Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Системы искусственного интеллекта**

**Лабораторная работа №1**

**«Генетические алгоритмы»**

**Вариант №4**

Выполнил:

cтудент группы ИВТАСбд-41

Рубцов Денис Алексеевич

Проверил:

Хайруллин И. Д.

Ульяновск

2025

# **СОДЕРЖАНИЕ**

[**СОДЕРЖАНИЕ** 2](#_Toc209973400)

[**Постановка задачи** 3](#_Toc209973401)

[**Ход работы** 4](#_Toc209973402)

[**Заключение** 6](#_Toc209973403)

[**Приложение А.** 7](#_Toc209973404)

## **Постановка задачи**

**Общее задание:**

Необходимо разработать программу на языке python, реализующую генетический алгоритм по предложенному вариантом заданию.

Провести эксперименты по разным способам скрещивания (не менее 3-х), разным способам мутирования (не менее трех). Результат отобразить в виде графиков

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.

Предоставить отчет о проделанной работе.

Вариант 4: 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 с минимальным его превышением, а транспортные расходы укладывались в определенные рамки.

## **Ход работы**

Первым делом в программе заданы количество пунктов производства n, количество городов k, мощность каждого пункта supply [n], потребности каждого города demand [k], бюджет на транспортные расходы transport\_budget и матрица стоимости перевозки transport\_costs.

Особь представляет собой матрицу распределения товаров для каждого города.

Функция приспособленности вводит такие штрафы за неудовлетворение спроса города, за превышение поставок над потребностями, за превышение производственных мощностей, за превышение транспортного бюджета.

В программе присутствуют 3 вида мутаций: случайное перераспределение, SWAP и гауссова мутация.

Случайное перераспределение берёт случайный завод (строку матрицы), считает, сколько он всего производит сумма и полностью заново перераспределяет этот объём между городами случайным образом.

В SWAP 2 случайных элемента меняются между собой.

В гауссовой мутации к значению прибавляется случайный шум из нормального распределения, не превышающий производства завода.

Также в программе приведены 3 вида скрещивания:

Одноточечное (single-point crossover) - случайная точка разреза хромосомы, где для потомка первая часть берётся у родителя 1, вторая – у родителя 2.

Двухточечное (two-point crossover) - выбираются две точки, участок между ними берётся от одного родителя, остальное – от другого.

Однородное / равномерное (uniform crossover)-каждый ген берётся случайно у одного из родителей с вероятностью 50%

В качестве метода селекции используется турнирный отбор: каждый раз из популяции случайным образом отбирается несколько претендентов (от двух и более). Затем, среди отобранных участников выбирается наиболее приспособленный (с наибольшим значением функции принадлежности). Он и переходит в новую выборку.

В качестве параметров генетического метода выступили:

* Размерность популяции 300.
* Количество генераций 300.
* Вероятность мутаций 50%.

Итогом программы является график зависимости приспособленности от поколения для всех возможных комбинаций видов скрещивания и мутаций

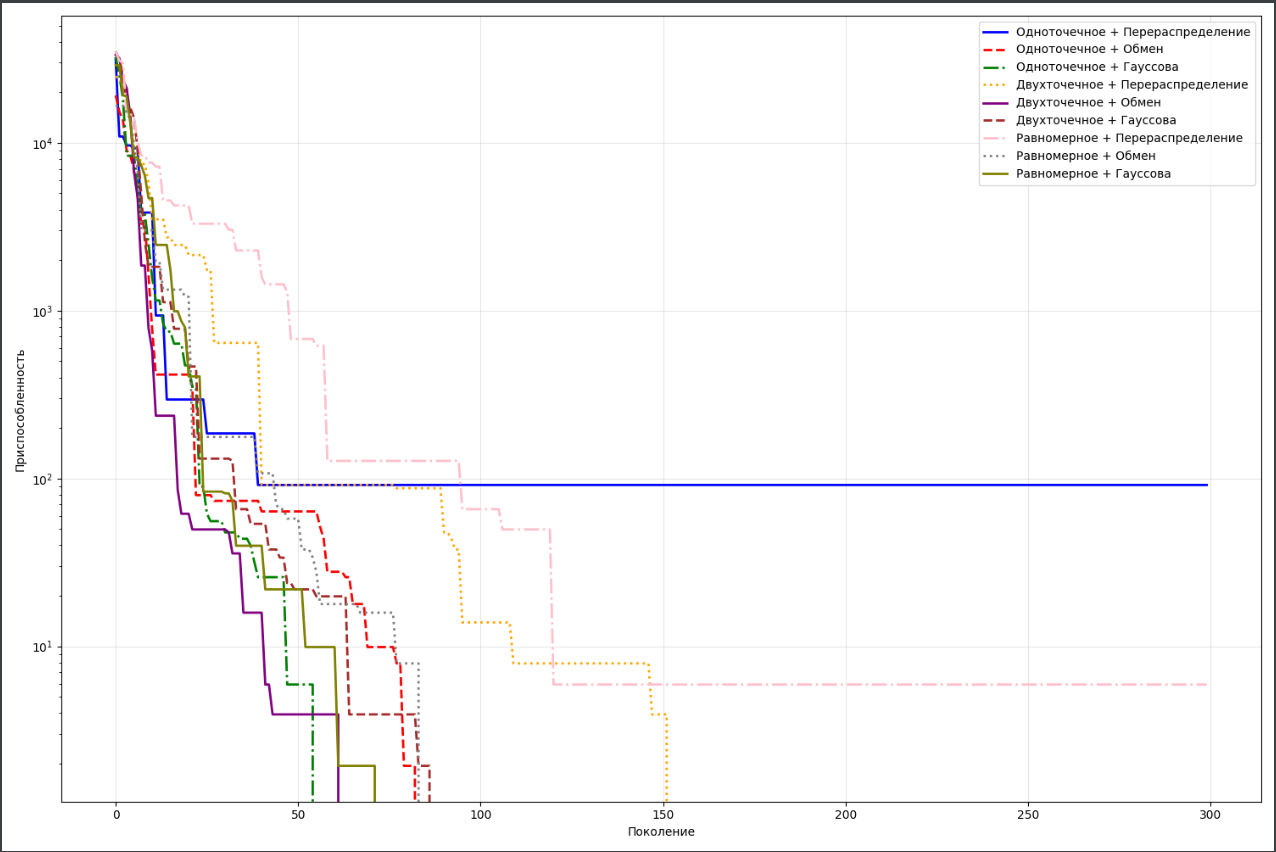


Рис. 1. Сравнение комбинаций скрещивания и мутаций в генетическом алгоритме.

Хуже всего показали себя комбинации с мутацией случайное перераспределение. Одноточечное гауссово показало лучший результат.

# **Заключение**

В рамках выполнения лабораторной работы были приобретены умения и навыки в реализации генетических алгоритмов.

Также была написана программа, реализующая все пункты задания

# **Приложение А.**

from random import randint, choices, choice, random, gauss

import numpy as np

import matplotlib.pyplot as plt

n = 4 # количество пунктов

k = 5 # количество городов

supply = [200, 300, 250, 350] # мощности каждого пункта

demand = [150, 200, 180, 220, 250] # потребности каждого города

transport\_budget = 32000 # бюджет на транспортные расходы

# стоимость перевозки одного продукта из пункта i в город j

transport\_costs = [

[15, 20, 25, 30, 35],

[18, 22, 28, 32, 38],

[12, 18, 20, 28, 32],

[20, 25, 30, 35, 40]

]

best\_fitness\_history = []

current\_method\_name = ""

def create\_individual():

"""Случайное решение (особь)"""

individual = []

for i in range(n):

# Распределяем производство пункта i между городами

max\_production = supply[i]

production = randint(0, max\_production)

distribution = []

for j in range(k - 1):

if production > 0:

alloc = randint(0, production)

distribution.append(alloc)

production -= alloc

else:

distribution.append(0)

# Последний город получает остаток

distribution.append(max(0, production))

individual.extend(distribution)

return individual

def decode\_individual(individual):

"""Декодирует в матрицу"""

matrix = []

idx = 0

for i in range(n):

row = []

for j in range(k):

row.append(individual[idx])

idx += 1

matrix.append(row)

return matrix

def fitness\_function(individual):

"""Функция приспособленности"""

matrix = decode\_individual(individual)

# Рассчитываем общие транспортные расходы

total\_cost = 0

for i in range(n):

for j in range(k):

total\_cost += matrix[i][j] \* transport\_costs[i][j]

# Рассчитываем распределение по городам

city\_supply = [0] \* k

for j in range(k):

for i in range(n):

city\_supply[j] += matrix[i][j]

# Рассчитываем использование производственных мощностей

factory\_usage = [0] \* n

for i in range(n):

factory\_usage[i] = sum(matrix[i])

# Комплексная функция приспособленности

fitness = 0

# Штраф за неудовлетворение спроса

for j in range(k):

if city\_supply[j] < demand[j]:

fitness += (demand[j] - city\_supply[j]) \* 100

# Штраф за превышение поставок над потребностями

for j in range(k):

if city\_supply[j] > demand[j]:

fitness += (city\_supply[j] - demand[j]) \* 2

#Штраф за превышение производственных мощностей

for i in range(n):

if factory\_usage[i] > supply[i]:

fitness += (factory\_usage[i] - supply[i]) \* 50

# Штраф за превышение транспортного бюджета

if total\_cost > transport\_budget:

fitness = float('inf')

# Поощрение за эффективное использование мощностей

for i in range(n):

if factory\_usage[i] <= supply[i] and factory\_usage[i] > 0:

fitness -= factory\_usage[i] \* 0.01

return max(0, fitness)

def mutation\_random\_reallocate(individual):

mutated = individual.copy()

# случайный пункт производства для мутации

factory\_idx = randint(0, n - 1)

# Индексы в массиве, соответствующие этому пункту

start\_idx = factory\_idx \* k

end\_idx = start\_idx + k

current\_production = sum(mutated[start\_idx:end\_idx])

# Генерируем новое распределение

new\_distribution = []

remaining = current\_production

for j in range(k - 1):

if remaining > 0:

alloc = randint(0, remaining)

new\_distribution.append(alloc)

remaining -= alloc

else:

new\_distribution.append(0)

new\_distribution.append(max(0, remaining))

mutated[start\_idx:end\_idx] = new\_distribution

return mutated

def mutation\_swap(individual):

"""обмен значений между двумя случайными позициями"""

mutated = individual.copy()

idx1 = randint(0, len(mutated) - 1)

idx2 = randint(0, len(mutated) - 1)

mutated[idx1], mutated[idx2] = mutated[idx2], mutated[idx1]

return mutated

def mutation\_gaussian(individual):

mutated = individual.copy()

num\_mutations = randint(1, 3)

for \_ in range(num\_mutations):

pos = randint(0, len(mutated) - 1)

# Определяем, к какому производству относится эта позиция

factory\_idx = pos // k

# Гауссово изменение с ограничением в пределах производственных мощностей

change = int(gauss(0, supply[factory\_idx] \* 0.1))

new\_value = max(0, mutated[pos] + change)

new\_value = min(new\_value, supply[factory\_idx])

mutated[pos] = new\_value

return mutated

def mutation(individual, method='random\_reallocate'):

if method == 'random\_reallocate':

return mutation\_random\_reallocate(individual)

elif method == 'swap':

return mutation\_swap(individual)

elif method == 'gaussian':

return mutation\_gaussian(individual)

else:

return mutation\_random\_reallocate(individual)

def crossover\_one\_point(parent1, parent2):

child = []

for i in range(n):

start\_idx = i \* k

end\_idx = start\_idx + k

if random() < 0.5:

# Берем распределение от первого родителя

child.extend(parent1[start\_idx:end\_idx])

else:

# Берем распределение от второго родителя

child.extend(parent2[start\_idx:end\_idx])

return child

def crossover\_two\_point(parent1, parent2):

child = parent1.copy()

# Выбираем две случайные точки разрыва

point1 = randint(1, len(parent1) - 2)

point2 = randint(point1 + 1, len(parent1) - 1)

# Заменяем сегмент между точками разрыва

child[point1:point2] = parent2[point1:point2]

return child

def crossover\_uniform(parent1, parent2):

"""Равномерное скрещивание"""

child = []

for i in range(len(parent1)):

# С вероятностью 50% берем ген от одного из родителей

if random() < 0.5:

child.append(parent1[i])

else:

child.append(parent2[i])

return child

def crossover(parent1, parent2, method='one\_point'):

if method == 'one\_point':

return crossover\_one\_point(parent1, parent2)

elif method == 'two\_point':

return crossover\_two\_point(parent1, parent2)

elif method == 'uniform':

return crossover\_uniform(parent1, parent2)

else:

return crossover\_one\_point(parent1, parent2)

def selection(population, fitnesses):

# Турнирный отбор

selected = []

tournament\_size = 3

for \_ in range(len(population)):

# Выбираем случайных особей для турнира

tournament = choices(range(len(population)), k=tournament\_size)

# Выбираем лучшую из турнира

winner = min(tournament, key=lambda x: fitnesses[x])

selected.append(population[winner])

return selected

def genetic\_algorithm(population\_size=100, generations=200, mutation\_rate=0.1,

crossover\_method='one\_point', mutation\_method='random\_reallocate'):

population = [create\_individual() for \_ in range(population\_size)]

best\_individual = None

best\_fitness = float('inf')

global best\_fitness\_history, current\_method\_name

best\_fitness\_history = []

current\_method\_name = f"{crossover\_method}\_{mutation\_method}"

for generation in range(generations):

# Вычисление приспособленности

fitnesses = [fitness\_function(ind) for ind in population]

# Обновление лучшего решения

current\_best = min(fitnesses)

if current\_best < best\_fitness:

best\_fitness = current\_best

best\_individual = population[fitnesses.index(current\_best)]

best\_fitness\_history.append(best\_fitness)

# Отбор

selected = selection(population, fitnesses)

# Скрещивание

new\_population = []

for i in range(0, len(selected), 2):

if i + 1 < len(selected):

child1 = crossover(selected[i], selected[i + 1], crossover\_method)

child2 = crossover(selected[i + 1], selected[i], crossover\_method)

new\_population.extend([child1, child2])

# Мутация

for i in range(len(new\_population)):

if random() < mutation\_rate:

new\_population[i] = mutation(new\_population[i], mutation\_method)

population = new\_population

if generation % 50 == 0:

print(f"Поколение {generation}, лучшая приспособленность: {best\_fitness}")

return best\_individual, best\_fitness

def plot\_fitness\_progress(all\_methods\_data):

colors = ['blue', 'red', 'green', 'orange', 'purple', 'brown', 'pink', 'gray', 'olive']

line\_styles = ['-', '--', '-.', ':', '-', '--', '-.', ':', '-']

crossover\_methods = ['one\_point', 'two\_point', 'uniform']

mutation\_methods = ['random\_reallocate', 'swap', 'gaussian']

plt.figure(figsize=(15, 10))

color\_idx = 0

for crossover\_method in crossover\_methods:

for mutation\_method in mutation\_methods:

method\_key = f"{crossover\_method}\_{mutation\_method}"

if method\_key in all\_methods\_data:

data = all\_methods\_data[method\_key]

generations = range(len(data))

crossover\_name = {

'one\_point': 'Одноточечное',

'two\_point': 'Двухточечное',

'uniform': 'Равномерное'

}[crossover\_method]

mutation\_name = {

'random\_reallocate': 'Перераспределение',

'swap': 'Обмен',

'gaussian': 'Гауссова'

}[mutation\_method]

label = f"{crossover\_name} + {mutation\_name}"

plt.plot(generations, data,

color=colors[color\_idx], linestyle=line\_styles[color\_idx],

linewidth=2, label=label)

color\_idx += 1

plt.xlabel('Поколение')

plt.ylabel('Приспособленность')

plt.legend()

plt.grid(True, alpha=0.3)

plt.yscale('log')

plt.tight\_layout()

plt.savefig('combined\_comparison.png', dpi=300)

plt.show()

def brute\_force():

if n > 2 or k > 3:

print("Слишком большая задача для полного перебора")

return None, float('inf')

# Генерируем все возможные распределения

from itertools import product

# Определяем диапазоны для каждого элемента

ranges = []

for i in range(n):

for j in range(k):

ranges.append(range(0, supply[i] + 1))

best\_solution = None

best\_fitness = float('inf')

count = 0

for solution in product(\*ranges):

count += 1

fitness = fitness\_function(solution)

if fitness < best\_fitness:

best\_fitness = fitness

best\_solution = solution

print(f"Перебрано {count} решений")

return best\_solution, best\_fitness

def print\_solution(solution):

if solution is None:

print("Решение не найдено")

return

matrix = decode\_individual(solution)

print(f"{'Привезено':<12}", end="")

for j in range(k):

actual\_demand = sum(matrix[i][j] for i in range(n))

print(f"{actual\_demand:<10}", end="")

print()

print(f"{'Требуется':<12}", end="")

for j in range(k):

print(f"{demand[j]:<10}", end="")

print()

total\_cost = 0

for i in range(n):

for j in range(k):

total\_cost += matrix[i][j] \* transport\_costs[i][j]

print(f"\nОбщие транспортные расходы: {total\_cost}")

print(f"Бюджет: {transport\_budget}")

print(f"Превышение бюджета: {max(0, total\_cost - transport\_budget)}")

if \_\_name\_\_ == "\_\_main\_\_":

print(f"Пунктов производства: {n}, Городов: {k}")

print(f"Производственные мощности: {supply}")

print(f"Потребности городов: {demand}")

print(f"Транспортный бюджет: {transport\_budget}")

crossover\_methods = {

'one\_point': 'Одноточечное скрещивание',

'two\_point': 'Двухточечное скрещивание',

'uniform': 'Равномерное скрещивание'

}

mutation\_methods = {

'random\_reallocate': 'Случайное перераспределение',

'swap': 'Обмен значениями',

'gaussian': 'Гауссова мутация'

}

results = {}

all\_methods\_data = {}

for crossover\_key, crossover\_name in crossover\_methods.items():

for mutation\_key, mutation\_name in mutation\_methods.items():

method\_key = f"{crossover\_key}\_{mutation\_key}"

method\_display\_name = f"{crossover\_name} + {mutation\_name}"

print(f"\n" + "=" \* 70)

print(f"КОМБИНАЦИЯ: {method\_display\_name}")

print("=" \* 70)

ga\_solution, ga\_fitness = genetic\_algorithm(

population\_size=300,

generations=300,

mutation\_rate=0.5,

crossover\_method=crossover\_key,

mutation\_method=mutation\_key

)

results[method\_key] = {

'solution': ga\_solution,

'fitness': ga\_fitness,

'name': method\_display\_name

}

# Сохраняем данные для графика

all\_methods\_data[method\_key] = best\_fitness\_history.copy()

print(f"\nЛучшая приспособленность: {ga\_fitness}")

print\_solution(ga\_solution)

plot\_fitness\_progress(all\_methods\_data)

# # Полный перебор

# print("\n" + "=" \* 50)

# print("ПОЛНЫЙ ПЕРЕБОР")

# print("=" \* 50)

#

# bf\_solution, bf\_fitness = brute\_force()

#

# if bf\_solution is not None:

# print(f"\nЛучшая приспособленность перебора: {bf\_fitness}")

# print\_solution(bf\_solution)

#

# # Сравнение результатов

# print("\n" + "=" \* 50)

# print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ")

# print("=" \* 50)

# print(f"Генетический алгоритм: {ga\_fitness}")

# print(f"Полный перебор: {bf\_fitness}")

# print(f"Разница: {ga\_fitness - bf\_fitness}")