Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Системы искусственного интеллекта**

(название дисциплины)

**Лабораторная работа № 1**

**Вариант № 7**

Выполнил

Студент группы \_\_\_ИВТАПбд-41\_\_\_\_

\_\_\_\_\_\_\_\_\_\_Карпова А.С.\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О.)

Проверил(а):

ст. преподаватель кафедры «ВТ»

(должность)

\_\_\_\_\_\_\_\_\_Хайруллин И.Д.\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О.)

Ульяновск

2024

**Оглавление**

[Задание 3](#_Toc185885339)

[Ход работы 3](#_Toc185885340)

[Тестирование 5](#_Toc185885341)

[Вывод 6](#_Toc185885342)

[Приложение. Исходный код 7](#_Toc185885343)

# 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.

Вариант:

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.

# Ход работы

1. Программа задает количество фабрик (n=7n = 7) и городов (k=3k = 3). Для каждой фабрики и города случайным образом генерируются:

* **Объем продукции**, который можно произвести (для фабрик) или потребить (для городов).
* **Расстояния** между каждой фабрикой и каждым городом, выраженные в километрах.

2. Сначала программа решает задачу с использованием **перебора всех возможных маршрутов**. Она вычисляет **стоимость маршрута** с помощью функции total\_cost. Стоимость определяется как сумма произведений расстояний и объемов продукции, доставляемой с фабрик в города. Программа проверяет все перестановки маршрутов с помощью функции permutations из модуля itertools. Для каждого маршрута вычисляется его стоимость. После проверки всех возможных маршрутов выбирается маршрут с минимальной стоимостью.

3. Для более эффективного решения задачи используется **генетический алгоритм**. Его работа включает следующие шаги:

**Индивидуумы и популяция:**

* Каждый индивид представляет собой маршрут распределения продукции.
* Индивидуумы формируют начальную популяцию.

**Фитнес-функция:**

* Для каждого индивидуума рассчитывается стоимость маршрута, которая служит мерой его "приспособленности" (чем ниже стоимость, тем лучше).

Программа реализует три различных метода мутации для изменения маршрутов:

* mutate\_swap**:** Обмен местами двух случайных пунктов в маршруте.
* mutate\_random**:** Замена одного пункта маршрута случайным новым значением.
* mutate\_shift**:** Перемещение сегмента маршрута в другое место.

**Эволюция поколений:**

* Отбираются лучшие индивиды (с наименьшей стоимостью маршрута) для формирования новой популяции.
* В новую популяцию добавляются дети, созданные из лучших индивидов, с применением мутаций.
* Средняя и минимальная стоимость маршрута (фитнес) записываются для последующего анализа.

4.Программа запускает генетический алгоритм для каждого метода мутации. В результате для каждого метода сохраняются:

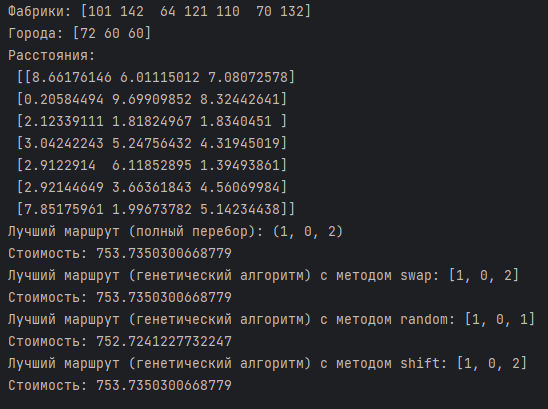
* Средняя стоимость маршрута для каждого поколения.
* Лучшая (минимальная) стоимость маршрута.

5. Программа отображает (рис. 1):

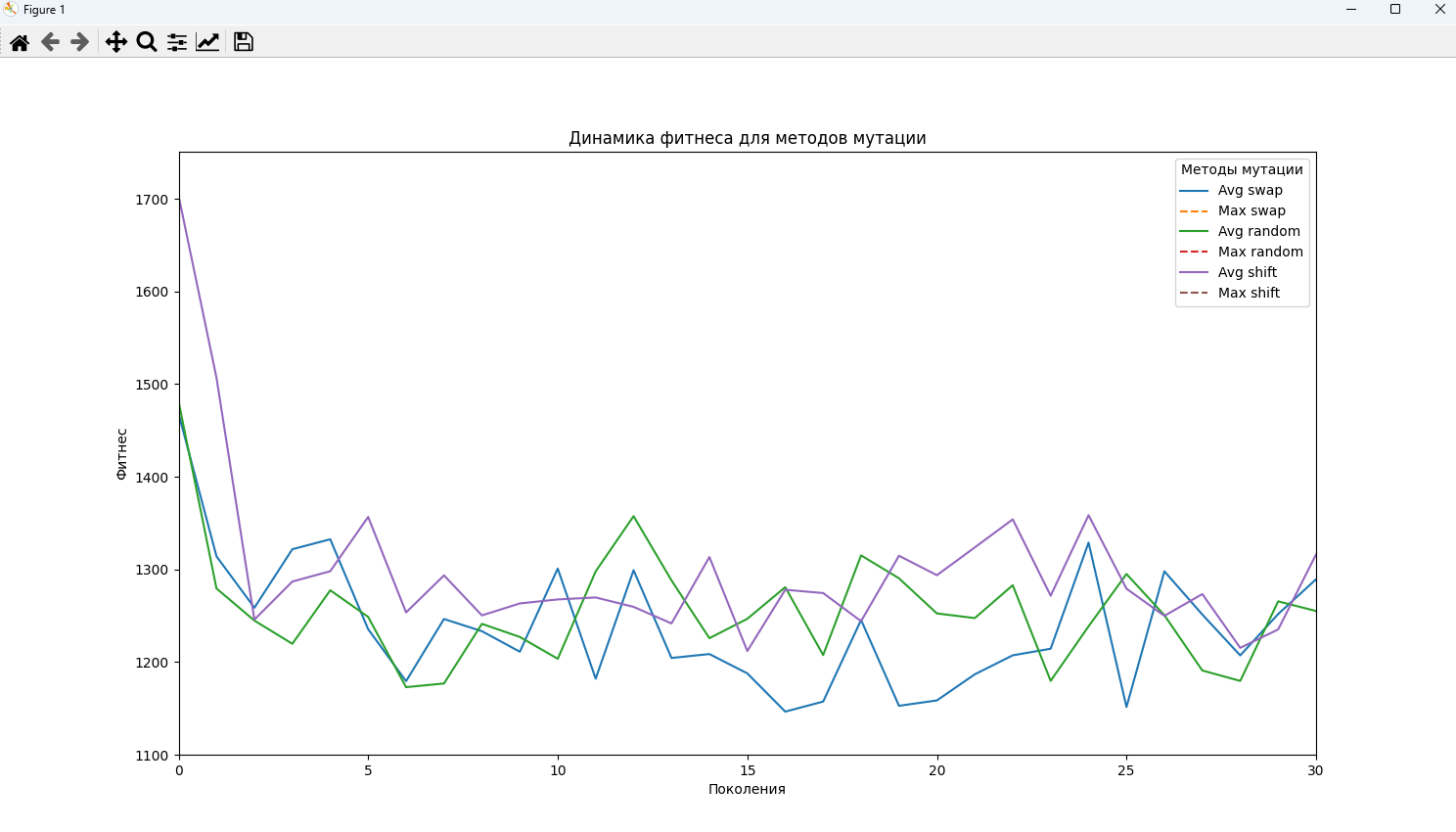
* **Лучший маршрут и его стоимость**, найденные методом полного перебора.
* **Лучшие маршруты и их стоимости** для каждого метода мутации в генетическом алгоритме.

6. Для каждого метода мутации строятся графики (рис. 2) динамика средней стоимости маршрута по поколениям. На графиках отображаются эффективность методов мутации и сравнение их производительности.

# Тестирование



*Рис 1. Отображение в консоли лучшего маршрута и его стоимости, найденные методом полного перебора и для каждого метода мутации*



*Рис. 2 Графики динамики средней стоимости маршрута по поколениям*

# Вывод

В данной лабораторной работе был разработан скрипт, который подбирал оптимальный вариант распределения ресурсов между пунктами производства (фабриками) и потребления (городами) с учетом расстояний при помощи генетического алгоритма.

# Приложение. Исходный код

|  |
| --- |
| import numpy as np import random import matplotlib.pyplot as plt from itertools import permutations  # Параметры n = 7 # Количество пунктов производства k = 3 # Количество городов p\_crossover = 0.8 # вероятность скрещивания (не используется) p\_mutation = 0.2 # вероятность мутации индивидуума max\_products\_per\_city = 100 # Максимум продуктов на город max\_products\_per\_factory = 150 # Максимум продуктов на фабрику  # Генерация случайных данных np.random.seed(42) # Для воспроизводимости factories = np.random.randint(50, max\_products\_per\_factory, n) cities = np.random.randint(50, max\_products\_per\_city, k)  # Расстояния между фабриками и городами (в километрах) distances = np.random.rand(n, k) \* 10 # случайные расстояния от 0 до 10 км  print("Фабрики:", factories) print("Города:", cities) print("Расстояния:\n", distances)   # Полный перебор def total\_cost(route):  cost = 0  for i, city in enumerate(route):  cost += distances[i][city] \* factories[i]  return cost   def exhaustive\_search():  best\_cost = float('inf')  best\_route = None  for perm in permutations(range(k)):  current\_cost = total\_cost(perm)  if current\_cost < best\_cost:  best\_cost = current\_cost  best\_route = perm  return best\_route, best\_cost   best\_route, best\_cost = exhaustive\_search() print("Лучший маршрут (полный перебор):", best\_route) print("Стоимость:", best\_cost)   # Генетический алгоритм с методами мутации. class Individual:  def \_\_init\_\_(self, factories, cities):  self.route = self.create\_route(len(cities))  self.fitness = self.calculate\_fitness(factories, cities)   def create\_route(self, length):  return random.sample(range(length), length)   def calculate\_fitness(self, factories, cities):  total\_cost = sum(distances[f][c] \* factories[f] for f, c in enumerate(self.route))  return total\_cost   # Функции мутации def mutate\_swap(individual):  idx1, idx2 = random.sample(range(len(individual.route)), 2)  individual.route[idx1], individual.route[idx2] = individual.route[idx2], individual.route[idx1]  individual.fitness = individual.calculate\_fitness(factories, cities)   def mutate\_random(individual):  index = random.randint(0, len(individual.route) - 1)  individual.route[index] = random.randint(0, len(cities) - 1)  individual.fitness = individual.calculate\_fitness(factories, cities)   def mutate\_shift(individual):  start = random.randint(0, len(individual.route) - 1)  end = random.randint(start + 1, len(individual.route))  segment = individual.route[start:end]  position = random.randint(0, len(individual.route) - len(segment))  individual.route = individual.route[:position] + segment + individual.route[position:]  individual.fitness = individual.calculate\_fitness(factories, cities)   def genetic\_algorithm(population\_size=100, generations=500):  mutation\_methods = ['swap', 'random', 'shift']   fitness\_results\_avg = {m: [] for m in mutation\_methods}  fitness\_results\_max = {m: [] for m in mutation\_methods}   for mutation\_method in mutation\_methods:  population = [Individual(factories, cities) for \_ in range(population\_size)]  for generation in range(generations):  population.sort(key=lambda x: x.fitness)  new\_population = population[:population\_size // 2]   while len(new\_population) < population\_size:  parent1, parent2 = random.sample(new\_population[:20], 2)   # Поскольку мы не используем методы скрещивания,  # просто создаем новые индивиды.  child1 = Individual(factories, cities)  child2 = Individual(factories, cities)   if random.random() < p\_mutation:  if mutation\_method == 'swap':  mutate\_swap(child1)  mutate\_swap(child2)  elif mutation\_method == 'random':  mutate\_random(child1)  mutate\_random(child2)  elif mutation\_method == 'shift':  mutate\_shift(child1)  mutate\_shift(child2)   new\_population.extend([child1, child2])   population = new\_population   avg\_fitness = np.mean([ind.fitness for ind in population])  max\_fitness = min(ind.fitness for ind in population) # Минимальная стоимость (лучший фитнес)   fitness\_results\_avg[mutation\_method].append(avg\_fitness)  fitness\_results\_max[mutation\_method].append(max\_fitness)   best\_individual\_genetic = min(population, key=lambda x: x.fitness)   print(f"Лучший маршрут (генетический алгоритм) с методом {mutation\_method}: {best\_individual\_genetic.route}")  print(f"Стоимость: {best\_individual\_genetic.fitness}")   return fitness\_results\_avg, fitness\_results\_max   fitness\_results\_avg\_genetic\_algorithm, fitness\_results\_max\_genetic\_algorithm = genetic\_algorithm()  # Визуализация результатов только для методов мутации plt.figure(figsize=(15, 8))  # Отображение только для методов мутации for mutation\_method in ['swap', 'random', 'shift']:  plt.plot(range(len(fitness\_results\_avg\_genetic\_algorithm[mutation\_method])),  fitness\_results\_avg\_genetic\_algorithm[mutation\_method],  label=f'Avg {mutation\_method}')   plt.plot(range(len(fitness\_results\_max\_genetic\_algorithm[mutation\_method])),  fitness\_results\_max\_genetic\_algorithm[mutation\_method],  label=f'Max {mutation\_method}', linestyle='--')  plt.xlabel('Поколения') plt.ylabel('Фитнес') plt.title('Динамика фитнеса для методов мутации')  # Установка пределов осей plt.xlim(0, 30) # Ограничение по оси X plt.ylim(1100, ) # Ограничение по оси Y (от нуля до максимума фитнеса)  plt.legend(title='Методы мутации') # Заголовок для легенды plt.show() |