О. Н. Родзина

Проблемно-ориентированные алгоритмы мягких вычислений

О. Н. Родзина

ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ АЛГОРИТМЫ МЯГКИХ ВЫЧИСЛЕНИЙ

Монография

Чебоксары
Издательский дом «Среда»
2020

УДК 004.8+519.7
ББК 32.813+22.18
Р60

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-01-00412

Рецензенты:
д-р техн. наук, профессор, заместитель директора по науке Санкт-Петербургского филиала ФГБУН «Институт земного магнетизма, ионосферы и распространения радиоволн им. Н. В. Пушкова» РАН
Анатолий Григорьевич Коробейников;
д-р техн. наук, профессор, директор центра интеллектуальных и информационных технологий Ростовского филиала АО «Научно-исследовательский и проектно-конструкторский институт информатизации, автоматизации и связи на железнодорожном транспорте»
Сергей Михайлович Ковалев

Родзина О. Н.
Р60 **Проблемно-ориентированные алгоритмы мягких вычислений** : монография / О. Н. Родзина. – Чебоксары: ИД «Среда», 2020. – 96 с.

ISBN 978-5-907313-55-2

В книге рассматриваются проблемно-ориентированные алгоритмы мягких вычислений. Представлены алгоритмы нечеткой кластеризации графических образов, мягких вычислений для представления графических образов, популяционный муравьиный алгоритм транспортной логистики, популяционный масштабируемый алгоритм для задач многомерной оптимизации многоэкстремальных функций. Рассмотрены подходы, связанные с параллелизмом мягких вычислений, параллельной кластерной обработкой продукционных правил в базах знаний интеллектуальных систем, а также представлен нейробиостохастический алгоритм для задачи балансирования тележки с двумя флагштоками разной длины.

Монография адресована магистрантам и аспирантам, изучающим интеллектуальные информационные системы и технологии, а также специалистам по теоретическим основам информатики, программной инженерии и обработке информации.

DOI 10.31483/a-200
ISBN 978-5-907313-55-2

Оглавление

Введение

В информационных технологиях понятие «мягкие вычисления» объединяет в общий класс неточные, приближенные и математически строго не обоснованные проблемно-ориентированные алгоритмы решения задач, для которых не существует подходов, позволяющих получить точный результат за приемлемое время. Примером здесь являются такие NP-полные задачи, как задача коммивояжёра, проблема Штейнера, проблема раскраски графа, задача о покрытии множества, задача о клике. Область мягких вычислений включает такие технологии, как нечеткая логика, нейрокомпьютинг, а также алгоритмы, инспирированные природными процессами.

Термин «мягкие вычисления» (*soft computing*) ввел в 1994 г. основоположник теории нечетких множеств, профессор Лотфи Заде [1]. Мягкие вычисления, как новый метод вычислительной математики, оказался более эффективным, нежели традиционные методы, в целом ряде проблемных областей, таких как биология, медицина, конвергентные науки, робастное управление, менеджмент. Особую роль мягкие вычисления играют в компьютерных реализациях искусственного интеллекта и поддержки принятия решений, проблемно-ориентированных системах, основанных на знаниях, в системах распознавания, классификации, кластеризации и прогнозирования.

По словам Л. Заде, основополагающим принципом мягких вычислений является: «терпимость к неточности, неопределенности и частичной истинности для достижения удобства манипулирования, робастности, низкой стоимости решения и лучшего согласия с реальностью».

Исходной моделью для мягких вычислений служит человеческое мышление. Отметим также сходство алгоритмов мягких вычислений с биологическими процессами. В отличие от классических «жестких» вычислений, основанных на формально-логических подходах или методах численного анализа, мягкие вычисления не требуют точности и единственности решения, а также допускают некоторую погрешность и неопределенность для конкретной задачи [2]. Иными словами, в традиционных «жестких» вычислениях основными целями являются строгость, определенность и точность результата. Для достижения этих целей необходимо опи-

сать все элементы вычислительного процесса. Иногда это в принципе невозможно или требует немалых ресурсных и финансовых затрат. В то время как при организации «мягких» вычислений, основной задачей является обеспечение толерантности их результата с учетом НЕ-факторов (неточность, неопределенность) обрабатываемых информационных объектов [3; 4].

Мягкие вычисления широко используются в машинном обучении. Например, с помощью эволюционных вычислений [5] исследуется пространство поиска, синтезируются решения, являющиеся точками этого пространства, запрашивается оценка их качества или «приспособленности», которая затем используется для осуществления «естественного отбора». Тем самым биоэвристики обучаются тому, какие области пространства поиска содержат наилучшие решения.

Ниже в таблице представлены сильные и слабые стороны технологий мягких вычислений.

Таблица

Сильные и слабые стороны технологий мягких вычислений

Мягкие вычисления *Свойство*	**Нечеткая логика**	**Нейросети**	**Биоэвристики**
Обучение по прецедентам	нет	да	да
Параллельная обработка данных	нет	да	да
Поддержка немаркированных данных	нет	да	да
Вычислительная сложность	низкая	высокая	средняя
Неполная информация	да	нет	нет
Поддержка лингвистической информации	да	нет	нет

Нечеткий логический вывод включает в себя четыре этапа: фаззификацию, оценку правил, агрегирование правил и дефаззификацию [6; 7]. Нечеткий вывод имеет низкую вычислительную сложность, способен работать в условиях неполной информации, поддерживает лингвистическую информацию, однако имеет ограниченные возможности параллельной обработки данных.

Биоэвристики включают различные виды эволюционных вычислений, а также муравьиные и пчелиные алгоритмы, разнообразные алгоритмы, моделирующие роевое и стайное поведение [8]. Они не гарантируют поиск оптимальных решений, имеют среднюю вычислительную сложность, во многом зависят от правильной

настройка параметров, способа кодирования информации и используемых операторов.

Нейросеть представляет собой систему соединённых и взаимодействующих между собой простых искусственных нейронов [2]. Нейросети способны обучаться на примерах, обладают возможностями параллельной обработки данных для выполнения сложных задач.

При создании гибридных технологий необходимо выяснить, какая из составляющих частей мягких вычислений или какая их комбинация наилучшим образом подходит для решения задачи. В частности, ключевыми аспектами гибридизации биоэвристик с нейросетями являются [9]:

- оптимизация межнейронных взаимодействий;
- оптимизация архитектуры нейросети;
- параллелизм выполнения оптимизации архитектуры и весов связей между нейронами;
- синтез для нейросети обучающей выборки с помощью биоэвристик;
- оптимизация параметров обучающей выборки;
- инициализация алгоритма биоэвристики с помощью нейросети;
- использование нейросети для прогноза целевой функции биоинспирированного алгоритма.

Проблемы, связанные с гибридизацией нечеткой логики и биоэвристик, возникли относительно недавно. Большинство работ в этой области условно можно разделить на две группы: применение теории нечетких множеств в биоинспирированных алгоритмах и применение биоинспирированных алгоритмов в нечетких системах.

Исследования в области применения теории нечетких множеств в биоинспирированных алгоритмах, в основном, сосредоточены на двух направлениях:

- оптимизация параметров биоинспирированных алгоритмов с помощью нечеткого регулятора на основе экспертных знаний;
- нечеткий кроссинговер для биоинспирированных алгоритмов.

Среди работ в области применения биоинспирированных алгоритмов в нечетких системах отметим следующие направления исследований [9]:

- задачи нечеткой многокритериальной оптимизации на основе фитнесс-функций;

• применение биоинспирированных алгоритмов для задачи нечеткого линейного программирования;

• оптимизация архитектуры нечетких нейросетей;

• оптимизация структуры и параметров нечетких систем на основе биоинспирированных алгоритмов.

В настоящее время в литературе представлено множество алгоритмов мягких вычислений, популярность которых также связана с необходимостью параллельной обработки данных, с задачами оптимизации, с возможностями создания интеллектуальных систем [7]. Потребность развития мягких вычислений и их последующего применения заявлена во многих областях: в телекоммуникациях, в финансовых компаниях, в бизнесе и торговле, логистике, здравоохранении, а также в государственном управлении. Однако возникают и новые направления, связанные с разработкой проблемно-ориентированных алгоритмов в областях знаний, имеющих дело с моделированием технических, естественных или социально-экономических систем. В частности, содержанием данной книги являются проблемно-ориентированные алгоритмы мягких вычислений.

Книга состоит из трех глав.

В первой главе представлены проблемно-ориентированные алгоритмы анализа графических образов: алгоритм нечеткой кластеризации графических образов и алгоритм мягких вычислений для представления графических образов. Нечёткую кластеризацию от просто кластеризации отличает то, что объекты, которые подвергаются кластеризации, относятся к конкретному кластеру не однозначно, как это бывает при обычной кластеризации, а с некоторой степенью принадлежности. Ключевым вопросом компьютерного анализа графических образов является алгоритм кластеризации. Эффективность алгоритмов кластеризации зависит от применяемой функции расстояния и формы кластеров. В качестве примера работы алгоритма мягких вычислений для представления графических образов рассмотрен механизм ассоциативной презентации рабочего кабинета с целью определения его наилучшей конфигурации.

Вторая глава посвящена траекторным и популяционным проблемно-ориентированным алгоритмам. Проанализированы особенности поиска оптимальных решений траекторными биоэвристиками: принципы поиска, модели и характеристические свойства

ландшафта целевой функции. Если множество допустимых решений велико, то траекторные алгоритмы зачастую позволяют найти близкие к оптимальным решения с меньшим вычислительными затратами, нежели многие другие алгоритмы. Это перспективный подход к решению многих оптимизационных проблем. Альтернативой траекторным являются популяционные биоэвристики. В качестве примера рассматривается популяционный муравьиный алгоритм транспортной логистики и его визуализация на географической карте. Поставленной задаче соответствует одна из разновидностей задачи коммивояжера. В задачах с нелинейным, многомерным, многоэкстремальным пространством поиска популяционные биоэвристики выполняют более исчерпывающее исследование пространства поиска по сравнению с детерминированными и траекторными методами поиска. Однако встает вопрос об их масштабируемости. Предлагается подход к решению проблемы высокой размерности задачи при разумной поддержке разнообразия популяции. Отличительными особенностями предлагаемой биоэвристики является построение иерархических субпопуляций и выполнение когнитивного оператора мутации. В качестве примера использовался набор многомерных функций-бенчмарок Гриванка, Растригина, Розенброка и Швефеля. Показатели предлагаемого алгоритма сравнивались с показателями следующих конкурирующих алгоритмов: стандартным эволюционным алгоритм, самоорганизующимся эволюционным алгоритмом, клеточным эволюционным алгоритмом и эволюционным алгоритмом с управляемым разнообразием популяции.

Третья глава посвящена вопросам параллелизма мягких вычислений и разработке нейроэволюционного алгоритма. Параллелизация биоэвристик в гетерогенных компьютерах сетях или на параллельных высокопроизводительных кластерных вычислителях может обеспечить значительные выгоды с точки зрения производительности и масштабируемости. Поэтому актуальной является проблема их переноса в распределенную вычислительную среду. Однако требуется особая организация алгоритма путём его логического разбиения на ортогональные по отношению к обрабатываемым данным гранулы (зерна) параллелизма с минимизацией числа и объемов обменов между процессорами. Рассмотрены различные

модели и способы организации параллелизма: глобальный параллелизм, миграционная модель, диффузионная модель. Получена оценка верхней границы для максимального ускорения, которое может быть достигнуто на многопроцессорном компьютере, если объём вычислений биоэвристики равномерно распределён между процессорами. В главе также рассмотрены возможности организации параллельных вычислений и параллельной обработки продукционных правил в базах знаний на примере использования кластерного вычислителя. Представлен нейробиостохастический алгоритм балансирования тележки с двумя флагштоками разной длины.

Монография наряду с материалами обзорного характера включает в себя авторские разработки проблемно-ориентированных алгоритмов мягких вычислений, полученные с 2015 года по настоящее время.

ГЛАВА 1. ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ АЛГОРИТМЫ АНАЛИЗА ГРАФИЧЕСКИХ ОБРАЗОВ

1.1. Алгоритм нечеткой кластеризации графических образов

Нечёткую кластеризацию от просто кластеризации отличает то, что объекты, которые подвергаются кластеризации, относятся к конкретному кластеру не однозначно, как это бывает при обычной кластеризации, а с некоторой степенью принадлежности [10; 11]. Сфера прикладных задач нечеткого кластерного анализа простирается от проблем классификации и аппроксимации данных до распознавания, например, геометрических контуров графических образов в ходе их анализа и обработки [12; 13].

Пусть, например, задан черно-белый графический объект, состоящий из 20×20 точек (массив данных $D:=N \times N$, где N – множество натуральных чисел от 1 до 20). Графический объект представляется через подмножество $X \subseteq D$ белых точек.

Если, скажем, нас интересует яркость картины, измеряемая по шкале $E:=\{h|h \in [0,1]\}$, то оцениваемой функцией будет функция

$$b\colon A(D,E) \to R,$$

где R – множество вещественных чисел, $A(D,E)$ – пространство поиска данных, причем

$$A(D,E):=\{f|f\colon X \to K, X \subseteq D, X \neq \emptyset, K \in E\},$$

где K – кластерное множество. В этом случае

$$f\colon X \to \{h\} \text{ и } f \to |X|/400 - h.$$

Тогда задача анализа множества данных $X \subseteq D$ состоит в поиске такого h, при котором

$$b(f) = 0 \text{ и } f = h.$$

Если бы нас интересовало местоположение точки p, которая расположена на минимальном суммарном евклидовом расстоянии до всех остальных белых точек, то

$$min\, b : A(D,E) \to R, f \to \sum_{x \in X} \|x - p\|^2 \quad .$$

Если же целью поиска являются c кругов с указанием формы круга $(x,y,r) \in R$, где *x, y, r* – координаты центра и радиус круга, то результатом будет кластеризация данных на c подмножеств.

Теперь представим себе, что целью является поиск, скажем, двух пересекающихся кругов. Возникает вопрос, к какому кластеру

относится точка, лежащая на пересечении? Решение этой проблемы достигается путем применения нечеткой логики [14]. Под нечетким подмножеством множества *X* будем понимать отображение

$$\mu: X \to [0,1].$$

Множество всех нечетких подмножеств *X* будем обозначать через $F(x) := \{\mu | \mu: X \to [0,1]\}$ и говорить о фазировании степени принадлежности данных к тому или иному кластеру. Принято различать вероятностную и возможностную кластеризацию.

Пусть $AF(D,E)$ – нечеткое пространство поиска. Тогда отображение

$$f: X \to F(K) \in AF(D,E)$$

называется *вероятностной* кластеризацией, если

$$\forall x \in X: \sum_{k \in K} f(x,k) = 1 \text{ и } \sum_{x \in X} f(x,k) \rangle 0, \forall k \in K,$$

где $f(x,k)$ – степень принадлежности $x \in X$ к кластеру $k \in K$ по отношению ко всем другим кластерам.

Пусть $AF(D,E)$ – нечеткое пространство поиска. Тогда отображение

$$f: X \to F(K)$$

называется *возможностной* кластеризацией, если

$$\forall k \in K: \sum_{x \in X} f(x,k) \rangle 0,$$

где $f(x,k)$ интерпретируется как степень возможной принадлежности $x \in X$ кластеру $k \in K$.

Ключевым вопросом компьютерного анализа графических образов является алгоритм кластеризации. Базовым в этой области считается следующий алгоритм альтернативной оптимизации. Пусть $X = \{x_1, \dots, x_n\}$ – множество данных, $K = \{k_1, \dots, k_c\}$ – множество кластеров. Тогда результаты анализа можно представить в виде (с $\times n$) матрицы $U = \|u_{ij}\|$, где $u_{ij} = f(x_j, k_i)$, причем

$$f: X \to F(K) \text{ и } f \in AF(D,E)$$

является целевой минимизируемой функцией.

Псевдокод алгоритма включает следующие шаги.

Шаг 1. Выбор числа кластеров c, $2 \le c \le n$.

Шаг 2. Выбор размерности *m* пространства R^m.

Шаг 3. Выбор условия окончания поиска ε.

Шаг 4. Инициализация $U^{(0)}$, $i := 0$.

REPEAT

Увеличиваем *i* на 1.

Определяем $K^{(i)}$ так, чтобы оценочная функция *b* для $K^{(i)}$ и $U^{(i-1)}$ была минимальной.

Определяем $U^{(i)}$ с учетом того, что

$$f(x,k) \mapsto \begin{cases} \dfrac{1}{\sum_{j \in K} \dfrac{d^2(x,k)^{1/(m-1)}}{d^2(x,j)}} & \text{для} \quad I_x = \{j \in K | d(x,j) = 0\}, \\ \sum f(x,i) = 1 & \text{для} \quad I_x \neq 0, k \in I_x, \\ 0 & \text{для} \quad I_{x\neq} \neq 0, k \notin I_x \end{cases}$$

где d – функция расстояния

$$UNTIL \left\| U^{(i-1)} - U^{(i)} \right\| \leq \varepsilon.$$

В некоторых прикладных задачах анализа и распознавания число кластеров c заранее неизвестно. В этом случае либо указывают верхнюю границу c_{max} и сравнивают результаты кластеризации, полученные по алгоритму для $c = 2,\ldots, c_{max}$, либо вначале устанавливают функцию качества для отдельных кластеров, затем определяют c_{max} и вновь проводят процедуру сравнительного анализа результатов кластеризации.

Эффективность алгоритмов кластеризации зависит от применяемой функции расстояния и формы кластеров. На практике чаще всего используют для кластеризации евклидову метрику. Что касается формы кластеров, то базовый алгоритм и его многочисленные модификации больше подходят для описания кластеров с гладкими формами типа круга, эллипса, параболы или гиперболы. Особое значение имеют прямоугольные кластерные формы.

Существуют разные способы представления прямоугольника. Один из них состоит в представлении прямоугольника в виде четырех попарно параллельных прямых. Тогда функция расстояния

$$d^2 : \langle (x_0, x_1), (v_0, v_1, r_0, r_1) \rangle \rightarrow (\prod_{s=0}^{1} (x_s - v_s)^2 - r_s)^2,$$

где v_0, v_1 – координаты центра прямоугольника, r_0, r_1 – половины длин его ребер. Если требуется минимизировать оценочную функцию b при кластеризации $X \rightarrow F(K)$, где $K = \{k_1, \ldots, k_c\} \in E$ с заданной функцией принадлежности $f(x_j, r_i) = u_{ij}$ и $k_i = (v_{i,0}, v_{i,1}, r_{i,,0}, r_{i,1})$, то для этого необходимо выполнение следующей системы уравнений:

$$\sum_{j=1}^{n} u_{i,j}^{m} \left((x_{j,0} - v_{i,0})^2 - r_{i,0} \right) \left((x_{j,1} - v_{i,1})^2 - r_{i,1} \right)^2 (x_{j,0} - v_{i,0}) = 0,$$

$$\sum_{j=1}^{n} u_{i,j}^{m} \left(\left(x_{j,0} - v_{i,0} \right)^2 - r_{i,0} \right)^2 \left(\left(x_{j,1} - v_{i,1} \right)^2 - r_{i,1} \right) \left(x_{j,1} - v_{i,0} \right) = 0,$$

$$\sum_{j=1}^{n} u_{i,j}^{m} \left(\left(x_{j,0} - v_{i,0} \right)^2 - r_{i,0} \right) \left(\left(x_{j,1} - v_{i,1} \right)^2 - r_{i,1} \right)^2 = 0,$$

$$\sum_{j=1}^{n} u_{i,j}^{m} \left(\left(x_{j,0} - v_{i,0} \right)^2 - r_{i,0} \right)^2 \left(\left(x_{j,1} - v_{i,1} \right)^2 - r_{i,1} \right) = 0.$$

Пусть прямоугольник имеет вид, представленный на рисунке 1.1.

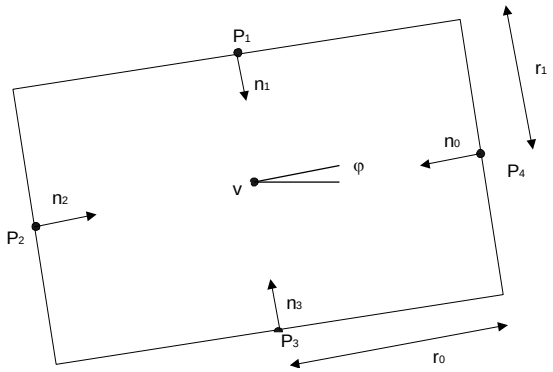


Рис. 1.1. Прямоугольная кластерная форма

Тогда, если P – это некоторая точка, лежащая на одной из сторон прямоугольника, а n – перпендикуляр единичной длины, направленный к центру v, то множество всех точек, лежащих на сторонах прямоугольника, представляет собой множество

$$\{(x - p)^T n = 0\},$$

где T – знак операции транспонирования. Угол φ указывает на величину поворота прямоугольника относительно оси абсцисс, следовательно, вектор $n_0 = -n_2 = (-cos\,\phi\,, -sin\,\phi)$, $n_1 = -n_3 = (sin\,\phi\,, -cos\,\phi)$. Таким образом, все точки P_s, лежащие внутри прямоугольника, будут иметь положительное евклидово расстояние до его центра, а все точки вне прямоугольника – отрицательное. Все это позволяет эффективно проводить кластеризацию прямоугольных графических форм. Моделирование показывает, что время распознавания оказывается на порядок меньше, нежели распознавание с помощью базового алгоритма.

1.2. Алгоритм мягких вычислений для представления графических образов

Реальные задачи обработки графической информации сопровождаются множеством технических деталей, которые трудно учесть в «жестком» алгоритме [15].

Чаще всего при обработке графической информации используются так называемые решетки. Каждая точка решетки соответствует точке некоторого объекта и является переменной, характеризуемой пикселем, как элементом двумерного растрового изображения. Между соседними точками существует сильная зависимость, которая хорошо описывается стохастической марковской моделью. Это обстоятельство можно использовать для устранения случайных ошибок при пиксельном представлении графического объекта, а также при доопределении неполного графического образа с минимальной вероятностью ошибки [16].

Алгоритм мягких ассоциативных вычислений характеризуется функцией следующего вида:

$$Y = f(X) + E \qquad (1.1)$$

где X – входной вектор, Y – выходной вектор, E – возможная ошибка (помеха, шум). Векторы X, Y могут являться количественными переменными (длина, координата), либо некоторой степенью качества (например, «объект является компьютером»). Если X и Y кодируются различным образом, то говорят о гетерогенной ассоциации. При одинаковой кодировке X и Y речь идет об автоассоциации, которая применяется для корректировки ошибок или доопределении образа X.

Ассоциативный вывод можно использовать для анализа сцен с несколькими объектами, причем пространственное расположение объектов определяется не по детерминированным правилам, а через «мягкое» сходство [17]. Для того чтобы использовать ассоциативный вывод (1.1) необходимо кодировать вектор X. Выбор способа кодирования влияет на результат ассоциативного поиска, в особенности, если речь идет о таком трудном вопросе, как описание геометрической формы графического объекта и взаимное расположение объектов на сцене друг относительно друга. К этому можно добавить возможные передвижения и перестановки объектов.

Традиционные методы представления полигонных ситуаций используют несколько уровней абстракции, на каждом из которых любой признак, характеризующий объект, кодируется пикселем. Обработка информации упрощается, если каждому пикселю входного объекта поставить в соответствие такие свойства как, например размер, цвет, форма, плотность, яркость. Каждое свойство является переменной, характеризующей степень его принадлежности входному объекту. Таким образом, любое выбранное свойство объекта образует некоторый уровень абстракции (решетку), характеризующий объект, состоящий из пикселей. Объединение решеток дает представление об объекте в целом.

Для того, чтобы сложный объект можно было ассоциировать с простыми структурными признаками, каждый пиксел должен быть представлен отдельной ассоциативной сетью. Примером реализации подобного рода подхода является неокогнитрон. Его особенностью является то обстоятельство, что между упомянутыми выше решетками существует промежуточный (рецепторный) слой. Это делает сеть нечувствительной к небольшим деформациям.

Рассмотрим пример ассоциативной презентации рабочего кабинета. Механизм презентации должен удовлетворять следующим свойствам:

• одинаковое представление нескольких объектов;

• одинаковая ассоциативная зависимость при представлении объектов различной величины;

• представление должно быть динамичным, поскольку расположение объектов в кабинете зависит от свойств соседних объектов с учетом их позиционирования, например, рабочий стол должен быть максимально освещен светом из окна, что требует введения дополнительной переменной «степень освещенности» данного места.

Величина растра определяется по следующим критериям:

• каждый пиксел должен характеризовать один объект;

• один рецепторный слой пикселов должен охватывать все важные аспекты локальной взаимозависимости объектов.

Выберем растр с квадратными пикселями, поскольку предполагается, что кабинет имеет прямоугольную форму, а все элементы обстановки кабинета также является прямоугольными, располагаются параллельно стенам кабинета, имеют одну из четырех ориентаций (север, запад, юг, восток).

Пусть имеется девять типов ориентированных в пространстве объектов: дверь, часть стены, часть окна, угол стены, кресло, часть книжного шкафа, часть письменного стола, компьютер, лампа. Для каждого из них в ходе ассоциативного вывода устанавливается степень принадлежности пикселя объекту некоторого из перечисленных типов. Свободное пространство кабинета будем считать еще одним типом объекта, но без ориентации его в пространстве. Каждый пиксел характеризуется 37 переменными.

Результатом работы алгоритма должен быть общий план обстановки рабочего кабинета.

В качестве демонстрационного примера возьмем план (вид сверху) кабинета, представленный на рисунке 1.2.

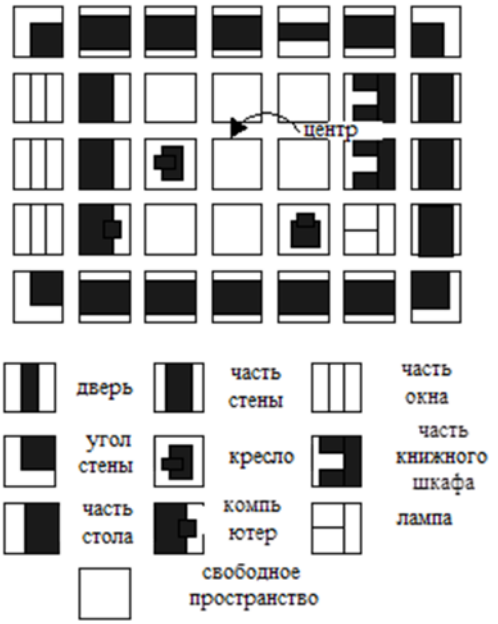


Рис. 1.2. Кодирование плана рабочего кабинета с помощью растра пикселей

Общее число крупных пикселей равно 5×7=35, общее число переменных равно 35×37=1295. Если местоположение (*g, h*) пикселя неизвестно, то необходимо определить вероятность $P(X_{g,h}=i)$ принадлежности *i*-переменной пикселю (*g, h*) исходя из имеющейся

информации о местоположении соседних пикселей. Полигон пикселей можно представить стохастической моделью Маркова, которая полностью определяется распределением условных вероятностей:

$$P(X_{g,h}|X_{m,n}; (m,n) \in N_{g,h}) \qquad (1.2)$$

где множество $N_{g,h}$ образует пиксели соседние (g, h) пикселю. При решении задачи доопределения графического образа должны быть также заданы определенные ограничения и дополнительные условия: размеры полигона, положение окон и дверей и т.п.

Целью алгоритма является определение наилучшей конфигурации, при которой вероятность $P(X_{g,h}=i)$ будет равна или близка 1.

Итак, пусть задана решетка случайных переменных $X_{g,h}$, $g = 1, 2, \ldots, G$, $h = 1, 2, \ldots, H$. Каждая переменная $X_{g,h}$ может принимать одно из k дискретных значений $i = 1, 2, \ldots, k$, которые отображают различные объекты пикселя (g, h).Отметим, что одинаковые типы объектов, имеющие различную ориентацию, соответствуют различным значениям i. Будем считать, что вид объекта в позиции (g, h) определяется исключительно конфигурацией соседних пикселей. Отсюда следует, что распределение условных вероятностей (1.2) является динамическим и не зависит от координат (g, h). Например, множество соседей в окрестности (3×3) равно

$$N_{g,h} = \{(m,n) \neq (g,h) | m \in \{g-1, g, g+1\}, n \in \{h-1, h, h+1\}\} \quad (1.3)$$

Общее распределение случайных переменных $X_{g,h}$ характеризуется условным распределением (1.2) относительно некоторого соседства. Отношение соседства является симметричным:

$$(g,h) \in N_{m,n} \leftrightarrow (m,n) \in N_{g,h}$$

Если считать, что все вероятности больше нуля, то условное распределение представляется следующей экспоненциальной формой:

$$P(x_{1,1}, \ldots, x_{G,H}) = exp(-U(x_{1,1}, \ldots, x_{G,H})/T)/W \quad (1.4)$$

$$U(x_{1,1}, \ldots, x_{G,H}) = \sum_{d \in D} V_d(x_{1,1}, \ldots, x_{G,H}) \qquad (1.5)$$

где W – нормирующий коэффициент (сумма вероятностей нормируется к единице), D – множество всех подмножеств пикселей (g, h), которые находятся в отношении соседства. В случае окрестности (3×3) каждое такое подмножество содержит максимум четыре пикселя. Разбиение считается полностью заданным, если известны все целочисленные значения $X_{g,h} = i$, $I = 1, 2, \ldots, k$.

Например, для $V_{\{(g,h),(m,n)\}}$ получаем k^2 свободных параметров. Логарифмируя (1.4), получим

$$\log P(x_{1,1},...,x_{G,H}) = -\log W - \frac{1}{T}\sum_{d\in D} V_d(x_{1,1},...,x_{G,H}).$$

Тогда в условном распределении все термы V_d, содержащие (g, h), сокращаются, и в результате получим:

$$P(x_{g,h}=i \mid x_{m,n},(m,n)\neq(g,h)) = \frac{P(x_{1,1},...,x_{g,h}=i,...,x_{G,H})}{\sum_{j=1}^{k} P(x_{1,1},...,x_{g,h}=j,...,x_{G,H})} \quad (1.6)$$

Условное распределение вида (1.6) служит моделью обратного распределения ошибки обучения нейросети с мультипликативными входными переменными. Эта модель не применима для оценки расстояния между прогнозируемым значением выходной функции Y' вида (1.1) и наблюдаемым значением Y, она, скорее, отражает «энтропийный» характер функции. Альтернативой для данной модели является машина Больцмана, которая использует марковское случайное распределение. Однако применение машины Больцмана для практических задач является затруднительным, поскольку этот алгоритм имеет трудоемкость существенно более высокую, нежели, алгоритм обратного распространения ошибки.

Распределение вида (1.6) можно использовать для определения неизвестных параметров в примере планирования обстановки рабочего кабинета (см. рис. 1.2). Речь идет, например, о следующей ситуации. Определено местоположение кресла в кабинете, но неизвестна его ориентация в пространстве (С, Ю, В, З). Речь идет о том, что для переменных $i_с$, $i_ю$, $i_в$, $i_з$ известно лишь то, что их сумма вероятностей равна 1:

$$P(x_{g,h}=i_с)+P(x_{g,h}=i_ю)+P(x_{g,h}=i_в)+P(x_{g,h}=i_з)=1 \quad (1.7)$$

Это значит, что можно использовать простой способ для оценки неизвестного параметра, который состоит во введении новой переменной выражающей сумму (1.7).

Итак, после того как будут определены параметры условного распределения, можно начинать ассоциативное доопределение графического образа с учетом конкретных условий и ограничений различного типа, например:

• размеры рабочего кабинета, определяющие размерность некоторой марковской модели $\{X_{g,h} \mid 1 \le g \le G; 1 \le h \le H\}$;

• заданное местоположение стен, окон и дверей кабинета, для которых $P(X_{g,h}=i)=1$ и которое в ходе работы алгоритма изменяться уже не будет;

• частота появления объекта i в наблюдаемой конфигурации;

• сумма некоторых переменных заранее задана, например, известно, что в кабинете должно находиться два кресла.

Доопределение графического образа считается оптимальным, если найденное значение переменных $X_{g,h}$ для объектов $i = 1, .., k$ имеет наивысшую вероятность с учетом условий и ограничений. Необходимо отметить, что распределение вида (1.5) в аналитическом виде имеется не всегда. Поэтому можно применить метод имитационного моделирования, а вместо (1.7) использовать распределения вида

$$P_T(x_{g,h}=i \mid x_{m,n};(m,n)\in N_{g,h}) := \frac{p(x_{g,h}=i \mid x_{m,n};(m,n)\in N_{g,h})^{1/T}}{\sum_{j=1}^{k} P(x_{g,h}=j \mid x_{m,n};(m,n)\in N_{g,h})^{1/T}} \quad (1.8)$$

которое совпадает с (1.7) при T=1. В теории стохастических цепей Маркова доказано, что при $T>-\infty$ выражение (1.8) приводит к наиболее вероятной конфигурации.

Тогда алгоритм мягких вычислений для ассоциативного доопределения образа имеет следующий вид.

Шаг 1. Выбор начального значения $T>1$, а все переменные $X_{m,n}$ принимают случайные значения в соответствии с априорным распределением или фиксированные значения в соответствии с заданными ограничениями.

Шаг 2. Случайный выбор пикселя (g, h).

Шаг 3. В соответствии с (1.8) расчет условных вероятностей:

$$P_T(x_{g,h}=i \mid x_{m,n};(m,n)\in N_{g,h})$$

Шаг 4. Случайный выбор значения переменной $X_{g,h}$, полученного с помощью (1.8).

Шаг 5. Уменьшение значения T, переход на *шаг 2* до тех пор, пока не будет выполнено условие остановки работы алгоритма.

Данный алгоритм обобщается на случай, когда расчет вероятностей производится не для одного выбранного пикселя (*шаг 2*), а для двух или более пикселей, находящихся в отношении соседства. Например, для двух пикселей необходимо определять k^2 вероятностей. Ясно, что трудоемкость расчетов при этом возрастает, однако

возникает некоторое преимущество, связанное с тем, что одновременно изменяются значения нескольких пикселей.

Алгоритм был успешно применен для представленного на рис. 1.2 демонстрационного примера. Для работы алгоритма были установлены следующие параметры. Число ориентированных в пространстве объектов равно 9 плюс один неориентированный объект (см. рис 1.1). Поэтому параметр $k = 4*9 + 1 = 37$. Поскольку применялось отношение соседства в окрестности (3×3), то число входных переменных для условного распределения вероятностей (1.7) и, соответственно, для нейросетей составило множество из 8*37 = 296 переменных. Для пикселя в центре сгенерировано 10 нейросетей: по одной для каждого ориентированного на север объекта плюс одна сеть неориентированного объекта. Каждая сеть содержала 895 различных весовых значений. Эксперименты показали, что при выборе слишком малых значений параметра T возникала опасность «попадания» в локальный минимум. Поэтому устанавливалось начальное значение T=5, которое после каждой итерации алгоритма делилось на 1,05. После примерно 70 итераций получалась устойчивая конфигурация плана рабочего кабинета.

В перспективе интерес представляет гибридизация предложенной модели с эволюционными алгоритмами и нечеткими множествами [7]. Это обеспечит более широкое применение модели и ее оптимизацию в сферах, где традиционно используются модели нейросетей: распознавание образов, таксономия, прогнозирование.

ГЛАВА 2. ТРАЕКТОРНЫЕ И ПОПУЛЯЦИОННЫЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ АЛГОРИТМЫ

2.1. Особенности поиска решений траекторными биоэвристиками

Особенность траекторных проблемно-ориентированных алгоритмов принятия оптимальных решений состоит в том, что на каждой итерации имеется одно допустимое решение и осуществляется переход к следующему. Бывает, что неизвестен вид и свойства оптимизируемой функции, но можно изменять значения входных переменных функции, вычислять ее значение и оценивать его. Или имеется возможность тестирования потенциального решения задачи, но нет предположений относительно характера поверхности оптимизируемой функции. К тому же решение не обязательно является числовым вектором: оно может быть деревом, продукционным правилом, схемой, всем, что может иметь отношение к задаче. Чтобы решать задачу в таких условиях необходимо наличие хотя бы одного начального решения, процедуры оценки качества решения и возможностей для его улучшения.

Большинство разработанных и применяемых на практике траекторных проблемно-ориентированных алгоритмов принадлежат к недетерминированным алгоритмам [19; 20]. Многие постановки оптимизационных задач зачастую содержат данные, которые имеют определенные погрешности, это делает неоправданными значительные вычислительные затраты для нахождения точного решения.

Идея траекторных проблемно-ориентированных алгоритмов основана на предположении, что целевая функция решаемой задачи имеет много локальных экстремумов, а просмотр всех допустимых решений невозможен, несмотря на конечность их числа. В такой ситуации нужно сосредоточить поиск в наиболее перспективных частях допустимой области. Каждый из траекторных проблемно-ориентированных алгоритмов решает эту проблему по-своему [21].

Основной принцип поиска решений траекторными алгоритмами представлен на рисунке 2.1.

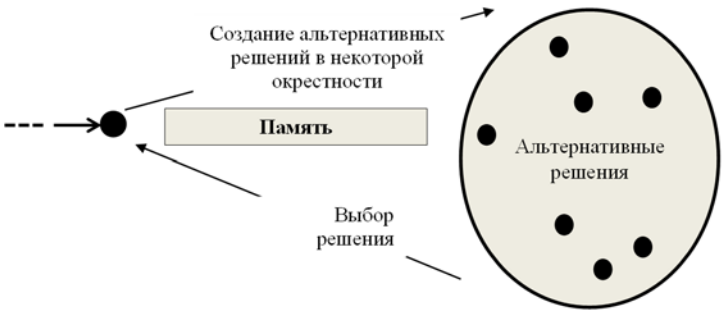


Рис. 2.1. Принцип поиска решений траекторными алгоритмами

Траекторный алгоритм состоит в многократном применении процедуры поиска альтернативных решений в некоторой окрестности и замены текущего решения на альтернативное решение. Для создания альтернативных решений используется текущее решение *s*. Множество альтернативных решений *C(s)*, как правило, получают путем локальных преобразований решения *s*. На этапе замены текущего решения *s* новым решением *s^* выбор осуществляется из множества альтернативных решений *C(s)*. Этот итерационный процесс повторяется многократно до выполнения заданного критерия остановки. Этапы создания и выбора альтернативных решений могут проводиться как без использования, так и с использованием памяти. Если память не используется, то этапы создания и выбора альтернатив основываются только на текущем решении *s*. Иначе, хранящаяся в памяти история поиска может быть использована при создании и выборе альтернативных решений.

Траекторные алгоритмы с краткосрочной памятью запоминают определенное количество последних принятых, сгенерированных решений или частей решений. Траекторные алгоритмы с долгосрочной памятью запоминают информацию обо всем осуществленном процессе поиска в виде набора специальных переменных. Траекторные алгоритмы оставляют в пространстве поиска траекторию, последовательность решений, где каждое решение является соседним для предыдущего относительно некоторой окрестности.

Ниже приводится программа на псевдокоде, иллюстрирующая поиск решений траекторными алгоритмами.

Input: Инициализация начального решения s_0.
t:=0;
Repeat
Создание альтернативных решений $C(s_t)$ путем применения определенных операторов в некоторой окрестности решения s_t;
Замена решения s_t на лучшее решение s_t+1 из множества $C(s_t)$;
t:=t+1;
Until Проверка условия останова;
Output: Найдено лучшее решение.

Общими понятиями для всех траекторных алгоритмов являются окрестность поиска и инициализация начального решения.

Окрестность поиска играет важную роль в траекторных алгоритмах. Если структура окрестности поиска не адекватна задаче, то проблему вряд ли удастся решить. Понятие «окрестность» для каждой конкретной задачи оптимизации обладает своей спецификой.

Окрестность точки s_0 – это множество, содержащее данную точку, и близкие к ней. Пусть $\varepsilon > 0$ произвольное фиксированное число. Окрестностью точки s_0 на числовой прямой (ε-окрестность) называется множество точек, удаленных от s_0 менее чем на ε:

$$O_\varepsilon(s_0) = \{s: |s - s_0| < \varepsilon\} \qquad (2.1.)$$

В многомерном случае для непрерывной задачи роль окрестности выполняет ε-шар с центром в точке s_0. В метрическом пространстве (M, ρ) окрестностью с центром в точке s_0 называют множество

$$A = \{s \in M : \rho(s_0, s) < \varepsilon\} \qquad (2.2)$$

Решение s в окрестности точки s_0 называется соседним для s_0.

Рисунки 2.2 а и 2.2 б иллюстрируют понятие соседства для непрерывной и дискретной задачи соответственно.

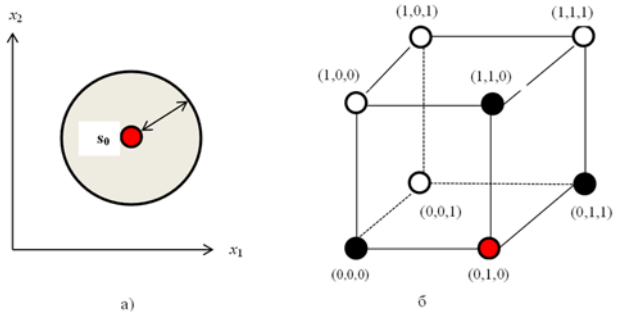


Рис. 2.2. Окрестности точки s_0:
а) для непрерывной задачи; б) для дискретной задачи

Для задач дискретной оптимизации определить понятие окрестности решения как для непрерывных функций, трудно или невозможно. Часто понятие близости вводится искусственно, основываясь на особенностях задачи.

Так для некоторого двоичного вектора (0, 1, 0), являющегося решением задачи о ранце, окрестность определяется множеством всех векторов, получаемых из данного вектора заменой любого нуля на единицу или единицы на ноль, как это показано на рис. 2.1б. Иными словами, окрестность двоичного вектора составляют вектора, хемминговo расстояние до которых равно единице.

Например, в задаче коммивояжера окрестностью для гамильтонова маршрута обхода всех городов будут все обходы, полученные перестановкой любых двух городов [22].

В общем, соседнее решение порождается применением некоторого оператора.

Пусть задана задача оптимизации (D, f), где D – допустимая область, f – целевая функция. Тогда системой окрестностей называется отображение N: $D \rightarrow 2D$, определенное для каждой индивидуальной задачи ($2D$ – множество всех подмножеств множества D).

Если D – это обычное евклидово пространство, то множество точек, находящихся ближе определенного расстояния в евклидовой метрике от заданной точки, являются «естественной» окрестностью. Если речь идет о задачах дискретной и комбинаторной оптимизации, то выбор системы окрестностей совсем не очевиден и существенно зависит от структуры допустимого множества D.

Пусть дана индивидуальная задача оптимизации (D, f) и система окрестностей N. Тогда некоторое допустимое решение X^* называется локально оптимальным или оптимальным относительно системы окрестностей N, если $f(X^*) < f(Y)\ \forall\ Y \in N(X^*)$.

Решение X задачи (D, f) называется глобально оптимальным, если $f(X) < f(Y)\ \forall Y \in D$.

Рисунок 2.3 иллюстрирует понятия глобального и локального оптимума.

Выбор системы окрестностей осуществляется на основе человеческой интуиции. Теоретических рекомендаций не имеется. Однако существует конкуренция между малыми и большими (по количеству содержащихся в них точек) окрестностями. Окрестности больших размеров могут дать лучший локальный оптимум, но это

займет больше времени, а значит будет найдено меньшее количество локальных оптимумов в течение фиксированного временного интервала. Возникает вопрос: сгенерировать больше малых окрестностей или меньше больших окрестностей? Такого рода вопросы обычно имеют эмпирический ответ.

Рис. 2.3. Локальный и глобальный оптимум функции *f(X)*

Еще один вопрос связан с выбором порядка просмотра окрестности. Для этого необходимо в первую очередь упорядочить окрестность. Для оптимизации на дискретной решетке это, например, сделать легко, потому что порядок естественным образом порождается нумерацией координат. Напротив, в задаче коммивояжера упорядочение не совсем очевидно, для подобного рода комбинаторных структур более подходящим является лексикографическое упорядочение. В сложных случаях может быть применено случайное упорядочение элементов окрестности.

После упорядочения точек окрестности возникает задача выбора тактики просмотра окрестности. В основном различают тактику перехода по первому улучшению и наискорейший спуск, который состоит в том, что каждая текущая окрестность просматривается полностью и переход осуществляется в наилучшую точку этой окрестности. Однако эксперименты с траекторными проблемно-ориентированными алгоритмами показывают, что лишние затраты на наискорейший спуск редко бывают оправданы и для применения этой тактики нужны серьезные основания. В случае перехода по первому улучшению появляется еще одна проблема – это выбор способа продолжения просмотра окрестности после перехода в новую точку. Возможны такие варианты как продолжение просмотра с начала (в новой окрестности вначале просматривается

точка с номером один, затем – два и т. д.), продолжение просмотра с точки со следующим номером и продолжение просмотра с точки с тем же номером. В последнем случае возможно вырождение алгоритма в покоординатный спуск или зацикливание.

Различают эвристические и точные подходы к тактике просмотра больших окрестностей. Эвристические подходы были успешно применены для решения таких задач комбинаторной оптимизации как кластеризация, маршрутизация, назначение, разбиение графов, планирование. Анализ точных подходов к тактике просмотра больших окрестностей приводится в [23; 24].

Если любая точка $X \in D$, локально оптимальная относительно системы окрестностей N, является глобально оптимальной, то система окрестностей N называется точной. В частности, для задачи коммивояжера точной будет система окрестностей, порожденная n-заменой (заменяются n дуг, где n – число городов), а для задачи оптимизации с булевыми переменными точной будет система окрестностей $\{Y \in D: |x_i - y_i| \le 1, \forall i = 1,\ldots, n\}$.

Поиск решений траекторными алгоритмами предполагает необходимость делать выбор. Первой проблемой является выбор начального решения. На практике бывает полезно выполнить алгоритм из нескольких начальных точек (мультистарт) и выбрать лучший результат. В таком случае придется решать, сколько начальных точек выбирать и как распределить их в пространстве.

Для выбора начального решения используются две основные стратегии: случайный выбор и жадный подход. Существует компромисс между использованием случайного и жадного подхода к выбору начального решения в плане качества найденного оптимального решения и времени вычислений. Например, чем больше окрестность выбора, тем менее чувствительным от выбора начального решения является траекторный алгоритм.

Использование более эффективных решений в качестве исходных решений не всегда приводит к лучшему локальному оптимуму. Поэтому для повышения надежности вычислений рекомендуется использование гибридной стратегии путем объединения случайного и жадного подходов. Например, пул исходных решений для мультистарта может включать как жадные, так и случайно сгенерированные решения. Кроме того, в процессе инициализации могут быть использованы различные жадные стратегии. Отметим

также, что для отдельных задач оптимизации затруднительно сгенерировать допустимые случайные решения. В этом случае, альтернативой являются жадные подходы к генерации возможных начальных решений.

Определим пространство поиска решений через ориентированный граф $G = (S, E)$, где множество вершин S соответствует множеству кодируемых решений задачи, а множество ребер E соответствует множеству операторов перемещения, используемых для генерации новых решений. Вершины s_i и s_j являются соседними и связаны ребром, если решение s_j может быть получено из решения s_i с использованием оператора перемещения.

Ландшафт L функции оптимизации определяется как кортеж (G, f), где граф G представляет собой пространство поиска, а f – оптимизируемую функцию. Двух или трехмерный ландшафт можно представить с использованием географических терминов (равнины, пики, овраги, плато и др.), а траекторный алгоритм – как траекторию поиска, например, пика.

Более общей моделью является *NK*-модель ландшафта. С ее помощью классифицируются целевые функции ландшафтов в зависимости от двух параметров N и K. Параметр N определяет размерность задачи, а параметр K – степень влияния переменных на оптимизируемую функцию пригодности.

Пусть, например, каждое решение в пространстве поиска описывается двоичным вектором размера N, а пространство поиска решений представляет собой N-мерный гиперкуб, причем целевая функция определяется следующим образом:

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x_i, x_{i1}, x_{i2}, \dots, x_{ik}) \qquad (2.3)$$

где f_i выражает зависимость решения от переменной x_i и от K значений других переменных x_{i1}, x_{i2}, …, x_{ik}. Переменные решения x_{i1}, x_{i2}, …, x_{ik} выбираются случайным образом из N переменных. Для 2^{K+1} различных входов, функция f_i принимает значения равномерно распределенного случайного числа в диапазоне от 0 до 1. Тогда при $K = 0$ ландшафт имеет только одно оптимальное решение, а при $K = N - 1$ число локальных оптимумов равно $2^N/(N + 1)$.

Чтобы иметь возможность анализировать ландшафт целевых функций необходимо проверить важное свойство пространства поиска – связность, другими словами, установить существует ли путь в графе G между любой парой вершин s_i и s_j. Если путь существует, то, следовательно, существует путь из любой исходной вершины к глобально оптимальному решению s^*.

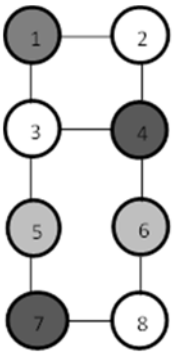
Продемонстрируем свойство связности на примере известной квадратичной задачи о назначении, которая встречается в различных приложениях (анализ данных, синтез изображений и др.). Имеется множество $O = \{O_1, O_2,\ldots, O_n\}$ из n объектов и множество $P = \{P_1, P_2,\ldots, P_n\}$ возможных позиций объектов. Известна матрица $C=||c_{ij}||_{n\times n}$, каждый элемент которой обозначает стоимость потока между объектами O_i и O_j, и матрица $D=||d_{ij}||_{n\times n}$, каждый элемент которой равен расстоянию между позициями P_i и P_j. Требуется найти назначение объектов на позиции $O \rightarrow P$, чтобы минимизировалась функция:

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} d_{ij} \qquad (2.4)$$

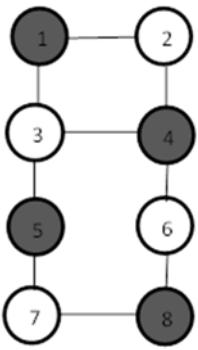
Здесь каждое решение кодируется с помощью перестановки, а отношение соседства основано на свопинге между двумя элементами. Поэтому пространство поиска (граф) здесь является связным.

Однако так бывает не всегда. В другой известной задаче раскраски графа (любым двум смежным вершинам должны соответствовать разные цвета) пространство поиска может быть несвязным. Например, рисунок 2.4 показывает, что оптимальная раскраска графа s^* не может быть достигнута из исходного решения s.

Для анализа ландшафта оптимизируемой функции введем понятие расстояния в графе G, представляющего пространство поиска. Это позволит определять пространственную структуру ландшафта при разработке таких поисковых механизмов, как диверсификация и интенсификации процедуры поиска оптимума.



а)



б)

Рис. 2.4. Раскраска графа: а) исходная раскраска графа *s*, б) оптимальная раскраска графа *s**

Определим расстояние $d(s_i, s_j)$ между решениями s_i и s_j на графе G как длину самого короткого пути, т.е. минимальное количество применений оператора перемещения для перехода из s_i в s_j. Расстояние должно обладать следующими топологическими свойствами:

- $d(x, y) = 0$, если и только если $x = y$;
- $d(x, y) = d(y, x)$;
- $d(x, y) + d(y, z) >= d(x, z)$.

Отметим, что для некоторых задач оптимизации вычисление расстояния между решениями может оказаться *NP* – полной проблемой или же иметь оценку сложности высокого полиномиального порядка.

Ландшафт характеризуется различными свойствами: числом локальных оптимумов, их распределением, неровностью, дисперсией и др. Большинство из этих свойств основаны на статистике и являются глобальными или локальными характеристиками ландшафта.

Глобальные характеристики дают информацию о структуре ландшафта в целом, используя все пространство поиска. Однако целью оптимизации является поиск «хороших» решений, которые, обычно, представляют собой малую часть пространства поиска. Поэтому глобальных характеристик для исследования ландшафта недостаточно.

Траекторные алгоритмы сосредотачивают внимание на «хороших» решениях в локальном пространстве. Поэтому интерес представляет анализ этих решений и области, где они локализованы. Глобальные характеристики здесь мало чем помогут, поскольку они дают информацию в «среднем» о большом числе решений, порой не связанных с применяемой в траекторном алгоритме эвристикой.

Однако имеются и обратные примеры. Рассмотрим следующую задачу. Дано некоторое множество S из n натуральных чисел $\{a_1, a_2, \ldots, a_n\}$. Найти разбиение S на две равные по количеству $n/2$ непересекающиеся части S_1 и S_2, чтобы разность их сумм была минимальной по абсолютному значению:

$$f = \left|\sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i\right| \qquad (2.5)$$

Это *NP*-задача. Проблема возникает уже при $n > 35$. Число локальных оптимумов в ландшафте данной функции, согласно [18] оценивается как $2{,}764 * 2^n / n^{3/2}$. Это при том, что размер пространства поиска равен 2^n. Таким образом, число локальных оптимумов в задаче очень велико по сравнению с размером пространства поиска. Их отношение составляет порядок $O(n^{-1,5})$ [18].

В целом, глобальные и локальные характеристики должны дополнять друг друга. Поэтому для анализа ландшафта, как в пространстве решений G, так и функций f предлагается применять две различные характеристики: меру распределения локальных решений для оценки топологии пространства и корреляционную меру для анализа неровностей ландшафта и оценки соотношения между качеством решений и относительными расстояниями между ними [25].

Для множества полученных решений P определим среднее расстояние $dmm(P)$ и нормированное среднее расстояние $Dmm(P)$:

$$dmm(P) = \frac{\sum_{s \in P} \sum_{t \in P, t \neq s} dist(s,t)}{|P|(|P|-1)}, \qquad (2.6)$$

$$Dmm(P) = \frac{dmm(P)}{diam(S)}, \qquad (2.7)$$

$$diam(P) = \max_{s,t \in P} dist(s.t) \qquad (2.8)$$

Здесь $diam\ (S)$ – диаметр множества решений S, равный максимальному расстоянию между решениями.

Нормированное среднее расстояние характеризует концентрацию множества решений P в пространстве поиска. Небольшое расстояние указывает на то, что решения группируются в малой области пространства поиска. Величина

$$\Delta_{Dmm} = (Dmm(U) - Dmm(O))/Dmm(U) \quad (2.9)$$

представляет собой изменение среднего расстояния между множеством однородно распределенных решений U и множеством локальных оптимальных решений. При этом мощность множества O в зависимости от сложности траекторного алгоритма локального поиска выбирается на интервале $10^3 \le O \le 10^4$.

Энтропия позволяет измерять разнообразие множества решений в пространстве поиска. Величина

$$\Delta_{ent} = \text{ent}(U) - ent(O)/ent(U) \quad (2.10)$$

представляет изменение энтропии между начальными решениями U и конечным множеством локальных оптимумов O, полученных после выполнения траекторного алгоритма. Энтропия дает информацию о разбросе, а не о зернистости решений. Хотя решения, сконцентрированные в нескольких областях, и решения, сконцентрированные в одной области, могут иметь одинаковую небольшую энтропию.

Для анализа распределения решений в пространстве функции f может использоваться амплитуда. Амплитуда – это величина относительной разности между лучшим и худшим значением функции f в некотором множестве решений P:

$$Amp(P) = \frac{|P|(\max\limits_{s\in P} f(s) - \min\limits_{s\in P} f(s))}{\sum_{s\in P} f(s)} \quad (2.11)$$

Относительное изменение амплитуды Δ_{Amp} определяется соотношением между начальными решениями U и конечным множеством локальных оптимальных решений O по формуле:

$$\Delta_{Amp} = \frac{Amp(U) - Amp(o)}{Amp(U)} \quad (2.12)$$

Корреляционная мера позволяет оценить гладкость ландшафта и корреляцию между качеством решения и его расстоянием до глобального оптимума. Ландшафт оценивается как негладкий, если он содержит много локальных оптимальных решений и характеризуется низкой корреляцией между соседними решениями. В качестве

корреляционной меры можно использовать такие показатели как длина склона у пика функции, автокорреляционная функция и др.

В частности, средняя длина склона у пика $Lmm(P)$ для множества решений P определяется как

$$Lmm(P) = \frac{\sum_{p \in P} l(p)}{|P|} \qquad (2.13)$$

где $l(p)$ – длина пути до пика оптимизируемой функции, начиная с некоторого решения $p \in P$. Величина $Lmm(P)$ дает некоторую информацию о гладкости ландшафта. При гладком ландшафте количество локальных оптимумов небольшое и длина пути по склону до оптимума больше, нежели при большом количестве пиков.

С другой стороны, целевые функции некоторых задач оптимизации характеризуются наличием обширного плато, преодолеть которое траекторный алгоритм, подчас, не в состоянии [26; 27].

Пусть имеется решение s в пространстве поиска S, v – некоторое значение критерия f. Обозначим через $N(s)$, подмножество точек s' в окрестности решения s. Пусть X является подмножеством $N(s)$, которое состоит из решений $s'' \in X$ таких, что $f(s'') = v$. X представляет собой *плато*, тогда и только тогда, когда $|X| \geq 2$.

Если ландшафт функции включает несколько плато, то целевой функции оптимизации f недостаточно, чтобы различить подмножества $N(s)$ окрестностей текущей точки s. Чтобы отделить одно плато от другого необходимо использовать дополнительную информацию. Либо изменить целевую функцию.

В целом анализ ландшафта, являясь интересным и ценным ресурсом, однако требует такие данные, как значение глобального оптимума, количество локальных оптимумов, определение расстояния.

Если множество допустимых решений велико, то траекторные алгоритмы зачастую позволяют найти близкие к оптимальным решения с меньшим вычислительными затратами, нежели многие другие алгоритмы. Это перспективный подход к решению многих оптимизационных проблем.

2.2. Популяционный муравьиный алгоритм транспортной логистики и его визуализация

Мягкие роевые вычисления представляют собой математические преобразования, описывающие коллективное поведение децентрализованной самоорганизующейся системы, состоящей из множества агентов, локально взаимодействующих между собой и с окружающей средой [5].

Термин был введен Х. Бени и Ван Цзином в 1989 г. в контексте системы клеточных роботов. Примерами в природе может служить колония муравьев [8], рой медоносных пчел [28], косяк рыб [29], стая птиц [30], рой бактерий сальмонеллы [30], поведение кукушек в процессе вынужденного гнездового паразитизма, рой светлячков, колонизация сельскохозяйственных угодий сорняками, аспекты поведения обезьян в процессе поиска ими пищи, рой мух, поведение группы лягушек при поиске пищи, стая летучих мышей, эволюция растущих деревьев и другие естественные сущности, для которых характерна роевая модель поведения [19]. Каждый агент функционирует автономно по довольно примитивным правилам. В противовес почти примитивному поведению агентов, алгоритм поведение всей системы получается на удивление разумным.

Роевые алгоритмы позволяю получать неплохие результаты при решении многих задач оптимизации, принятия решений, календарного планирования, автоматизации проектирования, интеллектуального анализа данных, кластеризации. Их главной особенностью является бионическая природа. Для роя необходима определенная форма связи, чтобы сотрудничать при решении общей задачи. Эта связь может быть прямой или косвенной. Косвенная связь предполагает изменение индивидуумом окружающей среды так, чтобы это изменило поведение других индивидуумов, проходящих через эту измененную среду в будущем. Пример косвенной связи – наложение следов феромона, оставляемого некоторыми разновидностями муравьев. Тем самым муравьи обеспечивают коммуникацию через среду, придавая ей семиотические характеристики.

Принцип изменения окружающей среды как средство связи для изменения поведения называется стигмергией (*stigmergie*, стимулирующий работу). Стигмергия обозначает спонтанное непрямое взаимодействие между индивидами, которое происходит через оставленные индивидами в окружающей среде следы или метки,

которые стимулируют дальнейшую активность этих же или других индивидов. Она является основным принципом в организации муравьиной колонии, несмотря на наличие в колонии королевы – специализированного муравья, ответственного за размножение. Королева не имеет управляющей функции. Вместо этого муравьиные колонии самоорганизуются. Выгоды самоорганизации очевидны – колония может эффективно поддерживать свое существование, даже когда большое число муравьев неспособно к содействию. Имитация самоорганизации роя составляет основу роевых алгоритмов.

Представим популяционный алгоритм муравьиной колонии для решения задачи транспортной логистики и его визуализацию на географической карте [31].

Муравьи живут в колониях, состоящих из большого количества взаимодействующих индивидуумов. Колонии способны решать задачи, которые ни один индивидуум не был бы способен решать отдельно (поиск кратчайшего пути к пище, назначение рабочей силы, кластеризация при организации колонии и др.). Например, муравей-фуражир в процессе добычи пищи на обратном пути в муравейник в случае, если он нашел источник пищи, помечает свой путь, оставляя определенное количество ферментов внешней секреции, биологических маркеров собственного вида (феромонов). Тем самым он дает знак другим муравьям. Фермент действует не постоянно, а постепенно улетучивается. Знаки, оставленные всеми муравьями, суммируются, и таким образом решается динамическая пространственная оптимизационная задача коллективного поиска, формируется наиболее оптимальная сеть фуражировочных дорог.

Феромонные следы служат распределенной численной информацией. Она используется муравьями для недетерминированного конструирования решений задачи и отражает опыт, накопленный в процессе поиска решения.

Первоначально алгоритм муравьиной колонии был применен к задаче коммивояжера (*Traveling Salesman Problem - TSP*), где необходимо найти самый выгодный маршрут, проходящий через указанные города хотя бы по одному разу с последующим возвратом в исходный город. Первым алгоритмом муравьиной колонии был [28]. Этот алгоритм является базовым для большинства из его многочисленных модификаций и разновидностей [8]. Основные этапы работы

алгоритма муравьиной колонии для задачи коммивояжера можно представить в виде следующей последовательности шагов:

Шаг 1. Задание параметров (число муравьев в популяции; максимальное число итераций, начальная концентрация феромона, устойчивость феромона, интенсификация феромона, интенсификация эвристики), инициализация алгоритма.

Шаг 2. Движение муравьев.

Шаг 3. Обновление уровня феромона на дугах сети, расчет длины пути.

Шаг 4. Проверка на достижение оптимального результата. Критериями окончания поиска могут быть максимальное число итераций, отсутствие изменений в выборе наилучшего пути. Если результат проверки положительный, то переход к *шагу 6*, иначе – к *шагу* 5.

Шаг 5. Перезапуск итерации алгоритма, переход к *шагу* 2.

Шаг 6. Остановка поиска и определение наилучшего результата, который является решением задачи.

В дальнейшем алгоритм стали применять к любой задаче, которая допускает следующую интерпретацию в терминах теории графов [32]:

• дан граф $G = (X, U)$ с конечным набором компонент решений $X = \{x_1, x_2, ..., x_n\}$;

• определены состояния проблемы, множество их возможных последовательностей и множество последовательностей, которые удовлетворяют ограничениям задачи;

• определен конечный набор возможных соединений U между элементами X;

• с компонентами и их соединениями ассоциируются феромонные следы, представляющие долговременную память муравьев;

• память используется для хранения информации о пройденном пути, для нахождения допустимых решений, для оценки найденного решения или для возвращения назад с целью размещения феромона;

• муравей, находясь в некотором состоянии, может переместиться в любой узел графа из множества допустимых соседних узлов;

• переход осуществляется с помощью вероятностного правила;

• найдя решение, муравей может пройти этот же путь назад и обновить феромонный след на использованных соединениях или компонентах.

Условием окончания поиска обычно является достижение t_{max} итераций.

На этапе инициализации необходимо указать [33]:

– коэффициент, определяющий относительную значимость пути;

– параметр, показывающий значимость расстояния;

– коэффициент количества феромона, которое муравей оставляет на пути;

– коэффициент устойчивости феромона;

– в некоторых модификациях начальное значение феромона, которое находится на путях следования муравьев до начала моделирования.

Затем создается популяция муравьев, равномерно распределенная по узлам сети (вершинам графа) таким образом, чтобы узлы могли стать отправной точкой с примерно равными шансами.

Затем описывается движение муравьев. Если муравей еще не посетил все узлы сетевого графа, то для определения следующей дуги рассчитывается вероятность перехода из вершины, в которой муравей находится, в следующую вершину по формуле [34]:

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(n_{i,j}^{\beta})}{\Sigma(\tau_{i,j}^{\alpha})(n_{i,j}^{\beta})} \qquad (2.14)$$

где $p_{i,j}$ – вероятность перехода от узла i к узлу j, $\tau_{i,j}$ – количество феромонов на ребре (i, j); α – параметр, контролирующий влияние $\tau_{i,j}$; $\eta_{i,j}$ –привлекательность ребра (i, j); β – параметр, контролирующий влияние $\eta_{i,j}$. Шаги и расчеты на данном этапе алгоритма повторяются до тех пор, пока каждый муравей не завершит путь.

После завершения перемещений муравьев подсчитывается длина пути. Она равна сумме длин всех дуг, по которым путешествовал муравей. Полученный результат в виде количества феромона позволяет измерить путь: короткий путь характеризуется высокой концентрацией феромона, а более длинный путь – более низкой.

Процесс испарения является полезной формой «забывания», он содействует исследованию новых областей в пространстве поиска, позволяет избежать быстрой сходимости алгоритма в субоптимальной области. Следует ли выполнять этот процесс параллельно и независимо для каждого муравья, или необходима какая-то

форма их синхронизации – эти вопросы оставляются на усмотрение разработчика конкретного алгоритма, допуская свободу в определении способа взаимодействия этих процессов.

Важной особенностью алгоритма является то, что муравьи передвигаются одновременно и независимо. Хорошие решения, обычно, появляются только в результате коллективного взаимодействия между муравьями. В известном смысле это является распределенным процессом обучения. Математические модели описания процесса обучения могут быть различными. Поэтому были разработаны разновидности муравьиных алгоритмов. К ним относятся алгоритм муравьиных систем, основанный на элитной стратегии; алгоритм муравьиных систем, основанный на ранжировании; алгоритм системы муравьиных колоний; максиминный алгоритм муравьиных систем и другие [8]. Были сделаны попытки улучшить базовый алгоритм путем использования иных механизмов добавления феромона, правил выбора муравьем следующего пункта, применение элитной стратегии, использование ограничений для различных параметров, применение локальной оптимизации, оценки влияние количества муравьев на результат.

Рассмотрим проблемно-ориентированный алгоритм муравьиной колонии для решения задачи транспортной логистики.

На сегодняшний день большая часть компаний предоставляет клиентам возможность получения своей продукции путем её доставки курьером. От того, насколько оптимальны и корректны маршруты доставки заказов, и, соответственно, высока скорость работы курьеров, зависит не только лояльность заказчиков к компании, но и максимальная выручка.

Поставленной задаче соответствует одна из разновидностей классической задачи коммивояжера, а именно задача о маршрутизации транспорта с несколькими депо (англ. *Multiple depot vehicle routing problem*, *MDVPR*). Ее постановка заключается в следующем: в каждом депо располагается парк транспорта, каждая машина которого может выезжать, обслуживать потребителей, но затем обязательно должна возвращаться обратно. Целью классической задачи маршрутизации транспорта с несколькими депо является минимизация количества используемого транспорта и времени, затрачиваемого на выполнение одной программы доставки.

Считаем, что для достижения наименьших временных затрат используется весь имеющийся у компании парк транспорта.

Для решения задачи транспортной логистики предлагается следующая модификация муравьиного алгоритма, позволившая улучшить производительность и точность получаемого решения [31].

Вместо отдельных муравьев-агентов, которые являются сущностью базового алгоритма, были введены группы агентов. Внутри группы каждому транспортному средству поставлен в соответствие муравей, начальными пунктами для движения являются депо. Все агенты начинают движение и осуществляют выбор ребра для перехода, по формуле вероятности базового алгоритма.

Ребра, по которым прошел путь муравьев, помечаются феромонами, уровень которых после каждой итерации уменьшается из-за испарения и вычисляется по формуле:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j} \qquad (2.15)$$

где ρ – коэффициент устойчивости феромонов.

Выбор следующего пункта предоставляется муравьям с минимальной суммарной длиной уже пройденного маршрута, что позволяет максимально эффективно распределять затраченное время. После полного обхода пунктов всеми группами происходит расстановка феромонов уже с учетом испарения ферромона. Далее используется идея алгоритма муравьиных колоний с ранжированием. Группы агентов сортируются по длине пройденного пути. Количество феромона, расставляемое на ребрах, вошедших в маршрут, пропорционально позиции группы в отсортированном списке, что видно из формулы:

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{(L_k * r)}, \text{если } k \text{ муравей проходил по ребру } i,j; \\ 0 \quad\quad , \text{иначе,} \end{cases} \qquad (2.16)$$

где r – номер в отсортированном списке, L_k – длина пути k-ого муравья, Q – константа, отображающая общее количество оставляемого муравьями феромона.

Также в алгоритм введена модификация, использующая понятие элитных муравьев. Некоторое количество групп агентов называем элитарными и пути, которые они проходят, являются наилучшими. Следовательно, на этих путях повышаем уровень феромона.

В силу того, что группы агентов во время одной итерации не зависят друг от друга и не изменяют совместные данные, их работу

можно производить параллельно, то есть вычисления производятся в отдельном потоке для каждой группы. Расчет уровня феромонов выполняет главный поток.

Одним из основных этапов реализации муравьиного алгоритма является настройка его параметров, в частности количества групп муравьев, веса феромонов, веса ребер, коэффициента устойчивости феромона, количества элитных групп.

Пример работы программы, позволяющей варьировать перечисленными параметрами алгоритма и оценивать результаты путем сравнения графиков работы алгоритмов с разными настройками на одних и тех же тестовых наборах данных, представлен на рисунке 2.5.

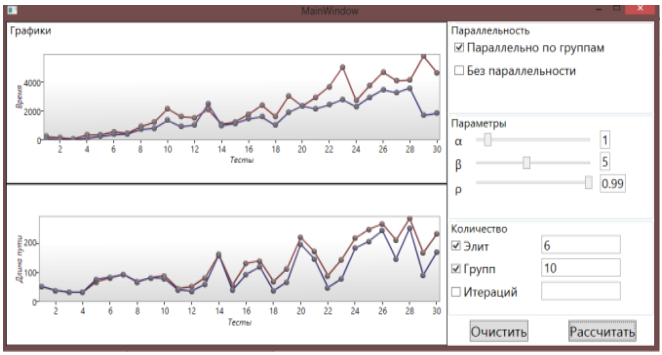


Рис. 2.5. Пример работы программы, позволяющей варьировать параметрами муравьиного алгоритма и оценивать результаты их работы путем сравнения графиков

Путем проведения экспериментов оптимальными были приняты следующие значения коэффициентов: количество групп муравьев – 15; вес феромонов – 1; вес ребер – 5; коэффициент устойчивости феромона – 0,99; количество элитных групп – 6.

Алгоритм с уже настроенными параметрами был проверен на бенчмарке Кордо (*Cordeau*), в котором содержатся данные именно под задачу маршрутизации транспорта с несколькими депо. Для каждой задачи в бенчмарке указывается оптимальное или лучшее известное решение. Целью оптимизации нашей задачи является время при максимальном использовании имеющегося парка транспорта, в то время как классический алгоритм минимизирует и количество средств перевозки. Результаты работы разработанного

муравьиного алгоритма для разных наборов данных дали похожую статистику.

Например, для одной из тестовых задач (бенчмарка *№ pr07*), в условии которой фигурирует 72 пункта-потребителя и 6 депо, получены следующие лучшие результаты: длина маршрута с учетом минимизации парка транспорта – 1085 условных единиц, максимальное время маршрута – 315 условных единиц. В предлагаемом алгоритме длина маршрута составила 1366 условных единиц, что несколько хуже, нежели результат, полученный в бенчмарке Кордо для этого набора данных. Однако максимальное время маршрута составило 263 условные единицы, что на 16,5% лучше, нежели в бенчмарке.

В настоящее время существует несколько программных продуктов, реализующих решение задачи транспортной логистики, включая картографические сервисы и специализированные веб-приложения, но у большинства из них есть ограничения, которые препятствуют их эффективному использованию. Некоторые из этих программ (например, картографические сервисы) предоставляют слишком узкий функционал, ограниченный возможностью построения только одного маршрута с жестко фиксированными начальным и конечным пунктами. Другие (веб-сервисы, клиентское программное обеспечение) обладают более широкими возможностями, но являются платными и добавляют расходы для небольших предприятий.

Для реализации решения задачи транспортной логистики муравьиным алгоритмом с визуализацией на географической карте было разработано приложение в виде веб-сервиса. Для создания динамического пользовательского интерфейса, а также асинхронной отправки и получения обрабатываемых данных (посредством технологии *AJAX*) были использованы возможности языка *JavaScript*. Веб-сервис использует функционал, который предоставляют существующие картографические службы. После рассмотрения доступных вариантов, предоставляющих необходимые возможности, предпочтение было отдано сервису «Яндекс.Карты».

«Яндекс.Карты» предоставляют достаточно объемный функционал, его карты для России имеют высокую точность. Точность карт очень важна для построения оптимальных маршрутов, а так как разработанное приложение в первую очередь направлено на

российскую аудиторию, то поставщиком картографической информации был выбран сервис «Яндекс.Карты».

Отдельно стоит упомянуть картографический сервис от *Bing*. При разработке клиентского приложения решение о выборе поставщика было не столь очевидным, поскольку *Bing Maps* предоставляет возможности для разработки на языке *C#* с использованием технологии *WPF*, являющиеся уникальными и не представленными ни в одном другом сервисе.

Разработанное приложение, а также пример его работы представлены на рисунке 2.6.

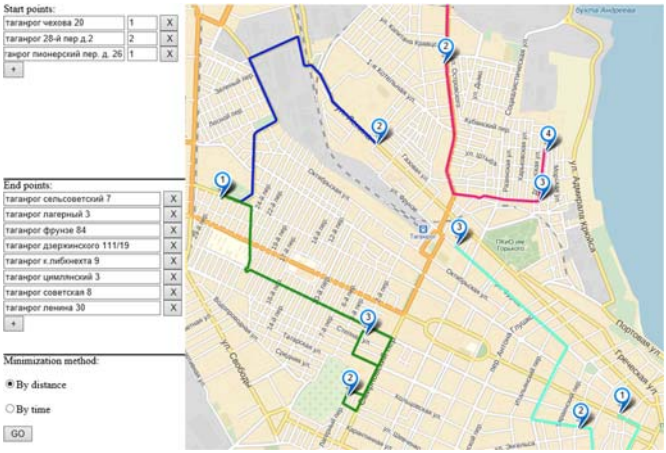


Рис. 2.6. Пример построения маршрута по модифицированному муравьиному алгоритму

Сценарий работы с веб-сервисом выглядит следующим образом:

• пользователь вводит адреса исходных пунктов и указывает количество курьеров в каждом из них;

• пользователь вводит адреса доставки;

• пользователь выбирает метод оптимизации пути (по дистанции или времени в пути) и запускает работу приложения;

• выполняя запросы к сервису Яндекс.Карты, приложение формирует сетку расстояний для всех пунктов на карте;

• выполняется *AJAX*-запрос к серверу с передачей сформированных данных;

• в *MVC* контроллере данные обрабатываются с использованием разработанной нами библиотеки и возвращаются в исходное представление;

• с помощью асинхронных запросов к *API*-функциям картографического сервиса на карте строятся искомые маршруты.

Отметим, что полученное веб-приложение работоспособно, удобно для пользователя, обладает интуитивно-понятным интерфейсом и, что немаловажно, выполняет возложенные на него задачи.

В дальнейшем планируется добавить возможность взаимодействия с курьерами, посредством использования мобильных приложений, а также введение работы с базами данных, улучшение пользовательского интерфейса и добавление дополнительных функциональных возможностей.

2.3. Популяционный масштабируемый алгоритм для задач многомерной оптимизации

С большой скоростью растет не только объем данных, но и сложность задач, которые требуется решить для этих данных с помощью информационных технологий [35]. Следовательно, актуальной становится задача адаптации интеллектуальных информационных технологий на основе методов машинного обучения, к обработке все возрастающих объемов данных. Определенную научную и практическую значимость имеет вопрос о том могут или нет алгоритмы мягких вычислений адекватно ответить на современные вызовы в области обработки больших данных: эффективно масштабироваться и распараллеливаться для решения задач с миллиардами переменных, конкурировать с методами обучения глубоких нейросетей в решении задач высокой сложности при существенно лучшем качестве исследования пространства поиска.

В задачах с нелинейным, многомерным, многоэкстремальным пространством поиска биостохастические алгоритмы (БСА) выполняют более исчерпывающее исследование этого пространства по сравнению с детерминированными методами поиска. Однако масштабируемость алгоритма предполагает, что его вычислительные затраты растут прямо пропорционально увеличению объема обрабатываемых данных, что алгоритм способен выдать наилучшее на данный момент решение в любое время, даже если процесс

вычислений не завершен естественным остановом, а также работать в пределах ограниченного объема памяти компьютера.

В этой связи требуется анализ масштабируемости БСА и улучшение следующих показателей их эффективности. Во-первых, для улучшения масштабируемости необходимо снижать число вычислений целевой функции. Во-вторых, необходимо снижать вычислительную сложность операций, не связанных с вычислением целевой функции, поскольку при обработке БСА больших данных требуется увеличивать размеры популяций и другие численные параметры, которые входят в оценки вычислительной алгоритмов. В-третьих, необходимо повышать эффективность использования БСА параллельных и распределенных вычислительных систем, поскольку несмотря на определенные возможности по распараллеливанию, большинство известных БСА включают базовый цикл вычислений, включающий выполнение некоторых однопоточных действий после получения результатов вычисления функции приспособленности для особей популяции. Иными словами, для улучшения масштабируемости БСА целесообразно снижать как время ожидания других потоков, так и трудоемкость выполнения критических операций.

Предлагаемый подход направлен на обеспечение разнообразия популяций, чтобы повысить производительность БСА при решении оптимизационных задач большой размерности.

Бурный рост данных в таких областях как биомедицина, инженерия, финансы и социальные науки является результатом технологической революции. Источниками больших данных выступают данные с измерительных устройств, события от радиочастотных идентификаторов, потоки сообщений из социальных сетей, метеорологические данные, данные дистанционного зондирования Земли, потоки данных о местонахождении абонентов сетей сотовой связи, от устройств аудио- и видео регистрации. Однако традиционные технологии обработки больших данных при десятках тысяч объектов, извлеченных из документов и изображений, испытывают известные пространственно-временные проблемы. Разреженность данных – еще одна особенность, связанная с большими данными.

Как влияет рост размерности задач оптимизации на производительность, вычислительные затраты, устойчивость работы БСА?

Какие из существующих подходов используются в БСА для преодоления указанных проблем?

В существующих БСА среднее значение функций приспособленности особей в популяции постепенно увеличивается, популяция теряет разнообразие, а поиск сходится к глобальному или локальному оптимуму. Поэтому в некоторых работах предлагается увеличить разнообразие популяций, генерируемых БСА [36]. Однако этого недостаточно. Более веская причина сходимости к субоптимальным решениям, заключается в увеличении среднего значения целевой функции при генерации новых популяций решений. Даже решения (особи в популяции) с конкурентным значением целевой функции, но расположенные в отдаленных окрестностях поискового пространства, из-за их редкого появления могут исчезнуть из популяции в результате дрейфа. Эта проблема на порядок выше в случае многомерных задач с многоэкстремальными функциями из-за огромного объема пространства поиска.

В других работах [37] предлагается контролировать количество особей в локальных поисковых пространствах, избегая избыточности и одновременно контролируя среднее значение функции приспособленности в популяции особей. В [38] предлагается генерировать несколько субпопуляций, контролируя миграцию особей из одной субпопуляции в другую. В [39] для решения проблемы масштабируемости предлагается использовать метод дифференциальной эволюции, в [40] – метод кооперативной коэволюции.

Однако эти подходы и предлагаемые в них БСА с точки зрения масштабируемости требуют использования нереалистично больших популяций или субпопуляций, чтобы эффективно справляться с «проклятием размерности». Например, в работе [41] использовалась популяция из 350 000 особей. При этом увеличение размеров популяции не гарантировало улучшение результатов.

Коэволюционный подход к сокращению размеров популяции в генетическом алгоритме предлагался в [42]. Его идея состоит в декомпозиции многомерной задачи на низкоразмерные задачи с последующей их оптимизацией и комбинированием. Однако попытки применения данного подхода к несепарабельным задачам оказались затруднительными.

Предлагается более простой подход к решению проблемы высокой размерности при разумной поддержке разнообразия популяции. Гипотеза состоит в том, что вначале определяется тенденция

к локальной сходимости БСА, а затем поддерживается разнообразие популяции путем замены избыточных особей из локальных кластеров на особи из неисследованных областей пространства решений. При этом с увеличением размерности задачи не происходит существенного роста размеров популяции и поддерживается разнообразие популяции.

Идея предлагаемого масштабируемого БСА заключается в объединении преимуществ иерархической структуры популяции и специализированного оператора мутации для поддержки разнообразия популяции и использованию перспективных областей поискового пространства.

Псевдокод алгоритма БСА имеет следующий вид [43]:

Procedure БСА

1: *begin*

2: t=0

3: Инициализация популяции $P(t)$

4: Оценка особей популяции $P(t)$

5: *while* (проверка условия останова)

6: *begin*

7: $t := t + 1$

8: *Call Procedure* (Построение иерархической структуры популяций)

9: *Call Procedure* (Выполнение когнитивного оператора мутации)

10: Создание новой популяции с помощью механизма поддержки разнообразия популяции, и когнитивного оператора мутации

11: Оценка особей популяции $P(t)$

14: *end while*

15: *for* (каждая особь в популяции) *do*

16: *begin*

17: Миграция особей из основной популяции

18*: end for*

19: *for* (каждая из n предварительно отобранных субпопуляций)

20: *begin*

21: Популяция эволюционирует заданное количество раз

22: Миграция особей

23: *end*

24: *end*

Отличительными особенностями БСА является построение иерархических субпопуляций и выполнение когнитивного оператора мутации.

Псевдокод процедуры построения иерархических субпопуляций имеет следующий вид:

1: *begin*

2: Вычисление плотности кластеров в иерархии D_{BI}, где $D_{BI} = N_{BI}/N_{POP}$; N_{BI} и N_{POP} –количество особей в кластере *BI* и размер популяции, соответственно.

3: *for* (каждый кластер из *BI*)

4: *begin*

5: *if* (D_{BI} > Порог α) *then*

6: Определение центра кластера BI_{HUB}

7: *end for*

8: *for* (каждый отмеченный кластер BI_{HUB})

9: *while* (Количество особей в кластере меньше порогового значения $N_{\text{порог}}$) *AND* (не меньше, чем при пересечении с любым отмеченным кластером)

10: Маркировка найденного кластера *K*

11: *end while*

12: *end for*

13: *Return* (список найденных кластеров *K*)

14: *end*

Согласно этой процедуре в процессе поиска отдельные особи перемещаются из основной популяции в иерархические субпопуляции. Эти субпопуляции поддерживают основную популяцию в определенных диапазонах функции приспособленности. Субпопуляции формируются случайно и имеют определенный ограниченный размер.

Псевдокод процедуры выполнения когнитивного оператора мутации имеет следующий вид:

1: *begin*

2: Ввод списка найденных кластеров *K*

3: *for* (каждый кластер *K*)

4: *begin*

5: *if* (стандартное отклонение функции приспособленности для кластера *K* среднем не больше значения стандартного отклонения

функции приспособленности для всех кластеров) *AND* (плотность D_{CR} кластера не меньше плотности всех кластеров) *then*

6: *begin*

7: Произвольная выборка N особей из малоисследованных областей и замена ими особей с наихудшей приспособленностью из выявленного кластера M

8: *end if*

9: *else if* (среднее значение приспособленности кластера K больше среднего значения приспособленности всех кластеров) *then*

10: *begin*

11: Определение лучшей особи, которая передается в новой популяции без изменений

12: *end else if*

13: *end for*

14: Применение оператора мутации ко всей популяции $P(t)$

15: Применение оператора рекомбинации ко всей популяции $P(t)$

16: *Return* $P(t)$

17: *end*

Когнитивный оператор мутации поддерживает разнообразие популяции и применяется только к особям основной популяции. Согласно БСА, прежде чем принять решение о применении когнитивного оператора мутации, необходимо определить кластеры, в которых наблюдается сходимость к локальному оптимуму. Наименее приспособленные особи из этих кластеров заменяются на перспективные особи из малоисследованных областей поискового пространства.

Для тестирования производительности БСА использовался набор многомерных (от 20 до 1000 переменных) функций-бенчмарок.

Вначале оценим эффективность БСА для задач размерности n=20 переменных. Далее, алгоритм тестируется на функциях размерности n=50, n=100 и т. д. переменных. Показатели БСА сравнивались с показателями следующих конкурирующих алгоритмов: стандартным эволюционным алгоритм (*SEA*), самоорганизующимся эволюционным алгоритмом (*SOCEA*), клеточным эволюционным алгоритмом (*CEA*), эволюционным алгоритмом с управляемым разнообразием популяции (*DGEA*), подробное описание которых представлено в [44–47].

В качестве тестовых функций использовались следующие многомерные функции-бенчмарки [43]:

• Гриванка $F_{gri}(X) = \frac{1}{40000}\sum_{i=1}^{n}(x_i - 100)^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i-100}{\sqrt{i}}\right) + 1$, где $-600 \le x_i \le 600$;

• Растригина $F_{rtg}(X) = \sum_{i=1}^{n}[x_i^2 - 10cos(2\pi x_i) + 10]$, где $-5.12 \le x_i \le 5.12$;

• Розенброка $F_{ros}(X) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$, где $-100 \le x_i \le 100$;

• Швефеля $F_{sch}(X) = 418{,}9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$, где $-400 \le x_i \le 400$.

Бенчмарки выбирались по следующим принципам:

• оптимизируемые функции должны быть непохожими;

• вызывать затруднения у точных методов оптимизации;

• быть нелинейными, несепарабельными, масштабируемыми.

Указанные выше функции удовлетворяют этим принципам. Ниже, в качестве иллюстрации, приводятся графики этих функций в двумерном случае (рисунки 2.7–2.10).

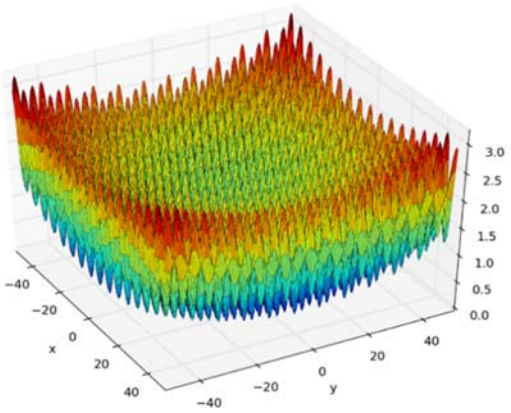


Рис. 2.7. График функции Гриванка

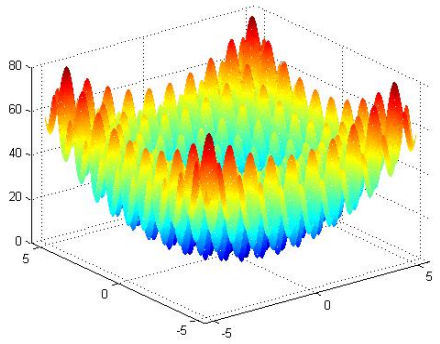


Рис. 2.8. График функции Растригина

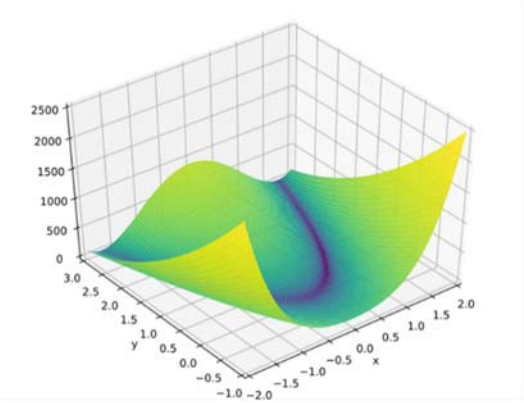


Рис. 2.9. График функции Розенброка

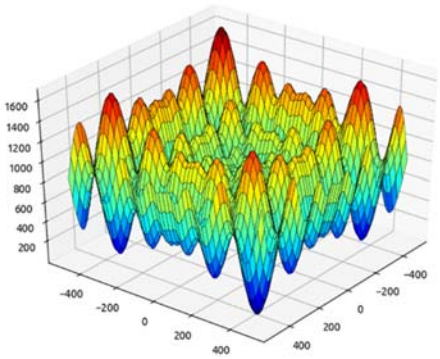


Рис. 2.10. График функции Швефеля

Глобальный минимум этих функций: $F_{gri}(X = 0) = 0$, $F_{rtg}(X = 0) = 0$, $F_{ros}(X=1) = 0$, $F_{sch}(X = 0) = 0$.

В экспериментах использовались следующие настройки: размер основной популяции $N = 250$, вероятность мутации $p_m = 0.05$, вероятность кроссинговера $p_c = 0.9$. Полученные результаты усреднялись по 30 независимым прогонам. Максимальное число поколений в каждом прогоне составляет 500, 1000 и 2000 для числа переменных n=20, n=50 и n=100.

Эмпирические результаты, полученные алгоритмом БСА на четырех функциях-бенчмарках, представлены в таблице 2.1.

Таблица 2.1

Значения ошибок на тестовых функциях для алгоритма БСА

	$F_{gri}(X)$	$F_{rtg}(X)$	$F_{ros}(X)$	$F_{sch}(X)$
$n = 20$				
лучший	4.00E-62	1.01E-61	0.50E-60	1.05E-50
средний	4.91E-62	1.95E-61	1.11E-60	1.15E-50
худший	8.11E-62	3.00E-61	1.92E-60	2.29E-50
$n = 50$				
лучший	4.73E-40	1.10E-40	1.11E-40	1.03E-30
средний	4.81E-40	1.10E-40	1.14E-40	1.04E-30
худший	5.10E-40	1.30E-40	1.20E-40	1.19E-30
$n = 100$				
лучший	8.92E-21	1.21E-20	1.20E-20	1.09E-15
средний	8.93E-21	1.21E-20	1.20E-20	1.09E-15
худший	8.95E-21	1.22E-20	1.21E-20	1.68E-15

В таблице указаны значения ошибок ($F(X) – F(X)$*), где $F(X)$* – значение глобального минимума функции. Каждый столбец соответствует функциям Гриванка, Растригина, Розенброка и Швефеля.

Значения ошибок за 30 прогонов в порядке возрастания (лучшее, среднее, худшее) представлены для n=20, n=50 и n=100. Видно, что алгоритм БСА показывает устойчивые результаты при разной размерности в различных прогонах моделирования. Это показатель надежности работы алгоритма.

В таблице 2.2 представлены лучшие результаты конкурирующих алгоритмов *SEA*, *SOCEA*, *CEA*, *DGEA* и алгоритма БСА.

Таблица 2.2

Результаты сравнения алгоритмов
SEA, *SOCEA*, *CEA*, *DGEA* и БСА

Алгоритм		*SEA*	*SOCEA*	*CEA*	*DGEA*	БСА
$n=20$	$F_{gri}(X)$	1.171	0.930	0.642	7.88E-8	4.00E-62
	$F_{rtg}(X)$	11.12	2.875	1.250	3.37E-8	1.01E-61
	$F_{rosi}(X)$	8292.3	406.5	149.06	8.127	0.50E-60
	$F_{sch}(X)$	–	–	–	–	1.50E-50
$n=50$	$F_{gri}(X)$	1.616	1.147	1.032	1.19E-3	1.11E-40
	$F_{rtg}(X)$	44.674	22.46	14.22	1.97E-6	1.00E-40
	$F_{ros}(X)$	41426	4783.3	1160	59.789	1.00E-40
	$F_{sch}(X)$	–	–	–	–	1.90E-30
$n=100$	$F_{gri}(X)$	2.250	1.629	1.179	3.24E-3	1.01E-21
	$F_{rtg}(X)$	106.21	86.364	58.38	6.56E-5	1.01E-20
	$F_{ros}(X)$	91251	30428	6054	880.324	1.00E-20
	$F_{sch}(X)$	–	–	–	–	1.00E-15

С помощью *t*-критерия Стьюдента (уровень значимости 0.05, достоверность 95%) проведена проверка того, являются ли различия в результатах (значения оптимизируемой функции) для предлагаемого биоинспирированного масштабируемого алгоритма статистически значимыми по сравнению с конкурирующими алгоритмами.

Сравнение показало, рассчитанные значения *t*-критерия превышают соответствующие критические значения, указанные в специальных таблицах. Следовательно, наблюдаемые различия являются статистически значимыми. Статистически значимые различия свидетельствуют в пользу масштабируемого БСА алгоритма для всех рассмотренных функций-бенчмарок, особенно с возрастанием размерности задачи.

Отметим, что разработанный масштабируемый БСА алгоритм, использующий иерархический мультипопуляционный подход и специальные операторы для поддержки разнообразия популяции решений, расширения области поиска решений за счет менее перспективных решений, является достаточно перспективным при решении задач многомерной оптимизации со сложными мультимодальными пространствами решений и большими данными. Оценка эффективности предложенного алгоритма производилась на наборе многомерных оптимизационных функций-бенчмарок: Гриванка, Растригина, Розенброка, Швефеля. Показатели разработан-

ного алгоритма сравнивались с показателями четырех конкурирующих алгоритмов: стандартным эволюционным алгоритмом, саморганизующимся эволюционным алгоритмом, клеточным эволюционным алгоритмом, эволюционным алгоритмом с управляемым разнообразием популяции. Эксперименты свидетельствуют в пользу масштабируемого БСА алгоритма, причем различия в значениях оптимизируемых функций для него являются статистически значимыми по сравнению с конкурирующими алгоритмами, в особенности с возрастанием размерности задачи. На наш взгляд, это объясняется возможностями масштабируемого БСА алгоритма поддерживать разнообразие популяции и находить баланс между скоростью сходимости алгоритма и диверсификацией поиска.

ГЛАВА 3. ПАРАЛЛЕЛИЗМ МЯГКИХ ВЫЧИСЛЕНИЙ И НЕЙРОЭВОЛЮЦИОННЫЙ АЛГОРИТМ

3.1. Параллелизм мягких эволюционных вычислений

Создание многоядерных структур процессоров, недорогих кластерных решений на основе стандартных компонентов, массовое использование распределенных вычислений отражают тенденцию к параллелизму в различных его проявлениях. Необходимость в параллелизме объясняется существованием задач, требующих большого количества ресурсов памяти, и процессорного времени, а также тем обстоятельством, что индустрия упёрлась в технологический барьер и дальнейшее увеличение тактовой частоты затруднено. Проблема компенсируется параллельным программированием работы компонентов процессора и многоядерностью на кристалле. Особенностью параллельного программирования является недетерминированное поведение многопроцессорной системы и необходимость знания деталей архитектуры системы. В частности, разработчику параллельной программы необходимо сформировать фрагменты программ для исполнения на процессорах, закодировать их, организовать взаимодействие и обмен данными.

В таких условиях особую ценность представляют алгоритмы, допускающие возможность распараллеливания по нескольким процессорным элементам или компьютерам. К ним относятся биостохастические алгоритмы (БСА), которые обладают большим потенциалом для распараллеливания и представляют собой мягкие вычисления, трансформирующие входной поток информации в выходной и основанные на правилах имитации механизмов в искусственных или естественных системах, а также на статистическом подходе к исследованию пространства поиска оптимальных решений. Под термином «биостохастические» здесь понимается совокупность принципов и подходов, моделирующих некоторые поведенческие процессы в системах и направленных на поиск такого способа распределения ресурсов, при котором обеспечивается минимальное или максимальное значение критериального показателя. Все БСА представляют собой компромисс между рандомизацией и локальным поиском. Иными словами, с одной стороны, генерируются разнообразные решения в глобальном пространстве поиска, с другой стороны, фокусируется поиск в локальной области.

Параллелизация БСА в гетерогенных компьютерах сетях или на параллельных высокопроизводительных кластерных вычислителях может обеспечить значительные выгоды с точки зрения производительности и масштабируемости [48]. Поэтому актуальной является проблема их переноса в распределенную вычислительную среду.

Для этого имеются следующие веские основания. Все разновидности БСА обладают свойством параллелизма вследствие наличия механизмов поддержания разнообразия в популяции на протяжении всего поиска в распределенном пространстве решений [5]. Практическая реализация параллелизма БСА позволит всесторонне исследовать пространство решений, что, в частности, является ключевым моментом для решения трудных проблем многоэкстремальной оптимизации, свойственной множеству практических задач из различных прикладных областей.

Уточним основные понятия и способы измерения параллелизма.

Параллелизм в информатике и вычислительной технике означает возможность одновременного выполнения нескольких операций. Особое значение параллельная обработка информации приобретает для выполнения алгоритмов искусственного интеллекта. К таким алгоритмам относятся БСА. Они связаны с комбинаторным поиском решений в огромном пространстве и требуют большой вычислительной мощности.

Однако на стадии постановки задачи параллелизм не определен, он появляется только после выбора алгоритма вычислений и может меняться в довольно больших пределах. Параллелизм используемой компьютерной системы также меняется в широких пределах, но он зависит от числа процессоров, способа размещения данных, методов коммутации и способов синхронизации процессов. Наконец, средством переноса параллелизма алгоритма на параллелизм компьютерной системы является язык программирования, тип которого может в сильной степени влиять на результат переноса.

Одновременное выполнение операций возможно, если они логически независимы. Таким образом, описание зависимостей операций по данным полностью определяет параллелизм алгоритма вычислений. При каких условиях программные объекты (команды, операторы, программы) не являются независимыми и не могут выполняться параллельно?

Если для программных объектов V, W обозначить через I_V, I_W наборы их входных переменных, а через O_V, O_W – наборы их выходных переменных, то следующие три условия отражают различные виды зависимостей между программными объектами:

- $I_V \cap O_W \neq \varnothing$ (прямая зависимость);
- $I_w \cap O_V \neq \varnothing$ (обратная зависимость);
- $O_V \cap O_W \neq \varnothing$ (конкурентная зависимость).

Примером прямой зависимости программных объектов является V: $R = D*Pi/2$, W: $S = Pi*R*R$. Здесь операторы V и W не могут выполняться одновременно, так как результат V является операндом W.

Примером обратной зависимости программных объектов является V: $A = B + R$, W: $B = C + D$. Здесь операторы V и W не могут выполняться одновременно, так как выполнение W вызывает изменение операнда в V.

Примером конкурентной зависимости программных объектов является V: $R = A + B$, W: $R = C + D$. Здесь одновременное выполнение операторов V и W дает неопределенный результат.

Поиск и устранение указанных зависимостей означает увеличение параллелизма алгоритмов. Однако для сравнения параллельных алгоритмов необходимо уметь оценивать степень параллелизма.

Степень параллелизма алгоритма – это число его операций, которые могут выполняться параллельно. Например, если требуется сложить два вектора X и Y длины n, то сложения $x_i + y_i$ независимы и могут выполняться параллельно. Степень параллелизма здесь равна n, она является характеристикой параллелизма алгоритма и напрямую не связана с числом процессоров вычислительной системы. От числа процессоров зависит время, необходимое для завершения вычислений. Например, если $n = 1024$ и число процессоров p также равно 1024, то все суммы условно можно вычислить за один временной шаг, однако при $p = 16$ потребуется 64 временных шага.

Несколько иначе обстоит дело, если решается задача сложения (или умножения, или поиска максимума/минимума) n чисел $a_1, a_2, \ldots, a_n$. Обычный последовательный алгоритм вида $s = a_1$, $s = s + a_i$, $i = 2, \ldots, n$ оказывается малопригодным для параллельных вычислений. Однако задача допускает распараллеливание вычислений, если независимо производить суммирование пар чисел. Так, если $n = 2^q$, то алгоритм включает $q = log_2 n$ этапов; причём на

первом этапе выполняются $n/2$ сложений (степень параллелизма равна $n/2$), на втором – $n/4$ и т.д., пока на последнем этапе не будет выполнено единственное сложение.

Средняя степень параллелизма алгоритма – это отношение общего числа операций алгоритма к числу его этапов. Для алгоритма попарного суммирования (сдваивания) средняя степень параллелизма равна $(n/2 + n/4 + \ldots + 1)/q = (2^q - 1)/q = (n - 1)/log_2 n = O(n/log_2 n)$. Средняя степень параллелизма при сложении двух векторов длины n совпадает со степенью параллелизма, т. е. равна n. Это случай «идеального» параллелизма алгоритма, тогда как при сложении n чисел по алгоритму сдваивания средняя степень параллелизма становится в $log_2 n$ раз меньше идеальной. Однако это неплохо по сравнению с последовательным алгоритмом, для которого средняя степень параллелизма равна $(n - 1)/(n - 1) = 1$.

В [5] был установлен факт наличия базового цикла в БСА. Базовый цикл БСА включает следующую последовательность шагов: вычисление целевой функции, её оценку, селективный отбор решений и репродукцию новых решений. Цикл обладает свойством массового параллелизма при обработке информации, как на уровне организации работы алгоритма, так и на уровне его компьютерной реализации, что предполагает эффективную параллельную реализацию процесса вычислений.

Параллелизм на уровне компьютерной реализации БСА означает вычисление на параллельных системах или процессорах значений целевой функции для разных решений, параллельное выполнение операторов, что пропорционально повышает скорость работы алгоритма. Параллелизм на уровне организации работы БСА достигается структурированием популяции решений.

Организация распараллеливания БСА на базе распределенной вычислительной системы позволяет существенно сократить время, затрачиваемое на решение задачи, как за счет параллельного выполнения вычислений, так и за счет применения более эффективных, чем в последовательном случае, способов реализации алгоритмов.

Оба подхода эффективно реализуются на параллельных вычислителях. Однако требуется особая организация БСА путём логического разбиения алгоритма на ортогональные по отношению к обрабатываемым данным гранулы (зерна) параллелизма с минимизацией числа и объемов обменов между процессорами, чего практи-

чески никогда не требуется при классической общей шине вследствие ее огромной производительности и малой инерционности [49].

Гранула параллелизма – это последовательности инструкций процессора, не требующих выполнения обменов данными с соседними процессорами. Выделение гранул параллелизма позволяет избежать чрезмерных потерь на простой процессоров во время обмена данными и, как следствие, катастрофического снижения вычислительной мощности. Иными словами, гранула параллелизма – это участок алгоритма, выполняющийся на отдельном процессоре и не требующий при своём исполнении данных от других процессоров. Выявление в алгоритме гранул, которые могут выполняться независимо и параллельно, является одной из главных задач при разработке параллельных БСА.

Модели и способы организации параллелизма могут быть различными: глобальный параллелизм, миграционная модель, диффузионная модель [5]. Ответ на вопрос о том, какой из этих способов организации распределенных БСА является более эффективным, зависит от архитектуры вычислительной системы, используемой для реализации БСА. Решающим обстоятельством для оценки эффективности является то, насколько успешно решается та или иная проектная задача с точки зрения качества решения и вычислительной сложности.

Рассмотрим модель глобального параллелизма, представленную на рисунке. 3.1.

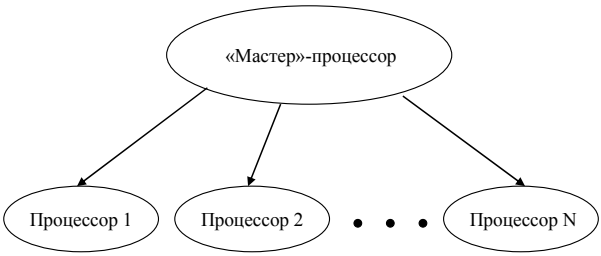


Рис. 3.1. Модель глобального параллелизма

В этой модели основной гранулой параллелизма или участком БСА, выполняющимся на отдельном процессоре и не требующим

при своём исполнении данных от других процессоров, являются такие операции базового цикла БСА как расчёт целевой функции и параллельное выполнение операторов репродукции новых решений. Целевая функция отдельного индивидуума в популяции, как правило, не зависит от других индивидуумов. Поэтому вычисление целевых функций допускает распараллеливание. При этом популяция хранится в общей памяти, отдельный процессор считывает код решения (например, хромосому в генетическом алгоритме) из памяти и возвращает результат после вычисления целевой функции. Процедура синхронизации здесь проводится лишь при формировании новой популяции. В случае если вычислительная система имеет распределенную память, то популяция и значения функции качества отдельных индивидуумов хранятся в «мастер»-процессоре, а расчеты производятся на вспомогательных процессорах.

При использовании модели глобального параллелизма трудность для распараллеливания при реализации базового цикла БСА представляет процедура селективного отбора решений. Здесь преимущество в смысле распараллеливания получают те формы селекции, которые не используют глобальную статистику о популяции, например «соревновательные» способы селекции. Напротив, легко распараллеливаемыми являются такие наиболее широко применяемые операторы (мутация и кроссинговер), так как они выполняются независимо друг от друга и в основном случайным образом.

Если между отдельными подпопуляциями установить правила миграции отдельных индивидуумов, то получится миграционная модель, показанная на рисунке 3.2.

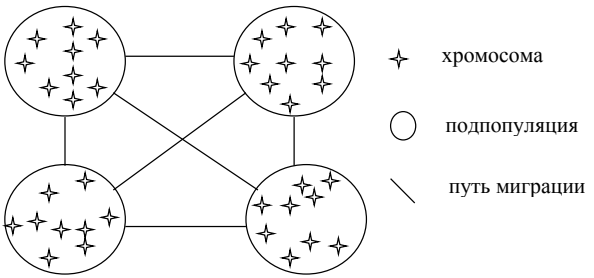


Рис. 3.2. Миграционная модель параллелизма

В этой модели основной гранулой параллелизма являются все операции базового цикла БСА, выполняемые внутри подпопуляции. Для построения миграционной модели требуется задать количество и размеры подпопуляций, топологию связей между ними, частоту и интервал миграции, а также стратегию эмиграции и иммиграции. Размеры и число подпопуляций определяются числом и быстродействием используемых процессоров или транспьютеров. Что касается топологии связей в миграционной модели, то можно применять различные варианты: «островную» модель, предусматривающую миграцию между любыми подпопуляциями; модель, предусматривающую обмен решениями между соседними подпопуляциями и др. Применение того или иного варианта миграционной модели зависит от архитектуры параллельного вычислителя (сетевая, кольцевая или др.). Частота миграции определяется размерами подпопуляций. При высокой частоте подпопуляция может быстро потерять гетерогенность. Наоборот, при слабой интенсивности миграции и большом числе подпопуляций возрастает вероятность получения нескольких хороших решений, однако сходимость алгоритма замедляется.

Для обеспечения выбора особей для миграции предлагается использовать общий буфер хромосом. Буфер заполняется хромосомами нескольких популяций. При наступлении определенной ситуации отдельная популяция обращается к буферу и считывает оттуда часть или все хромосомы, добавляя в буфер часть своих хромосом. В ходе эволюционных вычислений (ЭВ) необходимо лишь определить механизм регулирования размера буфера. Работа параллельного БСА при этом происходит следующим образом. Каждая популяция эволюционирует отдельно от других. На каждой итерации проверяется условие необходимости миграции.

Миграционная модель наиболее эффективно реализуется в вычислительных системах с *MIMD*-архитектурой (множественный поток команд и множественный поток данных) путем разделения общей популяции на отдельные подпопуляции, эволюционное моделирование которых осуществляется на отдельных процессорах.

При разных запусках БСА популяция может сходиться к разным субоптимальным решениям. Данная модель позволяет запустить алгоритм сразу несколько раз и совместить «достижения» разных «островов» для получения наилучшего решения.

Если большую популяцию разделить на множество немногочисленных подпопуляций, а композицию эволюционных операторов применять лишь в ограниченной области, определяемой отношением соседства, то получится диффузионная модель эволюционных вычислений, представленная на рисунке 3.3.

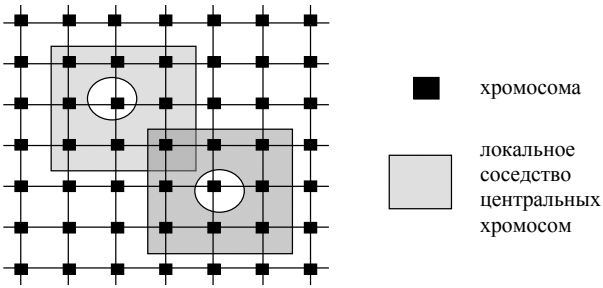


Рис. 3.3. Диффузионная модель параллелизма

В этой модели основной гранулой параллелизма являются все операции базового цикла БСА, выполняемые внутри небольших по размеру подпопуляций. Это приводит к тому, что хорошие решения медленно распространяются в популяции (диффузия) и проблема попадания в «локальную яму» становится менее острой [50].

Популяция в природе, как правило, состоит из отдельных, более или менее изолированных локальных подпопуляций, допускающих диффузию отдельных индивидуумов. Большинством существующих БСА это обстоятельство игнорируется, алгоритмы работают с одной большой неструктурированной популяцией.

Другое преимущество диффузионной модели состоит в том, что она также допускают эффективную реализацию БСА на массиве параллельных вычислителей с различной коммутационной структурой.

Повышение вычислительной мощности не только отдельного процессора, но и путем параллелизации вычислений на многих процессорах или вычислительных узлах, одновременно выполняющих программу, считается одним из наиболее перспективных направлений [51]. Однако процесс разработки параллельных программ существенно более труден, чем обычных последовательных, как и количественная оценка их эффективности с точки зрения соответствия созданного программного обеспечения архитектурным особенностям многопроцессорной вычислительной системы. Как

отмечалось выше, требуется особая организация вычислений путём выделения в алгоритме гранул (зерен) параллелизма, ортогональных по отношению к обрабатываемым данным. Это позволяет минимизировать число и объем обменов между процессорами.

Сейчас большой популярностью пользуются многоядерные и кластерные архитектуры вследствие их значительной масштабируемости и конкретной реализации в виде вычислительных кластеров. В архитектуре этих вычислителей узлы близко связаны (по диаметру сети и ее быстродействию). Напротив, в системах метакомпьютинга допускается расположение узлов на значительных расстояниях, причем узлы, обычно гетерогенны. Соответственно приемы параллельного программирования для указанных систем количественно разнятся.

Относительно большое время обмена данных в системах метакомпьютинга требует реализации значительно большего размера гранулы параллелизма (крупнозернистый параллелизм и модель глобального или миграционного параллелизма), чтобы избежать чрезмерных потерь, связанных с простоем процессоров во время обмена данными, а также катастрофического снижения вычислительной мощности. В общем случае производительность многопроцессорной структуры зависит от производительности отдельных процессоров, от параметров коммуникационной среды, от класса решаемых задач, а также от конкретных технологий гранулляции алгоритмов.

Именно в этом контексте часто говорят о мелкозернистом и крупнозернистом параллелизме. Представленная выше миграционная модель параллелизма относится к крупнозернистому, а диффузионная модель – к мелкозернистому параллелизму. Принадлежность к тому или иному виду параллелизма, в основном, зависит от параметров коммуникационной сети, обеспечивающей обмен сообщениями между процессорами.

В большинстве случаев не удается полностью совместить обмен данными с вычислениями внутри гранулы. Поэтому с целью определения показателей эффективности работы параллельных алгоритмов разделим общее время обработки данных в ходе выполнения БСА на время обмена данными $t_{обмена}$ и время выполнения гранулы параллелизма $t_{гранула}$. Их отношение будет определять показатель гранулярности параллельной программы. Сокращение времени $t_{обмена}$ представляет собой в основном аппаратную задачу, а

вот повышение времени $t_{гранула}$ является алгоритмической задачей и достигается именно на этапе распараллеливания БСА.

Таким образом, отношение $t_{гранула} / t_{обмена}$ указывает на эффективность БСА и применяемой для решения задачи вычислительной структуры. Это отношение также позволяет определить круг задач, эффективно решаемых на данной вычислительной структуре, принять решение о ее модернизации или модернизации параллельной программы БСА. Если отношение $t_{гранула} / t_{обмена} \leq 1$, то это мелкозернистый параллелизм, если отношение $t_{гр} / t_{об.} >> 1$, то это крупнозернистый параллелизм.

Как уже отмечалось, выявление в алгоритме гранул, выполняемых независимо и параллельно, является одной из основных задач при разработке эффективных БСА. Однако загрузка ядер вычислительной работой не является самоцелью. Она является только средством улучшения таких показателей и характеристик БСА как время работы, качество поиска и т. д.

В качестве основного показателя эффективности работы гранулированного БСА примем следующее отношение:

$$K_{эфф} = \frac{T_{посл}}{T_{гран}},$$

где $T_{посл}$ и $T_{гран}$ обозначает время выполнения последовательного и гранулированного алгоритма соответственно. Все вычисления в процессе выполнения алгоритма можно разделить на последовательные (s), гранулированные (γ) вычисления и накладные расходы по организации параллелизма (P). Тогда для задачи размерности N, решаемой на параллельном вычислителе с M процессорами, имеем следующую зависимость:

$$K_{эфф}(N,M) = \frac{s(N) + \gamma(N)}{s(N) + \frac{\gamma(N)}{M} + P(N,M)}.$$

Эта оценка указывает на верхнюю границу для максимального ускорения, которое может быть достигнуто на параллельном компьютере с M процессорами, если объём вычислений равномерно распределён между процессорами.

Согласно закону Амдала [5], ускорение выполнения программы за счет распараллеливания её инструкций на множестве вычисли-

телей ограничено временем, необходимым для выполнения её последовательных инструкций. Предположим, что необходимо решить некоторую задачу. БСА решения этой задачи таков, что некоторая доля α от общего объёма вычислений может быть получена только последовательными расчётами, а, соответственно, доля $(1 - \alpha)$ может быть распараллелена идеально (то есть время вычисления будет обратно пропорционально числу задействованных узлов M). Тогда ускорение, которое может быть получено на вычислительной системе из M процессоров, по сравнению с однопроцессорным решением не будет превышать величины

$$K_{эфф}(M) = \frac{M}{\alpha(M-1)+1}.$$

Таким образом, по закону Амдала только алгоритм, совсем не содержащий последовательных вычислений ($\alpha = 0$), позволяет получить линейный прирост производительности с ростом количества вычислителей в системе. При доле последовательных вычислений α общий прирост производительности не может превысить величины $1/\alpha$. Если половина программного кода БСА – последовательная, то общий прирост никогда не превысит двух. Закон Амдала показывает, что прирост эффективности вычислений зависит от алгоритма задачи и ограничен сверху для любой задачи с величиной $\alpha \neq 0$. Иными словами, не для всякой задачи имеет смысл наращивание числа процессоров в вычислительной системе.

Если учесть время передачи данных между узлами вычислительной системы, то график зависимости времени вычислений от числа узлов будет иметь максимум. Это накладывает ограничение на масштабируемость вычислительной системы, то есть означает, что с определенного момента добавление новых узлов в систему будет увеличивать время расчёта задачи.

С учётом этого соотношение для $K_{эфф}(N, M)$ примет следующий вид:

$$K_{эфф}(N,M) = \frac{s(N)+\gamma(N)}{s(N)+\frac{\gamma(N)}{M}+P(N,M)} \le K_{\max эфф}(N,M) = \frac{s(N)+\gamma(N)}{s(N)+\frac{\gamma(N)}{M}}.$$

Эту оценку можно рассматривать в качестве теоретической границы максимального ускорения, которого может достигнуть параллельный БСА.

Будем различать параллелизм на уровне алгоритмов, итераций и решений. Параллелизм на алгоритмическом уровне предполагает

использование различных БСА. Параллелизм на уровне итераций означает, что для ускорения времени поиска распараллеливается каждая итерация алгоритма. Параллелизм на уровне решений предусматривает распараллеливание наиболее трудоемкой операции в БСА – вычисление значения целевой функции.

При распараллеливании на алгоритмическом уровне различные траекторные и популяционные БСА инициализируются различными начальными решениями, установочными параметрами, поисковыми операторами, способами кодирования, ограничениями, критерии остановки, и т. д. Эта модель параллелизма проста для разработки и реализации на основе модели глобального параллелизма. Определенный процессор реализует некоторый алгоритм, мастер-процессор распределяет между работниками различные БСА и определяет наилучшее из найденных работниками решение. Однако открытым остается следующий вопрос: что лучше выполнить m БСА за время t или выполнить за время t один БСА m раз? Ответ на этот вопрос во многом зависит от ландшафтных свойств целевой функции оптимизационной задачи.

Известна кооперативная модель распараллеливания БСА, предусматривающая обмен информацией в ходе поиска оптимума [5]. Однако при использовании кооперативной модели необходимо: установить протокол обмена информацией между БСА; учесть топологию связей между БСА (кольцо, гиперкуб, полный граф); определить содержание и объем информации для обмена (лучшие решения на текущей итерации, глобальные лучшие решения и пр.); указать порядок использования полученной информации. Для распараллеливания БСА также используются миграционная и диффузионная модели. В миграционной модели каждая популяция решений обрабатывается на отдельном процессоре, между которыми устанавливается определенный протокол миграции решений. В диффузионной модели общая популяция разделяется на большое число немногочисленных субпопуляций, а операторы кроссинговера и селекции применяются лишь в ограниченной области, определяемой отношением соседства. Это приводит к тому, что хорошие решения очень медленно распространяются в популяции (диффузия) и проблема попадания в «локальную яму» становится менее острой.

Большинство БСА являются итерационными алгоритмами, поэтому распараллеливание может проводиться для каждой итерации. Для траекторных БСА (например, локальный поиск, табуированный поиск, поиск в переменной окрестности), генерацию и оценку соседних решений можно выполнять параллельно. Это позволит исследовать окрестности большого размера. Для популяционных БСА параллелизм на уровне итераций возникает естественным образом, так как каждая особь популяции [52; 53].

Модель распараллеливания БСА на уровне решений интересна для задач оптимизации, требующих трудоемких расчетов целевой функции или доступа к большим входным файлам и базам данных.

Модели параллелизма БСА на уровне алгоритма, итераций и решений могут быть использованы совместно. В этом случае степень параллельности гибридной модели будет равна произведению mnk, где m – количество используемых БСА, n – размер популяции или области соседних решений, k - число операций, связанный с оценкой одного решения. Например, если $m = 100$, $n = 50$, и $k = 20$, то степень параллельности будет равна 100000.

Параллельная реализация БСА требует эффективного отображения на параллельной архитектуре и соответствующего программирования.

Параллельные архитектуры классифицируют по количеству потоков команд и данных:

- *SISD* (один процессор выполняет один поток команд, оперируя одним потоком данных). Это класс постепенно исчезающих однопроцессорных компьютеров;
- *SIMD* (одиночный поток команд, множественный поток данных) – популярная в прошлом архитектура, эффективная при синхронизированном выполнении параллельных алгоритмов, состоящая одного командного процессора (контроллера) и нескольких процессоров для обработки данных;
- *MISD* (множественный поток команд, одиночный поток данных) – тип несуществующей на практике архитектуры параллельных вычислений, где несколько функциональных модулей выполняют различные операции над одними данными;

• *MIMD* (множественный поток команд, множественный поток данных) – наиболее общая модель параллельных архитектур, в которой машины имеют несколько процессоров, функционирующих асинхронно и независимо с общей либо с распределяемой памятью.

В разработках параллельных архитектур доминируют два типа: архитектуры с общей и распределенной памятью.

В параллельных архитектурах с общей памятью процессоры соединены с общей памятью, возможно, различными способами (шина, многоступенчатый ригель). Эта архитектура легко программируется, однако плохо масштабируется (от 2 до 128 процессоров в современных технологиях), к тому же программист должен заботиться о синхронизации доступа к общей памяти. При увеличении числа процессоров возрастает время на передачу данных. Примерами параллельных архитектур с общей памятью являются *SMP* (симметричная многопроцессорная машина), а также архитектура с аппаратной поддержкой когерентности кэшей и многоядерные процессоры *Intel, AMD*.

Параллельные архитектуры с распределенной памятью строятся из отдельных узлов, содержащих процессор, локальный банк оперативной памяти, коммуникационные процессоры или сетевые адаптеры, иногда, жёсткие диски и другие устройства ввода-вывода. Доступ к банку оперативной памяти данного узла имеют только процессоры из этого же узла. Узлы соединяются специальными коммуникационными каналами. Главным преимуществом таких архитектур является хорошая масштабируемость: каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров. Практически все рекорды по производительности в 1990-е годы устанавливались на машинах именно такой архитектуры, состоящих из нескольких тысяч процессоров. Однако отсутствие общей памяти заметно снижает скорость межпроцессорного обмена, требуется специальная техника программирования для реализации обмена сообщениями между процессорами; высокая цена программного обеспечения.

Кластерные системы являются логическим продолжением развития идей, заложенных в системах с параллельной архитектурой с распределенной памятью [54]. Развитие коммуникационных технологий, а именно появление высокоскоростного сетевого обору-

дования и специального программного обеспечения, реализующего механизм передачи сообщений над стандартными сетевыми протоколами, сделало кластерные технологии общедоступными. Сегодня не составляет большого труда создать небольшую кластерную систему, объединив вычислительные мощности компьютеров отдельных лабораторий. Кластер – это связанный набор полноценных компьютеров, используемый в качестве единого ресурса. Привлекательной чертой кластерных технологий является то, что они позволяют для достижения необходимой производительности объединять в единые вычислительные системы компьютеры самого разного типа, начиная от персональных компьютеров и заканчивая мощными суперкомпьютерами. Для измерения производительности компьютеров и кластерных систем используется единица *FLOPS*. Данная величина определяется путём запуска на испытуемом компьютере тестовой программы, которая решает задачу с известным количеством операций и подсчитывает время, за которое она была решена.

Грид-системы – это форма распределённых вычислений, в которой «виртуальный суперкомпьютер» представлен в виде кластеров, соединённых с помощью сети, слабосвязанных гетерогенных компьютеров, работающих вместе для выполнения многочисленных задач [55]. Технология грид сегодня применяется для решения научных, математических задач, а также задач экономического прогнозирования, сейсмоанализа, изучения свойств новых лекарств. Грид-системы являются разновидностью параллельных вычислений, основанных на обычных компьютерах, подключенных к сети при помощи обычных протоколов. В то время как обычный суперкомпьютер содержит множество процессоров, подключенных к локальной высокоско-ростной шине. Их преимуществом перед обычным суперкомпьютером является то, что отдельная ячейка вычислительной системы может быть приобретена как обычный неспециализированный компьютер. Таким образом, можно получить практически те же вычислительные мощности, что и на суперкомпьютерах, но с гораздо меньшей стоимостью.

В таблице 3.1 представлены основные характеристики параллельных архитектур в соответствии с указанными критериями.

Эти критерии будут использованы для анализа эффективности параллельной реализации различных моделей БСА.

Таблица 3.1

Характеристики параллельных архитектур

Критерий / ***Параллельная архитектура***	**Память**	**Одно-родность**	**Обмен информацией**	**Сеть**	**Волатиль-ность**
Симметричное мульти-процессирование	общая	да	да или нет	локальная	нет
Кластеры рабочих станций	распре-деленная	да или нет	нет	локальная	нет
Сеть рабочих станций	распре-деленная	нет	да	локальная	да
Высоко-производительные кластеры	распре-деленная	нет	нет	широкая	нет
Кластер	распре-деленная	нет	да	широкая	да

Для последовательных алгоритмов основной мерой производительности является время выполнения в зависимости от размерности задачи. Для параллельных алгоритмов эта мера также зависит от количества процессоров и характеристик параллельной архитектуры. Поэтому для оценки масштабируемости (пропорциональное увеличение объема задачи с увеличением числа используемых для ее решения процессоров) параллельных алгоритмов были введены такие показатели как ускорение и эффективность распараллеливания.

Ускорение – это отношение времени выполнения задачи в последовательном режиме на одном процессоре ко времени выполнения задачи в параллельном режиме на p процессорах:

$$S = \frac{T_{sec}}{T_{par}} \leq \frac{p}{(p-1)f+1},$$

где f – доля последовательного участка в общем объеме вычислений. При $f = 0$ получим $S = p$, при $f > 0$ и $p \to \infty$, получим $S < 1/f$. Следовательно, ни на каком числе процессоров ускорение вычислений не может превысить обратной величины доли последовательного участка.

Эффективность распараллеливания – это способность алгоритма использовать все задействованные в выполнении задачи процессоры на 100%:

$$E = \left(\frac{S}{p}\right) 100\%$$

Если ускорение $S = p$ (максимально возможное на p процессорной машине), то эффективность распараллеливания задачи равна 100%. Например, $E \leq 52{,}25\%$ для p=100, f=0,01 и $E \leq 9.1\%$ для p=1000 и f=0,01. При малой доли последовательной работы увеличение количества процессов приводит к ухудшению параллельной эффективности. Для повышения эффективности, как правило, не распараллеливают управляющие части программы или небольшие участки вычислений, которые требуют интенсивной синхронизации процессов.

Рассмотрим свойства БСА, которые должны быть приняты во внимание для эффективной параллельной реализации.

Производительность БСА на параллельной архитектуре, главным образом, зависит от его зернистости. Зернистость – это мера отношения количества вычислений, сделанных в задаче, к количеству пересылок данных. Зернистость параллельного алгоритма, существенно влияет на коэффициент ускорения и эффективность реализации параллельных вычислений. Чем больше зернистость, тем выше ускорение и эффективность.

Модели параллелизма БСА на уровне алгоритма, итераций и решений имеют убывающую зернистость – крупнозернистые, среднезернистые и мелкозернистые.

Степень согласованности параллельного БСА определяется максимальным числом параллельных процессов, идущих в любой момент времени. Это показатель количества процессоров, которые могут быть использованы в процессе вычислений параллельным БСА.

Планирование и отказоустойчивость процессов, составляющих параллельный БСА, также влияет на эффективность реализации БСА. В частности, статическое планирование является более подходящим для однородных и гетерогенных параллельных архитектур. Поскольку заметные различия между процессорами могут привести к тому, что значительное число задач будут простаивать в ожидании завершения выполнения других задач. Динамическое планирование задач на разные процессоры параллельной архитектуры подходит для многопользовательских архитектур, в них нагрузка каждого процессора не может быть определена во время компиляции. Наконец, адаптивная балансировка нагрузки имеет важное значение для таких архитектур как грид-системы.

Модель параллелизма БСА на уровне алгоритма имеет наибольшую зернистость, поскольку на этом уровне частота обмена информацией наименьшая. Эта модель наиболее подходит для сетевых распределенных архитектур, таких как грид-системы, а также для низкоскоростных Интранет сетей из рабочих станций. Что касается масштабируемости алгоритмической модели параллелизма БСА, то при увеличении числа алгоритмов пропорционально увеличивается число используемых для их решения процессоров. На практике она ограничена лишь эффективностью использования большого числа алгоритмов. Синхронизация коммуникаций для модели параллелизации БСА на алгоритмическом уровне является менее эффективной для сетевых архитектур. В этом случае производительность будет определяться наименее мощной машиной.

Модель параллелизма БСА на итерационном уровне имеет среднюю зернистость. Модель эффективна, если оценка решений отнимает много времени. Масштабируемость этой модели ограничена размерностью пространства соседних решений для траекторных алгоритмов и размером популяции для популяционных алгоритмов. Асинхронность коммуникаций в данной модели повышает эффективность параллельных вычислений. Статическое планирование параллельной оценки решений в данной модели менее эффективно по сравнению с динамическим планированием. Однако более эффективным является адаптивное планирование, особенно для гетерогенных параллельных архитектур, таких как грид-системы. Что касается отказоустойчивости, то неисправность процессора не влияет на выполнение параллельного БСА в гетерогенных сетях.

Модель параллелизма БСА на уровне решений имеет малую зернистость. Это делает модель наименее подходящей для масштабных распределенных архитектур. Масштабируемость модели ограничена количеством функций или разделов данных, требуемых для оценки решений. Хотя, в целом, использование этой модели в сочетании с двумя другими моделями параллелизма позволяет расширить масштабируемость параллельной обработки БСА. Модель требует синхронизации процессов, поскольку мастер-процессор должен ждать получения всех результатов, чтобы выбрать глобальное значение целевой функции. Динамическое планирование для модели является более эффективным, нежели статическое планирование.

В таблице 3.2 представлены рекомендации для эффективной реализации моделей параллелизма БСА на алгоритмическом, итерационном и на уровне решений в зависимости от критериев зернистости, масштабируемости, асинхронности, планирования и отказоустойчивости.

Таблица 3.2

Критериальная оценка моделей параллелизма БСА

Уровень *Свойства*	**Алгоритмический**	**Итерационный**	**Решения**
Зернистость	крупная	средняя	малая
Масштабируемость	число алгоритмов	размер окрестности, популяции	число функций или разделов данных
Асинхронность	значительный обмен информацией	средняя (оценка решений)	исключена
Планирование и отказоустойчивость	алгоритм	решения	части решения

Рассмотрим вопрос о распараллеливании операций базового цикла БСА.

При программировании распараллеливания базового цикла БСА необходимо иметь в виду следующее:

- на параллелизм значительно влияют две составляющие базового цикла алгоритма – оценка качества решения и репродукция новых решений;
- процесс вычисления целевой функции для каждого решения может осуществляться параллельно и независимо от остальных решений;
- оценка качества решений может потребовать сложных расчетов и является одним из наиболее трудоемких этапов БСА, т. к. в процессе оценки требуется сравнить значения функции качества каждого решения с остальными и обновить структуры данных, что затрудняет процесс распараллеливания;
- выполнение фазы селекции в базовом цикле алгоритма может потребовать доступа к определенным подмножествам решений в популяции или обновления данных. Поэтому возможности ее распараллеливания зависят от выбора схемы селекции;

• выполнение фазы репродукции в базовом цикле алгоритма можно распараллелить. Репродукция, являясь независимой задачей, может включать либо копирование решения, либо выполнение унарных, бинарных или *n*-арных операторов БСА.

На рис. 3.4 в общих чертах представлены потенциальные возможности распараллеливания базового цикла БСА [5]

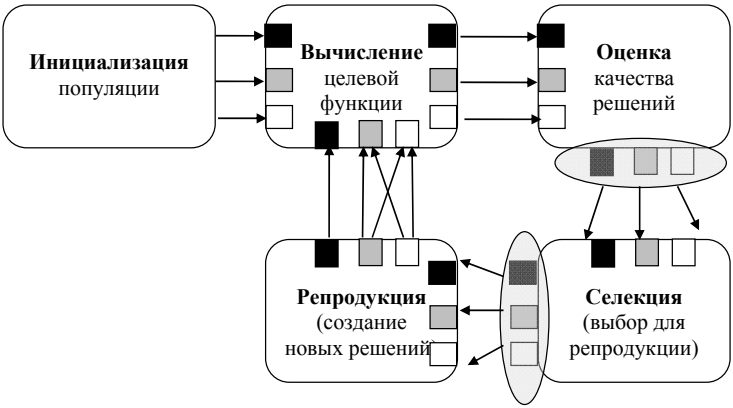


Рис. 3.4. Потенциальные возможности распараллеливания базового цикла БСА

БСА с выделенными гранулами параллелизма будет эффективным только в том случае, если задержки, вызванные передачей и обменом необходимыми данными, будут незначительными. Например, если математическая функция *sqrt*(*x*) вычисляется на одном процессоре, а подкоренное выражение *x* – на другом, то это займет гораздо больше времени, чем локальное вычисление *x* и *sqrt*(*x*).

Наибольшее влияние на скорость БСА, оказывают размеры популяций и хромосом, а также такие особенности целевых функций качества решений как многоэкстремальность (множество ложных аттракторов), «обманчивость», изолированность, дополнительный «шум» [56].

Однако простота идеи программирования параллельных БСА на практике требует выполнения некоторых условий. В частности, в многопроцессорной системе разбиение на слишком крупные гранулы не позволяет равномерно загрузить процессоры и добиться минимального времени вычислений, а излишне мелкая «нарезка»

означает рост непроизводительных расходов на связь и синхронизацию. Параллелизму присуща зависимость структуры алгоритма от топологии вычислительной платформы. Процесс создания максимально эффективного алгоритма практически не автоматизируется и связан с большими трудозатратами на поиск специфической структуры алгоритма, оптимальной для конкретной топологии используемой вычислительной системы. Найденная структура обеспечит минимальное время получения результата, но, скорее всего, будет неэффективна для другой конфигурации. Более суровое следствие зависимости структуры БСА от вычислительной платформы – тотальная непереносимость не только исполняемых кодов, но и самого исходного текста программы.

3.2. Параллельная кластерная обработка продукционных правил в базах знаний интеллектуальных систем

Рассмотрим возможности организации параллельных вычислений и параллельной обработки продукционных правил в базах знаний на примере использования кластерного вычислителя *HP BladeSystem c*-класса.

Этот кластер, относится к системам, обладающим большими вычислительными возможностями, и включает множество специально разработанных технологий, обеспечивающих многократную масштабируемость, возможность быстрой адаптации инфраструктуры под изменяющиеся запросы приложений и адекватную производительность. Важным преимуществом блейд-системы также является плотность компоновки серверной инфраструктуры, не влияющая на производительность оборудования. В случае грамотного развертывания кластеры на базе блейд-серверов могут быть значительно более гибкими и управляемыми, нежели суперкомпьютеры.

Попробуем оценить выигрыш во времени, получаемый при использовании распараллеливания работы БСА. Обозначим через $V(M)$ относительный выигрыш по времени, получаемый при различных вариантах распараллеливания работы БСА; через $L(M)$ число узлов, сгенерированных на M процессорах в течение одного цикла работы алгоритма.

Величину $V(M)$ можно оценивать по-разному. Например, как отношение величины $L(1)/1$ (среднее число узлов, сгенерированных

на однопроцессорной машине в течение одного цикла работы алгоритма) к величине $L(M)/M$ (среднее число узлов, сгенерированных на M процессорах в течение одного цикла работы алгоритма):

$$V(M) = [L(1)/1] / [L(M)/M] = M*L(1)/L(M) = V_1 \qquad (3.1)$$

Соотношение (3.1) предполагает полную загрузку процессоров без учета простоев. Обозначим через $C_G(M)$ фактический номер цикла работы алгоритма. Тогда

$$V(M) = C_G(1)/C_G(M) = L(1)/C_G(M) \qquad (3.2)$$

Пусть t_G – время простоя на однопроцессорной машине, требуемое в течение одного цикла работы алгоритма, а $T_L(M)$ – среднее время, необходимое для балансирования загрузки на M процессорах. Тогда

$$V(M) = [C_G(1)\, t_G]/[C_G(M) \cdot (t_G + T_L(M)/C_G(M))] \qquad (3.3)$$

Эксперименты с 4, 8 и 16 параллельно работающими процессорами показали, что относительный выигрыш во времени лежать в диапазоне от 2/3 до 3/4, причем максимальный выигрыш достигается при использовании процедуры «жадного» перераспределения при динамической балансировке загрузки процессоров [50; 54].

Параллелизм в системе можно расширить за счет распределенного представления знаний [57]. Удобной структурной моделью для описания асинхронных параллельных недетерминированных процессов и систем продукционного типа является *сеть Петри*, которая содержит два типа вершин (позиции и переходы, изображаемые кружками и полочками соответственно), соединяемых дугами [58]. Статически модель сети задается двудольным орграфом; ее исходное состояние – начальной маркировкой некоторых позиций, динамика вносится соглашением о правиле срабатывания переходов, а процесс функционирования сети состоит в переходе от одной маркировки к другой посредством срабатываний переходов. Каждый антецедент и последовательность высказываний можно представить позициями и переходами.

Как организовать параллелизм при продукционном представлении знаний? – В продукционном программировании система включает множество правил, рабочую память и интерпретатор правил. Правила, содержащие антецедент и консеквент, выполняются последовательно. Понятно, что эффективность работы системы может быть улучшена при параллельной обработке правил. Однако не

все правила можно обрабатывать параллельно. Например, два правила, в которых элементы антецедента одного совпадают с элементами консеквента другого, не могут выполняться параллельно. Поэтому необходима определенная формальная процедура, позволяющая избежать подобного рода ситуации.

Пусть L_i и R_i соответственно антецедент и консеквент некоторого правила вывода PR_i. Если они имеют общие элементы K_i, то

$$K_i = L_i \cap R_i \tag{3.4}$$

Тогда справедливо следующее:

1. Если консеквент правила PR_1 имеет с правилом PR_2 общие элементы, такие что

$$R_1 \cap (R_2 \setminus K_2) \neq \varnothing \tag{3.5}$$

то правило PR_1 является *зависимым по выходу* от правила PR_2.

2. Если антецедент правила PR_1 имеет с правилом PR_2 общие элементы, такие что

$$L_1 \cap (L_2 \setminus K_2) \neq \varnothing \tag{3.6}$$

то правило PR_1 является зависящим по входу от правила PR_2.

3. Если для двух правил PR_1 и PR_2

$$K_1 \cap K_2 \neq \varnothing \tag{3.7}$$

то правила являются взаимосвязанными.

4. Если антецедент правила PR_1 имеет с правилом PR_2 общие элементы, такие что

$$L_1 \cap (R_2 \setminus K_2) \neq \varnothing \tag{3.8}$$

то правило PR_1 является зависящим по входу-выходу от правила PR_2.

5. Если консеквент правила PR_1 имеет с правилом PR_2 общие элементы, такие что

$$R_1 \cap (L_2 \setminus K_2) \neq \varnothing \tag{3.9}$$

то правило PR_1 является зависящим по выходу-входу от правила PR_2.

6. Правило PR_1 совместимо с правилом PR_2, если оно не зависит от него ни по входу, ни по выходу-входу.

7. Два правила PR_1 и PR_2 можно выполнять *параллельно*, если они являются совместимыми.

С учетом этих утверждений нетрудно получить списки параллельно выполняемых правил продукционной системы, составив матрицу параллелизма [59; 60]:

$$MP = || mp_{ij} ||_{n \times n},$$

где $mp_{ij} = 0$, если правило PR_i совместимо с правилом PR_j; $mp_{ij} = 1$, в противном случае.

Эта матрица обладает следующим свойством: если $mp_{ij} = mp_{ji} = 0$, то правила PR_i и PR_j являются взаимно совместимыми. Используя это свойство матрицы нетрудно определить *максимальный* набор совместимых правил.

Пусть, например, продукционная система включает 6 правил, а матрица параллелизма для них имеет следующий вид:

$$MP = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Тогда списки совместимых правил имеют вид:

$$S_1 = \{PR_3, PR_4, PR_6\}, S_2 = \{PR_1, PR_2, PR_4, PR_6\}, S_3 = \{PR_1, PR_5\}.$$

Совместимые правила можно параллельно обрабатывать на различных процессорах, что обеспечит выигрыш во времени [61; 62].

3.3. Нейробиостохастический алгоритм балансирования тележки

Нейроэволюция использует для обучения нейросети биостохастические алгоритмы в областях, где реализация обучения по прецедентам практически невозможна [63]. Нейроэволюция относится к категории методов искусственного интеллекта обучения с подкреплением, где роль объектов играют пары <ситуация – принятое решение>, а ответами являются значения целевого критерия, характеризующего правильность принятых решений, в зависимости от реакции среды. Обучение с подкреплением является промежуточным типом задач между обучением с учителем по прецедентам (регрессионный анализ, классификация) и обучением без учителя, где нужно найти закономерности в данных (кластеризация) [64].

В процессе обучения с подкреплением учитывается взаимодействие со средой: информация на входах нейросети отображается в выходной вектор, однако, поскольку обучающая выборка отсутствует, то нет возможности получить разность между эталонным значением выхода и реальным значением выходного вектора. Однако имеются некоторые априорные знания о связи выходов нейросети с наблюдаемыми событиями во внешней среде [65].

Нейробиостохастический алгоритм (НБА) обучения с подкреплением нацелен на решение таких проблем как сокращение пространства поиска, повышение скорости обучения и качества получаемых решений. Эвристики, применяемые в НБА, подразделяются на следующие категории: ограничения на структуру нейросетей; выбор операторов изменения нейросетей; способы использования биостохастических операторов кроссинговера и мутации; эвристики для целевой функции.

В зависимости от решаемых задач различают три типа НБА: алгоритмы поиска значений весов соединений нейросети при фиксированной структуре; алгоритмы настройки структуры нейросети; алгоритмы настройки параметров функций активации нейронов; алгоритмы, представляющие разные комбинации типов НБА [3].

В настоящее время наиболее значимые результаты в области нейроэволюции получены исследовательскими группами под руководством *R. Miikkulainen* (*http://www.cs.utexas.edu/~risto/*), *K. Stanley* (*http://www.cs.ucf.edu/?kstanley*), *D. Floreano*, *S. Nolfi*, *X. Yao* (*http://www.cs.bham.ac.uk/~xin*), *Ch. Igel*, *J. Schmidhuber* (*http://www.idsia.ch/~juergen/*) [66]. В России исследования НБА ведутся в центре оптико-нейронных технологий совместно с Институтом прикладной математики им. Келдыша, в Сибирском федеральном университете, в Южном федеральном университете, в Ульяновском государственном техническом университете и Томском политехническом университете.

Разработка НБА сопряжена с определенными трудностями. Во-первых, это кодирование нейросети в виде генотипа. Во-вторых, это деструктивное влияние оператора кроссинговера. В-третьих, создание начальной популяции нейросетей со случайной топологией не представляется удачным подходом.

В [67] для обозначения нейросетей, в которых производится эволюция, как узлов сети, так и топологических межсоединений, используется термин *TWEANNs* (*Topology & Weight Evolving*

Artificial Neural Networks). Используя *TWEANNs*, удалось решить одну из самых известных задач обучения с подкреплением – балансирования тележки с двумя флагштоками [68]. Нейросеть, не имея информации о скорости, управляет тележкой, с которой через шарниры соединены два флагштока разной длины, находящиеся в состоянии неустойчивого равновесия. *TWEANNs* не зависит от вида функций активации нейронов, нет необходимости в обучающей выборке, поиск топологии нейросети осуществляется автоматически. К проблемным вопросам *TWEANNs* относятся трудности с оценкой структуры нейросети; повышенные требования к памяти, сложности с поиском оптимальной топологии нейросети.

Существует несколько фундаментальных вопросов, связанных с разработкой НБА:

• алгоритмы используют генотипические представления. Генотип – это код нейросети. Как эффективно кодировать сетевую структуру в виде генотипа? В настоящее время исследователи использует различные методы кодирования. Например, при прямом кодировании фенотип (раскодированное решение) совпадает с генотипом, нейроны и их топология непосредственно кодируются в генотипе. Напротив, при неявном кодировании либо указываются параметры сети (число слоев и нейронов в них), либо используются специализированные грамматические правила и последовательности использования правил;

• проблема кроссинговера состоит в том, что оператор кроссинговера нельзя применять к сетям, если их генетическая информация отличается по длине. К тому же потомки, генерируемые оператором кроссинговера процессе репродукции, зачастую имеют гораздо худшее значение целевой функции, нежели у родителей;

• как создать подходящую начальную популяцию нейросетей? Генерация начальной популяции нейросетей со случайной топологией не представляется удачным подходом. Влияние оператора кроссинговера на нейроэволюцию оказывается деструктивным.

Рассмотрим предлагаемый НБА балансирования тележки с двумя флагштоками разной длины для *TWEANNs*. Его отличительная особенность – основным эволюционным оператором является мутация, а оператор кроссинговера не используется вообще.

Уточним постановку задачи балансирования тележки с двумя флагштоками разной длины [69].

Два флагштока соединены с движущейся тележкой с помощью шарниров (рисунок 3.5)

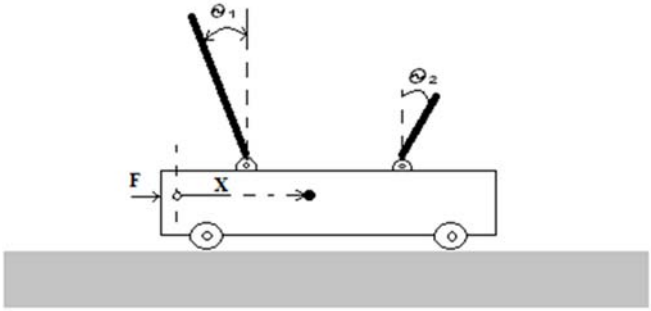


Рис. 3.5. Тележка с двумя флагштоками разной длины

Нейросеть должна управлять движением тележки, чтобы сбалансировать оба флагштока так долго, насколько это возможно. Состояние системы определяется положением x и скоростью v_x тележки, а также углом Θ_1 отклонения от вертикали первого флагштока и его угловой скоростью ω_1, углом Θ_2 отклонения от вертикали второго флагштока и его угловой скоростью ω_2. В целях упрощения информация о v_x, ω_1 и ω_2 не используются в качестве входных данных.

Флагштоки имеют различную длину. Компьютерное моделирование проводится с четырьмя различными соотношениями длины низкого и высокого флагштока: 1:10, 1:5, 1:3 и 1:2,5. Известно, что управление балансировкой флагштоков становится более трудным, когда соотношение длин флагштоков приближается к единице. Данные для компьютерного моделирования выбираются такими же, как и в [67]. Для сравнения используются данные приведенные в [70] для нейроэволюционного алгоритма *NEAT* (*Neuro-Evolution of Augmenting Topologies*). Алгоритм *NEAT* выбирается для сравнения, потому что он решает задачу балансирования флагштока на тележке в 25 раз быстрее, нежели алгоритм, использующий клеточное кодирование [68].

Работа предлагаемого НБА начинается с нейросетей без нейронов скрытого слоя и идет в направлении усложнения их топологии. Кодирование нейросети должно позволять проводить осмысленный кроссинговер различных топологий. Генотип представляется двумя списками: списком смежных вершин (нейронов) и списком

синаптических весов. Вершина кодируется двумя компонентами: одна идентифицирует ее в поколениях популяции, а другая указывает на тип слоя (входной, скрытый, выходной). Список синаптических весов состоит из указателей на смежные вершины, на значение веса, на флаг (используется ли соединение) и на «историческую метку», которая служит для идентификации соединения для всех поколений в популяции. Это позволяет применять оператор кроссинговера без дублирования или удаления генетической информации, полученной от родителей.

Тем не менее, новое потомство, порожденное оператором кроссинговера, как правило, имеет целевую функцию хуже, нежели у родительских хромосом. В [67] предлагалось применять кроссинговер к особям с небольшими структурными различиями. Однако это быстро приводит к топологическому однообразию получаемых нейросетей.

Предлагается подход, в котором не используется кроссинговер. Популяция рассматривается как центральный объект НБА. Алгоритм не предусматривает рекомбинации, поэтому оператор кроссинговера отсутствует. Мутация является единственным оператором поиска альтернативных решений.

Рассмотрим способ кодирования нейросетей, имея в виду разработку надежного способа генерации нейроструктур с небольшими изменениями лишь в ограниченной области [71]. Предположим, что нейросеть представляет собой набор подсетей, каждая из которых мало зависит от других. Поэтому небольшое мутационное изменение в подсети будет влиять лишь на ограниченную область.

Воспользуемся биологическим понятием оперона. *Оперон* – это группа функционально связанных между собой генов, активность которых упорядочена и зависит как от внешних условий, так и от деятельности других генов. Опираясь на это понятие, под опероном будем понимать некоторую подсеть, состоящую из некоторого подмножества нейронов (узлов сети) и подмножества соединений между нейронами.

Генотип, как строковая кодировка, состоящая из символов, представляется в виде подмножества оперонов следующего вида:

$$string = \{operon_1, operon_2, ..., operon_N\} \tag{3.10}$$

$$operon_i = \{\{node_j | \, j = 1..J\}, \{edge_k | \, k = 1..K\}\} \tag{3.11}$$

где N – максимальное число оперонов ($i = 1..N$); $node_j$ – идентификационный номер j-го нейрона, входящего в множество $operon_i$;

J – число нейронов, входящих в $operon_i$ со своими идентификационными номерами; $edge_k$ – k-е соединение в $operon_i$; K – число соединений в $operon_i$. На рисунке 3.6 представлено кодирование нейронной сети: строка-оперон (узлы + ребра).

Начальная популяция состоит из особей, имеющих только один operon0, который содержит только входные и выходные узлы и соединения между ними (нет узлов скрытого слоя).

Рассмотрим подробнее две разновидности оператора мутации, которые используются в алгоритме НБА для добавления узлов и для добавления соединений нейросети.

Оператор мутации для добавления узлов применяется к каждому оперону с постоянным значением вероятности p_m. Случайно выбирается для удаления одно из синаптических соединений, а затем вместо него добавляется нейрон скрытого слоя и два связанных с ним синаптических соединения. Если один из концов удаленного синаптического соединения был подключен к узлу $operon_0$, то добавленный нейрон скрытого слоя и его соединения образуют новый оперон.

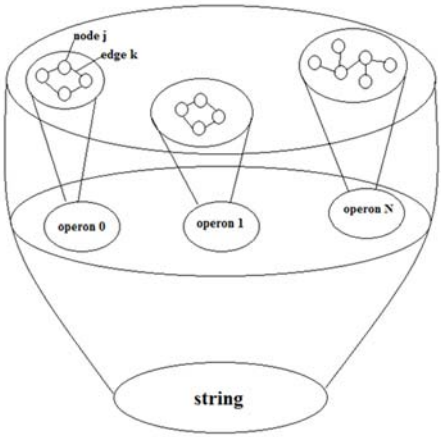


Рис. 3.6. Кодирование нейронной сети:
строка-оперон (узлы + ребра)

Например, пусть задан $operon_0$ с двумя входами x_1, x_2, веса которых равны w_1, w_2 соответственно, и выходным нейроном y_1, функцией активации которого является сигмоида (рисунок 3.7).

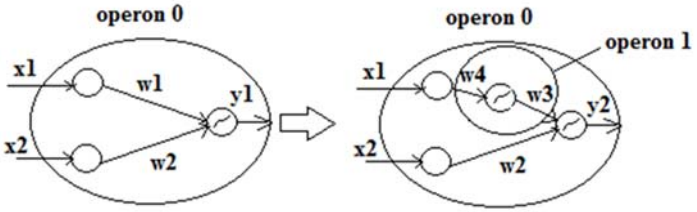


Рис. 3.7. Оператор мутации для добавления узлов нейросети

После удаления случайно выбранного синаптического соединения с весом w_1, добавляется нейрон скрытого слоя и два соединения с весами w_3 и w_4, где w_4 – весовое значение входа x_1, w_3 – вес соединения между выходом нового нейрона скрытого слоя и выходным нейроном y_2 $operon_0$. Добавленный нейрон и его соединения образуют новый $operon_1$.

Для того чтобы после мутации не изменился выходной сигнал $operon_0$, значения y_1 и y_2 должны быть равны. Поэтому в предположении, что S является сигмоидальной функцией, должно выполняться следующее условие:

$$S[w_1x_1 + w_2x_2] = S[w_3S[w_4x_1] + w_2x_2] \qquad (3.12)$$

Отсюда получаем, что

$$w_1x_1 - w_3S[w_4x_1] = 0 \qquad (3.13)$$

Кроме того, при условии, что $w_1 = w_3$ и $S[x] = 1/(1 + exp^{\beta(\alpha - x)})$, получим следующее значение целевой функции $f(x_1)$ после выполнения оператора мутации:

$$f(x_1) = x_1 - 1/(1 + exp^{\beta(\alpha - w4x1)}) \qquad (3.14)$$

где с целью упрощения и по результатам экспериментов были установлены следующие значения: $w_4 = 1$, $\alpha = 0{,}5$, $\beta = 5$.

Оператор мутации для добавления соединений также применяется к каждому оперону с постоянным значением вероятности. Пример такого оператора мутации представлен на рисунке 3.8.

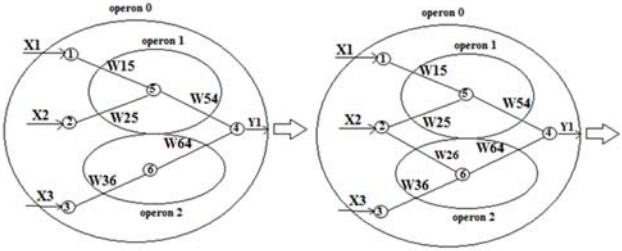


Рис. 3.8. Оператор мутации для добавления соединений нейросети

Случайно выбирается некоторый узел и добавляется соединение с узлом в $operon_i$ или в $operon_0$. Вес нового соединения устанавливается в 0, так что это не изменяет передаваемый сигнал.

Чем предлагаемый алгоритм НБА лучше классического обучения нейросетей, например, методом обратного распространения ошибки? Основной плюс заключается в его «креативности», – за счет мутаций нейросети возникают топологические инновации, которые трудно было бы предсказать. Классические методы обучения нейросетей сэмплируют меньшую область пространства решений.

Результаты экспериментов для задачи балансирования тележки с двумя флагштоками разной длины показали следующее.

Параметры алгоритма НБА и конкурирующего алгоритма *NEAT* в ходе экспериментов имели следующие значения:

- размер популяции – 1000;
- размер турнирной селекции – 20;
- вес соединения при использовании оператора мутации для добавления узлов – 1;
- значения параметров сигмоидальной функции $\alpha = 0{,}5$, $\beta = 5$.

Было проведено 10 серий экспериментов и получены следующие результаты.

Задача с вероятностью близкой к единице успешно решалась сравниваемыми алгоритмами при соотношении длин низкого и высокого флагштока 1:10 и 1:5. Однако при дальнейшем выравнивании длин флагштоков результаты, показываемые по алгоритму *NEAT*, становились хуже. В частности, в самых сложных условия при соотношении длин флагштоков 1:2,5 задача управления балансом успешно решалась по алгоритму *NEAT* только один раз из десяти экспериментов. Для алгоритма НБА вероятность успешного решения составила 0,8.

Однако алгоритм *NEAT* при соотношении длин низкого и высокого флагштока 1:10 и 1:5 находит решение примерно в течение 40 поколений. Этот результат существенно лучше, нежели для алгоритма НБА. При соотношении длин низкого и высокого флагштока 1:3 и 1:2,5 *NEAT* находит решение примерно в течение 200 поколений, а предлагаемый алгоритм – в течение примерно 350–400 поколений.

В наиболее успешную нейросеть при соотношении длин низкого и высокого флагштока 1:10 алгоритм *NEAT* добавил 6 узлов и 11 соединений. При том же соотношении длин алгоритм НБА дает сопоставимый результат. Однако при соотношении длин 1:2,5 сложность сетей, получаемых по алгоритму *NEAT*, быстро растет и достигает размера свыше 50 узлов и свыше 300 соединений. В этом отношении сложность топологии нейросети, генерируемой по алгоритму НБА при всех четырех соотношениях длин низкого и высокого флагштока, остается практически неизменной – около 10 узлов и 20 соединений.

Таким образом, результаты моделирования подтверждают гипотезу о преимуществах генерации нейроструктур путем небольших мутационных изменений в ограниченной области. Отличием алгоритма НБА от аналогов является возможность изменения архитектуры нейросети за счет одновременного добавления, как нейронов сети, так и соединений. Эволюция сети происходит небольшими мутационными изменениями в ограниченной области без использования оператора кроссинговера. Алгоритм не зависит от вида функций активации нейронов, нет необходимости в обучающей выборке, поиск топологии нейросети осуществляется автоматически.

Алгоритм НБА является перспективным для применения в областях, где требуется осуществлять действия в среде, лишь иногда получая обратную связь.

Заключение

Мягкие вычисления – это не единственный метод, а сочетание нескольких подходов, которые дополняют друг друга и могут использоваться вместе, чтобы решить данную проблему.

Интеллектуальные системы и алгоритмы мягких вычислений становятся все более важными для принятия сложных решений и выбора наилучшей альтернативы из множества возможных. Признанием этого факта может служит концепция Интернета вещей (*IoT*) – вычислительной сети физических предметов, оснащённых встроенными технологиями и приложениями для взаимодействия друг с другом или с внешней средой, исключающее из части действий и операций необходимость участия человека [72]. Приложения могут быть эффективно обработаны недорогими, но сверхбыстрыми микроконтроллерами на основе использования нечеткой логики, искусственных нейронных сетей и биоэвристик как для многих повседневных бытовых ситуаций, так и для промышленных и коммерческих применений. В течение следующего десятилетия этот показатель будет расти, теория, методы мягких вычислений и их приложения будут быстро развиваться вместе с использованием *IoT*-устройств.

В книге представлены проблемно-ориентированные алгоритмы анализа графических образов, траекторные, популяционные и нейробиостохастические алгоритмы. Алгоритмы нечеткой кластеризации проиллюстрированы на задачах анализа и представления графических образов. Установлено, что эффективность алгоритмов кластеризации зависит от применяемой функции расстояния и формы кластеров. В качестве примера работы алгоритма мягких вычислений для представления графических образов рассмотрен механизм ассоциативной презентации рабочего кабинета с целью определения его наилучшей конфигурации.

Проведен анализ особенностей поиска оптимальных решений траекторными и популяционными биоэвристиками. Предложен популяционный муравьиный алгоритм транспортной логистики и его визуализация на географической карте для задачи коммивояжера, а также масштабируемый популяционный алгоритм для задачи поиска оптимума многомерных многоэкстремальных функций Гриванка, Растригина, Розенброка и Швефеля.

Рассмотрены аспекты параллелизма мягких вычислений в гетерогенных компьютерах сетях и на параллельных высокопроизводительных кластерных вычислителях. Рассмотрены различные модели и способы организации параллелизма, получена оценка верхней границы для максимального ускорения, которое может быть достигнуто на многопроцессорном компьютере, а также возможности организации параллельных вычислений и параллельной обработки продукционных правил в базах знаний на примере использования кластерного вычислителя. Представлен нейробиостохастический алгоритм балансирования тележки с двумя флагштоками разной длины.

Список используемой литературы

1. *Zadeh L.* Fuzzy Logic, Neural Networks, and Soft Computing // Communications of the ACM. – 1994. – Vol. 37. – No. 3. – P. 77–84.

2. *Рутковская Д.* Нейронные сети, генетические алгоритмы и нечеткие системы / *Д. Рутковская, М. Пилиньский, Л. Рутковский*. – М.: Горячая линия – Телеком, 2013. – 384 с.

3. *Нариньяни А.С.* Введение в недоопределенность / *А.С. Нариньяни* // Информационные технологии. Приложение. – 2007. – №4. –32 с.

4. *Ковалев С.М.* Аналитический обзор современных интеллектуальных информационных технологий в технике и на производстве / *С.М. Ковалев, В. Снашел, А.Н. Гуда* [и др.] // Вестник Ростовского государственного университета путей сообщения. – 2019. – №1. – С. 60–75.

5. *Курейчик В.В.* Теория эволюционных вычислений / *В.В. Курейчик, В.М. Курейчик, С.И. Родзин*. – М.: Физматлит, 2012. – 260 с.

6. *Тарасов В.Б.* Технологии индустрии 4.0: от цифрового производства и интернета вещей до интеллектуального имитационного моделирования и коллаборативных роботов / *В.Б. Тарасов* // Мягкие измерения и вычисления. – 2018. – №12. – С. 3–15.

7. *Тарасов В.Б.* Мягкие вычисления и измерения. Т. 1 / *В.Б. Тарасов* [и др.]; под редакцией С.В. Прокопчиной. – М.: ИД «Научная библиотека», 2017. – 420 с.

8. *Родзин С.И.* Биоэвристики: теория, алгоритмы и приложения: монография / *С.И. Родзин, Ю.А. Скобцов, С.А. Эль-Хатиб*. – Чебоксары: ИД «Среда», 2019. – 224 с.

9. *Ковалев С.М.* Информационные технологии: интеллектуализация обучения, моделирование эволюции, распознавание речи / *С.М. Ковалев, С.И. Родзин*. – Ростов н/Д: Изд-во СКНЦ ВШ, 2002. – 224 с.

10. *Родзина О.Н.* Методы, модели и технология разработки нечетких кластерных анализаторов для классификации и распознавания графических образов / *О.Н. Родзина* // Перспективные информационные технологии и интеллектуальные системы. – 2000. – №3. – С. 96–99.

11. *Родзина О.Н.* Механизм нечеткого кластерного анализа как элемент открытой интеллектуальной системы / *О.Н. Родзина* // Вестник ТРТУ–ДонГТУ. – 2001. – №1. – С. 125–129.

12. *Гашников М.В.* Методы компьютерной обработки изображений / *М.В. Гашников, Н.И. Глумов, Н.Ю. Ильясова* [и др.]; под ред. В.А. Сойфера. – М.: Физматлит, 2003. – 784 с.

13. *Гришенцев А.Ю.* Методы и модели цифровой обработки изображений / *А.Ю. Гришенцев, А.Г. Коробейников*. – СПб.: Политехнический университет, 2014. – 190 с.

14. *Родзина О.Н.* Правила нечеткой кластеризации при распознавании образов / *О.Н. Родзина* // Известия Таганрогского радиотехнического университета. – 2001. – №4. – С. 101–105.

15. *Korobeynikov A.G., et. al.* Calculation of regularization parameter in the problem of blur removal in digital image // Optical Memory & Neural Networks (Information Optics). – 2016. – Vol. 25. – №3. – P. 184–191.

16. *Коробейников А.Г.* Разработка автоматизированной процедуры для решения задачи восстановления смазанных цифровых изображений / *А.Г. Коробейников, М.Е. Федосовский, С.А. Алексанин* // Кибернетика и программирование. –2016. – №1. – С. 270–291.

17. *Родзин С.И.* Представление графических объектов с помощью ассоциативных правил / *С.И. Родзин, О.Н. Родзина* // Вестник ТРТУ–ДонГТУ. Т. 1. – 2003. – С. 140–149.

18. *Родзина О.Н.* Особенности поиска решений траекторными метаэвристиками / *О.Н. Родзина, Л.С. Родзина* // Труды Конгресса по интеллектуальным системам и информационным технологиям (IS&IT'18). Т. 1. – Таганрог: Изд-во Ступина С.А., 2018. – С. 48–60.

19. *Карпенко А.П.* Популяционные алгоритмы глобальной поисковой оптимизации. Обзор новых и малоизвестных алгоритмов / *А.П. Карпенко* // Приложение к журналу «Информационные технологии». – 2012. – №7. – С. 1–31.

20. *Rodzin S.* Smart dispatching and metaheuristic swarm flow algorithm // Journal of Computer and Systems Sciences International. – 2014. – Vol. 53. – No. 1. – P. 109–115.

21. *Родзин С.И.* Метод биогеографии для решения трансвычислительных задач комбинаторной оптимизации / *С.И. Родзин, О.Н. Родзина* // Информационные технологии в науке, образовании и управлении: труды межд. конф. IT+S&E'15. – М.: ИНИТ, 2015. – С. 204–213.

22. *Kravchenko Y.A., Kureichik V.V.* Bioinspired algorithm applied to solve the travelling salesman problem // World Applied Sciences Journal. – 2013. – No. 22. – P. 1789–1797.

23. *Blum C., Roli A.* Metaheuristics in combinatorial optimization: overview and conceptual comparison // ACM computing surveys. – 2003. – No. 35. – P. 268–308.

24. *Glover F., Kochenberger G.A.* Handbook of metaheuristics. – Springer, 2010. – 648 p.

25. *Родзина О.Н.* Анализ ландшафта оптимизируемых функций при принятии информационно-управляющих решений / *О.Н. Родзина, Л.С. Родзина* // Международный научно-исследовательский журнал. – 2019. – №9-1. – С. 6–9.

26. *Родзин С.И.* Теория и биогеографические модели как средство индуктивного поиска оптимальных решений / *С.И. Родзин, О.Н. Родзина* // Информатика, вычислительная техника и инженерное образование. – 2015. – №1. – С. 1–7.

27. *Родзин С.И.* Вычислительная модель биогеографии и ее применение для задачи коммивояжера / *С.И. Родзин, О.Н. Родзина* // Известия ЮФУ. Технические науки. – 2015. – №6. – С. 210–222.

28. *Dorigo M.*, et. A survey on metaheuristics for stochastic combinatorial optimization // International Journal of Natural Computing. – 2009. – No. 8 (2). – P. 239–287.

29. *Abraham A., Grosan G., Ramos V.* Swarm intelligence in data mining. – Berlin-Heidelberg: Springer verlag, 2006. – DOI 10.1007/ 978-3-540-34956-3.

30. *Eberhart R., Shi Yu., Kennedy J.* Swarm intelligence. – Morgan Kaufmann, 2010. – 512 p.

31. *Родзина О.Н.* Муравьиный алгоритм эффективного решения задачи транспортной логистики и его визуализация на географической карте / *О.Н. Родзина, Д.К. Головченко, М.А. Ковалев* [и др.] // Труды Конгресса по интеллектуальным системам и технологиям "IS&IT'14". Т. 3. – М.: Физматлит, 2014. – С. 32–42.

32. *Rodzin S., Rodzina O.* Effectiveness evaluation of memetics and biogeography algorithms using benchmark and trans computational tasks of combinatorial optimization // Proc. of the First Int. Scientific Conf. “Intelligent Information Technologies for Industry” (IITI'16), Vol. 1, Advances in Intelligent Systems and Computing, Springer, 2016. – P. 463–475.

33. *Родзин С.И.* Кластеризация МРТ-изображений: биостохастический метод муравьиной колонии / *С.И. Родзин, О.Н. Родзина, С.А. Эль-Хатиб* // ИТНОУ: Информационные технологии в науке, образовании и управлении. – 2017. – №2. – С. 66–74.

34. *Родзин С.И.* Гибридный муравьиный алгоритм сегментации медицинских изображений / *С.И. Родзин, О.Н. Родзина, С.А. Эль-Хатиб* // Вестник Чувашского университета. – 2017. – №3. – С. 262–272.

35. *Rodzin S., Rodzina O*. New computational models for big data and optimization // Proc. of the 9th Int. Conf. on Application of Information and Communication Technologies (AICT'2015). – 2015. – P. 3–7.

36. *Črepinšek M., Liu S.-H., Mernik M.* Exploration and exploitation in evolutionary algorithms: a survey // ACM Computing Surveys. – 2013. – Vol. 45. – No. 3. – P. 35–44.

37. *Virginia Y., Analía A.* A deterministic crowding evolutionary algorithm to form learning teams in a collaborative learning context // Expert System Appl. – 2012. – Vol. 39. – №10. – P. 8584–8592.

38. *Jie Y., Nawwaf K., Grogono P.* Bi-objective multi population genetic algorithm for multimodal function optimization // IEEE Trans. Evol. Comput. – 2010. – Vol. 14. – No. 1. – P. 80–102.

39. *Hui W., Zhijian W.,* Shahryar *R.* Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems // Soft Computing. – 2011. – Vol. 15. – No. 11. – P. 2127–2140.

40. *Xiaodong L., Yao X.* Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms // Proc. of the IEEE congress on evolutionary computation (CEC'09). – 2009. – P. 1546–1553.

41. *Koza J., et. al.* Genetic programming III: Darwinian invention and problem solving. – Cambridge. MA: MIT Press, 1999.

42. *García-Pedrajas N., Castillo J., Ortiz-Boyer D.* A cooperative coevolutionary algorithm for instance selection for instance-based learning // Machine Learning. – 2010. – Vol. 78. – No. 3. – P. 381–420.

43. *Родзин С.И.* Сравнение программных реализаций эволюционных вычислений для задач многомерной оптимизации / *С.И. Родзин, О.Н. Родзина* // Программная инженерия. – 2019. – Т. 10, №11-12. – С. 451–456.

44. *Bhattacharya M.* Exploiting landscape information to avoid premature convergence in evolutionary search // Proc. of the 2006 IEEE congress on evolutionary computation (CEC2006). IEEE Press. – 2006. – P. 2575–2579.

45. *Ursem R.K.* Diversity-guided evolutionary algorithms. In: Proceedings of parallel problem solving from nature VII (PPSN-2002). – 2002. – P. 462–471.

46. *Aranha C., Iba H.* Modelling cost into a genetic algorithm-based portfolio optimization system by seeding and objective sharing // Proc. of the IEEE congress on evolutionary computation, CEC2007. – 2007. – P. 196–203.

47. *Aranha C., Iba H.* The memetic tree-based genetic algorithm and its application to portfolio optimization // Memeting Computer. – 2009. – No. 1. – P. 139–51.

48. *Родзина О.Н.* Интеллектуальные системы. Гранулы параллелизма в эволюционных вычислениях: монография / *О.Н. Родзина;* под ред. В.М. Курейчика. – М.: Физматлит, 2011. – С. 30–62.

49. *Родзина О.Н., Родзина Л.С.* Эволюционные алгоритмы для динамической балансировки нагрузки в сети распределенных вычислений / *О.Н. Родзина, Л.С. Родзина* // Труды Конгресса по интеллектуальным системам и информационным технологиям (AIS-IT '09). Т. 3. – М.: Физматлит, 2009. – С. 302–305.

50. *Родзина О.Н.* Об эффективности эволюционных вычислений в многоядерных и кластерных архитектурах, системах метакомпьютинга / *О.Н. Родзина* // Труды Конгресса по интеллектуальным системам и информационным технологиям (AIS-IT '11). Т. 2. – М.: Физматлит, 2011. – С. 21–30.

51. *Родзин С.И.* Теоретические основы эволюционных вычислений для поддержки принятия оптимальных решений в многоцелевых информационных системах / *С.И. Родзин, О.Н. Родзина* // Информатика, вычислительная техника и инженерное образование. – 2014. – №4. – С. 1–15.

52. *S. Rodzin, O. Rodzina.* Metaheuristics memes and biogeography for trans computational combinatorial optimization problems // Proc. of the 6th Int. Conf. on Cloud System and Big Data Engineering (Confluence'2016). AI and Soft Computing. – 2016. – P. 1–5.

53. *S. Rodzin, O. Rodzina.* Effectiveness evaluation of memetic and biogeography algorithms using benchmark and trans computational tasks of combinatorial optimization // Proc. of the First Int. Scientific Conf. "Intelligent Information Technologies for Industry" (IITI'16), Vol. 1, Advances in Intelligent Systems and Computing, Springer International Publishing. – 2016. – P. 463–475.

54. *Родзина О.Н.* Организация распределенной обработки продукционных правил на кластере / *О.Н. Родзина* // Перспективные информационные технологии и интеллектуальные системы. – 2009. – №2. – С. 45–49.

55. *Родзина О.Н.* Генетическая оптимизация рабочей нагрузки при GRID-вычислениях / *О.Н. Родзина, Л.С. Родзина* // Перспективные информационные технологии и интеллектуальные системы. – 2009. – №3–4. – С. 20–23.

56. *Родзин С.И.* Масштабируемый биоинспирированный алгоритм для задач многомерной оптимизации / *С.И. Родзин, И.Г. Данилов, К.С. Ковалева* [и др.] // Известия ЮФУ. Технические науки. – 2019. – №4. – С. 49–59.

57. *Емельянов В.В.* Динамические продукции и нечётко-темпоральные модели знаний / *В.В. Емельянов, С.М. Ковалев, А.Е. Колоденкова* // Мягкие измерения и вычисления. –2018. – №12. – С. 51–56.

58. *Eremeev A.P., Vagin V.N.* A real-time decision support system prototype for management of a power block // International Journal "Information Theories and Applications". – 2003. – Vol. 10. – No. 3. – P. 248–255.

59. *Гришенцев А.Ю.* Постановка задачи оптимизации распределённых вычислительных систем / *А.Ю. Гришенцев, А.Г. Коробейников* // Программные системы и вычислительные методы. – 2013. – №4. – С. 370–375.

60. *Korobeynikov A., Fedosovsky M., Zharinov I., Polyakov V., Shukalov A., Arustamov S., Gurjanov A.* Method for conceptual presentation of subject tasks in knowledge engineering for computer-aided design systems // Advances in Intelligent Systems and Computing. – 2018. – Vol. 680. – C. 50–56.

61. *Ковалев С.М.* Иерархический интеллектуальный препроцессинг нечетко-стохастической информации в интегрированных системах динамического типа / *С.М. Ковалев, А.Н. Шабельников* // Известия ЮФУ. Технические науки. – 2017. – №3. – С. 148–157.

62. *Kovalev S., Sukhanov A.* Reconstructed phase space method for event prediction based on sugeno-type fuzzy inference // Frontiers in Artificial Intelligence and Applications. – 2016. – Vol. 281. – P. 142–148.

63. *S. Rodzin, O. Rodzina, L. Rodzina.* Neuroevolution: problems, algorithms, and experiments // Proc. of the 10th IEEE Int. Conf. Application of Information and Communication Technologies (AICT'2016). – P. 469–472.

64. *Родзин С.И., Родзина О.Н.* Построение прогнозов в рекомендательных системах с помощью машинного обучения на основе популяционного алгоритма / *С.И. Родзин, О.Н. Родзина* // Вестник компьютерных и информационных технологий. – 2020. – №1. – С. 48–56.

65. *Dolgiy A.I., Kovalev S.M., Sukhanov A.V., Styskala V.* Hybrid fuzzy neural model based dempster-shafer system for processing of diagnostic information // Lecture Notes in Electrical Engineering. – 2020. – Vol. 554. – P. 24–33.

66. *Kohl N.* Evolving neural networks for strategic decision-making problems / N. Kohl, R. Miikkulainen // Neural Networks. – 2009. – Vol. 22. – P. 326–337.

67. *Stanley K.O.* Evolving Neural Networks Through Augmenting Topologies / K.O. Stanley, R. Miikkulainen // Evolutionary Computation. – 2002. – No. 10(2). – P. 99–127.

68. *Moriguchi H.* CMA-TWEANN: Efficient Optimization of Neural Networks via Self-Adaptation and Seamless Augmentation / H. Moriguchi, S. Honiden // Proc. of the 14th Annual Conf. on Genetic and Evolutionary Computation. – 2012. – P. 903–910.

69. *Родзина О.Н.* Задача балансирования тележки с двумя флагштоками разной длины: нейроэволюционный алгоритм / *О.Н. Родзина, Л.С. Родзина* // Вестник РГУПС. – 2017. –№3. – С. 90–95.

70. *Stanley K.O.* URL: http://www.cs.ucf.edu/~kstanley/

71. *Родзина О.Н.* Бенчмаркинг нейроэволюционного подхода для задачи управления балансом тележки / *О.Н. Родзина, Л.С. Родзина* // Труды Конгресса по интеллектуальным системам и информационным технологиям "IS&IT'17". Т. 1. – Таганрог: Изд-во Ступина С.А., 2017. – С. 61–70.

72. *Смирнов А.В.* Контекстно-управляемая поддержка принятия решений в распределенной информационной среде / *А.В. Смирнов* [и др.] // Информационные технологии и вычислительные системы. – 2009. – №1. – С. 38–48.

Список используемых сокращений

В книге используются следующие сокращения:
БСА – биостохастический алгоритм.
НБА – нейробиостохастический алгоритм.
ГА – генетический алгоритм.
МА – муравьиный алгоритм.
ОК – оператор кроссинговера.
ОМ – оператор мутации.
ПМА – простой муравьиный алгоритм.
ЭВ – эволюционные вычисления.

API – Application Programming Interface.
FLOPS – FLoating-point Operations Per Second.
IoT – Internet of Things.
MIMD – Multiple Instruction stream, Multiple Data stream.
MISD – Multiple Instruction stream, Single Data stream.
MDVRP – Multiple Depot Vehicle Routing Problem.
NEAT – Neuro-Evolution of Augmenting Topologies.
NP – No Polynomial.
SIMD – Single Instruction, Multiple Data.
SISD – Single Instruction, Single Data.
SMP – Symmetric Multi Processing.
TSP – Travelling Salesman Problem.
TWEANNs – Topology & Weight Evolving Artificial Neural Networks.
WPF – Windows Presentation Foundation.

Для заметок

Научное издание

Родзина Ольга Николаевна

ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ АЛГОРИТМЫ МЯГКИХ ВЫЧИСЛЕНИЙ

Монография

Чебоксары, 2020 г.

Редактор *О. Н. Родзина*
Компьютерная верстка и правка *А. А. Кузьмина*
Дизайн обложки *Н. В. Фирсова*

Подписано в печать 24.07.2020 г.
Дата выхода издания в свет 10.08.2020 г.
Формат 60×84/16. Бумага офсетная. Печать офсетная.
Гарнитура Times. Усл. печ. л. 5,58. Заказ К-692. Тираж 500 экз.

Издательский дом «Среда»
428005, Чебоксары, Гражданская, 75, офис 12
+7 (8352) 655-731
info@phsreda.com
https://phsreda.com

Отпечатано в Студии печати «Максимум»
428005, Чебоксары, Гражданская, 75
+7 (8352) 655-047
info@maksimum21.ru
www.maksimum21.ru