Оглавление

Цель работы

Изучить методы имитационного моделирования и ускоренного имитационного моделирования для решения задачи о определении вероятности связности двух вершин в случайном графе.

Исходные данные

Граф изображен ниже:

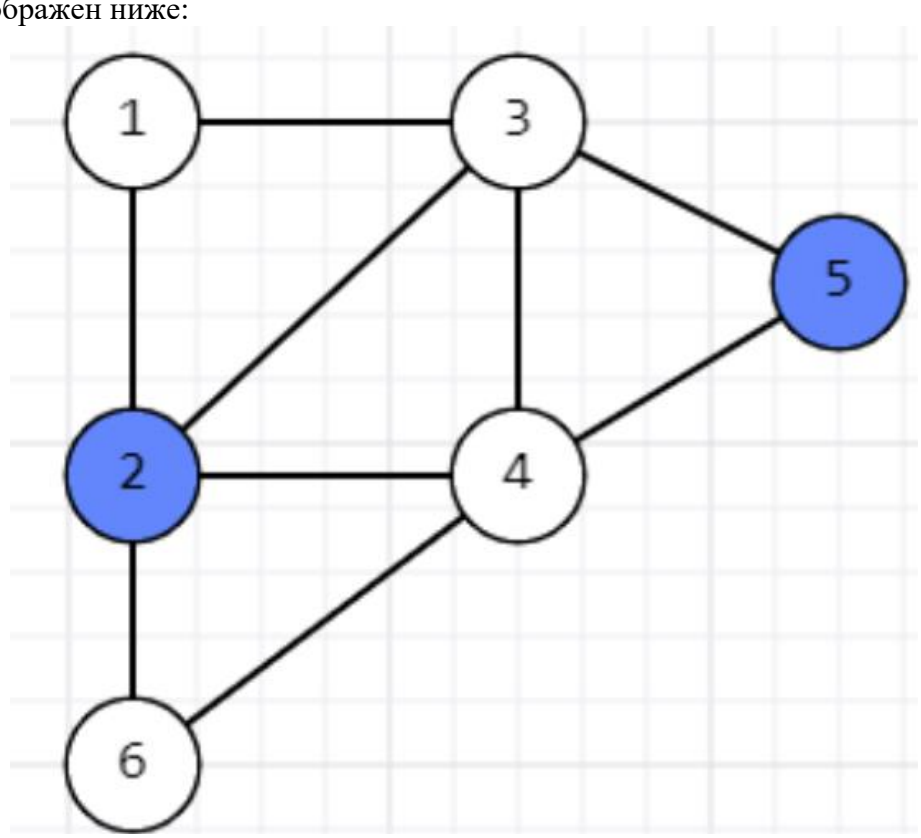

Рисунок 1. Исходный граф

Метод имитационного моделирования

Используется метод имитационного моделирования для получения оценки вероятности пути между двумя вершинами. Для этого воспользуется следующий алгоритмом моделирования:

Рисунок 2. Алгоритм имитационного моделирования.

Доверительный интервал ε = 0.01.

Число экспериментов $N = \frac{2.25}{\varepsilon^2} = \frac{2.25}{0.01^2} = 22500$

Сравним результаты, полученные при полном переборе и в результате имитационного моделирования:

P	Полный перебор	Имитационное моделирование	разница
0	0	0	0
0.1	0,023429332	0,022888889	0,000540443
0.2	0,102050304	0,104311111	-0,002260807
0.3	0,235075896	0,238622222	-0,003546326
0.4	0,405434368	0,4088	-0,003365632
0.5	0,5859375	0,583066667	0,002870833
0.6	0,748380672	0,749155556	-0,000774884
0.7	0,872369344	0,872666667	-0,000297323
0.8	0,950747136	0,950755556	-8,41956E-06
0.9	0,989307108	0,988666667	0,000640441
1	1	1	0

Как видно, оценка вероятности существования пути отличается от вероятности полного перебора не более чем на ±ε, что свидетельствует о правильности выполнения

имитационного моделирования.

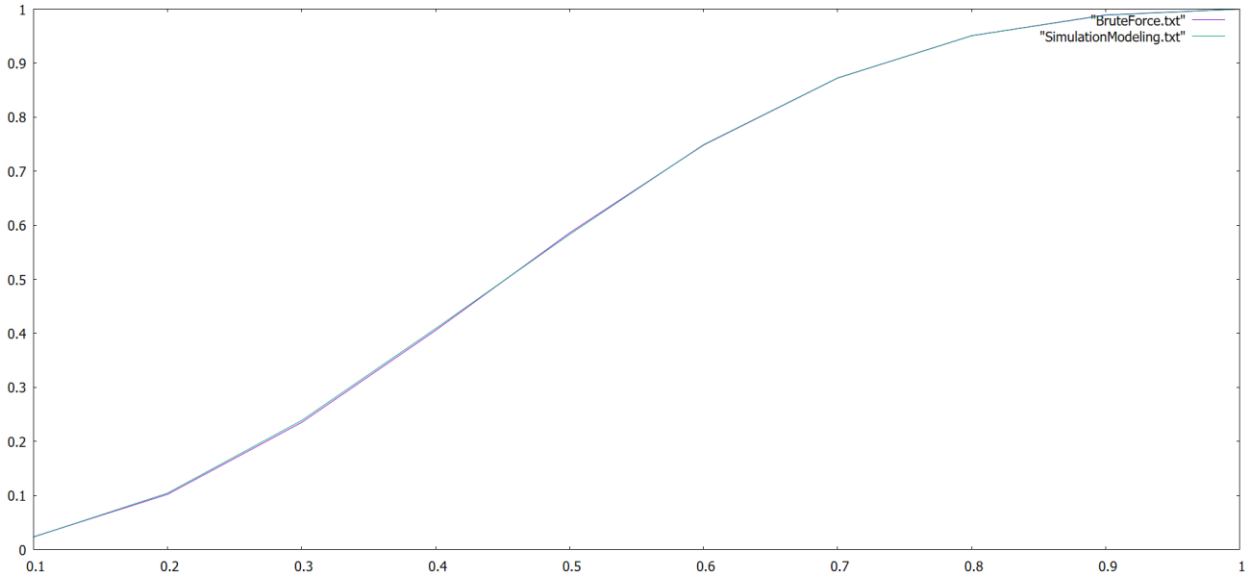

Рисунок 3. Графики полного перебора и имитационного моделирования

Графики подтверждают верность выполнения моделирования.

Метод ускоренного имитационного моделирования

Используется метод ускоренного имитационного моделирования для получения оценки вероятности пути между двумя вершинами. Для этого используется алгоритм ускорения за счет снижения временных затрат на проведение некоторых экспериментов:

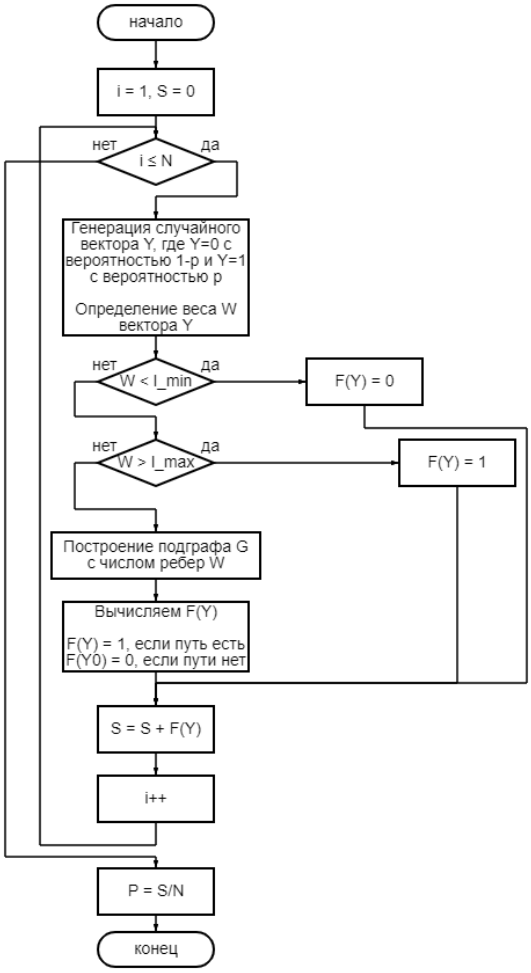

Рисунок 4. Алгоритм ускоренного имитационного моделирования.

Доверительный интервал ε = 0.01.

Число экспериментов $N = \frac{2.25}{\varepsilon^2} = \frac{2.25}{0.01^2} = 22500$

Для исходного графа $l_{min} = 2$, так как при количестве ребер < 2 подграф не будет иметь пути между двумя необходимыми вершинами, а $l_{max} = 9 - 2 = 7$, так как при числе ребер > 7 подграф всегда будет иметь путь между двумя рассматриваемыми вершинами.

P	Полный перебор	Имитационное моделирование	Ускоренное имитационное моделирование	Разница между ускоренным моделированием и полным перебором
0	0	0	0	0
0.1	0,023429332	0,022888889	0,023422222	7,10978E-06
0.2	0,102050304	0,104311111	0,107422222	-0,005371918
0.3	0,235075896	0,238622222	0,232844444	0,002231452
0.4	0,405434368	0,4088	0,398666667	0,006767701
0.5	0,5859375	0,583066667	0,584044444	0,001893056
0.6	0,748380672	0,749155556	0,745066667	0,003314005
0.7	0,872369344	0,872666667	0,873288889	-0,000919545
0.8	0,950747136	0,950755556	0,9492	0,001547136
0.9	0,989307108	0,988666667	0,989511111	-0,000204003
1	1	1	1	0

Как видно, оценка вероятности существования пути отличается от вероятности полного перебора не более чем на ±ε, что свидетельствует о правильности выполнения ускоренного имитационного моделирования.

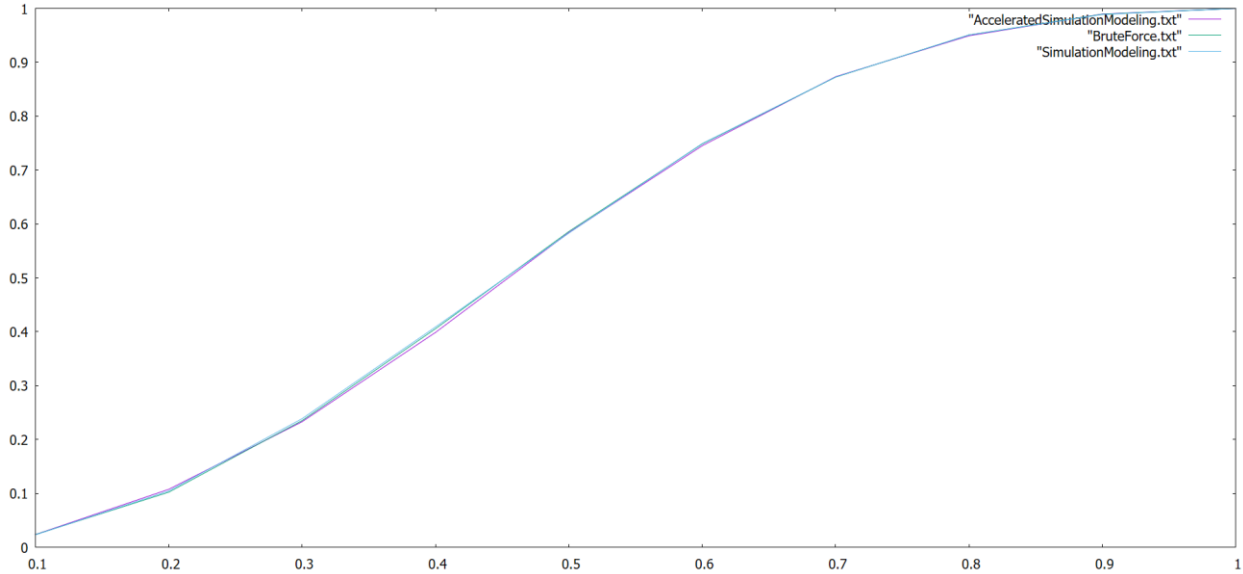

Рисунок 5. Графики полного перебора, имитационного моделирования и ускоренного имитационного моделирования.

Графики опять же подтверждают верность выполнения моделирования.

Выигрыш ускоренного имитационного моделирования.

Построим график зависимости отношения $\frac{N}{N\prime}$ от вероятности появления ребра, где N – число экспериментов имитационного моделирования, N’ – число экспериментов ускоренного имитационного моделирования.

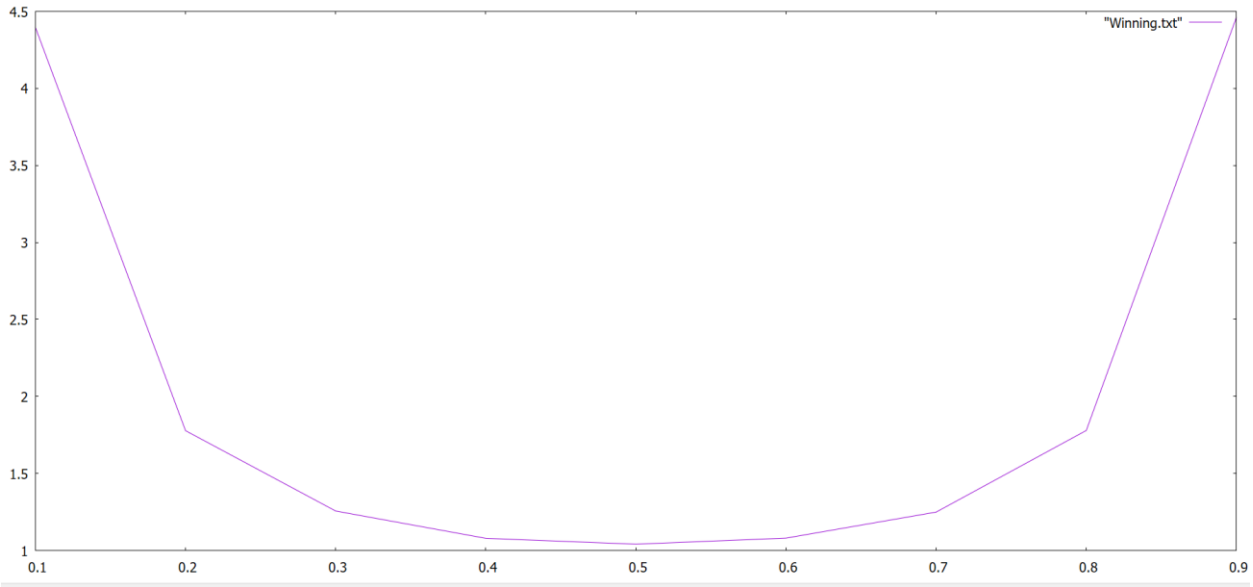

Рисунок 6. График выигрыша ускоренного имитационного моделирования.

Из графика видно, что наибольший выигрыш ускоренное имитационное моделирование дает при вероятности существования ребра близкой либо к 0, либо к 1. По мере приближения от границ к вероятности равной ½ выигрыш будет заметно уменьшаться.

Вывод

В ходе выполнения работы, были изучили методы имитационного моделирования и ускоренного имитационного моделирования; определено число испытаний, необходимое для заданного доверительного интервала.

Также было выяснено, что ускоренное имитационное моделирование дает больший выигрыш при вероятности существования ребра близкой к границам (то есть близкой к 0 или 1), так как при вероятности ребра p, близкой к 0, число ребер у подграфов чаще будет меньше, чем l_{min}, а при вероятности ребра p, близкой к 1, число ребер у подграфов чаще будет больше, чем l_{max} .

Листинг программы

```
package src.Graph;

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 = 6;
```

```
  l = 9;
  v1 = 2;
  v2 = 5;
  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, 4));
  edgeList.add(new Pair(2, 6));
  edgeList.add(new Pair(3, 4));
  edgeList.add(new Pair(3, 5));
  edgeList.add(new Pair(4, 5));
  edgeList.add(new Pair(4, 6));
}

// 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 {
    filename = "files\\" + filename;
    FileWriter file = new FileWriter(filename, true);
    StringBuilder str = new StringBuilder();
    str.append(p).append(" ").append(probability).append("\n");
    file.write(str.toString());
    file.flush();
    System.out.println("File " + filename + " was wrote");
  } catch (IOException exception) {
    System.out.println(exception.getMessage());
  }
}

public void printWinningToFile(String filename, int newNum) {
  try {
    filename = "files\\" + filename;
    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<>();
    p = 0.1;
    while (p <= 1) {
      int sNum = 0; // число подходящих подграфов
      int k = 0; // общий счетчик итераций для фиксированного p
      int newNum = 0; // число упрощений
      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;
    }
  }
}

package src.Graph;

public class Main {

    public static void main(String[] args) {
        Graph1 g1 = new Graph1();
        g1.bruteForce();
        Graph2 g2 = new Graph2();
        g2.simulationModeling();
        g2.acceleratedSimulationModeling();
    }
}
```