

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**

**ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«Донской государственный технический университет»**

**(ДГТУ)**

Кафедра «Программное обеспечение вычислительной техники и

автоматизированных систем»

**ЛАБОРАТОРНАЯ РАБОТА № 7**

по дисциплине «Эвристические методы и алгоритмы»

Выполнил:

ст. гр. ВПР 31 Д. С. Кононов

Проверил:

д.н., профессор В. Г. Кобак

Ростов-на-Дону

2020

1. **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 с точки зрения времени выполнения параллельных программ.

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.

# 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 .

# Алгоритм работы генетического алгоритма

Шаг 1: Формируется начальное поколение, состоящее из заданного числа особей.

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

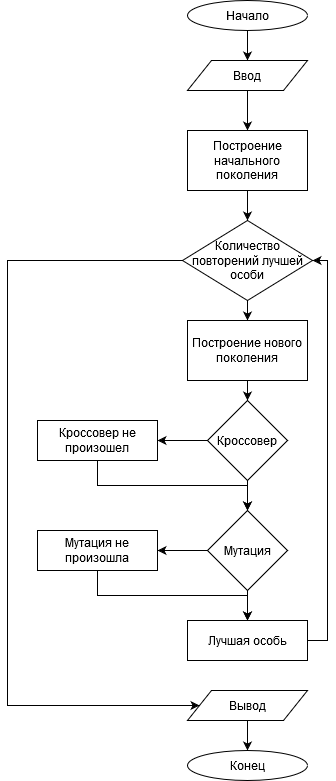
Шаг 3: Проверка условия останова, которая обычно заключается в неизменности лучшего решения в течение заданного числа поколений. Если проверка прошла не успешно, то переход на Ш.2.

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

**4. Блок-схема алгоритма работы генетического алгоритма**



**5. Реализации генетического алгоритма на языке Python**

from random import randint, choice

class HDA:

def krit(self, mass, n):

p = [[] for i in range(n)]

for i in mass:

sums = list(map(sum, p))

p[sums.index(min(sums))].append(i)

return p

def HDA(self, mass, proc):

Pa, Pb = self.krit(mass, 2)

Pa = self.krit(Pa, proc // 2)

Pb = self.krit(Pb, proc // 2)

P = Pa + Pb

sums = list(map(sum, P))

return sums

def main(self, mass, proc):

return max(self.HDA(list(reversed(sorted(mass))), proc))

class GA:

def \_\_init\_\_(self, Osob=10, Gen=10, Pcross=0.8, Pmut=0.2, Lim=3, Left=1, Right=100, Nproc=4):

self.C = [[randint(Left, Right) for i in range(Gen)] for i in range(Osob)]

self.Pcross = Pcross

self.Pmut = Pmut

self.Lim = Lim

self.Left = Left

self.Right = Right

self.Nproc = Nproc

def crossover(self, left, right):

if randint(0, 100) / 100 <= self.Pcross:

new\_left = []

new\_right = []

for i in range(5):

new\_left.append(left[i])

new\_right.append(right[i])

for i in range(5, 10):

new\_left.append(right[i])

new\_right.append(left[i])

return new\_left, new\_right

def mut(self, mass):

for i in mass:

if randint(0, 100) / 100 <= self.Pmut:

a = i.copy()

first = choice(a)

del a[a.index(first)]

second = choice(a)

i[i.index(first)] = randint(self.Left, first)

i[i.index(second)] = randint(self.Left, second)

def check\_end(self, T):

for i in range(len(self.C)):

if HDA().main(self.C[i], self.Nproc) != T[i]:

return False

else:

return True

def create\_new\_couples(self):

new\_C = self.C.copy()

new\_couples = []

for i in range(5):

first = choice(new\_C)

del new\_C[new\_C.index(first)]

second = choice(new\_C)

del new\_C[new\_C.index(second)]

new\_couples.append([first, second])

return new\_couples

def find\_top\_individuals(self, T):

tmp\_T = {}

for i in range(len(T)):

tmp\_T[T[i]] = i

rating = [tmp\_T[min(tmp\_T)]]

del tmp\_T[min(tmp\_T)]

while tmp\_T:

rating.append(tmp\_T[min(tmp\_T)])

del tmp\_T[min(tmp\_T)]

return rating[:10]

def default(self):

new\_couples = self.create\_new\_couples()

new\_C = self.C.copy()

self.mut(new\_C)

for i in new\_couples:

new\_couple = self.crossover(\*i)

if new\_couple:

new\_C.append(new\_couple[0])

new\_C.append(new\_couple[1])

print('All genes at the generation')

self.output\_mass(new\_C)

T = [HDA().main(i, self.Nproc) for i in new\_C]

rating = self.find\_top\_individuals(T)

for i in range(len(rating)):

self.C[i] = new\_C[rating[i]]

def main(self):

count = 0

repeats = 0

print('Generation quality at the start:', min([HDA().main(i, self.Nproc) for i in self.C]))

print()

while repeats < self.Lim:

count += 1

T = [HDA().main(i, self.Nproc) for i in self.C]

self.default()

print('Top 10 genes at the generation: ')

self.output\_mass()

if self.check\_end(T):

repeats += 1

else:

repeats = 0

self.output\_mass()

print('Generation quality at the end of GA:', min([HDA().main(i, self.Nproc) for i in self.C]))

print('All generations:', count)

print()

def output\_mass(self, mass=None):

#print('-'\*25)

if not mass:

mass = self.C

for i in range(len(mass)):

print('Gene vector №{} = '.format(i + 1), mass[i])

print('-'\*25)

def main():

a = GA(Lim=3)

print('Primary generation:')

a.output\_mass()

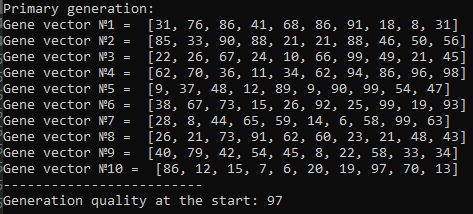
a.main()

if \_\_name\_\_ == '\_\_main\_\_':

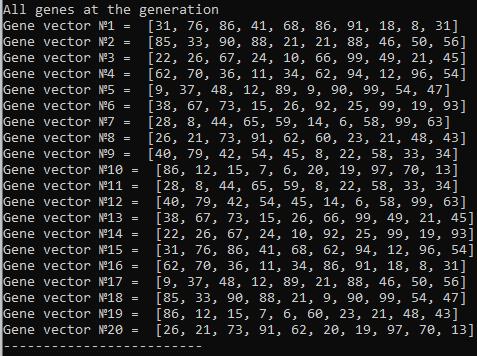
main()

Результат работы программы, выводы:

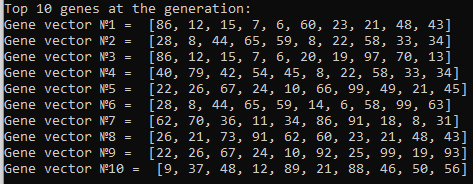
Начальное поколение имеет вид:



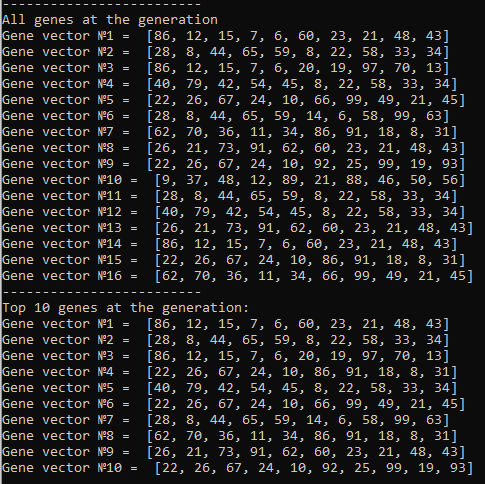
В результате первого шага произошли кроссоверы и мутации:



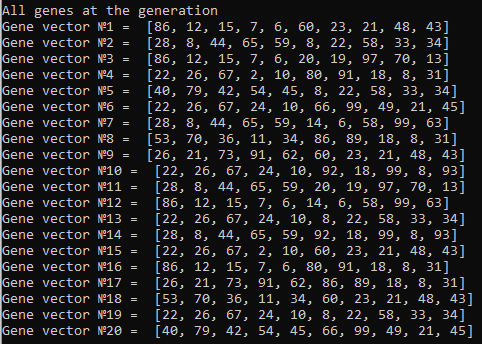
Из 20 полученных особей мы отбираем 10 лучших:

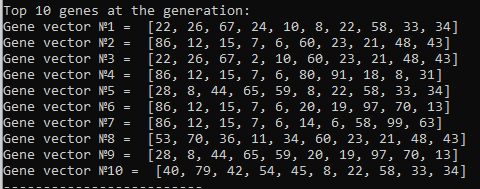


Второй шаг:



Третий шаг:





И так далее, пока не произойдёт необходимое количество повторений для остановки цикла

В результате произошло 73 итерации с итоговым поколением:

