

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 №7**

По дисциплине: «Эвристические методы и алгоритмы»

Выполнил студент группы ВПР32

Бояршинов Никита

Проверил

проф. Кобак В.Г.

Ростов-на-Дону  
2024

### **Введение**

В последнее время с усложнением транспортной системы, увеличением товарооборота и развитием сферы услуг повышается интенсивность перевозки товаров и предоставления услуг.

Для увеличения количества клиентов и повышения качества функционирования компаний требуется свести к минимуму затраты на транспортировку товаров или услуг. Кроме того решение данной задачи необходимо во многих областях, связанных с замкнутыми и при этом жестко связанными по времени системами, такими как: конвейерное производство, многооперационные обрабатывающие комплексы, судовые и железнодорожные погрузочные системы, перевозки грузов по замкнутому маршруту, расчет авиационных линий.

Задача коммивояжера является классической задачей комбинаторной оптимизации. Первое упоминание о ней появляется в 1759 г. Смысл задачи состоит в отыскании самого выгодного маршрута, который проходит через требуемые города единожды и затем возвращается в исходный город, т.е. в нахождении кратчайшего гамильтонова цикла в графе.

### **Постановка задачи**

Дан граф G = (X, U), где |X| = n – множество вершин (города), |U| = m – множество рёбер (возможные пути между городами). Дана симметрическая матрица чисел R(i, j), i, j ϵ 1, 2,…,n, представляющих собой веса рёбер (расстояний между вершинами xi и xj). Требуется найти перестановку fi из элементов множества X, такую, что значение целевой функции равно:



### **Гинетический алгоритм для решение задачи коммивояжера**

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.

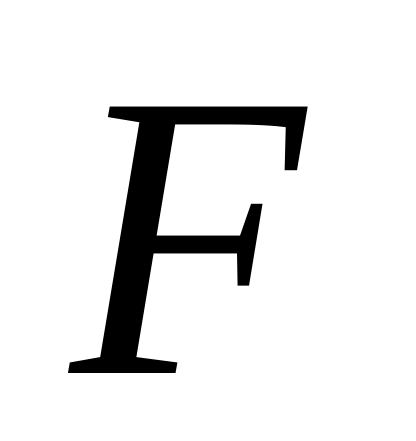
Ш.2 Отбор особей и применение ГА операторов кроссовера и мутации с заданной вероятностью для создания нового поколения.

Ш.3 Формирование списка из новых особей объединенных с родителями. Сортировка по наименьшему максимальному фенотипу, отсекание половины. Формирование нового поколения.

Ш.4 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.5 Лучшая особь выбирается как найденное решение.

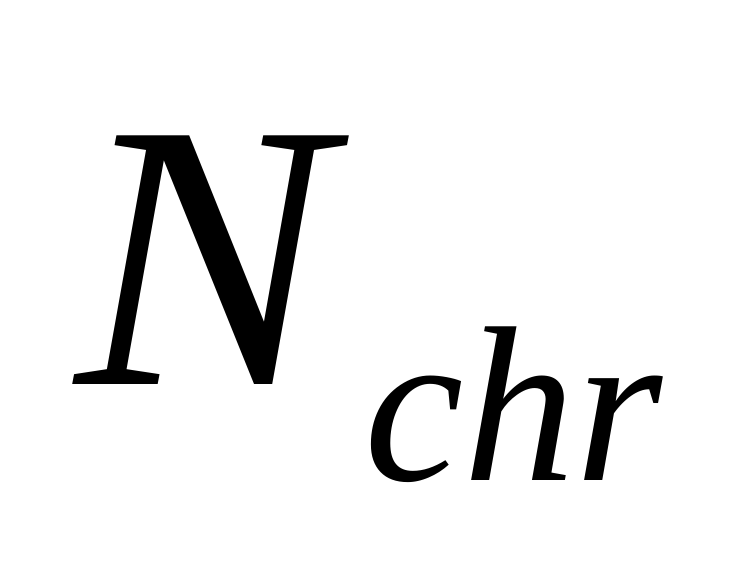
ГА является общим алгоритмом для решения любой задачи, и при его применении к конкретной необходимо выбрать механизм кодирования параметров задачи (фенотипа) в гены особи (генотипа), определить оптимизационную функцию (fitness function) и выбрать условия останова.



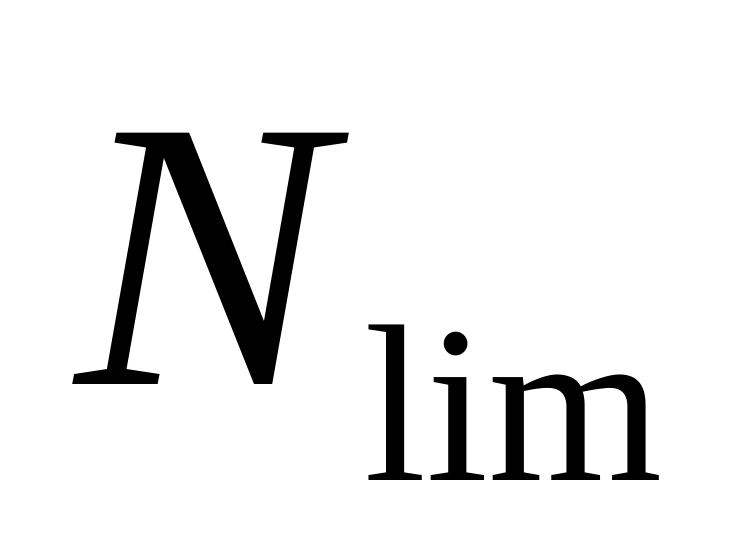
В данном случае решения задачи теории расписания минимаксный критерий будет являться оптимизационной функцией, а условием останова - неизменность лучшего решения в течение заданного числа поколений.

Введем следующие обозначения:

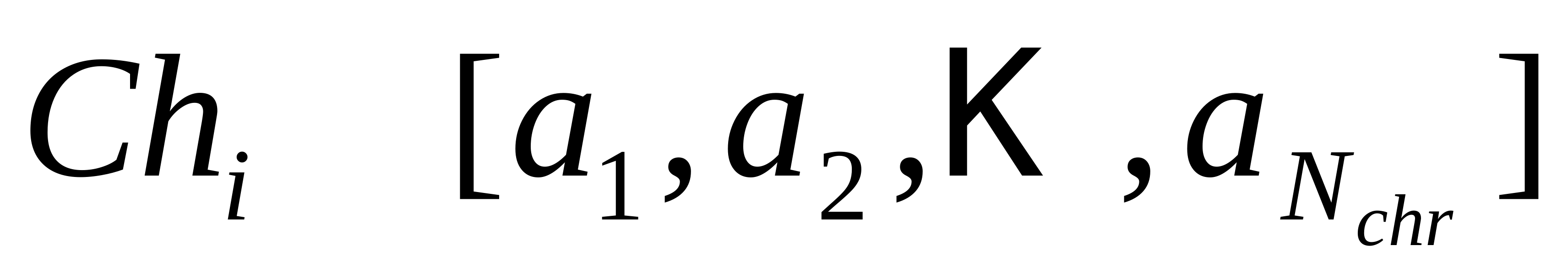
- число особей (хромосом);



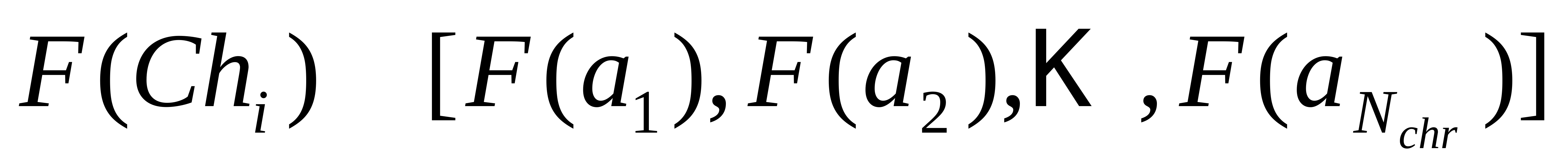
- условие останова по количеству одинаковых поколений;



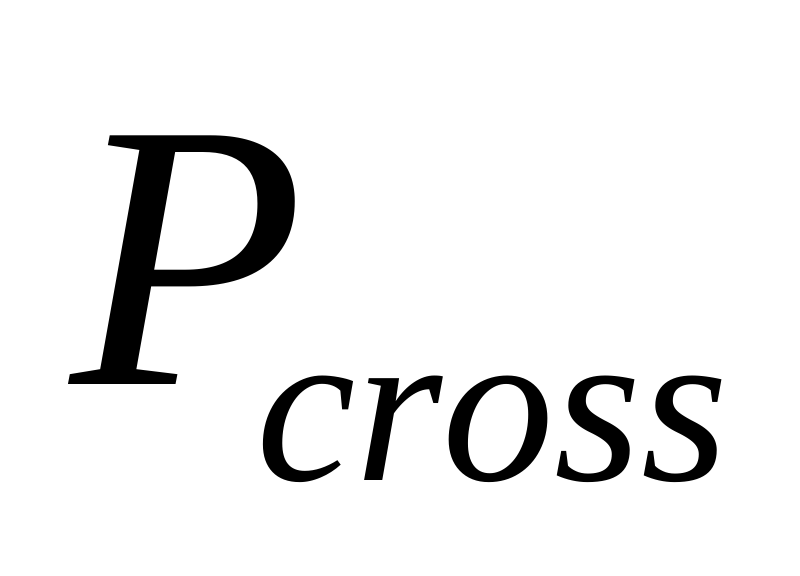
- вектор особей (хромосом);



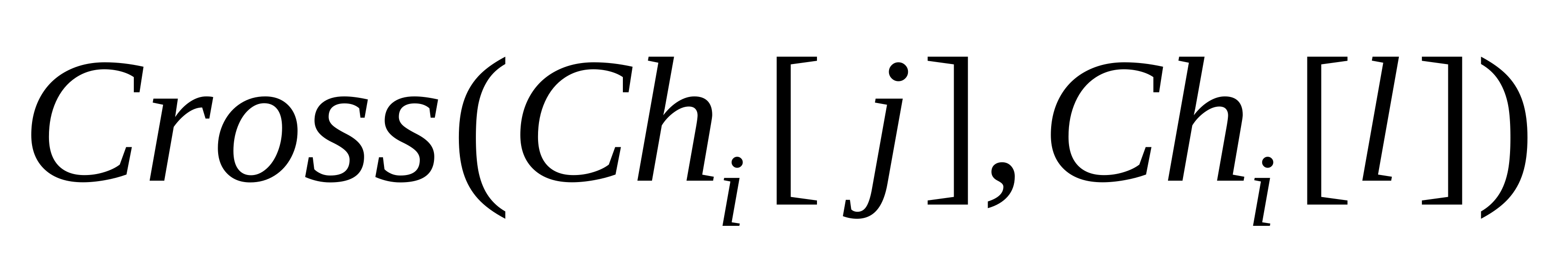
- вектор приспособленности;



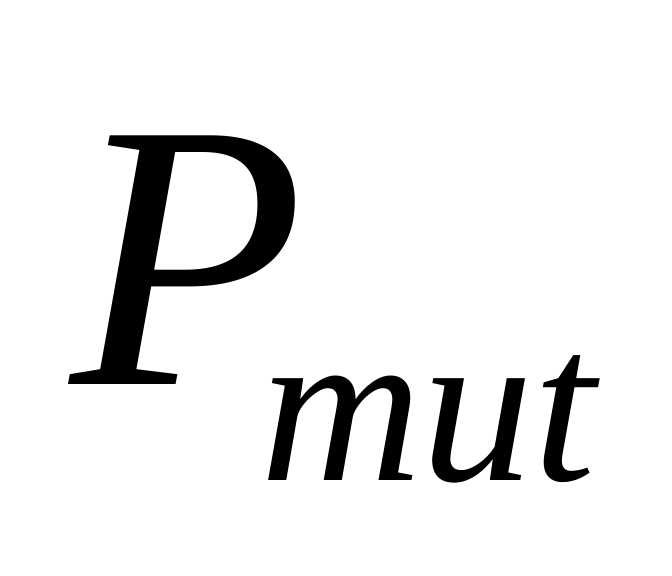
- вероятность оператора кроссовера;



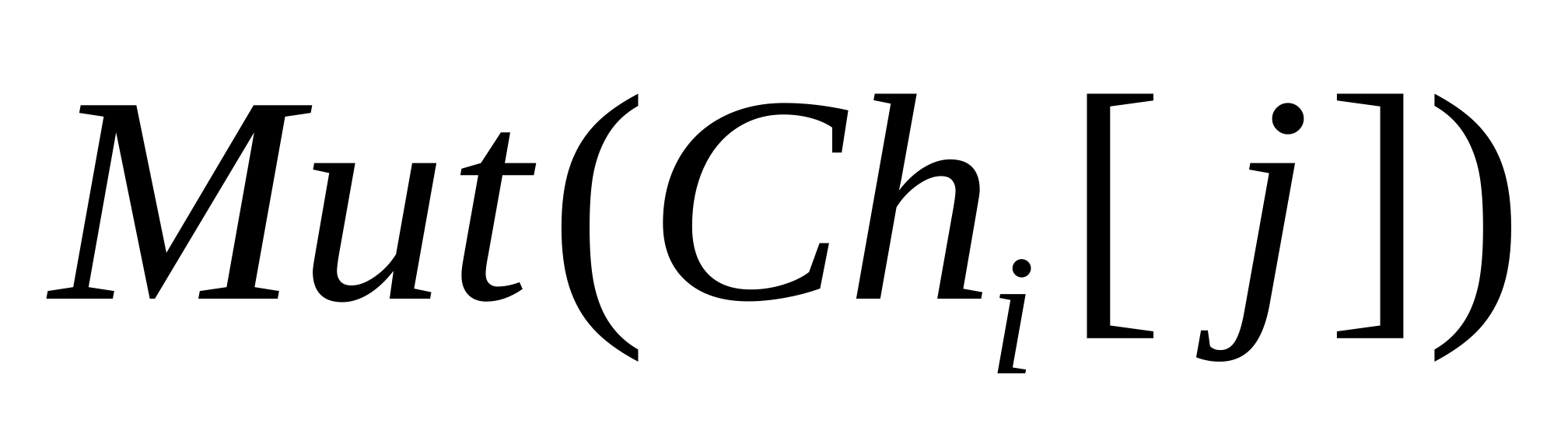
- оператор кроссовера;



- вероятность оператора мутации;

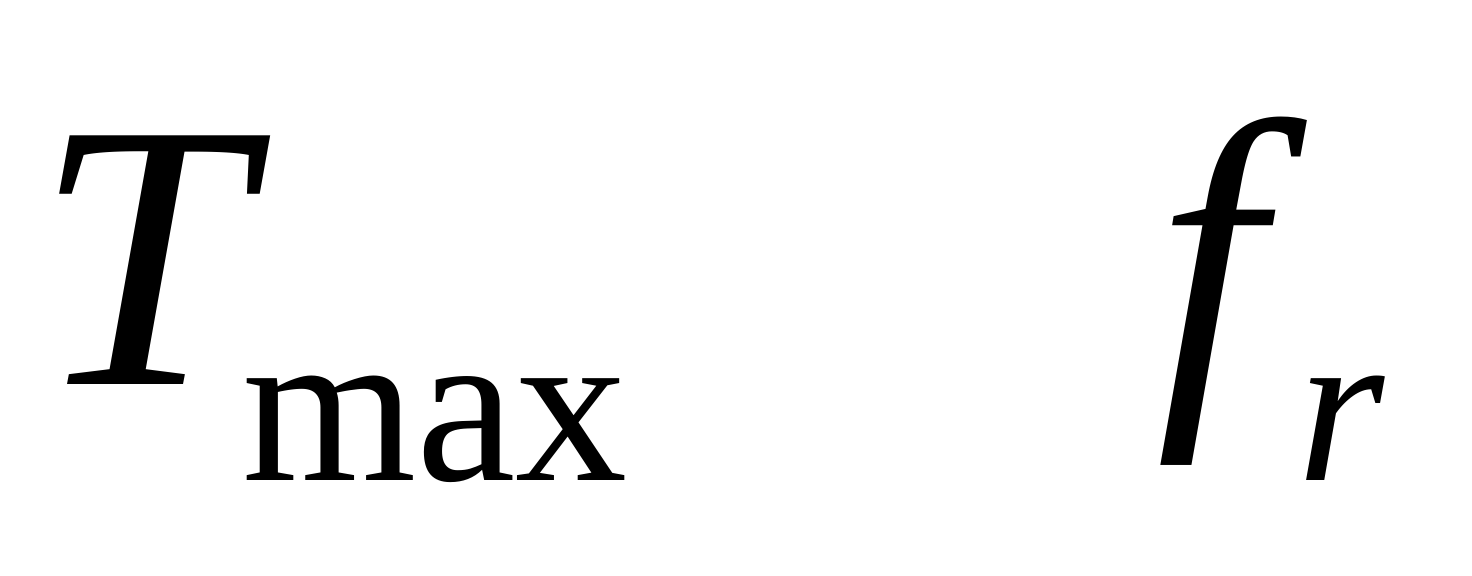
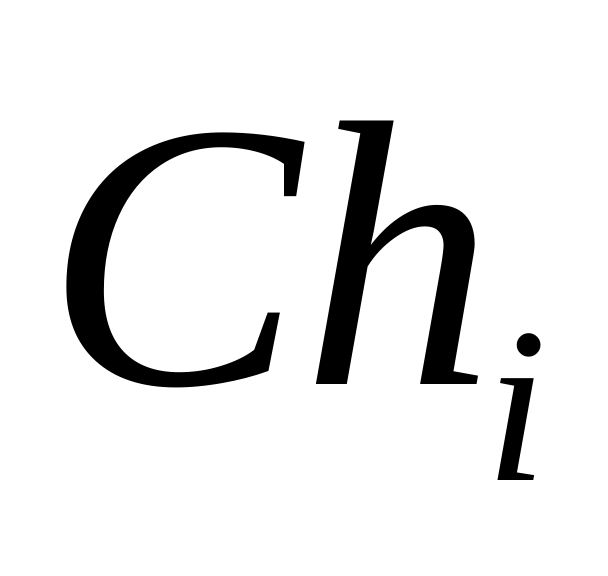


- оператор мутации.

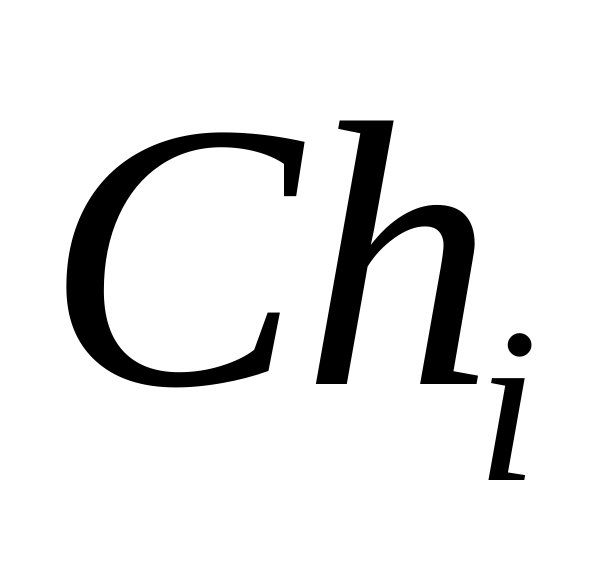
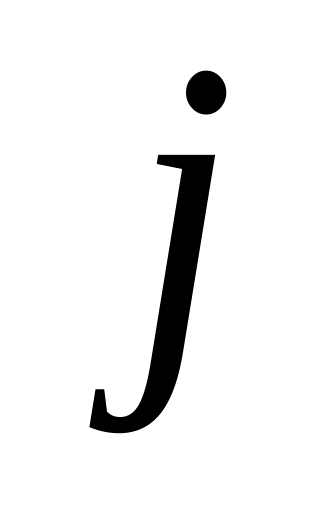
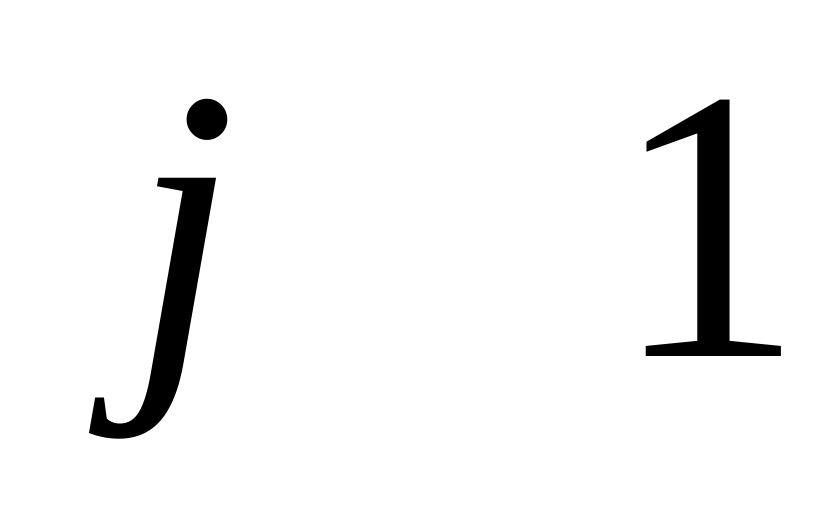


Рассмотрим подробнее Ш.2 алгоритма:

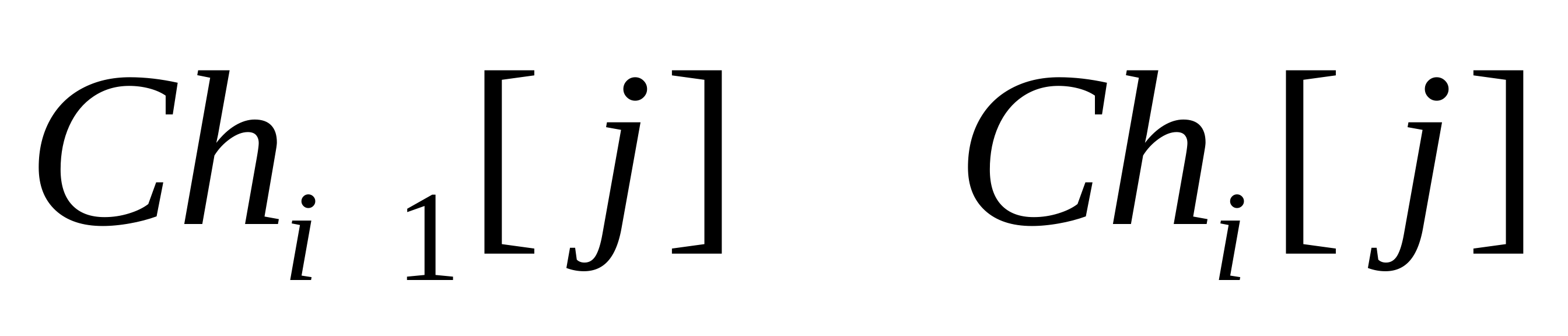
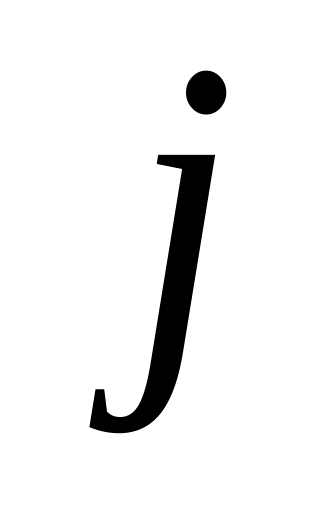
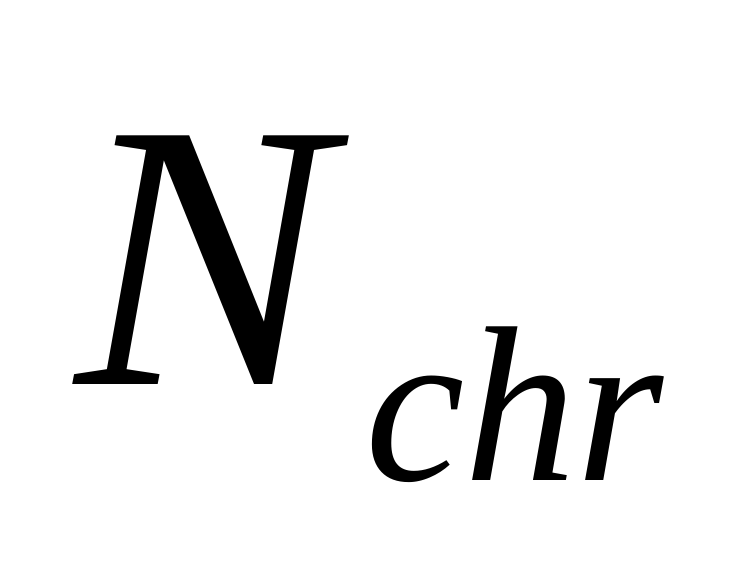
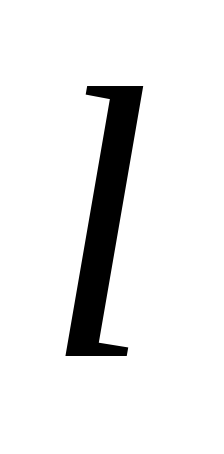
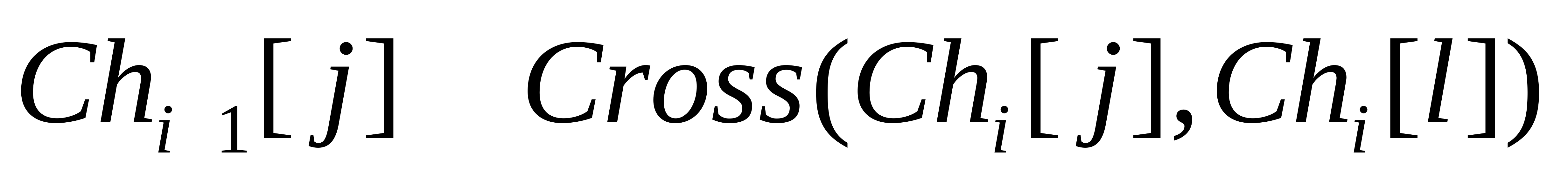
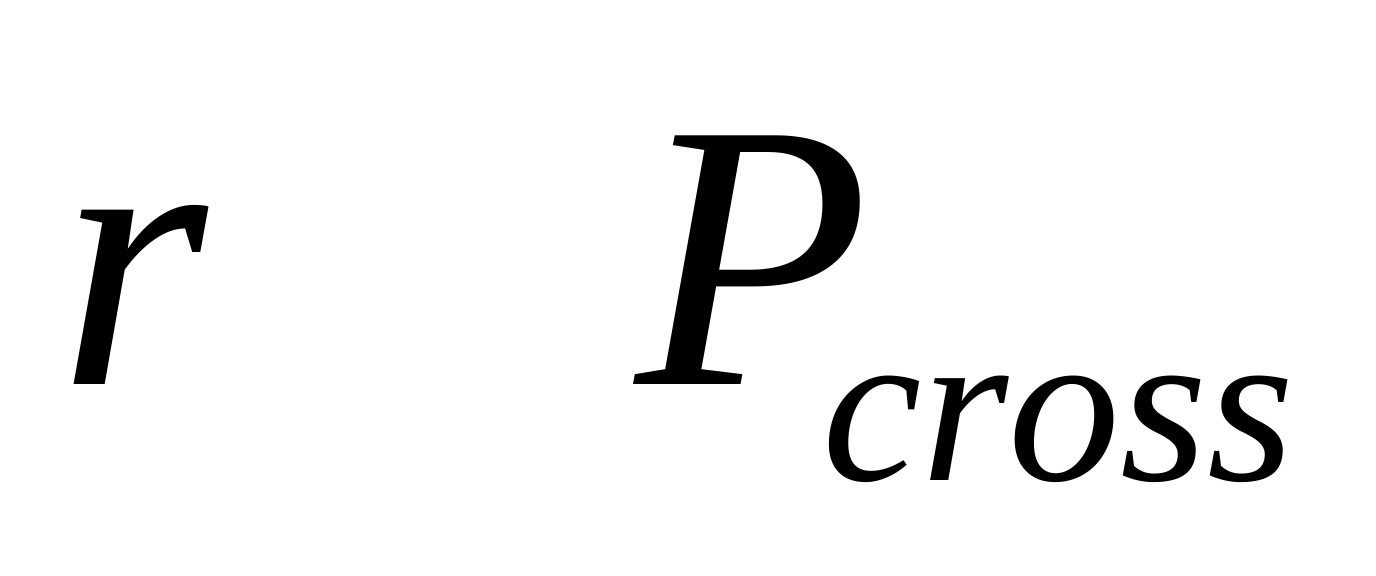
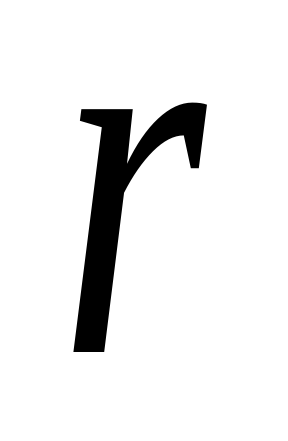
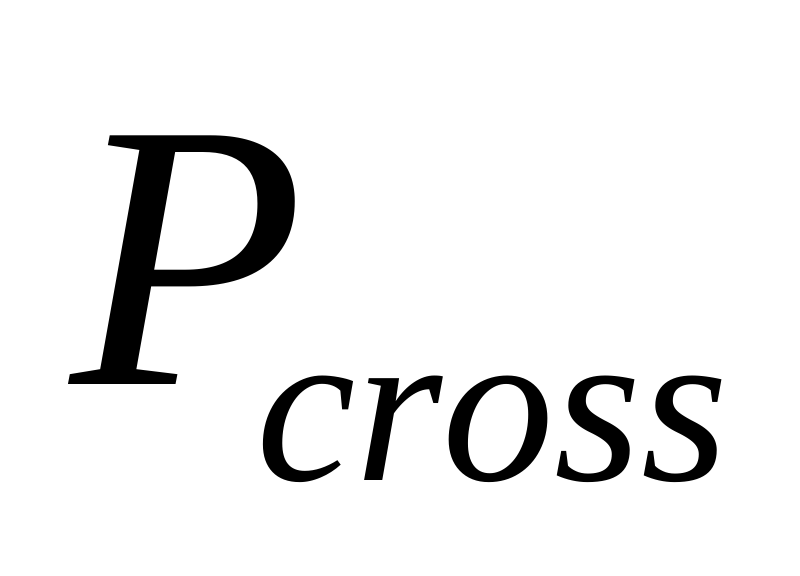
Ш.2.1 На вход шага поступает исходный вектор особей (хромосом) , каждая из особей представляет какое-то расписание с соответствующим ему и соответственно приспособленностью особи.



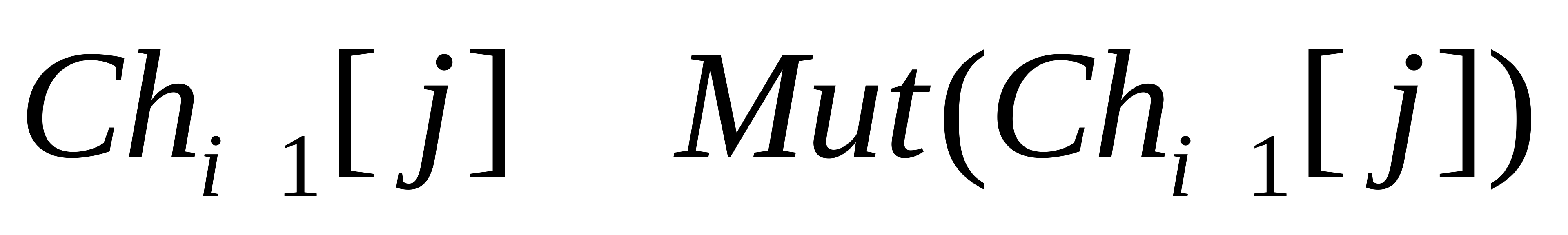
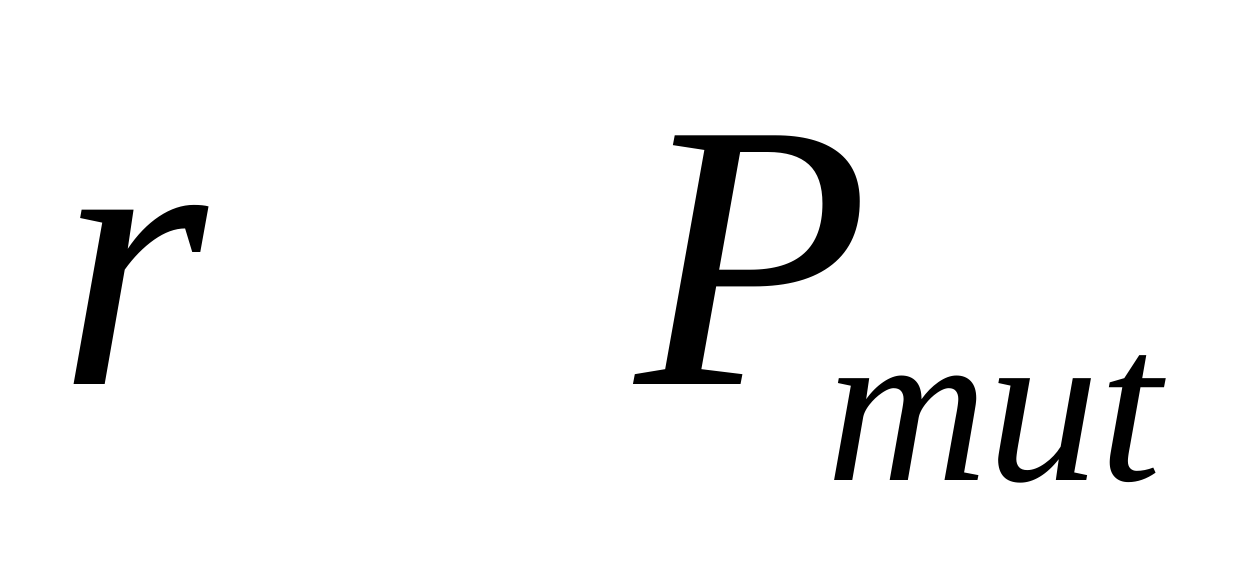
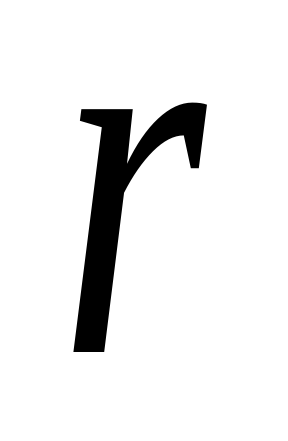
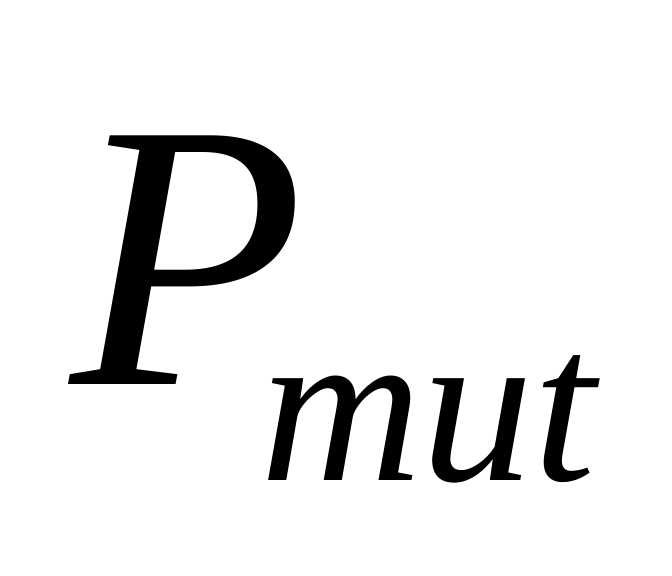
Ш.2.2 , где - порядковый номер особи в векторе .



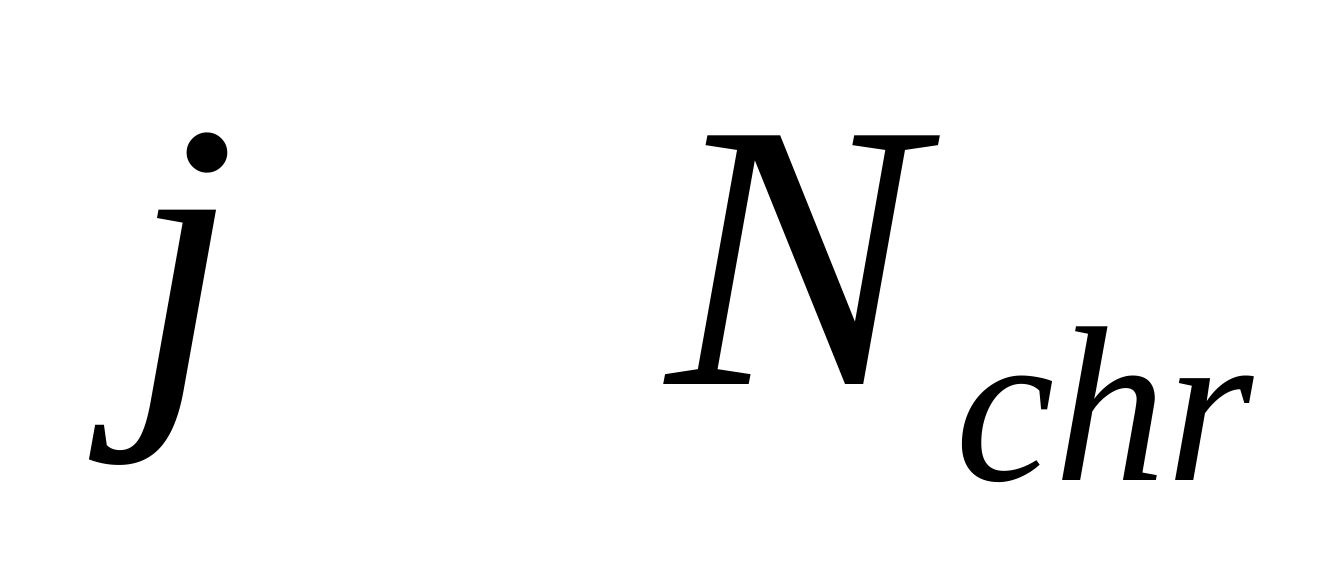
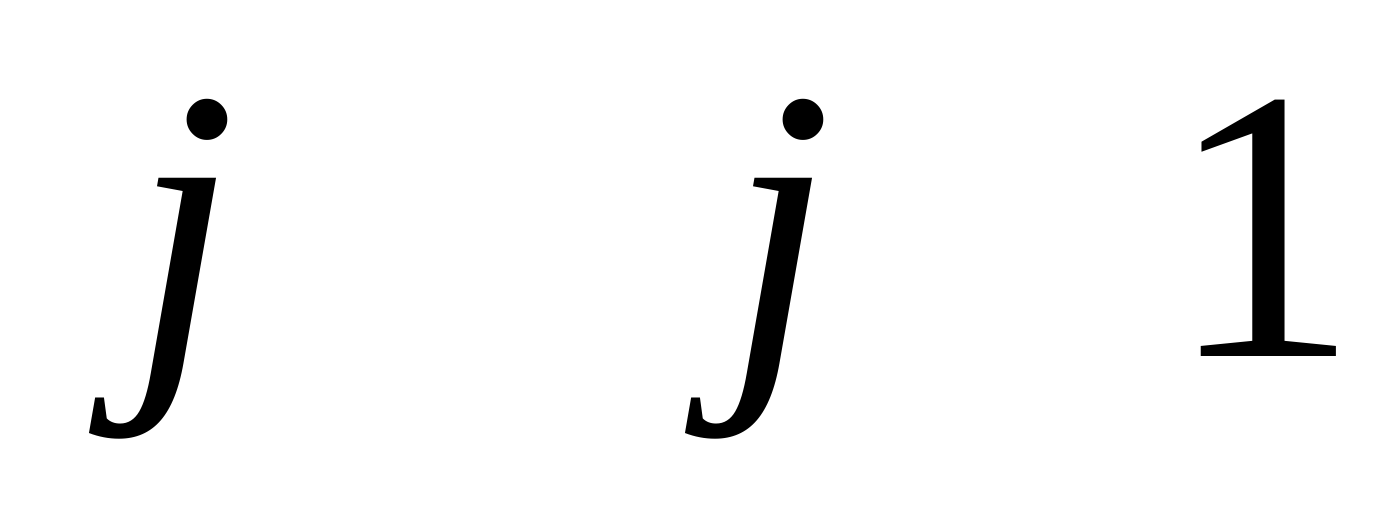
Ш.2.3 Выполняется оператор кроссовера с заданной вероятностью . Для этого генерируется два случайных числа в интервале [0..1]. Если , то , где - случайное число в интервале [1..], исключая совпадение с числом , иначе особь переходит в новое поколение без изменений, т.е. .



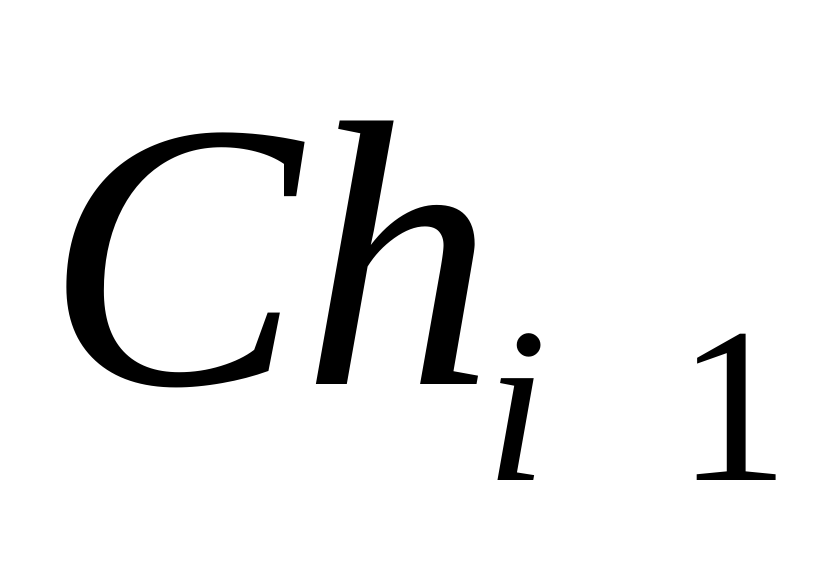
Ш.2.4 Выполняется оператор мутации с заданной вероятностью . Для этого генерируется случайное число в интервале [0..1]. Если , то .



Ш.2.5 . Если , то перейти к Ш.2.3.

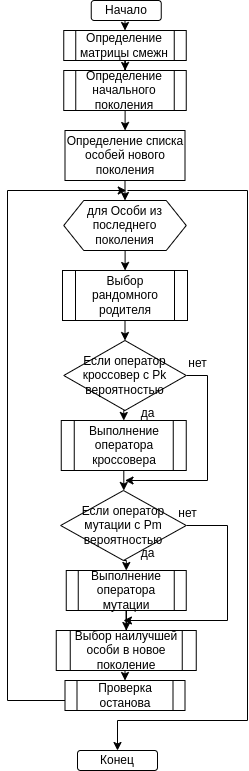


Ш.2.6 Сформировано новое поколение . Осуществляется переход на Ш.3.



Схематически работа данного алгоритма представлена в разделе <<Блок-схема>>

**Блок-схема**



Подробный листинг алгоритма приведен в приложении A.1

Пример результата работы программы изображен на рис 1.

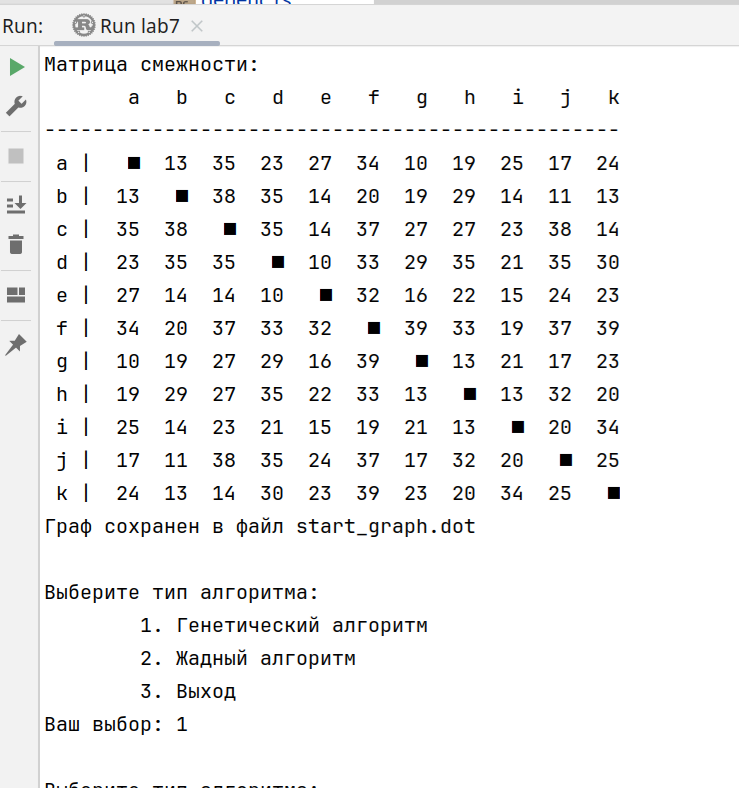


Рисунок 1 – Результат работы алгоритма

### **Вывод**

В рамках данной лабораторной работы было рассмотрено значимое направление исследований, связанных с решением задачи коммивояжера генетическим алгоритмом.

### **Список литературы**

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

### **Приложение A**

Листинг A.1 – Исходный код алгоритма

use std::io::Write;

use rand::Rng;

use rand::seq::SliceRandom;

use crate::utils::{alphabet, crossover, mutation, save\_graph\_with\_path};

fn calc\_phenotype(matrix: &Vec<Vec<u32>>, genotype: &Vec<char>) -> (u32, String) {

let alphas\_vec = alphabet().collect::<Vec<char>>()[..matrix.len()].to\_vec();

let alphas\_hash: std::collections::HashMap<char, usize> = alphas\_vec.iter().enumerate().map(|(i, &x)| (x, i)).collect();

let pairs = genotype.iter().zip(genotype.iter().skip(1));

let mut sum = 0;

let mut log\_str = String::new();

for (a, b) in pairs {

let val = matrix[alphas\_hash[a]][alphas\_hash[b]];

sum += val;

log\_str.push\_str(&format!("{:4}", val));

}

log\_str.push\_str(&format!(" = {}\n", sum));

(sum, log\_str)

}

pub fn main(matrix: &Vec<Vec<u32>>, k: u32, z: u32, p\_k: u32, p\_m: u32, start\_vertice: u32) -> u32 {

let p\_m = 30;

let mut log\_file = std::fs::File::create("genetic.log").unwrap();

let mut rnd = rand::thread\_rng();

let mut generations: Vec<Vec<(Vec<char>, u32)>> = vec![];

let alphas\_vec = alphabet().collect::<Vec<char>>()[..matrix.len()].to\_vec();

let alphas\_hash: std::collections::HashMap<char, usize> = alphas\_vec.iter().enumerate().map(|(i, &x)| (x, i)).collect();

let available\_genes: Vec<char> = alphas\_vec.clone().iter().filter(

|&x| \*x != alphas\_vec.get(start\_vertice as usize).unwrap().clone()

).map(|&x| x).collect();

let start\_alpha = alphas\_vec.get(start\_vertice as usize).unwrap();

let mut best\_sum: u32 = std::u32::MAX;

log\_file.write("\nНачальное поколение: \n".to\_string().as\_ref()).expect("Ошибка записи");

let start\_generation = {

let mut generation = vec![];

for i in 0..z {

let mut genotype: Vec<char> = vec![];

log\_file.write(format!("\nОсобь [{}] Генотип: \n", i + 1).as\_ref()).expect("Ошибка записи");

let mut genes = available\_genes.clone();

genes.shuffle(&mut rnd);

log\_file.write(format!("{:4}", start\_alpha).as\_ref()).expect("Ошибка записи");

genotype.push(start\_alpha.clone());

for el in genes.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

genotype.push(el.clone());

}

log\_file.write(format!("{:4}\n", start\_alpha).as\_ref()).expect("Ошибка записи");

genotype.push(start\_alpha.clone());

let (max\_sum, log\_str) = calc\_phenotype(matrix, &genotype);

log\_file.write(format!("{}\n", log\_str).as\_ref()).expect("Ошибка записи");

generation.push((genotype, max\_sum));

}

generation

};

generations.push(start\_generation);

let start\_time = std::time::Instant::now();

let mut gen\_counter = 0;

loop {

let last\_generation = generations.last().unwrap();

let mut new\_generation: Vec<(Vec<char>, u32)> = vec![];

log\_file.write(format!(

"\n------------- {} №{} -------------\n",

"Формирование нового поколения",

gen\_counter + 1

).as\_ref()).expect("Ошибка записи");

for (i1\_index, individual1) in last\_generation.iter().enumerate() {

let (individual2, i2\_index) = {

let mut rnd\_individual2 = rnd.gen\_range(0..last\_generation.len());

while rnd\_individual2 == i1\_index {

rnd\_individual2 = rnd.gen\_range(0..last\_generation.len());

}

(&last\_generation[rnd\_individual2], rnd\_individual2)

};

let great\_child= {

log\_file.write(format!(

"\n> - - - - - - - Скрещивание особей {} и {} - - - - - - - <\n",

i1\_index + 1,

i2\_index + 1

).as\_ref()).expect("Ошибка записи");

log\_file.write(format!("\nОсобь [{}] Генотип: \n", i1\_index + 1).as\_ref()).expect("Ошибка записи");

for el in individual1.0.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

log\_file.write(format!("Фенотип: {}\n", individual1.1).as\_ref()).expect("Ошибка записи");

log\_file.write(format!("\nОсобь [{}] Генотип: \n", i2\_index + 1).as\_ref()).expect("Ошибка записи");

for el in individual2.0.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

log\_file.write(format!("Фенотип: {}\n", individual2.1).as\_ref()).expect("Ошибка записи");

let mut child1 = individual1.clone();

let mut child2 = individual2.clone();

if rnd.gen\_range(0..100) < p\_k {

log\_file.write(format!("\nВыполнился оператор кроссовера с вероятностью {}%\n", p\_k).as\_ref()).expect("Ошибка записи");

let (geno1, geno2) = crossover(&child1.0, &child2.0);

log\_file.write(format!("Особь [#1] Генотип: \n").as\_ref()).expect("Ошибка записи");

for el in geno1.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

let (sum1, log\_str1) = calc\_phenotype(matrix, &geno1);

log\_file.write(format!("{}\n", log\_str1).as\_ref()).expect("Ошибка записи");

log\_file.write(format!("\nОсобь [#2] Генотип: \n").as\_ref()).expect("Ошибка записи");

for el in geno2.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

let (sum2, log\_str2) = calc\_phenotype(matrix, &geno2);

log\_file.write(format!("{}\n", log\_str2).as\_ref()).expect("Ошибка записи");

child1 = (geno1, sum1);

child2 = (geno2, sum2);

}

if rnd.gen\_range(0..100) < p\_m {

log\_file.write(format!("\nВыполнился оператор мутации с вероятностью {}%\n", p\_m).as\_ref()).expect("Ошибка записи");

let (geno1, log\_str1) = mutation(&child1.0);

let (geno2, log\_str2) = mutation(&child2.0);

log\_file.write(format!("Особь [1] Генотип: \n").as\_ref()).expect("Ошибка записи");

for el in geno1.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

log\_file.write(format!("{}\n", log\_str1).as\_ref()).expect("Ошибка записи");

let (sum1, log\_str1) = calc\_phenotype(matrix, &geno1);

log\_file.write(format!("{}\n", log\_str1).as\_ref()).expect("Ошибка записи");

log\_file.write(format!("\nОсобь [2] Генотип: \n").as\_ref()).expect("Ошибка записи");

for el in geno2.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

log\_file.write(format!("{}\n", log\_str2).as\_ref()).expect("Ошибка записи");

let (sum2, log\_str2) = calc\_phenotype(matrix, &geno2);

log\_file.write(format!("{}\n", log\_str2).as\_ref()).expect("Ошибка записи");

child1 = (geno1, sum1);

child2 = (geno2, sum2);

}

if child1.1 < child2.1 {

child1

} else {

child2

}

};

log\_file.write(format!("\nЛучший ребенок: \n").as\_ref()).expect("Ошибка записи");

for el in great\_child.0.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

log\_file.write(format!("Фенотип: {}\n", great\_child.1).as\_ref()).expect("Ошибка записи");

if great\_child.1 < individual1.1 && great\_child.1 < individual2.1 {

log\_file.write(format!(

"Ребенок лучше обоих родителей: {} < {} и {}\n",

great\_child.1,

individual1.1,

individual2.1

).as\_ref()).expect("Ошибка записи");

}

new\_generation.push(great\_child);

}

if generations.len() >= k as usize {

let mut last\_greet: Vec<u32> = vec![];

for index in (generations.len() - k as usize)..generations.len() {

let last\_gen = &generations[index];

let min\_max\_sum = last\_gen.iter().min\_by\_key(|el| el.1).unwrap().1;

last\_greet.push(min\_max\_sum);

}

log\_file.write(format!(

"Последние {} поколений имеют лучший определитель фенотипа {:?} соответственно\n",

k,

last\_greet

).as\_ref()).expect("Ошибка записи");

if last\_greet.iter().all(|&x| x == last\_greet[0]) {

log\_file.write(format!(

"Остановка алгоритма: последние {} поколений имеют одинаковый определитель фенотипа лучшей особи\n",

k

).as\_ref()).expect("Ошибка записи");

let last\_gen = generations.last().unwrap();

let best\_genotype = last\_gen.iter().min\_by\_key(|el| el.1).unwrap().0.clone();

let (best\_sum, best\_phenotype) = calc\_phenotype(matrix, &best\_genotype);

log\_file.write(format!("\nЛучшая особь: \n").as\_ref()).expect("Ошибка записи");

for el in best\_genotype.iter() {

log\_file.write(format!("{:4}", el).as\_ref()).expect("Ошибка записи");

}

log\_file.write("\n".to\_string().as\_ref()).expect("Ошибка записи");

log\_file.write(format!("{}\n", best\_phenotype).as\_ref()).expect("Ошибка записи");

break;

}

}

generations.push(new\_generation);

gen\_counter += 1;

}

let delta\_time = start\_time.elapsed().as\_millis();

log\_file.write(format!("\n\nВремя выполнения: {:?} мс", delta\_time).as\_ref()).expect("Ошибка записи");

log\_file.write(format!("Количество поколений: {}\n", gen\_counter + 1).as\_ref()).expect("Ошибка записи");

save\_graph\_with\_path(

matrix.clone(),

generations.last().unwrap().iter().min\_by\_key(|el| el.1).unwrap().0.clone().iter().map(

|&x| alphas\_hash[&x]

).collect(),

"genetic\_genetic".to\_string()

);

best\_sum

}