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 группы ИВТАПбд-41

Каршибоев Я.Ш.

Проверил:

преподаватель кафедры «ВТ»

Хайрулин И.Д.

Ульяновск

2025

**Постановка задачи**

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.

**Ход работы**

Вариант содержит постановку задачи: 6. На языке Python разработайте скрипт, который с помощью генетического алгоритма и полного перебора решает следующую задачу. Дано N наименований продуктов, для каждого из которых известно m характеристик. Необходимо получить самый дешевый рацион из k наименований, удовлетворяющий заданным медицинским нормам для каждой из m характеристик.

## **Данные**

Пример данных о продуктах:

# Список продуктов: (название, калории, белки, жиры, углеводы, цена)  
products = [  
 ("Хлеб", 250, 8, 2, 50, 20),  
 ("Молоко", 60, 3, 3, 5, 25),  
 ("Яйца", 150, 12, 10, 1, 30),  
 ("Курица", 200, 20, 8, 0, 70),  
 ("Рыба", 180, 22, 6, 0, 90),  
 ("Сыр", 300, 20, 25, 2, 100),  
 ("Яблоки", 50, 0, 0, 14, 15),  
 ("Картофель", 80, 2, 0, 18, 10),  
 ("Гречка", 330, 12, 3, 68, 40),  
 ("Морковь", 40, 1, 0, 10, 8)  
]

**Медицинские нормы:**

* Калории: 2000–2500 ккал
* Белки: 60–200 г
* Жиры: 40–80 г
* Углеводы: 300–400 г

## **3. Структура хромосомы**

Хромосома представляет собой **список индексов продуктов**, выбранных для рациона:

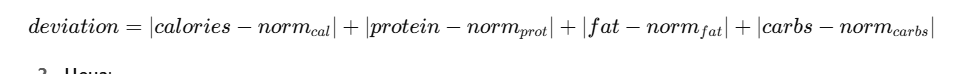
chromosome = [0, 3, 5, 7] индексы продуктов из списка

Каждый ген соответствует одному продукту. Длина хромосомы = k.

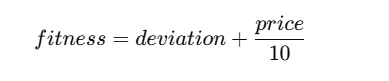
## **4. Функция приспособленности (fitness)**

Функция приспособленности оценивает рацион по двум критериям:

1. **Отклонение от нормы по характеристикам**:



**Итоговая функция:**



Цель: **минимизация функции приспособленности**.

### 5. Генетический алгоритм

Генетический алгоритм был использован для поиска оптимального рациона из заданного числа продуктов, учитывая медицинские нормы и минимизацию стоимости. Алгоритм реализует эволюционную модель, имитируя процессы отбора, скрещивания и мутации в природе.

**Этапы алгоритма:**

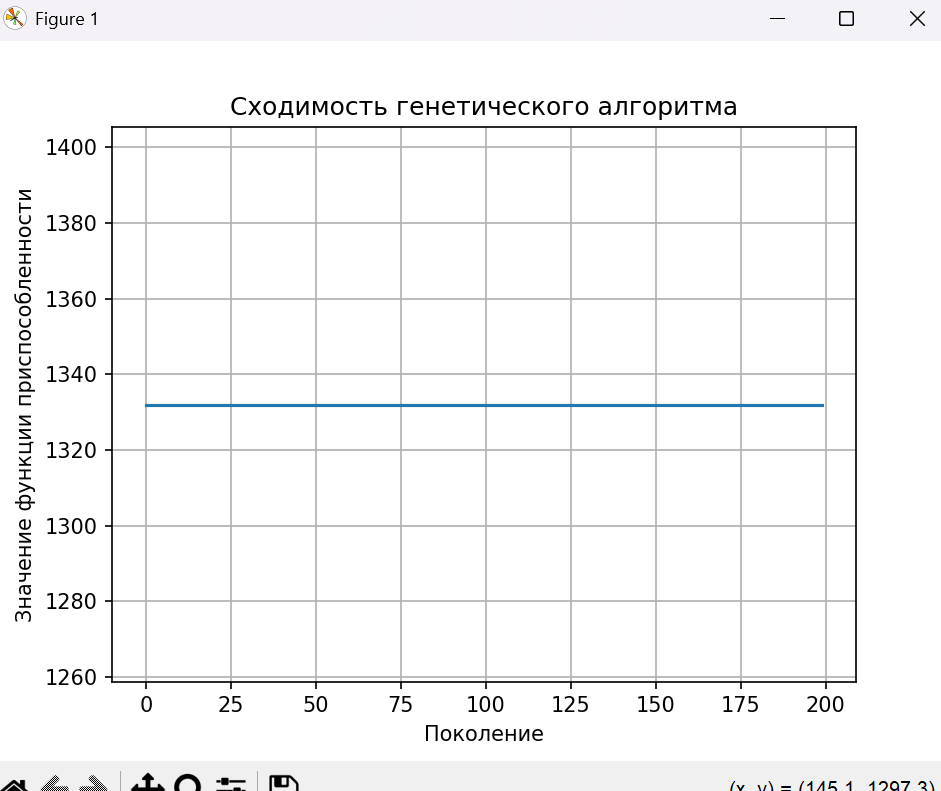
1. **Инициализация популяции**  
   Для каждого индивида (варианта рациона) случайным образом выбирается k продуктов из общего списка. Каждый индивид представляет собой возможный рацион.
2. **Селекция**  
   Для отбора родителей используется **турнирный отбор**: из случайной подвыборки популяции выбирается индивид с наилучшей приспособленностью (fitness), который будет участвовать в формировании потомства.
3. **Кроссовер (скрещивание)**  
   Для получения новых индивидов применяются разные методы:
   * **Одноточечный кроссовер** – разделение хромосом родителей в одной точке и обмен половинками.
   * **Двухточечный кроссовер** – обмен сегментами хромосом между двумя точками.
   * **Равномерный кроссовер** – поочерёдное и попарное объединение генов родителей на основе случайного выбора.
4. **Мутация**  
   Для поддержания разнообразия и исследования пространства решений применяются различные мутации:
   * **Замена случайного продукта** – один продукт в рационе заменяется другим случайным образом.
   * **Инверсия последовательности** – выбранный сегмент хромосомы разворачивается.
   * **Перестановка элементов** – случайная перестановка генов внутри хромосомы.
5. **Эволюция**  
   Алгоритм повторяет циклы **селекции, кроссовера и мутации** заданное количество поколений. На каждом поколении сохраняется лучший рацион по значению функции приспособленности.

Результатом работы алгоритма является рацион продуктов, который минимизирует стоимость и максимально удовлетворяет заданным медицинским нормам.

## **6. Эксперименты**

* Размер популяции: 100
* Количество поколений: 50–200
* Использовались различные комбинации методов кроссовера и мутации
* На каждом поколении сохранялось **лучшее значение приспособленности**

## **7. Результаты**



На графике показан процесс работы генетического алгоритма при поиске оптимального рациона питания.

* **Ось X** — номер поколения. Каждое поколение соответствует одной итерации эволюционного процесса.
* **Ось Y** — значение функции приспособленности (fitness), которая учитывает отклонение от медицинских норм по калориям, белкам, жирам и углеводам, а также стоимость рациона. Меньшие значения означают более подходящий рацион.

График демонстрирует, как со временем **лучшие решения становятся более оптимальными**, а средняя приспособленность популяции постепенно улучшается. Появление **разных траекторий при разных запусках** объясняется случайностью в генерации начальной популяции, турнирном отборе, кроссовере и мутациях.

Листинг

import numpy as np  
import random  
import itertools  
import matplotlib.pyplot as plt  
  
# Список продуктов: (название, калории, белки, жиры, углеводы, цена)  
products = [  
 ("Хлеб", 250, 8, 2, 50, 20),  
 ("Молоко", 60, 3, 3, 5, 25),  
 ("Яйца", 150, 12, 10, 1, 30),  
 ("Курица", 200, 20, 8, 0, 70),  
 ("Рыба", 180, 22, 6, 0, 90),  
 ("Сыр", 300, 20, 25, 2, 100),  
 ("Яблоки", 50, 0, 0, 14, 15),  
 ("Картофель", 80, 2, 0, 18, 10),  
 ("Гречка", 330, 12, 3, 68, 40),  
 ("Морковь", 40, 1, 0, 10, 8)  
]  
  
N = len(products) # количество продуктов  
M = 4 # число характеристик (калории, белки, жиры, углеводы)  
K = 4 # число продуктов в рационе  
  
# Медицинские нормы (диапазоны для калорий, белков, жиров и углеводов)  
norms = {  
 "calories": (2000, 2500),  
 "protein": (60, 200),  
 "fat": (40, 80),  
 "carbs": (300, 400)  
}  
  
# ФУНКЦИЯ ПРИСПОСОБЛЕННОСТИ  
  
def evaluate(chromosome):  
 *"""  
 Функция оценивает, насколько хороший рацион (хромосома).  
 Возвращает стоимость + штраф, если нормы не соблюдены.  
 """* chosen = [products[i] for i in chromosome]  
  
 # Суммируем характеристики по выбранным продуктам  
 total\_cal = sum(p[1] for p in chosen)  
 total\_prot = sum(p[2] for p in chosen)  
 total\_fat = sum(p[3] for p in chosen)  
 total\_carb = sum(p[4] for p in chosen)  
 total\_cost = sum(p[5] for p in chosen)  
  
 # Штраф за выход за пределы нормы  
 penalty = 0  
 if not (norms["calories"][0] <= total\_cal <= norms["calories"][1]):  
 penalty += abs(total\_cal - np.clip(total\_cal, \*norms["calories"]))  
 if not (norms["protein"][0] <= total\_prot <= norms["protein"][1]):  
 penalty += abs(total\_prot - np.clip(total\_prot, \*norms["protein"]))  
 if not (norms["fat"][0] <= total\_fat <= norms["fat"][1]):  
 penalty += abs(total\_fat - np.clip(total\_fat, \*norms["fat"]))  
 if not (norms["carbs"][0] <= total\_carb <= norms["carbs"][1]):  
 penalty += abs(total\_carb - np.clip(total\_carb, \*norms["carbs"]))  
  
 return total\_cost + penalty  
  
# ВСПОМОГАТЕЛЬНАЯ ФУНКЦИЯ  
  
def repair\_chromosome(chrom):  
 *"""  
 Исправляем хромосому:  
 - убираем дублирующиеся продукты  
 - добавляем случайные продукты до K штук  
 """* chrom = list(dict.fromkeys(chrom)) # удаляем дубликаты, сохраняя порядок  
 while len(chrom) < K:  
 chrom.append(random.randrange(N)) # добавляем случайный продукт  
 chrom = list(dict.fromkeys(chrom)) # убираем возможные дубли  
 return chrom[:K]  
  
# ГЕНЕТИЧЕСКИЙ АЛГОРИТМ  
  
def initialize\_population(pop\_size):  
 *"""Создаем начальную популяцию из случайных рационов."""* return [random.sample(range(N), K) for \_ in range(pop\_size)]  
  
# --- Кроссоверы (способы скрещивания) ---  
def crossover\_one\_point(p1, p2):  
 *"""Одноточечный кроссовер"""* point = random.randint(1, K - 1)  
 child = p1[:point] + [g for g in p2 if g not in p1[:point]]  
 return repair\_chromosome(child)  
  
def crossover\_two\_point(p1, p2):  
 *"""Двухточечный кроссовер"""* a, b = sorted(random.sample(range(K), 2))  
 child = p1[:a] + p2[a:b] + [g for g in p1 if g not in p2[a:b]]  
 return repair\_chromosome(child)  
  
def crossover\_uniform(p1, p2):  
 *"""Равномерный кроссовер"""* child = [p1[i] if random.random() < 0.5 else p2[i] for i in range(K)]  
 return repair\_chromosome(child)  
  
# --- Мутации ---  
def mutation\_swap(chrom):  
 *"""Меняем местами два продукта"""* if len(chrom) < 2:  
 return chrom  
 i, j = random.sample(range(len(chrom)), 2)  
 chrom[i], chrom[j] = chrom[j], chrom[i]  
 return repair\_chromosome(chrom)  
  
def mutation\_replace(chrom):  
 *"""Заменяем один продукт на случайный другой"""* i = random.randrange(len(chrom))  
 chrom[i] = random.randrange(N)  
 return repair\_chromosome(chrom)  
  
def mutation\_shuffle(chrom):  
 *"""Перемешиваем продукты внутри рациона"""* random.shuffle(chrom)  
 return repair\_chromosome(chrom)  
  
# --- Селекция ---  
def selection(pop, fitnesses):  
 *"""  
 Турнирная селекция:  
 выбираем лучший из двух случайных особей  
 """* i, j = random.sample(range(len(pop)), 2)  
 return pop[i] if fitnesses[i] < fitnesses[j] else pop[j]  
  
# --- Основной цикл генетического алгоритма ---  
def genetic\_algorithm(generations=100, pop\_size=50):  
 pop = initialize\_population(pop\_size) # начальная популяция  
 best\_scores = [] # история лучших значений  
 best\_solution = None # лучшее решение  
  
 for gen in range(generations):  
 fitnesses = [evaluate(ind) for ind in pop] # оценка популяции  
 best\_idx = np.argmin(fitnesses)  
 if best\_solution is None or fitnesses[best\_idx] < evaluate(best\_solution):  
 best\_solution = pop[best\_idx] # сохраняем лучшее решение  
 best\_scores.append(min(fitnesses))  
  
 new\_pop = []  
 for \_ in range(pop\_size):  
 # выбираем родителей  
 p1, p2 = selection(pop, fitnesses), selection(pop, fitnesses)  
 # случайный кроссовер  
 cross = random.choice([crossover\_one\_point, crossover\_two\_point, crossover\_uniform])  
 child = cross(p1.copy(), p2.copy())  
 # случайная мутация  
 mut = random.choice([mutation\_swap, mutation\_replace, mutation\_shuffle])  
 child = mut(child)  
 new\_pop.append(child)  
 pop = new\_pop  
 return best\_solution, best\_scores  
  
# ПОЛНЫЙ ПЕРЕБОР (для сравнения)  
def brute\_force():  
 *"""  
 Полный перебор всех комбинаций продуктов  
 (точное, но медленное решение).  
 """* best\_cost = float("inf")  
 best\_sol = None  
 for comb in itertools.combinations(range(N), K):  
 cost = evaluate(list(comb))  
 if cost < best\_cost:  
 best\_cost = cost  
 best\_sol = comb  
 return best\_sol, best\_cost  
  
# Запускаем генетический алгоритм и полный перебор  
best\_ga, scores = genetic\_algorithm(generations=200, pop\_size=100)  
best\_brute, brute\_cost = brute\_force()  
  
# Выводим результаты  
print("Лучшее решение (ГА):", [products[i][0] for i in best\_ga], "стоимость:", evaluate(best\_ga))  
print("Лучшее решение (перебор):", [products[i][0] for i in best\_brute], "стоимость:", brute\_cost)  
  
# График сходимости ГА  
plt.plot(scores)  
plt.title("Сходимость генетического алгоритма")  
plt.xlabel("Поколение")  
plt.ylabel("Значение функции приспособленности")  
plt.grid(True)  
plt.show()