МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**ВЛИЯНИЕ ПАРАМЕТРОВ МНОГОСВЯЗНОЙ ОБЛАСТИ НА СВОЙСТВА AD HOC СЕТИ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.И.Шиян

 (подпись, дата) (инициалы, фамилия)

Факультет Компьютерных технологий и прикладной математики курс 3

Направление 020302 – «Фундаментальная информатика и информационные технологии»

Научный руководитель, проф.,

д-р физ.-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И.Миков

(подпись, дата) (инициалы, фамилия)

Нормоконтролер, ст. преп.,

канд. техн. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Е.Полупанова

(подпись, дата) (инициалы, фамилия)

Краснодар 2018

СОДЕРЖАНИЕ

[Введение 3](#_Toc516135007)

[1 Математические модели сетей 6](#_Toc516135008)

[1.1 Ориентированные графы 6](#_Toc516135009)

[1.1.1 Способы задания графов 6](#_Toc516135010)

[1.1.2 Связность в орграфах 8](#_Toc516135011)

[2 Сведения, необходимые для реализации 10](#_Toc516135012)

[2.1 Работа с точками и отрезками на плоскости 10](#_Toc516135013)

[2.2 Характеристики вариации случайной величины 11](#_Toc516135014)

[3 Реализация алгоритмов 13](#_Toc516135015)

[3.1 Генерация графа 13](#_Toc516135016)

[3.2 Сравнительный анализ влияния параметров многосвязной области свойства ad hoc сети 16](#_Toc516135017)

[Заключение 21](#_Toc516135018)

[Список использованных источников 23](#_Toc516135019)

[Приложение А Реализация алгоритма генерации графов на языке С++ 24](#_Toc516135020)

[Приложение Б Реализация вычисления математических ожиданий и дисперсий максимальной степени вершины на языке С++ 28](#_Toc516135021)

[Приложение В Реализация вычисления математических ожиданий и дисперсий числа ребер на языке С++ 30](#_Toc516135022)

[Приложение Г Реализация вычисления математических ожиданий и дисперсий числа изолированных вершин на языке С++ 32](#_Toc516135023)

[Приложение Д Реализация вычисления вероятностей связности на языке С++ 34](#_Toc516135024)

[Приложение Е Реализация вычисления соотношений между числом ребер и вершин на языке С++ 37](#_Toc516135025)

# ВВЕДЕНИЕ

Компьютерные ad hoc сети – это беспроводные децентрализованные сети с n узлами, расположенными в некоторой ограниченной или неограниченной области S. Каждый узел характеризуется координатами (местоположением на поверхности или в пространстве). В каждом узле находятся передатчик и приемник сигналов. Уровень электромагнитного поля, создаваемого передатчиком, уменьшается при удалении от него. В области расположения сети имеется естественный электромагнитный шум некоторого уровня. Таким образом, если на некотором расстоянии R от передающего узла находится принимающий узел, то, возможно, что отношение сигнал/шум соответствует чувствительности стандартного приемника. При удалении меньшем, чем R, приемник способен принять и распознать сигнал, а при большем или равном расстоянии – нет (либо вообще не принимает сигнал, либо не может его декодировать и считает шумом).

В области расположения узлов ad hoc сети могут находиться препятствия (например, здания) распространению волн электромагнитного (в частности, видимого – светового диапазона) поля. Тогда, принимающий узел, находящийся в «тени» препятствия, не получает сигнал.

Математической моделью такой сети является так называемый геометрический граф. Он представляет собой геометрическую конфигурацию – объединение кругов заданного радиуса R. Если центры двух кругов (местоположения узлов) находятся, на расстоянии, меньшем R, то между этими центрами (вершинами графа) имеется ребро. В противном случае ребро отсутствует. Ребро отсутствует также в случае, когда препятствие перекрывает прямую видимость между узлами. Таким образом, геометрическая конфигурация порождает обыкновенный граф.

В курсовой работе рассматриваются плоские многосвязные области S расположения компьютерной сети, моделирующие городские кварталы. В целом область представляет собой квадрат со стороной L1. Внутри этого квадрата располагается m×m квадратов меньшего размера (со стороной L2). Между соседними квадратами имеются промежутки равной величины. Такие же промежутки имеются и между крайними квадратами и внешним. В совокупности промежутки образуют область расположения сети (т.е. внутри внешнего квадрата, но вне внутренних квадратов). Внутренние квадраты изображают препятствия.

Будем считать граф сети случайным, со случайными положениями узлов внутри описанной решетчатой области.

Задача КР состоит в разработке алгоритмов и программ для исследования графов ad hoc сетей в решетчатых областях (L1, L2, m), и параметров их связности.

Изменяемыми характеристиками в постановке задачи являются:

Количество n узлов сети, 2 ≤ n ≤ 50;

Отношение β = mL2/L1. При этом m, L1 и L2 должны выбираться таким образом, чтобы площадь решетчатой области не изменялась.

Величина радиуса R задается постоянной во всех экспериментах, R = 1.

Площадь решетчатой области выбирается после предварительных вычислительных экспериментов так, чтобы при изменениях n и β структурные характеристики (количество ребер, максимальная степень вершины, количество изолированных вершин) графов сетей и связанные с ними вероятностные оценки (математическое ожидание, дисперсия, вероятность связности) менялись в широких пределах (например, вероятности изменялись от 0,1 до 0,9 и т.п.), а не оставались предельно малыми или предельно большими.

Для связных графов требуется оценить вероятность их близости к типу графа «дерево», соотношение между количеством вершин и ребер.

При исследовании статистическими методами (имитационное моделирование) должно быть проведено достаточное количество экспериментов (не менее 105) для того, чтобы графики зависимостей статистических оценок параметров от размера графа n и от параметра области β были достаточно гладкими. Должны быть найдены формулы и/или статистические числовые оценки для параметров связности.

Результаты исследования должны быть представлены в виде формул и/или графиков, показывающих зависимости.

# 1 Математические модели сетей

## 1.1 Ориентированные графы

### 1.1.1 Способы задания графов

Задать граф означает описать множества его вершин и ребер и отношения инцидентности. Если граф канонический, то для описания его вершин и ребер достаточно их занумеровать [1].

Одним из способов задания графа является матрица инцидентности. Пусть $\{v\_{1}, v\_{2}, ..., v\_{n}\} = V$ – множество вершин, а $\{e\_{1}, e\_{2}, ..., e\_{m}\} = E$ – множество ребер.

Отношение инцидентности можно определить матрицей $||E\_{ij}||$, имеющей $i = \overbar{1, m}$ (от 1 до m), $j = \overbar{1, n}$. Столбцы соответствуют вершинам графа, а строки – ребрам. При этом, если ребро $e\_{i}$ инцидентно вершине $v\_{j}$, то $E\_{ij} = 1$ в противном случае $E\_{ij} = 0$.

Полученная матрица называется матрицей инцидентности (для неориентированных графов).

Если граф ориентированный, то в матрице инцидентности $E\_{ij} = -1$, если вершина $v\_{j}$ является началом ребра $e\_{i}$. $E\_{ij} = 1$, если вершина $v\_{j}$ – конец ребра $e\_{i}$. Если $e\_{i}$ является петлей, а $v\_{j}$ – инцидентная ей вершина, то $E\_{ij} = α$, где $α$ – некоторое число, не равное 0, 1, -1. В остальных случаях $E\_{ij} = 0$.

Другой способ задания ориентированного графа – задание с помощью списка ребер. В списке ребер каждая строка этого списка соответствует ребру, и в ней записаны номера вершин, инцидентных этому ребру. Для неориентированных графов вершины перечисляются в произвольном порядке, а для ориентированных первой указывается вершина начала ребра, а второй – его конца.

По списку ребер графа легко построить матрицу его инцидентности. Действительно, каждая строка этого списка соответствует строке матрицы с тем же номером. Для неориентированного графа строка списка ребер содержит номера элементов строки матрицы инцидентности (номера столбцов), равных единице. А для ориентированного графа первым в списке ребер указывается номер элемента строки матрицы, равного -1, а вторым – номер элемента, равного единице.

Отметим, что недостатком такого способа представления графа является возможность утраты информации об изолированных вершинах.

Граф также можно задать его матрицей смежности. Матрица смежности – это квадратная матрица $||δ\_{ij}||$, столбцам и строкам которой соответствуют вершины графа.

Для неориентированного графа элемент $δ\_{ij}$ равен количеству ребер с началом в i-ой вершине и концом в j-ой. Таким образом, для неориентированного графа его матрица смежности является симметричной: $δ\_{ij} = δ\_{ji}$.

Для ориентированного графа матрица смежности необязательно симметрична. Если же матрица смежности ориентированного графа симметрична, то для каждого ребра ориентированного графа имеется ребро, соединяющее те же вершины, но идущее в противоположенном направлении. Таким образом, ориентированный граф с симметричной матрицей смежности канонически соответствует неориентированному графу, имеющему ту же матрицу смежности.

Отметим, что по матрице смежности легко строится список ребер, однако матрица смежности, как и матрица инцидентности, содержит большое количество нулей, и хранение такой матрицы для больших графов нерационально.

Также граф можно задать списками смежности. Каждой вершине ставится в соответствие список, и в каждом списке перечислены вершины, к которым ведут ребра из данной вершины. Для программной реализации выбран этот способ задания графа.

### 1.1.2 Связность в орграфах

В связном графе, как известно, любые две вершины соединены простой цепью. В общем случае произвольный граф может оказаться несвязным, но состоящим из нескольких частей, каждая из которых является связным графом [2].

Компонентами связности графа называются максимальные связные подграфы этого графа. Максимальность здесь понимается как максимальность числа вершин и ребер, входящих в компоненты связности.

Иначе компоненту связности можно определить, как связный подграф $G\_{1}$ графа $G$ $G\_{1} ⊂ G$, у которого множество вершин $V\_{1}$ и множество ребер $E\_{1}$ являются подмножествами вершин и ребер графа $G$ $V\_{1} ⊂ V$, $E\_{1} ⊂ E$, называется компонентой связности, если он содержит все ребра из множества ребер E, начала и концы которых лежат в множестве $V\_{1}$, и множество $V\_{1}$ не может быть расширено так, чтобы граф $G\_{1}$ остался связным.

Таким образом, несвязный граф всегда можно представить, как объединение связных компонент. Эти компоненты можно рассматривать независимо, и поэтому во многих случаях можно без ограничения общности считать, что рассматриваемый граф связен.

Отметим, что связность – это одно из немногих понятий теории графов, которое не распространяется непосредственно с неориентированных графов на ориентированные, и связано это с тем, что в ориентированном графе отношение связности вершин несимметрично.

Пусть $G$ – ориентированный граф, $v'$ и $v''$ – его вершины.

Говорят, что две вершины $v'$ и $v''$ сильно связаны между собой, если существует путь из $v'$ в $v''$ и из $v''$ в $v'$.

Говорят, что вершины $v'$ и $v''$ односторонне связаны в ориентированном графе $G$, если существует путь либо из $v'$ в $v''$, либо из $v''$ в$ v'$.

Говорят, что две вершины $v'$ и $v''$ слабо связаны в ориентированном графе $G$, если они связаны в графе $G'$, полученном из $G$ отменой ориентации ребер.

Если все вершины в ориентированном графе сильно (односторонне, слабо) связаны, то ориентированный граф называется сильно (односторонне, слабо) связанным.

Отметим, что сильная связность влечет за собой одностороннюю, которая, в свою очередь, влечет за собой слабую связность. Обратное не верно.

Компоненты сильной связности ориентированного графа $G$ – это его максимальные сильно связанные подграфы.

# 2 Сведения, необходимые для реализации

## 2.1 Работа с точками и отрезками на плоскости

Наиболее простыми геометрическими объектами на плоскости являются точки и отрезки. Для начала напомним некоторые определения. [3]

Далее будем полагать, что на плоскости задана декартова прямоугольная система координат, базис имеет правую ориентацию.

Точка на декартовой плоскости задается двумя числами: ее абсциссой и ординатой. В дальнейшем точку будем отождествлять с ее радиус-вектором.

Отрезок с концами в точках $p1$ и $p2$ – это множество точек, представимых в виде $p = ap\_{1} + (1 - a)p\_{2}$, где $0 \leq a \leq 1$.

Важную роль во многих геометрических алгоритмах играет определение направления поворота одного вектора к другому. Для решения этой задачи введем понятие векторного произведения.

Векторным произведением $\left[\vec{A}×\vec{B}\right]$ двух векторов $\vec{A} = (Ax, Ay)$ и $\vec{B} = (Bx, By)$ назовем число, равное $Ax∙By - Bx∙Ay$.

Если сопоставить данное определение векторного произведения с тем, которое дается в курсах аналитической геометрии, то ответ на поставленный вопрос очевиден: если $\left[\vec{A}×\vec{B}\right] > 0$, то $A$ надо поворачивать против часовой стрелки, если $\left[\vec{A}×\vec{B}\right] < 0$, то A надо поворачивать по часовой стрелке, если $\left[\vec{A}×\vec{B}\right] = 0$, то векторы либо сонаправлены, либо направлены в противоположные стороны.

Теперь рассмотрим еще одну простую задачу. Пусть даны два отрезка. Необходимо ответить на вопрос, имеют ли они хотя бы одну общую точку. Первое приходящее на ум решение таково: провести через них прямые (найти их уравнения), найти их точку пересечения и проверить ее принадлежность каждому из отрезков. Однако реализация этого решения осложняется техническими трудностями и необходимостью рассматривать частные случаи (прямые могут быть параллельны или совпадать), к тому же придется использовать вещественную арифметику. Приведем решение, лишенное этих недостатков:

1. проверим пересечение ограничивающих прямоугольников для этих отрезков. Ограничивающим прямоугольником для некоторой фигуры F называется прямоугольник минимальной площади со сторонами, параллельными осям координат и содержащий фигуру F. Если ограничивающие прямоугольники не пересекаются, то отрезки также не пересекаются.
2. Теперь проверим для каждого из отрезков, пересекает ли он прямую, на которой лежит другой. Отрезок пересекает прямую, если его концы лежат по разные стороны от прямой или один из концов лежит на прямой. Последнее легко проверить, используя векторное произведение.

Это нужно для проверки пересечения отрезка, соединяющего две вершины с препятствиями. Если есть пересечение хотя бы с одним препятствием, то ребра между этими вершинами нет.

## 2.2 Характеристики вариации случайной величины

Основной наиболее употребляемой характеристикой центральной тенденции является математическое ожидание МХ случайной величины. [4]

Пусть Х – дискретная случайная величина, $p\_{i}=P\left(X=x\_{i}\right)$, $i=\overbar{1, \infty }$, $\sum\_{i=1}^{\infty }p\_{i}=1$, тогда:

 $MX=\sum\_{i=1}^{\infty }x\_{i}p\_{i}$, (1)

если ряд сходится абсолютно.

Характеристики вариации дают представления о степени отклонения случайной величины от центра группирования. Одной из характеристик вариации является дисперсия.

Дисперсией случайной величины X называют число, равное математическому ожиданию квадрата отклонения случайной величины от её математического ожидания:

 $DX=M\left(X-MX\right)^{2}$. (2)

Часто пользуются другой формулой:

 $DX=MX^{2}-M^{2}X=M\left(X^{2}\right)-\left(MX\right)^{2}$. (3)

# 3 Реализация алгоритмов

## 3.1 Генерация графа

Для хранения графа используется две структуры. Первая – вектор вершин tops, хранящий для каждой вершины пару чисел – координаты этих вершин на плоскости. Вторая – вектор списков смежности g, где для каждой вершины хранятся номера смежных с ней вершин. Также хранится список obstacles, хранящий отрезки, являющиеся сторонами маленьких квадратов. Отрезок хранится в виде пары точек, а каждая точка – в виде пары вещественных чисел – её координат на плоскости. Максимальное расстояние между вершинами хранится в виде константы R. Если расстояние между вершинами больше R, они не могут быть смежными.

Для генерации графа используются некоторые вспомогательные функции.

Функция belong получает на вход координаты двух точек на плоскости p1 и p2 и вещественное число a и проверяет, принадлежит ли первая точка квадратной области плоскости, левым нижним углом которой является вторая точка, а длиной стороны квадрата – вещественное число a. Чтобы проверить, принадлежит ли точка p1 указанной области, достаточно проверить, что обе её координаты больше соответствующей координаты точки p2, но меньше суммы этой координаты и числа a.

Функция obstacle проверяет, принадлежит ли вершина p требуемой решётчатой области. Вычисляется величина zn равная сумме длины маленького квадрата L2 и расстояния k между маленькими квадратами. Чтобы точка принадлежала решётчатой области, необходимо, чтобы, во-первых, она принадлежала квадрату с левым нижним углом в точке (0,0) и стороной L1 – это проверяется с помощью описанной выше функции, а во-вторых, чтобы для каждой координаты точки p целая часть от деления этой координаты на zn не была равна целой части от суммы этой же координаты и L2 на zn.

Пример модели ad hoc сети приведен на рисунке 1.



Рисунок 1 – Пример модели ad hoc сети

Функция intersect получает на вход два отрезка e1 и e2 (отрезок задаётся двумя точками, а каждая точка – двумя координатами) и возвращает true, если они пересекаются, иначе возвращает false.

Функция bind получает на вход две вершины и проверяет, являются ли они смежными. Сначала вычисляется расстояние между точками с помощью формулы $((x\_{2}-x\_{1})+(y\_{2}-y\_{1}))^{2}$, где x1, y1 – координаты первой вершины, x2, y2 – координаты второй вершины. Если это расстояние больше R, то вершины несмежны, иначе для всех элементов списка obstacles (элементы списка являются отрезками, список должен быть сгенерирован ранее описанной ниже процедурой generate\_obstacles) проверяется, не пересекается ли элемент с отрезком, соединяющих две входные вершины. Если найдено хотя бы одно пересечение, вершины несмежны. В противном случае вершины смежны.

Функция generate\_obstacles генерирует список отрезков, являющихся сторонами маленьких квадратов для данных L1, L2 и m. Для каждого квадрата вычисляются координаты его центра, и затем координаты четырёх его вершин, начиная от верхней правой и далее против часовой стрелки. Вершины хранятся в массиве a. Для каждой вершины в массиве создаётся отрезок, соединяющий эту вершину со следующей за ней в массиве вершине и помещается и список obstacles. Для последней вершины в массиве создаётся отрезок, соединяющий её с нулевой вершиной массива. На выходе получается список отрезков, являющихся сторонами маленьких квадратов.

Граф генерируется с помощью функции generate\_graph. Функция генерирует граф необходимой конфигурации. Число вершин задаётся глобальной переменной n, параметры L1 и L2 также задаются одноимёнными глобальными переменными. Число маленьких квадратов задаётся глобальной переменной m.

Генерация проходит в два этапа, каждый этап разделён на несколько шагов.

Первый этап – генерация вершин.

1. очищается вектор top, его размер устанавливается равным n – требуемому числу вершин;
2. номер текущей генерируемой вершины i устанавливается равным нулю (для удобства программирования вершины нумеруются с нуля);
3. если i > n, перейти на шаг 6;
4. генерируется пара случайных чисел – координаты i-й вершины, эта пара помещается в вектор на i-ю позицию;
5. проверяется, принадлежит ли вершина решётчатой области (для этого вызывается функция obstacle для этой вершины). Если принадлежит, i увеличивается на 1. Переход на шаг 3.

Таким образом, сгенерированные вершины, не попавшие в решётчатую область, отбрасываются, и цикл будет продолжаться до тех пор, пока не будет сгенерировано n вершин, принадлежащих решётчатой области.

Второй этап – генерация рёбер.

1. очищаются списки смежности g, количество списков устанавливается равным количеству вершин n;
2. для каждой пары вершин i, j с помощью функции bind проверяется, смежны ли они. Если вершины смежны, то между ними есть ребро – в список смежности вершины i добавляется вершина j, и наоборот.
3. Результатом работы процедуры будет граф g, хранящийся в виде списков смежности. Он и будет использоваться для дальнейших вычислений.

Текст программы, реализующей алгоритм генерации графов, представлен в приложении A.

## 3.2 Сравнительный анализ влияния параметров многосвязной области свойства ad hoc сети

Графики, показывающие зависимость между числом ребер в графе и математическим ожиданием от числа вершин и β = mL2/L1, приведены на рисунке 2.

Рисунок 2 – Графики зависимости между числом ребер в графе и математическим ожиданием от числа вершин и β = mL2/L1

Графики, показывающие зависимость между числом ребер в графе и дисперсией от числа вершин и β = mL2/L1, приведены на рисунке 3.

Рисунок 3 – Графики зависимости между числом ребер в графе и дисперсией от числа вершин и β = mL2/L1

Графики, показывающие зависимость максимальной степени вершины в графе и математическим ожиданием от числа вершин и β = mL2/L1, приведены на рисунке 4.

Рисунок 4 – Графики зависимости максимальной степени вершины в графе и математическим ожиданием от числа вершин и β = mL2/L1

Графики, показывающие зависимость максимальной степени вершины в графе и дисперсией от числа вершин и β = mL2/L1, приведены на рисунке 5.

Рисунок 5 – Графики зависимости максимальной степени вершины в графе и дисперсией от числа вершин и β = mL2/L1

Графики, показывающие зависимость числа изолированных вершин в графе и математическим ожиданием от числа вершин и β = mL2/L1, приведены на рисунке 6.

Рисунок 6 – Графики зависимости числа изолированных вершин в графе и математическим ожиданием от числа вершин и β = mL2/L1

Графики, показывающие зависимость числа изолированных вершин в графе и дисперсией от числа вершин и β = mL2/L1, приведены на рисунке 7.

Рисунок 7 – Графики зависимости числа изолированных вершин в графе и дисперсией от числа вершин и β = mL2/L1

Графики, показывающие зависимость вероятности связности от числа вершин и β = mL2/L1, приведены на рисунке 8.

Рисунок 8 – Графики зависимости вероятности связности от числа вершин и β = mL2/L1

Графики, показывающие зависимость соотношения числа ребер к числу вершин в графах от числа вершин и β = mL2/L1, приведены на рисунке 9.

Рисунок 9 – Графики зависимости соотношения числа ребер к числу вершин от числа вершин в графах и β = mL2/L1

Графики, показывающие зависимость соотношения числа ребер к числу вершин в связных графах от числа вершин и β = mL2/L1, приведены на рисунке 10.

Рисунок 10 – Графики зависимости соотношения числа ребер к числу вершин в связных графах от числа вершин и β = mL2/L1

# ЗАКЛЮЧЕНИЕ

В процессе работы статистическими методами были исследованы плоские многосвязные области расположения компьютерной сети, зависимости структурных характеристик графов сетей и связанных с ними вероятностными оценками от числа вершин и β = mL2/L1:

1. при увеличении числа вершин математические ожидания и дисперсии максимальной степени вершины и числа ребер в графе увеличиваются;
2. при увеличении числа вершин математическое ожидание и дисперсия числа изолированных вершин в графе уменьшаются;
3. при уменьшении числа вершин математические ожидания и дисперсии максимальной степени вершины и числа ребер в графе уменьшаются;
4. при уменьшении числа вершин математическое ожидание и дисперсия числа изолированных вершин в графе увеличиваются;
5. при увеличении β математические ожидания и дисперсии максимальной степени вершины и числа ребер в графе уменьшаются;
6. при увеличении β математическое ожидание и дисперсия числа изолированных вершин в графе увеличиваются;
7. при уменьшении β математические ожидания и дисперсии максимальной степени вершины и числа ребер в графе увеличиваются;
8. при уменьшении β математическое ожидание и дисперсия числа изолированных вершин в графе уменьшаются;
9. при одном из исследованных значений, а именно β = 0,174964 дисперсии максимальной степени вершины и числа ребер в графе меньше, чем дисперсия максимальной степени вершины и числа ребер при β = 0,394405, что может быть связано с тем, что размеры препятствий относительно размеров всей площадки малы и практически не влияют на конфигурацию графа;
10. при увеличении числа вершин вероятность связности увеличивается;
11. при уменьшении числа вершин вероятность связности уменьшается;
12. при увеличении β вероятность связности увеличивается;
13. при уменьшении β вероятность связности уменьшается;
14. при малом числе вершин вероятность близости связных графов к типу «дерево» больше, чем при большем числе вершин;
15. при большем β вероятность близости связных графов к типу «дерево» больше, чем при меньшем β.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Харари Ф. Теория графов. – М.: Мир, 1973, 300 с.
2. Хаггарти Р. Дискретная математика для программистов. Москва: Техносфера, 2003, 320 с.
3. Дискретная математика: алгоритмы. Вычислительная геометрия [Электронный ресурс].-URL: <http://rain.ifmo.ru/cat/view.php/theory/math/geometry-2005> (дата обращения 20 мая 2018).
4. Гмурман В.Е. Теория вероятностей и математическая статистика. Изд. 4-е, доп. Учеб. пособие для вузов. М. «Высш. школа», 1972, 368 с.

# ПРИЛОЖЕНИЕ А

# Реализация алгоритма генерации графов на языке С++

Файл GenerateGraph.cpp:

#include <iostream>

#include <vector>

#include <list>

#include <fstream>

#include <string>

#include <random>

using namespace std;

const int R = 1;

double L1, L2, k;

int m, n, num\_vertices, num\_edges;

struct point {

 double x, y;

};

vector <point> tops;

struct edge {

 point org, dest;

};

vector <edge> obstacles;

vector <list <int>> g;

// функция проверки на принадлежность p1 квадратной области с длиной стороны a и левой нижней точкой p2

bool belong(point p1, point p2, double a)

{

 double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;

 return x2 < x1 && x1 < (x2 + a) && y2 < y1 && y1 < (y2 + a);

}

// функция проверки принадлежности точки p решётчатой области

bool obstacle(point p)

{

 double x = p.x, y = p.y;

 double zn = k + L2;

 point p1 = p, p2;

 p2.x = 0, p2.y = 0;

 return (int)(x / zn) != (int)((x + L2) / zn) && (int)(y / zn) != (int)((y + L2) / zn) && belong(p1, p2, L1);

}

//функция проверки отрезков e1 и e2 на предмет их пересечения

bool intersect(edge e1, edge e2)

{

 point p1 = e1.org, p2 = e1.dest, p3 = e2.org, p4 = e2.dest;

 double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y, x3 = p3.x, y3 = p3.y, x4 = p4.x, y4 = p4.y, v1, v2, v3, v4;

 v1 = (x4 - x3) \* (y1 - y3) - (y4 - y3) \* (x1 - x3);

 v2 = (x4 - x3) \* (y2 - y3) - (y4 - y3) \* (x2 - x3);

 v3 = (x2 - x1) \* (y3 - y1) - (y2 - y1) \* (x3 - x1);

 v4 = (x2 - x1) \* (y4 - y1) - (y2 - y1) \* (x4 - x1);

 return v1 \* v2 < 0 && v3 \* v4 < 0;

}

//функция проверки смежности вершин p1 и p2

bool bind(point p1, point p2)

{

 double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;

 if (sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2)) > R)

 return false;

 for (int i = 0; i < obstacles.size(); ++i)

 {

 edge e1, e2 = obstacles[i];

 e1.org = p1;

 e1.dest = p2;

 if (intersect(e1, e2))

 return false;

 }

 return true;

}

//генерация отрезков, являющихся сторонами препятствий

void generate\_obstacles()

{

 obstacles.clear();

 for (int i = 1; i <= m; ++i)

 for (int j = 1; j <= m; ++j)

 {

 double x0, y0;

 x0 = i \* k + (2 \* i - 1) \* (double)L2 / 2;

 y0 = j \* k + (2 \* j - 1) \* (double)L2 / 2;

 point a[4];

 for (int k = 1; k <= 4; ++k)

 {

 a[k - 1].x = x0 + ((k == 1 || k == 4) ? 1 : -1) \* (double)L2 / 2;

 a[k - 1].y = y0 + ((k == 1 || k == 2) ? 1 : -1) \* (double)L2 / 2;

 }

 for (int k = 0; k < 4; ++k)

 {

 edge e;

 e.org = a[k];

 e.dest = a[(k + 1) % 4];

 obstacles.push\_back(e);

 }

 }

}

//генерация графа

void generate\_graph()

{

 tops.clear();

 tops.resize(n);

 random\_device rnd;

//генерация вершин

 for (int i = 0; i < n; ++i)

 {

 tops[i].x = (double)rnd() / rnd.max() \* L1;

 tops[i].y = (double)rnd() / rnd.max() \* L1;

 //пока не попадет в решётчатую область – генерация новой вершины

 for (; obstacle(tops[i]);)

 {

 tops[i].x = (double)rnd() / rnd.max() \* L1;

 tops[i].y = (double)rnd() / rnd.max() \* L1;

 }

 }

 g.clear();

 g.resize(n);

 num\_vertices = n, num\_edges = 0;

//составление списка смежности

 for (int i = 0; i < n - 1; ++i)

 for (int j = i + 1; j < n; ++j)

//если вершины смежны, добавить ребро

 if (bind(tops[i], tops[j]))

 {

 g[i].push\_back(j);

 g[j].push\_back(i);

 num\_edges += 2;

 }

}

int main()

{

 const double S = 1.9; //площадь решётчатой области

 int num\_graphs = 1;

 for (L1 = 1.4; L1 < 2.4; L1 += 0.1)

 {

 double L2\_new;

 L2 = 5;

//L2 подбирается исходя из L1, площади и количества препятствий

 for (m = 1; L2\_new = sqrt(pow(L1, 2) - S) / m, L2\_new > 0.1 && (L2 - L2\_new) > 0.05; ++m)

 {

 L2 = L2\_new;

 k = (L1 - m \* L2) / (m + 1);

 generate\_obstacles();

 for (n = 2; n <= 50; ++n)

 {

 for (int i = 0; i < 50; ++i)

 {

 generate\_graph();

//запись графа в файл

 ofstream wr("graph\_" + to\_string(num\_graphs) + ".txt");

 wr << (double)m \* L2 / L1 << " " << n << endl;

 wr << num\_vertices << " " << num\_edges << endl;

 for (int v = 0; v < g.size(); ++v)

 for (int u : g[v])

 wr << v + 1 << " " << u + 1 << endl;

 wr.close();

 ++num\_graphs;

 }

 }

 }

 }

 system("pause");

 return 0;

}

# ПРИЛОЖЕНИЕ Б

# Реализация вычисления математических ожиданий и дисперсий максимальной степени вершины на языке С++

Файл MaxDegVertex.cpp:

#include <iostream>

#include <vector>

#include <map>

#include <fstream>

#include <string>

using namespace std;

vector <map <double, vector <double>>> v(51);

vector <map <double, double>> m(51), d(51);

int main()

{

 setlocale(LC\_ALL, "rus");

 for (int num\_graphs = 1; num\_graphs <= 112700; ++num\_graphs)

 {

 double betta, num\_vertices, num\_edges;

 ifstream rd("C:/Users/я/source/repos/course\_work/graph\_generator/graph\_" + to\_string(num\_graphs) + ".txt");

 rd >> betta;

 for (int i = 0; i < 2; ++i)

 rd >> num\_vertices;

 rd >> num\_edges;

 vector <int> deg(num\_vertices, 0);

 for (int i = 0; i < num\_edges; ++i)

 {

 int u, v;

 rd >> u >> v;

 u--;

 v--;

 ++deg[u];

 ++deg[v];

 }

 int max\_deg = 0;

 for (int i = 0; i < num\_vertices; ++i)

 if (deg[i] > max\_deg)

 max\_deg = deg[i];

 v[num\_vertices][betta].push\_back(max\_deg);

 rd.close();

 if (num\_graphs % 1000 == 0)

 cout << "Файл " << num\_graphs << " считан" << endl;

 }

 cout << "Файлы считаны" << endl;

 for (int i = 2; i <= 50; ++i)

 for (auto it = v[i].begin(); it != v[i].end(); ++it)

 {

 double sum = 0, sum\_2 = 0, num = 0;

 vector <double> v2 = it->second;

 for (int j = 0; j < v2.size(); ++j)

 {

 sum += v2[j];

 sum\_2 += pow(v2[j], 2);

 ++num;

 }

 double m\_ = (double)sum / num;

 double m\_2 = (double)sum\_2 / num;

 double d\_ = m\_2 - pow(m\_, 2);

 m[i][it->first] = m\_;

 d[i][it->first] = d\_;

 }

 cout << "Мат. ожидания и дисперсии посчитаны" << endl;

 ofstream wr("m.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = m[i].begin(); it != m[i].end(); ++it)

 wr << i << " " << it->first << " " << it->second << endl;

 wr.close();

 cout << "Мат. ожидания записаны" << endl;

 ofstream wr\_("d.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = d[i].begin(); it != d[i].end(); ++it)

 wr\_ << i << " " << it->first << " " << it->second << endl;

 wr\_.close();

 cout << "Дисперсии записаны" << endl;

 system("pause");

 return 0;

}

# ПРИЛОЖЕНИЕ В

# Реализация вычисления математических ожиданий и дисперсий числа ребер на языке С++

Файл NumEdges.cpp:

#include <iostream>

#include <vector>

#include <map>

#include <fstream>

#include <string>

using namespace std;

vector <map <double, vector <double>>> v(51);

vector <map <double, double>> m(51), d(51);

int main()

{

 setlocale(LC\_ALL, "rus");

 for (int num\_graphs = 1; num\_graphs <= 112700; ++num\_graphs)

 {

 double betta, num\_vertices, num\_edges;

 ifstream rd("C:/Users/я/source/repos/course\_work/graph\_generator/graph\_" + to\_string(num\_graphs) + ".txt");

 rd >> betta;

 for (int i = 0; i < 2; ++i)

 rd >> num\_vertices;

 rd >> num\_edges;

 v[num\_vertices][betta].push\_back(num\_edges);

 rd.close();

 if (num\_graphs % 1000 == 0)

 cout << "Файл " << num\_graphs << " считан" << endl;

 }

 cout << "Файлы считаны" << endl;

 for (int i = 2; i <= 50; ++i)

 for (auto it = v[i].begin(); it != v[i].end(); ++it)

 {

 double sum = 0, sum\_2 = 0, num = 0;

 vector <double> v2 = it->second;

 for (int j = 0; j < v2.size(); ++j)

 {

 sum += v2[j];

 sum\_2 += pow(v2[j], 2);

 ++num;

 }

 double m\_ = (double)sum / num;

 double m\_2 = (double)sum\_2 / num;

 double d\_ = m\_2 - pow(m\_, 2);

 m[i][it->first] = m\_;

 d[i][it->first] = d\_;

 }

 cout << "Мат. ожидания и дисперсии посчитаны" << endl;

 ofstream wr("m.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = m[i].begin(); it != m[i].end(); ++it)

 wr << i << " " << it->first << " " << it->second << endl;

 wr.close();

 cout << "Мат. ожидания записаны" << endl;

 ofstream wr\_("d.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = d[i].begin(); it != d[i].end(); ++it)

 wr\_ << i << " " << it->first << " " << it->second << endl;

 wr\_.close();

 cout << "Дисперсии записаны" << endl;

 system("pause");

 return 0;

}

# ПРИЛОЖЕНИЕ Г

# Реализация вычисления математических ожиданий и дисперсий числа изолированных вершин на языке С++

Файл NumIsolatedVertices.cpp:

#include <iostream>

#include <vector>

#include <map>

#include <fstream>

#include <string>

using namespace std;

vector <map <double, vector <double>>> v(51);

vector <map <double, double>> m(51), d(51);

int main()

{

 setlocale(LC\_ALL, "rus");

 for (int num\_graphs = 1; num\_graphs <= 112700; ++num\_graphs)

 {

 double betta, num\_vertices, num\_edges;

 ifstream rd("C:/Users/я/source/repos/course\_work/graph\_generator/graph\_" + to\_string(num\_graphs) + ".txt");

 rd >> betta;

 for (int i = 0; i < 2; ++i)

 rd >> num\_vertices;

 rd >> num\_edges;

 vector <int> deg(num\_vertices, 0);

 for (int i = 0; i < num\_edges; ++i)

 {

 int u, v;

 rd >> u >> v;

 u--;

 v--;

 ++deg[u];

 ++deg[v];

 }

 int num\_isolated\_vertices = 0;

 for (int i = 0; i < num\_vertices; ++i)

 if (0 == deg[i])

 ++num\_isolated\_vertices;

 v[num\_vertices][betta].push\_back(num\_isolated\_vertices);

 rd.close();

 if (num\_graphs % 1000 == 0)

 cout << "Файл " << num\_graphs << " считан" << endl;

 }

 cout << "Файлы считаны" << endl;

 for (int i = 2; i <= 50; ++i)

 for (auto it = v[i].begin(); it != v[i].end(); ++it)

 {

 double sum = 0, sum\_2 = 0, num = 0;

 vector <double> v2 = it->second;

 for (int j = 0; j < v2.size(); ++j)

 {

 sum += v2[j];

 sum\_2 += pow(v2[j], 2);

 ++num;

 }

 double m\_ = (double)sum / num;

 double m\_2 = (double)sum\_2 / num;

 double d\_ = m\_2 - pow(m\_, 2);

 m[i][it->first] = m\_;

 d[i][it->first] = d\_;

 }

 cout << "Мат. ожидания и дисперсии посчитаны" << endl;

 ofstream wr("m.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = m[i].begin(); it != m[i].end(); ++it)

 wr << i << " " << it->first << " " << it->second << endl;

 wr.close();

 cout << "Мат. ожидания записаны" << endl;

 ofstream wr\_("d.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = d[i].begin(); it != d[i].end(); ++it)

 wr\_ << i << " " << it->first << " " << it->second << endl;

 wr\_.close();

 cout << "Дисперсии записаны" << endl;

 system("pause");

 return 0;

}

# ПРИЛОЖЕНИЕ Д

# Реализация вычисления вероятностей связности на языке С++

Файл ProbabilityConnectivity.cpp:

#include <iostream>

#include <vector>

#include <map>

#include <list>

#include <fstream>

#include <string>

using namespace std;

vector <map <double, vector <double>>> v1(51), v2(51), v3(51);

vector <map <double, double>> p(51);

vector <map <double, double>> r(51);

vector <list <int>> g;

vector <bool> used;

int num\_used;

void dfs(int u)

{

 used[u] = true;

 ++num\_used;

 for (int v : g[u])

 if (!used[v])

 dfs(v);

}

int main()

{

 setlocale(LC\_ALL, "rus");

 for (int num\_graphs = 1; num\_graphs <= 112700; ++num\_graphs)

 {

 double betta, num\_vertices, num\_edges;

 ifstream rd("C:/Users/я/source/repos/course\_work/graph\_generator/graph\_" + to\_string(num\_graphs) + ".txt");

 rd >> betta;

 for (int i = 0; i < 2; ++i)

 rd >> num\_vertices;

 rd >> num\_edges;

 g.clear();

 g.resize(num\_vertices);

 for (int i = 0; i < num\_edges; ++i)

 {

 int u, v;

 rd >> u >> v;

 --u;

 --v;

 g[u].push\_back(v);

 g[v].push\_back(u);

 }

 used.clear();

 used.resize(num\_vertices, false);

 num\_used = 0;

 dfs(0);

 if (num\_used == num\_vertices)

 {

 v1[num\_vertices][betta].push\_back(num\_edges);

 /\*v2[num\_vertices][betta].push\_back((double)(num\_edges / 2) / (num\_vertices - 1));\*/

 }

 v3[num\_vertices][betta].push\_back(num\_edges);

 rd.close();

 if (num\_graphs % 1000 == 0)

 cout << "Файл " << num\_graphs << " обработан" << endl;

 }

 cout << "Файлы считаны" << endl;

 for (int i = 2; i <= 50; ++i)

 for (auto it = v1[i].begin(); it != v1[i].end(); ++it)

 p[i][it->first] = (double)v1[i][it->first].size() / v3[i][it->first].size();

 /\*for (int i = 2; i <= 50; ++i)

 {

 for (auto it = v2[i].begin(); it != v2[i].end(); ++it)

 {

 double sum = 0;

 for (int j = 0; j < v2[i][it->first].size(); ++j)

 sum += v2[i][it->first][j];

 r[i][it->first] = (double)sum/ v2[i][it->first].size();

 }

 }\*/

 cout << "Вероятности и отношения посчитаны" << endl;

 ofstream wr1("p.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = p[i].begin(); it != p[i].end(); ++it)

 wr1 << i << " " << it->first << " " << it->second << endl;

 wr1.close();

 /\*ofstream wr2("r.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = r[i].begin(); it != r[i].end(); ++it)

 wr2 << i << " " << it->first << " " << it->second << endl;

 wr2.close();\*/

 cout << "Вероятности и отношения записаны" << endl;

 system("pause");

 return 0;

}

# ПРИЛОЖЕНИЕ Е

# Реализация вычисления соотношений между числом ребер и вершин на языке С++

Файл Relation.cpp:

#include <iostream>

#include <vector>

#include <map>

#include <list>

#include <fstream>

#include <string>

using namespace std;

vector <map <double, vector <double>>> v1(51), v2(51), v3(51);

vector <map <double, double>> p(51);

vector <map <double, double>> r(51);

vector <list <int>> g;

vector <bool> used;

int num\_used;

void dfs(int u)

{

 used[u] = true;

 ++num\_used;

 for (int v : g[u])

 if (!used[v])

 dfs(v);

}

int main()

{

 setlocale(LC\_ALL, "rus");

 for (int num\_graphs = 1; num\_graphs <= 112700; ++num\_graphs)

 {

 double betta, num\_vertices, num\_edges;

 ifstream rd("C:/Users/я/source/repos/course\_work/graph\_generator/graph\_" + to\_string(num\_graphs) + ".txt");

 rd >> betta;

 for (int i = 0; i < 2; ++i)

 rd >> num\_vertices;

 rd >> num\_edges;

 g.clear();

 g.resize(num\_vertices);

 for (int i = 0; i < num\_edges; ++i)

 {

 int u, v;

 rd >> u >> v;

 --u;

 --v;

 g[u].push\_back(v);

 g[v].push\_back(u);

 }

 used.clear();

 used.resize(num\_vertices, false);

 num\_used = 0;

 dfs(0);

 if (num\_used == num\_vertices)

 {

 /\*v1[num\_vertices][betta].push\_back(num\_edges);\*/

 v2[num\_vertices][betta].push\_back((double)(num\_edges / 2) / (num\_vertices - 1));

 }

 v3[num\_vertices][betta].push\_back(num\_edges);

 rd.close();

 if (num\_graphs % 1000 == 0)

 cout << "Файл " << num\_graphs << " считан" << endl;

 }

 cout << "Файлы считаны" << endl;

 /\*for (int i = 2; i <= 50; ++i)

 for (auto it = v1[i].begin(); it != v1[i].end(); ++it)

 p[i][it->first] = (double)v1[i][it->first].size() / v3[i][it->first].size();\*/

 for (int i = 2; i <= 50; ++i)

 {

 for (auto it = v2[i].begin(); it != v2[i].end(); ++it)

 {

 double sum = 0;

 for (int j = 0; j < v2[i][it->first].size(); ++j)

 sum += v2[i][it->first][j];

 r1[i][it->first] = (double)sum/ v2[i][it->first].size();

 }

 }

 cout << "Вероятности и отношения посчитаны" << endl;

 /\*ofstream wr1("p.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = p[i].begin(); it != p[i].end(); ++it)

 wr1 << i << " " << it->first << " " << it->second << endl;

 wr1.close();\*/

 ofstream wr2("r.txt");

 for (int i = 2; i <= 50; ++i)

 for (auto it = r[i].begin(); it != r[i].end(); ++it)

 wr2 << i << " " << it->first << " " << it->second << endl;

 wr2.close();

 cout << "Вероятности и отношения записаны" << endl;

 system("pause");

 return 0;

}