МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»**

Кафедра Математического и компьютерного моделирования

Влияние выбора генов на результат работы генетических алгоритмов в «задачи коммивояжера»

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студентки 2 курса 247 группы

направление 09.04.03 — Прикладная информатика

механико-математического факультета

Беляев Эдуард Евгеньевич

Научный руководитель
зав. каф., д.ф.-м.н., доцент Ю.А. Блинков

Зав. кафедрой
зав. каф., д.ф.-м.н., доцент Ю.А. Блинков

Саратов 2025

Введение. «Задача коммивояжера» – это классическая задача комбинаторной оптимизации, которая заключается в поиске оптимального маршрута для коммивояжера, который должен посетить все города только один раз и вернуться в исходный город. Эта задача имеет множество применений в различных областях, таких как логистика, планирование маршрутов и другие.

Генетические алгоритмы – это алгоритмы, основанные на принципах естественного отбора и эволюции. Они используются для решения задач оптимизации, включая задачу коммивояжера.

Сравнение решений задачи коммивояжера генетическими алгоритмами с разными наборами генов позволяет оценить эффективность и точность алгоритма. В зависимости от выбранного набора генов, алгоритм может давать разные результаты. Например, если набор генов содержит только расстояния между городами, то алгоритм будет искать оптимальный маршрут, учитывая только расстояния. Если же набор генов содержит также информацию о времени посещения городов, то алгоритм будет учитывать и время при поиске оптимального решения.

Сравнение решений задачи коммивояжера генетическими алгоритмами с разными наборами генов позволяет выбрать наиболее эффективный и точный алгоритм для конкретной задачи. Это может быть полезно при разработке программного обеспечения для оптимизации процессов в различных областях.

Цель магистерской работы состоит в исследование эффективности различных методов кодирования решений в генетических алгоритмах для задачи коммивояжера (TSP) и выявление оптимального подхода в зависимости от размера задачи и структуры графа.

Основная часть. Задача коммивояжёра — одна из самых известных и изучаемых проблем в теории оптимизации и компьютерных науках. Её история уходит корнями в **XIX век**, но формальное определение и систематическое изучение началось в **XX веке**.

Генетические алгоритмы (ГА) — это эвристические методы оптимизации и поиска, основанные на принципах естественного отбора и генетики. Они применяются для решения сложных задач, где традиционные методы (например, градиентный спуск или полный перебор) неэффективны.

ГА работают с популяцией возможных решений, которые эволюционируют через операции**селекции**,**скрещивания** и**мутации**, имитируя биологическую эволюцию.

ГА основаны на идеях Чарльза Дарвина:

- **Наследственность** — лучшие решения передают свои "гены"потомкам.
- **Изменчивость** — случайные мутации и рекомбинация создают разнообразие.
- **Естественный отбор** — выживают наиболее приспособленные особи.

Основные компоненты генетического алгоритма. **Представление решения (хромосома).**

Каждое решение кодируется в виде**хромосомы** — строки генов (обычно бинарной, но возможны и другие кодировки):

- **Бинарное кодирование**: ‘101010‘ (для дискретных задач).
- **Вещественное кодирование**: ‘[3.14, 2.71, 1.41]‘ (для непрерывных задач).
- **Порядковое кодирование**: ‘[A, D, C, B]‘ (для задач комбинаторной оптимизации, например, TSP).

Функция приспособленности (fitness function).

Функция $f(x)$, которая оценивает качество решения. Чем выше значение, тем лучше особь.

Пример:

- Для задачи минимизации $f(x) = 1/(1 + x^2)$.
- Для задачи коммивояжера $f(x) = 1/$(длина маршрута).

Генетические операторы.

Селекция (отбор родителей).

Выбирает особи для скрещивания на основе их приспособленности. Популярные методы:

- **Рулетка (Fitness-proportionate selection)** — вероятность выбора пропорциональна $f(x)$.
- **Турнирный отбор** — случайно выбирается k особей, лучшая проходит дальше.

- **Ранговая селекция** — особи сортируются по $f(x)$, вероятность зависит от ранга.

Скрещивание (кроссовер, recombination).

Создает потомков, комбинируя гены родителей. Основные методы:

- **Одноточечный кроссовер**:

```
Родитель 1: 101|010 → Потомок 1: 101110
Родитель 2: 110|110 → Потомок 2: 110010
```

- **Двухточечный кроссовер**:

```
Родитель 1: 10|10|10 → Потомок 1: 100110
Родитель 2: 11|01|10 → Потомок 2: 111010
```

- **Однородный кроссовер (Uniform crossover)**: каждый ген берется случайно от одного из родителей.

Мутация.

Вносит случайные изменения в гены для поддержания разнообразия:

- **Битовая мутация (для бинарных строк)**: $1010 \mapsto 1011$ (инвертируется случайный бит).
- **Вещественная мутация (для чисел)**: $x \mapsto x + \Delta$, где Δ — малое случайное число.
- **Swap-мутация (для перестановок)**: $[A, B, C, D] \mapsto [A, D, C, B]$ (меняются местами два элемента).

Замена поколений (survivor selection).

Определяет, какие особи перейдут в следующее поколение:

- **Полная замена (Generational GA)** — все родители заменяются потомками.
- **Элитизм** — лучшие особи сохраняются.
- **Steady-state GA** — заменяется только часть популяции.

Псевдокод генетического алгоритма.

```
1. Инициализация: создать начальную
популяцию P из N случайных особей.
2. Оценка: вычислить fitness для каждой особи.
3. Пока не выполнено условие остановки
```

```
(макс. поколений / достаточный fitness):
   3.1. Селекция: выбрать родителей из P.
   3.2. Скрещивание: создать потомков.
   3.3. Мутация: применить мутацию к потомкам.
   3.4. Оценка потомков.
   3.5. Замена: сформировать новую популяцию P'.
   3.6. P = P'.
4. Вернуть лучшую особь.
```

Параметры генетического алгоритма:

- **Размер популяции (N)** — обычно 50–500.
- **Вероятность кроссовера (Pc)** — 0.6–0.9.
- **Вероятность мутации (Pm)** — 0.001–0.1.
- **Критерий остановки**:
  - Максимальное число поколений.
  - Достижение целевого значения fitness.
  - Отсутствие улучшений за несколько поколений.

Преимущества и недостатки.

Преимущества:

- Работает с дискретными, непрерывными и комбинаторными задачами.
- Не требует информации о градиенте (подходит для недифференцируемых функций).
- Устойчив к локальным оптимумам (благодаря мутациям). v Параллелизуем (можно запускать на нескольких ядрах).

Недостатки:

- Медленная сходимость для некоторых задач.
- Требует настройки параметров (Pc, Pm, размер популяции).
- Не гарантирует нахождение глобального оптимума.

Примеры применения:

- **Оптимизация**: нахождение минимума функции.
- **Машинное обучение**: подбор гиперпараметров нейросетей.
- **Комбинаторные задачи**: задача коммивояжера (TSP), раскройка материалов.
- **Искусственная жизнь**: моделирование эволюции поведения.

Ordered Crossover (OX) — это оператор кроссовера, который сохраняет относительный порядок городов из родителей, создавая корректного потомка без дублирования городов. Он особенно эффективен для задач, где важна последовательность элементов (например, TSP).

1. Выбирается случайный сегмент из первого родителя и копируется в потомка.
2. Остальные города заполняются из второго родителя, пропуская те, что уже есть в выбранном сегменте.
3. Сохраняется порядок из второго родителя (относительное расположение городов).

Partially Mapped Crossover (PMX) — это оператор кроссовера, который:

- Сохраняет абсолютное положение некоторых городов (из выбранного сегмента).
- Сохраняет относительный порядок остальных городов
- Гарантирует корректные перестановки (без повторяющихся городов)

Cycle Crossover (CX) — это оператор кроссовера, который:

- Сохраняет точные позиции городов из родителей
- Формирует потомка на основе циклов между родительскими хромосомами
- Гарантирует корректные перестановки без дублирования городов
- Особенно эффективен для задач, где важна структура маршрута

Edge Recombination Crossover (ERX) — это продвинутый оператор кроссовера, который:

- Специально разработан для задач маршрутизации (TSP)
- Работает с ребрами графа (соседними связями между городами)
- Минимизирует разрывы существующих связей в родительских маршрутах
- Сохраняет локальную структуру маршрутов

Alternating Position Crossover (APX) — это простой, но эффективный оператор кроссовера, который:

- Поочередно выбирает города из двух родителей
- Автоматически исключает дубликаты
- Сохраняет частичную структуру обоих родительских маршрутов

- Особенно полезен на ранних стадиях эволюционного алгоритма

Жадный алгоритм «Ближайший сосед» — это эвристический подход, который на каждом шаге выбирает локально оптимальное решение (ближайший непосещённый город), надеясь, что это приведёт к глобально оптимальному решению.

Алгоритм пошагово:

1. Инициализация:
    - Выберите начальный город (можно случайный или первый в списке).
    - Пометить его как посещённый.
    - Инициализировать маршрут с этого города.
2. Пока есть непосещённые города:
    - Для текущего города найдите ближайший непосещённый город (с минимальным расстоянием).
    - Переместитесь в этот город.
    - Пометить его как посещённый.
    - Добавить город в маршрут.
3. Завершение:
    - Вернуться в начальный город из последнего посещённого города.
    - Добавить начальный город в маршрут.
4. Вывод результата:
    - Полный маршрут (цикл) и его общая длина.

Жадный алгоритм, основанный на методе наименьшего ребра, пытается строить маршрут, выбирая на каждом шаге самое короткое доступное ребро, не нарушающее условия задачи.

Алгоритм:

1. Инициализация:
    - Задаём взвешенный полный граф (матрица расстояний между городами).
    - Создаём пустой список рёбер маршрута.
    - Помечаем все рёбра как непосещённые.
2. Повторяем, пока не построим гамильтонов цикл:

- Выбираем ребро с наименьшим весом из оставшихся непосещённых.
- Проверяем, что добавление этого ребра не приводит к:
  – образованию цикла (кроме полного цикла в конце);
  – появлению вершины со степенью больше 2.
- Если ребро подходит, добавляем его в маршрут.
- Помечаем ребро как посещённое.

3. Завершение:
   - Когда все вершины соединены в единый цикл, алгоритм завершается.

Особенности жадного алгоритма:

- Простота реализации.
- Быстро работает ($O(n^2 \log n)$ для сортировки рёбер).
- Не всегда находит оптимальное решение (как в примере).
- Чувствителен к порядку городов.

Кроссовер на основе жадных алгоритмов:

1. Входные параметры:
   - списки городов (маршруты родителей)
   - матрица расстояний между городами
2. Инициализация:
   - Создается множество генов из первых двух городов каждого родителя
3. Выбор начальной пары:
   - Находим пару городов с наименьшим расстоянием между ними
   - Удаляем выбранные города из множества доступных
   - Инициализируем потомка `child` выбранной парой
4. Основной цикл построения маршрута:
   - На каждой итерации добавляем следующий город из каждого родителя (если его еще нет в потомке)
5. Поиск ближайшего соседа:
   - Ищем город, ближайший к началу маршрута
   - Ищем город, ближайший к концу маршрута
6. Добавление города в маршрут:

- Добавляем город с меньшим расстоянием либо в начало, либо в конец маршрута
- Удаляем добавленный город из множества доступных

7. Возврат результата:
   - Возвращаем построенный маршрут-потомок

Особенности алгоритма:

1. Комбинирует генетический алгоритм с жадной стратегией ближайшего соседа
2. На каждом шаге расширяет маршрут, добавляя наиболее подходящий город
3. Рассматривает возможность добавления как в начало, так и в конец маршрута
4. Использует города из обоих родительских маршрутов

Этот подход позволяет сочетать преимущества генетического алгоритма (комбинация решений родителей) с локальной оптимизацией (жадный выбор ближайших городов).

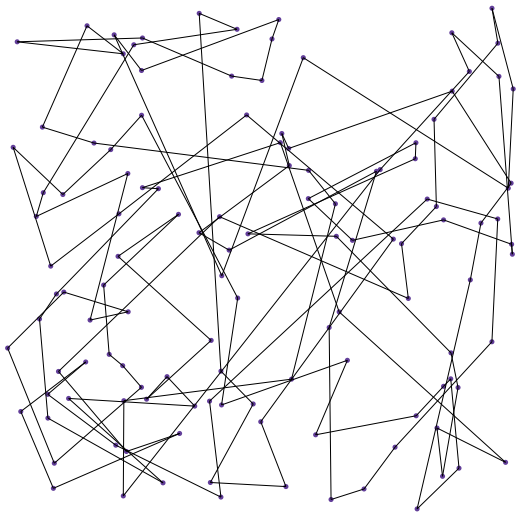
Эксперименты проводились на двух типах данных: случайно сгенерированных 125 городах и тестовом примере XQF131 (131 город) из базы данных TSPLIB `https://www.math.uwaterloo.ca/tsp/vlsi/index.html`.

Рассмотрим решение задачи коммивояжёра для случайно с генерированных 125 городов с координатами от 0 до 200. Размер популяции выбран 200, элита 40, вероятность мутации 0.01 (1%) и количество поколений равно 300. Построенные решения задачи коммивояжера для различных видов операторов кроссовера представлены в соответствии с рисунком 1.

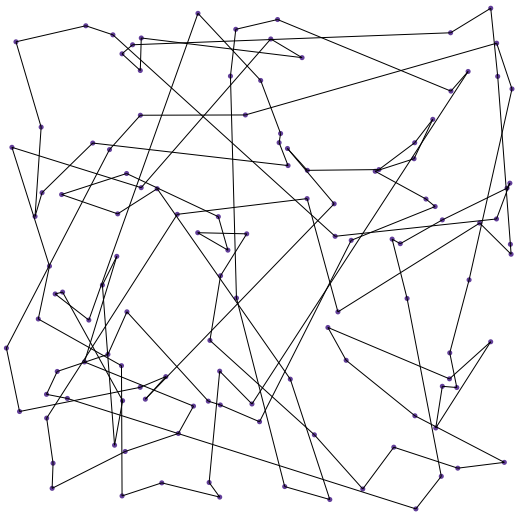
Результаты для различных операторов кроссовера:

- Ordered Crossover (OX): Длина маршрута — 5224.85.
- Partially Mapped Crossover (PMX): Длина маршрута — 4485.77.
- Cycle Crossover (CX): Длина маршрута — 5049.67.
- Edge Recombination Crossover (ERX): Длина маршрута — 4547.57.
- Alternating Position Crossover (APX): Длина маршрута — 6504.03.
- Greedy Crossover (greedy1): Длина маршрута — 1881.76.

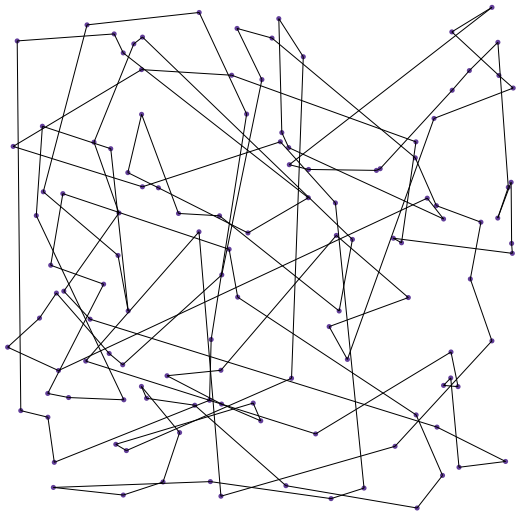
Выводы:



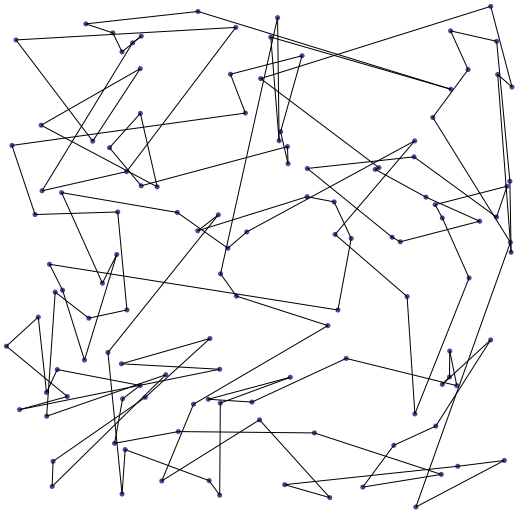
(a) Кроссовер OX, дистанция: 5224.85



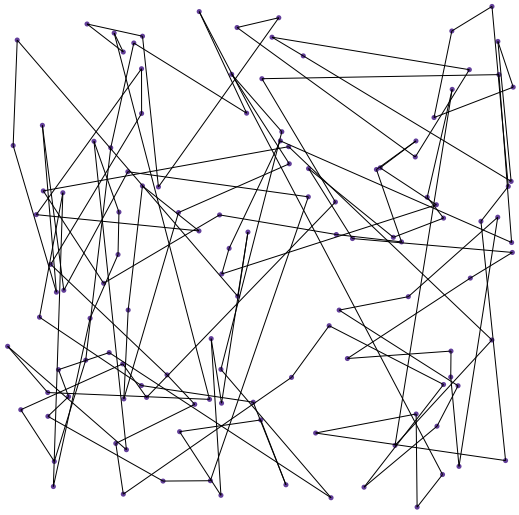
(b) Кроссовер PMX, дистанция: 4485.77



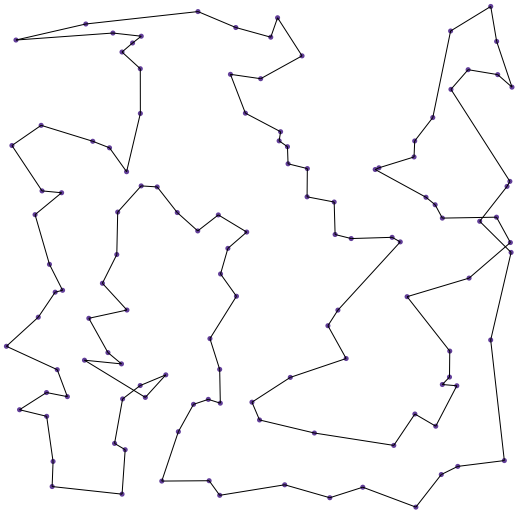
(c) Кроссовер CX, дистанция: 5049.67



(d) Кроссовер ERX, дистанция: 4547.57



(e) Кроссовер APX, дистанция: 6504.03



(f) Кроссовер greedy1, дистанция: 1881.76

Рисунок 1 — Решения задачи коммивояжера для случайных 125 городов

- Наилучший результат показал оператор Greedy Crossover, который значительно превзошёл остальные методы, сократив длину маршрута более чем в 2 раза по сравнению с другими кроссоверами.
- Операторы PMX и ERX также продемонстрировали относительно хорошие результаты, что связано с их способностью сохранять локальную структуру маршрутов.
- APX оказался наименее эффективным, что объясняется его простотой и отсутствием оптимизации на локальном уровне.

Заключение. Оператор кроссовера Greedy Crossover, сочетающий жадную стратегию ближайшего соседа с генетическим алгоритмом, показал наилучшие результаты для обоих тестовых случаев. Это связано с его способностью локально оптимизировать маршрут на каждом шаге. Другие виды кроссоверов:

- PMX и ERX демонстрируют стабильно хорошие результаты благодаря сохранению структуры родительских маршрутов.
- OX и CX менее эффективны, но остаются полезными для задач среднего размера.
- APX не рекомендуется для использования из-за низкой точности.

Результаты подтверждают, что выбор оператора кроссовера критически важен для качества решения TSP. Greedy Crossover может быть рекомендован для задач с большим количеством городов, где важна как скорость, так и точность.