МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 52

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент, канд.техн.наук		Марковская Н.В.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

ИССЛЕДОВАНИЕ ВЕРОЯТНОСТИ СУЩЕСТВОВАНИЯ ПУТИ В СЛУЧАЙНОМ ГРАФЕ

по курсу: Надежность инфокоммуникационных систем

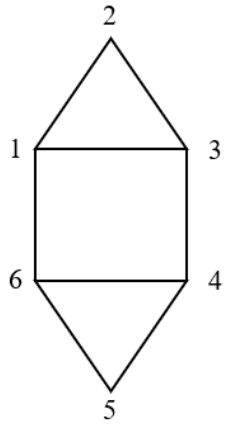
РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5912		Исаева В.И.
	подпись, дата	инициалы, фамилия

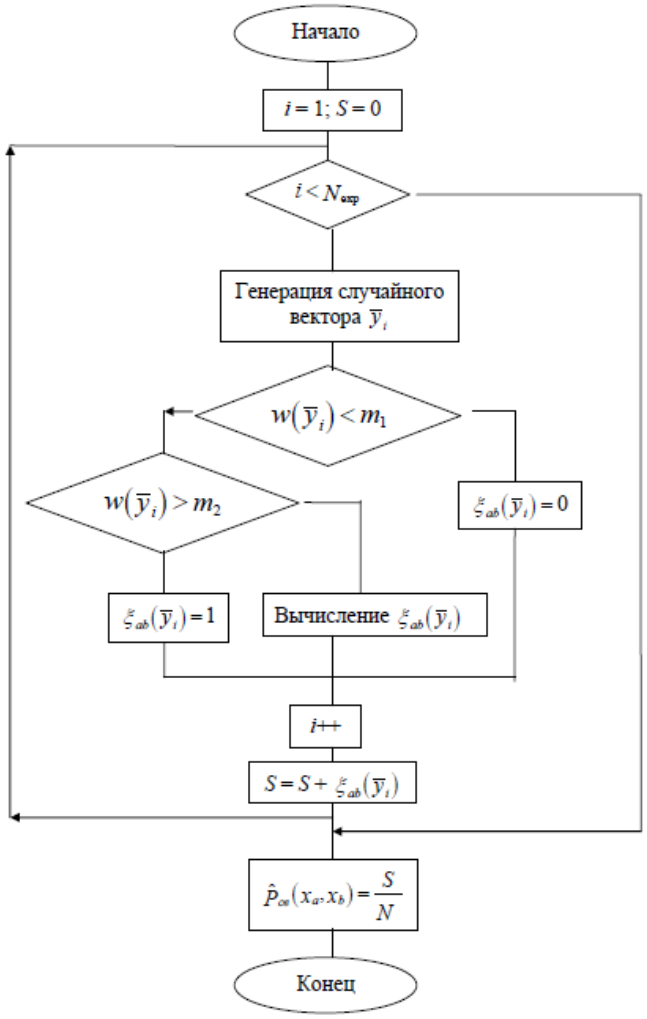
Санкт-Петербург
2022

1. Задание
   1.1. Применение простого имитационного моделирования.
   1.2. Ускоренное моделирование с использованием метода расслоенной выборки.
2. Входные данные

   Заданный граф:



   Алгоритм:



3. Простое моделирование

Вероятность ɜ ребра	Вероятность ɜ пути		
	Полный перебор	При Ɛ = 0.1	При Ɛ = 0.01
0.1	0.002368360000000001	0.004444444444444445	0.002977777777777778
0.2	0.021096959999999974	0.01777777777777778	0.02071111111111111
0.3	0.07486235999999984	0.08444444444444446	0.07435555555555555
0.4	0.17698816000000045	0.11555555555555558	0.1844
0.5	0.328125	0.3644444444444445	0.3315111111111111
0.6	0.5130777600000005	0.4755555555555557	0.5073333333333333
0.7	0.7031911599999998	0.6977777777777779	0.7088
0.8	0.864092	0.8666666666666669	0.8640444444444444
0.9	0.966975	0.9422222222222224	0.9671555555555555
1.0	1.0	0.9955555555555559	0.9999555555555556

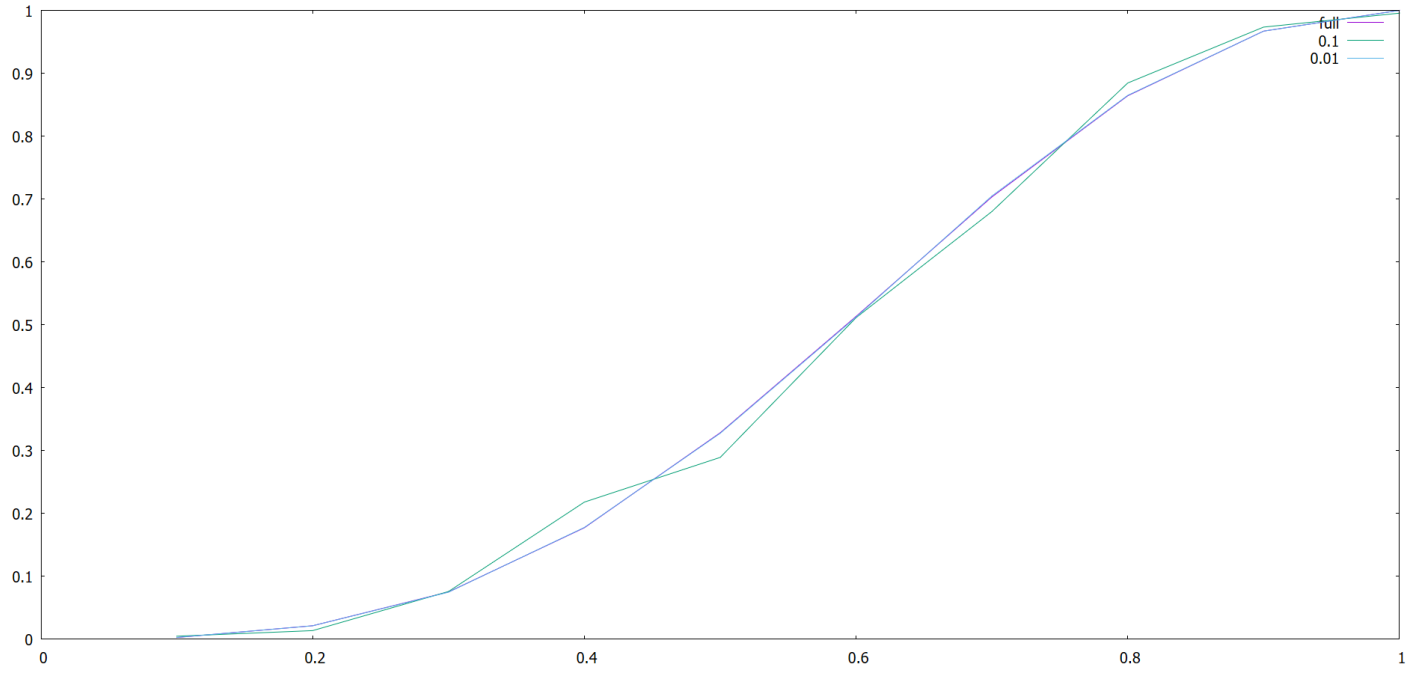


Рисунок 1. График зависимости вероятности существования пути от вероятности существования ребра

4. Ускоренное моделирование

Вероятность ɜ ребра	Вероятность ɜ пути		
	Полный перебор	При Ɛ = 0.1	При Ɛ = 0.01
0.1	0.002368360000000001	0.0044444444444445	0.0022222222222222
0.2	0.021096959999999974	0.013333333333333336	0.020755555555555555
0.3	0.07486235999999984	0.07111111111111112	0.07493333333333334
0.4	0.17698816000000045	0.19111111111111115	0.1806222222222222
0.5	0.328125	0.31555555555555564	0.33657777777777775
0.6	0.5130777600000005	0.5333333333333334	0.5372
0.7	0.7031911599999998	0.728888888888889	0.7353333333333333
0.8	0.864092	0.897777777777778	0.8967111111111111
0.9	0.966975	0.9733333333333336	0.9817777777777777
1.0	1.0	0.9955555555555559	0.9999555555555556

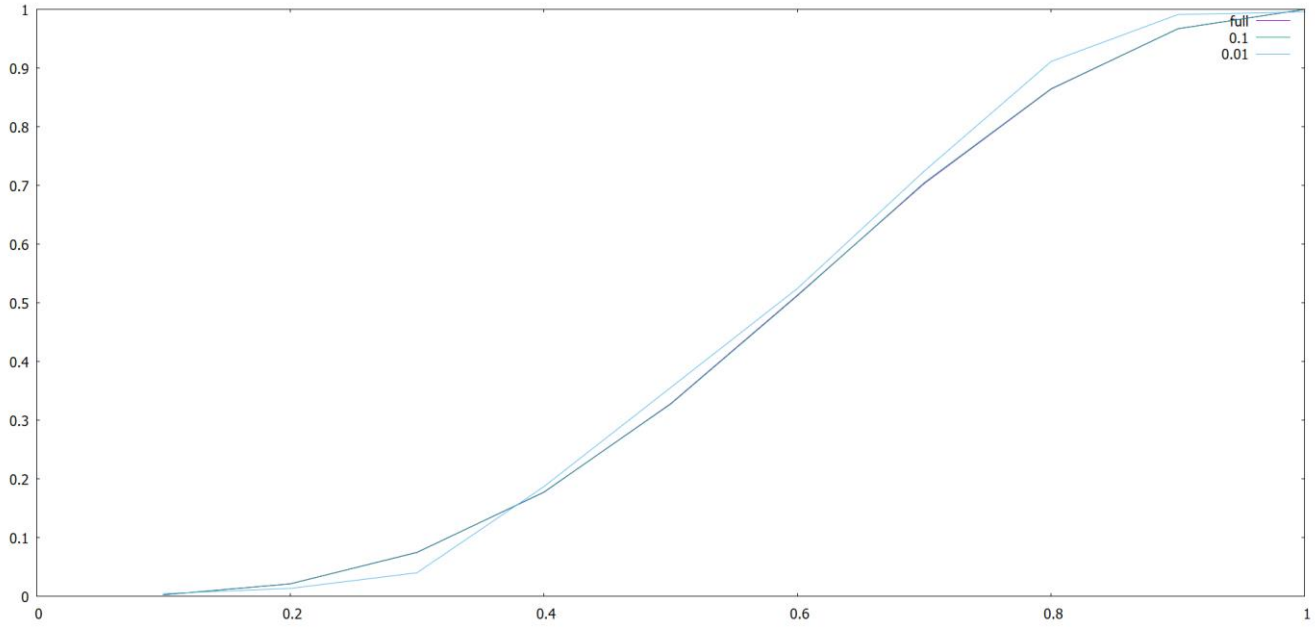


Рисунок 2. График зависимости вероятности существования пути от вероятности существования ребра

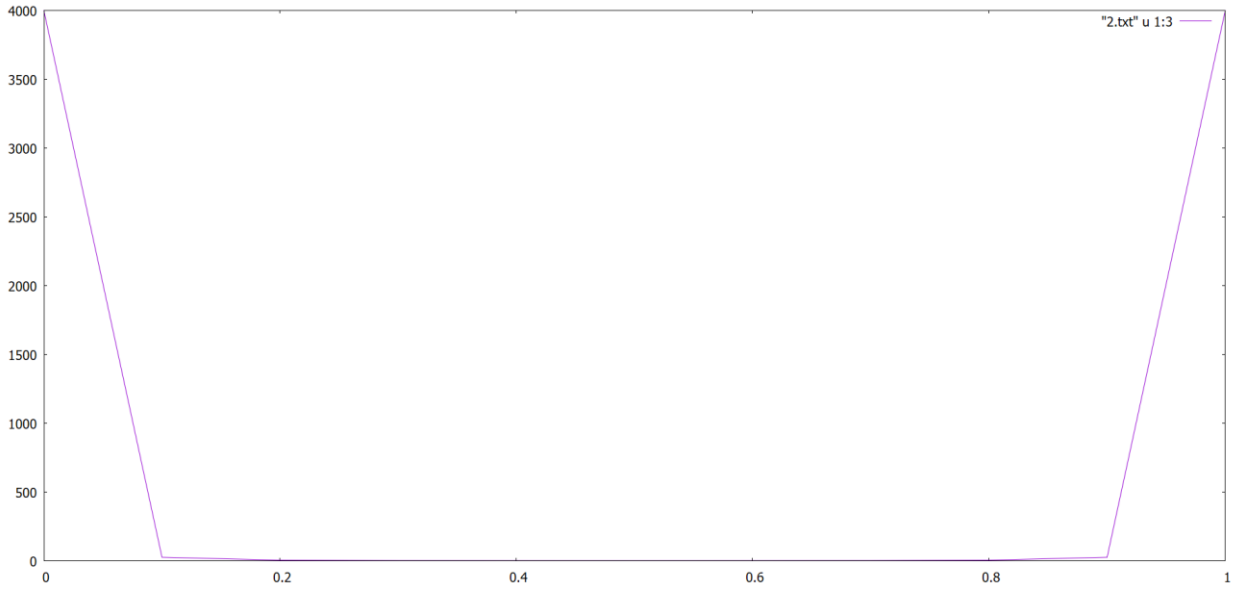


Рисунок 3. График выигрыша при использовании ускоренного имитационного моделирования

Вывод

В ходе выполнения лабораторной работы была вычислена вероятность существования пути между заданной парой вершин 2 и 5 в графе с помощью простого и ускоренного имитационного моделирования, правильность работы которого была проверена с помощью программно реализованного полного перебора, а также построен график выигрыша при использовании ускоренного моделирования

```
public class Graph {
    int vertices;
    LinkedList<Edge> edges;
    LinkedList<String> paths;

    public Graph(int v, LinkedList<Edge> e){
        vertices = v;
        edges = e;
        paths = new LinkedList<>();
    }

    public double findP(Graph g, double p){
        double sum = 1;
        if(p > 0.9)
            sum = 1;
        for(Edge e : edges){
            if(g.findEdge(e.getPoint1(), e.getPoint2()))
                sum = sum*p;
            else sum= sum*(1-p);
        }
        return sum;
    }

    public boolean findPath(int x, int y){
        if(!findVertex(x)||!findVertex(y))
            return false;
        find(x, y, 0, Integer.toString(x));
        return !paths.isEmpty();
    }

    public boolean find(int x, int y, int num, String path){
        if(num > pow(2, edges.size()))
            return false;

        for(int i = 1; i<= vertices; i++){
            if(!path.contains(Integer.toString(i))) {
                if (findEdge(x, i) && i!=y) {
                    find(i, y, num, path.concat(Integer.toString(i));

                }
            }
        }
        if(findEdge(x, y)) {
            paths.addLast(path.concat(Integer.toString(y)));
            num++;
            return true;
        }
        return false;
    }

    public boolean findEdge(int x, int y){
        for(Edge e : edges){
            if(e.isEdge(x, y))
                return true;
        }
        return false;
    }
```

```
    public boolean findVertex(int x){
        for(Edge e : edges){
            if(e.isVertex(x))
                return true;
        }
        return false;
    }
}


public class Edge {
    int point1;
    int point2;

    public Edge(int x, int y){
        point1 = x;
        point2 = y;
    }

    public boolean isVertex(int x){
        if(point1 == x || point2 == x)
            return true;
        return false;
    }

    public boolean isEdge(int x, int y){
        if(point1 == x && point2 == y || point1 == y && point2 == x)
            return true;
        return false;
    }

    public int getPoint1(){
        return point1;
    }

    public int getPoint2(){
        return point2;
    }
}


public class Model {

    Graph graph;

    int minPath;

    int maxPath;




    public Model(Graph g){

        graph = g;

    }
```

```
public double[] imitation(int s, int f, double p, double e){
    double res[] = new double[2];
    double n = 2.25/pow(e, 2);
    graph.findPath(s, f);
    minPath = graph.getMinPath();
    maxPath = graph.getMaxPath();
    int S = 0;
    for(int  i = 1; i < n; i++) {
        String path = getRandomPath(p);
        if (graph.isPath(path, s, f)) {
            S++;
        }
    }
    res[0] =  S/n;
    res[1] = p*(1-p)/n;
    return res;
}

public double[] optimization(int s, int f, double p, double e){
    double[] res = new double[2];
    double n = 2.25/pow(e, 2);
    graph.findPath(s, f);
    minPath = graph.getMinPath();
    maxPath = graph.getMaxPath();
    int iCount = 0;
    int m = graph.getEdges();
    double nj = 0.0;
    for(int j = minPath;j<=maxPath;j++ )
        nj += C(j, m) * pow(p, j) * pow(1 - p, m - j);
    res[0] = 1.0/nj;
    int S = 0;
    for(int  i = 1; i < n; i++) {
        String path = getRandomPath(p);
        int w = 0;
        for(int j = 0; j < path.length(); j++)
            if(path.charAt(j) == '1')
```

```
                w++;
            if (w < minPath)
                continue;
            if (w > maxPath) {
                S++;
                continue;
            }
            if (graph.isPath(path, s, f)) {
                iCount++;
                S++;
            }
        }
        //res[0] = n/iCount;
        res[1] = S/n;
        return res;
    }

    private String getRandomPath(double p){
        StringBuilder path = new StringBuilder();
        int length = graph.getEdges();
        int i  = 0;
        while(i < length) {
            double k = (Math.random());
            if(k < p )
                path.append(1);
            else path.append(0);
            i++;
        }
        return path.toString();
    }

    private double C(int i, int n){
        double c = (double)factorial(n)/(factorial(i)*factorial(n-i));
        return c;
    }
```

```
    private int factorial(int n)
    {
        int ret = 1;
        for (int i = 1; i <= n; ++i) ret *= i;
        return ret;
    }
}


public class Main {
    static LinkedList<LinkedList<Edge>> enumerates;

    public static boolean contains(LinkedList<Edge> edges){
        for(LinkedList<Edge> e : enumerates) {
            int sum = 0;
            for (Edge edge : edges){
                if(e.contains(edge))
                    sum++;
            }
            if(sum == edges.size())
                return true;
        }
        return false;
    }

    public static boolean enumerate(LinkedList<Edge> data, int n,
LinkedList<Edge> edges){
        if(edges.size() == n){
            LinkedList<Edge> e = new LinkedList<>(edges);
            if(!contains(e))
                enumerates.addLast(e);
            return true;
        }
        for(Edge e : data){
            if(!edges.contains(e)){
                edges.addLast(e);
                enumerate(data, n, edges);
                edges.remove(e);
            }
        }
        return  false;
    }

    public static void main(String[] args){
        LinkedList<Edge> edges = new LinkedList<>();
        edges.addLast(new Edge(1, 2));
        edges.addLast(new Edge(1, 3));
        edges.addLast(new Edge(1, 6));
        edges.addLast(new Edge(3, 2));
        edges.addLast(new Edge(3, 4));
        edges.addLast(new Edge(4, 5));
```

```
        edges.addLast(new Edge(4, 6));
        edges.addLast(new Edge(5, 6));
        Graph graph = new Graph(6, edges);
        //graph.findPath(2, 5);
        Model model = new Model(graph);

        for (int i = 1; i < 11; i++) {
            double p = (double) i / 10;
            double[] res = model.imitation(2, 5, p, 0.1);
            double[] res2 = model.optimization(2, 5, p, 0.1);
        }
}
```