МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра систем автоматизированного проектирования

Отчет по лабораторной работе № 5

на тему: «Изучение основных операторов кроссинговера»

по курсу «Методы машинного обучения»

Выполнили:   
студенты гр. КТбо4-4

Батагов С.С.\_\_\_\_\_\_\_\_\_\_

Москаленко М.В.\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Проверил:

Доцент

Лебедев О.Б.\_\_\_\_\_\_\_\_\_

(подпись, дата)

Таганрог 2022

**СОДЕРЖАНИЕ**

[**1 ЦЕЛЬ РАБОТЫ 3**](#_Toc119149495)

[**2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 3**](#_Toc119149496)

[**2.1 Кроссинговер 3**](#_Toc119149497)

[**2.2 Оператор Кроссинговера 5**](#_Toc119149498)

[**3 ПРАКТИЧЕСКАЯ ЧАСТЬ 11**](#_Toc119149501)

[**3.1 Одноточечный кроссинговер 11**](#_Toc119149502)

[**3.2 Двухточечный кроссинговер 13**](#_Toc119149503)

[**3.3 Трёхточечный кроссинговер 14**](#_Toc119149504)

[**3.4 Циклический кроссинговер 16**](#_Toc119149505)

[**4 ВЫВОД 18**](#_Toc119149506)

[**ПРИЛОЖЕНИЕ А 19**](#_Toc119149507)

# ЦЕЛЬ РАБОТЫ

Ознакомиться с алгоритмом и стратегиями выполнения операции кроссинговера. Научиться выполнять оператор кроссинговера с различными вероятностями.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 2.1 Кроссинговер

Для всех организмов, исследованных к настоящему времени, характерна рекомбинация сцепленных генетических факторов. Типы генетической рекомбинации:

* рекомбинация хромосомных и нехромосомных генов;
* рекомбинация целых негомологичных хромосом;
* рекомбинация участков хромосом, представленных непрерывными молекулами ДНК.

При решении задач оптимизации возможно моделирование процессов рекомбинации. В этом случае любое решение рассматриваемой задачи представляется как некоторая информация, способная к обновлению посредством введения элементов другого решения. В задачах оптимизации условно считают, что хромосомы являются закодированным представлением альтернативных решений. Хромосомы, представляющие собой отображения решений, должны быть гомологичны, так как являются взаимозаменяемыми альтернативами. Новый механизм решения оптимизационных задач в отличие от существующих механизмов, осуществляет не замену одного сгенерированного решения на другое, что осуществимо простой оценкой исходных решений в соответствии с принятым критерием, а получение новых решений посредством обмена между ними информацией.

Каждый участок хромосомы (альтернативного решения) несет определенную функциональную нагрузку. Желательно создание такой комбинации участков хромосом, которая составляла бы наилучшее из решений, возможное при исходном генетическом материале. Поэтому целью рекомбинации является накопление в конечном решении всех лучших функциональных признаков, какие имелись в наборе исходных решений.

Рассмотрим рекомбинацию участков хромосом, представленных непрерывными молекулами ДНК. Здесь может быть выделено несколько подтипов рекомбинации:

* регулярная (общая) рекомбинация
* спрайт - специфическая рекомбинация
* нереальная рекомбинация

Т. Морган предположил, что кроссинговер может происходить не только в одной, но и в двух и даже большем числе точек.

Точки разрыва определяются в местах пересечения хромосом. Первая хромосома – потомок содержит по краям две части хромосомы f и в центре одну часть из хромосомы w. Вторая хромосома – потомок содержит по краям две части хромосомы w и в центре одну часть из хромосомы f. В настоящее время существует большое число схем реализации механизма кроссинговера.

Таким образом, в общей схеме реализации кроссинговера можно выделить два варианта обмена информацией. В первом варианте используется одна точка разрыва, а во втором – две и более точек разрыва.

## 2.2 Оператор Кроссинговера

Оператор кроссинговера (ОК) используется для получения новых решений на базе уже имеющихся. Известно достаточно много различных типов ОК, но наиболее часто используются следующие:

**Стандартный оператор кроссинговера.** Может применяться для бинарных, векторных и гомологичных числовых хромосом. Существуют его одноточечная, двухточечная и многоточечная модификации.

Для проведения скрещивания по одной точке случайно выбирается точка внутри хромосомы. Затем часть первого родителя, расположенная левее точки скрещивания, и часть второго родителя, расположенная правее точки скрещивания, копируются в первого потомка. Второй потомок формируется из правой части первого родителя и левой части второго родителя.

Таким образом, путем перекомпоновки двух старых (родительских) решений генерируются два качественно новых решения (потомство).

Родительские хромосомы перед операцией кроссинговера будут выглядеть, как показано на Рисунке 1. причем точка скрещивания выделена двойной линией. После выполнения одноточечного кроссинговера получается новый вариант распределения ребер по полуплоскостям, закодированный в хромосомах потомков.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1 – Одноточечный кроссинговер

При двухточечном кроссинговере в хромосоме случайно выбираются уже две точки скрещивания. Левую точку будем считать первой, а правую - второй. Первый потомок получается из частей первого потомка, расположенных левее первой точки скрещивания и правее второй точки, и части второго потомка, расположенной между первой и второй точкой скрещивания. Второй потомок получается из левой и правой частей второго потомка и центральной части первого потомка (Рисунок 2).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2 – Двухточечный кроссинговер

Помимо одноточечного ОК в ГА также широко применяются многоточечные ОК (Рисунок 3).

*Изображение выглядит как стол

Автоматически созданное описание*

Рисунок 3 – Многоточечный кроссинговер

Циклический оператор кроссинговера (Cycle crossover, CX) разработали Оливер, Смит и Холланд в 1985 году. Оператор выполняет перестановки в соответствии с циклами, которые отображают установленные соответствия между элементами (генами) первого и второго родителей (Рисунок 4).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4 – Циклический кроссинговер

# ПРАКТИЧЕСКАЯ ЧАСТЬ

Полный код программы представлен в Приложении А.

Для генерации популяции, содержащей хромосомы для кроссинговера, создадим класс Chromos, который будет генерировать хромосомы числового типа (Рисунок 4).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 — Класс для генерации популяции

## 3.1 Одноточечный кроссинговер

**Простой (одноточечный) ОК**. Перед началом работы одноточечного оператора кроссинговера определяется так называемая точка ОК, или разрезающая точка ОК, которая обычно определяется случайно. Эта точка определяет место в двух хромосомах, где они должны быть «разрезаны». Например, пусть популяция P состоит из хромосом P = {P1, P2}, которые выступают в качестве родителей (Рисунок 5).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 — Простой кроссинговер

На вход функции подаётся популяция, обозначенная переменной *ls*. new*Population* — массив, в котором будет храниться новая популяция, cutPoint — переменная, отвечающая за разрезающую точку, которая определяется случайным образом в промежутке от 2 до np-1 элемента. Далее в цикле с шагом 2, в дополнительные переменные присваиваются хромосомы\индивиды по порядку, то есть кроссинговер происходит для 1 и 2 хромосомы, далее для 3 и 4 и так далее. После в массив *newPopulation* добавляется срез из нескольких массивов. После полученный итоговый массив передаётся во вспомогательную функцию *show,* которая выводит значение популяции на экран пользователя (Рисунок 6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 — Результат выполнения функции

## 3.2 Двухточечный кроссинговер

**Двухточечный ОК**. В каждой хромосоме определяются две точки ОК, и хромосомы обмениваются участками, расположенными между двумя точками ОК (Рисунок 7).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 — Алгоритм двухточечного кроссинговера

Представленный исходный код функции имеет несколько отличий от предыдущего, а именно: добавилась переменная cut\_point2, отвечающая за вторую точку разделения, принимающая значения от точки разреза. Также было добавлено новое слагаемое в сумму срезов для формирования новой хромосомы (Рисунок 8).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 — Результат выполнения алгоритма двухточечного кроссинговера

## 3.3 Трёхточечный кроссинговер

**Трехточечный кроссинговер**. Здесь точки ОК делят хромосому на ряд строительных блоков (в данном случае 4). Потомок Р'1 образуется из нечетных блоков родителя P1 и четных блоков родителя P2. Потомок Р'2 образуется соответственно из нечетных блоков родителя P2 и четных блоков родителя P2 (Рисунок 6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 — Алгоритм трёхточечного кроссинговера

Стоит обратить внимание, что данная реализация алгоритм имеет несколько отличий от предыдущего. Была добавлена 3 переменная, которая отвечает за разделительную точку. А также были изменены слагаемые суммы срезов списков (массивов) (Рисунок 7).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 — Результат выполнения функции

## 3.4 Циклический кроссинговер

**Циклический ОК.** Циклический ОК выполняет рекомбинации согласно циклам, которые существуют при установлении соответствия между генами первого и второго родителей (Рисунок 8).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 — Алгоритм циклового кроссинговера

Стоит отметить, что данный алгоритм реализован в 2 функциях. Первая, *loop*, является вспомогательной, на вход которой передаются две хромосомы, далее формируется массив заданного размера, заполненный нулями, а дальше в цикле выполняется заполнение списка с помощью индексов.

Вторая, *loopAllg*, является основной, которой передаётся популяция хромосом, далее в цикле выделяются 2 хромосомы и передаются во вспомогательную функцию, передающую результат переменной res. Которая в последствии будет разделена на отдельные списки, и добавлены в новую популяцию (Рисунок 9).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 — Результат выполнения циклического кроссинговера

# ВЫВОД

В данной лабораторной работе были получены знания об основах кроссинговера, а также на практике были реализованы часть алгоритмов, из представленных в методических пособиях, а именно одно- двух- трех- точечный и циклический кроссинговер.

# ПРИЛОЖЕНИЕ А

**Исходный код программы кроссинговера**

Ниже будет представлен исходный код файла lab5.js в котором находятся все рассмотренные в данной работе алгоритмы.

// Реализация методов создания новой популяции

let readline = require('readline-sync');

const SIZE\_POPULATION = 6;

const SIZE\_CHR = 7;

function getRandomArbitrary(min, max) {

return Math.round(Math.random() \* (max - min) + min);

}

class Chromos {

mass = [];

constructor() {

let tempArray = []

for (let i = 0; i < SIZE\_POPULATION; i++) {

tempArray = [];

for(let j = 0; j < SIZE\_CHR; j++) {

tempArray.push(getRandomArbitrary(0,12))

}

this.mass.push(tempArray);

}

}

getPopul = () => {

return this.mass;

}

show = () => {

console.log('\nПопуляция\n')

for (let i = 0; i < this.mass.length; i++) {

console.log(`Индивид № ${i} `, JSON.stringify(this.mass[i]))

}

}

}

function onePoint(ls) {

console.log('\nПростой кроссинговер')

let newPopulation = [];

let cutPoint = getRandomArbitrary(0, ls[0].length-1)

for (let i = 0; i < ls.length; i+=2) {

let p1 = ls[i];

let p2 = ls[i + 1];

let leftP1 = p1.slice(0, cutPoint);

let rightP1 = p1.slice(cutPoint);

let leftP2 = p2.slice(0, cutPoint);

let rightP2 = p2.slice(cutPoint);

newPopulation.push(leftP1.concat(rightP2))

newPopulation.push(leftP2.concat(rightP1))

}

show(newPopulation);

}

function twoPoint(ls) {

console.log('\nДвухточечный кроссинговер')

let newPopulation = [];

let cutPoint1 = getRandomArbitrary(1, (ls[0].length / 2).toFixed(0));

let cutPoint2 = getRandomArbitrary(cutPoint1 + 1, ls[0].length - 1)

for (let i = 0; i < ls.length; i+=2) {

let p1 = ls[i];

let p2 = ls[i + 1];

let leftP1 = p1.slice(0, cutPoint1);

let middleP1 = p1.slice(cutPoint1, cutPoint2);

let rightP1 = p1.slice(cutPoint2);

let leftP2 = p2.slice(0, cutPoint1);

let middleP2 = p2.slice(cutPoint1, cutPoint2);

let rightP2 = p2.slice(cutPoint2);

newPopulation.push(leftP1.concat(middleP2.concat(rightP1)));

newPopulation.push(leftP2.concat(middleP1.concat(rightP2)));

}

show(newPopulation);

}

function threePoint(ls) {

console.log('\nТрехточечный кроссинговер');

let newPopulation = [];

let cutPoint1 = getRandomArbitrary(1, (ls[0].length/2).toFixed(0));

let cutPoint2 = getRandomArbitrary(cutPoint1 + 1, ls[0].length - 2);

let cutPoint3 = getRandomArbitrary(cutPoint2 + 1, ls[0].length - 1);

for (let i = 0; i < ls.length; i+=2) {

let p1 = ls[i];

let p2 = ls[i + 1];

let leftP1 = p1.slice(0, cutPoint1);

let middleLeftP1 = p1.slice(cutPoint1, cutPoint2);

let middleRigthP1 = p1.slice(cutPoint2, cutPoint3);

let rightP1 = p1.slice(cutPoint3);

let leftP2 = p2.slice(0, cutPoint1);

let middleLeftP2 = p2.slice(cutPoint1, cutPoint2);

let middleRigthP2 = p2.slice(cutPoint2, cutPoint3);

let rightP2 = p2.slice(cutPoint3);

newPopulation.push(leftP1.concat(middleLeftP2.concat(middleRigthP1.concat(rightP2))));

newPopulation.push(leftP2.concat(middleLeftP1.concat(middleRigthP2.concat(rightP1))));

}

show(newPopulation);

}

function loop(p1, p2) {

res = [[], []]

for (let i = 0; i < 2; i++) {

for (let j = 0; j < SIZE\_CHR; j++) {

res[i].push(0);

}

}

for (let i = 0; i < p1.length; i++) {

res[0][p1.indexOf(p1[i])] = p2[i];

res[1][p2.indexOf(p2[i])] = p1[i];

}

return res;

}

function loopAlg(ls) {

console.log('\nЦиклический алгоритм')

let newPopulation = [];

for (let i = 0; i < ls.length; i+=2) {

p1 = ls[i];

p2 = ls[i+1];

res = loop(p1, p2);

for (let i = 0; i < 2; i++ ) {

newPopulation.push(res[i]);

}

}

show(newPopulation);

}

function show(ls) {

console.log('Новая популяция');

for (let i = 0; i < ls.length; i++) {

console.log(`Хромосома ${i}: `, JSON.stringify(ls[i]))

}

console.log()

}

(() => {

const popul = new Chromos();

popul.show();

let variable = popul.getPopul();

onePoint(variable);

twoPoint(variable);

threePoint(variable);

loopAlg(variable);

})()