

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

**Лабораторная работа №6**

по дисциплине «Основы теории принятия решений»

**Выполнила:**

студентка уч. гр. ВМО31

Волкова Эмилия Юрьевна

**Проверил:**

проф. Кобак В. Г.

Ростов-на-Дону

2025

**Введение**

Предметом области исследования расписаний является круг задач проектирования и организационного управления в различных системах, в которых требуется найти наилучшее (оптимальное) значение выбранных критериев их функционирования с учетом имеющихся ограничений.

Программирование для многопроцессорных машинных систем связано с распараллеливанием и синхронизацией вычислений и организацией выполнения параллельных вычислительных процессов. Это выдвигает целый ряд сложных задач, среди которых весьма важными являются расчет характеристик времени и количества операций, требующихся для выполнения параллельных программ и построения расписаний (планов), выполнения параллельных программ на многопроцессорных и многомашинных вычислительных системах.

Модели параллельных программ и операционные характеристики процессов их выполнения служат основой для планирования параллельных вычислительных процессов, т. е. для построения расписаний указанных процессов. Расписания параллельных вычислительных процессов определяют порядок выполнения программы на вычислительной системе, включая распределение частей программы по процессам. С увеличением числа распределяемых частей программ и количества используемых процессоров сложность построения оптимальных расписаний обычно резко возрастает. Поэтому важное значение имеют простые в построении и удобные в реализации приближенные расписания параллельных вычислительных процессов, близкие к оптимальным расписаниям с точки зрения времени выполнения параллельных программ.

**Постановка задачи**

Имеется вычислительная система (ВС), состоящая из  несвязанных идентичных устройств (приборов, процессоров и т. п.) .

На обслуживание в ВС поступает набор из  независимых параллельных заданий (работ) , известно время решения  задания  на любом из устройств. При этом каждое задание может выполняться на любом из устройств (процессоре), в каждый момент времени отдельный процессор обслуживает не более одного задания и выполнение задания не прерывается для передачи на другой процессор. Требуется найти такое распределение заданий по процессорам, при котором суммарное время выполнения заданий на каждом из процессоров было бы минимальным. Под расписанием следует понимать отображение , такое, что если , то говорят, что задание  в расписании , назначенного на процессор . При сделанных выше допущениях расписание можно представить разбиением множества заданий на  непересекающихся подмножеств , где .

Критерий, используемый для минимизации времени завершения обслуживания заданий, является минимальным критерием и определяется в следующем виде: , где  - время завершения работы процессора .

**Стандартная схема работы генетического алгоритма**

Описанный ниже метод более эффективен по скорости поиска приемлемого по точности решения.

Модифицированный алгоритм построения расписания с произвольной загрузкой:

Дана прямоугольная матрица .

Ш.1 Упорядочим строки матрицы *T* по убыванию сумм всех их элементов.

Ш.2 В преобразованной матрице *T’* первой строке проведем расчёт согласно формуле (1):

Выберем минимальный элемент T. Индекс данного элемента примем за индекс элемент распределения и прибавим элемент строки с данным индексом к соответствующему элементу следующей строки.

Ш.3 Следующая строка теперь учитывает предыдущее решение. Выполним расчёт элементов второй строки по формуле (1), найдем индекс минимального элемента T, прибавим элемент второй строки с данным индексом к соответствующему элементу третьей строки и т. д.

Генетические алгоритмы (ГА) являются одной из парадигм эволюционных вычислений, представляют собой алгоритмы поиска лучшего, а не оптимального решения задачи, построены на принципах, сходных с принципами естественного отбора и генетики.

ГА имеет вероятностную природу и в связи с этим результаты, получаемые с помощью него отличаются в каждом запуске и определяются случайной последовательностью, переданной в схему алгоритма. Точность алгоритма зависит не только от входной последовательности случайных чисел, но и от условий задачи, таких как размерность задачи и конкретное распределение весов.

Рассмотрим общую схему работы генетического алгоритма:

Ш.1 Формируется начальное поколение, состоящее из заданного числа особей.

Ш.2 Отбор особей и применение ГА операторов кроссовера и мутации с заданной вероятностью для создания нового поколения.

Ш.3 Проверка условия останова, которая обычно заключается в неизменности лучшего решения в течение заданного числа поколений. Если проверка прошла не успешно, то переход на Ш.2.

Ш.4 Лучшая особь выбирается как найденное решение.

ГА является общим алгоритмом для решения любой задачи, и при его применении к конкретной необходимо выбрать механизм кодирования параметров задачи (фенотипа) в гены особи (генотипа), определить оптимизационную функцию  (fitness function) и выбрать условия останова.

В данном случае решения задачи теории расписания минимаксный критерий будет являться оптимизационной функцией, а условием останова - неизменность лучшего решения в течение заданного числа поколений.

Введем следующие обозначения:

 - число особей (хромосом);

 - условие останова по количеству одинаковых поколений;

 - вектор особей (хромосом);

 - вектор приспособленности;

 - вероятность оператора кроссовера;

 - оператор кроссовера;

 - вероятность оператора мутации;

 - оператор мутации.

Рассмотрим подробнее Ш.2 алгоритма:

Ш.2.1 На вход шага поступает исходный вектор особей (хромосом) , каждая из особей представляет какое-то расписание с соответствующим ему  и соответственно приспособленностью особи.

Ш.2.2 , где  - порядковый номер особи в векторе .

Ш.2.3 Выполняется оператор кроссовера с заданной вероятностью . Для этого генерируется случайное число  в интервале [0..1]. Если , то , где - случайное число в интервале [1..], исключая совпадение с числом , иначе особь переходит в новое поколение без изменений, т.е. .

Ш.2.4 Выполняется оператор мутации с заданной вероятностью . Для этого генерируется случайное число  в интервале [0..1]. Если , то .

Ш.2.5 . Если , то перейти к Ш.2.3.

Ш.2.6 Сформировано новое поколение . Осуществляется переход на Ш.3.

A diagram of a flowchart

Description automatically generated

Рисунок 1 – Схема работы генетического алгоритма

**Вычислительные эксперименты**

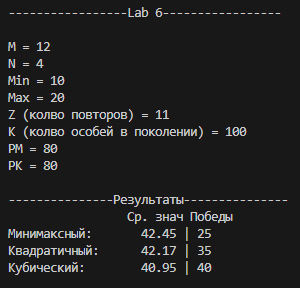
****

Рисунок 2 – Эксперимент №1

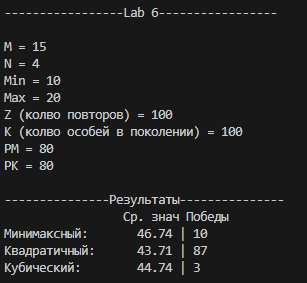


Рисунок 3 – Эксперимент №2

**Листинг программы**

from random import randint, randrange

import numpy as np

def divide\_into\_intervals(number, N):

    interval\_size = number // N

    intervals = []

    start = 1

    for i in range(N):

        end = start + interval\_size - 1

        intervals.append((start, end))

        start = end + 1 if i < (N-1) else end

    intervals[-1] = (intervals[-1][0], 255)

    return intervals

def invert\_nth\_bit(num, n):

    binary\_num = '{:08b}'.format(num)

    inverted\_bit = '0' if binary\_num[n] == '1' else '1'

    new\_binary\_num = binary\_num[:n] + inverted\_bit + binary\_num[n+1:]

    inverted\_int = int(new\_binary\_num, 2)

    return inverted\_int

def Alg(T, N, M, p):

    matrix = []

    for i in T:

        matrix.append(i)

    mins = np.zeros((M, N))

    lead = np.zeros(N)

    for row in range(0, M):

        if row != M-1:

            current\_row = matrix[row]

            Tt = np.power(current\_row, p)

            for i in range(0, len(lead)):

                Tt = Tt + lead[i]\*\*p

                Tt[i] -= lead[i]\*\*p

            ind = np.argmin(Tt)

            elem = current\_row[ind]

            mins[row][ind] = elem

            lead[ind] = elem

            matrix[row+1] = matrix[row+1] + lead

        elif row == M-1:

            current\_row = matrix[row]

            Tt = np.power(current\_row, p)

            for i in range(0, len(lead)):

                Tt = Tt + lead[i]\*\*p

                Tt[i] -= lead[i]\*\*p

            ind = np.argmin(Tt)

            elem = current\_row[ind]

            mins[row][ind] = elem

            lead[ind] = elem

    tasks = np.array([])

    for i in np.transpose(mins):

        non\_zero\_i = i[i != 0]

        if non\_zero\_i.size > 0:

            tasks = np.append(tasks, non\_zero\_i[-1])

    return int(max(tasks))

def Plotnikov\_Zverev(T, N, M):

    matrix = []

    for i in T:

        matrix.append(i)

    load = np.zeros(N)

    path = []

    for row in range(M):

        matrix[row] += load

        tmp\_row = matrix[row].copy()

        e, i = np.min(tmp\_row), np.argmin(tmp\_row)

        load[i] = e

        path.append(i)

    return int(max(load))

def calculate\_adaptation(DNA, p):

    if p == 0:

        return Plotnikov\_Zverev(DNA, N, M)

    elif p == 2:

        return Alg(DNA, N, M, 2)

    elif p == 3:

        return Alg(DNA, N, M, 3)

def MUTATE(DNAp):

    elemind = randint(0, len(DNAp)-1)

    genind = randint(0, len(DNAp[elemind])-1)

    p = randint(0, 7)

    mutated = invert\_nth\_bit(int(DNAp[elemind][genind]), p)

    DNAp[elemind][genind] = mutated

    return DNAp

def CROSSOVER(p1, p2, split):

    global PM

    DNAp = p1[0:split] + p2[split:]

    r = randrange(100)

    if r < PM and PM != 100:

        return DNAp

    else:

        DNAp = MUTATE(DNA)

        return DNAp

def main(gen, p):

    GENERATIONS, BEST = {0 : gen}, min([specie[1] for specie in gen])

    COUNTER = 1

    dictid = 0

    while COUNTER != Z:

        nextgen = []

        for i in range(len(gen)):

            r = randrange(100)

            if r < PK and PK != 100:

                nextgen.append(gen[i])

            else:

                partner\_id = i

                while partner\_id == i:

                    partner\_id = randint(0, (len(gen)-1))

                split = randint(0, len(T)-1)

                Pdna1 = CROSSOVER(gen[i][0], gen[partner\_id][0], split)

                Pdna2 = CROSSOVER(gen[partner\_id][0], gen[i][0], split)

                P1 = (Pdna1, calculate\_adaptation(Pdna1, p))

                P2 = (Pdna2, calculate\_adaptation(Pdna2, p))

                contest = [gen[i][1], P1[1], P2[1]]

                win = min(contest)

                win\_id = contest.index(win)

                if win\_id == 0:

                    nextgen.append(gen[i])

                elif win\_id == 1:

                    nextgen.append(P1)

                elif win\_id == 2:

                    nextgen.append(P2)

        newBEST = min([specie[1] for specie in nextgen])

        if newBEST == BEST:

            COUNTER += 1

        elif newBEST < BEST:

            BEST = newBEST

            COUNTER = 1

        dictid += 1

        GENERATIONS[dictid] = nextgen

        gen = nextgen

    return COUNTER, BEST, GENERATIONS

print("\n-----------------Lab 6-----------------\n")

M = int(input("M = "))

N = int(input("N = "))

min\_val = int(input("Min = "))

max\_val = int(input("Max = "))

Z = int(input("Z (колво повторов) = "))

K = int(input("K (колво особей в поколении) = "))

PM = int(input("PM = "))

PK = int(input("PK = "))

'''M = 12

N = 4

min\_val = 10

max\_val = 20

Z = 100

K = 100

PM = 80

PK = 80'''

T = [[randint(min\_val, max\_val) for \_ in range(N)] for \_ in range(M)]

INTERVALS = divide\_into\_intervals(255, N)

gen\_minmax, gen\_square, gen\_cube = [], [], []

all\_minmax, all\_square, all\_cube = [], [], []

win\_minmax, win\_square, win\_cube = 0, 0, 0

for i in range(K):

    DNA = []

    for j in T:

        DNA.append([randint(min\_val, max\_val) for \_ in range(N)])

    gen\_minmax.append((DNA, calculate\_adaptation(DNA, 0)))

    gen\_square.append((DNA, calculate\_adaptation(DNA, 2)))

    gen\_cube.append((DNA,   calculate\_adaptation(DNA, 3)))

counts\_minmax, best\_minmax, gens\_minmax = main(gen\_minmax, 0)

counts\_square, best\_square, gens\_square = main(gen\_square, 2)

counts\_cube, best\_cube, gens\_cube = main(gen\_cube, 3)

with open("lab6\\minimax.txt", "w", encoding='utf-8') as file:

    file.write("===============Минимакс===============\n")

    file.write(f"ЛУЧШАЯ ПРИСПОСОБЛЯЕМОСТЬ: {best\_minmax}\n")

    file.write(f"КОЛИЧЕСТВО ПОВТОРОВ: {counts\_minmax}\n")

    file.write(f"КОЛИЧЕСТВО ПОЛУЧЕННЫХ ПОКОЛЕНИЙ: {len(list(gens\_minmax.keys()))}\n")

    for key in gens\_minmax:

        generation\_as\_lists\_minmax = [(np.array(dna[0]).tolist(), dna[1]) for dna in gens\_minmax[key]]

        file.write(f"Поколение {key}: \n")

        for i in generation\_as\_lists\_minmax:

            all\_minmax.append(i[1])

            for j in i[0]:

                if i[0][len(i[0])-1] == j:

                    file.write(f"{j} ")

                else:

                    file.write(f"{j}\n")

            file.write(f"{i[1]}\n")

            file.write('-----------------\n')

    file.write("========================================\n")

with open("lab6\\square.txt", "w", encoding='utf-8') as file:

    file.write("==============Квадратичный==============\n")

    file.write(f"ЛУЧШАЯ ПРИСПОСОБЛЯЕМОСТЬ: {best\_square}\n")

    file.write(f"КОЛИЧЕСТВО ПОВТОРОВ: {counts\_square}\n")

    file.write(f"КОЛИЧЕСТВО ПОЛУЧЕННЫХ ПОКОЛЕНИЙ: {len(list(gens\_square.keys()))}\n")

    for key in gens\_square:

        generation\_as\_lists\_square = [(np.array(dna[0]).tolist(), dna[1]) for dna in gens\_square[key]]

        file.write(f"Поколение {key}: \n")

        for i in generation\_as\_lists\_square:

            all\_square.append(i[1])

            for j in i[0]:

                if i[0][len(i[0])-1] == j:

                    file.write(f"{j} ")

                else:

                    file.write(f"{j}\n")

            file.write(f"{i[1]}\n")

            file.write('-----------------\n')

    file.write("============================================\n")

with open("lab6\\cube.txt", "w", encoding='utf-8') as file:

    file.write("================Кубический================\n")

    file.write(f"ЛУЧШАЯ ПРИСПОСОБЛЯЕМОСТЬ: {best\_cube}\n")

    file.write(f"КОЛИЧЕСТВО ПОВТОРОВ: {counts\_cube}\n")

    file.write(f"КОЛИЧЕСТВО ПОЛУЧЕННЫХ ПОКОЛЕНИЙ: {len(list(gens\_cube.keys()))}\n")

    for key in gens\_cube:

        generation\_as\_lists\_cube = [(np.array(dna[0]).tolist(), dna[1]) for dna in gens\_cube[key]]

        file.write(f"Поколение {key}: \n")

        for i in generation\_as\_lists\_cube:

            all\_cube.append(i[1])

            for j in i[0]:

                if i[0][len(i[0])-1] == j:

                    file.write(f"{j} ")

                else:

                    file.write(f"{j}\n")

            file.write(f"{i[1]}\n")

            file.write('-----------------\n')

    file.write("============================================\n")

for i in range(K):

    min\_val\_res = min([generation\_as\_lists\_minmax[i][1], generation\_as\_lists\_square[i][1], generation\_as\_lists\_cube[i][1]])

    if min\_val\_res == generation\_as\_lists\_minmax[i][1]:

        win\_minmax += 1

    elif min\_val\_res == generation\_as\_lists\_square[i][1]:

        win\_square += 1

    elif min\_val\_res == generation\_as\_lists\_cube[i][1]:

        win\_cube += 1

for i in [all\_minmax, all\_square, all\_cube]:

    i.append(sum(i)/len(i))

print("\n---------------Результаты---------------")

print(f"{'':<17}{'Ср. знач':<5} {'Победы':<5}")

print(f"{'Минимаксный:':<15} {round(all\_minmax[-1], 2):>8} | {win\_minmax:<5}")

print(f"{'Квадратичный:':<15} {round(all\_square[-1], 2):>8} | {win\_square:<5}")

print(f"{'Кубический:':<15} {round(all\_cube[-1], 2):>8} | {win\_cube:<5}")

**Вывод**

Проанализировав данные, полученные входе экспериментов, можно сделать вывод, что модификация алгоритма построения расписания с произвольной загрузкой более эффективна по скорости поиска подходящего по точности решения, чем классическая версия алгоритма.

Литература

1. Коффман Э.Г. Теория расписания и вычислительные машины. – M.: Наука, 1987.