Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Ульяновский государственный Технический университет

Кафедра «Вычислительная техника»

Дисциплина «Системы искусственного интеллекта»

**Лабораторная работа №1**

**«Генетические алгоритмы»**

**Выполнил**:

студент гр. ИВТАСбд-41

Борков И.Е.

**Проверил работу:**

Хайруллин И. Д.

Ульяновск,

2025

**Общее задание**

1. Необходимо разработать программу на языке python, реализующую генетический алгоритм по предложенному вариантом заданию.
2. Провести эксперименты по разным способам скрещивания (не менее 3-х), разным способам мутирования (не менее трех). Результат отобразить в виде графиков.
3. Моделирование данных производить на основе максимально правдоподобных данных. Т.е. если рассматривается задача, в которой есть калорийность продуктов, то должны использоваться данные о реальных продуктах с реальной калорийностью.
4. Предоставить отчет о проделанной работе.

**Задача. Вариант №7:**

На языке Python разработайте скрипт, который с помощью генетического алгоритма и полного перебора решает следующую задачу. Дано n пунктов производства продуктов и k городов, которые в них нуждаются. Каждый город может потребить x продуктов, а каждый пункт произвести y продуктов. Необходимо получить оптимальный маршрут, так, чтобы все города получили нужный им объем продуктов без сильного его превышения, а транспортные расходы были минимальными.

**Теоретическая подготовка**

*Что такое генетический алгоритм?*

Генетический алгоритм (ГА) — это метод поиска оптимальных решений, вдохновлённый принципами естественного отбора и эволюции. Он не перебирает все варианты, а "эволюционирует" набор решений, постепенно улучшая их на основе их "приспособленности" к условиям задачи.

В моем коде это выглядит так:

1. Создание популяции: Алгоритм начинает с набора случайных решений, которые в коде представлены как матрицы потоков N x K. Это и есть начальная популяция особей.
2. Оценка приспособленности: Каждое решение оценивается функцией calculate\_fitness(). Эта функция определяет, насколько "хороша" данная матрица, вычисляя общую стоимость и штрафы. Чем ниже стоимость, тем выше приспособленность. Функция Fitness=1/(1+Стоимость + Штрафы) преобразует задачу минимизации (расходов) в задачу максимизации (приспособленности).
3. Эволюция поколений: Алгоритм повторяет следующие шаги, чтобы создать новое, более совершенное поколение:
   * Отбор: Лучшие решения (с самым высоким Fitness) выбираются в качестве родителей. В моем коде для этого используется турнирный отбор.
   * Скрещивание: Родители обмениваются частями своих матриц, создавая новые решения — потомков. Я использую одноточечное, двухточечное и равномерное скрещивание.
   * Мутация: В потомков вносятся небольшие случайные изменения. Это помогает алгоритму избежать застревания в локальных оптимумах и находить новые, потенциально лучшие решения. У меня реализовано три типа мутации: одноточечная, сброса и перераспределения.
4. Конечный результат: После заданного количества поколений (GENERATIONS), алгоритм выдаёт самое приспособленное решение, найденное за всё время.

ГА подходит для задачи, потому что она относится к классу NP-трудных задач, где полный перебор всех возможных вариантов занимает нереально много времени. ГА находит оптимальное решение за разумное время, что и является целью лабораторной работы.

*Скрещивание (Crossover)*

Скрещивание — это процесс, при котором два родительских решения (матрицы потоков) обмениваются своими частями для создания новых решений (потомков). Его цель — объединить лучшие характеристики от разных родителей. Например, если у одного родителя низкая стоимость доставки, а у другого — низкие штрафы, скрещивание может создать потомка с обоими этими преимуществами.

Я использовал три типа скрещивания:

* single\_point\_crossover: Матрицы разрезаются по одной строке, и части обмениваются.
* two\_point\_crossover: Матрицы разрезаются по двум строкам, и средняя часть обменивается.
* uniform\_crossover: Гены (отдельные ячейки матрицы) обмениваются между родителями со случайной вероятностью.

*Мутация (Mutation)*

Мутация — это случайное, небольшое изменение в генах (ячейках матрицы) нового решения. Его цель — внести генетическое разнообразие в популяцию, что помогает алгоритму находить новые, неожиданные решения и избегать застревания в локальных оптимумах.

Я использовал три типа мутации:

* single\_point\_mutation: Случайное значение в одной ячейке матрицы немного изменяется (увеличивается или уменьшается).
* reset\_mutation: Значение в одной случайной ячейке полностью сбрасывается и заменяется на новое случайное.
* redistribution\_mutation: Потоки из одного случайного пункта производства полностью сбрасываются и заново распределяются между городами.

*Типы скрещивания*

Скрещивание, или кроссовер, объединяет генетический материал от двух родителей для создания нового потомка. Это основной двигатель эволюции в генетическом алгоритме.

1. Одноточечное скрещивание (single\_point\_crossover)
   * Как работает: Матрицы-хромосомы "разрезаются" по одной случайной строке. Первая часть одного родителя объединяется со второй частью другого, чтобы создать первого потомка, и наоборот.
   * Преимущества: Простота и эффективность. Оно сохраняет большие "блоки" информации, унаследованные от родителей, что часто ведёт к хорошим результатам.
   * Недостатки: Может быть неэффективным, если лучшие гены (ячейки) распределены по всей матрице, а не сосредоточены в одной области.
2. Двухточечное скрещивание (two\_point\_crossover)
   * Как работает: Матрицы разрезаются в двух случайных точках. Средняя часть одной матрицы меняется местами со средней частью другой.
   * Преимущества: Позволяет обмениваться не одним, а двумя блоками информации. Это делает поиск более разнообразным, так как у потомков будет больше комбинаций генов.
   * Недостатки: Всё ещё ограничено блочной структурой.
3. Равномерное скрещивание (uniform\_crossover)
   * Как работает: Для каждой ячейки матрицы случайным образом выбирается, от какого родителя взять ген. Это как бросить монетку для каждой ячейки: "орёл" — берём от первого родителя, "решка" — от второго.
   * Преимущества: Самый сильный способ смешивания генов. Он позволяет комбинировать информацию на детальном уровне, а не блоками. Это может помочь выбраться из локальных оптимумов.
   * Недостатки: Может разрушать полезные комбинации генов, которые находились рядом.

*Типы мутаций*

Мутация вносит случайные изменения, которые поддерживают генетическое разнообразие и помогают находить решения, недостижимые с помощью только скрещивания.

1. Одноточечная мутация (single\_point\_mutation)
   * Как работает: Выбирается одна случайная ячейка в матрице, и её значение немного изменяется. Например, к нему добавляется или вычитается небольшое случайное число.
   * Преимущества: Малое, "мягкое" изменение, которое не разрушает большую часть решения. Это помогает алгоритму тонко настраивать параметры.
   * Недостатки: Может быть слишком "слабой", чтобы вытащить алгоритм из глубокого локального оптимума.
2. Мутация сброса (reset\_mutation)
   * Как работает: Случайная ячейка в матрице полностью заменяется на новое, случайное значение.
   * Преимущества: Вносит более резкие изменения, чем одноточечная мутация. Это может помочь "перепрыгнуть" через небольшой пик и найти более выгодную область поиска.
   * Недостатки: Есть риск, что случайное значение окажется очень плохим и сильно ухудшит решение.
3. Мутация перераспределения (redistribution\_mutation)
   * Как работает: Выбирается один пункт производства (одна строка в матрице). Все его потоки сбрасываются, а затем заново распределяются между городами, но так, чтобы общая сумма не превышала производственную мощность этого пункта.
   * Преимущества: Это "умная" мутация. Она вносит значительное изменение, но при этом сохраняет одно из ключевых ограничений задачи (общую мощность производства), что делает её более эффективной, чем случайные изменения.
   * Недостатки: Она может быть более вычислительно затратной, чем простые мутации, так как ей нужно пересчитывать распределение.

*Что такое отбор*

Отбор — это процесс выбора "родителей" из текущей популяции, которые будут производить потомство для следующего поколения. Чем выше приспособленность особи (чем ниже стоимость её решения), тем больше шансов у неё быть выбранной.

В моем коде используется турнирный отбор (selection):

* Как работает: Случайным образом выбирается небольшая группа особей, и из них выбирается самая приспособленная. Этот процесс повторяется, пока не будет набрано достаточно родителей для скрещивания.
* Почему это эффективно: Турнирный отбор прост в реализации, но при этом достаточно силён. Он не только выбирает "абсолютных" лидеров, но и даёт шанс менее приспособленным особям, что помогает поддерживать генетическое разнообразие в популяции.

**Описание набора данных**

В основе набора данных лежат два ключевых элемента: пункты производства и города. Производственных пунктов 5 (N=5), каждый из которых имеет свои координаты (x, y) и объём производства (supply), который не может превысить. Городов 8 (K=8), и у каждого из них есть свои координаты и чётко определённая потребность в продуктах (demand), которую нужно удовлетворить. Координаты позволяют вычислить расстояние между любым пунктом и городом, что необходимо для расчёта транспортных расходов.

Помимо основных элементов, набор данных включает в себя параметры стоимости и штрафов. Это ключевые компоненты, которые помогают генетическому алгоритму найти оптимальное решение. price\_per\_unit\_distance определяет, сколько стоит доставить одну единицу продукта на определённое расстояние. Штрафы, в свою очередь, "наказывают" решения, которые не соответствуют условиям задачи. Штраф за недостаток продуктов (penalty\_per\_shortage\_unit) установлен выше, чем за излишек (penalty\_per\_excess\_unit), что вынуждает алгоритм в первую очередь обеспечивать полное удовлетворение спроса, а затем уже оптимизировать расходы.

**Описание реализации**

Мой код представляет собой комплексное решение для задачи оптимизации маршрутов доставки, использующее два разных подхода: генетический алгоритм (ГА) и полный перебор. Главная цель — найти оптимальную матрицу поставок, чтобы минимизировать общую стоимость, которая включает расходы на доставку и штрафы за несоответствие спросу.

Программа разделена на несколько логических блоков:

**1. Инициализация и данные**

Сначала определяются все исходные данные и параметры, как для задачи, так и для самого алгоритма. Это включает количество пунктов производства (N) и городов (K), а также все финансовые показатели (стоимость доставки, штрафы). Затем генерируются случайные координаты, объемы производства и потребности, а на их основе рассчитываете матрицу расстояний. Эти данные составляют основу, с которой работает весь алгоритм.

**2. Функция приспособленности**

Функция calculate\_fitness является "сердцем" генетического алгоритма. Она берёт любое возможное решение (матрицу потоков) и оценивает его. Расчёт включает в себя:

* Общую стоимость доставки: Сумма всех перевезённых продуктов, умноженная на расстояние и стоимость.
* Штрафы: Наказания за недопоставку, перепоставку и превышение производственных мощностей.

В итоге, эта функция превращает общую стоимость (которую нужно минимизировать) в значение приспособленности (которое нужно максимизировать) по формуле Fitness=1/(1+Общая стоимость + Штрафы) ​.

**3. Операторы генетического алгоритма**

Этот блок содержит функции, которые имитируют эволюционные процессы:

* create\_initial\_population: Создает первое поколение случайных решений.
* selection: Отбирает лучшие решения для создания следующего поколения, используя турнирный отбор.
* Скрещивание (crossover): Функции single\_point\_crossover, two\_point\_crossover и uniform\_crossover отвечают за обмен генами между родителями, создавая новых потомков.
* Мутация (mutation): Функции single\_point\_mutation, reset\_mutation и redistribution\_mutation вносят случайные изменения в потомков, чтобы обеспечить разнообразие.

**4. Основной цикл ГА**

Функция genetic\_algorithm объединяет все операторы. Она запускает цикл на заданное количество поколений (GENERATIONS), в каждом из которых она:

1. Оценивает приспособленность всех особей в популяции.
2. Выбирает родителей.
3. Создаёт новых потомков с помощью скрещивания и мутации.
4. Обновляет популяцию, заменяя старые решения новыми.

В конце цикла функция возвращает лучшее найденное решение.

**5. Полный перебор**

Функция brute\_force\_solver\_simplified демонстрирует, почему генетический алгоритм необходим. Она решает ту же задачу, но для сильно упрощённых данных (маленькой матрицы). Она перебирает каждую возможную комбинацию потоков, чтобы найти абсолютно оптимальное решение. Это наглядно показывает, что для реальных, более крупных задач такой подход невозможен из-за экспоненциального роста времени вычисления.

**6. Запуск и анализ результатов**

В последнем блоке код запускает все эксперименты, измеряет время выполнения и выводит результаты. Он сравнивает эффективность разных типов мутаций и показывает разницу во времени выполнения между ГА и полным перебором. Финальный график визуально отображает, как меняется приспособленность популяции с каждым поколением.

**Общее задание**

1. **Какова структура хромосомы для вашей задачи?**

Хромосома в моей задаче представляет собой матрицу потоков размером N×K.

* N — это количество пунктов производства (в моем случае, 5).
* K — это количество городов (в моем случае, 8).

Каждая ячейка этой матрицы, individual[i, j], содержит целое число — количество продукта, которое отправляется из i-го пункта производства в j-й город.

Таким образом, вся матрица целиком является одним решением задачи — "особью" или "хромосомой" в терминологии генетического алгоритма. Она полностью описывает, как распределяется весь производимый продукт.

1. **Какую функцию приспособленности вы выбрали и почему?**

Я выбрал функцию приспособленности, которая преобразует задачу минимизации общей стоимости в задачу максимизации показателя fitness. Функция определяется так:

### Почему именно эта функция

* **Преобразование задачи:** Генетические алгоритмы стремятся максимизировать приспособленность. Моя задача — минимизировать стоимость. Эта формула позволяет мне превратить одну задачу в другую: чем меньше стоимость в знаменателе, тем больше приспособленность в целом.
* **Имитация "непригодных" решений:** Если какое-то решение нарушает основные правила (например, отправляет больше продуктов, чем может произвести пункт), функция возвращает 0. Это немедленно исключает такое решение из дальнейшей эволюции, поскольку оно будет считаться "непригодным".
* **Чёткая иерархия:** Формула с обратной пропорциональностью позволяет сравнивать любые два решения: если одно решение имеет меньшую стоимость, оно будет иметь более высокую приспособленность. Это позволяет алгоритму последовательно отбирать лучшие решения на каждом поколении.

1. **Что такое кроссовер и мутация?**

**Кроссовер** — это процесс, при котором две родительские особи (матрицы потоков) обмениваются частями своих "генов" для создания новых потомков. Его главная цель — объединять лучшие качества от разных решений. Например, если один родитель имеет низкую стоимость доставки, а другой — низкие штрафы, кроссовер может создать потомка, который наследует оба эти преимущества.

У меня реализовано три типа скрещивания:

* **Одноточечное:** Матрица делится на две части в одной случайной точке (строке), и эти части меняются между родителями.
* **Двухточечное:** Матрица делится на три части в двух случайных точках, и средние части меняются местами.
* **Равномерное:** Для каждой ячейки матрицы случайным образом определяется, от какого родителя она унаследует своё значение.

**Мутация** — это процесс случайного, небольшого изменения в генах (ячейках матрицы) нового решения. Её цель — внести генетическое разнообразие в популяцию. Это помогает алгоритму избежать застревания в локальных оптимумах и находить новые, потенциально лучшие решения, которые не могли быть созданы только с помощью скрещивания.

У меня реализовано три типа мутации:

* **Одноточечная:** Выбирается одна случайная ячейка, и её значение слегка изменяется.
* **Сброса:** Выбранная ячейка получает совершенно новое случайное значение.
* **Перераспределения:** Для одного пункта производства все его потоки сбрасываются и заново случайным образом распределяются между городами, но так, чтобы не превысить лимит производства.