Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет ИТМО

Генетические алгоритмы

Осень

2025

Лабораторная работа №3

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:

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

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).

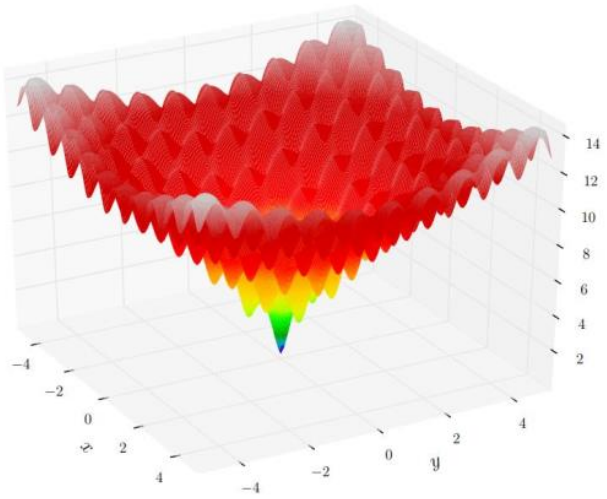

Рисунок 2.1 – Функция Экли для двух переменных

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.

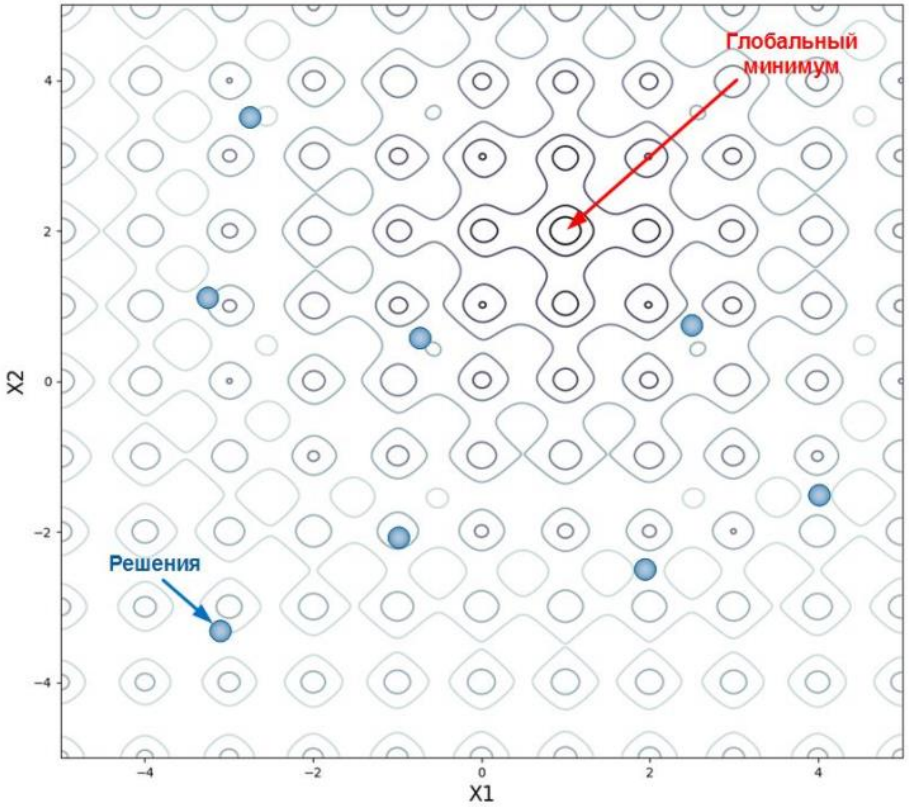

Рисунок 2.2 – Ландшафт смещенной функции Экли и популяция решений

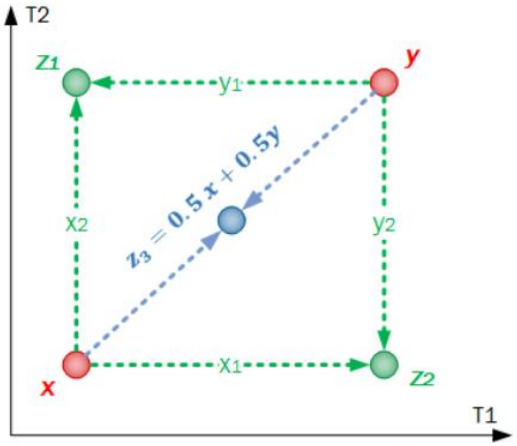

Рисунок 2.3 – Пример дискретного (z_1 и z_2) и арифметического кроссовера (z_3) для веществennозначного представления решений

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.

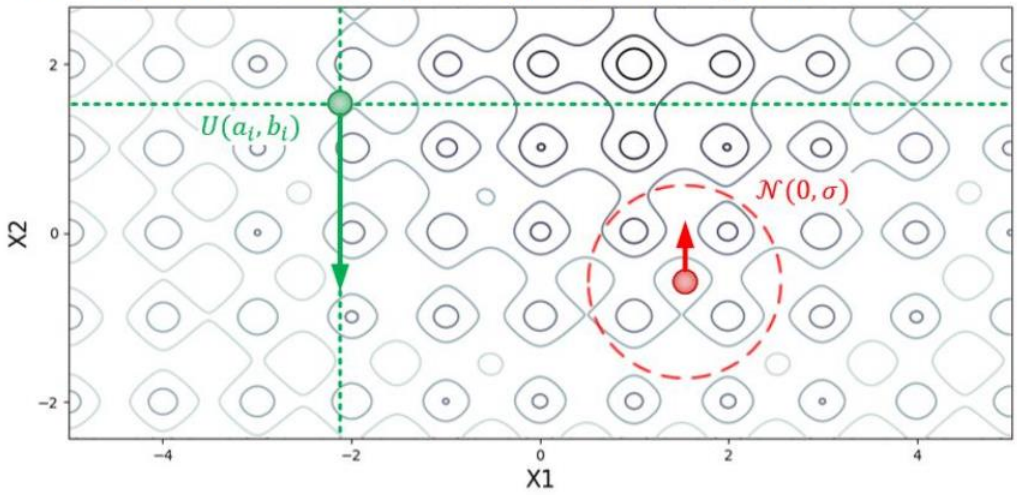

Рисунок 2.4 – Пример равномерной (слева) и неравномерной (справа) мутации

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.

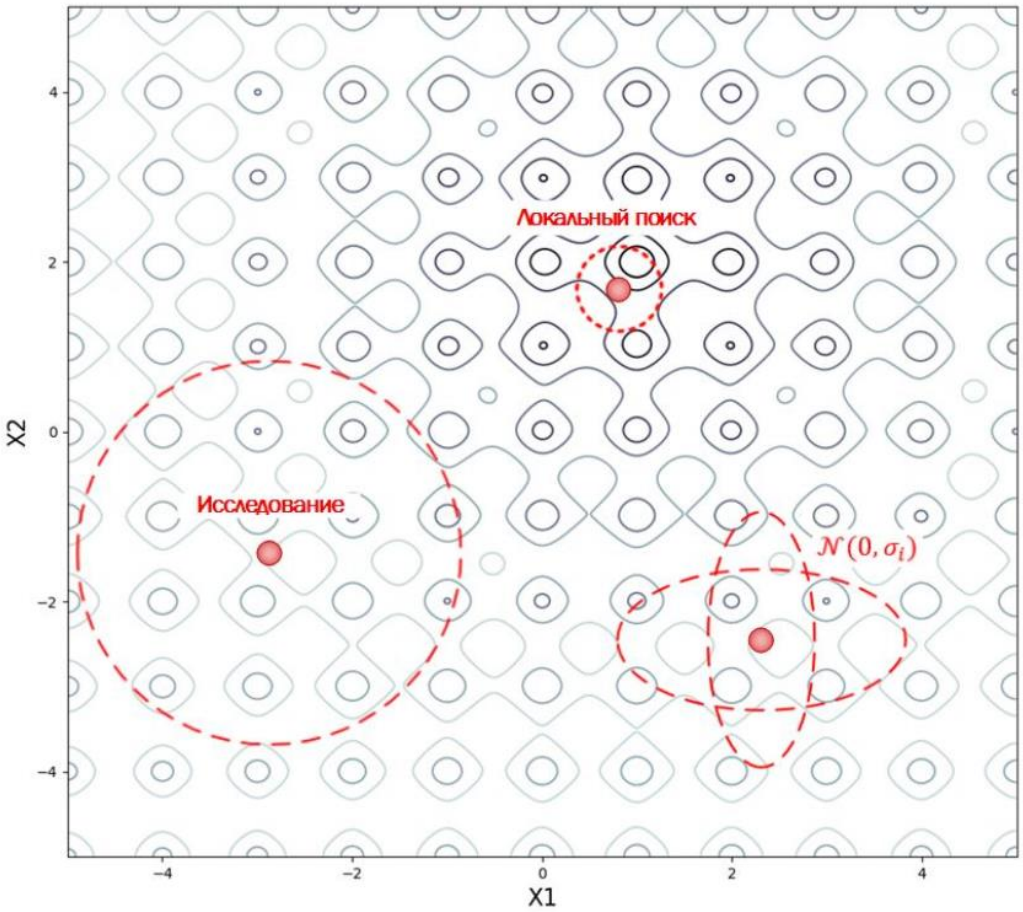

Рисунок 2.5 – Пример различных способов осуществления неравномерной мутации

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:

https://gitlab.com/itmo_ec_labs/lab2

- 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.