МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №3
по дисциплине
«Генетические алгоритмы»
Вариант 18

Студент,
группы 5130201/20101 _____ Тищенко А. А.

Преподаватель _____ Большаков А. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

1 Постановка задачи

В данной работе были поставлены следующие задачи:

- Реализовать с использованием генетических алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта.
- Сравнить найденное решение с представленным в условии задачи оптимальным решением.
- Представить графически найденное решение.
- Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.

Индивидуальное задание вариант 18:

Дано: Эвклидовы координаты городов 38 городов в Джибути (см. Приложение А). Оптимальный тур представлен на Рис. 1, его длина равна 6659.

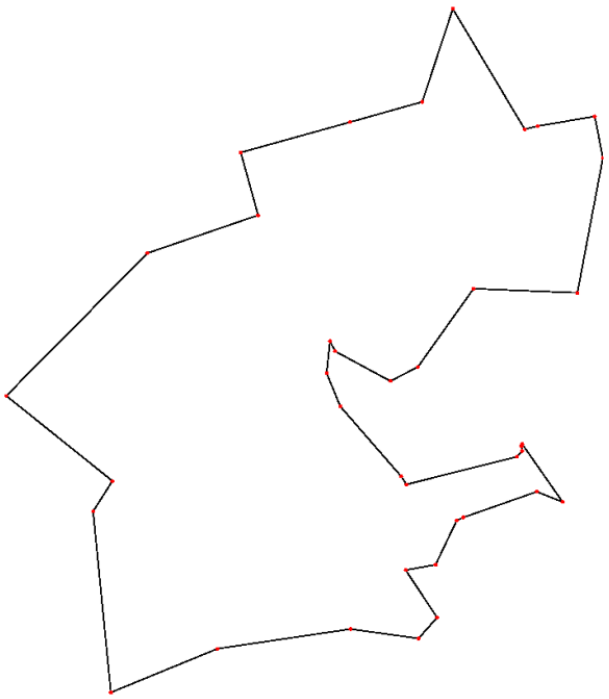

Рис. 1. Оптимальный тур для заданного набора данных

Требуется:

1. Реализовать с использованием генетических алгоритмов решение задачи коммивояжера.
2. Для туров использовать путевое представление.

2 Теоретические сведения

Генетические алгоритмы (ГА) используют принципы и терминологию, заимствованные у биологической науки – генетики. В ГА каждая особь представляет потенциальное решение некоторой проблемы. В классическом ГА особь кодируется строкой двоичных символов – хромосомой. Однако представление хромосомы зависит от постановки задачи: для непрерывных задач удобны векторы вещественных чисел (real-coded), тогда как для комбинаторных задач, таких как задача коммивояжера (ЗК), естественно представлять тур как перестановку городов. Длина хромосомы совпадает с числом элементов задачи; двоичное кодирование ЗК, как правило, неэффективно из-за необходимости «ремонта» решений после применения операторов.

Множество особей – потенциальных решений составляет популяцию. Поиск (суб)оптимального решения проблемы выполняется в процессе эволюции популяции - последовательного преобразования одного конечного множества решений в другое с помощью генетических операторов репродукции, кроссинговера и мутации.

Предварительно простой ГА случайным образом генерирует начальную популяцию стрингов (хромосом). Затем алгоритм генерирует следующее поколение (популяцию), с помощью трех основных генетических операторов:

1. Оператор репродукции (ОР);
2. Оператор скрещивания (кроссинговера, ОК);
3. Оператор мутации (ОМ).

ГА работает до тех пор, пока не будет выполнено заданное количество поколений (итераций) процесса эволюции или на некоторой генерации будет получено заданное качество или вследствие преждевременной сходимости при попадании в некоторый локальный оптимум. На Рис. 2 представлен простой генетический алгоритм.

Рис. 2. Простой генетический алгоритм

2.1 Основная терминология в генетических алгоритмах

Ген – элементарный код в хромосоме s_i, называемый также знаком или детектором (в классическом ГА $s_i = 0, 1$).

Хромосома – упорядоченная последовательность генов в виде закодированной структуры данных $S = (s_1, s_2, \ldots, s_n)$, определяющая решение. Представление зависит от типа задачи: для непрерывных задач — вектор вещественных чисел; для ЗК — перестановка городов (см. раздел о представлениях: соседское, порядковое и путевое).

Локус – местоположение (позиция, номер бита) данного гена в хромосоме.

Аллель – значение, которое принимает данный ген (например, 0 или 1).

Особь – одно потенциальное решение задачи (представляемое хромосомой).

Популяция – множество особей (хромосом), представляющих потенциальные решения.

Поколение – текущая популяция ГА на данной итерации алгоритма.

Генотип – набор хромосом данной особи. В популяции могут использоваться как отдельные хромосомы, так и целые генотипы.

Генофонд – множество всех возможных генотипов.

Фенотип – набор значений, соответствующий данному генотипу. Это декодированное множество параметров задачи (например, десятичное значение x, соответствующее двоичному коду).

Размер популяции N – число особей в популяции.

Число поколений – количество итераций, в течение которых производится поиск.

Селекция – совокупность правил, определяющих выживание особей на основе значений целевой функции.

Эволюция популяции – чередование поколений, в которых хромосомы изменяют свои признаки, чтобы каждая новая популяция лучше приспосабливалась к среде.

Фитнесс-функция – функция полезности, определяющая меру приспособленности особи. В задачах оптимизации она совпадает с целевой функцией или описывает близость к оптимальному решению.

2.2 Представления хромосом для задачи коммивояжера

Задача коммивояжера (ЗК) формулируется так: требуется посетить каждый из N городов ровно один раз и вернуться в исходную точку, минимизируя суммарную стоимость (или длину) тура. Естественным является представление тура как перестановки городов. На практике используются три основных представления, каждое со своими операторами рекомбинации:

2.2.1 Представление соседства

Тур задаётся списком из N городов, где в позиции i указан город j, означающий переход из города i в город j. Например, вектор (2 4 8 3 9 7 1 5 6) соответствует туру $1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7$. У каждого корректного тура есть единственное соседское представление, однако не всякая строка в этом представлении корректна (возможны преждевременные циклы, например $1 \to 2 \to 4 \to 1 \ldots$).

2.2.2 Порядковое представление

Тур представляется списком из N позиций; i-й элемент равен индексу города в текущем упорядоченном списке доступных городов. Например, при опорном списке $C = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ тур $1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7$ кодируется как $l = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$, последовательно «выбирая» элементы из C.

2.2.3 Путевое представление

Наиболее интуитивное представление: тур записывается как последовательность городов, например $5 \to 1 \to 7 \to 8 \to 9 \to 4 \to 6 \to 2 \to 3$ кодируется как (5 1 7 8 9 4 6 2 3). Это представление сохраняет относительный порядок городов и широко применяется на практике.

2.3 Кроссинговеры для представлений ЗК

Операторы рекомбинации должны сохранять допустимость туров (перестановочную природу решения). Для разных представлений используются различные кроссинговеры.

2.3.1 Кроссинговеры для представления соседства

Alternating Edges (обмен рёбрами): потомок строится, поочерёдно выбирая ребра у родителей: одно ребро у первого родителя, следующее — у второго, затем снова у первого и т.д. Если выбранное ребро замыкает цикл преждевременно, выбирается другое ещё не использованное ребро того же родителя, не образующее цикл.

Subtour Chunks (обмен подтурами): потомок формируется конкатенацией кусочков (подтуров), поочерёдно взятых у родителей. При образовании преждевременного цикла производится «ремонт» аналогично предыдущему оператору.

Heuristic Crossover (эвристический): стартуя из случайного города, на каждом шаге сравниваются два инцидентных ребра, предлагаемых родителями, и выбирается более короткое; если возникает цикл или ребро уже использовано, выбирается случайный ещё не посещённый город. Оператор нацелен на сохранение коротких рёбер, но может иметь нестабильную производительность.

2.3.2 Кроссинговеры для порядкового представления

Для порядкового представления корректность потомков обеспечивает классический одноточечный кроссовер: любые два родителя, разрезанные в одной позиции и склеенные, порождают допустимых потомков (поскольку выбор «по индексу» в оставшемся списке городов остаётся корректным).

2.3.3 Кроссинговеры для путевого представления

Для путевого представления широко применяются три оператора, гарантирующие корректную перестановку у потомков.

PMX (Partially Mapped Crossover). Идея: обменять подпоследовательности между родителями и построить отображение соответствий, которым затем разрешать конфликты (дубликаты).

Пример. Пусть точки разреза задают сегмент позиций $4 \dots 7$:

$$p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9), \quad p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3).$$

1) Копируем сегмент второго родителя в потомка o_1 и формируем отображение $\{4 \leftrightarrow 1,\ 5 \leftrightarrow 8,\ 6 \leftrightarrow 7,\ 7 \leftrightarrow 6\}$:

$$o_1 = (\_\ \_\ \_ \mid 1\ 8\ 7\ 6 \mid \_\ \_).$$

2) Заполняем прочие позиции по порядку из p_1, применяя отображение при конфликтах: $1 \mapsto 4$, $8 \mapsto 5$.

$$o_1 = (4\ 2\ 3 \mid 1\ 8\ 7\ 6 \mid 5\ 9).$$

Аналогично для o_2 (копируем сегмент из p_1, заполняем остальное из p_2):

$$o_2 = (1\ 8\ 2 \mid 4\ 5\ 6\ 7 \mid 9\ 3).$$

PMX сохраняет как позиции части элементов, так и относительный порядок/соответствия на остальной части хромосомы.

OX (Order Crossover). Идея: скопировать сегмент одного родителя и дозаполнить оставшиеся позиции элементами второго родителя в их порядке появления (пропуская уже скопированные).

Пример. С теми же родителями и разрезами $4 \dots 7$:

$$p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9), \quad p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3).$$

1) Копируем сегмент p_1 в o_1:

$$o_1 = (\_\ \_\ \_ \mid 4\ 5\ 6\ 7 \mid \_\ \_).$$

2) Обходя p_2 с позиции после правого разреза, дозаполняем: получаем

$$o_1 = (2\ 1\ 8 \mid 4\ 5\ 6\ 7 \mid 9\ 3).$$

Симметрично для o_2 (копируем сегмент из p_2 и дозаполняем порядком из p_1):

$$o_2 = (3\ 4\ 5 \mid 1\ 8\ 7\ 6 \mid 9\ 2).$$

Оператор OX сохраняет относительный порядок городов; циклический сдвиг тура несущественен.

CX (Cycle Crossover). Идея: находить циклы позиций, индуцированные взаимным расположением значений у родителей, и наследовать циклы по очереди из разных родителей.

Пример. Возьмём

$$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9), \quad p_2 = (4\ 5\ 2\ 1\ 8\ 7\ 6\ 9\ 3).$$

Построив циклы позиций, получим допустимых потомков, например:

$$o_1 = (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5), \quad o_2 = (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9).$$

CX сохраняет абсолютные позиции части элементов и способствует передаче «циклами» взаимных расположений.

Отметим, что путевое представление акцентирует порядок городов (а не стартовый город), поэтому туры, отличающиеся циклическим сдвигом, эквивалентны.

2.4 Мутации для путевого представления

Операторы мутации в ГА для задачи коммивояжёра должны сохранять допустимость решения (перестановочную структуру). Для путевого представления применяются специализированные операторы, которые модифицируют порядок городов, не нарушая корректности тура.

Swap (обмен двух элементов). Идея: выбрать случайным образом две позиции в маршруте и обменять находящиеся на них города местами.

Пример. Пусть исходный тур:

$$t = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9).$$

Выбираем позиции $i = 2$ и $j = 6$ (элементы 3 и 7). После обмена получаем:

$$t' = (1\ 2\ 7\ 4\ 5\ 6\ 3\ 8\ 9).$$

Оператор swap обеспечивает локальную модификацию тура, изменяя положение только двух городов.

Inversion (инверсия сегмента). Идея: выбрать случайный сегмент маршрута и обратить порядок городов внутри него.

Пример. Для того же тура выбираем позиции разреза $i = 3$ и $j = 7$ (сегмент 4 5 6 7):

$$t = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9).$$

Инвертируем выделенный сегмент:

$$t' = (1\ 2\ 3 \mid 7\ 6\ 5\ 4 \mid 8\ 9).$$

Инверсия сохраняет связность частей маршрута, меняя направление обхода в подтуре. Этот оператор особенно эффективен при наличии пересечений рёбер, так как инверсия может «распутать» некоторые из них и улучшить длину маршрута.

Insertion (вырезка и вставка). Идея: выбрать случайный город, удалить его из текущей позиции и вставить в другую случайную позицию маршрута.

Пример. Пусть исходный тур:

$$t = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9).$$

Выбираем город на позиции $i = 3$ (элемент 4) и целевую позицию $j = 7$. Удаляем элемент 4:

$$t_{\text{tmp}} = (1\ 2\ 3\ 5\ 6\ 7\ 8\ 9).$$

Вставляем 4 на позицию 7:

$$t' = (1\ 2\ 3\ 5\ 6\ 7\ 4\ 8\ 9).$$

insertion изменяет расположение одного города относительно других, смещая соседей.

Все три оператора гарантируют сохранение корректной перестановки: каждый город остаётся в туре ровно один раз.

3 Особенности реализации

В рамках работы создана мини-библиотека `gen.py` для решения задачи коммивояжёра (TSP) генетическим алгоритмом с путевым представлением хромосом. Второй модуль `expirements.py` организует серийные эксперименты (перебор параметров, форматирование и сохранение результатов).

- **Кодирование особей**: каждая хромосома представлена как перестановка городов (`Chromosome = list[int]`), где каждый элемент – индекс города. Популяция – список хромосом (`Population = list[Chromosome]`). Инициализация случайными перестановками без повторений:
  - `initialize_random_population(pop_size: int, cities: Cites) -> Population`
- **Фитнесс-функция**: целевая функция принимает хромосому (маршрут) и возвращает скалярное значение фитнесса (длину пути). Для режима минимизации используется внутреннее преобразование при селекции (сдвиг и инверсия знака), что позволяет применять рулетку:
  - `eval_population(population: Population, fitness_func: FitnessFn) -> Fitnesses`
  - Логика режима минимизации в `genetic_algorithm(config: GARunConfig) -> GARunResult`
- **Селекция (рулетка)**: вероятности нормируются после сдвига на минимальное значение в поколении (устойчиво к отрицательным фитнессам). Функция: `reproduction(population: Population, fitnesses: Fitnesses) -> Population`.
- **Кроссинговер**: реализованы специализированные операторы для перестановок: PMX (Partially Mapped Crossover), OX (Ordered Crossover) и CX (Cycle Crossover). Кроссинговер выполняется попарно по перемешанной популяции с вероятностью p_c. Функции:
  - `partially_mapped_crossover_fn(p1: Chromosome, p2: Chromosome) -> tuple[Chromosome, Chromosome]`
  - `ordered_crossover_fn(p1: Chromosome, p2: Chromosome) -> tuple[Chromosome, Chromosome]`
  - `cycle_crossover_fn(p1: Chromosome, p2: Chromosome) -> tuple[Chromosome, Chromosome]`
  - `crossover(population: Population, pc: float, crossover_fn: CrossoverFn) -> Population`
- **Мутация**: реализованы три типа мутаций для перестановок: обмен двух городов (swap), инверсия сегмента (inversion), вырезка и вставка города (insertion). Мутация применяется с вероятностью p_m. Функции:
  - `swap_mutation_fn(chrom: Chromosome) -> Chromosome`
  - `inversion_mutation_fn(chrom: Chromosome) -> Chromosome`

  - `insertion_mutation_fn(chrom: Chromosome) -> Chromosome`
  - `mutation(population: Population, pm: float, mutation_fn: MutationFn) -> Population`

- **Критерий остановки**: поддерживаются критерии по максимальному количеству поколений, повторению лучшего результата, достижению порогового значения фитнесса. Хранится история всех поколений. Проверка выполняется в функции:

  `genetic_algorithm(config: GARunConfig) -> GARunResult`.

- **Визуализация**: реализована отрисовка маршрутов обхода городов на плоскости с отображением лучшей особи поколения. Функции:

  - `plot_tour(cities: list[tuple[float, float]], tour: list[int], ax: Axes)`
  - `save_generation(generation: Generation, history: list[Generation], config: GARunConfig)`
  - `plot_fitness_history(result: GARunResult, save_path: str | None) -> None`

- **Элитизм**: поддерживается перенос лучших особей без изменения в следующее поколение (`elitism` параметр).

- **Измерение времени**: длительность вычислений возвращается в миллисекундах как часть `GARunResult.time_ms`.

- **Файловая организация**: результаты экспериментов сохраняются в структуре `experiments/N/` с таблицами результатов. Задействованные функции:

  - `clear_results_directory(results_dir: str) -> None`
  - Функции для проведения экспериментов в модуле `expirements.py`

В модуле `expirements.py` задаются координаты городов и параметры экспериментов. Серийные запуски и сохранение результатов реализованы для исследования влияния параметров ГА на качество решения задачи коммивояжёра.

4 Результаты работы

На Рис. 4–11 представлены результаты работы генетического алгоритма со следующими параметрами:

- $N = 500$ – размер популяции.
- $p_c = 0.9$ – вероятность кроссинговера.
- $p_m = 0.3$ – вероятность мутации.
- 2500 – максимальное количество поколений.
- 3 – количество "элитных"особей, переносимых без изменения в следующее поколение.
- Partially mapped crossover - кроссовер.
- Inversion mutation - мутация

На Рис. 3 показан график изменения фитнесса по поколениям. Видно, что алгоритм постепенно сходится к минимально возможному значению фитнеса. Лучший маршрут был найден на поколнении №1896 (см. Рис. 11).

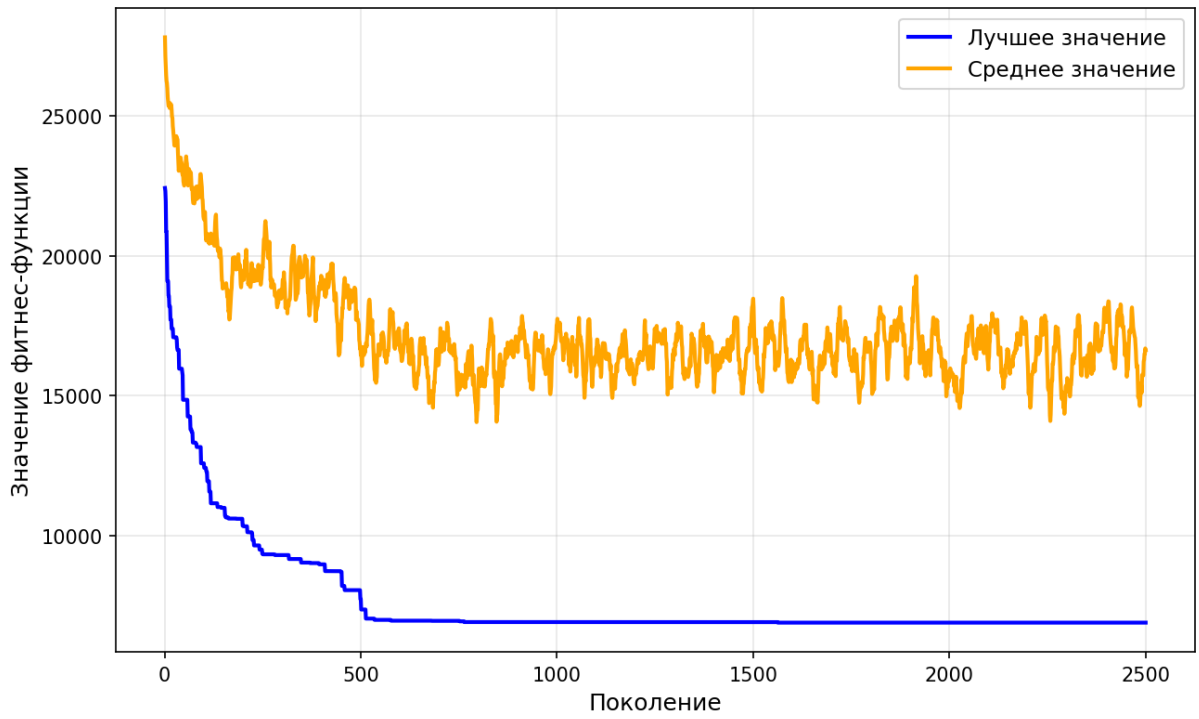

Рис. 3. График изменения фитнесса по поколениям

Поколение #1. Лучшая особь: 22422. Среднее значение: 27804

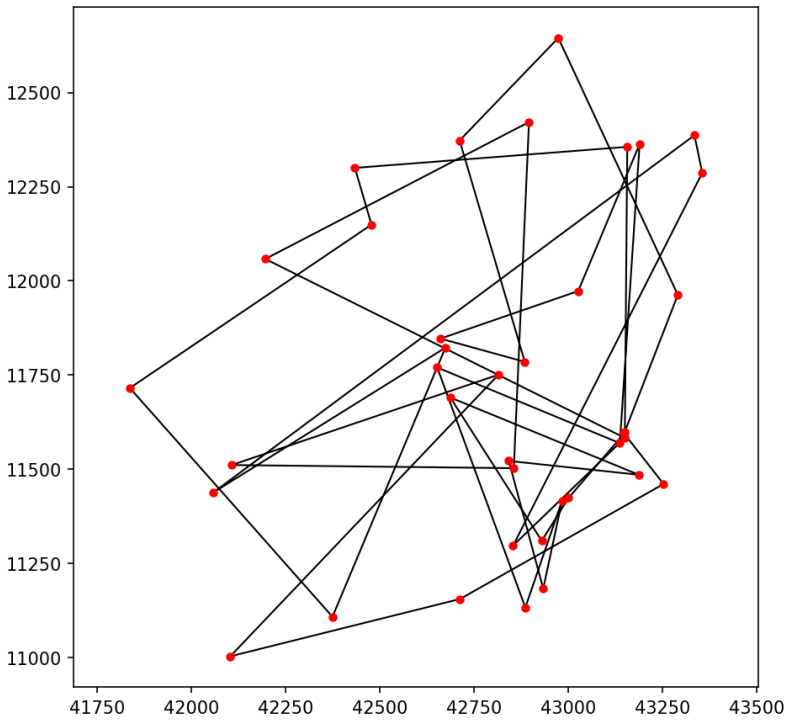

Рис. 4. Лучший маршрут поколения №1

Поколение #5. Лучшая особь: 20866. Среднее значение: 26249

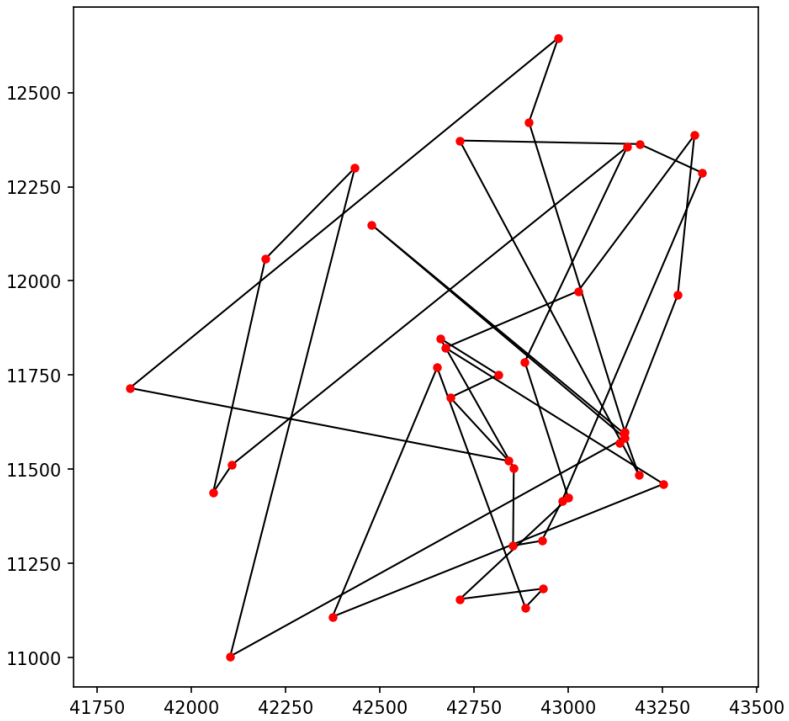

Рис. 5. Лучший маршрут поколения №5

Поколение #50. Лучшая особь: 14855. Среднее значение: 22508

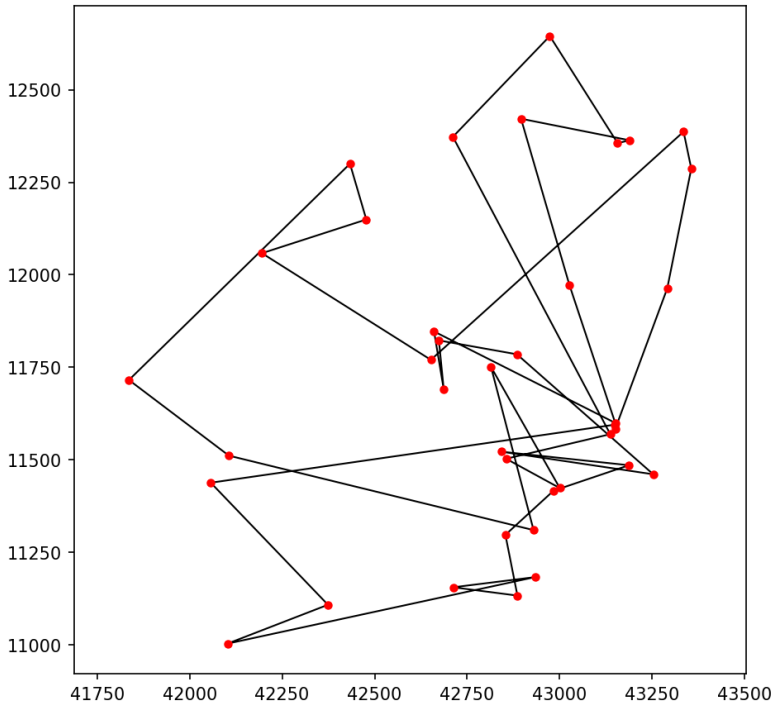

Рис. 6. Лучший маршрут поколения №50

Поколение #100. Лучшая особь: 12587. Среднее значение: 21322

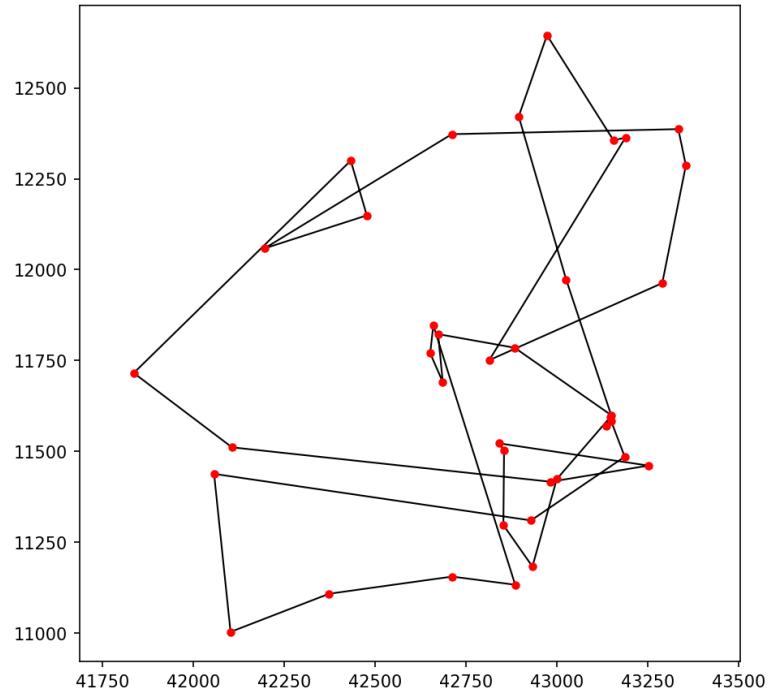

Рис. 7. Лучший маршрут поколения №100

Поколение #300. Лучшая особь: 9308. Среднее значение: 18848

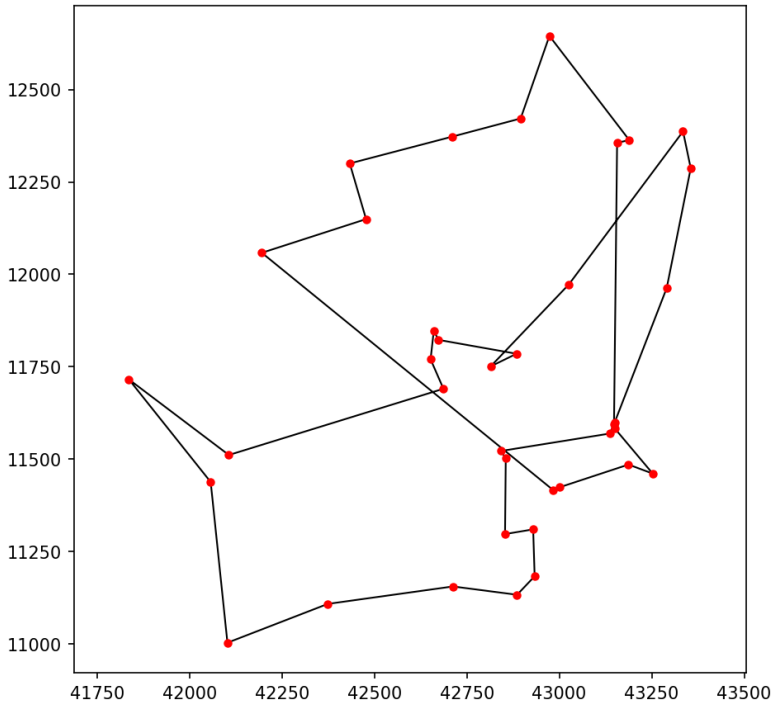

Рис. 8. Лучший маршрут поколения №300

Поколение #500. Лучшая особь: 7731. Среднее значение: 16462

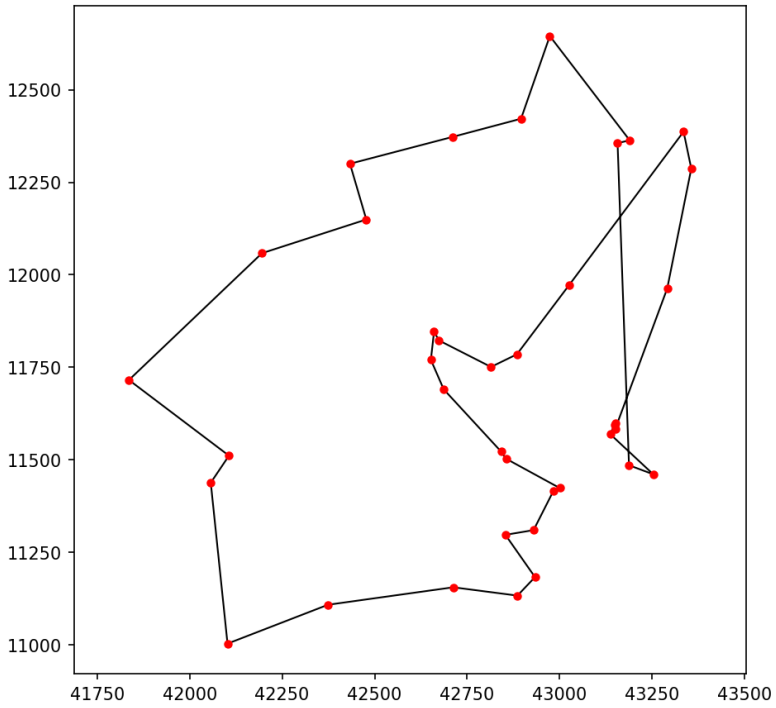

Рис. 9. Лучший маршрут поколения №500

Поколение #900. Лучшая особь: 6915. Среднее значение: 16561

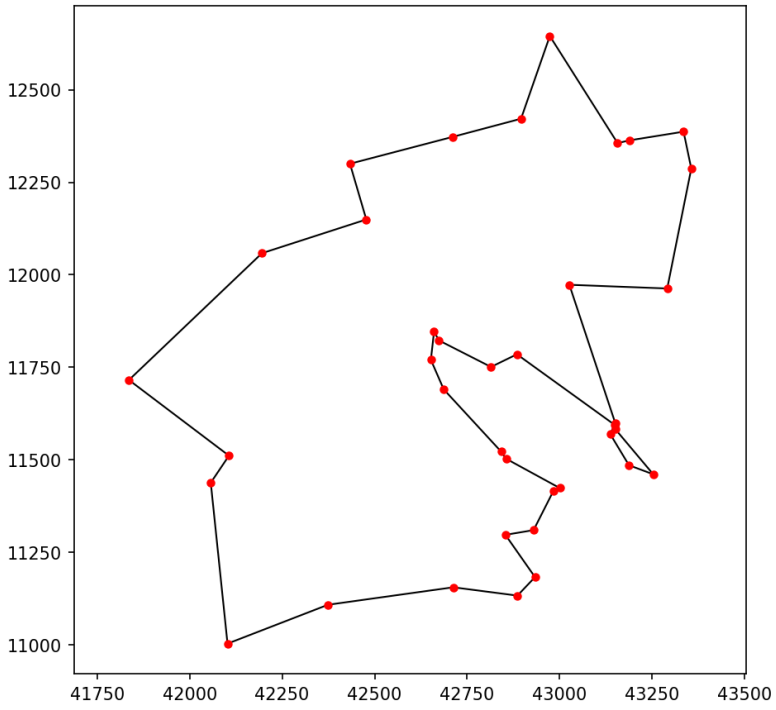

Рис. 10. Лучший маршрут поколения №900

Поколение #1896. Лучшая особь: 6894.1007. Среднее значение: 16348.7848

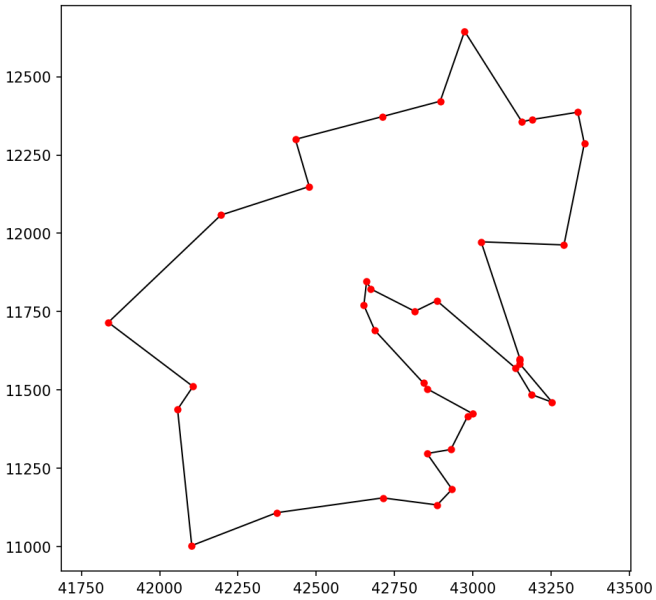

Рис. 11. Лучший маршрут поколения №1896

5 Исследование реализации

5.1 Проведение измерений

В рамках лабораторной работы необходимо было исследовать зависимость времени выполнения задачи и количества поколений от популяции и вероятностей кроссинговера и мутации хромосомы

Для исследования были выбраны следующие значения параметров:

- $N = 10, 50, 100, 500$ – размер популяции.
- $p_c = 0.5, 0.6, 0.7, 0.8, 0.9$ – вероятность кроссинговера.
- $p_m = 0.05, 0.2, 0.3, 0.4, 0.5, 0.8$ – вероятность мутации.
- 3 – количество "элитных"особей, переносимых без изменения в следующее поколение.
- Partially mapped crossover - кроссовер.
- Inversion mutation - мутация
- 7000 - пороговое значение фитнеса для остановки алгоритма.

Результаты измерений представлены в таблицах 1–4. В ячейках указано время в миллисекундах нахождения минимума функции. В скобках указано количество поколений, за которое было найдено решение. Во второй строке указано усреднённое по всем запускам лучшее значение фитнеса. Если в ячейке стоит прочерк, то это означает, что решение не было найдено за 2500 поколений. Лучшее значение по времени выполнения и по значению фитнеса для каждого размера популяции выделено цветом и жирным шрифтом.

Таблица 1. Результаты для $N = 10$

$P_c \setminus P_m$	0.050	0.200	0.300	0.400	0.500	0.800
0.5	—	—	674.4 (1783) 6943.28027	715.1 (1856) 6925.47290	—	225.5 (567) 6984.75016
0.6	—	—	550.6 (1427) 6899.82219	649.4 (1653) 6897.01699	—	—
0.7	—	—	476.7 (1216) 6796.98342	287.4 (724) 6977.43028	**201.0** **(503)** 6794.32839	—
0.8	—	—	—	767.2 (1852) 6810.96744	253.3 (623) 6905.36866	—
0.9	—	—	—	—	—	—
1.0	—	750.9 (1847) 6988.52746	415.7 (1016) 6897.99266	465.7 (1126) **6762.96572**	275.9 (662) 6997.70453	—

Таблица 2. Результаты для $N = 50$

$P_c \setminus P_m$	0.050	0.200	0.300	0.400	0.500	0.800
0.5	1711.4 (1083) 6927.73356	—	1642.7 (1015) 6894.10066	1355.6 (809) 6938.12550	—	936.3 (544) 6925.57274
0.6	1338.4 (828) 6952.02461	889.1 (552) 6951.40489	1142.5 (687) 6963.17379	1446.9 (864) 6992.95281	—	2646.2 (1509) 6932.85788
0.7	1860.8 (1146) 6996.63686	—	2387.8 (1378) 6999.00110	—	**809.9** **(474)** 6965.83938	1614.7 (918) 6990.50067
0.8	—	—	1244.4 (713) **6704.60011**	1500.5 (859) 6970.42362	1013.5 (581) 6998.68282	—
0.9	—	—	—	—	—	—
1.0	—	891.6 (503) 6952.80522	—	—	1489.6 (824) 6735.40661	3685.9 (1978) 6989.21247

Таблица 3. Результаты для $N = 100$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.050**	**0.200**	**0.300**	**0.400**	**0.500**	**0.800**
0.5	1342.3 (441) 6988.26353	1467.7 (459) 6958.81642	4041.3 (1269) **6839.94363**	—	—	3635.1 (1046) 6966.14098
0.6	2460.6 (763) 6872.20321	**1316.5** **(409)** 6861.65860	—	2310.7 (691) 6912.50054	2220.9 (663) 6907.57533	—
0.7	—	1934.1 (591) 6933.87982	—	1966.0 (587) 6943.09435	2872.9 (840) 6998.39699	—
0.8	3227.9 (969) 6990.28735	1754.4 (523) 6996.67018	—	2152.8 (621) 6988.30495	8057.2 (2236) 6899.21400	—
0.9	—	—	3794.4 (1079) 6963.79199	2549.3 (721) 6975.22091	4469.6 (1249) 6945.46938	8919.4 (2375) 6858.03529
1.0	4164.4 (1215) 6927.53288	—	—	—	3618.7 (1019) 6898.56773	—

Таблица 4. Результаты для $N = 500$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.050**	**0.200**	**0.300**	**0.400**	**0.500**	**0.800**
0.5	11709.8 (782) 6994.15844	**5232.3** **(341)** 6957.38204	10676.9 (674) 6980.66167	6849.7 (430) 6782.99526	—	13051.6 (775) 6880.72481
0.6	7193.3 (461) 6960.64487	—	—	14856.5 (866) 6941.77959	12944.9 (776) 6958.57319	19051.6 (1102) 6951.30787
0.7	18611.7 (1150) 6810.96744	23286.9 (1413) 6895.65139	—	14141.6 (830) 6976.37927	—	—
0.8	25456.0 (1556) 6962.40902	—	20592.3 (1223) 6998.71555	—	—	38979.2 (2097) 6842.54074
0.9	14260.1 (825) 6967.60134	26692.6 (1551) 6922.32909	—	—	29235.4 (1644) **6667.02991**	41352.0 (2252) 6765.87009
1.0	34026.1 (1996) 6953.24255	—	—	—	—	—

5.2 Анализ результатов

Наилучшее найденное решение составило **6667.03** при параметрах $N = 500$, $P_c = 0.9$, $P_m = 0.5$ за 1644 поколения. Это всего на **0.12%** хуже оптимального значения 6659, что демонстрирует высокую эффективность алгоритма. Наихудшие результаты показала конфигурация с $N = 10$, $P_c = 0.7$, $P_m = 0.3$ (лучший фитнес 6796.98), что на 2.07% хуже оптимума. Малый размер популяции в 10 особей оказался недостаточным для стабильного поиска качественных решений — более половины конфигураций при $N = 10$ вообще не нашли решение за 2500 поколений.

Наиболее быстрая конфигурация — $N = 10$, $P_c = 0.7$, $P_m = 0.5$ — нашла решение за **201 мс** (503 поколения). Однако качество решения при таких параметрах нестабильно. Среди конфигураций с большой популяцией лучшее время показала $N = 500$, $P_c = 0.5$, $P_m = 0.2$ — **5232 мс** (341 поколение), что является оптимальным балансом скорости и качества для больших популяций.

С ростом размера популяции наблюдается явное улучшение качества решений: при $N = 10$ лучший результат 6762.97, при $N = 500$ — 6667.03. Одновременно количество необходимых поколений снижается (с 503 до 341), но общее время выполнения растет линейно из-за увеличения числа особей в каждом поколении. Этот эффект объясняется тем, что большая популяция обеспечивает большее генетическое разнообразие, позволяя алгоритму быстрее находить оптимальные решения.

Что касается вероятности кроссовера, средние значения $P_c = 0.6$–0.8 показывают стабильные результаты для всех размеров популяций. Экстремальные значения ($P_c = 0.9$ или 1.0) работают хорошо только при больших популяциях ($N \geq 100$), при малых — часто приводят к преждевременной сходимости (наблюдается много прочерков в таблицах). Это связано с тем, что высокая вероятность кроссовера при малой популяции быстро приводит к гомогенизации генофонда.

Анализ влияния вероятности мутации показал, что низкие значения $P_m = 0.05$ неэффективны для малых популяций — недостаточно разнообразия для выхода из локальных минимумов. Умеренные значения $P_m = 0.2$–0.5 демонстрируют лучшие результаты, обеспечивая баланс между эксплуатацией найденных решений и исследованием нового пространства поиска. Высокое значение $P_m = 0.8$ часто приводит к расхождению алгоритма, так как слишком сильные изменения разрушают хорошие решения быстрее, чем алгоритм успевает их найти (многие конфигурации не нашли решение за отведенное время).

6 Ответ на контрольный вопрос

Вопрос: Тур в порядковом представлении, используемые кроссинговеры.

Ответ: Тур представляется списком из N позиций; i-й элемент равен индексу города в текущем упорядоченном списке доступных городов. Например, при опорном списке $C = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ тур $1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7$ кодируется как $l = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$, последовательно «выбирая» элементы из C.

Для порядкового представления корректность потомков обеспечивает классический одноточечный кроссовер: любые два родителя, разрезанные в одной позиции и склеенные, порождают допустимых потомков (поскольку выбор «по индексу» в оставшемся списке городов остаётся корректным).

Заключение

В ходе третьей лабораторной работы была успешно решена задача коммивояжера с использованием генетических алгоритмов для 38 городов Джибути:

1. Изучен теоретический материал о представлениях туров (соседское, порядковое, путевое) и специализированных операторах кроссинговера и мутации для задачи коммивояжера;
2. Создана программная библиотека на языке Python с реализацией путевого представления хромосом, операторов PMX, OX и CX для кроссинговера, операторов swap, inversion и insertion для мутации, а также селекции методом рулетки с поддержкой элитизма;
3. Проведено исследование влияния параметров генетического алгоритма на качество и скорость нахождения решения для популяций размером 10, 50, 100 и 500 особей с различными значениями вероятностей кроссинговера и мутации;
4. Получено решение с длиной маршрута 6667.03, отклоняющееся от оптимального значения 6659 всего на 0.12%.

Список литературы

[1] Методические указания по выполнению лабораторных работ к курсу «Генетические алгоритмы», 119 стр.