**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Исследование генетического алгоритма. Изучение различных кодировок генотипа»**

**Вариант – 1**

**Студент гр. 23Б15-пу**

**Абрахин Е.Д.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

Оглавление

1. [Цель работы 3](#_Toc180894431)
2. [Описание задачи (формализация задачи) 3](#_Toc180894432)
3. [Теоретическая часть 4](#_Toc180894433)
4. [Основные шаги программы 5](#_Toc180894434)
5. [Блок схема программы 7](#_Toc180894435)
6. [Описание программы 8](#_Toc180894436)
7. [Рекомендации пользователя 9](#_Toc180894437)
8. [Рекомендации программиста 9](#_Toc180894438)
9. [Исходный код программы 9](#_Toc180894439)
10. [Контрольный пример 10](#_Toc180894440)
11. [Анализ результатов работы алгоритма и вводных условий 11](#_Toc180894441)
12. [Вывод 15](#_Toc180894442)
13. [Источники 16](#_Toc180894443)

# Цель работы

Целью данной лабораторной работы является исследование и сравнение различных способов кодирования генотипа в рамках генетического алгоритма, а также проверка их эффективности при решении задачи поиска минимума функции. В ходе работы будет разработана программа, реализующая генетический алгоритм, который будет использовать два типа кодирования: вещественное и целочисленное. Программа будет протестирована на заданной тестовой функции для анализа влияния выбранного способа кодирования на скорость нахождения оптимального решения.

# Описание задачи (формализация задачи)

1. Изучение особенностей кодирования в генетических алгоритмах:

Исследовать различные методы кодирования генотипов, включая вещественное и целочисленное кодирование, их преимущества и недостатки.

1. Разработка программы для поиска минимума функции:

Написать программу, реализующую генетический алгоритм, который будет использовать выбранный способ кодирования для решения задачи поиска минимума функции.

1. Тестирование программы на выбранной тестовой функции:

Протестировать программу на одной из тестовых функций из предоставленной таблицы, оценить эффективность каждого из способов кодирования и сравнить результаты.

1. Анализ результатов:

Провести анализ результатов работы алгоритма с различными способами кодирования, выявить влияние кодирования на скорость нахождения минимума функции.

# Теоретическая часть

Генетические алгоритмы

Генетические алгоритмы (ГА) представляют собой оптимизационные методы, основанные на принципах естественного отбора и генетики. Основная идея заключается в том, что популяция возможных решений (хромосом) проходит через циклы мутации, скрещивания и селекции, чтобы постепенно улучшать свое приспособление к задаче. Хромосомы могут быть представлены в различных формах, и выбор метода кодирования генотипа играет ключевую роль в эффективности генетического алгоритма.

Кодирование генотипа

1. Вещественное кодирование:

В этом методе гены представляются в виде вещественных чисел. Данный подход позволяет более точно представлять решения и обеспечивать большую непрерывность пространства поиска. Однако, он может привести к проблеме, связанной с локальными минимумами, если не будет правильно настроен механизм селекции и мутации.

1. Целочисленное кодирование:

При использовании целочисленного кодирования гены представляются целыми числами. Этот метод часто используется, когда задача предполагает дискретные решения. Хотя целочисленное кодирование может быть менее точным по сравнению с вещественным, оно обеспечивает хорошую производительность в задачах, где решения ограничены целыми значениями, таких как комбинаторные задачи.

Оценка эффективности

Эффективность генетического алгоритма определяется его способностью находить оптимальные или близкие к оптимальным решения за разумное время. Это включает в себя:

* Скорость сходимости: время, необходимое для достижения заданного уровня качества решения.
* Качество решения: насколько близко найденное решение к истинному оптимуму.

Применение генетических алгоритмов

Генетические алгоритмы находят применение в различных областях, включая:

* Оптимизацию производственных процессов
* Проектирование и планирование
* Машинное обучение и искусственный интеллект

# Основные шаги программы

GeneticAlgorithm.py

1. Запуск программы:

Инициализация графического интерфейса с настройками и элементами управления.

1. Настройка параметров:

Отображение функции оптимизации: x1^2 + 3\*x2^2 + 2\*x1\*x2.

Пользователь вводит параметры:

* Вероятность мутации (%).
* Количество хромосом.
* Минимальное и максимальное значения генов.
* Количество поколений для выполнения.

1. Выбор типа кодировки:

Пользователь может выбрать между вещественной и целочисленной кодировкой с помощью переключателя.

1. Запуск алгоритма:

Пользователь нажимает кнопку "Рассчитать хромосомы", что инициирует выполнение генетического алгоритма.

1. Инициализация популяции:

Создание начальной популяции с использованием заданных пользователем параметров.

Генерация хромосом в зависимости от выбранного типа кодировки.

1. Оценка приспособленности:

Вычисление значения фитнес-функции для каждой хромосомы в текущей популяции.

1. Селекция родителей:

Применение турнирной селекции для выбора родителей для скрещивания.

1. Скрещивание и мутация:

Выполнение операций скрещивания для создания потомков.

Применение мутации к потомкам на основе заданной вероятности.

1. Создание нового поколения:

Формирование новой популяции из потомков.

Обновление информации о лучших решениях и значениях фитнес-функции.

1. Обновление интерфейса:

Отображение количества прошедших поколений.

Вывод лучших решений (значения x1 и x2) и их фитнес-значения в интерфейсе.

1. Отображение результатов:

В таблице отображаются хромосомы последнего поколения с их индексами, значениями x1, x2 и фитнес-значениями.

# Блок схема программы

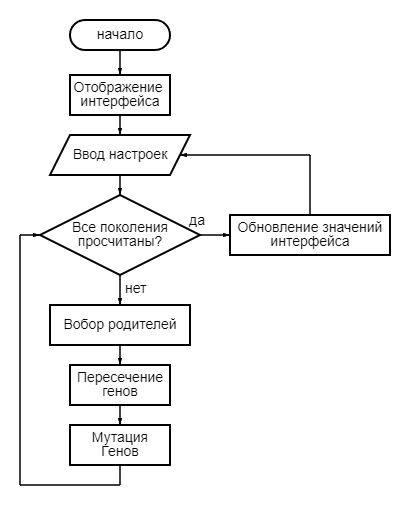


Рис 1. Блок-схема GeneticAlgorithm.py

# Описание программы

Программная реализация написана на языке Python 3.12.6 с использованием следующих библиотек: numpy [[1]](#_Источники), tkinter [[2]](#_Источники) и random [[3]](#_Источники). Программа организована в виде графического интерфейса для решения оптимизационных задач с использованием генетического алгоритма. В процессе разработки программы использовался основной класс GeneticAlgorithmApp включающий 7 методов, каждый из которых имеет чётко определённое назначение:

Таблица 1. GeneticAlgorithm.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| \_\_init\_\_ | Инициализация интерфейса и параметров приложения. | None |
| fitnessFunction | Оценка приспособленности (фитнесс-функция). | float |
| initializePopulation | Инициализация популяции хромосом. | numpy.ndarray |
| tournamentSelection | Селекция родителей с использованием турнира. | numpy.ndarray |
| crossover | Скрещивание двух родительских хромосом. | numpy.ndarray |
| mutateChromosome | Мутация хромосомы с заданной вероятностью. | numpy.ndarray |
| runAlgorithm | Запуск генетического алгоритма и обновление интерфейса. | None |

# Рекомендации пользователя

Программа позволяет запустить генетический алгоритм для оптимизации функции

x\_1^2+3\*x\_2^2+2⋅x\_1⋅x\_2 с использованием графического интерфейса.

Процесс запуска:

1. Запустите программу и задайте параметры генетического алгоритма:

* Вероятность мутации,
* Количество хромосом,
* Минимальное и максимальное значения генов,
* Количество поколений,
* Тип кодировки (вещественная или целочисленная).

1. Нажмите "Рассчитать хромосомы" для начала оптимизации.
2. После выполнения процесса в таблице отобразятся хромосомы последнего поколения и их значения функции приспособленности.
3. В полях "Лучшие решения" отобразится наилучшее найденное решение.

# Рекомендации программиста

Для корректной работы программы убедитесь, что установлены:

* Python версии 3.12.0 или выше;
* Библиотеки: numpy, tkinter, random.

Проверка работы программы:

1. Разместите все файлы программы в одной директории.
2. Выполните запуск программы через команду python GeneticAlgorithm.py.
3. Проверьте работу интерфейса, корректность расчета значений и отображения результатов.

# Исходный код программы

<https://github.com/FasterXaos/Algorithms_and_Data_Structures>

# Контрольный пример

Decryption.py

1. Запуск программы и ввод параметров

Для запуска программы откройте файл, содержащий код GeneticAlgorithmApp. Программа открывает графический интерфейс для настройки и запуска генетического алгоритма для оптимизации функции x\_1^2+3\*x\_2^2+2⋅x\_1⋅x\_2.

1. Ввод начальных параметров

При запуске программы отобразится окно с полями ввода параметров генетического алгоритма (Рис. 2).

* Введите значения для следующих параметров:
* Вероятность мутации (%) — определяет вероятность изменения генов.
* Количество хромосом — размер популяции, участвующей в процессе оптимизации.
* Минимальное и максимальное значения генов — определяют диапазон значений, которые могут принимать гены хромосом.
* Количество поколений — задает, сколько раз алгоритм будет обновлять популяцию, чтобы приблизиться к оптимальному решению.
* Тип кодировки — выберите целочисленную или вещественную кодировку для значений генов.

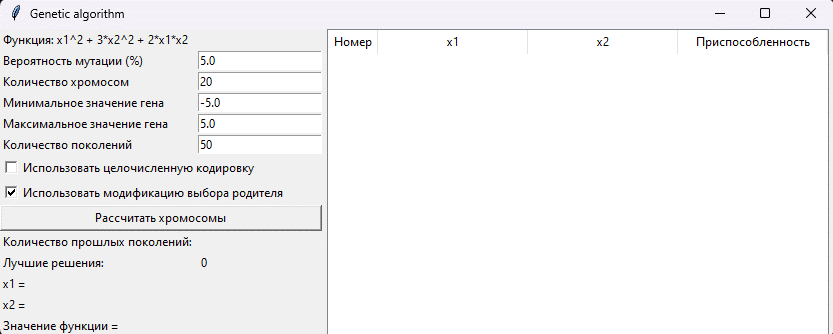


Рис 2. Пример окна программы

1. Запуск алгоритма

После задания всех параметров нажмите кнопку «Рассчитать хромосомы» для запуска алгоритма. Программа выполнит заданное количество итераций, создавая новое поколение на каждом этапе, и отобразит результаты (Рис. 3).

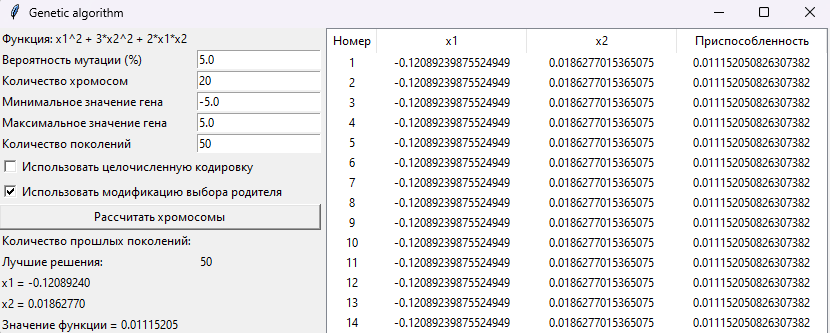


Рис 3. Пример результатов программы

1. Просмотр результатов

После выполнения расчетов на экране отобразятся:

* Количество пройденных поколений.
* Лучшие найденные значения для переменных
* Наилучшее значение функции для этих переменных.

В правой части окна (Рис. 3) будет доступна таблица, в которой отображены хромосомы последнего поколения и их значение фитнес-функции.

# Анализ результатов работы алгоритма и вводных условий

Тесты проводились с 5% вероятностью мутации.

Таблица 2: тесты вещественного генома с разным количеством итераций

|  |  |  |  |
| --- | --- | --- | --- |
| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| 10 | 20 | (0.00502764, -0.14277264) | 0.05974174 |
| 20 | 20 | (0.00502764, -0.12096734) | 0.04270814 |
| 30 | 20 | (0.00502764, -0.12096734) | 0.04270814 |
| 40 | 20 | (0.12216207, -0.09777696) | 0.01971530 |

Таблица 3: тесты вещественного генома с разным размером популяции

|  |  |  |  |
| --- | --- | --- | --- |
| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| 50 | 10 | (-0.32862189, 0.28264199) | 0.16188714 |
| 50 | 20 | (-0.05992936, 0.04166277) | 0.01379253 |
| 50 | 30 | (-0.05319165, 0.01211177) | 0.00198095 |
| 50 | 40 | (0.05928857, -0.02751230) | 0.00252358 |

Таблица 4: тесты бинарного генома с разным количеством итераций

|  |  |  |  |
| --- | --- | --- | --- |
| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| 10 | 50 | (0, 0) | 0 |
| 20 | 50 | (0, 0) | 0 |
| 30 | 50 | (0, 0) | 0 |
| 40 | 50 | (0, 0) | 0 |

Таблица 5: тесты бинарного генома с разным размером популяции

|  |  |  |  |
| --- | --- | --- | --- |
| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| 50 | 10 | (0, -1) | 3 |
| 50 | 20 | (0, 0) | 1 |
| 50 | 30 | (0, 0) | 0 |
| 50 | 40 | (0, 0) | 0 |

Вещественный геном

Таблица 2 показывает, что увеличение количества итераций для небольшой популяции способствует постепенному улучшению решений. Например, за 10 итераций достигается значение функции 0.0597, тогда как за 40 итераций значение снижается до 0.0197. Это указывает на то, что при использовании вещественной кодировки алгоритм лучше приближается к оптимальному решению при увеличении числа итераций.

Таблица 3 демонстрирует зависимость точности решения от размера популяции при фиксированном числе итераций. С увеличением популяции точность решения растет: с 0.1619 для 10 хромосом до 0.0019 для 30 хромосом, что указывает на улучшение поиска при более широком разнообразии особей.

Таким образом, для вещественной кодировки оптимальными условиями являются высокая вариативность в популяции и большее количество итераций, так как это позволяет алгоритму гибко адаптироваться и достигать точных решений.

Бинарный геном

Таблица 4 демонстрирует, что при фиксированном размере популяции увеличении числа итераций алгоритм быстро достигает оптимума, получая значение функции, равное нулю, уже с 10 итераций и далее. Это может свидетельствовать о том, что бинарная кодировка позволяет быстро достигать оптимума, особенно при четко выраженных решениях в дискретном виде.

Таблица 5 показывает зависимость точности от размера популяции. При увеличении популяции от 10 до 40 хромосом решение стабилизируется на оптимуме, так как при достигает значения 0. Этот результат говорит о том, что для бинарной кодировки также важно разнообразие, но даже с меньшими размерами популяции уже можно достичь хороших результатов.

Таким образом, бинарная кодировка является более эффективной, когда требуется быстрое достижение стабильного и четкого решения, особенно в дискретных задачах. Вещественная кодировка, напротив, лучше подходит для задач, где важна высокая точность и требуется более плавное и непрерывное приближение к оптимуму, как в задачах с непрерывными значениями.

Модификация выбора родителей

В процессе разработки и было принято решение о замене традиционного подхода турнира на модифицированный метод с использованием параметра probablyTheBest, который с определенным шансом позволяет выбрать случайного родителя, вместо лучшего. Это решение было обусловлено несколькими факторами:

1. Улучшение селекции: Стандартный турнирный отбор может быть подвержен недостаткам, связанным с преждевременным схлопыванием популяции, когда в силу случайности сильные хромосомы не выбираются на этапе селекции. Использование probablyTheBest увеличивает вероятность выбора наиболее приспособленных хромосом, что позволяет лучше сохранять качество популяции.
2. Гибкость в выборе родителей: Модификация с probablyTheBest дает возможность более гибко управлять процессом селекции. Участие случайных хромосом в выборе родителя при этом все равно сохраняется, что обеспечивает разнообразие и предотвращает зацикленность на узком круге решений.

Таблица 6: тесты генома с разным размером популяции без модификации

|  |  |  |  |
| --- | --- | --- | --- |
| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| 10 | 20 | (-0.65485349, 0.12162072) | 0.31392038 |
| 20 | 20 | (0.03873958, 0.05778593) | 0.01599560 |
| 30 | 20 | (-0.02635879, 0.02368954) | 0.00112951 |
| 40 | 20 | (-0.02635879, 0.02368954) | 0.00112951 |

Таблица 7: тесты генома с разным размером популяции без модификации

|  |  |  |  |
| --- | --- | --- | --- |
| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| 50 | 10 | (0.38852955, -0.10181556) | 0.10293773 |
| 50 | 20 | (0.12280929, -0.07538370) | 0.01361459 |
| 50 | 30 | (0.08803826, -0.00358967) | 0.00715734 |
| 50 | 40 | (0.00274969, 0.00546776) | 0.00012732 |

Таблицы 6 и 7 демонстрируют, как алгоритм ведет себя без модификации родительского отбора при увеличении количества итераций и размера популяции.

В таблице 6 видно, что при фиксированном размере популяции и увеличении числа итераций результат постепенно улучшается: значение функции уменьшается с 0.3139 для 10 итераций до 0.0011 при 30 и 40 итерациях, указывая на устойчивое улучшение решений при увеличении числа итераций.

Таблица 7 показывает, что при фиксированном количестве итераций увеличение популяции также способствует достижению более точного решения. Например, при размере популяции 10 значение функции составляет 0.1029, а при увеличении популяции до 40 достигается значение 0.0001. Эти результаты подтверждают, что, как и в случае с модификацией, алгоритм без модификации также выигрывает от большего числа итераций и размера популяции.

Однако, более низкие начальные значения функции в таблицах с модификацией указывают на то, что вероятность выбора лучших родителей позволяет достичь лучших результатов быстрее, особенно на ранних этапах оптимизации.

# Вывод

В ходе лабораторной работы была разработана и исследована эффективность генетического алгоритма с двумя различными типами кодировки: вещественной и бинарной. Проведенные тесты показали, что алгоритм ведет себя по-разному в зависимости от параметров популяции, числа итераций и используемой кодировки.

# Источники

1. numpy // Numpy URL: <https://numpy.org/> (дата обращения: 27.10.2024).
2. tkinter — Python interface to Tcl/Tk // Tkinter URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 27.10.2024).
3. random // Random URL: <https://docs.python.org/3/library/random.html> (дата обращения: 27.10.2024).