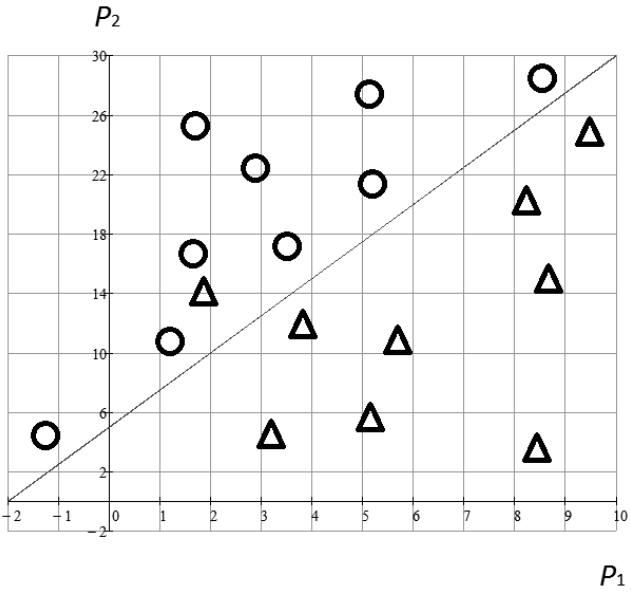

Рис. 2. Пример линейного классификатора

Объекты класса –1 показаны кружками, а класса +1 треугольниками. По осям абсцисс и ординат отложены значения признаков p_1 и p_2 соответственно. Для каждого объекта значение признаков p_1 и p_2 зада-

ют координаты его центра на плоскости P_1P_2. Линейный классификатор, показанный линией, разделяющей классы, имеет вид (x_0 = 10, x_1 = 5, x_2 = –2):

$$a(v,\ \{5,-2,10\}) = \text{sign}\,(5p_1(v) - 2p_2(v) + 10)\,.$$

Все объекты, расположенные выше черты на рис. 2, будут отнесены к классу кружков, расположенные ниже – к классу треугольников. Черту называют разделяющей плоскостью.

2.2.6. Примеры использования стохастической оптимизации для задач обучения

Для задачи, показанной на рис. 2, подобный классификатор можно построить вручную, хотя линейный классификатор рассматриваемого вида не способен дать 100 %-е качество для этой задачи. Обратите внимание, что классификатор допускает одну ошибку (треугольник с центром в точке (1.8, 14) отнесен к классу кружков.

На практике решаются проблемы на порядок сложнее и для них подобрать параметры алгоритма вручную уже невозможно. Невозможно выполнить и полный перебор всех возможных значений, так как на это уйдет слишком много времени. Так, если нужно подобрать значения 10 параметров в диапазоне от –100 до 100 с шагом 0.1, то потребуется построить и оценить примерно 1×10^{33} возможных комбинаций значений параметров. Поэтому часто для подбора параметров (обучения) используют стохастические методы оптимизации. В этом случае задача обучения алгоритма a, имеющего вектор параметров X, на выборке V^* может быть записана так:

$$X = \underset{X\in D}{\text{argmin}}\left(\frac{1}{k}\sum_{i=1}^{k} q(a(X),\ v_i)\right),$$

где D – область допустимых значений вектора X параметров алгоритма $a(X)$, а k – количество объектов в обучающей выборке V^*. Таким образом, задача обучения алгоритма заключается в минимизации ошибок, которые алгоритм допускает на обучающей выборке.

При решении задачи алгоритмом роевого интеллекта, показанной на рис. 2, каждый агент будет представлять некоторую разделяющую плоскость, а его фитнесс-функция будет равняться количеству совершенных ошибок классификации. В процессе работы алгоритма опти-

мизации агенты будут перемещаться, стремясь снизить количество ошибок, в итоге будет определена разделяющая плоскость, дающая минимум ошибок (создание приложения визуализации процесса обучения можно порекомендовать в качестве полезного учебного примера, позволяющего потренироваться и в машинном обучении, и в оптимизации, и в компьютерной графике).

Необходимо отметить, что в общем случае мера качества алгоритма зависит не только от количества ошибок, однако этот вопрос не входит в рамки рассмотрения данного пособия, как и многие другие аспекты машинного обучения (явление переобучения, разделение тренировочной выборки на собственно тренировочную и тестовую с перекрестной проверкой алгоритмов, дополнительные критерии эффективности и др.), которые можно изучить по материалам, например, К.В. Воронцова [2]. Приведенный линейный классификатор является одним из самых простых, для других алгоритмов классификации вектор параметров может задавать не только численные параметры, но и саму структуру алгоритма, например, число узлов скрытого слоя нейронной сети.

В качестве еще одного примера рассмотрим решение задачи кластеризации алгоритмом имитации отжига. При этом в качестве вектора X можно взять не параметры алгоритма кластеризации, а вектор, показывающий, к какому кластеру какой объект обучающей выборки относится. Положим, что число кластеров известно и равно m (сами кластеры обозначим 1, 2, 3, … m), тогда вектор X будет иметь длину m, а значение x_i будет определять кластер объекта v_i, $x_i \in \{1, 2, \ldots m\}$, $i = 1,\ldots m$. Критерием оптимальности может быть сумма расстояний между объектами одного кластера и сумма расстояний между объектами разных классов:

$$f(X) = \sum_{i=1}^{m} \sum_{j=1}^{k} \sum_{\substack{l=1, l \geq j \\ x_j = x_l = i}}^{k} d(v_j, v_l) - \sum_{j=1}^{k} \sum_{\substack{l=1, l \geq j \\ x_j \neq x_l}}^{k} d(v_j, v_l).$$

Формула показывает, что для каждого из кластеров определяется сумма расстояний между всеми объектами класса и из полученной величины (близость объектов одного кластера) вычитается сумма расстояний между всеми объектами, относящимися к разным кластерам (различие разных кластеров). Полученный критерий нужно минимизировать. Расстояние между объектами можно определять разными спо-

собами. Одним из наиболее популярных является евклидово расстояние

$$d(v_j,\ v_l) = \frac{1}{n}\sqrt{\sum_{h=1}^{n}(p_h(v_j) - p_h(v_l))^2}\ .$$

Конечно, не нужно каждый раз при вычислении критерия определять расстояния, достаточно сделать это один раз и сохранить расстояния между всеми объектами в матрицу.

Алгоритм имитации отжига начинает работу с генерации произвольного разбиения объектов множества V^* на классы. Для полученного разбиения, заданного вектором X, определяется значение критерия $f(X)$. Затем произвольно выбирается элемент x_i вектора X и его значение меняется на любое другое из множества $\{1, 2, \ldots, m\}$, что соответствует перемещению объекта v_i из одного кластера в другой. В результате получается вектор X_{new}, для которого тоже определяется значение критерия (для этого нужно учесть перемещение только одного объекта, не пересчитывая все выражение). Затем определяется, выполнить переход к полученному разбиению ($X = X_{new}$) или вернуться к предыдущему (см. раздел 1.3 «Алгоритм имитации отжига»). На этом итерация завершается, выполняется понижение температуры и вновь выбирается случайный элемент вектора X и его значение случайным образом меняется, и так далее, пока не будет выполнено условие завершения алгоритма. На первых итерациях, когда температура высока, объекты будут постоянно перемещаться из одного кластера в другой с небольшой тенденцией к улучшению разбиения на кластеры (поиск приблизительного решения). На последних итерациях, когда приблизительные разбиения будут получены и температура будет низкой, объекты будут перемещаться редко, но почти все перемещения будут приводить к улучшению разбиения, пока это возможно (стадия детального уточнения). Чтобы получить высокую эффективность, нужно грамотно настроить график понижения температуры и установить достаточное количество итераций. Подбор графика понижения температур выполняется исходя из экспериментов, опыта работы со стохастическими алгоритмами.

В настоящее время для применения машинного обучения на практике не требуется реализовывать алгоритм классификации и алгоритмы обучения самостоятельно. Существует множество готовых инструментов: Weka, KINME, RapidMiner, PredictionIO, различные библиоте-

ки для языков программирования R и Python, которые наиболее часто применяются для интеллектуального анализа данных. Однако для их успешного применения все же необходимо знать математические основы машинного обучения и методов оптимизации.

2.3. Рекомендации по реализации алгоритмов

Настоящий раздел рассчитан на желающих создавать свои реализации стохастических алгоритмов. Он содержит рекомендации по созданию гибкой и удобной архитектуры алгоритмов и несколько полезных приемов, повышающих быстродействие или эффективность алгоритмов

Интерфейс. Для повышения скорости реализации алгоритмов, гибкости и масштабируемости разрабатываемых программных систем необходимо избегать зависимости между алгоритмами, интерфейсами к ним и решаемыми задачами.

В рамках объектно-ориентированного программирования каждый алгоритм оптимизации можно рассматривать как класс. Полезно использовать единые интерфейсы таких классов, чтобы применение того или иного алгоритма выполнялось единообразно. В этом случае можно применить шаблон проектирования «Template method», который задает общую структуру поведения связанных классов. При этом детали поведения отдельного алгоритмов задаются в реализации дочерних классов. Общими для дочерних классов будут методы управления алгоритмами, настройки, инициализации, перемещения агентов и возвращения результатов работы. Пример базового класса на языке программирования C++ приведен ниже.

```
class Optimizator
{
public:
    virtual void start() = 0;
    virtual void stop() = 0;
    void setIterationNumber(int ni); //установка
         количества итераций
    void setAgentsNumber(int na); //установка
         количества агентов
    virtual void setParameters(double *arrParams) = 0;
        //установка параметров
    void setCriterion(ICreator *pCreator);//этот
            метод будет описан ниже
```

```
    double getBest(double *bestSolution);//возврат
             // найденного решения
             // (в bestSolution записывается вектор
             // искомых переменных, метод возвращает
             // значение критерия
    virtual ~Optimizator () {}

protected:
    virtual void initialization() = 0;
    virtual void move() = 0;

    int numberAgents;
    int numberIterations;
    double *bestSolution;
    ICreator *creator;
    //указатель на описанный ниже объект для
    //вычисления критерия
};
```

Для обеспечения единообразной работы алгоритмов оптимизации с различными задачами можно реализовать некоторый интерфейс, обозначенный здесь как *ICreator*, имеющий метод *getCriterion*, который соответствует критерию $f(X)$ и принимает управляющие переменные, а возвращает значение целевой функции. Можно разбить вектор X на целочисленную и непрерывную части. Например, на языке программирования C++ реализация такого интерфейса с помощью абстрактного класса может быть следующей:

```
class ICreator
{
public:
    virtual double getCriterion(double *arrayDouble,
                                int *arrayInt) = 0;
    virtual ~ICreator() {}
};
```

В языке Java реализация интерфейса немного короче:

```
public interface ICreator
{
    double getCriterion(double [] arrayDouble,
                        int [] arrayInt);
};
```

Затем в любом классе, представляющем модель задачи оптимизации, нужно реализовать метод *getCriterion*, а сам класс должен насле-

доваться от класса *ICreator* (в C++) или реализовать интерфейс *ICreator* (в Java), например, для задачи Химмельблау:

```
//C++
class Test : public ICreator
{
public:
    virtual double getCriterion(double *arrDouble,
                                int *arrInt);
};

double Test::getCriterion(double *arrDouble, int *arrInt)
{
    double x0 = arrDouble[0], x1 = arrDouble[1];
    return pow((x0*x0 + x1 - 11.), 2)
              + pow((x0+ x1*x1 - 7.), 2);
}

//Java
public class Test implements ICreator
{
    double getCriterion(double [] arrayDouble,
                        int [] arrayInt);
    double x0 = arrDouble[0], x1 = arrDouble[1];
    return Math.pow((x0*x0 + x1 - 11.), 2)
                     + Math.pow((x0+ x1*x1 - 7.), 2);
}
```

Сам класс, представляющий модель задачи, может быть только оболочкой для целой архитектуры, если рассматривается оптимизация сложного объекта управления. Но каким бы сложным ни была внутренняя логика расчета критерия, для алгоритма оптимизации она целиком будет сокрыта интерфейсом *ICreator*. Если в приведенном примере нужно будет заменить целевую функцию, это никак не повлияет на интерфейс *ICreator* и код алгоритмов оптимизации.

Выше в коде класса *Optimizator* есть метод *setCriterion* и указатель на объект класса *ICreator*. Объект, вычисляющий целевую функцию, передается по указателю в класс *Optimizator* через метод *setCriterion*, в этом методе данный объект связывается с указателем *creator*. В результате благодаря полиморфизму в классах, реализующих алгоритмы оптимизации, можно вызывать вычисление целевой функции через метод *getCriterion* интерфейса *ICreator* независимо от класса, в котором будет производиться это вычисление. Таким образом, код алгоритма оптимизации никак не связан с кодом класса, реализующего решаемую задачу оптимизации.

Использование параллельных вычислений. Согласно закону Амдала, параллельные вычисления тем эффективнее, чем больше доля расчетов, выполняемых параллельно:

$$S(p) \le \frac{1}{a + \frac{1-a}{p}},$$

где $S(p)$ – теоретическое ускорение, т. е. эффект от распараллеливания; p – количество используемых процессоров (ядер); a – доля операций, которые нужно выполнять последовательно.

Как правило, в рамках одной итерации процесс перемещения агентов популяционных алгоритмов легко поддается распараллеливанию, так как при этом не происходит конфликтов чтения данных и каждый процессор просто работает со своей частью популяции. Но это не всегда верно, например, в муравьином алгоритме процесс обновления феромона на ребрах графа может приводить к конфликтам потоков. Можно предложить более универсальный и часто более эффективный другой подход – каждый процессор работает со своим экземпляром алгоритма, а обмен результатами происходит лишь время от времени. В простейшем случае, если не происходит обмена данными между экземплярами алгоритмов, это равносильно последовательному запуску алгоритмов с точки зрения полученных решений задачи, а ожидаемое повышение скорости работы будет близко к количеству используемых процессоров, так как доля последовательно выполняемых операций очень мала.

Взаимодействие с пользователем. Близким к предыдущему является вопрос организации диалога с пользователем во время работы. Как правило, решение задач оптимизации большой размерности выполняется не моментально. При выполнении расчетов, занимающих заметное для пользователя время, необходимо выводить информацию о ходе решения и предоставлять возможность досрочного корректного завершения работы программы. Решение вопроса взаимодействия приложения, решающего задачу оптимизации, и пользователя имеет несколько вариантов:

– приложение никак не отвечает во время расчетов на действия пользователя и не выводит данных о ходе решения;

– приложение регулярно приостанавливает вычисления, выводит информацию и обрабатывает действия, которые успел совершить пользователь;

– вычисления и организация диалога с пользователем разделены на два параллельно работающих потока.

Очевидно, что последний вариант будет надежнее и удобнее для пользователя, чем первый, и немного эффективнее с точки зрения производительности, чем второй. Поэтому при реализации оптимизационных алгоритмов рекомендуется всегда учитывать работу в многопоточной среде и как минимум разделить алгоритм оптимизации и пользовательский интерфейс в отдельные потоки (интерфейс обычно обрабатывается основным потоком).

Выбор используемых структур данных. Общеизвестно, что в программировании очень важен правильный выбор структур данных. Рассматриваемые алгоритмы обычно просты в реализации, хотя и здесь возникает вопрос такого выбора.

Как сказано выше, нет необходимости использовать отдельный класс для реализации агентов. Можно реализовать набор агентов с помощью структур или матриц для повышения скорости работы программы и экономии памяти. Заметим, что это относится и к самой структуре данных для хранения агентов, когда необязательно применять массив или вектор. Можно использовать связный список, а для алгоритма роя пчел имеет смысл применить такую структуру данных, как пирамида, потому что в нем на каждой итерации требуется определять часть агентов с наилучшим значением фитнесс-функции.

Можно привести еще один пример. В алгоритме колонии муравьев сводится к многократному обходу графа и изменению феромона на его дугах, поскольку выбор структуры графа очень важен. На первый взгляд, может показаться, что естественной реализацией графа будет набор узлов, содержащих список дуг, каждая из которых содержит указатель на соседний узел и количественные характеристики (вес и количество феромона). Однако легко заметить, что граф можно представить в виде матрицы весов и матрицы феромона. Такой вариант будет, во-первых, проще в реализации, во-вторых, меньше по объему требуемой памяти (не нужно хранить списки указателей в каждом узле), в-третьих, скорее всего, алгоритм с таким графом будет работать быстрее.

Настройка параметров. Многие исследования показывают, что для стохастических методов необходима настройка, учитывающая особенности задач [6, 10, 15, 29]. Для выполнения такой настройки можно использовать различные метаэвристические способы адаптации. При этом один алгоритм оптимизации подбирает параметры другого алгоритма, который, в свою очередь, решает прикладную задачу оптимизации. Такой подход называют мета-оптимизацией. Для задач автоматического оперативного управления, которые требуют получения решения в реальном времени наискорейшим образом, использование адаптивных алгоритмов может быть затруднено из-за высоких затрат времени на расчеты.

В некоторых задачах адаптивные алгоритмы целесообразно применять, поскольку в этих задачах время, сэкономленное на решении, обычно намного меньше потерь времени и других ресурсов от неэффективных решений, например, в планировании работ. Но даже и для задач, которые необходимо решать быстро, полезно использовать машинное обучение, чтобы опыт решения предыдущих задач помогал эффективнее решать новые, поскольку для одного объекта управления или для одного типа задач можно ожидать достаточно близкие условия, чтобы параметры алгоритмов, настроенные по предыдущим задачам, дали хорошие результаты для новых задачах.

Учет постоянной составляющей целевой функции. Во многих алгоритмах оптимизации значение целевой функции используется не только для качественного, но и для количественного управления алгоритмом. Например, в формулах для алгоритма роя частиц $f(X)$ используется только качественно для определения лучшей позиции в пространстве поиска решений. А в формулах для алгоритма колонии муравьев от значения целевой функции пропорционально зависит количество феромона, наносимое на граф, а следовательно, и вероятности выбора ребер графа. В генетическом алгоритме от значения целевой функции агента зависит вероятность выбора этого агента для скрещивания.

Очень часто при реализации стохастических алгоритмов не обращают внимания на тот факт, что целевая функция может иметь некоторую постоянную составляющую f_{const}, значение которой намного больше, чем диапазон изменяемой части $f_{\text{var}}(X)$, т. е.

$$f(X) = f_{\text{const}} + f_{\text{var}}(X), \;\; f_{\text{const}} \gg \max_{X \in D} f_{\text{var}}(X) - \min_{X \in D} f_{\text{var}}(X).$$

В этом случае относительные различия между значениями фитнесс-функций агентов малы, например, если постоянная составляющая в 100 раз больше диапазона изменяемой части, то максимально возможная разница между фитнессами агентов составит всего 1 %.

Из двух приведенных выше абзацев следует простой вывод: если целевая функция количественно влияет на процесс работы алгоритма и ее постоянная часть намного превышает изменяемую часть, то эффективность алгоритма может очень сильно упасть. Например, для генетического алгоритма такая ситуация приведет к тому, что вероятность отбора всех агентов на этапе селекции будет примерно одинаковой независимо от качества их позиций в пространстве поиска решений. А в алгоритме колонии муравьев количество феромона, наносимое на ребра графа, будет также почти одинаковым независимо от эффективности найденного каждым агентом маршрута. Таким образом, алгоритмы лишаются одного из своих главных свойств – учета опыта, полученного на предыдущих итерациях.

Для устранения этого негативного эффекта можно использовать различные способы. В общем случае можно на первой итерации выбрать наименьшее найденное решение, принять его за примерную постоянную величину (f_{appr}) и затем вычитать его на всех итерациях из полученных значений целевой функции. Либо после окончания работы алгоритма сохранить наилучшее значение целевой функции, и при следующем запуске вычитать это значение для определения фитнессов агентов. Нужно учесть, что не все алгоритмы корректно работают с отрицательными значениями целевой функции, которые могут появляться в данном подходе, если фитнесс какого-либо агента окажется меньше величины f_{appr}.

Прочие рекомендации. При оптимизации часто большая часть вычислений связана с расчетом целевой функции $f(X)$. Целевая функция вычисляется для каждого агента популяционного алгоритма на каждой итерации, т. е. если задано 100 агентов и 1000 итераций, то целевая функция будет рассчитана 10^5 раз. Для задач коммивояжера это может быть некритично, но вычисление целевой функции может оказаться очень трудоемким процессом, например, в задачах календарного планирования, где для вычисления ЦФ необходимо составить сложное расписание, или в задачах, использующих сложные модели объектов управления, таких как система энергоснабжения или производственная линия. Поэтому нужно об-

ращать особое внимание на эффективную реализацию модели задачи оптимизации.

Иногда в задачах оптимизации можно использовать целочисленные вычисления там, где кажется необходимым применить вычисления с плавающей точкой, особенно в комбинаторных задачах. Например, если в задаче коммивояжера расстояния заданы нецелыми числами, но точность расстояний составляет два знака после запятой, то можно, умножив все расстояния на 100, перейти к целочисленным вычислениям. Однако нужно учесть, что скорость вычислений с разными типами данных зависит от множества факторов (язык программирования, компилятор, процессор).

Выбор языка программирования и вовсе является отдельной обширной темой, можно только указать, что опыт убеждает в преимуществе языка программирования C++ для реализации алгоритмов оптимизации [10, 15, 29].

3. ПРАКТИЧЕСКАЯ РАБОТА ПО СТОХАСТИЧЕСКОЙ ОПТИМИЗАЦИИ

3.1. Описание

Практическая работа по стохастическим методам дискретной оптимизации направлена на ознакомление студентов с наиболее распространенными алгоритмами, которые часто применяются для решения практических оптимизационных задач и могут в дальнейшем быть использованы студентами в их научно-исследовательских работах. Данная работа не предполагает самостоятельную реализацию алгоритмов оптимизации. Студентам предлагается использование готового программного продукта «Система визуализации стохастических алгоритмов оптимизации» (свидетельство о регистрации программы для ЭВМ № 2015613847, автор Матренин П.В., 2015 г.) для визуального анализа работы алгоритмов и проведения экспериментов.

В ходе работы студенты изучают четыре стохастических алгоритма дискретной оптимизации: генетический алгоритм, алгоритм имитации отжига и два алгоритма роевого интеллекта (алгоритм роя пчел и алгоритм роя частиц).

Приведенные далее указания содержат описание работы с программным обеспечением, пошаговую инструкцию проведения экспериментов, самостоятельные задания и контрольные вопросы для защиты. Перед выполнением работы необходимо изучить теоретическую часть (см. в разделе «Описание алгоритмов»).

Объем работы в часах и уточненное задание на работу определяются преподавателем, ориентировочно работа рассчитана на четыре часа. За отведенное время студенты должны выполнить и защитить работу. К работе студенты приступают после домашней подготовки и предварительного изучения теории. После выполнения работы студент предъявляет оформленный отчет с указанными ниже результатами. Допустима работа в группах по 2–4 человека, защита индивидуальная.

3.2. Описание программного обеспечения визуализации

Для визуализации алгоритма имитации отжига, генетического алгоритма и алгоритмов роя частиц и пчел в двумерном пространстве, ознакомления пользователей и демонстрации влияния параметров алгоритма на процесс поиска решений были созданы четыре соответствующих приложения. Приложения имеют типизированный интерфейс, написаны на языке C++ в среде Qt и могут быть использованы в операционных системах семейств Windows, Linux и Mac OS. Для наглядности было решено использовать задачи вида

$$\begin{cases} z = f(x, y), \\ x_{\min} \le x \le x_{\max}, \\ y_{\min} \le y \le y_{\max}, \\ z \to \max, \end{cases}$$

для которых процесс поиска можно представить как перемещение текущих решений на плоскости *XY*.

Общее представление об интерфейсе приложений (рис. 3 для Windows, рис. 4 для Linux) приведено для алгоритма роя частиц.

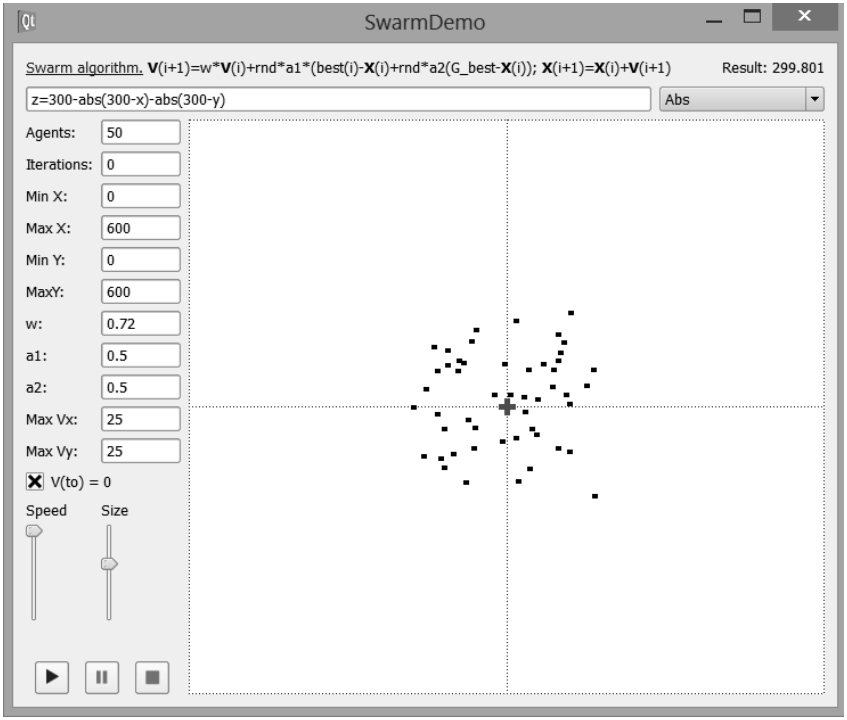

Рис. 3. Интерфейс приложения визуализации Windows

Сверху расположена основная расчетная формула, ссылка на файл справки и метка для вывода результата. Ниже находится поле для ввода задачи, справа от которого расположен элемент для выбора функций списка. Большую часть формы занимает поле для визуализации процесса работы алгоритма, слева расположены элементы для ввода параметров. Текущее лучшее решение обозначается на поле красным крестиком.

Для запуска алгоритма необходимо выбрать задачу из раскрывающегося списка сверху справа или ввести свою, нажать кнопку «Старт» внизу слева, пауза позволяет поменять параметры алгоритма и продолжить работу. Кнопка «Стоп» прерывает работу. Если в поле «Iterations» ввести 0, то алгоритм будет работать до нажатия кнопки «Стоп», иначе алгоритм выполнит заданное число итераций, если его не прервать.

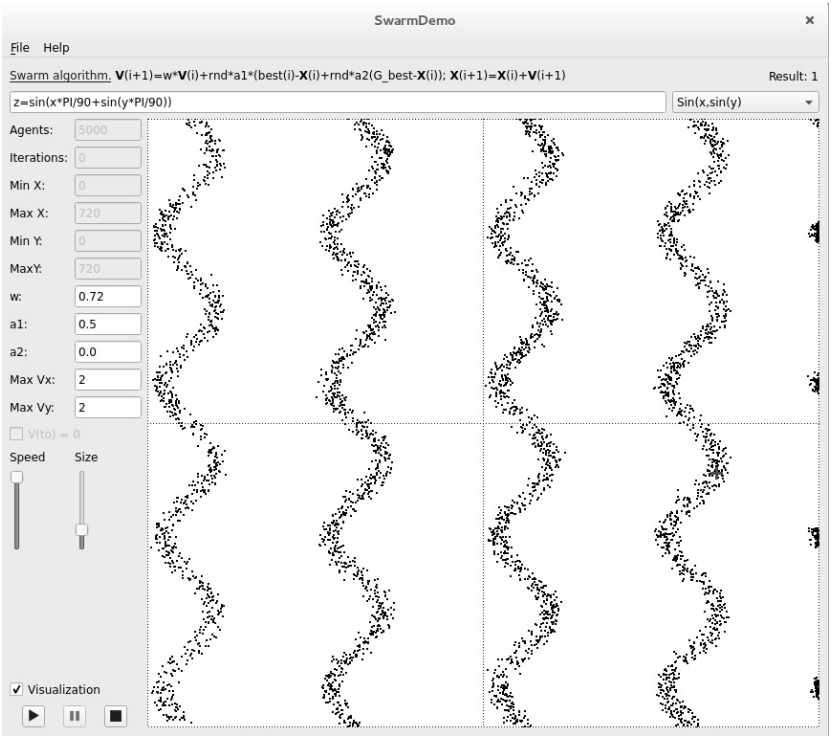

Рис. 4. Интерфейс приложения визуализации Linux

Пользователь может вводить свои функции *z(x,y)* на языке Qt Script (стандарт ECMAScript, он же JavaScript 2.0). Кроме арифметических действий доступны указанные в табл. 1 математические функции и константы (указывать перед ними имя объекта Math не нужно). Кроме того, можно использовать условные операторы (if-else), циклы (for, while).

Заданные пользователем задачи обрабатываются медленнее встроенных тестовых задач из-за необходимости интерпретации введенного пользователем кода.

Таблица 1

Константы и функции, доступные для использования в веденных задачах

Обозначение	Описание
E	Число *e* (константа Эйлера)
LN2	Натуральный логарифм числа 2
LN10	Натуральный логарифм числа 10
LOG2E	Логарифм числа *e* по основанию 2
LOG10E	Логарифм числа *e* по основанию 10
PI	Число π
SQRT1_2	Квадратный корень из 1/2
SQRT2	Квадратный корень из числа 2
abs(a)	Модуль числа
ceil(a)	Округление в большую сторону
floor(a)	Округление в меньшую сторону
round(a)	Округление до ближайшего целого
max(a,b,c,…)	Максимальный из переданных аргументов
min(a, b, c,…)	Минимальный из переданных аргументов
pow(a, n)	Возведение в указанную степень
sqrt(a)	Квадратный корень
random()	Генератор случайных чисел от 0 до 1
sin(a)	Синус
cos(a)	Косинус
tan(a)	Тангенс
asin(a)	Арксинус
acos(a)	Арккосинус
atan(a)	Арктангенс
log(a)	Натуральный логарифм

Каждый шаг алгоритма запускается встроенным в программу таймером, интервалы отсчетов которого можно менять, таким образом, можно настраивать скорость работы алгоритмов. Кроме того, использование таймера позволяет приложению оперативно реагировать на все действия пользователя, что дает возможность менять параметры алгоритма, скорость работы, размеры окна приложения, размеры точек на плоскости решения и даже решаемую задачу прямо во время работы алгоритма или во время паузы.

3.3. Практическое задание

3.3.1. Начало работы

Прочитайте описание четырех стохастических алгоритмов, приведенное выше и описание программного обеспечения для визуализации этих алгоритмов. Если вы вначале прочитаете и поймете основные теоретические положения, выполнение работы будет намного проще и быстрее.

Запустите SwarmDemo.exe, убедитесь, что вам понятно, как работать с приложением. Проверьте на любых задачах, соответствует ли показанный процесс работы алгоритма вашему представлению, сложившемуся после прочтения его описания. Аналогично запустите AnnealingDemo, BeesDemo, GeneticDemo и понаблюдайте за работой алгоритмов. При наведении на элементы для ввода параметров появляется всплывающая подсказка, если назначение каких-либо элементов вам непонятно, обратитесь к преподавателю.

Для алгоритма роя частиц на любой из тестовых задач определите и укажите в отчете, при каком порядке количества частиц (агентов) и максимальной скорости отображения возникают задержки между итерациями более 0.5 с.

Далее по каждому из алгоритмов описаны шаги, которые необходимо выполнить для более подробной демонстрации принципов работы алгоритмов. Для каждого из алгоритмов сделайте и вставьте в отчет 2–3 снимка экрана приложения, на которых будут отражены различные этапы работы. Чтобы понимать процесс работы алгоритмов, необходимо представлять себе структуры решаемых задач, для этого нужно построить поверхность $z(x, y)$ с помощью любого удобного вам средства (наиболее просто это сделать, скопировав формулу из строки редактирования в верхней части экрана и вставив ее в строку поиска «Google», затем поставить нужные диапазоны по всем осям). Некоторые поверхности показаны на рис. 5–9, но статичные 2D изображения не так наглядны, как интерактивные псевдо-3D.

Частично процесс работы алгоритма поясняется, однако для полного понимания работы часть объяснений пропущена, чтобы вы могли проконтролировать, насколько хорошо понимаете данные алгоритмы. В описании заданий по каждому алгоритму есть выделенные указания о том, что включить в отчет, и вопросы, ответы, на которые нужно указать в отчете (достаточно кратких ответов в одно предложение).

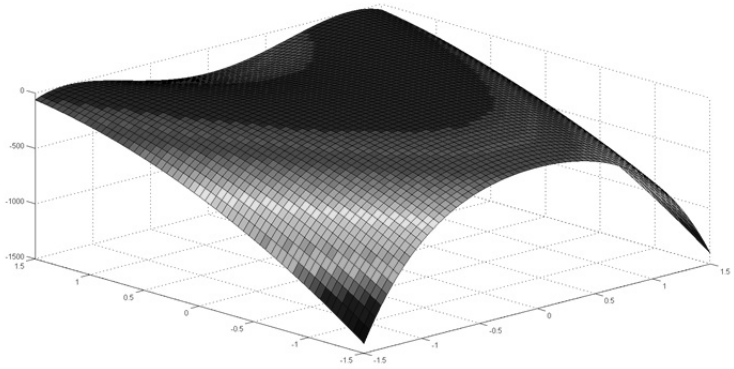

Рис. 5. Поверхность к задаче Розенброка

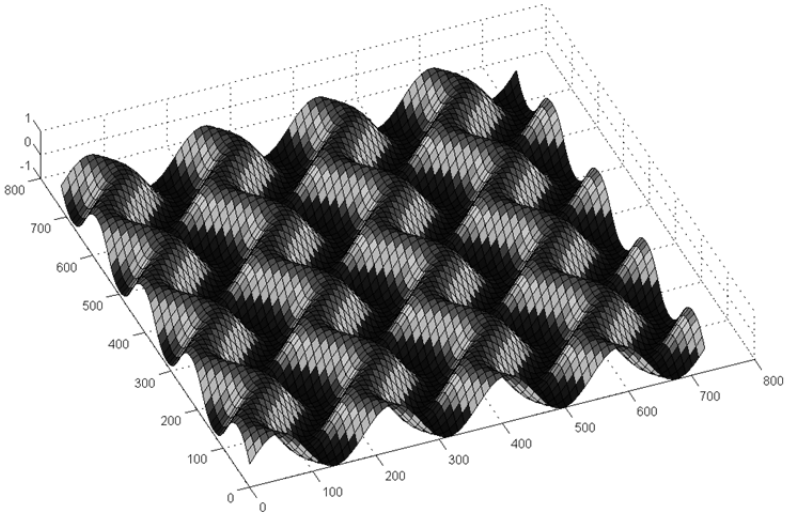

Рис. 6. Поверхность к задаче Sin(*x*, Sin(*y*))

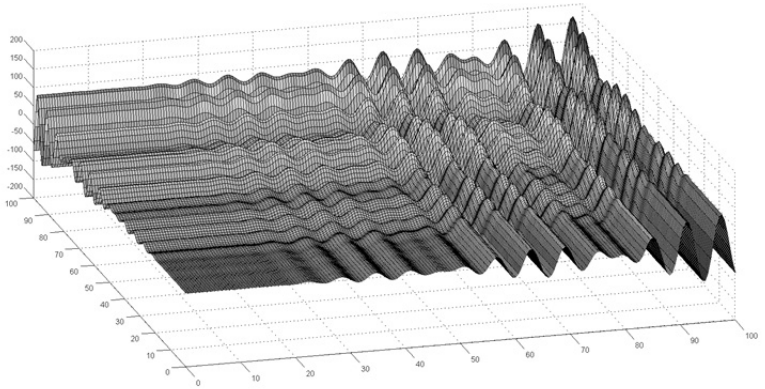

Рис. 7. Поверхность к задаче Ex_180.422

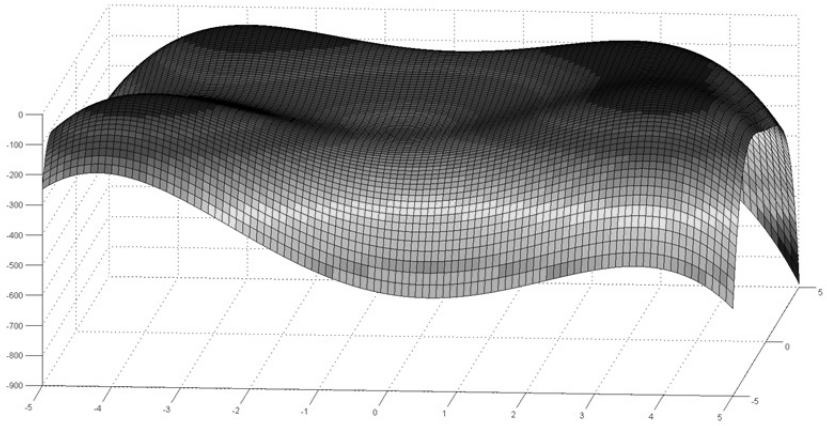

Рис. 8. Поверхность к задаче Химмельблау

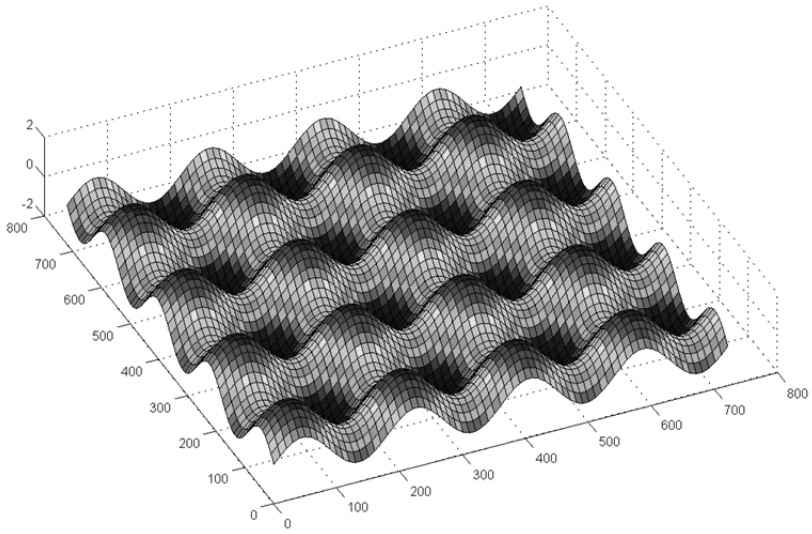

Рис. 9. Поверхность к задаче Sin(*x*, *Y*)

3.3.2. Демонстрация алгоритма роя частиц

Запустите SwarmDemo, выберите задачу Розенброка (см. рис. 5), поставьте 500 частиц, коэффициенты *w*, *a1*, *a2*, *Max Vx*, *Max Vy*, равными 1.1, 0.1, 0.0, 0.01, 0.01 соответственно. Флаг «V(to) = 0» снимите. Запустите алгоритм, частицы должны образовать дугу, после этого поставьте на паузу. Так как коэффициент *a2*, отвечающий за взаимодействие между частицами, равен нулю, каждая частица действует независимо, поэтому многие из них находятся не в оптимальной позиции. Увеличьте коэффициент *w* до 1.3 и запустите алгоритм – дуга станет шире. Затем верните коэффициенту *w* предыдущее значение, а коэффициент *a1* сделайте равным 0.3, и продолжите работу алгоритма. Каждая частица после этого будет стремиться быть ближе к своему локальному оптимуму. Последним шагом поставьте значение *a2*, равное 1. Теперь частицы используют обмен своим опытом и многие частицы быстро переместятся в окрестность глобального оптимума функции, другие же образуют отдельно расположенную группу, из которой будут постепенно по одной-две переходить в окрестность глобального оптимума. **Объясните в отчете, почему не все частицы**

сразу переместились к глобальному оптимальному решению? После этого прекратите работу алгоритма.

Выберите задачу «sin (x, sin(y))» (см. рис. 6). Число частиц нужно сделать равным 5000. Поставьте коэффициенты равными 1.2, 0.3, 0, 1 и 1 и запустите алгоритм. Затем поставьте *a2* равным 0.3 и посмотрите, как изменится форма образованных частицами синусоид. Видно, что несмотря на сближение частиц, форма синусоид в искаженном виде сохранилась. Постепенно повышайте коэффициент *a2* (с шагом 0.5) и посмотрите, как рой будет все больше сжиматься.

Последняя демонстрационная задача для роя частиц называется «Ex_180.422» (см. рис. 7). Запустите на ней алгоритм с коэффициентами 1, 1, 1, 1 и 1, 50 агентами и 1000 итераций (флаг «V(to) = 0» установите). Сделайте 10 запусков и **запишите в отчет полученные значения *z*.** Они будут сильно отличаться друг от друга и от оптимального значения 180.422. Можете повысить число итераций, но это не приведет к существенному улучшению. В таких случаях говорят о преждевременной сходимости или «застревании» алгоритма поиска в локальном экстремуме. Рой нашел локальный экстремум, и при данных параметрах вероятность выйти из него очень мала (но не равна 0). Для решения этой трудности есть два пути: увеличить число агентов в надежде, что какой-либо из них успеет попасть в окрестность глобального оптимума до застраивания алгоритма, либо регулировать параметры алгоритма, чтобы он мог выходить из таких ситуаций.

Попробуйте поставить число агентов 1000. Теперь даже при числе итераций меньше 500 алгоритм будет находить близкие к оптимальному решения, **выпишите 10 результатов**. Но такой подход не всегда является применимым, поскольку при увеличении числа частиц возрастает вычислительная трудоемкость, особенно для сложных задач, в которых вычисление целевой функции может требовать длительного времени. Если для задачи размерности 2 потребовалась 1000 частиц, то для задач размерности выше 100 количество частиц для подобного охвата всего пространства решений окажется слишком велико.

Теперь поставьте всего 25 частиц и 1000 итераций, но увеличьте максимальные скорости до 10, а коэффициент *a2* сделайте равным 0.5. Это позволит частицам, во-первых, с большей вероятностью «выскакивать» из локальных экстремумов благодаря возможности набрать высокую скорость, а во-вторых, каждая частица имеет больше индивидуальности и частицы менее стремятся собраться вместе (*a1* в два раза больше, чем *a2).* С такими коэффициентами будут получаться резуль-

таты, близкие к оптимальному (**укажите в отчете 10 результатов**), намного ближе, чем в первом варианте и намного быстрее, чем во втором (при визуализации скорость расчетов не очень заметна, поскольку этап рисования занимает значительное время, поэтому **оцените и укажите в отчете повышение скорости** работы третьего варианта относительно второго исходя из предпосылки, что большую часть времени работы алгоритма занимало бы вычисление целевой функции). Но это не означает, что повышение скорости всегда повышает эффективность алгоритма, потому что при слишком высокой скорости частицы могут «проскакивать» мимо экстремумов.

3.3.3. Демонстрация алгоритма роя пчел

Запустите BeesDemo и откройте задачу Химмельблау (см. рис. 8) и запустите алгоритм с параметрами по умолчанию. Вы не увидите какой-либо логики в перемещениях агентов. Дело в соотношении размеров пространства поиска и параметров *RadX* и *RadY*. Когда пчела отправляется на участок, она может отклониться от него на случайную величину, равномерно распределенную от *–RadX* до *RadX* по оси *X*, и от *–RadY* до *RadY* по оси *Y*, в данном случае размер области, в которую может попасть пчела, превышает размер области поиска, поэтому движение пчел носит характер, близкий к случайному поиску. Уменьшите во время паузы *RadX* и *RadY* до 0.1 и продолжите работу алгоритма. Вы увидите, что теперь можно наблюдать небольшие группы пчел. Поставьте количество пчел, посылаемых на лучшие и выбранные участки, равными 50 и 10 (*CN* и *CM*), в этом случае в группах будет больше пчел, а размер групп позволит отличать лучшие участки от выбранных. Если вы посмотрите на график функции Химмельблау, то увидите, что функция имеет четыре экстремума, а групп пчел сейчас больше четырех. Поставьте количество лучших участков 3, а выбранных 1 (*N* и *M*) и запустите алгоритм. Вы увидите три больше группы и одну небольшую (возможно, для этого придется снизить скорость). Обратите внимание, что распределение групп по окрестностям экстремумов не статично – в любом из экстремумов может быть то большая группа, то небольшая. **Объясните, почему так произошло, и почему часть пчел продолжают хаотичные перемещения по всему пространству поиска.**

Следующая задача – Розенброка (см. рис. 5). При тех же параметрах, которые были в последнем этапе на предыдущей задаче, вновь

можно видеть четыре группы пчел, хотя график этой функции имеет только один экстремум. Разделение по группам связано на этот раз с коэффициентами *dX* и *dY*, влияющими на расстояние между центрами участков. Измерьте примерное расстояние между центрами групп по осям *X* и *Y*. Попробуйте подобрать величины *dX* и *dY*, чтобы между группами пчел не было заметно промежутков, **укажите эти величины в отчете и напишите, можно ли было сразу определить их значения**, не проводя экспериментов, исходя только из параметров алгоритма и условий задачи.

Последнее задание для алгоритма роя пчел связано с задачей «Sin(*x*, sin(*y*))» (см. рис. 6). Запустите алгоритм с параметрами по умолчанию (для этого можно перезапустить BeesDemo). Вы не увидите синусоид, как это было для алгоритма роя частиц. Попробуйте подобрать параметры алгоритма так, чтобы синусоиды были видны. Подсказка: потребуется менять и число агентов, но слишком много ставить не обязательно (*N***CN* + *M***CM* <=1000). **Вставьте снимок экрана и значения параметров в отчет.** Задача с синусоидами не имеет прямого отношения к оптимизации, но и алгоритмы роевого интеллекта (рой частиц и рой пчел) используются не только для решения задач оптимизации. Алгоритмы роевого интеллекта часто применяются для интеллектуального управления группами роботов, для визуализации в играх и фильмах движений большого количества объектов и других задач, требующих гибкого децентрализованного управления множеством элементов. Задача «Sin (*x*, sin(*y*))» как раз демонстрирует, как возникает коллективное поведение агентов роя и как можно влиять на него.

3.3.4. Демонстрация алгоритма имитации отжига

Запустите AnnealingDemo. Откройте задачу «Sin(*x*, *y*)» (см. рис. 9, не «Sin(*x*, sin(*y*))»), запустите алгоритм на не самой высокой скорости с 500 агентами, параметрами *M*[*x*] и *M*[*y*], равными 2, *To* – 10000, *Tf* – 1, *vTo* – 0.01. Понаблюдайте, как точки будут постепенно сходиться в экстремумы. При начальных значениях температуры вероятность перехода агента в новое состояние, несмотря на его качество, очень высока, поэтому точки как будто хаотично перемещаются по пространству решений, что напоминает колебания частиц в кристаллической решетке при высоких температурах. При уменьшении температуры точки будут оказываться все ближе к экстремумам, продолжая совершать колебания, т. е. их движение не будет на каждом шаге направле-

но к экстремумам. При некотором уровне температуры на экране будут четко видны все экстремумы функции, но заметные колебания точек не прекратятся до конца работы алгоритма. Алгоритм не предполагает использование нескольких агентов, в данном случае 100 агентов используется только для визуализации, они никак не взаимодействуют между собой. Используя аналогию с процессом кристаллизации вещества, на которой основан алгоритм имитации отжига, можно сделать вывод, что в алгоритме должно быть задействовано много агентов, как частиц в веществе, однако классический алгоритм имитации отжига и большинство его модификаций не используют многоагентный (его еще называют «популяционный») подход. А запуск алгоритма имитации отжига с n агентами равен параллельной работе n независимых алгоритмов или последовательному запуску алгоритма n раз. Вы должны понимать, что для трех других приведенных алгоритмов это категорически не так, они относятся к популяционным алгоритмам, в которых ключевой особенностью является взаимодействие агентов.

Количество итераций алгоритма имитации отжига можно задавать явно, а можно через остановку алгоритма при опускании температуры ниже некоторой отметки. **Выведите и запишите в отчет зависимость** числа шагов алгоритма от начала до завершения от его параметров (формула понижения температуры приведена в описании алгоритма выше).

Откройте задачу Химмельблау (см. рис. 8) и запустите алгоритм с параметрами, которые использовали до этого, и одним агентом. Запустите алгоритм пять раз (скорость можно вернуть на максимальную) и **запишите результаты**. При работе алгоритма не будет заметно сужения области, в которой производится поиск, как это было для предыдущего примера, а результаты не очень близки к нулю. Поставьте конечную температуру в 100 раз ниже и повторите пять запусков, **выписывая результаты**. Время работы алгоритма увеличилось, но результаты остались не очень хорошими. Теперь верните значение конечной температуры, равное 1, а коэффициенты $M[x]$ и $M[y]$ поставьте 0.1. Снова выполните пять запусков и **запишите результаты**. Эффективность алгоритма значительно повысилась. Снижение коэффициентов, влияющих на максимальное расстояние, на которое может переместиться поиск из текущего состояния, позволило алгоритму больше времени потратить на «исследование» окрестности экстремумам и найти точки, которые ближе к нему. При высоких значениях этих коэффициентов алгоритм перемещался в среднем на расстояние в 20 раз

больше, таким образом, поиск «рассеивался» слишком далеко вокруг экстремума. Однако высокие значения коэффициентов *M[x]* и *M[y]* могут быть полезны для других задач, в которых алгоритму сложнее попасть в окрестность экстремума.

Поставьте значение параметра T_0, равное 1000, T_f, равное 0.01 и откройте задачу Розенброка (см. рис. 5). Запустите алгоритм 10 раз и **укажите полученные результаты в отчете**. Теперь поставьте значения коэффициентов *M[x]* и *M[y]* 0.01 и снова запустите алгоритм 10 раз, **выписывая результаты**. Посмотрите на график задачи Розенброка и сопоставьте его с показанным процессом поиска решения, после этого **напишите, какой из вариантов параметров и почему** дал более эффективные результаты по среднему значению, по лучшему значению.

3.3.5. Демонстрация генетического алгоритма

Запустите GeneticDemo. Параметры с метками «Pm x» и «Pm y» определяют вероятность мутации в хромосоме (мутации гена, представляющего значение *x*, и гена, представляющего значение *y*). Параметры с метками «Vm x» и «Vm y» задают максимально возможное отклонение значения соответствующего гена при мутации, например, значение 0.1 означает, что ген может измениться на величину от 0 до 10 % в большую или меньшую сторону с равной вероятностью (конечно, с учетом того, что значение гена не может выйти из допустимого диапазона). При значении 0 ген при мутации может принять любое допустимое значение с равной вероятностью.

Откройте задачу Химмельблау (см. рис. 8) и запустите алгоритм на невысокой скорости с 1000 агентов (хромосом), вероятностью мутации 1% (0.01), значения коэффициентов *VmX* и *VmY* оставьте равными нулю. Вы должны увидеть четыре центра, вокруг которых собираются особи, по одному в каждой четверти пространства поиска. Поставьте алгоритм на паузу, уменьшите вероятности мутаций в 10 раз и продолжите работу – через некоторое время точек в пространстве поиска станет намного меньше, а если снова приостановить алгоритм и поставить вероятности мутаций 20 %, точек снова станет много и при этом уже не будут заметны четыре отдельные группы особей. **Объясните, что произошло при уменьшении и увеличении вероятностей мутации в отчете**.

Следующая демонстрационная задача – $\sin(x, \sin(y))$ (см. рис. 6). Для алгоритмов роя частиц, пчел и отжига на этой задаче можно было

увидеть, как агенты образуют синусоиды. Поставьте 1000 особей и вероятности мутации 5 % и посмотрите, образуют ли особи синусоиды на этот раз? **Укажите в отчете, почему.**

Выберите задачу Розенброка (см. рис. 5), поставьте 100 особей, 1000 итераций, вероятности мутации 5 %. Выполните пять запусков алгоритма и **запишите в отчет полученные результаты**. Затем повторите пять запусков с параметрами *VmX* и *VmY*, равными 0.1 (т. е. при мутации ген не может измениться более чем на 10 %), также **указав результаты в отчете**. Ограничение на степень мутации приведет к быстрому сокращению разнообразия популяции и снижению средней эффективности алгоритма (популяцией называют набор особей или хромосом на текущей итерации алгоритма). Однако в других типах задач ограничение на степень мутации может повысить эффективность. Убедитесь в этом на задаче «Abs», для этого выполните по пять запусков с теми же параметрами, что в двух прошлых вариантах (100 особей, вероятности 5 %, *Vm* =0 в первом варианте и *Vm* = 0.1 во втором), а чтобы сделать это быстрее, число итераций сократите до 100, ведь задача является очень простой. **Укажите полученные результаты двух вариантов в отчете.** С большой вероятностью эффективность второго варианта окажется выше, поскольку в этой задаче всего один экстремум и ограничение на изменение генов приводит к концентрации особей вокруг него, окрестность экстремума обрабатывается большим количеством особей, это повышает вероятность попадания особей во все более близкие к оптимуму позиции, в то время как отсутствие ограничений на мутацию постоянно переводило бы часть особей в произвольные точки пространства поиска. Трудность заключается в том, что, как правило, заранее структура пространства решений неизвестна либо нет времени на исследование условий задачи, поэтому нельзя сказать, какие параметры алгоритма будут эффективнее. Для разрешения этого затруднения используются различные способы автоматической адаптации стохастических алгоритмов под условия задач.

3.3.6. Индивидуальные задания

Заключительная часть работы предполагает проведение самостоятельных экспериментов с указанными для каждого варианта задачами и алгоритмами (табл. 2).

В каждом варианте необходимо провести некоторое число простых экспериментов, т. е. запусков алгоритма на указанной задаче, меняя

параметры. Число агентов и число итераций нужно выбрать самостоятельно, но произведение числа агентов на число итераций должно быть равно 50 000. Нужно выбрать число агентов и итераций, зафиксировать и провести 10 экспериментов с разными значениями параметров, стараясь добиться как можно лучших значений целевой функции. Каждый эксперимент состоит в запуске алгоритма три раза с фиксированными параметрами и определении лучшего по трем запускам. Затем нужно поменять число агентов хотя бы в четыре раза в большую или меньшую сторону, определить число итераций, чтобы произведение осталось равным 50 000, и провести еще 10 аналогичных экспериментов (по три запуска) с теми же параметрами, что использовались в 10 предыдущих экспериментах.

! Скорость проведения экспериментов можно в разы повысить, если отключить визуализацию работы алгоритмов (снять галочку “Visualization”).

Результаты (количество агентов, итераций, параметры, значение целевой функции) нужно занести в таблицу. Для алгоритма имитации отжига можно сэкономить время, если не по три раза запускать алгоритм с одним агентом и выбирать лучший, а поставить трех агентов и запустить один раз. Объяснения полученных экспериментальных результатов в отчет вносить не требуется, но для защиты работы нужно подготовить выводы из экспериментов.

Таблица 2

Варианты заданий

Вариант	Задача	Алгоритм	Число агентов
1	Гринвока	Роя частиц	*Agents*
2	Гринвока	Роя пчел	*N*CN + M*CM + S*
3	Гринвока	Имитации отжига	1 (один)
4	Гринвока	Генетический	*Agents*
5	Растригина	Роя частиц	*Agents*
6	Растригина	Роя пчел	*N*CN + M*CM + S*
7	Растригина	Имитации отжига	1 (один)
8	Растригина	Генетический	*Agents*
9	Ex_180.422	Роя частиц	*Agents*
10	Ex_180.422	Роя пчел	*N*CN + M*CM + S*
11	Ex_180.422	Имитации отжига	1 (один)
12	Ex_180.422	Генетический	*Agents*

Контрольные вопросы

1. Объясните, какие алгоритмы оптимизации относятся к стохастическим, почему они так называются?
2. В чем особенности стохастических алгоритмов, их преимущества и недостатки?
3. Какие классы стохастических алгоритмов можно выделить?
4. Почему даже для задач оптимизации, которые можно решить точными методами, часто применяют стохастические алгоритмы?
5. В чем преимущества и недостатки жадных эвристик?
6. Приведите примеры практических задач, в которых имеет смысл применять стохастические методы оптимизации.
7. Кратко опишите, как работает метод случайного поиска и поиска с возвратом?
8. Кратко опишите работу алгоритма имитации отжига.
9. Кратко опишите работу генетического алгоритма.
10. Кратко опишите работу алгоритма роя частиц.
11. Кратко опишите работу алгоритма роя пчел.
12. Кратко опишите работу алгоритма поиска косяком рыб.
13. Кратно опишите работу алгоритма колонии муравьев.
14. Как алгоритмы роевого интеллекта используют случайный поиск?
15. Как генетический алгоритм используем случайный поиск?
16. Как алгоритм имитации отжига использует случайный поиск?
17. Как вы думаете, является ли усложнение правил работы агентов популяционных алгоритмов хорошим способом повышения их эффективности?
18. Какие алгоритмы оптимизации относятся к популяционным, в чем их основное отличие?
19. Почему генетический алгоритм не относится к алгоритмам роевого интеллекта, как это проявляется при визуальном наблюдении за его работой?
20. В чем преимущества и недостатки алгоритма имитации отжига, использующего одного агента по сравнению с многоагентными алгоритмами?
21. С какой целью авторы алгоритма роя пчел ввели в него агентов-разведчиков? Есть ли недостатки у такого приема?
22. Каким образом инерция влияет на поведение алгоритма роя частиц, как вы думаете, зачем введено это свойство?

23. Почему при нулевой начальной скорости и нулевом коэффициенте a_1 рой частиц остается неподвижен?

24. Как бы изменилась эффективность алгоритма имитации отжига, если бы температура была постоянной?

25. Как вы думаете, при каком условии равняется единице вероятность обнаружения глобального экстремума произвольной задачи оптимизации с помощью любого из перечисленных алгоритмов?

26. Как изменится работа алгоритма имитации отжига при использовании линейного графика понижения температуры вместо экспоненциального?

27. В чем роль скрещивания в генетическом алгоритме?

28. Стоит ли всегда оставлять в популяции генетического алгоритма особь (хромосому) с наилучшей приспособленностью?

29. Как вы думаете, какой из рассмотренных в работе алгоритмов эффективнее для решения задач оптимизации с динамически меняющимися во времени условиями? Какой наименее эффективен?

30. Что означают слова «преждевременная сходимость»?

31. Оцените рассмотренные алгоритмы с точки зрения простоты и эффективности распараллеливания.

32. Каким образом можно использовать генетический алгоритм для решения комбинаторных задач оптимизации? Запишите структуру хромосомы для задачи коммивояжера.

33. Каким образом можно использовать алгоритм имитации отжига для комбинаторных задач? Запишите структуру состояния системы для задачи коммивояжера.

34. Каким образом можно использовать алгоритм роя частиц для комбинаторных задач? Запишите вид позиции частицы для этой задачи коммивояжера.

35. Каким образом можно использовать алгоритм роя пчел для комбинаторных задач? Запишите вид позиции пчелы для задачи коммивояжера.

36. Укажите важные моменты, которые нужно учитывать при реализации стохастических алгоритмов оптимизации.

37. Каким образом можно реализовать стохастические алгоритмы оптимизации, чтобы обеспечить гибкость и переносимость их программного кода?

38. Сравните два любых описанных в пособии алгоритма с точки зрения эффективности и простоты их распараллеливания.

39. Какие существуют способы повышения скорости работы стохастических алгоритмов?

40. Объясните понятие «машинное обучение».

41. Напишите формальную постановку задачи машинного обучения.

42. Какие виды задач машинного обучения вы знаете?

43. Покажите связь между задачей машинного обучения и задачей оптимизации.

44. Как можно использовать стохастические методы оптимизации для обучения линейного классификатора?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Алгоритмы: построение и анализ: пер. с англ. / Т. Кормен [и др.]. – 2-е изд. – М. : Издательский дом «Вильяме», 2005. – 1296 с. : ил.

2. *Воронцов К.В.* Машинное обучение: курс лекций [Электронный ресурс] // http://www.MachineLearning.ru.

3. *Гладков Л.А.* Генетические алгоритмы/ Л.А. Гладков, В.В. Курейчик, В.М. Курейчик; под ред. В.М. Курейчика. – 2-е изд., испр. и доп. – М.: ФИЗМАТЛИТ, 2006. – 320 с.

4. *Гриф М.Г.* Гибридная экспертная система проектирования человеко-машинных систем и принятия решений ИНТЕЛЛЕКТ-3: учеб. пособие. – Новосибирск: Изд-во НГТУ, 2007. – 162 с.

5. *Гэри М.* Вычислительные машины и труднорешаемые задачи: пер. с англ. / М Гэри, Д. Джонсон. – М.: Мир, 1982. – 416 с.

6. *Карпенко А.П.* Популяционные алгоритмы глобальной оптимизации. Обзор новых и малоизвестных алгоритмов // Приложение к журналу «Информационные технологии». – 2012. – № 7. – С. 1–32.

7. *Карпов В.Э.* Методологические проблемы эволюционных вычислений // Искусственный интеллект и принятие решений. – № 4. – 2012. – С. 95–102.

8. *Кочетов Ю.А.* Вероятностные методы локального поиска для задач дискретной оптимизации. Дискретная математика и ее приложения: сб. лекций молодежных и научных школ по дискретной математике и ее приложениям. – М.: Изд-во центра прикладных исследований при механико-математическом факультете МГУ, 2000. – С. 87–117.

9. *Курейчик В.М.* Использование роевого интеллекта в решении NP-трудных задач/ В.М. Курейчик, А.А. Кажаров // Известия ЮФУ. Технические науки. – 2011. – № 7 (120). – С. 30–37.

10. *Матренин П.В., Секаев В.Г.* Оптимизация адаптивного алгоритма муравьиной колонии на примере задачи календарного планирования // Программная инженерия. – 2013. – № 4. – С. 34–40.

11. *Матренин П.В. Секаев В.Г.* Системное описание алгоритмов роевого интеллекта // Программная инженерия. – 2013. – № 12. – С. 39–45.

12. *Матренин П.В.* Описание и реализация алгоритмов роевого интеллекта с использованием системного подхода // Программная инженерия. – 2015. – № 3. – С. 27–34.

13. *Матренин П.В.* Разработка приложений для визуализации методов стохастической оптимизации // Сборник научных трудов SWorld. Материалы международной научно-практической конференции «Современные проблемы и пути их решения в науке, транспорте, производстве и образовании'2012». – Вып. 4. Том 12. – Одесса: КУПРИЕНКО, 2012. ЦИТ:412-0099. – С. 28–35.

14. *Панченко Т.В.* Генетические алгоритмы: учеб.-метод. пособие / Т.В. Панченко; под ред. Ю.Ю. Тарасевича. – Астрахань: Издательский дом «Астраханский университет», 2007. – 87 с.

15. *Скиена С.* Алгоритмы. Руководство по разработке: пер. с англ. / Стивен Скиена. – 2-е изд. – СПб.: БХВ-Петербург, 2013. – 720 с.: ил.

16. *Штовба С.Д.* Муравьиные алгоритмы/ С.Д. Штовба // Exponenta Pro. Математика в приложениях. – 2003. – № 4. – С. 70–75.

17. *Andrew Ng.* Artificial intelligence | machine learning [Электронный ресурс] // Stanford engineering everywhere. URL: http://see.stanford.edu/see/lecturelist.aspx?coll=348ca38a-3a6d-4052-937dcb017338d7b1.

18. *Beni G.* From Swarm Intelligence to Swarm Robotics, Swarm Robotics. Lecture Notes in Computer Science, 2005, vol. 3342, pp. 1-9.

19. *Bertsimas D*. Simulated Annealing / D. Bertsimas, J. Tsitsiklis // Statistical Science, Vol.8, No 1, 1993. P. 10-15.

20. *Dorigo M.* The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B. 1996. V. 26. No. 1. P 29-41.

21. *Dorigo M., Birattari M., Stutzle T.* Ant Colony Optimization. Artificial Ants as a Computational Intelligence Technique // IRIDIA. Technical Report Series. Technical Report No. TR/IRIDIA/2006-023. Brussels, Belgium. 2006. 14 p.

22. *Glover F.* Future Paths for Integer Programming and Links to Artificial Intelligence / F. Glover // Computers and Operations Research. 13 (5), 1986. P. 533–549.

23. *Glover F.* Tabu search / F. Glover, M. Laguna // Vol. 22. Boston: Kluwer academic publishers, 1997.

24. *Holland J.H.* Adaptation in natural and artificial systems / J.H. Holland // University of Michigan Press, Ann Arbor, 1975.

25. *Karaboga D.* An idea based on honey bee swarm for numerical optimization [Электронный ресурс] // Technical report TR06. Erciyes University, Engineering Faculty, Computer Engineering Department. 2005. URL: http://mf.erciyes.edu.tr/ abc/pub/tr06_2005.pdf.

26. *Kennedy J.* Particle Swarm Optimization / J. Kennedy, R. C. Eberhart // Proc. of IEEE International Conference on Neural Network, Piscataway, NJ. 1995 P. 1942–1948.

27. *Kirkpatrick S.* (1983). Optimization by Simulated Annealing / S. Kirkpatrick, Jr. Gelatt, M.P. Vecchi // Science 220 (4598): 671–680.

28. *Pham D.T., Ghanbarzadeh A., Koc E., Otri S., Rahim S., Zaidi M.* The Bees Algorithm – A Novel Tool for Complex Optimisation Problems [Электронный ресурс] //Technical Note. Manufacturing Engineering Centre. Cardiff University. UK. 2005. URL: https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/Pham06%20-%20The%20Bee%20Algorithm.pdf.

29. *Pedersen M.* Simplifying Particle Swarm Optimization / M. Pedersen, A. Chipperfield // School of Engineering Sciences, University of Southampton, UK. Applied Soft Computing, 2009. P. 618–628.

30. *Poli R.* An Analysis of Publications on Particle Swarm Optimisation Applications. Department of Computer Science [Электронный ресурс] / R. Poli // University of Essex. Technical Report CSM-469. May 2007.

31. *Sahin E.* Swarm robotics: From sources of inspiration to domains of application, Swarm Robotics. Lecture Notes in Computer Science, 2005, vol. 3342, pp. 10–20.

32. *Shi Y.* A Modified Particle Swarm Optimizer / Y. Shi, R. Eberhart // Department of Electrical Engineering Indiana University Purdue University Indianapolis. Indianapolis, IN 46202-5160.

33. *Wolpert D.H.* No Free Lunch Theorems for Optimization / D.H. Wolpert, W.G. Macready // IEEE Transactions on Evolutionary Computation 1, 1997. P. 67–82.

Матренин Павел Викторович
Гриф Михаил Геннадьевич
Секаев Виктор Гилячевич

МЕТОДЫ СТОХАСТИЧЕСКОЙ ОПТИМИЗАЦИИ

Учебное пособие

Редактор *Л.Н. Ветчакова*
Выпускающий редактор *И.П. Брованова*
Дизайн обложки *А.В. Ладыжская*
Компьютерная верстка *С.И. Ткачева*

Налоговая льгота – Общероссийский классификатор продукции
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

Подписано в печать 14.03.2016. Формат 60 × 84 1/16. Бумага офсетная. Тираж 100 экз.
Уч.-изд. л. 3,95. Печ. л. 4,25. Изд. № 345/15. Заказ № 401. Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20