1 Цель работы

В ходе выполнения работы необходимо вычислить вероятность существования пути между заданными вершинами в графе $\widehat{P_{св}}(x_a, x_b)$ с помощью имитационного моделирования и ускоренного имитационного моделирования, построить зависимость вероятности существования пути в случайном графе от вероятности существования ребра.

2 Исходные данные

Пусть задан случайный граф $G(X, Y, P)$, где $X = \{x_i\}$ – множество вершин, $Y = \{(x_i, x_j)\}$ – множество ребер, $P = \{p_i\}$ – множество вероятностей существования ребер, причем

$$P = \{p_i\} : p_i = p \text{ для } \forall i$$

Согласно полученному варианту, дан граф с параметрами $G(7,9,P)$, где $P = \{p\}$ и p пробегает значения от 0 до 1 с шагом 0,1. Граф изображен ниже.

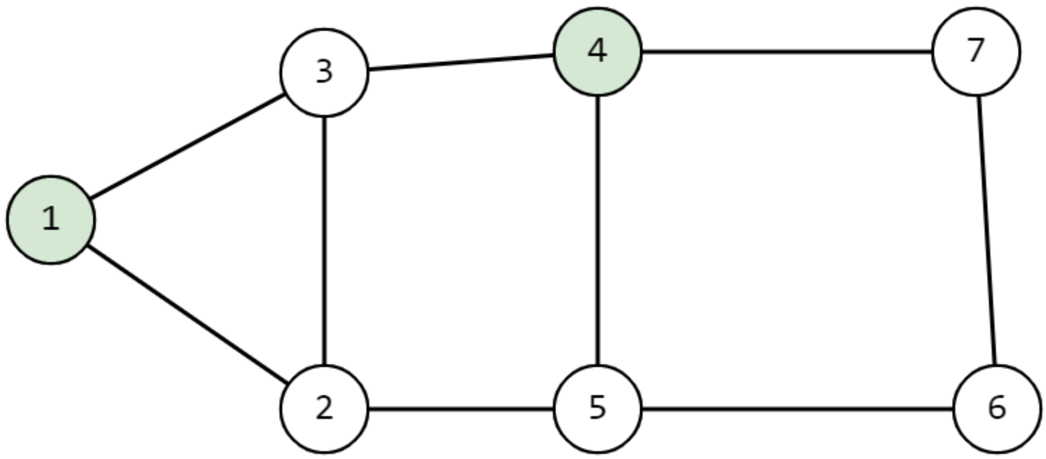

Схема 1 - Исходный граф

Необходимо вычислить вероятность существования пути между вершинами 1 и 4 ($\hat{P}_{св}(1,4)$) и построить зависимость вероятности существования пути в случайном графе от вероятности существования ребра.

3 Имитационное моделирование

Пусть имеется некоторая система (в данном случае случайный граф $\widetilde{G}\,(X, Y, P)$, состояние которой описывается случайным двоичным вектором $\overline{y}$ (в данном случае $\overline{y}$ соответствует множеству ребер Y, и нулевое значение соответствует отсутствию ребра, а единичное – его наличию в случайном графе $\widetilde{G}$). От значения вектора $\overline{y}$ зависит состояние системы $\xi(\,\overline{y})$, которое является двоичным значением (в данном случае $\xi(\,\overline{y})$ определяет связность пары вершин x_iи x_j при единичном значении и несвязность при нулевом). Поскольку $\overline{y}$ является случайным, то и значение $\xi(\,\overline{y})$ также является случайной величиной. В процессе имитационного моделирования требуется оценить математическое ожидание $M[\xi(\,\overline{y})] = P_{\text{св}}(x_a, x_b)$, которое будет являться вероятностью связности пары вершин. После проведения N_{exp} экспериментов оценка математического ожидания вычисляется по формуле

$$\widehat{P_{\text{св}}}(x_a, x_b) = \frac{\sum_{i=1}^{N_{exp}} \xi(\,\overline{y_i})}{N_{exp}}$$

где $\overline{y_i}$ является значением случайного вектора $\overline{y}$ в ходе i-го эксперимента.

В ходе выполнения была получена таблица зависимости $\widehat{P_{\text{св}}}(1,4)$ (вероятность существования пути из вершины 1 в вершину 4) от $P = \{p_i\}$ (вероятность существования ребра), где p_i пробегают значения от 0 до 1 с шагом 0.1, причем точность оценки $\varepsilon = 0.01$. Данная зависимость представлена в таблице ниже.

Таблица 1 - Зависимость, полученная имитационным моделированием

p	$\widehat{P_{\text{св}}}(1,4)$
0	0
0.1	0.0116
0.2	0.05422222222222222
0.3	0.1380888888888889
0.4	0.2601333333333333
0.5	0.41746666666666665
0.6	0.5837333333333333

0.7	0.7615111111111111
0.8	0.8919111111111111
0.9	0.9777777777777777
1	1.0

График зависимости представлен ниже.

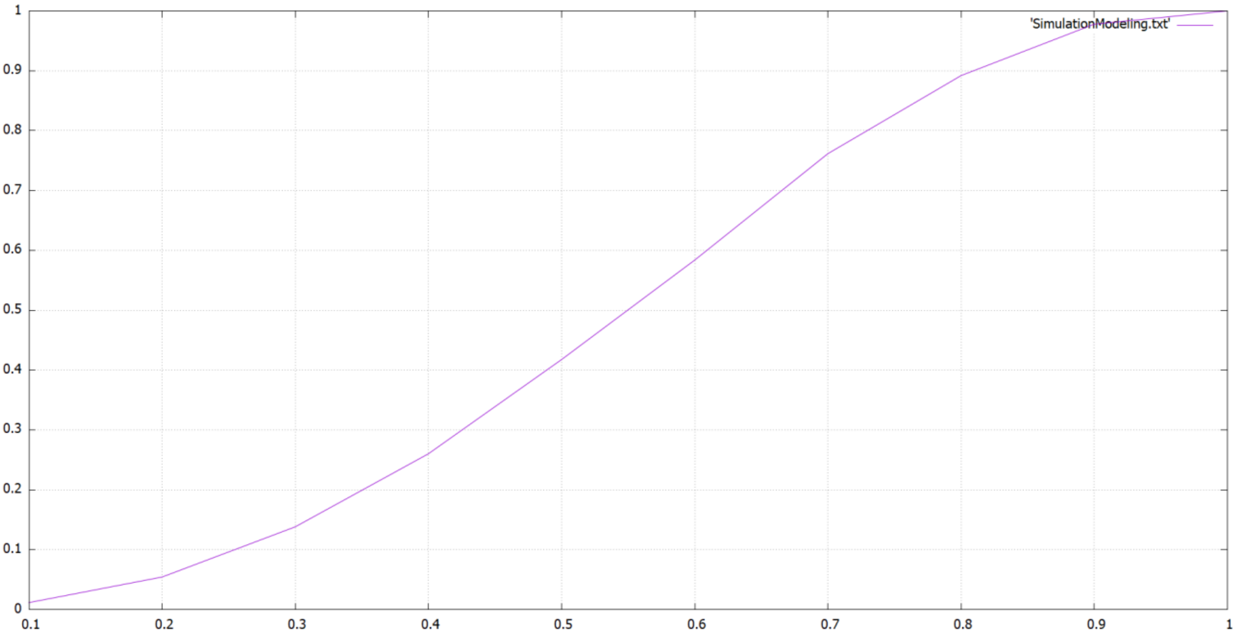

График 1 - График, полученный имитационным моделированием

Стоит, однако отметить, что при каждом новом запуске имитационной модели таблица зависимостей будет меняться, т.е. полученная зависимость и график являются частным случаем работы. При этом нам важно понимать в общем случае, совпадают ли все полученные результаты с допустимой погрешностью $\pm\varepsilon$ с результатами, полученными полным перебором. В противном случае, нельзя утверждать о корректной работе программы.

4 Ускоренное имитационное моделирование

Данный метод использует тот же алгоритм нахождения $\widehat{P_{\text{св}}}(1,4)$, что и обычное имитационное моделирование. Однако ускоренная модель учитывает верхнюю и нижнюю границу для числа ребер в случайном подграфе, а значит, исключает из алгоритма часть шагов. Для этого необходимо ввести два значения: l_{min}и l_{max}. Для данного графа эти значения равны $l_{min} = 2$ и $l_{max} = l - 2 = 9 - 2 = 7$. Если принять число ребер в случайном подграфе l', то можно сказать, что при $l' < l_{min}$ пути гарантированно нет, а при $l' > l_{max}$ пути гарантированно есть.

В ходе выполнения была получена таблица зависимости $\widehat{P_{\text{св}}}(1,4)$ (вероятность существования пути из вершины 1 в вершину 4) от $P = \{p_i\}$ (вероятность существования ребра), где p_i пробегают значения от 0 до 1 с шагом 0.1, причем точность оценки $\varepsilon = 0.01$. Данная зависимость представлена в таблице ниже.

Таблица 2 - Зависимость, полученная ускоренным имитационным моделированием

p	$\widehat{P_{\text{св}}}(1,4)$
0	0
0.1	0.012088888888888889
0.2	0.057777777777777775
0.3	0.13497777777777778
0.4	0.2632888888888889
0.5	0.4236888888888889
0.6	0.5916888888888889
0.7	0.7613333333333333
0.8	0.8938666666666667
0.9	0.9774666666666667
1	1.0

График зависимости представлен ниже.

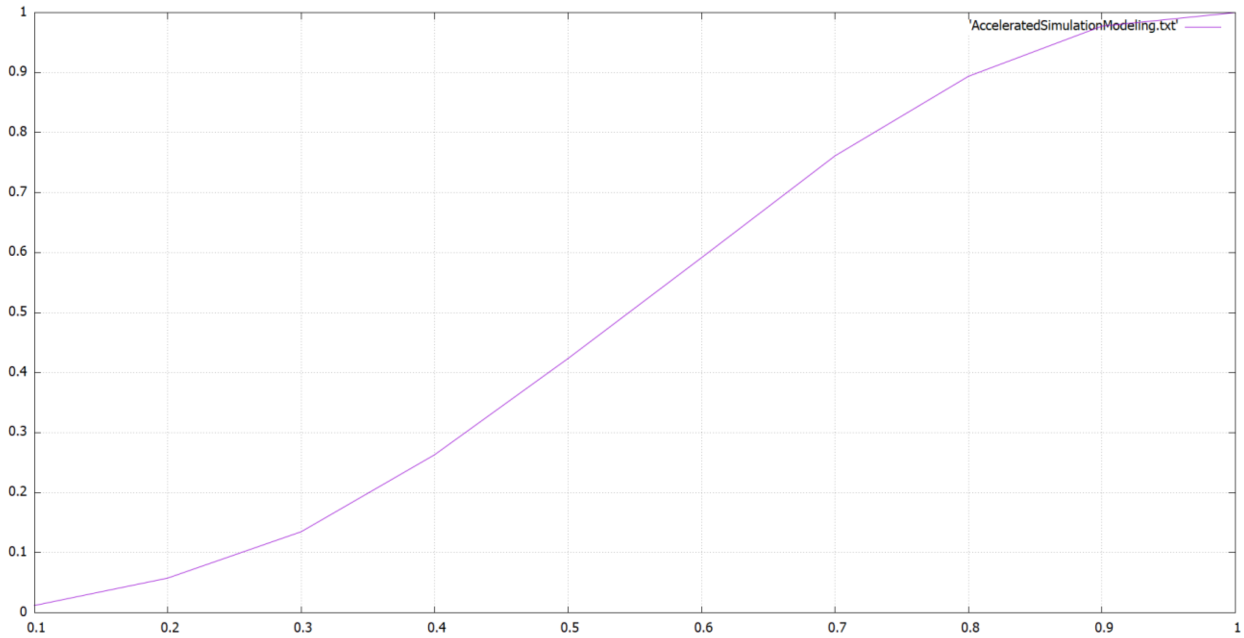

График 2 - График, полученный ускоренным имитационным моделированием

Стоит, однако отметить, что при каждом новом запуске имитационной модели таблица зависимостей будет меняться, т.е. полученная зависимость и график являются частным случаем работы. При этом нам важно понимать в общем случае, совпадают ли все полученные результаты с допустимой погрешностью $\pm\varepsilon$ с результатами, полученными полным перебором. В противном случае, нельзя утверждать о корректной работе программы.

5 Выводы

Для наглядного сравнения результаты представлены в таблице ниже.

Таблица 3 - Сравнительная таблица

p	Полный перебор	Имитационная модель		Ускоренная имитационная модель	
		$\widehat{P_{св}}(1,4)$	$\|\Delta\|$	$\widehat{P_{св}}(1,4)$	$\|\Delta\|$
0	0	0	0	0	0
0.1	0.0119	0.0116	0,0003	0.0120	0.0001
0.2	0.0551	0.0542	0,0009	0.0577	0,0026
0.3	0.1374	0.1380	0,0006	0.1349	0,0025
0.4	0.2606	0.2601	0,0005	0.2632	0,0026
0.5	0.4179	0.4174	0,0005	0.4236	0,0057
0.6	0.5926	0.5837	0,0089	0.5916	0,001
0.7	0.7604	0.7615	0,0011	0.7613	0,0009
0.8	0.8948	0.8919	0,0029	0.8938	0,001
0.9	0.9758	0.9777	0,0019	0.9774	0,0016
1	1.0	1.0	0	1.0	0

Из таблицы видно, что разница между результатами, полученными каждым из моделирований и полным перебором, не превышает $\varepsilon = 0.01$.

Сравнительный график представлен ниже.

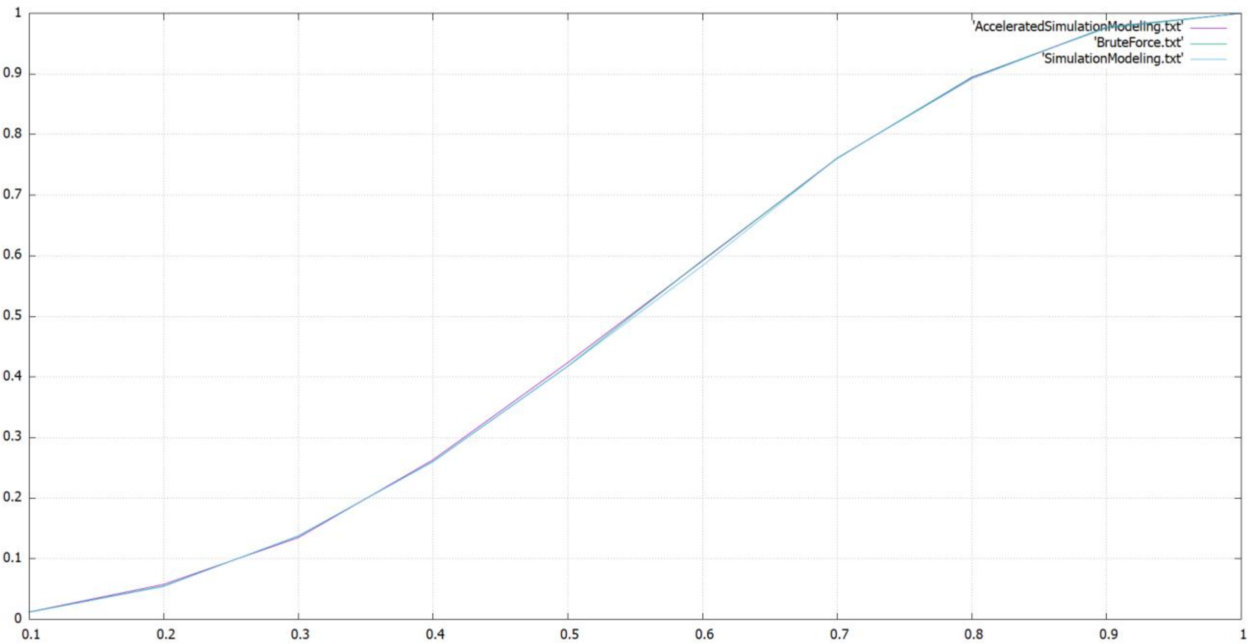

График 3 - Сравнение результатов

Для сравнения имитационного моделирования с ускоренным введем N' – число итераций в ускоренном моделировании, которые были выполнены без ускорения. Тогда выигрыш можно оценить как зависимость $\frac{N}{N'}(p)$. В ходе моделирования была получена таблица, представленная ниже. График такой зависимости представлен ниже. Из графика видно, что ускоренное моделирование дает выигрыш.

Таблица 4 - Выигрыш

p	N	N'	$\frac{N}{N'}$
0.1	22500	5085	4,424779
0.2	22500	12703	1,771235
0.3	22500	18096	1,243369
0.4	22500	20766	1,083502
0.5	22500	21616	1,040896
0.6	22500	20847	1,079292
0.7	22500	18098	1,243231
0.8	22500	12766	1,762494
0.9	22500	4999	4,5009

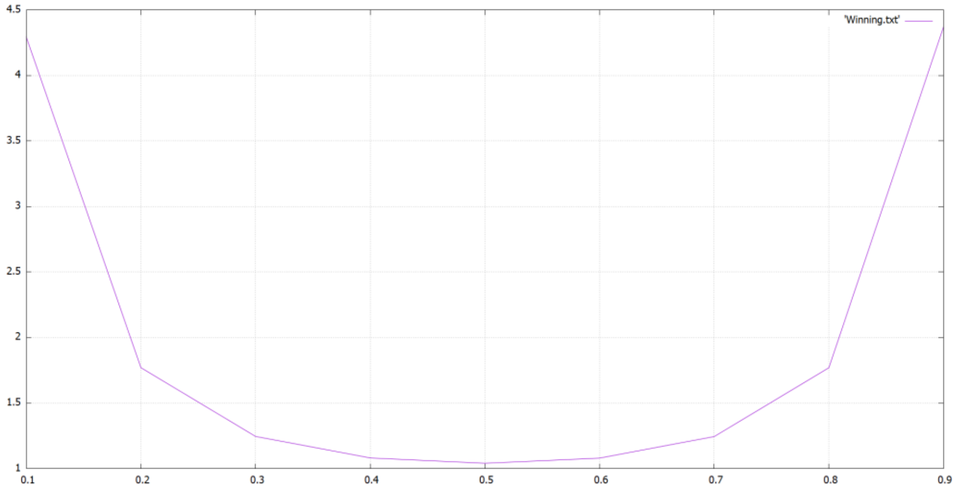

График 4 – Выигрыш

Листинг программы

```
package com.suai;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class Graph2 {

  private int n;
  private int l;
  private int v1;
  private int v2;
  private double p;
  private int N;
  private double e;
  private ArrayList<Pair> edgeList = new ArrayList<>();
  private boolean[] visited;
  private double probability;

  public Graph2() {
    n = 7;
    l = 9;
    v1 = 1;
    v2 = 4;
    p = 0.1;
    e = 0.01;
    N = (int) Math.ceil(2.25 / (e * e));
    visited = new boolean[n + 1];
    edgeList.add(new Pair(1, 2));
```

```
  edgeList.add(new Pair(1, 3));
  edgeList.add(new Pair(2, 3));
  edgeList.add(new Pair(2, 5));
  edgeList.add(new Pair(3, 4));
  edgeList.add(new Pair(4, 5));
  edgeList.add(new Pair(4, 7));
  edgeList.add(new Pair(5, 6));
  edgeList.add(new Pair(6, 7));
}

// очищение массива посещенных вершин
public void clearVisited() {
  for (int i = 0; i < visited.length; i++) {
    visited[i] = false;
  }
}

// генерация двоичного вектора ребер
public void createSubgraph(ArrayList<Pair> subEdgeList) {
  subEdgeList.clear();
  for (int i = 0; i < l; i++) {
    double rank = Math.random();
    if (rank <= p) { // есть ребро
      System.out.print("1 ");
      subEdgeList.add(edgeList.get(i));
    } else {
      System.out.print("0 ");
    }
  }
```

```
  System.out.print("Edge list: ");
  for (int i = 0; i < subEdgeList.size(); i++) {
    System.out.print(subEdgeList.get(i) + " ");
  }
}

// проверка связности вершин
public boolean isConnect(int v1, int v2, boolean[] visited, ArrayList<Pair> subEdgeList) {
  if (v1 == v2) {
    return true;
  }
  visited[v1] = true;
  for (int i = 0; i < subEdgeList.size(); i++) {
    int v = 0;
    if (subEdgeList.get(i).first() == v1) {
      v = subEdgeList.get(i).second();
    }
    if (subEdgeList.get(i).second() == v1) {
      v = subEdgeList.get(i).first();
    }
    if (v != 0 && !visited[v]) {
      if (isConnect(v, v2, visited, subEdgeList)) {
        return true;
      }
    }
  }
  return false;
}
```

```
public void printToFile(String filename) {
  try {
    FileWriter file = new FileWriter(filename, true);
    StringBuilder str = new StringBuilder();
    str.append(p).append(" ").append(probability).append("\n");
    file.write(str.toString());
    file.flush();
  } catch (IOException exception) {
    System.out.println(exception.getMessage());
  }
}

public void printWinningToFile(String filename, int newNum) {
  try {
    FileWriter file = new FileWriter(filename, true);
    StringBuilder str = new StringBuilder();
    if (newNum != N) {
      str.append(p).append(" ").append(((double)N / (N - newNum))).append("\n");
      file.write(str.toString());
      file.flush();
    }
  } catch (IOException exception) {
    System.out.println(exception.getMessage());
  }
}


public void simulationModeling() {
  ArrayList<Pair> subGraphEdge = new ArrayList<>();
```

```
  while (p <= 1) {
    int sNum = 0; // число подходящих подграфов
    int k = 0; // общий счетчик итераций для фиксированного p
    for (; k < N; k++) {
      createSubgraph(subGraphEdge);
      boolean isConnected = isConnect(v1, v2, visited, subGraphEdge);
      System.out.println(isConnected);
      if (isConnected) {
        sNum++;
      }
      clearVisited();
    }
    probability = ((double) sNum / N);
    printToFile("SimulationModeling.txt");
    System.out.println("S = " + sNum);
    System.out.println("N = " + k);
    p += 0.1;
  }
}

public void acceleratedSimulationModeling() {
  int lMin = 2;
  int lMax = l - 2;
  ArrayList<Pair> subGraphEdge = new ArrayList<>();
  while (p <= 1) {
    int sNum = 0; // число подходящих подграфов
    int k = 0; // общий счетчик итераций для фиксированного p
    int newNum = 0; // число упрощенияй
    for (; k < N; k++) {
```

```
      createSubgraph(subGraphEdge);
      if (subGraphEdge.size() < lMin) {
        newNum++;
        continue;
      }
      if (subGraphEdge.size() > lMax) {
        sNum++;
        newNum++;
        continue;
      }
      boolean isConnected = isConnect(v1, v2, visited, subGraphEdge);
      System.out.println(isConnected);
      if (isConnected) {
        sNum++;
      }
      clearVisited();
    }
    probability = ((double) sNum / N);
    printToFile("AcceleratedSimulationModeling.txt");
    // вывод в файл числа итераций с полными шагами и общего числа итераций
    printWinningToFile("Winning.txt", newNum);
    System.out.println("S = " + sNum);
    System.out.println("N = " + k);
    System.out.println("New N = " + (N - newNum));
    p += 0.1;
  }
 }
}
```