

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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 №5 по дисциплине:**

**«Эвристические методы»**

Выполнил студент группы ВПР31

Андросов Илья Алексеевич

Проверил:

профессор Кобак Валерий Григорьевич

г. Ростов-на-Дону

2023г

1. **Общие сведения**

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.

1. **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 .

1. **Генетический алгоритм**

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 .

1. **Схема алгоритма**

Далее на рисунке изображена схема генетического алгоритма, используемого в данной лабораторной

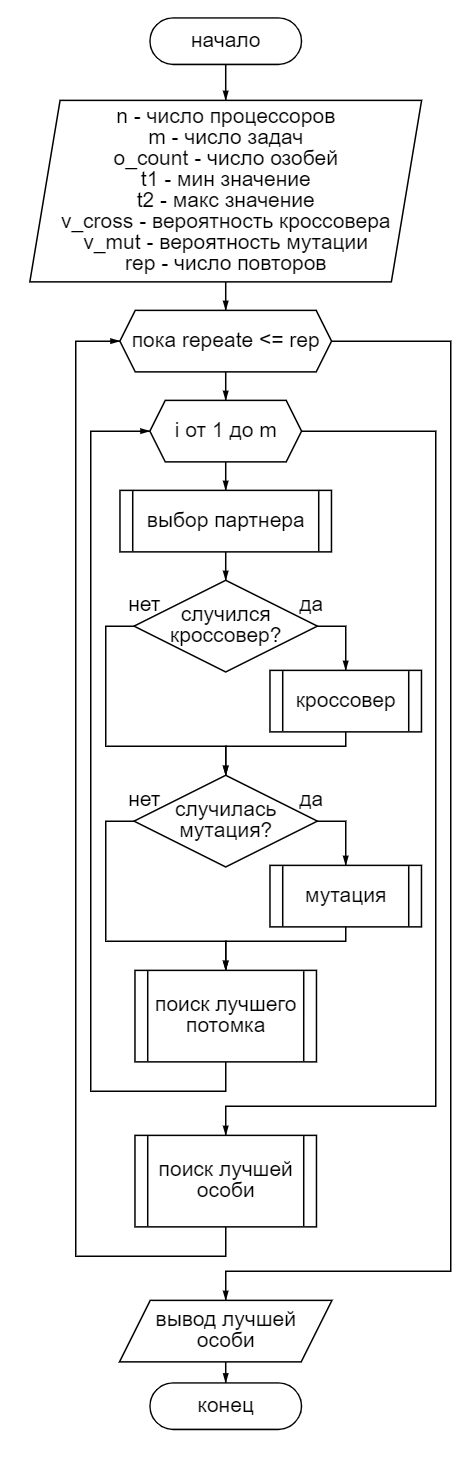
****

Рисунок 4.1 Блок схема алгоритма

1. **Исходный код программы**

import random

n = int(input('Число процессоров '))

m = int(input('Число задач '))

t1 = int(input('Мин процесса '))

t2 = int(input('Макс процесса '))

osob = int(input('Число особей '))

finalCount = int(input('Количество повторений '))

verCross = int(input('Вероятность кросовера '))

verMut = float(input('Вероятность мутации '))

f = open('./result.txt', 'w')

def raspred\_mass(raspred\_value, individual, n):

zero\_mass = [0] \* n

for i in individual:

zero\_mass[int(i[1] / raspred\_value)] += i[0]

return zero\_mass

def fitness(individual, raspred\_value, n):

return max(raspred\_mass(raspred\_value, individual, n))

def selection(population):

return random.sample(population, 2)

def mutation(individual, chanse):

ver = random.randint(0, 100)

f.write('\n Вер мутации: ' + str(ver))

if(ver <= chanse):

numIndex = random.randint(0, len(individual) - 1)

number = individual[numIndex][1]

binary\_number = bin(number)[2:].zfill(8)

random\_index = random.randint(0, 7)

binary\_number = binary\_number[:random\_index] + str(int(binary\_number[random\_index]) ^ 1) + binary\_number[random\_index+1:]

number = int(binary\_number, 2)

individual[numIndex][1] = number

f.write( '\n' + '\t'.join([str(i[0]) + ' ' + str(i[1]) for i in individual]) + '\n')

return individual

def crossover(parent1, parent2, chanse, ver\_result, crossover\_point):

if(ver\_result <= chanse):

child = parent1[:crossover\_point] + parent2[crossover\_point:]

f.write('\t'.join([str(i[0]) + ' ' + str(i[1]) for i in child]) + '\n')

return child

return parent1

def genetic\_algorithm(n):

raspred\_value = int(256/n)

counter = 0

pupul\_count = 0

lastMin = 0

initMass = [random.randint(t1, t2) for \_ in range(m)]

f.write("Вес задач: " + ' '.join([str(i) for i in initMass]))

population = [[[j, random.randint(0, 255)] for j in initMass] for i in range(osob)]

fitness\_scores = []

while(True):

pupul\_count += 1

fitness\_scores = [ fitness(individual, raspred\_value=raspred\_value, n=n) for individual in population ]

f.write("\n Популяция " + str(pupul\_count) + ":\n" + '\n'.join(['\t'.join([ ' '.join([str(k) for k in population[i][j]]) for j in range(len(population[i]))]) for i in range(len(population))]))

f.write('\n Вес популяции:\n' + ' '.join([ str(k) for k in fitness\_scores]))

min\_fitness\_score = min(fitness\_scores)

if lastMin == 0 or min\_fitness\_score < lastMin:

counter = 0

lastMin = min\_fitness\_score

else:

counter += 1

if(counter == finalCount - 1):

break

parent1, parent2 = selection(population)

parant\_index = population.index(parent1)

parant\_index2 = population.index(parent2)

f.write('\nВыбор родителей: ' + str(parant\_index+1) + ' ' + str(parant\_index2+1))

ver\_result = random.randint(0, 100)

f.write('\nВер кросовера:\n' + str(ver\_result))

crossover\_point = random.randint(0, len(parent1) - 1)

f.write('\nКросовер в: \n' + str(crossover\_point))

f.write('\nРезультат кросовера:\n')

child1 = crossover(parent1, parent2, verCross, ver\_result, crossover\_point)

child2 = crossover(parent2, parent1, verCross, ver\_result, crossover\_point)

f.write('\nВер Мутации:\n')

child1 = mutation(child1, verMut)

child\_fitness1 = fitness(child1, raspred\_value=raspred\_value, n=n)

child\_fitness2 = fitness(child2, raspred\_value=raspred\_value, n=n)

f.write(' Дети:\n' + str(child\_fitness1) + ' ' + str(child\_fitness2))

pic = 0

pic\_child = []

if(child\_fitness1 >= child\_fitness2):

pic = child\_fitness2

pic\_child = child2

else:

pic = child\_fitness1

pic\_child = child1

if(fitness(parent1, raspred\_value=raspred\_value, n=n) > pic):

population[parant\_index] = pic\_child

best\_individual = min(fitness\_scores)

return best\_individual

print(genetic\_algorithm(n))

f.close()

1. **Литература**
2. Нейронные сети, генетические алгоритмы и нечеткие системы – Рутковская Д., Пилиньский М., Рутковский Л
3. Генетические алгоритмы – Т.В. Панченко
4. Применение генетических алгоритмов к решению задач дискретной оптимизации – Д.И. Батищев, Е.А. Неймарк, Н.В. Старостин
5. Генетические алгоритмы. Учебное пособие – Л.А. Гладков, В.В. Курейчик, В.М. Курейчик
6. Генетические алгоритмы и оптимизация. Учебное пособие – А.В. Еремеев