**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЁТ**

**По лабораторной работе №4**

**По дисциплине “Алгоритмы и структуры данных”**

**На тему “Исследование генетического алгоритма. Изучение различных кодировок генотипа”**

**Вариант №15**

**Студент гр. 23Б16-пу**

**Пушкарев Н.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

[**Цель работы 3**](#_ixkrmexdlmx1)

[**Описание задачи 3**](#_2d07uirr35d9)

[**Теоретическая часть 3**](#_bc8ka7j09wvg)

[**Описание программы 5**](#_3znysh7)

[**Основные шаги программы 7**](#_tyjcwt)

[**Блок схема программы 7**](#_3dy6vkm)

[**Рекомендации пользователя 9**](#_jmd7r1jlo5hs)

[**Рекомендации программиста 10**](#_1t3h5sf)

[**Контрольный пример 10**](#_4d34og8)

[**Исследование 15**](#_2s8eyo1)

[**Вывод 17**](#_17dp8vu)

# Цель работы

Исследование особенностей генетических алгоритмов для решения задач глобальной оптимизации. Изучение работы любой модификации выбора родителя генетического алгоритма.

# Описание задачи

* Изучить особенности кодирования генетических алгоритмов
* Написать программу поиска минимума функции, выбрав вариант тестовой функции из таблицы
  + Реализовать модификацию в выборе родителя
  + Реализовать выбор использования модификации
* Протестировать программу на выбранной тестовой функции
  + Протестировать с модификацией и без неё

# Теоретическая часть

Генетический алгоритм (ГА) — это метод оптимизации, основанный на принципах эволюции, моделирующий естественные биологические процессы для поиска наилучших решений. Этот алгоритм, разработанный в области искусственного интеллекта, успешно применяется в различных задачах оптимизации и поиска.

Основные принципы генетического алгоритма

1. Популяция и генотипы

- Популяция: Это начальный набор возможных решений задачи, называемых "особями". Каждая особь соответствует одному из потенциальных решений задачи и принадлежит к популяции, которая будет улучшаться с течением времени.

- Генотип: Представление решения задачи, закодированное в виде набора генов (например, в виде строк или векторов). Эти гены определяют основные характеристики решения и могут изменяться в процессе эволюции.

2. Приспособленность

- Это мера того, насколько хорошо особь решает поставленную задачу. Чем более приспособлено решение, тем выше его оценка. Высокая приспособленность повышает вероятность использования особи в процессе создания нового поколения.

3. Селекция

- Определяет, какие особи будут участвовать в формировании следующего поколения. Обычно особи с высокой приспособленностью имеют больше шансов быть выбранными для передачи своих характеристик потомству.

Турнирная селекция — метод выбора родительских особей, основанный на проведении "турниров" среди случайно выбранных участников. Для каждого турнира случайным образом выбирается несколько особей (обычно 2 или 3), и в результате в родительскую пару попадает та, которая имеет наибольшую приспособленность среди участников турнира.

4. Скрещивание (Кроссовер)

- Процесс обмена генетической информации между двумя выбранными особями для создания потомства. Этот процесс моделирует естественную рекомбинацию ДНК и служит для создания новых решений, которые могут унаследовать черты от обоих родителей.

5. Мутация

- Небольшие случайные изменения генотипа особи. Мутация поддерживает разнообразие внутри популяции и предотвращает преждевременное застревание в локальных оптимумах, способствуя исследованию большего пространства решений.

6. Эволюция

- После создания нового поколения путем селекции, скрещивания и мутации процесс повторяется. С каждым новым поколением алгоритм стремится улучшить средний уровень приспособленности популяции, приближая её к оптимальному решению.

Применение в оптимизации

Генетические алгоритмы находят широкое применение в задачах оптимизации, среди которых:

- Функциональная оптимизация: Используется для поиска глобального экстремума функций, где важно учитывать сложные ограничения и нелинейные зависимости.

- Комбинаторная оптимизация: Применяется для решения задач с комбинаторными структурами, таких как задача о рюкзаке, задачи маршрутизации и планирования.

- Нейронные сети: Генетические алгоритмы могут оптимизировать параметры нейронных сетей, обучая их более эффективно и настраивая веса для улучшения производительности.

# Описание программы

исполняемый файл: main.py

Таблица 1. Описание функций и классов

| Название класса | Описание |
| --- | --- |
| class Constants | Класс Constants хранит параметры для работы генетического алгоритма, такие как размер популяции, вероятность мутации и функция для оптимизации. |
| class Func | Предназначен для хранения функции в виде строки и её вычисления с заданными значениями переменных x и y  Метод \_\_init\_\_ сохраняет функцию, представленную строкой, в атрибуте func.  Метод value выполняет вычисление функции, используя eval, при этом значения переменных x и y могут быть заданы при вызове метода, а если они не указаны, то по умолчанию будут равны 0. |
| class GUI | Класс для вывода графического интерфе са на tkinter. |
| class Individual | Класс Individual представляет одного индивида с характеристиками x и y, а также значением приспособленности.  Метод clone создает копию индивида, включая значение его текущей приспособленности.  Метод calculation\_fitness вычисляет значение целевой функции с текущими значениями x и y индивида.  Метод mutation вносит случайные изменения в характеристики x и y, имитируя мутацию.  Метод \_\_repr\_\_ возвращает строку с текущими характеристиками и приспособленностью для удобства вывода. |
| class Population | Класс Population — это наследник от списка, который используется для управления популяцией индивидов.  Метод crossing принимает двух родителей, создает нового индивида с усредненными характеристиками x и y родителей и рассчитывает его приспособленность с помощью переданного объекта func. |
| def Tournament | Функция Tournament реализует турнирный отбор для формирования новой популяции потомков из исходной популяции.  Аргумент mod\_selection определяет, использовать ли модифицированный отбор с порогом среднего значения приспособленности:  Включен: выбираются только индивиды с приспособленностью меньше или равной средней по популяции.  Отключен: выбирается группа из 3 случайных индивидов, из которых отбираются два лучших. |
| def Add\_in\_pop | Функция add\_in\_pop добавляет клонированных индивидов из текущей популяции в общий список pop\_ind, который может хранить индивидов всех поколений.  Каждый индивид из популяции клонируется (создается новый экземпляр) перед добавлением в список, чтобы избежать изменения оригинальных объектов. |
| def Individual\_creator | Функция Individual\_creator инициализирует нового индивида с случайными значениями координат x и y в пределах заданного диапазона.  Координаты индивида генерируются случайным образом, затем для него рассчитывается приспособленность с использованием функции func. |
| def Population\_creator | Функция Population\_creator инициализирует начальную популяцию заданного размера. |
| def main | Функция main реализует основной цикл работы генетического алгоритма, включающий этапы отбора, кроссинговера и мутации. Алгоритм повторяется для заданного числа поколений |

# 

# Основные шаги программы

1. Запуск программы (main.py)
2. Выбор нужных параметров
3. Сохранение нужных параметров
4. Запуск работы алгоритма
5. Вывод результатов последних n-поколений, выбранных пользователем

# Блок схема программы

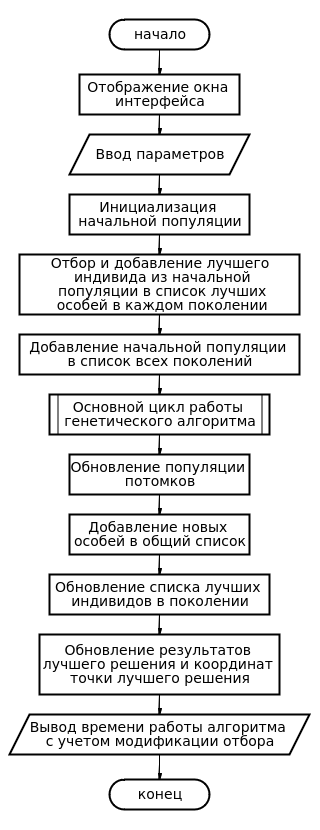


Рис1. Блок-схема программы

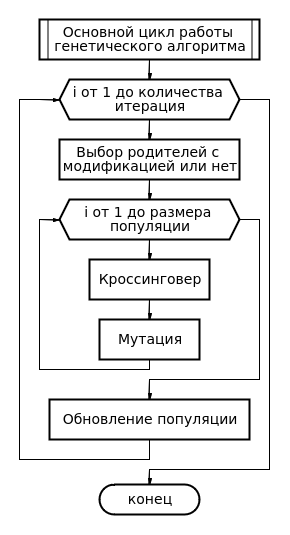


Рис2. Блок-схема подпрограммы

# Рекомендации пользователя

* Для запуска программы, откройте файл main.py в pycharm и нажмите Shift+F10
* Выберите нужные параметры
* Сохраните нужные параметров (кнопка “сохранить” в верхней строке интерфейса)
* Запустите работу алгоритма (кнопка “запустить” в верхней строке интерфейса)
* После окончания работы посмотрите время работы программы в терминале
* При необходимости проанализируйте результаты последних n-поколений (кнопка “показать последние поколения” в верхней строке интерфейса)

# Рекомендации программиста

* Используйте python версии не мене 3.11
* Используйте tkinter и matplotlib последних версий

# Контрольный пример

Запускаем файл main.py и видим перед собой окно с параметрами по умолчанию.

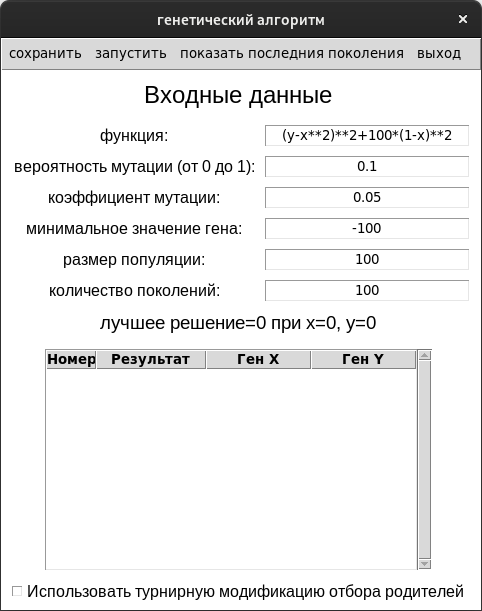
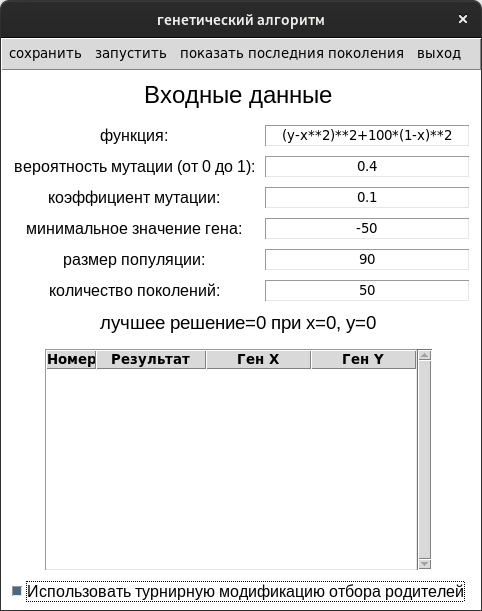


Рис3. Пример окна интерфейса

Далее меняем параметры и нажимаем “сохранить”.

Рис4. Пример окна интерфейса с измененными параметрами

Нажимаем “запустить” и получаем решение и время работы (в терминале).

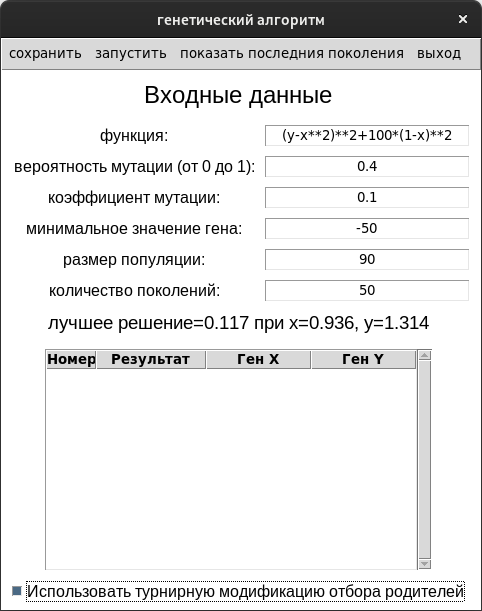
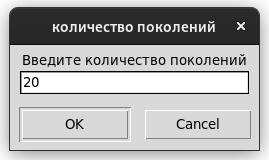
Рис5. Пример решения

Рис6. Пример вывода времени работы

Нажимаем “показать последние поколения” и вводим нужное число.

Рис7. Пример окна ввода количества поколений для отображения

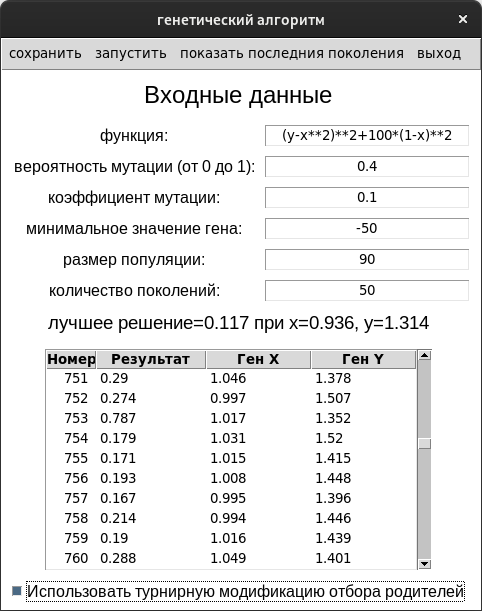


Рис8. Пример вывода поколений

# Исследование

Тесты проводились с 10% вероятностью мутации.

Таблица 2: тесты с разным количеством итераций без турнирной модификации отбора

| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| --- | --- | --- | --- |
| 60 | 50 | ( 0.982 , 0.827 ) | 0.015 |
| 70 | 50 | ( 0.997 , 1.235 ) | 0.059 |
| 80 | 50 | ( 0.997 , 0.481 ) | 0.264 |
| 90 | 50 | ( 1.0 , 1.0 ) | 0.0 |
| 100 | 50 | ( 1.0 , 0.998 ) | 0.0 |

Таблица 3: тесты с разным количеством итераций с турнирной модификации отбора

| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| --- | --- | --- | --- |
| 60 | 50 | ( 0.981 , 0.968 ) | 0.001 |
| 70 | 50 | ( 1.026 , 2.01 ) | 0.984 |
| 80 | 50 | ( 1.011 , 1.292 ) | 0.084 |
| 90 | 50 | ( 1.0 , 1.056 ) | 0.003 |
| 100 | 50 | ( 1.0 , 0.999 ) | 0.0 |

Таблица 4: тесты с разным размером популяции без турнирной модификации отбора

| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| --- | --- | --- | --- |
| 50 | 60 | (1.007 , 1.725 ) | 0.509 |
| 50 | 70 | ( 0.997 , 0.948 ) | 0.003 |
| 50 | 80 | ( 1.0 , 1.0 ) | 0.0 |
| 50 | 90 | ( 1.0 , 1.0 ) | 0.0 |
| 50 | 100 | ( 1.0 , 1.194 ) | 0.025 |

Таблица 5: тесты с разным размером популяции c турнирной модификации отбора

| Количество итераций | Размер популяции | Наилучшее решение | Значение в лучшей точке |
| --- | --- | --- | --- |
| 50 | 60 | ( 1.007 , 1.172 ) | 0.03 |
| 50 | 70 | ( 1.002 , 1.053 ) | 0.003 |
| 50 | 80 | ( 0.99 , 0.347 ) | 0.411 |
| 50 | 90 | ( 1.0 , 1.0 ) | 0.0 |
| 50 | 100 | ( 1.016 , 1.574 ) | 0.319 |

Таблица 2 демонстрирует, что при фиксированном размере популяции при увеличении числа итераций алгоритм достигает оптимума, получая значение функции, равное нулю, в нужной точке (1, 1) с 90 количества поколений и далее.

Таблица 3 демонстрирует, что при фиксированном размере популяции при увеличении числа итераций с использованием турнирной модификации алгоритм достигает оптимума быстрее, чем без модификации, хотя оказывается подвержен колебаниям в результатах. Можно сказать, что мы получаем достаточно близкое решение с 80 количества поколений, в точке ( 1.011 , 1.292 ) и значением 0.084 и далее.

Таблица 4 демонстрирует зависимость точности решения от размера популяции при фиксированном числе итераций без модификации. С увеличением популяции точность решения растет: с 0.509 для 60 хромосом до точного ответа 0.0 для уже 80 хромосом, что указывает на улучшение поиска в сравнении с вариантом, когда меняется только количество итераций.

Таблица 5 демонстрирует зависимость точности решения от размера популяции при фиксированном числе итераций с турнирной модификацией селекции. Для каждого количества хромосом результат оказывается точнее, чем для того же теста без модификации. Можно сказать, что алгоритм приходит к нужному ответу уже на 70 хромосомах в популяции.

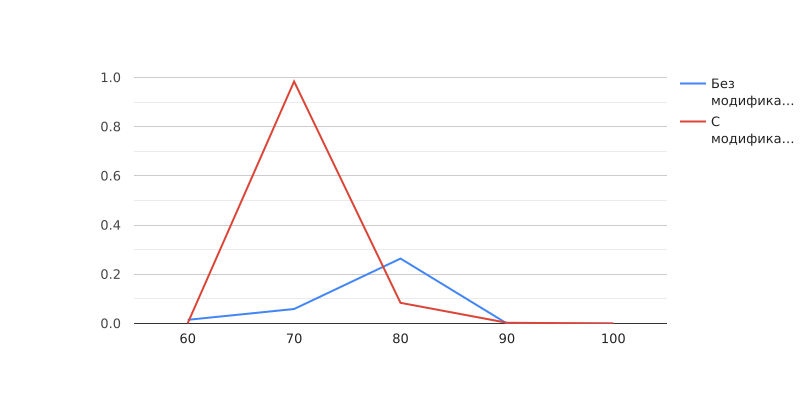
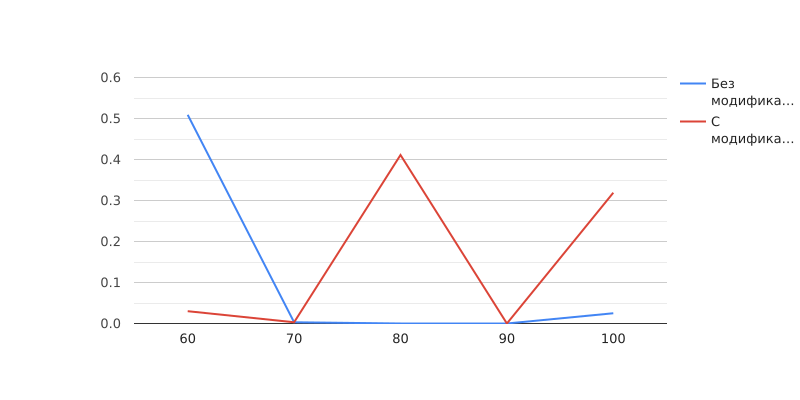
Рис9. График зависимости отклонения от ответа при разном кол-ве поколений

Рис10. График зависимости отклонения от ответа при разном размере популяции

# Вывод

В ходе исследования был рассмотрен генетический алгоритм. Он был использован для вычисления минимума функции, при этом с разными параметрами работы, в том числе с использованием турнирной селекции. Было выяснено, что увеличение размера популяции и числа итераций положительно влияет на качество решения, но достаточно сильно увеличивает время работы. Также было экспериментально доказано, что использование модификации турнирной селекции значительно увеличивает точность решения и время работы генетического алгоритма.