МИНИСТЕРСТВО НАУКИ и высшего ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Центр прикладных информационных технологий

Выпускная квалификационная работа бакалавра

на тему:

«Разработка и реализация процедур ЭГА для решения задачи упорядочения»

Выполнила:

Студентка группы 381607-2

Дукова Е.Е.

Научный руководитель:

доцент кафедры ИАНИ, к.т.н

 Кумагина Е.А.

Нижний Новгород

2020

**Оглавление**

[Введение 3](#_Toc42772047)

[1. Задачи теории расписаний 4](#_Toc42772048)

[2. Постановка задачи 6](#_Toc42772049)

[2.1. Математическая модель 6](#_Toc42772050)

[2.2. Сложность задачи 6](#_Toc42772051)

[3. Подходы к решению задачи 8](#_Toc42772052)

[3.1. Описание метода полного перебора 8](#_Toc42772053)

[3.2. Описание жадного алгоритма 8](#_Toc42772054)

[3.3. Описание генетического алгоритма 8](#_Toc42772055)

[3.3.1. Основной цикл ЭГА 8](#_Toc42772056)

[3.3.2. Выбор кодировки 9](#_Toc42772057)

[3.3.3. Формирование начальной популяции 9](#_Toc42772058)

[3.3.4. Воспроизводство 10](#_Toc42772059)

[3.3.5. Отбор 10](#_Toc42772060)

[3.3.6. Условие остановки 10](#_Toc42772061)

[3.3.7. Определение параметров ЭГА 10](#_Toc42772062)

[3.4. Вычислительный эксперимент 11](#_Toc42772063)

[4. Программная система 12](#_Toc42772064)

[Заключение 13](#_Toc42772065)

[Список литературы 14](#_Toc42772066)

[Приложение 1. Код модулей ЭГА 15](#_Toc42772067)

[Приложение 2. Код модулей жадного алгоритма 20](#_Toc42772068)

# Введение

Задачи упорядочения появляются везде, где есть проблема выбора очередности выполнения работ. В реальных системах число работ очень велико и перебрать все возможные варианты решений невозможно. Существует класс методов – эвристические методы. Они позволяют найти решение задачи (пусть не оптимальное) за приемлемое время.

Генетические алгоритмы возникли в результате наблюдения и попыток копирования естественных процессов, происходящих в мире живых организмов, в частности, эволюции, связанной с ней селекции (естественного отбора) популяции живых существ. Эволюция – это процесс постоянной оптимизации биологических видов.

Рассматриваемая задача – это задача определения оптимального порядка выполнения работ на одном приборе при неодновременном поступлении.

Метод решения ‑ с помощью эволюционно-генетического алгоритма.

Рассматриваемая задача упорядочения принадлежит классу задач дискретной оптимизации. Такие задачи так же называются задачами комбинаторной оптимизации или задачами переборного типа, так как область допустимых решений дискретна, то есть состоит из конечного (счетного) числа допустимых решений.

Выпускная квалификационная работа состоит из 4 глав, введения, заключения, списка литературы и приложения. Первая глава содержит основные понятия теории расписаний и классификацию задач. Во второй главе приводится математическая модель, постановка и оценка сложности задачи. В третьей главе описаны методы решения задачи, эвристический алгоритм и приведены результаты вычислительного эксперимента, показавшего эффективность предложенного подхода. Четвёртая глава содержит описание интерфейсной части и программной реализации разработанных алгоритмов. В приложении приведен исходный код программ.

# Задачи теории расписаний

Теория расписаний*–* это научная дисциплина, посвящённая методам оптимизации календарного планирования. В таких задачах ставиться основная цель – дискретной оптимизации расписания и минимизация времени выполнения таких работ, их стоимость или т. п.

Задачи теории расписаний разделяются на две группы: задачи упорядочения и задачи распределения.

В задачах упорядочения необходимо указать только порядок выполнения работ на приборах. Такие расписания называют перестановочными. Классическими задачами упорядочения являются задача с переналадками, задача мастера.

Задача с переналадками формулируется следующим образом: n требований обслуживаются 1 прибором. Все требования поступают в момент времени d=0. Длительности обслуживания требований одинаковые. Известны времена начальной и конечной наладок, времена переналадок. Найти такую последовательность выполнения работ, чтобы суммарное время пусконаладочных работ было минимальным.

Задача мастера формулируется следующим образом: имеется несколько неисправных станков. У мастера в наличии только одна ремонтная бригада. Соответственно, требуется определить порядок ремонта неисправных станков, учитывая потери от их простоя (потери зависят от производительности), чтобы суммарные итоговые потери оказались минимальны.

В задачах распределения кроме порядка выполнения работ надо определить еще и место выполнения работы. Это характерно для систем, в которых не один прибор, а несколько.

Характерной особенностью одноприборных задач теории расписаний является то, что система обслуживания представлена одним прибором, который может обслуживать (выполнять) только одно требование (задание). Несмотря на кажущуюся простоту в определении, решение одноприборных задач представляется трудным, и является важной вычислительной проблемой [4-8].

По дисциплине выполнения работ на машинах можно выделить четыре основные класса задач:

Open shop, открытая линия ‑ для каждого требования задано своё подмножество машин, на каждой из которых оно должно обслуживаться некоторое время. Порядок обслуживания на этих машинах произвольный. Задаются разнообразные целевые функции.

Job shop, рабочий цех ‑ для каждого требования задано своё упорядоченное подмножество машин (маршрут), на которых оно должно обслуживаться в заданном порядке. Задаются разнообразные целевые функции.

Flow shop, потоковая линия ‑ все машины упорядочены — {M1,M2,… ,Mm} и каждое требование проходит все машины в этом порядке. Расписание задано перестановкой требований. Как правило, минимизируется общее время обслуживания требований.

Задача с директивными сроками — для каждого требования задан момент поступления, время обслуживания и директивный срок окончания обслуживания. Порядок обслуживания на приборах произвольный. Необходимо найти расписание, соблюдающее директивные сроки.

Задача с директивными сроками называется одностадийной, а остальные три – многостадийными. Для задачи Flow shop существует алгоритм Джонсона[7]. Для задачи с директивными сроками с произвольным числом приборов и прерываниями обслуживания существует алгоритм временной сложности, который строит расписание соблюдающее директивные сроки [8]. Решением задач Open shop и Job shop с одним прибором без прерываний является некоторая перестановка требований и для произвольной целевой функции эти задачи NP-полны. Но для некоторых конкретных целевых функций найдены полиномиальные алгоритмы[5-8].

Задача определения оптимального порядка выполнения работ на одном приборе при неодновременном поступлении

Пусть требования множества *N* = *{1, 2, ..., n}* обслуживается одним прибором. Каждое требование *i, i*∈*N* характеризуется моментом поступления *di*, длительностью обслуживания *ti*, директивным сроком *Di*. В каждый момент времени прибор обслуживает не более одного требования. Порядок обслуживания требований произвольный. Прерывания в обслуживании каждого отдельного требования не допускаются. Требуется построить расписание (определить порядок выполнения работ), в котором минимизируется суммарное число тактов нарушений директивных сроков.

# Постановка задачи

## Математическая модель

Исходные параметры:

*N* = *{1, 2, ..., n}* – множество работ, подлежащих выполнению;

*= (d1, d2,…, dn)* – вектор моментов поступлений работ;

*= (D1, D2,…, Dn)* – вектор директивных сроков;

*= (t1, t2,…, tn)* – вектор длительности выполнения работ;

Варьируемые параметры:

= (x1, x2, …, xn) – вектор моментов начала выполнения;

= (y1, y2, …, yn) – вектор моментов окончания выполнения;

Ограничения:

1. *yi = xi + ti, i ∈ N* – выполнение работы на машине производится без прерываний;

2. *xj ≥ yi + ti, j > i, i, j ∈ N* – на машине одновременно может выполняться только одна работа;

3. *xi ≥ di, i ∈ N* – выполнение работы не может начаться раньше начального срока;

Постановка задачи:

Задача нахождения оптимального расписания будет заключаться в определении среди множества допустимых решений (выполняются ограничения (1)-(3)) такого, которое минимизирует суммарное число тактов нарушений директивных сроков:

## Сложность задачи

NP– класс всех задач поиска (для любой задачи из NP проверка кандидата S на соответствие ограничениям J ‑ проверка задачи распознавания ‑ выполняется за полиномиальное от |J| время).NP – полная задача, если она принадлежит NP и любая задача из NP может быть сведена к ней за полиномиальное время. Для доказательства NP-трудности задачи используется прием полиномиального сведения к ней заведомо NP-трудной задачи. Примером задач из класса NP может быть экспоненциальный алгоритмом (перебор всех потенциальных решений) – количество кандидатов полиноминально зависит от входа.

Задача упорядочения с неодновременным поступлением и директивными сроками [ТР 1] NP-полная в сильном смысле. Она разрешима за псевдополиномиальное время (*O*(*nA*)), если число значений, принимаемых величинами di и Di*,* ограниченно константой. И остается NP-полной в обычном смысле, даже когда эта константа равна 2.

В тех случаях, когда все длительности равны 1, либо разрешены прерывания, либо все моменты поступления заданий равны 0, задача может быть решена за полиномиальное время (*O*()), при этом в условии задачи могут присутствовать ограничения предшествования. Задача решается за полиномиальное время также и тогда, когда моменты поступления и директивные сроки являются произвольными рациональными числами и, кроме то, имеются ограничения предшествования. Для этого требуется, чтобы все задания имели одинаковые длительности, либо чтобы разрешались прерывания. К этой задаче сводится NP-полная задача 3-РАЗБИЕНИЕ [2].

# Подходы к решению задачи

Для задач упорядочения существует множество методов решений, как точные, так и эвристические. В данной работе рассмотрены точный метод полного перебора и генетический алгоритм, который относится к классу эвристических алгоритмов.

## Описание метода полного перебора

Требуется построить множество всех перестановок n-элементного множества. Количество подобных перестановок равно n!

Предполагается, что элементы множества запоминаются в виде массива  
P [1], …, P[N]. Элементарная операция, которая применяется к массиву P – поэлементная транспозиция, т.е. обмен значениями переменных P[i] и P[j], где 1≤ i, j ≤ N. На множестве всех последовательностей с элементами X = {1, …, N} – определяется лексический порядок [3]:

## Описание жадного алгоритма

Жадный метод ‑ метод, состоящий в принятии локально наилучших решений на любом шаге, предполагая, что окончательное разрешение также окажется наилучшим. Общеизвестно, что в случае если состав проблемы задается классификацией подмножеств некоторого множества, в то время использование жадного метода получим глобальный оптимум [7].

При применении жадного алгоритма рассчитается коэффициент "важности" требования (приоритет) и затем требования упорядочиваются по не убыванию приоритетов. Сложность этого алгоритма соответствует сложности алгоритма сортировки O(n2) или O(n logn).

Способы формирования коэффициентов приоритетов работ

|  |  |
| --- | --- |
| Способ 1 | (Di - di)/ti |
| Способ 2 | Di-di |
| Способ 3 | (Di - di)/di |
| Способ 4 | Di/(di\*ti) |

## Описание генетического алгоритма

Генетический алгоритм состоит в том, чтобы, используя механизмы, которые есть в живой природе, произвести поиск на множестве перестановок [1].

### Основной цикл ЭГА

Начальная популяция

Воспроизводство

Отбор

Условие остановки

выход

мутация

Популяция – совокупность особей, множество работ, подлежащих выполнению, т.е. множество перестановок на каждом шаге генетического алгоритма;

Особь – перестановка номеров работ, включающая в себя генотип и фенотип;

Генотип – кодировка, используется n-арное представление;

Фенотип – приспособленность (значение критерия F на расписании, построенном по перестановке);

Ген – позиция в генотипе, одна работа;

Наследственность – кроссовер, выбор части кодировки от двух родительских кодировок;

Естественный отбор – оператор селекции, при котором, происходит отбор особей с наилучшими приспособленностями;

Поколение – множества решений, кодировки-родители и кодировки-дети, а также их приспособленности.

Мутация – обмен 2 генов, стоящих на случайных позициях.

### Выбор кодировки

Каждое решение кодируется как перестановка N, называемый хромосомой. Его элементы – гены, изменяющиеся в определённых позициях, называемых аллелями. Геном ‑ совокупность всех хромосом.

Используется n-арное представление – перестановка номеров работ множества N.

### Формирование начальной популяции

Популяция ‑ совокупность решений на конкретной итерации, количество особей в начальной популяции задаётся изначально и в процессе поиска решения обычно не изменяется.

Я рассматривала 2 способа формирования начальной популяции:

1. Половина случайного формирования начальной популяции без повторения особей с помощью перестановки по номеру
2. Половина начальной популяции формируется случайно, а вторая половина формируется с использованием жадного алгоритма.

### Воспроизводство

Воспроизводство – стратегия выбора родительской пары и формирования новой кодировки (потомков). В данном случае используется случайна стратегия выбора родительской пары из популяции. Панмиксия – равновероятный выбор родителей из популяции [1].

Кроссовер моделирует перемешивание генов при половом скрещивании. Таким образом потомок должен нести часть генетической информации из генотипа первого родителя и часть из генотипа второго. При кроссовере не образуются новые аллельные формы, они только копируются из соответствующих частей генотипов родителей [1]. Используется одноточечный кроссовер, где точка разрыва num, .

### Отбор

Отбор моделирует процесс создания родительского генофонда, которое будет использоваться для формирования следующего поколения.

Параллельно с любой стратегией может использоваться элитарная стратегия селекции. Данная схема предотвращает потерю лучшей особи при переходе в следующее поколение. Особь-победитель копируется в следующее поколение принудительно, не смотря на выбранные схемы формирования следующего поколения. Данная особь победитель копируется в текстовый файл на каждой итерации.

Выбор лучших особей для дальнейшего поколения я осуществляю как копирование половины популяции с наилучшими фенотипами записанных по возрастанию. Алгоритм копирования половины популяции представляет собой механизм реализации схемы селекции. Данные алгоритм в большей мере относятся к пропорциональной схеме, но могут использоваться и для любой другой схемы, где вероятность копирования (ожидаемое число копий особи) может быть выражена в явном виде. Выбор особей для копирования происходит из репродукционного множества , примем .

### Условие остановки

Остановка работы ЭГА в моей реализации происходит через число поколений T=100.

### Определение параметров ЭГА

ЭГА запускался при следующих параметрах: число работ N=7, количество особей в начальной популяции M=10, число поколений T=100.

## Вычислительный эксперимент

Для проверки эффективности предложенных подходов к решению задачи был проведен вычислительный эксперимент.

Для этого был программа реализован генератор задач, задающий данные для задач числом работ равным 10 со случайными значениями времен выполнения из диапазона [0,10]. Каждая задача решалась двумя предложенными алгоритмами, а именно: генетическим алгоритмом и методом полного перебора для оценки точности решения.

Для приближенного решения F определялось относительное отклонение от оптимума (F–F\*)/F\*.

В таблице приведено среднее значение отклонение от оптимального значения критерия.

|  |  |
| --- | --- |
| Параметры ЭГА | Лучшее отклонение приближенных решений, % |
| Случайная начальная популяция | 4,929577 |
| Улучшенная начальная популяция | 4,225352 |

Вывод:

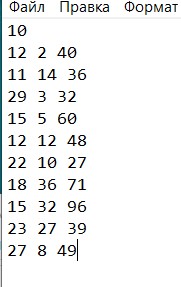
Проведенный эксперимент показал высокую скорость и приемлемое отклонение приближенных решений от оптимальных.

Предложенный подход применим к решению любых задач упорядочения и дает возможность находить приближенное решение.

# Программная система

Система реализована на языке С++ в среде разработки Visual Studio2015. Код приведен в Приложении 1.

Структура файла данных следующая:



Первая строка – количество требований. Далее в строках располагаются длительность, момент поступления, директивный срок работ.

Система позволяет:

1. Сгенерировать исходные данные задач, значения которых мы оптимизируем.

2. Настраивать параметры генетического алгоритма.

3. Получать точное решение методом полного перебора.

4. Получать приближенное решение ЭГА.

5. Записывать результаты на каждом шаге генетического алгоритма в отдельный текстовый файл.

Результат выводится на консоли, а также в виде текстового файла.

# Заключение

В выпускной квалификационной работе рассмотрена задача определения оптимального порядка выполнения работ на одном приборе при неодновременном поступлении.

Построена математическая модель одностадийной системы и поставлена оптимизационная задача поиска расписания, в котором минимизируется суммарное число тактов нарушений директивных сроков.

Для перебора перестановок были реализованы точный метод полного перебора и алгоритм класса эволюционно-генетических алгоритмов.

Для проверки эффективности разработанного ЭГА проведен вычислительный эксперимент, показавший приемлемое отклонение найденных решений от оптимальных.

# Список литературы

1. Д.И. Батищев., Применение генетических алгоритмов к решению задач дискретной оптимизации / Е.А. Неймарк, Н.В. Старостин - Н. Новгород: Изд-во ННГУ, 2006
2. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон – М.: Мир, 1982. – 419 с.
3. Липский В. Комбинаторика для программистов. – М.: Мир, 1988.
4. Танаев В.С., Шкурба В.В. Введение в теорию расписаний. – М.: Наука, 1975.
5. Коффман Э.Г. Теория расписаний и вычислительные машины.– М.: Наука, 1984.
6. Конвей Р.В., Максвелл В.Л., Миллер Л.В. Теория расписаний. – М.: Наука, 1975.
7. Танаев В.С., Теория расписаний. – М.: Знание, 1988.
8. Танаев В.С., Сотсков Ю.Н., Струсевич В.А. Теория расписаний. Многостадийные системы. – М.: Наука, 1989.
9. Прилуцкий М.Х. Задача упорядочения работ как задача о назначениях / М.Х. Прилуцкий, Е.А. Кумагина // Вестник Нижегородского государственного университета. Математическое моделирование и оптимальное управление. Нижний Новгород: Изд-во ННГУ, 1999. Вып. 21. C. 18–24.
10. Прилуцкий М.Х. Метод ветвей и границ с эвристическими оценками для конвейерной задачи теории расписаний / М.Х. Прилуцкий, В.С. Власов // Вестник Нижегородского университета им. Н.И. Лобачевского. – 2008. – № 3. – С. 147–153.
11. Прилуцкий М.Х. Построение оптимальных по быстродействию расписаний в канонических системах «конвейер-сеть» / М.Х. Прилуцкий, В.С. Власов // Информационные технологии. – 2011. – № 3. – С. 26-31.

# Приложение 1. Код модулей ЭГА

#include <iostream>

#include <algorithm>

#include <string>

#include <fstream>

using namespace std;

const int N = 10;// число работ (длина кодировки)

const int M = 10;// число перестановок (размер популяции)

const int T = 100;// число популяций (шагов)

int fact(int n)//подсчет факториала

{

if (n == 0) return 1;

return fact(n - 1) \* n;

}

int notUsed(int\* used, int blockNum, int N)

{

int j, pos = 0;

for (j = 1; j < N + 1; j++) {

if (!used[j]) pos++;

if (blockNum == pos)

break;

}

return j;

}

void permutation(int\* p, int num, int N)// перестановка длины N по номеру num

{

int\* used = new int[N + 1];

for (int i = 0; i < N + 1; i++)

used[i] = false; //vector<bool> used(n + 1, false);

for (int i = 0; i < N; i++) {

int blockNum = (num - 1) / fact(N - i - 1) + 1;

int j = notUsed(used, blockNum, N);

p[i] = j - 1;

used[j] = true;

num = (num - 1) % fact(N - i - 1) + 1;

}

}

int Raspis(int\* p, int N, int \*t, int\* d, int\* D)

{

int\* x = new int[N];

int\* y = new int[N];

int i;

x[0] = d[p[0]];

y[0] = x[0] + t[p[0]];

for (i = 1; i < N; i++)

{

x[i] = max(y[i - 1], d[p[i]]);

y[i] = x[i] + t[p[i]];

}

int sum = 0;

for (i = 0; i < N; i++)

if (y[i] > D[i])

{

sum += max(0, y[i] - D[i]);

}

return sum;

}

void repairs(int\* p, int N)//ремонт перестановок

{

int i, j, k;

int\* number;

number = new int[N];

for (i = 0; i < N; i++)

number[i] = 0;

for (i = 0; i < N; i++)

{

number[p[i]]++;

}

for (i = 0; i < N; i++)

{

if (number[i] == 0)

{

for (j = 0; j < N; j++)//ищем первый элемент, которых >1

if (number[j] > 1) { number[j] --; break; }

for (k = 0; k < N; k++)//ищем первый элемент j

if (p[k] == j) { p[k] = i; break; }

}

}

}

int Perebor(int k, int\* p, int N, int\* t, int\* d, int\* D)

{

int \*Best = new int[N];

static int min1 = INT\_MAX;

int m;

{

if (k == N)//полная перестановка

{

m = Raspis(p, N, t, d, D);

if (min1 > m)

{

min1 = m;

for (int j = 0; j < N; j++)

{

Best[j] = p[j];

}

}

}

}

for (int j = k; j < N; j++)

{

swap(p[k], p[j]);//переставили местами

Perebor(k + 1, p, N, t, d, D);//уменьшили хвост

swap(p[k], p[j]);

}

return min1;

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp; /\*Сохраняем значение, по адресу xp \*/

\*xp = \*yp; /\*Записываем значение, записанное по адресу yp, вячейку xp \*/

\*yp = temp; /\*Записываем исходное значение, записанное по адресу xp \*/

}

void bubbleSort(float\* arr1, int\* arr2, int n)

{

int i, j;

float temp;

//Последние элементы i уже на месте

for (i = 0; i < n - 1; i++)

{

for (j = i + 1; j < n; j++)

if (arr1[i] < arr1[j])

{

temp = arr1[i];

arr1[i] = arr1[j];

arr1[j] = temp;

swap(&arr2[i], &arr2[j]);

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

ofstream file;//создаем и открываем фаил

file.open("C:/Users/dukov/OneDrive/Рабочий стол/Диплом/ЭГА 1/Text.txt");

int n, i, j, k, step, num, a;

int first, second;// номера родителей

int\* t = new int[N];

int\* d = new int[N];

int\* D = new int[N];

int\* Best = new int[N];

int\* p = new int[N];

int\* F = new int[M];//приспособленность

for (int exp = 0;exp < 200;exp++) //цикл экспериментов

{

for (int i = 0; i < N; i++)

{ //генерировали длительности по равномерному распределению

t[i] = rand() % 20 + 11;

d[i] = rand() % 15;

D[i] = rand() % 77 + 20;

p[i] = i;

}

int M1 = Perebor(0, p, N, t, d, D);

for (int i = 0; i < N; i++)

p[i] = 0;

file << M1 << " ";//печатаем лучшую приспособленность через перебор

int\*\* population;

population = new int\*[M];

for (int i = 0; i < M; i++)

population[i] = new int[N];

int\*\* new\_population;

new\_population = new int\*[M / 2];

for (int i = 0; i < M / 2; i++)

new\_population[i] = new int[N];

for (n = 0; n < M / 2; n++)// идем по строкам и заполняем первую половину начальной популяцией.

{

num = rand() % fact(N);//номер перестановки

permutation(p, num, N);//получаем перестановку по номеру

for (int i = 0; i < N; i++)

{

population[n][i] = p[i];

}

F[n] = Raspis(population[n], N, t, d, D);

}

//запускаем цикл по поколениям

for (step = 0; step < T; step++)

{//создаем детей = заполнить вторую часть массива population

for (n = M / 2; n < M; n++)

{

first = rand() % M / 2;

second = rand() % M / 2;

population[first];//это перестановка первого родителя

num = rand() % N;// точка разрыва

for (i = 0; i < num; i++)

population[n][i] = population[first][i];

for (i = num; i < N; i++)

population[n][i] = population[second][i];

repairs(population[n], N);//восстанавливаем перестановку

F[n] = Raspis(population[n], N, t, d, D);

}

//отсев

for (n = 0; n < M; n++)

F[n] = Raspis(population[n], N, t, d, D);

for (a = 0; a < M / 2; a++)

{

int Min = F[0];

int n = 0;

for (q = 1; q < M; q++)

{

if (F[q] < Min)

{

Min = F[q];

}

if (abs(F[n]) > abs(F[q])) n = q;

}

for (int z = 0; z < M; z++)

{

if ((F[z] == Min))

{

F[z] = 1000;

break;

}

}

for (int y = 0; y < N; y++)

{

new\_population[a][y] = population[n][y];

}

}

//копируем новую популяцию в старую

for (n = 0; n < M / 2; n++)

{

for (int i = 0; i < N; i++)

{

population[n][i] = new\_population[n][i];

}

F[n] = Raspis(population[n], N, t, d, D);

}

}

file << F[0]<< endl;

}

file.close();//закрываем фаил

cout << 1;

getchar();

}

# Приложение 2. Код модулей жадного алгоритма

void Zadn0(int\* p, int N, int\* t, int\* d, int\* D)

{

float \*kof;

kof = new float[N];

for (int i = 0;i < N;i++)

{

kof[i] = ((float)D[i] - d[i]) / t[i]; //Способ 1

}

bubbleSort(kof, p, N);

delete[] kof;

}

void Zadn1(int\* p, int N, int\* t, int\* d, int\* D)

{

float \*kof;

kof = new float[N];

for (int i = 0;i < N;i++)

{

kof[i] = (float)D[i] - d[i]; //Способ 2

}

bubbleSort(kof, p, N);

delete[] kof;

}

void Zadn2(int\* p, int N, int\* t, int\* d, int\* D)

{

float \*kof;

kof = new float[N];

for (int i = 0;i < N;i++)

{

kof[i] = ((float)D[i] - d[i]) / d[i]; //Способ 3

}

bubbleSort(kof, p, N);

delete[] kof;

}

void Zadn3(int\* p, int N, int\* t, int\* d, int\* D)

{

float \*kof;

kof = new float[N];

for (int i = 0;i < N;i++)

{

kof[i] = (float)D[i] / (d[i] \* t[i]); //Способ 4

}

bubbleSort(kof, p, N);

delete[] kof;

}