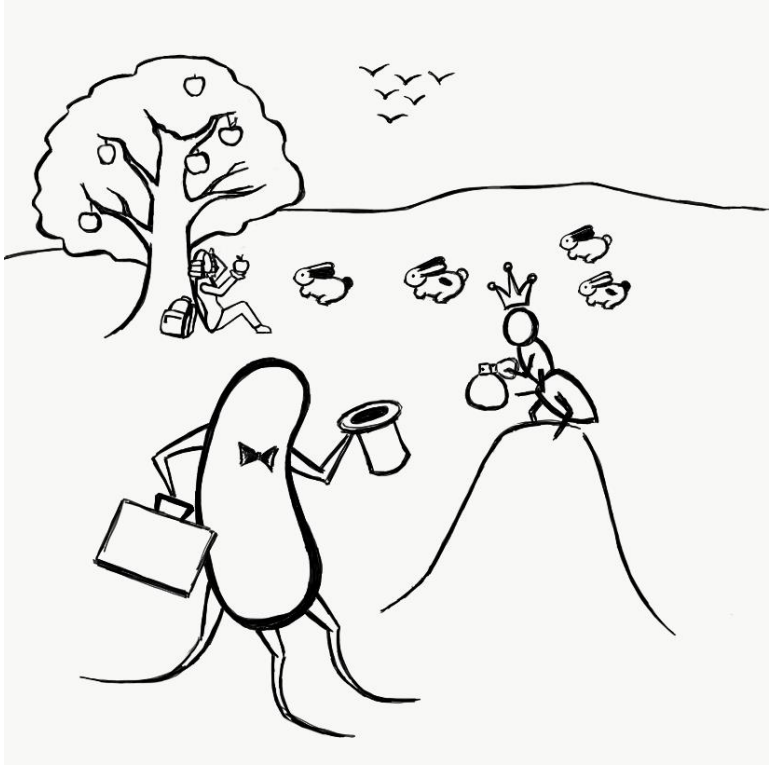
УНИВЕРСИТЕТ ИТМО

М.А. Мельник, А.А. Вишератин

ЭВОЛЮЦИОННЫЕ ВЫЧИСЛЕНИЯ

Учебно-методическое пособие



Санкт-Петербург
2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

М.А. Мельник, А.А. Вишератин

ЭВОЛЮЦИОННЫЕ ВЫЧИСЛЕНИЯ

Учебно-методическое пособие

по выполнению лабораторных работ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО

по направлению подготовки 01.04.02 Прикладная математика и информатика в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2018

Мельник М.А., Вишератин А.А. Эволюционные вычисления. Учебно-методическое пособие по выполнению лабораторных работ.

Учебно-методическое пособие. – СПб: Университет ИТМО, 2018. – 40 с.

Рецензент: Алоджанц Александр Павлович, д.ф.-м.н., ведущий научный сотрудник целевой поисковой лаборатории квантовой когнитивистики и интеллектуальных систем Университета ИТМО

Настоящее учебно-методическое пособие составлено в соответствии с ОС Университета ИТМО 01.04.02 – Прикладная математика и информатика

Пособие содержит учебно-методические разработки, предназначенные для выполнения лабораторных работ по следующим темам: введение в эволюционные алгоритмы; решение веществennознозначных оптимизационных проблем; решение комбинаторных оптимизационных проблем; проектирование эволюционных алгоритмов; распределенные и параллельные эволюционные алгоритмы.

Учебно-методическое пособие предназначено для студентов, обучающихся по направлению 01.04.02 «Прикладная математика и информатика».

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящем пособии представлены методические указания к выполнению лабораторных работ по дисциплине «Эволюционные вычисления».

По итогам выполнения всех работ студент должен получить практические навыки по разработке эволюционных алгоритмов для решения различных типов оптимизационных задач, включающих: численные, комбинаторные, многокритериальные; а также навыки анализа эффективности и производительности эволюционных алгоритмов. Основной целью данных лабораторных работ является выработка таких компетенций как: умение формализовать задачу оптимизации в терминах эволюционных вычислений; владение навыками проектирования и разработки эволюционных алгоритмов, в том числе высокопроизводительных; знание способов оценки производительности и эффективности эволюционных алгоритмов.

По итогам выполнения лабораторной работы, студенту необходимо подготовить письменный отчет, содержащий основные результаты работы. Отчет должен быть оформлен согласно требованиям ГОСТ и включать титульный лист и основную часть. Образец титульного листа представлен в Приложении 1.

Обязательным элементом сдачи отчета является устная защита лабораторной работы, в рамках которой студент отвечает на вопросы из контрольного списка.

Письменный отчет

Отчет по лабораторной работе представляется в печатном виде в формате, предусмотренном методическим пособием и шаблоном отчета по ЛР (приложение 1).

Обязательным элементом сдачи отчета является защита лабораторной работы, в рамках которой студент отвечает на вопросы из контрольного списка.

Лабораторная работа № 1
Введение в эволюционные вычисления

Цель работы

Целью данной лабораторной работы является получение студентом представления об возможностях применения эволюционных алгоритмов для решения различных классов задач и программных средств для их разработки.

Оборудование и программное обеспечение

Для выполнения лабораторной работы потребуется:

- браузер с доступом к сети Интернет;
- Java JDK версии 1.8 и выше;
- Watchmaker framework версии 0.7.1.

Краткие теоретические сведения

Эволюционные Вычисления (ЭВ) являются областью информатики в основе методов и алгоритмов которой лежат идеи и концепции, взятые из естественных природных процессов. Наиболее популярным примером является Генетический Алгоритм (ГА), вдохновением на создание которого был взят процесс эволюции. Эволюция считается одним из наиболее мощных природных процессов, который проявляется в создании разновидностей, стремящихся к выживанию в своей среде обитания за счёт адаптации к ней. Таким образом, ГА представляет собой популяцию индивидов, где индивид является одним из возможных решений некоторой поставленной задачи. Индивид или решение характеризуется двумя составляющими: генотип и фенотип. Генотип является внутренней структурой решения, тем, как решение представляется в программе для реализации алгоритма в виде набора генов. Фенотип представляет собой непосредственный предметно-ориентированный вид решения в терминах поставленной задачи. Качество каждого решения может быть оценено функцией полезности – фитнес-функцией. Начальный набор индивидов подвергается трём операциям: кроссовером, мутацией и селекцией. Кроссовер является ничем иным, как скрещиванием двух или более индивидов с целью воспроизведения потомства. Дочерние решения наследуют части генов от родителей. Мутация представляет собой небольшое случайное изменение в генотипе решения. Операторы мутации и кроссовера необходимы для обхода пространства поиска с целью нахождения оптимального решения. Селекция является регуляризатором сходимости алгоритма и осуществляет отбор индивидов среди популяции и полученных в процессе кроссовера потомков на основе значений их фитнес-функции. Перечисленные эволюционные операторы повторяются итерацией за итерацией над текущем поколением (популяцией решений) до тех пор, пока не будет удовлетворено

условие терминации алгоритма. Схема генетического алгоритма представлена на рисунке 1.1.

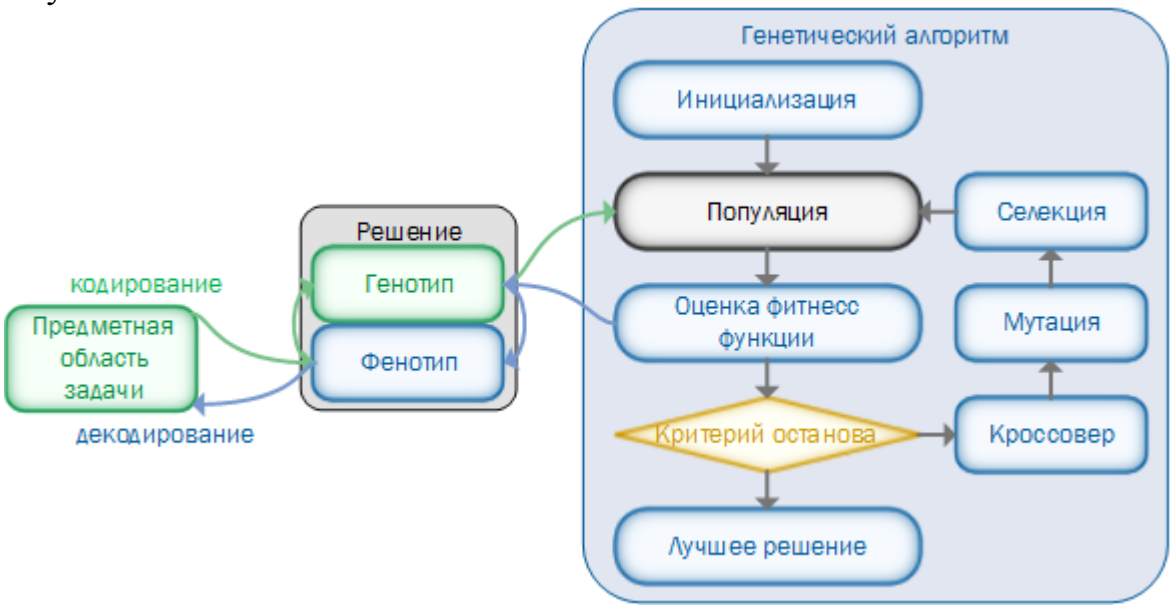


Рисунок 1.1 – Схема генетического алгоритма

Эволюционные алгоритмы получили широкое распространение и нашли своё применение для решение огромного числа задач, включающих промышленные, научные, образовательные и даже задачи развлекательного характера. В качестве примеров можно привести классические оптимизационные задачи:

- задача коммивояжёра;
- задача расстановки ферзей;
- транспортная задач;
- обучение параметров искусственной нейронной сети;
- задачи планирования и составления расписаний;
- разработка игровых стратегий.

Первой и наиважнейшей стадией при разработке эволюционного алгоритма является конструирование наиболее пригодного представления генотипа – структуры решений. Структура решений должна быть спроектирована таким образом, чтобы генотип решения мог быть раскодирован максимально однозначно в решение поставленной задачи с целью оценки его качества и валидности (при наличии ограничений). При этом, важно, чтобы генотип был удобен для реализации эволюционных операторов и позволял осуществлять полноценный поиск по пространству решений без усложнения размерности задачи и излишних вычислительных затрат. В общем можно выделить следующие типы представления решений: бинарные, целочисленные, вещественнозначные, комбинаторные и древовидные. Например, решения задачи обучения параметров искусственной нейронной сети очевидно могут быть закодированы в виде вектора вещественных чисел. В данном случае, вектор параметров является генотипом, а сама нейронная сеть, построенная на этом векторе значений, будет являться фенотипом.

К достоинствам эволюционных алгоритмов и мотивации для их применения относят:

- возможность представления решаемой задачи в формате оптимизации «чёрного ящика»;
- возможность изменять важность оптимизационных критериев или решать многокритериальную задачу без изменения в структуре самого алгоритма и кодировки решения;
- итеративность и возможность получения текущего результата в любой момент времени;
- возможность получения интуитивно неожидаемого решения;
- возможность динамической адаптации к изменяемой среде;
- интерес с точки зрения использования природных концепций и автоматического нахождения решений без использования специфики предметной области.

В настоящее время создано множество программных средств и фреймворков для упрощения разработки эволюционных алгоритмов и улучшения их производительности за счёт параллелизации вычислительных процессов. Также, существуют готовые примеры и решения задач, с некоторыми из которых и предлагается ознакомиться в рамках данной лабораторной работы.

Ход работы

Скачать проект, содержащий фрэймворк Watchmaker и реализованные на нём примеры по ссылке: https://watchmaker.uncommons.org/download.php.

Открыть среду разработки, поддерживающую Java (например, IntelijIDEA). Далее требуется создать проект из существующего источника, которым является путь к скачанному фреймворку. При создании проекта необходимо выбрать Maven в качестве средства сборки проекта. Структура проекта при правильной сборке должна принимать вид как показано на рисунке 1.2. Основные обязательные директории и файлы подчёркнуты красной линией.

Bits count. В данном примере решается задача подсчёта количество битов (‘1’) в строке заданной длины. Для работы с примером необходимо открыть файл BitsExample.java, который находится в директории:
“..\examples\src\java\main\org\uncommons\watchmaker\examples\bits\”.

Размерность проблемы определяется значением параметра ***BITS***, который изначально равен 20. Запустите пример, на экране должны выводиться результаты алгоритма на каждой итерации. Алгоритм останавливается в момент нахождения оптимального решения. Определите и выпишите, какой структурой данных и в каком виде закодированы решения в данной реализации примера.

Запустите программу несколько раз и заполните значения количеств итераций, которые были необходимы для получения конечного решения при

размерности проблемы 20 на основе таблицы 1.1. Повторите серию запусков для решения задачи с размерностями 50 и 100. Рассчитайте среднее количество итераций алгоритма для решения задачи в каждом случае.

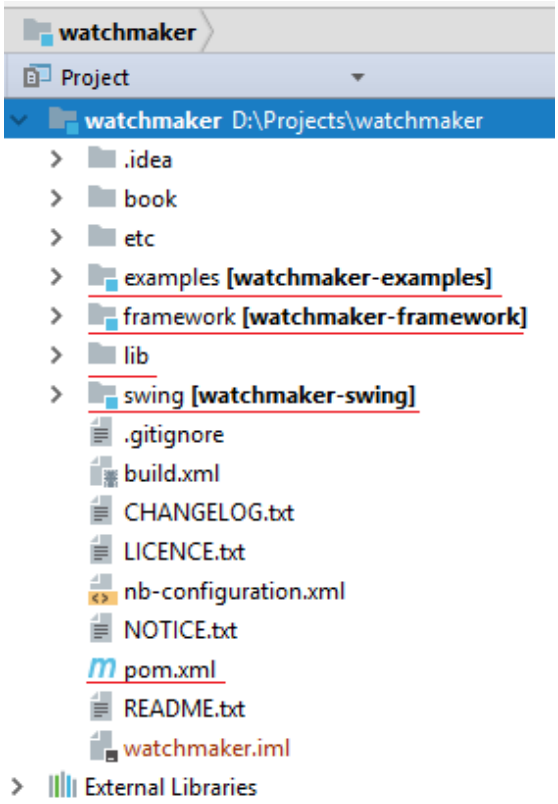


Рисунок 1.2 – Файловая структура проекта

Таблица 1.1 Результаты расчёта количества итераций алгоритма от размерности проблемы.

Размерность	Run 1	Run 2	Run 3	Run 4	Run 5	Среднее
20						
50						
100						

Travelling salesman problem. В данном примере решается задача коммивояжёра, целью которой является найти кратчайший путь, проходящий через все заданные точки (города). Для запуска примера необходимо запустить файл:
"..\examples\src\java\main\org\uncommons\watchmaker\examples\travellingsalesman\TravellingSalesmanApplet.java".

В открывшемся окне (рисунок 1.3) выберите все города и попробуйте найти оптимальное решение задачи. Выпишите найденный маршрут по всем 15 городам и дистанцию найденного маршрута.

Попробуйте поварьировать параметры алгоритма для анализа их влияния на его производительность.

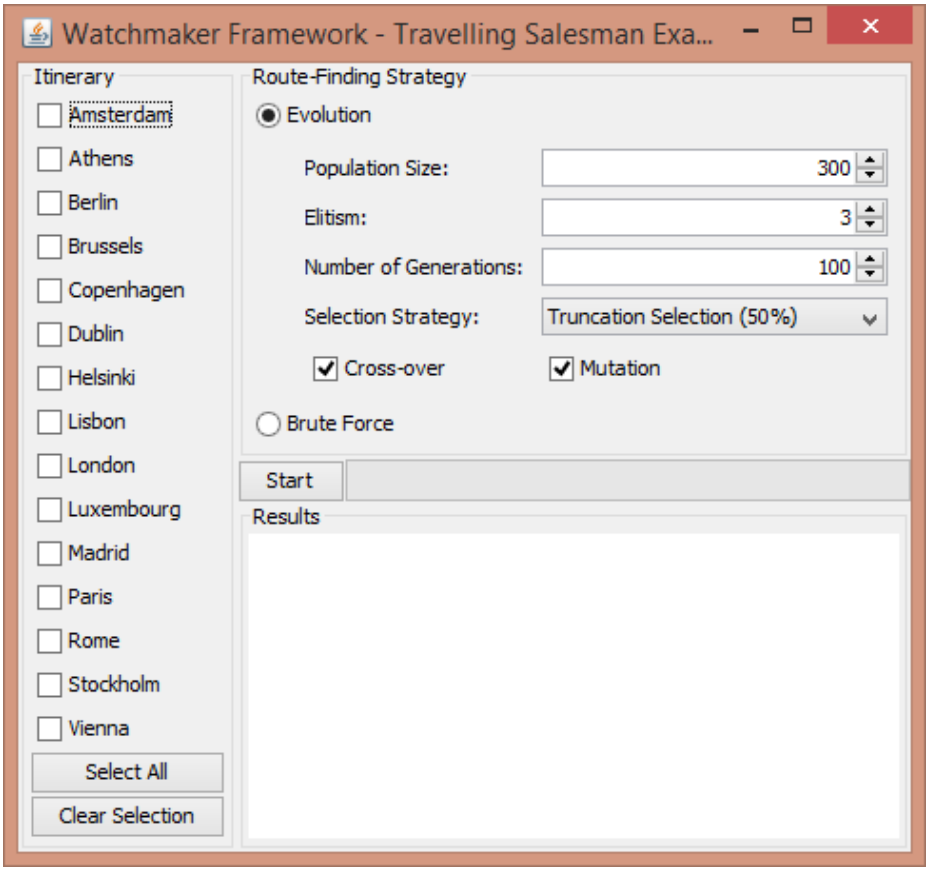


Рисунок 1.3 – Окно запуска алгоритма для решения задачи коммивояжёра

Mona Lisa. В последнем примере, который рассматривается в рамках лабораторной работы, представлена задача подбора множества разноцветных полигонов для наиболее точного воспроизведения картины. Для запуска примера необходимо запустить файл:
"..\examples\src\java\main\org\uncommons\watchmaker\examples\monalisa\MonaLisaApplet.java".

В открывшемся окне запустите алгоритм. Сделайте несколько снимков экрана, которые показывали бы плохое, среднее и хорошее решение задачи (субъективно). Вырежьте только части с получающимися на текущий момент картинами. Для выполнения можно остановить и запустить алгоритм несколько раз. В соответствии со структурой в таблице 1.2, заполните полученные результаты.

Таблица 1.2 Результаты оптимизации подбора полигонов для воспроизведения картины

Решение	Итерация	Фитнесс	Кол-во полигонов и углов	Рисунок
плохое				
среднее				
хорошее				

Вопросы:

1. К какому типу по структуре решений относится каждая из рассмотренных задач?
2. Как закодированы решения в задаче коммивояжёра?
3. Что является генотипом, а что фенотипом в задаче воспроизведения картины?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Satellite Truss Design, http://www.soton.ac.uk/~ajk/truss/welcome.html.
4. Genetic walkers, http://rednuht.org/genetic_walkers/.
5. Genetic cars, http://rednuht.org/genetic_cars_2/.

Лабораторная работа № 2
Генетический алгоритм для задачи оптимизации непрерывной функции

Цель работы

Целью данной работы является получение студентом навыков разработки и анализа эволюционных операторов генетического алгоритма для решения задачи оптимизации непрерывной вещественнозначной функции.

Оборудование и программное обеспечение

Для выполнения лабораторной работы потребуется:

- браузер с доступом к сети Интернет;
- Java JDK версии 1.8 и выше;
- Watchmaker framework версии 0.7.1.

Краткие теоретические сведения

Предположим, что имеется вектор вещественных чисел размерностью D,

$$\vec{X} = [x_1, x_2, x_3, \dots, x_D]$$

и целевая функция $f: R^D \to R$. Оптимизация является процессом нахождения наилучших, приближенных экстремальных значений целевой функции в некотором векторном пространстве решений (пространстве поиска). Задачей минимизации является нахождение такого решения $\vec{X}^*$, которое минимизировало бы целевую функцию f:

$$\forall\, \vec{X}^* \neq \vec{X}\colon f(\vec{X}^*) < f(\vec{X}).$$

Максимизация формулируется аналогичным образом, поскольку

$$\max\{f(\vec{X})\} = -\min\{-f(\vec{X})\}.$$

Представление решений ГА в виде вектора вещественных чисел часто является целесообразным при решении большого круга задач. Например, при решении инженерных задач, где необходимо подобрать физически исчисляемые параметры некоторого проектируемого объекта (длину, ширину компонента или угол между компонентами). Хорошим примером является дизайн максимально устойчивой конструкции балки для спутника [3]. Разработанный для этой задачи ГА нашёл намного более устойчивую конструкцию балки спутника, однако её вид сильно отличался от известного ранее логичного и сформированного инженерами решения. Данный пример наглядно показывает возможности эволюционных алгоритмов производить креативные решения.

В реальном мире большинство задач оптимизации имеет многомодальные целевые функции, т.е. имеет несколько оптимумов.

Существуют известные примеры тест-функций для оптимизации [4], например, функция Экли (рисунок 2.1).

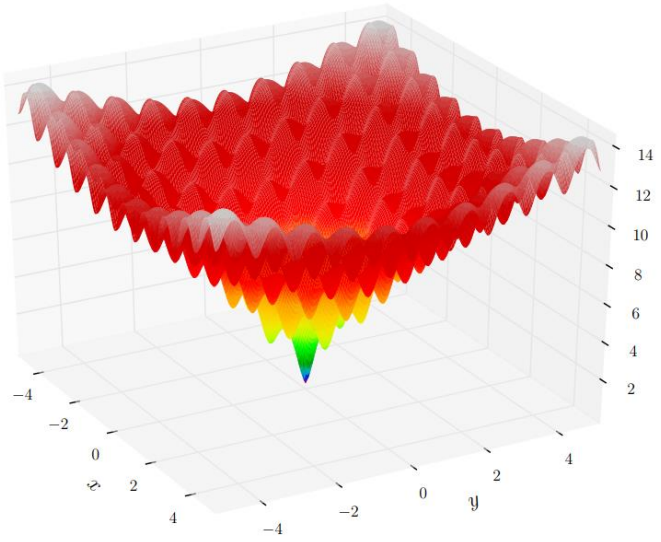


Рисунок 2.1 – Функция Экли для двух переменных

Нахождение глобального оптимума является намного более сложной задачей, чем оптимизация унимодальной функции, поскольку существует вероятность схождения алгоритма в локальном оптимуме. Целевая функция или фитнес-функция (в терминологии эволюционных алгоритмов) представляет собою ландшафт, по которому двигаются индивиды популяции решений с целью нахождения оптимума.

На рисунке 2.2 изображён ландшафт смещенной функции Экли в виде контуров. В данном примере глобальный минимум находится в точке (1, 2), а популяция решений изображена синими точками.

Для решения задачи оптимизации вещественнозначной функции ГА необходимо реализовать следующие функции: представление решения, способ инициализации популяции, операцию кроссовера и операцию мутации. Наиболее очевидным представлением решений в данном случае является массив вещественных чисел, размерность которого совпадает с размерностью проблемы. Инициализация начальной популяции может осуществляться случайной генерацией массивов в заданной области определения функции. Также, для лучшего разброса решений по ландшафту фитнес-функции, индивиды могут быть инициализированы с определенной дистанцией между собой.

Существует два основных типа кроссовера для вещественнозначного представления: дискретный и арифметический. Предположим, что имеется два родительских решения x и y, каждый из которых представлен в виде набора генов x_i и y_i. При дискретном способе дочернее решение z наследуют часть генов от одного предка, а другую часть от второго предка $z_i = x_i \; or \; y_i$ случайно. При арифметическом кроссовере часть генов дочернего решения наследуется от одного предка, а значения оставшихся генов рассчитываются как промежуточные значения между генами двух предков $z_i = \alpha \, x_i + (1 - \alpha) y_i$.

Графический пример изображён на рисунке 2.3. Красными точками являются два родительских предка x и y. Возможный результат дискретного кроссовера представлен зелёными точками z_1 и z_2, а результат полного арифметического кроссовера с параметром $\alpha = 0.5$ синей точкой z_3. Как можно заметить, дискретный кроссовер является частным случаем арифметического с параметром $\alpha = 1$. Существует и множество других вариаций рассмотренных способов реализации кроссовера, а также их комбинации.

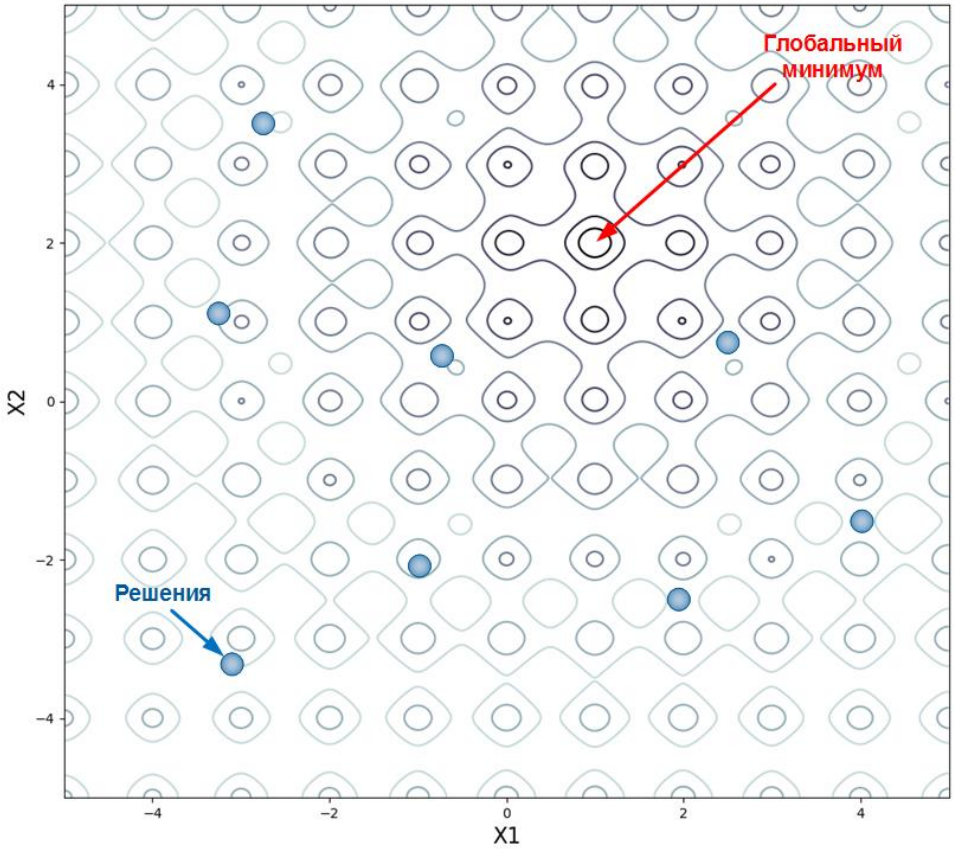


Рисунок 2.2 – Ландшафт смещенной функции Экли и популяция решений

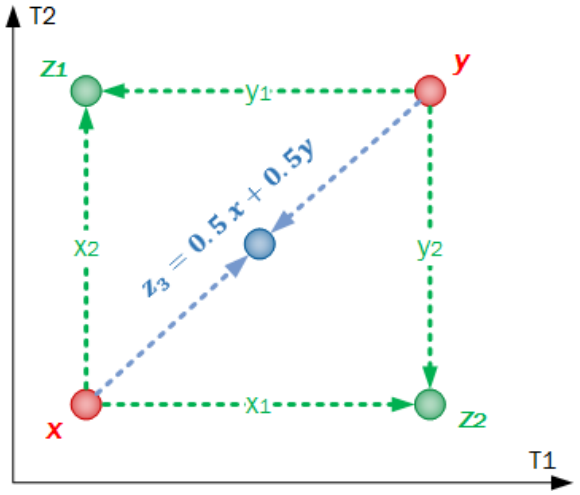


Рисунок 2.3 – Пример дискретного (z_1 и z_2) и арифметического кроссовера (z_3) для веществennозначного представления решений

Мутация для вещественнозначного представления подразумевает изменение значения одного или нескольких генов на новое случайное значение в области их определения: $< x_1, \dots, x_n > \rightarrow < x'_1, \dots, x'_n >$, где $x_i,\ x'_i \in [a_i, b_i]$. По типу изменения можно выделить два типа мутации: равномерная и неравномерная. При равномерном изменении значения гена в рамках нижней и верхней границ $x'_i = U(a_i, b_i)$ теряется его прошлое состояние, но зато такой тип мутации позволяет выйти за границы локального оптимума. Неравномерная мутация подразумевает сдвиг решения по одной или нескольким осям в соответствии с некоторым законом распределения. Как и в большинстве случаев будем подразумевать нормальный закон распределения $\mathcal{N}(0, \sigma)$. Неравномерная мутация позволяет проводить локальный поиск за счёт небольших изменений в окрестностях текущей локации индивида. На рисунке 2.4 представлены примеры обоих вариантов мутации.

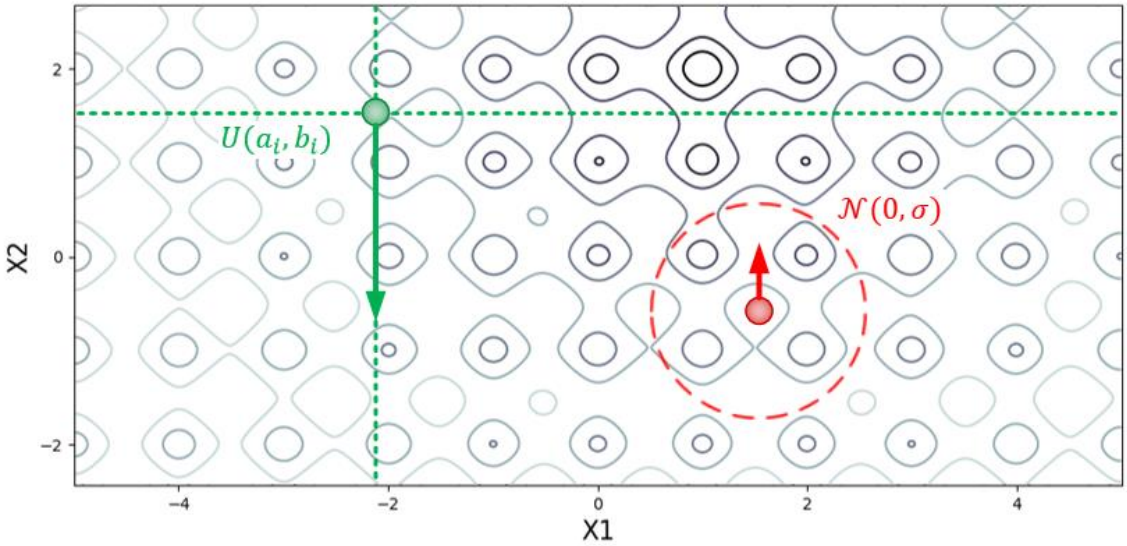


Рисунок 2.4 – Пример равномерной (слева) и неравномерной (справа) мутации

При работе с неравномерной мутацией возникает необходимость выбора параметра σ. Параметр может быть подобран вручную на основе знаний о границах задачи и серии экспериментальных запусков алгоритма. Также, параметр σ можно адаптировать и изменять в процессе эволюции в виде гипер-параметра алгоритма или даже в виде составляющей индивидов (генотипа) популяции. В последнем случае кодирование решений расширяется следующим образом: $< x_1, x_2, \dots, x_n, \sigma >$. Поскольку области определения компонент вектора могут отличаться друг от друга, то для каждой компоненты x_i может быть эффективней иметь свой параметр σ_i, что также может быть закодировано в генотип напрямую: $< x_1, x_2, \dots, x_n, \sigma_1, \dots, \sigma_n >$. Такое кодирование усложняет размерность проблемы поиска, но в тоже время помогает улучшить производительность самого поиска.

Таким образом, всегда, при выборе того или иного подхода к реализации мутации возникает компромисс между производительностью, исследованием (exploration) и локальным поиском (exploitation). На рисунке 2.5 продемонстрированы примеры различных способов выбора параметров неравномерной мутации, в том числе в ситуациях исследования и локального

поиска. Окружности и эллипсы определяют области вероятного появления индивида в процессе его мутации. Под исследованием понимается возможность алгоритма осматривать неизвестные участки пространства поиска. Локальный поиск в свою очередь подразумевает осмотр окрестностей с целью уточнения и улучшения текущего решения. Как правило, исследование в большей мере необходимо на начальном этапе работы алгоритма, а локальный поиск позже, когда возможные области оптимумов уже обнаружены.

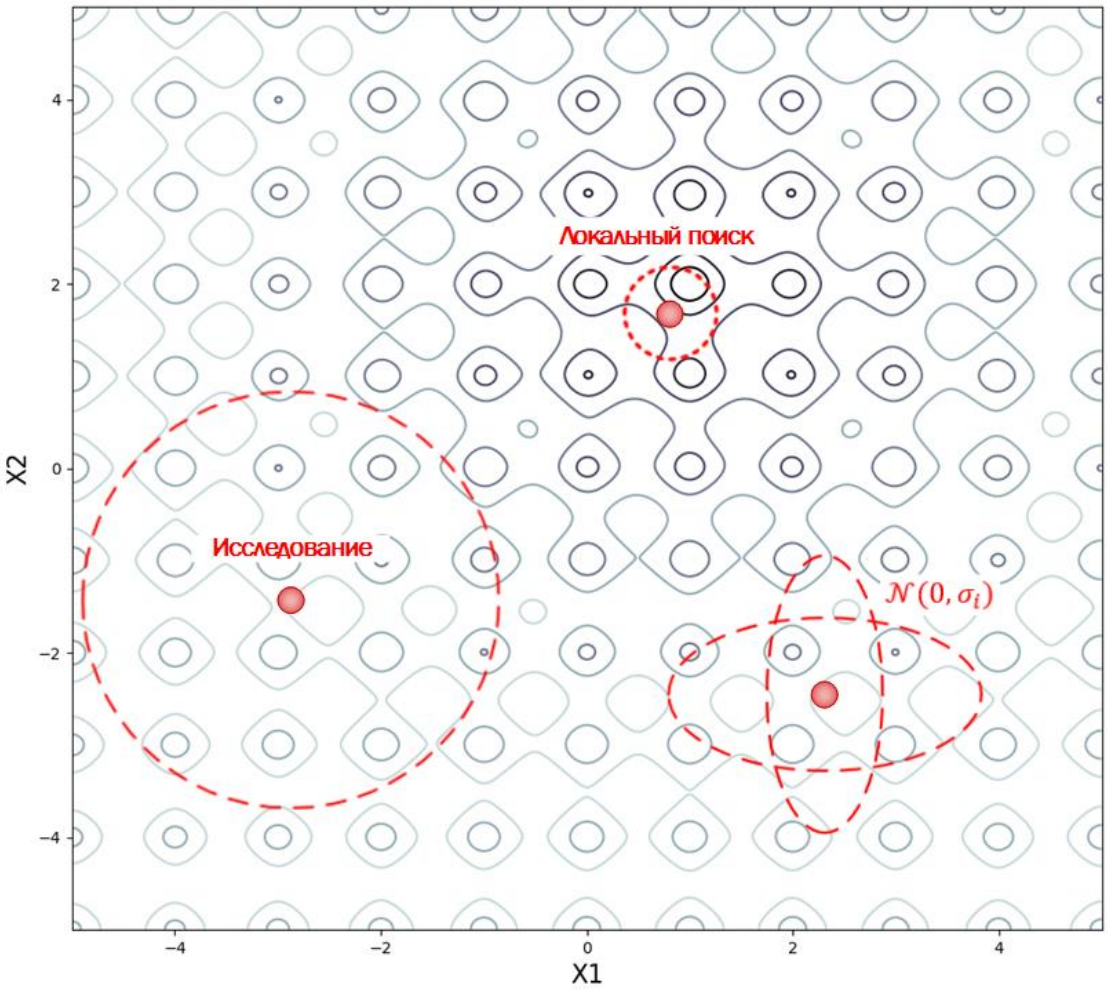


Рисунок 2.5 – Пример различных способов осуществления неравномерной мутации

Ход работы

В данной лабораторной работе предлагается решить задачу оптимизации вещественнозначной функции (**максимизации**), представленной в виде «чёрного ящика» посредством ГА. Для начала работы необходимо скачать шаблон для проекта:

https://gitlab.com/itmo_ec_labs/lab2

Проект необходимо открыть как новый из существующего источника и выбрать Maven в качестве утилиты для сборки проекта. Проект основан на фрэймворке Watchmaker и состоит из следующих классов:

- MyAlg.java;
- FitnessFunction.java;
- MyFactory.java;
- MyCrossover.java;
- MyMutation.java.

В классе MyAlg.java описывается основная структура ГА с его параметрами и необходимыми функциями.

Параметр **int** dimension = 2 определяет размерность проблемы, т.е. количество переменных в векторе решения.

Параметр **int** populationSize = 10 устанавливает количество индивидов в популяции, а **int** generations = 10 количество итераций алгоритма, т.е. поколений, которое будет сгенерировано в процессе эволюции решений.

Объект CandidateFactory<**double**[]> factory указывает на реализацию инициализации начальной популяции решений.

Список EvolutionPipeline<**double**[]> pipeline включает список operators, который содержит в себе реализацию операторов мутации и кроссовера.

Объект SelectionStrategy<Object> selection определяет способ селекции индивидов. В данном шаблоне указана стратегия рулетки.

Целевая функция определена и уже реализована в FitnessEvaluator<**double**[]> evaluator. В рамках лабораторной работы изменять фитнес-функцию не нужно!

Объект EvolutionEngine<**double**[]> algorithm агрегирует все необходимые для ГА компоненты, а также определяет стратегию эволюции. В данной работе выбрана стратегия стабильного состояния (SteadyStateEvolution) при которой выжившие индивиды для следующего поколения выбираются из предыдущего и из новых решений, полученных в ходе кроссовера. Также MyAlg.java является точкой запуска алгоритма посредством функции algorithm.evolve(populationSize, 1, terminate). Для выполнения работы необходимо реализовать три функции: инициализацию, кроссовер и мутацию.

Инициализацию индивида необходимо реализовать в классе MyFactory.java. В рамках лабораторной работы есть **ограничение**:

- область определения всех переменных целевой функции: $[-5, 5]$;
- область значений функции: $[0, 10]$.
- глобальный оптимум: 10.

Данное ограничение следует соблюдать не только на этапе инициализации, но и в процессе кроссовера и мутации.

Следующим шагом реализации алгоритма является разработка алгоритма кроссовера в классе MyCrossover.java. Наследуемая от класса фрэймворка функция принимает на вход два родительских решения и требует на выходе список с **новыми** решениями.

Последней функцией, которую нужно реализовать, является оператор мутации в классе MyMutation.java. На вход функции подаётся список со всеми индивидами в популяции. Таким образом, вопрос какие индивиды и каким образом будут меняться – является задачей реализации функции.

Важно. Вне зависимости от способа реализации представления решений, в фитнес-функцию должен подаваться массив вещественных чисел с размерностью равной размерности проблемы (**int** dimension).

После реализации описанных выше операторов необходимо провести экспериментальные исследования над производительностью алгоритма. Задачей экспериментов является достижение алгоритмом решений близких к значению фитнес-функции 10. При размерности проблемы 2 (**int** dimension = 2) алгоритм должен легко вырабатывать почти оптимальное решение за малое количество итераций.

За вывод данных о прогрессе алгоритма отвечает объект algorithm.addEvolutionObserver, созданный в классе MyAlgorithm.java.

Следующий этап лабораторной работы – увеличение размерности проблемы. Предположим, что существуют ограничения для параметров:

- **int** populationSize < 100;
- **int** generations < 10000.

Необходимо провести серии запусков при размерностях проблемы: 10, 20, 50, 100 для анализа производительности реализованного ГА. В процессе экспериментов необходимо настроить параметры размера популяции и другие дополнительные параметры алгоритма и операторов, если таковы были введены.

По результатам экспериментов необходимо заполнить следующую таблицу 2.1 на основе полученных результатов экспериментов. Для заполнения результатов необходимо провести минимум по 10 запусков алгоритма для каждой размерности задачи. В столбце «Результат» необходимо ввести усредненное значение по серии запусков. При заполнении столбца «Количество итераций» можно ввести усреднённые значения количества итераций, при которых алгоритм сошёлся, т.е. смог найти конечное решение.

Таблица 2.1 Результаты экспериментов по производительности алгоритма при увеличении размерности проблемы

Размер проблемы	Размер популяции	Количество итераций	Результат
2			
10			
20			
50			
100			

В отчете также необходимо кратко описать каким образом были организованы операторы инициализации, кроссовера и мутации. Также необходимо описать введенные параметры и их значения, полученные в ходе настройки. Удовлетворительным результатом работы алгоритма является нахождение решений со значением фитнес функции не менее 9.5.

Вопросы

1. Что важнее, кроссовер или мутация?
2. Как влияет значение параметра «размер популяции» на производительность и эффективность алгоритма?
3. Важно ли знать область определения переменных целевой функции?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Satellite Truss Design, http://www.soton.ac.uk/~ajk/truss/welcome.html.
4. Test optimization functions, https://www.sfu.ca/~ssurjano/optimization.html.

Лабораторная работа № 3
Генетический алгоритм для решения задачи коммивояжёра

Цель работы

Целью данной работы является получение навыков разработки эволюционных алгоритмов для решения комбинаторных задач на примере задачи коммивояжёра.

Краткие теоретические сведения

Большое множество задач оптимизации имеют комбинаторный или дискретный характер. Одним из наиболее известных примеров является задача коммивояжёра, которая формулируется следующим образом. Существует n городов $V = \{v_i\}$ с заданными расстояниями между ними: $E = \{e_{ij}\}\ 1 \leq i, j \leq n$. Необходимо построить маршрут, проходящий через все города с возвратом в исходный город, с наименьшей итоговой пройденной длиной. Предполагается, что каждый город может быть посещён только один раз (за исключением исходного).

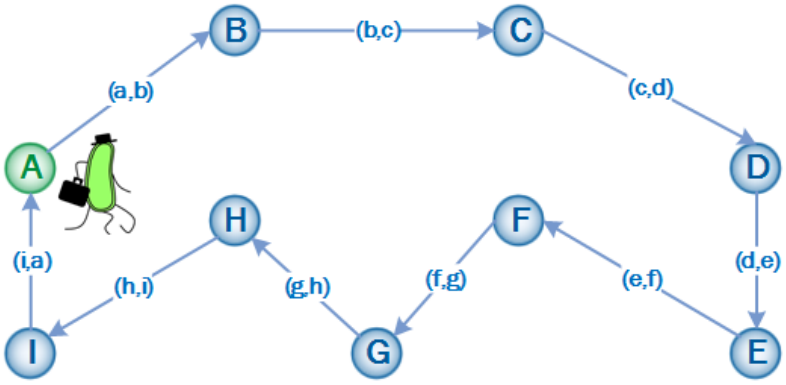


Рисунок 3.1 – Пример представления задачи коммивояжёра

Таким образом, решением задачи является перестановка городов – упорядоченный список длиной n, а результатом является полученная дистанция данного маршрута. В рамках разрабатываемого эволюционного алгоритма вид решения может быть изменён с целью более эффективных реализаций внутренних функций, а функция оценки качества решений может быть также преобразована для улучшения отличия между решениями.

ГА удобен для решения комбинаторных задач в силу того, что индивиды популяции легко представимы в комбинаторном виде, а также в силу комбинаторной природы операций мутации и кроссовера.

При решении комбинаторных задач сложность задачи резко (экспоненциально) возрастает с увеличением размерности задач. Также изменение одной переменной решения влияет и на другие. Таким образом, при решении задачи эволюционными алгоритмами следует учитывать специфику задачи, а также возможные ограничения. Например, если решение должно составлять комбинацию всех заданных уникальных объектов по одному разу,

то реализация мутации в виде замены одной переменной на любой другой объект является недопустимой (рисунок 3.2).

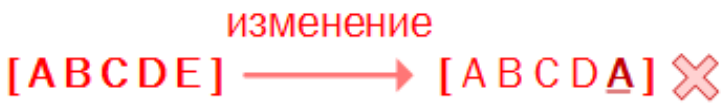


Рисунок 3.2 – Пример невалидной мутации для комбинаторного представления

Для реализации оператора мутации для комбинаторного представления можно выделить несколько наиболее очевидных стратегий. Предположим, что наши решения представляют из себя комбинации 9 чисел от 1 до 9. На рисунке 3.3 представлена структура решения (хромосомы). В терминологии эволюционных алгоритмов можно выделить следующие понятия. Структура хромосомы определяется определённым и упорядоченным набором аллелей. Аллель представляет собой индекс или позицию для генов. Ген представляет собой размещенный объект в позиции аллели. В данном примере, геном является любое из чисел от 1 до 9, а аллелью является индекс элемента массива.

Рисунок 3.3 – Структура хромосомы

Первый очевидный тип мутации – перестановка (**swap mutation**), где случайно выбираются две аллели по которым происходит перестановка их генов.

Вставка (**insert mutation**) – случайно выбираются два гена, затем второй помещается в позиции за первым геном.

Перемешивание (**scramble mutation**) – случайно выбирается диапазон аллелей, в рамках которого все гены перемешиваются.

Инверсия (**inversion mutation**) – случайно выбираются две аллели, между которыми все гены меняют свой порядок на противоположный.

Примеры стратегий мутации представлены на рисунке 3.4. Первые три варианта мутации представляют из себя небольшие изменения и больше подходят для локального поиска, в то время как инверсия представляет более существенное изменение.

Реализация оператора кроссовера также имеет свою специфику при работе с комбинаторными задачами. Как и в случаее мутации, генерируемые кроссовером новые решения должны быть валидны – содержать все элементы в единственном числе. Рассмотрим наиболее известную стратегию кроссовера – упорядоченный кроссовер (**order crossover**). Пример для построения одного потомка на основе двух родительских решений представлен на рисунке 3.5.

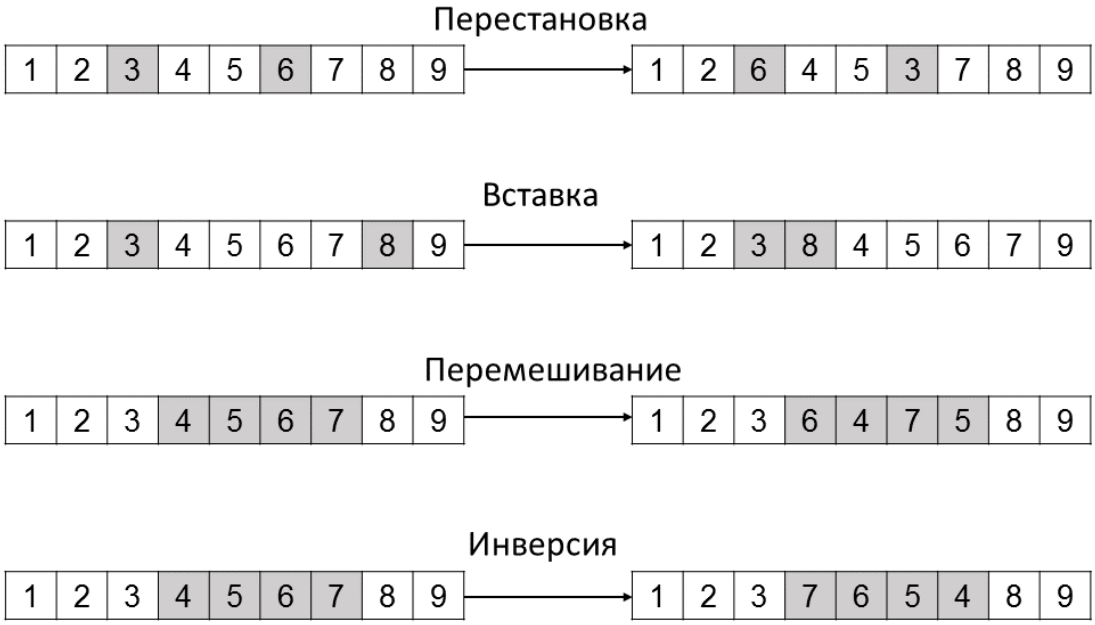


Рисунок 3.4 – Примеры мутации для комбинаторного представления

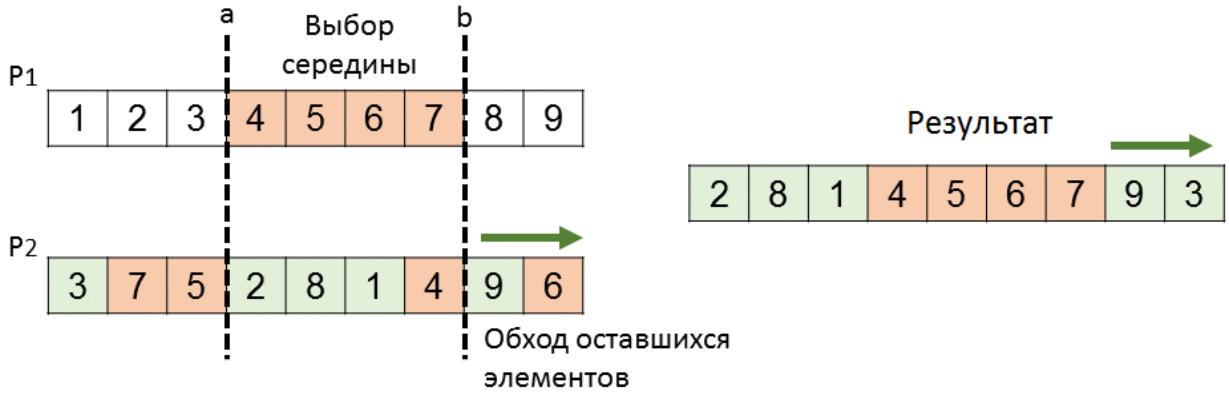


Рисунок 3.5 – Пример реализации упорядоченного кроссовера

Первый шаг алгоритма – выбор двух точек a и b. Потомок целиком наследует середину от первого родителя P_1. Второй шаг алгоритма – обход всех незадействованных генов в порядке второго родителя P_2, начиная от позиции b. Такая реализация является наиболее очевидной и простой. Однако существует и множество других стратегий [2], которые встречаются при решении различных задач.

Ход работы

В данной лабораторной работе решается задача коммивояжёра. Данные и описание тестовых задач доступны по адресу:

http://www.math.uwaterloo.ca/tsp/vlsi/index.html

Для начала возьмём задачу XQF131, в которой необходимо построить оптимальный маршрут по 131 городам. В файле с данными представлены 3 столбца, где первый столбец – индексы городов, второй столбец – координата

по оси X, третий столбец – координата по оси Y. Формат данных представлен ниже в листинге 3.1.

```
Индекс, X, Y
1 0 13
2 0 26
3 0 27
4 0 39
5 2 0
6 5 13
7 5 19
8 5 25
9 5 31
...
```

Листинг 3.1 – Формат входных данных для задачи коммивояжёра

В описании проблемы также находится и необходимый результат – оптимальный маршрут, пример которого представлен на рисунке 3.6.

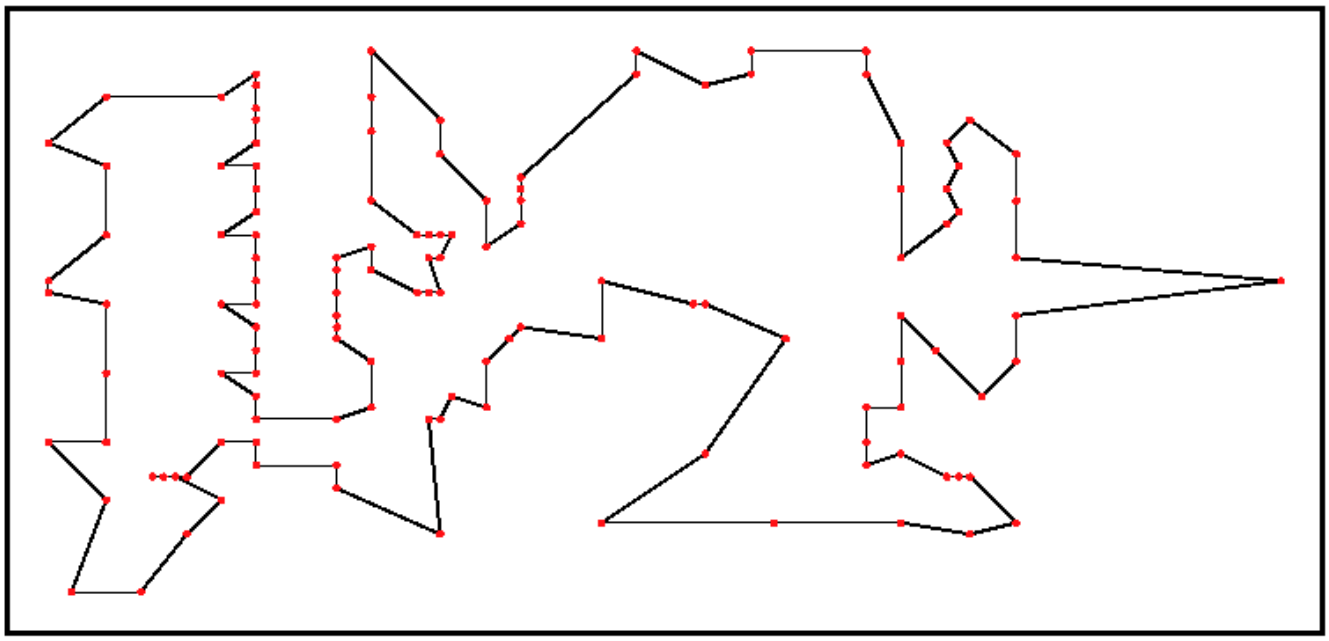


Рисунок 3.6 – Оптимальный маршрут для задачи XQF131 с длиной 564

Необходимо загрузить шаблон проекта по ссылке:
https://gitlab.com/itmo_ec_labs/lab3

Шаблон также основан на фреймворке Watchmaker. Проект содержит следующие классы:

- TspAlg.java – основной класс проекта;
- TspCrossover.java – реализация оператора кроссовера;
- TspFactory.java – реализация оператора инициализации решений;
- TspFitnessFunction.java – реализация фитнес-функции;
- TspMutation.java – реализация оператора мутации;
- TspSolution.java – реализация представления решений.

Основным отличием от лабораторной работы №2 является то, что теперь необходимо реализовать представление решений в классе TspSolution.java и фитнес-функцию в классе TspFitnessFunction.java. Важно отметить, что в классе фитнес-функции, функция *isNatural()* теперь возвращает значение *false*. Это необходимо для переключения алгоритма в режим решения задачи минимизации. Для реализации фитнес-функции необходимо считать файл с входными данными проблемы (например, xqf131.tsp). Рекомендуется проводить считывание входного файла при инициализации объекта с фитнес-функцией, передав ей на вход имя проблемы или путь к файлу. При реализации представления решений рекомендуется расширить стандартную функцию *toString()* для использования при выводе прогресса эволюции алгоритма.

При реализации инициализации, кроссовера и мутации требуется ввести все необходимые параметры и настроить их для достижения лучшей производительности алгоритма. Также можно варьировать **размер популяции**, **количество итераций**, стратегию селекции (например, TournamentSelection), схему алгоритма (например, GenerationalEvolutionEngine).

Для проведения экспериментальных исследований необходимо загрузить с источника тестовых проблем несколько экземпляров задач с разной размерностью (количеством городов). Для каждой задачи необходимо настроить параметры алгоритма и провести минимум по 10 запусков. В таблице 3.1 требуется ввести усреднённые значения по результатам серий запусков.

Таблица 3.1 Результаты решения тестовых экземпляров задачи коммивояжёра

Проблема	Размер	Параметры popsize и gens	Длина маршрута	Количество итераций до сходимости	Оптимальный маршрут
XQF131	131	10; 10	600	8	564
…					

Количество итераций до сходимости может быть вычислено путём хранения лучшего решения в процессе эволюции и значения итерации, на которой это решение было получено. В отчёте необходимо описать как было реализовано представление решений, операторы мутации и кроссовера, в том числе введенные параметры и их значения при проведении экспериментов.

Вопросы

1. Можно ли определить, что полученное решение является глобальным оптимумом?
2. Можно ли допускать невалидные решения (с повторением городов). Если да, то как обрабатывать такие решение и как это повлияет на производительность алгоритма?
3. Как изменится задача, если убрать условие необходимости возврата в исходную точку маршрута?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Тестовые экзмепляры проблем для задачи коммивояжёра, http://www.math.uwaterloo.ca/tsp/data/index.html.

Лабораторная работа № 4
Проектирование эволюционного алгоритма для задачи расстановки ферзей

Цель работы

Целью данной работы является освоение всего цикла разработки эволюционных алгоритмов, начиная с анализа проблемы и проектирования, заканчивая настройкой параметров и анализом эффективности.

Краткие теоретические сведения

Перед разработкой эволюционного алгоритма для решения некоторой проблемы, необходимо задать себе ряд вопросов.

Какова цель разработки алгоритма?

- найти некоторое решение задачи;
- найти глобально оптимальное решение задачи;
- находить хорошие решения при частом запуске алгоритма;
- найти множество разнородных экземпляров хороших решений;
- разработать алгоритм, который будет эффективней существующих.

Необходимо проанализировать контекст решаемой проблемы. Насколько быстро должно быть получено решение? Что подразумевается под решением проблемы? Как оценивается качество решения? Существуют ли ограничения? Например, функция оценки качества решений может быть осложнена следующими составляющими:

- нестационарность, когда оценка качества одного и того же решения зависит от времени;
- зашумленность, в функции оценки качества присутствует компонент случайности;
- ограниченность, невозможность вычисления функции при определенных условиях;
- многокритериальность, необходимость нахождения компромисса между несколькими критериями оценки, а также нахождения множества недоминантных решений;
- интерактивность, в роли функции оценки качества выступает человек-эксперт.

После анализа проблемы можно приступить к проектированию алгоритма. Существует большое множество эволюционных алгоритмов, кроме ГА:

- метод роя частиц (Particle Swarm Optimization), основан на модели движения стаи птиц;
- гравитационный алгоритм (Gravitational Search Algorithm), основан на принципах закона всемирного тяготения;
- муравьиный алгоритм (Ant Colony Optimization), основан на поведении муравьёв при поиске источников питания;

- гармонический поиск (Harmony Search), основан на принципах сочинения музыки;
- алгоритмы чёрной дыры или сверхновой;
- алгоритм капель воды и многие другие [4].

Нужно понять, как решения могут быть представимы в рамках алгоритмов и выбрать наиболее предпочтительный алгоритм. Также, следует заранее спроектировать, как будет осуществляться процесс вариации решений.

После реализации алгоритма, необходимо перейти к анализу его производительности и эффективности. В основном все эволюционные алгоритмы имеют итерационный характер, где на каждой итерации происходит вариация решений (популяции) и оценка их качества. Таким образом, для анализа эффективности алгоритма могут быть использованы следующие характеристики:

- шанс успеха, вероятность нахождения необходимого решения при множественном запуске алгоритма;
- среднее качество решений;
- скорость алгоритма.

Скорость алгоритма может быть представлена как:

- реальное время выполнения алгоритма;
- количество итераций алгоритма;
- количество вычислений фитнес функции.

В общем, количество вычислений фитнес функции является наиболее честным показателем скорости алгоритма, поскольку не зависит от размера популяции или от качества реализации самой программы.

Все эволюционные алгоритмы стохастичны, поэтому необходимо проводить серии запусков для получения устойчивого результата анализа эффективности. Также, важной проблемой является настройка и контроль параметров алгоритма. Настройка параметров является сложной и время-затратной задачей. Для её решения, в зависимости от цели разработки алгоритма, могут быть использованы следующие подходы:

- интуитивный подбор руками;
- поиск по сетке параметров;
- мета-эволюционный алгоритм;
- методы на основе моделирования (например, Sequential Parameter Optimization [3]).

При анализе эффективности алгоритма, требуется также оценить его возможности масштабирования, т.е. способность решения задачи большей размерности, а также влияния параметров при изменении размерности задачи.

В настоящее время существует большое число фрэймворков и инструментов для построения эволюционных алгоритмов со встроенными примерами, схемами распределения, параллелизации и анализа алгоритмов.

В рамках лабораторных работ за основу взят фреймворк Watchmaker, из-за наиболее интуитивно понятного построения и легкой конструкции базовых схем эволюции. Можно привести несколько других примеров:

- Jenetics (http://jenetics.io/), Java, основан на эффективном использовании возможностей Java Streams;
- ECJ (https://cs.gmu.edu/~eclab/projects/ecj/), Java, включено огромное количество схем алгоритмов для решения разнородных задач;
- DEAP (https://deap.readthedocs.io/en/master/), популярная библиотека для языка Python, с возможностью параллелизации;
- JMetal (http://jmetal.sourceforge.net/), обширная библиотека для Java, также есть адаптация для C++.

В данной лабораторной работе решается классическая **задача расстановки ферзей**. Дана шахматная доска NxN, на которой необходимо расставить N ферзей таким образом, чтобы они не били друг друга (не пересекались по горизонталям, вертикалям и диагоналям). Пример решения для задачи с N=8 представлена на рисунке 4.1.

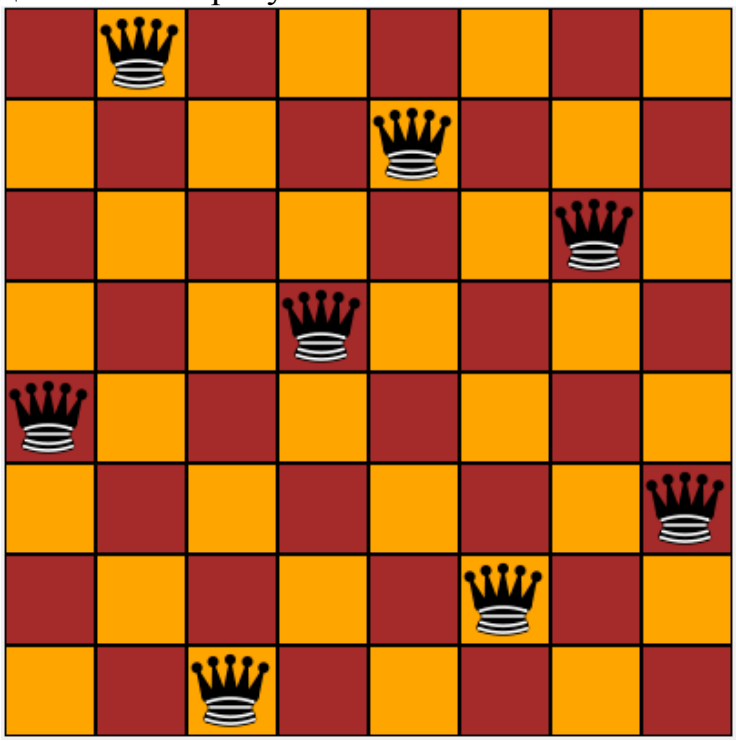


Рисунок 4.1 – Пример решения расстановки ферзей на доске 8x8.

Ход работы

1. Выбрать фрэймворк (можно продолжить в Watchmaker) для алгоритма или сделать свою реализацию.
2. Выбрать эволюционный алгоритм (ГА или другой).
3. Проанализировать задачу, выделить критерий оптимизации или ограничения.
4. Спроектировать решение поставленной задачи, вид решения, способы вариаций и обработки возможных ограничений.
5. Сформировать условия терминации или сходимости к алгоритму.
6. Установить характеристики для измерения эффективности алгоритма.
7. Настроить параметры алгоритма.
8. Провести серии запусков при разных значениях N и провести анализ эффективности.

В отчёте необходимо описать выполнение каждого пункта хода работы и заполнить результаты на основе таблицы 4.1.

Таблица 4.1 Результаты производительности алгоритма

N	Характеристика 1	Характеристика 2	…	Характеристика K
4				
8				
…				

Вопросы

1. Является ли задача оптимизационной или ограниченной?
2. Как растёт сложность задачи при увеличении размерности?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Bartz-Beielstein T., Lasarczyk C. W. G., Preuß M. Sequential parameter optimization //Evolutionary Computation, 2005. The 2005 IEEE Congress on. – IEEE, 2005. – Т. 1. – С. 773-780.
4. Rajakumar R., Dhavachelvan P., Vengattaraman T. A survey on nature inspired meta-heuristic algorithms with its domain specifications //Communication and Electronics Systems (ICCES), International Conference on. – IEEE, 2016. – С. 1-6.

Лабораторная работа № 5
Распределенные эволюционные алгоритмы

Цель работы

Целью данной работы является освоение принципов построения распределенных и параллельных эволюционных алгоритмов для повышения их производительности и эффективности.

Краткие теоретические сведения

При возрастании сложности решаемых задач возникает необходимость в разработке более эффективных и производительных эволюционных алгоритмов, способных ускорить либо сам вычислительный процесс (производительность), либо улучшить процесс поиска оптимальных решений (эффективность). Под термином распределенные эволюционные алгоритмы стоит понимать схемы алгоритмов, которые позволяют произвести не только параллельный расчёт с целью ускорения алгоритма, но и схем, влияющих на организацию эволюции в целом. Например, распределенные эволюционные алгоритмы могут быть построены таким образом, чтобы повысить разнородность в исследовании пространства решений, увеличить статистическую устойчивость алгоритма, разбить исходную проблему на подзадачи меньшей размерности, задать особую стратегию взаимодействия решений внутри популяции.

Наиболее очевидным способом улучшения работы эволюционного алгоритма является параллельная реализация его внутренних операций. Так мы получаем **master-slave** модель эволюционного алгоритма. В основе модели лежит концепция обработки общепопуляционных операций (кроссовер, мутация, селекция) на мастер-узле, а вычисление фитнес-функции для каждого индивида осуществляется посредством передачи вычислений на ведомые узлы. Такая модель распределенного алгоритма не влияет на логику эволюционного процесса, а только позволяет ускорить процесс вычисления. Основным узким местом такой схемы является предположение того, что вычисление фитнес-функции является наиболее вычислительно-ёмкой процедурой. Если фитнес-функция является слишком простой и быстрой операцией, то передача данных между мастером и ведомыми может занять больше времени, чем расчёт на одном узле, что может даже понизить производительность алгоритма. Однако, существуют различные решения указанного недостатка. Например, над переданным решением на ведомый узел может производится дополнительная локальная оптимизация. Еще одним недостатком схемы является синхронность. Для продолжения эволюции и перехода к следующей итерации алгоритма, мастер-узел должен получить ответ от всех ведомых узлов. При сильной вариативности времени выполнения фитнес-функции может возникнуть проблема разбалансировки вычислений. Ускорение алгоритма зависит от количества ведомых узлов и отношения затрат на передачу данных к времени

выполнения фитнес функции. Схема master-slave модели представлена на рисунке 5.1.

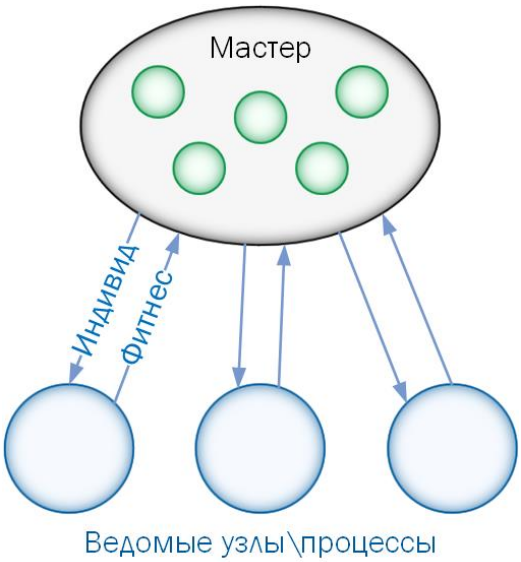


Рисунок 5.1 – Master-slave модель распределенного эволюционного алгоритма

Второй наиболее распространённой схемой распределенного алгоритма является островная **island-based** модель. Суть заключается в инициализации сразу нескольких независимых популяций (островов) с индивидами. Стратегии эволюции каждого острова могут отличаться для обеспечения разнородности поиска, а могут быть одинаковы для обеспечения лучшей устойчивости алгоритма. Предполагается, что процесс эволюции каждого острова осуществляется отдельным вычислительным узлом или процессом. Важной особенностью островной модели является дополнительная операция миграции. Миграция подразумевает обмен индивидами между популяциями с определенным периодом времени (количеством итераций). Промежуток времени между миграциями называется эпохой. Миграция сопровождается необходимостью выбора периода эпохи и стратегии селекции мигрантов. Таким образом, островная модель может позволить как улучшить производительность за счёт параллельного вычисления нескольких популяций меньшего размера, так и изменить эффективность поиска оптимальных решений в зависимости от реализации структуры островов и выбора их параметров. Схема островной модели представлена на рисунке 5.2. В целом, островная модель меньше всего подвержена влиянию со стороны специфики решаемой задачи и довольна проста для реализации.

Клеточная **cellular** модель включает в себя одну популяцию решений, однако индивиды популяции структурированы определенным образом в виде сетки. Каждый процесс или вычислительный узел выполняет обработку определенной области сетки с индивидами, а в идеальном случае – каждый процесс обрабатывает одну клетку. Взаимодействие между индивидами (кроссовер, селекция) осуществляется только между соседними клетками на основе заданной топологии сети. В процессе селекции индивидов в рамках соседей, хорошие решения могут быть распространены в разные участки сети.

Основной идеей разработки такой схемы является реализация алгоритмов под определенные вычислительные архитектуры: суперкомпьютеры, графические процессоры, FPGA, кластеры с определенной структурой узлов. Пример клеточной модели представлен на рисунке 5.3.

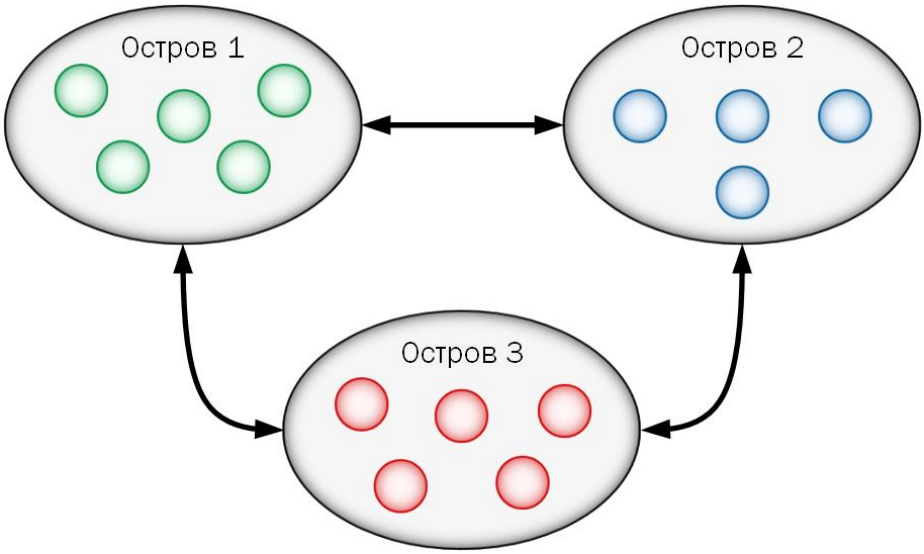


Рисунок 5.2 – Островная модель распределенного эволюционного алгоритма

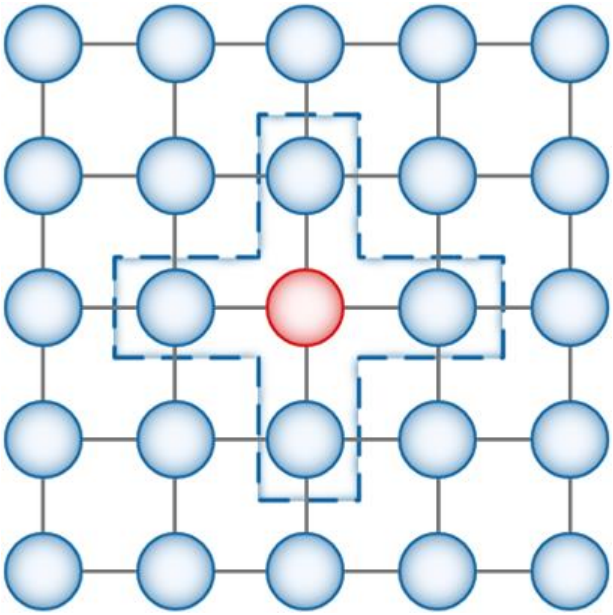


Рисунок 5.3 – Клеточная модель распределенного эволюционного алгоритма

Следующая распределенная схема направлена на уменьшение размерности исходной проблемы. Такие алгоритмы называются коэволюционными (**coevolutionary algorithms**). В основе коэволюции является разбиение решений на подзадачи меньшей размерности и их раздельная эволюция. Таким образом, коэволюция может быть рассмотрена как островная модель, где каждый остров представляет из себя эволюцию своей части от общего решения. Особенностью является этап комбинирования индивидов из разных популяций на этапе вычисления фитнес-функции для построения полноразмерных решений. Также, для реализации коэволюционной схемы необходимо выбрать способ распределения значения фитнес-функции между

частями скомбинированных решений. Пример коэволюционной схемы представлен на рисунке 5.4. По большей части, коэволюция представляет из себя усложнение вычислений и тем самым зачастую приводит к уменьшению производительности алгоритма. Однако, коэволюция позволяет повысить эффективность алгоритма при высокой размерности исходной проблемы, когда поиск в пространстве решений становится неохватываемым.

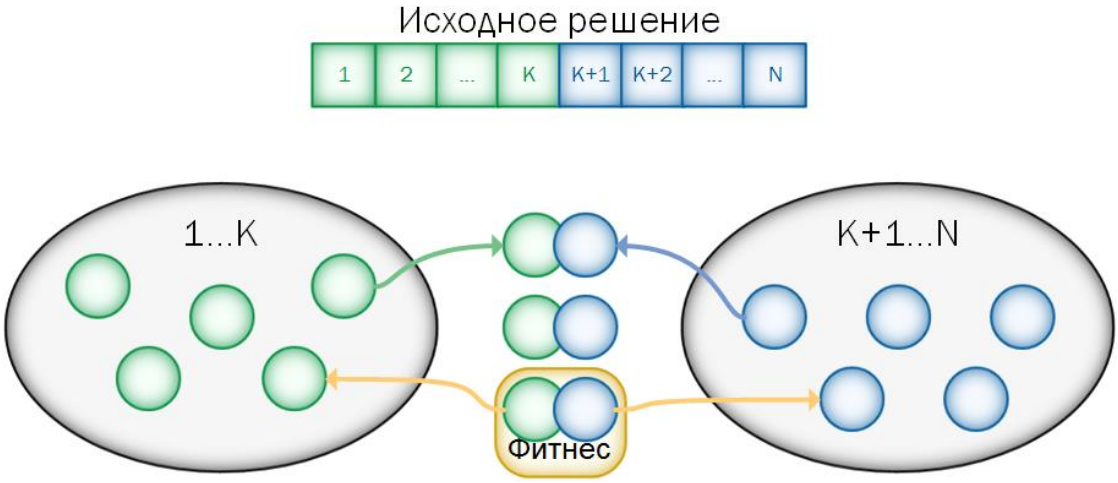


Рисунок 5.4 – Схема коэволюционного алгоритма

Все рассмотренные схемы распределенных эволюционных алгоритмов могут быть реализованы различными способами, в том числе и в виде гибридных схем (например, master-slave внутри островной модели). В зависимости от реализации и выбора параметров, целью распределенного алгоритма может являться повышение производительности или эффективности алгоритма. При разработке распределенного алгоритма необходимо учитывать специфику предметной области и вида решаемой проблемы для выбора наиболее подходящей схемы алгоритма.

Ход работы

Для выполнения данной работы могут быть использованы результаты, полученные при выполнении лабораторной работы №2. В частности, потребуется реализация функций инициализации, кроссовера и мутации для решения задачи оптимизации вещественнозначной функции. Шаблон проекта может быть загружен по ссылке:

https://gitlab.com/itmo_ec_labs/lab5

Структура проекта основана на проекте из лабораторной №2. Классы с реализацией инициализации, кроссовера и мутации остались без изменений. Класс с фитнес-функцией MultiFitnessFunction.java имеет одно отличие. Теперь фитнес-функция имеет параметр **int complexity**, который представляет собой множитель сложности вычисления фитнес-функции. Сложность заключается в повторном вычислении одной и той же функции. Таким образом, при значении параметра **complexity** = 5, время выполнения фитнес-функции приблизительно увеличится в 5 раз. В проекте присутствует два главных класса: MasterSlaveAlg.java и IslandsAlg.java. Первый является базовым алгоритмом,

который по умолчанию является master-slave моделью. Второй класс соответственно представляет реализацию островной модели.

Первый этап работы. Для анализа производительности двух моделей параллельных алгоритмов необходимо получить последовательный алгоритм. Для этих целей, в классе MasterSlaveAlg.java необходимо вызвать функцию algorithm.setSingleThreaded(**true**) после создания схемы алгоритма (AbstractEvolutionEngine<**double**[]> algorithm). При передаче в функцию значения true, алгоритм будет работать в однопотоковом режиме.

Второй этап работы. Для построения островной модели необходимо создать объект IslandEvolution<**double**[]> island_model в классе IslandsAlg.java. Для этого необходимо вызвать конструктор класса с необходимым набором параметров. В качестве стратегии миграции можно выбрать реализованную в рамках фрэймворка стратегию RingMigration. Создать островную модель можно указав количество одинаковых островов либо вручную создать список необходимых объектов класса EvolutionEngine<T>. Во втором случае имеется возможность реализовать разнородные популяции с разной стратегией эволюции.

Базовый вывод информации о прогрессе эволюции уже реализован, но необходимо реализовать функционал для замера времени работы алгоритмов. Далее, необходимо провести серии экспериментов для сравнения трёх алгоритмов: однопоточный, master-slave и островной. Каждая серия экспериментов должна проводится при одинаковом размере популяции и количестве итераций для каждого алгоритма. Важно учесть, что в случае островного алгоритма размер популяции складывается по всем островам, а количество итераций равно количеству эпох (generations), умноженному на период эпохи (epochLength). Предположим, что размерность проблемы dimension = 50, а размер популяции 100.

Каждая серия экспериментов проводится для определенного значения параметра **complexity**. В каждой серии экспериментов необходимо провести минимум 10 запусков каждого алгоритма для получения усредненных значений. Для выполнения работы необходимо провести эксперименты и заполнить результаты в соответствии с таблицей 5.1 для значений **complexity** от **0** до **5**. В отчете также нужно указать все выбранные параметры алгоритмов.

Таблица 5.1 Результаты экспериментов по производительности и эффективности распределенных алгоритмов

	complexity=0		complexity=1	
	Время выполнения	Результат	Время выполнения	Результат
Single-thread				
Master-slave				
Islands				
	complexity=2		complexity=3	
…	…	…	…	…

После заполнения таблицы необходимо построить графики зависимостей времени выполнения и результата работы каждого алгоритма от значения параметра complexity.

Вопросы

1. Какая модель алгоритма лучше при каких условиях?
2. Как повлияет увеличение размерности проблемы на алгоритмы?
3. Как повлияет увеличение размера популяции?
4. Есть ли ограничение для количества островов?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Gong Y. J. et al. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art //Applied Soft Computing. – 2015. – Т. 34. – С. 286-300.

ПРИЛОЖЕНИЕ 1

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Отчет

о выполнении лабораторной работы № X
«Тема работы»

Работу выполнил:
ст. группы <Номер группы> Фамилия И.О.
Работу принял:
Мельник М.А.

Санкт-Петербург
201_

Цель работы

Какая цель преследуется при выполнении лабораторной работы (2–3 строки).

Постановка задачи

Задача, которая решается при выполнении этой лабораторной работы (1 абзац на 0.2 – 0.3 стр.).

Краткая теоретическая часть

Краткие сведения о теме дисциплины, по которой выполняется лабораторная работа. Сведения об используемых методах, методиках, алгоритмах: свойства, достоинства, недостатки (не более 1 стр.).

Результаты

Представление результатов (промежуточные и итоговые изображения). Краткое обсуждение результатов (что означают конкретные значения) – 1–3 стр.

Заключение

Что сделано. Какие навыки и умения приобретены. Прогноз возможностей применения навыков и умений, а также полученных результатов (5–10 строк).

Миссия Университета– генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

ИНСТИТУТ ДИЗАЙНА И УРБАНИСТИКИ

http://idu.ifmo.ru

Институт дизайна и урбанистики (IDU) — это образовательный и проектно-исследовательский хаб Университета ИТМО, деятельность которого направлена на формирование устойчивых компетенций технологий производства городской среды обитания, анализа, управления, планирования инфраструктуры современных городов с использованием современных информационных технологий.

Для решения этой задачи в рамках структуры Института дизайна и урбанистики объединились лидирующие научно-технологические возможности Университета ИТМО направленные на формирование образовательных программ и подготовку специалистов, формирующих рынок труда будущего и обладающих компетенциями необходимыми для преодоления технологических барьеров, новых подходов к городским трансформациям, а также бизнес-решениям в динамичных условиях городов.

Институт дизайна и урбанистики имеет устойчивую связь с ведущими организациями, занимающимися вопросами городского развития, с органами государственной власти Санкт-Петербурга, международными образовательными и научными организациями, центрами и лабораториями, технологическими компаниями.

Институт дизайна и урбанистики Университета ИТМО является центром компетенций Университета ИТМО по реализации городского проекта «Умный Санкт-Петербург».

Программы Института дизайна и урбанистики являются новаторскими как по формату взаимодействия между преподавателями и студентами, так и по направлениям подготовки.

Мельник Михаил Алексеевич
Вишератин Александр Александрович

ЭВОЛЮЦИОННЫЕ ВЫЧИСЛЕНИЯ

Учебно-методическое пособие

В авторской редакции
Компьютерный набор и верстка М.А. Мельник.
Дизайн обложки К.Д. Мухина.
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49