**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Кроме очевидной задачи моделирования динамики социальных сетей, модели случайных графов [1] применимы к таким популярным сейчас практическим задачам, как поиск сообществ, максимизация распространения влияния, оценка безопасности и выносливости сложных сетей и другие. Они позволяют сравнивать качество предлагаемых алгоритмов на целых семействах случайных графов и сравнивать получаемые результаты с предсказанными теоретически.

Данное направление исследований переживает период интенсивного развития и привлекает внимание математиков, инженеров, специалистов по компьютерным наукам, социологов и других специалистов.

Данная бакалаврская работа представляет из себя исследование модели построения случайного графа, основа которого состоит из двух принципов: тройственного замыкания и предпочтительного соединения. Важность исследования достаточно высока, поскольку множество моделей построенных по такой же концепции, очень точно описывают поведение и характеристики актуальных веб-графов.

Целью работы является эмпирическое изучение модели построения случайного графа [2]. Для достижения этой цели ставятся следующие задачи:

- написать программу для генерации построения случайного графа с помощью предложенной модели в которой: на вход приходит некоторое количество графов $T$ и количество вершин в каждом графе $M$, на выход — рисунок сети состоящей из заданного количества графов и вершин;
- сравнить реализацию, использующую для представления графов матрицы смежности с реализацией, использующей библиотеку для построения графов и сетей;
- провести анализ результатов использования предложенной модели.

## 1 Постановка задачи

Большое количество систем во многих отраслях науки можно смоделировать с помощью механизма тройственное замыкание (triple closure), его используют в сложных сетях таких как социальные сети [3–6], компьютерные и мобильные сети, финансовые сети [7], а также метаболические сети некоторых организмов которые имеют логарифмически растущую среднюю геодезическую длину (кратчайший путь) [8]. Кроме того, социальные сети обычно демонстрируют высокую кластеризацию, или локальную транзитивность: если человек A знает B и C, то B и C, скорее всего, знают друг друга.

Стандартная модель тройственного замыкания (triple closure), предложенная Питтером Холмом и Бом Джун Кимом [2], представляет собой модификацию модели предпочтительного соединения (preferential attachment), которую принято называть «модель Барабаши—Альберт». Эта модель описывает ряд важных эмпирических закономерностей в поведении веб-графов, в последствии она получила большую популярность в этой среде.

Как и модель Барабаши—Альберт [9], модель тройственного замыкания генерирует безмасштабные сети [10]. Однако в отличие от модели Барабаши—Альберт, модель предложенная Питтером Холмом и Бом Джун Кимом создает сети с гораздо более высоким коэффициентом кластеризации по сравнению с реальными социальными сетями. [11] В данной модели вершина соединяется с уже существующей вероятностью, пропорциональной степени этой вершины (такая же как и в модели Барабаши—Альберт). Но для каждой из оставшихся ребер с вероятностью $p$ выбирается сосед вершины, с которой произошло соединение на прошлом этапе, и с вероятностью $1 - p$ этап пропускается, т.е выбирается вершина с помощью предпочтительного соединения (preferential attachment).

Модель генерирует графы с разными коэффициентами кластеризации в зависимости от $p$. Степенное распределение же подчиняется степенному закону с коэффициентами $-3$ для любого $p$. В данной работе рассматривается модель построения сети, которая основана на двух принципах тройственного замыкания (triple closure) и предпочтительного соединения (preferential attachment), исследования поведения модели является основной целью работы.

Выполняется следующий список выполнения задач:

— написать программу для генерации построения случайного графа с помощью предложенной модели в которой: на вход приходит некоторое количество графов $T$ и количество вершин в каждом графе $M$, на выход — рисунок сети состоящей из заданного количества графов и вершин;
— сравнить реализацию, использующую для представления графов матрицы смежности с реализацией, использующей библиотеку для построения графов и сетей;
— провести анализ результатов использования предложенной модели.

## 1.1 Детали реализации

Реализация модели построения случайного графа будет проводиться на языке программирования Python [12, 13].

Реализация модели осуществляется в двух вариантах:

— с использованием матрицы смежности, с представлением матрицы в виде двумерного массива;
— с использованием специальной библиотеки для представления графов и сетей.

### 1.1.1 `matplotlib.pyplot`

Библиотека `matplotlib.pyplot` состоит из множества модулей, они наполнены разными классами и функциями, которые иерархически связаны между собой. В программе же используется интерфейс `pyplot` т.к, в нем уже присутствуют готовые настройки рисунков и нам просто необходимо выбрать такой, какой нам в конкретный момент нужен.

### 1.1.2 `random`

Модуль `random` предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

Непосредственно в самом коде приложений будем активно использовать функцию `choice`. Данная функция может принимать один аргумент — список элементов: в этом случае функция выдаст случайный элемент списка с равновероятным выбором.

Когда функция `choice` принимает два аргумента, второй аргумент представляет список чисел, в котором столько же элементов, сколько в первом

аргументе. В этом случае она выдает элемент первого списка с вероятностью пропорциональной соответствующему элементу второго списка.

### 1.1.3 numpy

Библиотека numpy добавляет в Python поддержку больших многомерных массивов и матриц, вместе с большой коллекцией высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. В реализации использовались следующие функции библиотеки.

`full(shape =, fill_value =, dtype =)` — данная функция создает массив, заполненный тем же значением. Параметр shape определяет форму входного массива. формой массива является количество строк и столбцов. Параметр `fill_value` — значение которое будет использоваться в качестве отдельных элементов массива. Параметр *dtype* — позволяет указать тип данный элементов выходного массива.

reshape — функция меняет размерность массива.

### 1.1.4 networkx

Библиотека networkx предназначена для работы с графами и другими сетевыми структурами. Основными возможностями библиотеки являются:

- классы для работы с простыми, ориентированными и взвешенными графами;
- встроенные процедуры для создания графов базовых типов;
- визуализировать сети в виде 2D и 3D графиков;
- получение таких характеристик графа как степени вершин, высота графа, диаметр, радиус, длины путей, центр.

Библиотека networkx является очень производительной и может оперировать весьма большими сетевыми структурами, уровня графа с 10 миллионами узлов и 100 миллионами дуг между ними. В виду того, что он базируется на низкоуровневой структуре данных языка Python под названием «словарь-словарей», память расходуется эффективно, графы хорошо масштабируются, мало зависят от особенностей операционной системы в которой выполняется скрипт и отлично подходят для популярного на данный момент направления по анализу данных из социальных сетей и графов. Также в библиотеке реализовано большое количество типичных для работы над графами алгоритмов. Реализованы такие алгоритмы как нахождение кратчайшего пути, поиска в

высоту и ширину, кластеризация, нахождение изоморфизма графов и многое другое.

В работе были использованы следующие команды библиотеки.

`Graph()` — создание простого неориентированного графа. Дополнительные вершины между двумя узлами игнорируются, возможны узлы соединённые с самим собой.

`G.add_node(i)` — процедура для добавления в граф `G` вершины `i`.

`G.add_edge(i, j)` — процедура для добавления в граф `G` ребра между вершинами `i` и `j`.

`G.degree` — функция выдает вектор степеней вершин графа `G`.

`draw_circular(G, **options)` — процедура чертит граф `G` с круговой разметкой. Параметр `**options` предоставляет опции для отрисовки графа, с помощью которых можно изменить цвет вершин и ребер, размер, фон и многое другое.

## 2 Модели построения случайного графа

### 2.1 Элементы теории графов

Введем некоторые базовые понятия графов [14]. В дальнейшем будем использовать термины узел и вершина в качестве синонимов. Также будем поступать с терминами граф и сеть.

Определим граф $G$ как пару $G = (V, E)$, состоящую из множества узлов $V(G)$ и множества ребер $E(G) \subseteq V \times V$. Множество ребер $E(G)$ индуцирует симметричное бинарное отношение на $V(G)$, которое называется отношением смежности $G$. Узлы $i$ и $j$ смежные, если $(i, j) \in E(G)$.

Если $(i, j) \in E(G)$, то говорят, что вершины $i$ и $j$ инцидентны ребру $(i, j)$.

Степенью $d_i$ узла $i$ называется число инцидентных ему ребер. Обозначают

$$\deg(i) = d_i.$$

Граф может быть как ненаправленным или неориентированным, так и направленным или ориентированным. В случае направленного графа нужно различать степень $d_i^-$ и степень $d_i^+$ узла $i$.

Матрица смежности $A(G)$ графа $G$ представляет собой матрицу размерности $n \times n$, $n = |V(G)|$, в которой элемент $a_{ij}$ равен 1, если ребро $(i, j) \in E(G)$, в противном случае $a_{ij}$ равен 0.

Для неориентированного графа матрица смежности симметрична, т. е.

$$a_{ij} = a_{ji} \qquad \forall i, j \in V(G).$$

Пример простого ориентированного графа на четырех узлах приведен на Рис. 1.

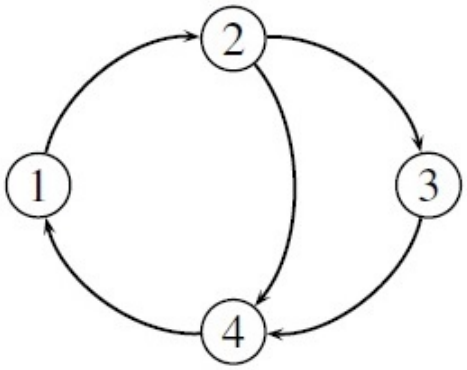


Рисунок 1 – Граф, содержащий 4 узла и 5 ребер

Матрица смежности, соответствующая графу на Рис. 1, примет вид

$$A(G) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Подграф $G'$ графа $G$ является графом подмножеств узлов $V(G') \subseteq V(G)$ и ребер $E(G') \subseteq E(G)$.

Блуждание — это чередующийся список $\{v_0, (v_0, v_1), v_1, \ldots, v_{k-1}, (v_{k-1}, v_k), v_k\}$ узлов и ребер. Тропа — это прогулка без повторяющихся ребер. Путь — это прогулка без повторяющегося узла. Кратчайший путь между двумя узлами также известен как геодезическое расстояние. Если конечные точки тропы одинаковы (закрытая тропа), то ее называют контуром. Контур без повторяющегося узла называется циклом. В частности, $C_n$ обозначает цикл на $n$ узлах. Цикл также является контуром, но контур не обязательно является циклом. Примеры простых графов показаны на рисунках 2–4.

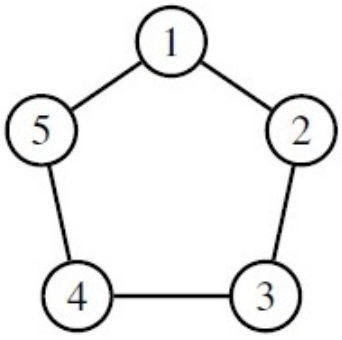


Рисунок 2 – Циклический граф $C_5$


Рисунок 3 – Цепь $P_5$

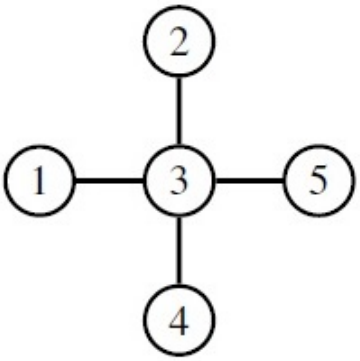


Рисунок 4 – Звезда $K_{1,4}$

Граф $G$ является связанным, если существует путь, соединяющий каждую пару узлов. В противном случае граф $G$ является несвязанным.

Граф называется полным, если каждый узел связан с любым другим узлом. $K_n$ обозначает полный граф на $n$ узлах.

Степень $k$ матрицы смежности связана с блужданиями длины $k$ в графе. В частности, $(A^k)_{ij}$ дает число переходов длины $k$ от узла $i$ до узла $j$.

Для каждого узла $i$ коэффициент локальной кластеризации $C_l(i)$ определяется как доля пар соседей $i$, которые сами являются соседями.

Число возможных связей между соседями узла $i$ равно $\dfrac{\deg(i)(\deg(i)-1)}{2}$. Таким образом, получаем

$$C_l(i) = \frac{|\{(j,k) \in E(G) : (i,j) \in E(G) \wedge (i,k) \in E(G)\}|}{\dfrac{\deg(i)(\deg(i)-1)}{2}}.$$

Глобальный коэффициент кластеризации $C_l$ имеет вид $C_l = \frac{1}{n}\sum\limits_{i=1}^{n} C_l(i)$. Высокий коэффициент кластеризации $C_l$ означает (на языке социальных сетей), что двое из ваших друзей, вероятно, также будут друзьями друг другу. Это также указывает на высокую избыточность сети. Для полного графа $K_n$ это тривиально $C_l = 1$.

## 2.2 Случайные графы

Случайные графы можно описать просто распределением вероятности или случайным процессом, создающим эти графы. Теория случайных графов находится на стыке теории графов и теории вероятностей. Теория случайных графов стала интенсивно развиваться с конца 1959 года после публикации статьи Эрдеша—Реньи [15] об эволюции случайных графов. В этой модели все ребра появляются случайно и независимо с одинаковой вероятностью $p$ и под эволюцией понимается изменение свойств графов с ростом вероятности $p$. Оказалось, что в некоторых значениях $p$ происходит так называемый фазовый переход и свойства графа кардинально меняются. В этом направлении было получено много интересных и глубоких результатов. Однако, в начале 2000-х выяснилось, что модель Эрдеша—Реньи плохо описывает реальные графы, возникающие в различных областях, в частности в графы социальных сетей.

Это породило много новых исследований математических моделей случайных графов. В задачах описания динамики социальных сетей основное значение имеет правильный выбор математической модели. На данный момент известно множество моделей случайных графов и безмасштабных (scale-free) сетей, некоторые из которых показали удовлетворительные результаты при сравнении с экспериментальными данными. Модели социальных сетей можно разделить на три класса:

- модели случайных графов (модель Эрдеша—Реньи и др.);
- простейшие модели безмасштабных сетей (модель Боллобаша—Риордана и др.);
- гибкие модели безмасштабных сетей (модель Чунг—Лу и др.).

В следующих разделах приводится описание некоторых из моделей, известных на сегодняшний день.

## 2.3 Модель Эрдеша—Реньи

Зафиксируем натуральное число $n$ и рассмотрим множество $V = \{1, \ldots, n\}$. Таким образом мы задали множество вершин случайного графа. Зададим полный граф $K_n$ на множестве вершин $V$. Пронумеруем ребра $K_n : e_1, \ldots, e_N$, где $N = \binom{n}{2}$. Зададим некоторе $p \in [0, 1]$ и будем выбирать ребра из множества $\{e_1, \ldots, e_n\}$ согласно схеме Бернулли. Таким образом мы получили случайный граф $G = (V, E)$.

Формально выражаясь, мы имеем вероятностное пространство $G(n,p) = (\Omega_n, F_n, P_{n,p})$ в котором:

$$|\Omega_n| = 2^N, \qquad P_{n,p}(G) = p^{|E|} q^{\binom{n}{2}-|E|}.$$

Таким образом в модели Эрдеша—Реньи каждое ребро независимо от других ребер входит в случайный граф с вероятностью $p$. Модель Эрдеша—Реньи на данный момент является самой изученной моделью случайных графов.

### 2.4 Модель Барабаши—Альберт

В своих статьях [9,10,16] авторы заметили следующие закономерности в веб-графе (графе, вершинами которого являются сайты, а ребра соответствуют ссылкам):

- веб-граф разрежен (на $n$ вершинах у него $mn$ ребер, $m \in \mathbb{N}$);
- веб-граф подчиняется феномену «малого мира» (его диаметр примерно равен 5–7);
- он подчиняется степенному закону:

$$\frac{|\{v : \deg(v) = d\}|}{n} \approx \frac{c}{d^\lambda},$$

где $\deg(v)$ — степень вершины $v$, $c = const$, $\lambda \sim 2.1$.

На основании своих наблюдений авторы ввели понятие предпочтительного присоединения (preferential attachment). Рассмотрим процесс генерации графа. На $n$-ом шагу мы добавляем новую вершину $n$ с $m$ ребрами, инцидентными ей, причем вероятность ребра к вершине $i$ пропорциональна степени вершины $i$.

$$P(\text{ребро из } n \text{ в } i) = \frac{\deg(i)}{\sum\limits_j \deg(j)}.$$

Основных проблем со спецификацией модели Барабаши—Альберт две. Во-первых, результирующий граф зависит от начального параметра $m$. Например, при $m = 1$ модель Барабаши—Альберт описывает генерацию дерева, если начальный граф — тоже дерево. Если начальный граф несвязный, то и все последующие тоже будут таковыми. Во-вторых, трудность с предпочтительным присоединением заключается в случайном выборе вершин (если $m \geqslant 2$), к которым присоединится новая вершина.

### 2.5 Модель Боллобаша–Риордана

Модель была предложена в [17, 18].

Построим последовательность случайных графов $\{G_1^n\}$, в которой у графа с номером $n$ число вершин и ребер равно $n$. Преобразуем ее в последовательность $\{G_k^n\}$ , в которой у графа с номером $n$ число вершин равно $n$, а число ребер равно $kn$, $k \in \mathbb{N}$.

Пусть $\{G_1^1\} = (\{1\}, \{(1,1)\})$. Предположим, что граф $\{G_1^{n-1}\}$ уже построен и у него $n-1$ ребер и $n-1$ вершин. Добавим вершину $n$ и ребро $(n,i)$, у которого $i \in \{1, ..., n\}$. Ребро $(n,n)$ появится с вероятностью $\dfrac{1}{2n-1}$, ребро $(n,i)$ — с вероятностью $\dfrac{\deg(i)}{2n-1}$, где $\deg(i)$ — степень вершины $i$ в графе $G_1^{n-1}$. Распределение вероятностей задано корректно, т.к.

$$\sum_{i=1}^{n-1} \frac{\deg(i)}{2n-1} + \frac{1}{2n-1} = 1.$$

Таким образом, граф $G_1^n$ построен, и он удовлетворяет принципу предпочтительного присоединения. Теперь перейдем к $G_1^{kn}$, у которого $kn$ вершин и $kn$ ребер. Делим множество его вершин на последовательные куски размера $k$:

$$\{1, \ldots, k\}, \{k+1, \ldots, 2k\}, \ldots, \{k(n-1)+1, \ldots, kn\}.$$

Каждый кусок примем за новую вершину, а ребра сохраним (ребра внутри куска становятся кратными петлями, ребра между разными кусками — кратными ребрами.) Оказалось, что модель Боллобаша–Риордана довольно хорошо сходится с эмпирическими данными. За время изучения этой модели было получено огромное множество полезных результатов.

### 2.6 Модель LCD

Подробное описание данной модели можно найти в [19].

Выделим в пространстве ось абсцисс и зафиксируем на ней 2n точек: 1, 2, 3, . . . , $2n$. Разобьем эти точки на пары, и элементы каждой пары соединим дугой, лежащей в верхней полуплоскости. Полученный объект назовем линейной хордовой диаграммой (LCD). Дуги в LCD могут как пересекаться, так и лежать друг под другом, но не могут иметь общих вершин. Количество

различных диаграмм равно

$$l_n = \frac{(2n)!}{2^n n!}. \tag{1}$$

По каждой диаграмме построим граф с $n$ вершинами и $n$ ребрами. Процесс построения описан в описывается следующими шагами и показан на Рис. 5.

1. Идем слева направо по оси абсцисс, пока не встретим правый конец какой либо дуги. Пусть этот конец имеет номер $i_1$.
2. Объявляем набор $\{1, \ldots, i_1\}$ первой вершиной графа.
3. Идем слева направо по оси абсцисс, пока не встретим правый конец какой либо дуги. Пусть этот конец имеет номер $i_2$.
4. Объявляем набор $\{i_1 + 1, \ldots, i_2\}$ второй вершиной графа.
5. Продолжаем процедуру по прохода по всем точкам.
6. Ребра порождаем дугами.

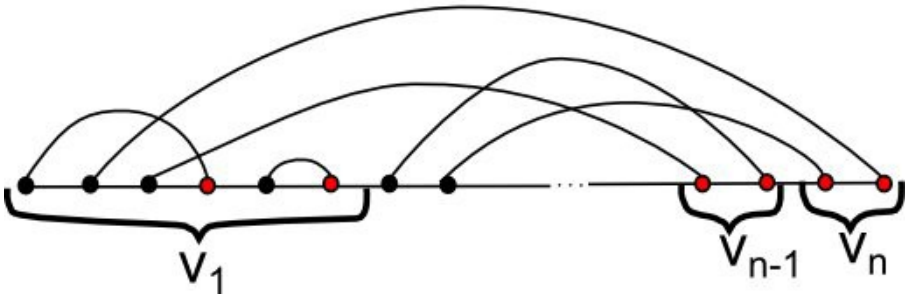


Рисунок 5 – Построение случайного графа в модели LCD

Теперь считаем LCD случайной, т.е. полагаем вероятность каждой диаграммы равной $1/l_n$, где $l_n$ — общее число диаграмм из (1). Таким образом мы получаем случайные графы. В [16] показано, что такие графы по своим вероятностным характеристикам почти неотличимы от графов $G_1^n$ в модели Боллобаша—Риордана.

### 2.7 Модель Чунг—Лу

Модель представляет генерацию графа [20]. Пусть нам задано некоторое конечное множество вершин $V = \{v_1, ..., v_n\}$ и степень каждой вершины $d_i$, $i = \overline{1, n}$. Генерация графа $G = (V, E)$ происходит следующим образом:

— формируем множество $L$, состоящее из $d_i$ копий $v_i$ для каждого $i$ от 1 до $n$;

— задаем случайные паросочетания на множестве $L$;

— для вершин $u$ и $v$ из $V_s$ количество ребер в графе $G$, соединяющее их, равно числу паросочетаний между копиями $u$ и $v$ в $L$.

Сгенерированный таким образом граф соответствует степенной модели $P(\alpha, \beta)$, описывающей графы, для которых:

$$|\{v : \deg(v) = x\}| = \frac{e^{\alpha}}{x^{\beta}}.$$

Верно, что

— при $\beta < 1$ граф является связным;

— при $1 < \beta < 2$ в графе есть гигантская компонента, при этом все остальные компоненты имеют размер $O(1)$;

— при $2 < \beta < \beta_0$ в графе есть гигантская компонента, при этом все остальные компоненты имеют размер $O(logn)$. $\beta \approx 3.47$ — решение уравнения $\zeta(\beta - 2) - 2\zeta(\beta - 1) = 0$.

— при $\beta = 2$ меньшие компоненты имеют размер $O\left(\frac{\log n}{\log \log n}\right)$.

— при $\beta > \beta_0$ в графе почти нет гигансткой компоненты.

### 2.8 Классы моделей случайных графов

Существующие модели случайных безмасштабных графов можно условно разделить на три класса. В первый класс попадают модели со строгим математическим обоснованием выполнения степенного закона для порождаемых графов, для которых также доказан ряд важных свойств (подсчитан диаметр графа, коэффициент кластеризации, и т.п.), однако, показатель степени фиксирован и не может выбираться заранее. Типичным примером такой модели является модель Боллобаша—Риордана (степень равна 3) и ее обобщения (степень не меньше 2). Ко второму классу можно отнести модели, в которых показатель степенной зависимости может задаваться произвольно, что позволяет изучать эффекты фазовых переходов при его изменении. К числу таких моделей можно отнести модель Чунг—Лу. Доказательство наличия фазового перехода в них по свойству содержать большую клику при переходе степени через значение 2 представляет несомненный интерес и стимулирует исследование других свойств случайных графов с этой точки зрения. Наконец к третьему, самому многочисленному классу, относятся модели, в которых характеристики и свойства графов определяются эмпирически. Про такие модели не доказано

никаких содержательных результатов однако, возможно это дело недалекого будущего.

## 3 Простая модель тройственного замыкания

Предметом исследования настоящей работы является простая модель тройственного замыкания.

Модель тройственного замыкания (triple closure) была предложена Питтером Холмом и Бом Джун Кимом и описывает модель в которой рост сети происходит итеративно согласно следующим правилам:

1. (*Рост*) На каждой итерации $t$ добавляется новая вершина;
2. На каждой итерации в сеть добавляется $m$ ребер, и каждая новая вершина $t$ соединяется с $m$ вершинами следующим образом:
   *а*) (*Предпочтительное соединение*) новый узел соединяется с одной из существующих вершин $i$ с вероятностью, пропорциональной ее степени $\deg(i)$;
   *б*) оставшиеся $m-1$ вершины соединяются с новой следующим образом:
   - (*Формирование триады*) с вероятностью $p$ новая вершины $t$ соединяется со случайно выбранном соседом вершины $i$, которая была выбрана в шаге предпочтительного соединения.
   - с вероятностью $1-p$ вершина $t$ соединяется с любой вершиной сети $s$ с вероятностью пропорциональной степени вершины $\deg(s)$

Данное описание модели показывает, что рост степеней узлов сети подчиняется степенному закону:

$$\overline{k}_i(t) = m\left(\frac{t}{i}\right)^{\frac{1}{2}}, \tag{2}$$

где $\overline{k}_i(t)$ обозначает ожидаемую степень вершины $i$ на итерации $t$. Питтер Холм и Бом Джун Ким показали, что распределение степеней также подчиняется степенному закону с показателем $\gamma = 3$, т.е $p_k \sim k^{-3}$, где $p_k$ обозначает вероятность того, что случайно выбранная вершина имеет степень $k$.

Благодаря этапу формирования триады (Triad formation), коэффициент кластеризации модели П. Холма и Б. Д. Кима выше, чем в модели Барабаши—Альберт, а геодезическая длина в обеих моделях растет как логарифм от $N$, где $N$ — размер сети.

### 3.1 Описание предложенной модели

Модель в данный работе, будет опираться на принципы тройственного замыкания и предпочтительного соединения [21], т.е в какой-то степени это частный случай модели, предложенной Питтером Холмом и Бом Джун Кимом. Правила по которым будет строиться безмасштабная сеть:

- на каждой итерации $t$ добавляется новая вершина,ей приписывается метка $t$;
- вершина $t$ соединяется ребром с вершиной $u$ с вероятностью, пропорциональной её степени $\deg(u)$;
- среди вершин, смежных с вершиной $u$ проводится равновероятный выбор одной из них $-v$, с которой вершина $t$ соединяется вторым ребром.

Интересом анализа новой модели будет выполняться в следующих моментах:

- степенное распределение будет также подчиняться степенному закону с показателем 3;
- динамика роста степени вершины, которая будет выполнять асимптотику выше освещенных моделей, и будет изменяться как квадратный корень от времени.

# 4 Моделирование случайного графа

## 4.1 Реализация, использующая матрицы смежности

Опишем реализацию алгоритма, использующего матрицы смежности для представления графов. Матрицу смежности будем представлять в виде двумерного массива "numpy",заполняемого в процессе генерации сети [13]. На вход программа получает два числа T, M — количество генерируемых графов, и количество вершин в этих графах.

Первым делом задаются значения: T и M, где T — количество графов, а M — количество вершин графов.

```
1 T = 20   # количество графов
2 M = 100  # количество вершин в графе
```

Для вычисления средних значений количества вершин определенной степени в графе, заведем словарь, в котором для степени вершин, будем хранить суммарное количество вершин с такой степенью во всех построенных графах.

```
1 deggs = {} # словарь, в котором будем хранить количество вершин
2           # в графах, у которых заданная степень
```

Организуем цикл для построения T графов

```
1 for l in range(1, T+1):
2     print(l)
```

Параметр цикла *l*, цикл изменяется в диапазоне от *1* до *T*

Создаем двумерный массив для матрицы смежности графа:

```
1     dists = np.full((M+1) * (M+1), 0).reshape(M+1, M+1)
```

Массив заполняется "нулями"((M+1) * (M+1), 0) и переформируется в двумерный массив в котором будет M+1 строка M+1 столбец.

Создаем массив для вектора степеней вершин графа:

```
1     degs = np.full(M+1, 0)      # создаем вектор степеней вершин: изначально
          все 0
2     dists[0][1] = dists[1][0] = 1 # соединяем нулевую и первую вершины ребром
3     degs[0] = 1                 # соответственно, у нулевой вершины теперь
          степень 1
4     degs[1] = 1                 # и у первой тоже
```

В самом начале в неориентированном графе две вершины: 0 и 1, они соединяются ребром. Так как граф неориентированный, элемент массива [0][1], также как и элемент [1][0] устанавливаем равными 1. Соответственно степени вершин 0 и 1 устанавливаем равными 1.

Далее запускаем цикл в котором добавляется со 2-й по M-1 вершины, т.е. в итоге, после цикла, получим M вершин в графе.

```
1    for k in range(2,M):
```

Мы должны k-ю (новую) вершину графа соединить ребром сначала со случайной вершиной, с вероятностью, пропорциональной её степени. Для этого сформируем аргументы, для передачи функции choices: первый аргумент — уже сформированные вершины графа — они имеют номера от 0 до k-1; второй аргумент — степени этих вершин. После этого вызовем функцию choices.

```
1        pairs = range(0, k)
2        weights = degs[0:k].copy()
3        [j1] = random.choices(pairs, weights)
```

Теперь в j1 — номер выбранной вершины для первого ребра, инцидентного вершине k.

Для выбора проведения второго ребра, выберем из матрицы смежности номера индексов, соответствующих ненулевым значениям в строке, соответствующей j1-ой вершине. Нужные номера вершин добавляем в список pairs. После этого из списка pairs с помощью choices с равными вероятностями выбирается один из элементов.

```
1        pairs = []
2        for i in range(0, k):
3            if dists[j1][i] > 0:
4               pairs.append(i)
5        [j2] = random.choices(pairs)
```

Теперь в j2 — номер второй выбранной вершины для добавления ребра от вершины с номером k.

Отмечаем в матрице смежности, что у нас добавляются ребра между вершинами k и j1 и между вершинами k и j2. Кроме того меняем степени вершин у вершин j1 и j2, а степень вершины k устанавливаем равной 2.

```
1        dists[j1][k] = dists[k][j1] = 1
2        dists[j2][k] = dists[k][j2] = 1
```

```
        degs[k] = 2   # степень новой (k-й) вершины становится равна 2
        degs[j1] += 1 # степени вершин j1 и j2 увеличиваются на 1
        degs[j2] += 1
```

Сейчас цикл по k окончен. Очередной граф построен.

Следующий цикл подсчитывает в deggs[ind] сколько вершин и с какой степенью имеется в графе.

```
    for i in range(0, M):
        ind = degs[i]
        if ind in list(deggs.keys()):
            deggs[ind] += 1
        else:
            deggs[ind] = 1
```

Цикл по построению T графов окончен.

Теперь высчитываем среднее значение количества вершин определенной степени в построенных графах. После чего, выводим график распределения количества вершин по степеням вершин, используя логарифмическую шкалу по обеим осям.

```
for key in list(deggs.keys()):
    deggs[key] = deggs[key] / T

its = dict(sorted(deggs.items()))
print(its) #сортирует и выводит значения в графе

xs = list(its.keys())
ys = list(its.values())

plt.loglog(xs, ys) # и по x и по y - логарифмическая шкала

plt.show() # выводим окно с изображением на экран
```

С полным кодом программы можно ознакомиться в приложении A.

### 4.2 Реализация, использующая библиотеку `networkx`

Опишем реализацию алгоритма, использующего библиотеку `networkx` для работы с графами.

Также как и в предыдущей программе задаем значения: `T` и `M`, где $T$ — количество графов, а $M$ — количество вершин в графах.

```
T = 5
M = 10
```

Далее организуем цикл по создаваемым графам. В начале цикла создаем граф, добавляем в него первые две вершины, которые соединяем ребром.

```
for l in range(1, T + 1):
    print(l)

    G = nx.Graph() # создаем граф
    G.add_node(0) # добавляем нулевую вершину
    G.add_node(1) # добавляем первую вершину
    G.add_edge(0, 1) # соединяем нулевую и первую вершины ребром
```

Параметр цикла `l`, цикл изменяется в диапазоне от `1` до `T`.

Для создания каждого из `T` графов организуем цикл по `k`, изменяющемся в диапозоне от 2 до `M-1`.

```
    for k in range(2, M):
        G.add_node(k) # добавляем k-ю вершину
                deg = G.degree # извлекаем массив степеней вершин графа
```

Инициализируются пустые списки:

```
        pairs = [] # Пары вершин
        weights = [] # Вес вершин
```

Пробежимся по вершинам графа, добавляя в список `pairs` номера вершин, а в список `weights` — их степени. После этого вызовем функцию `choices`, которой передадим сформированные параметры.

```
        for i in range(0, k):
            pairs.append(i)
            weights.append(deg[i])
        [j1] = random.choices(pairs, weights) # выбирается рандомная вершина j1
                                              # вероятность выбора вершины
                                                 пропорциональна её степени
```

Теперь в j1 номер первой выбранной вершины для соединения ребром с вершиной k.

Теперь сформируем список вершин, смежных с вершиной j1.

```
1       pairs = []
2       for i in G[j1]:     # все смежные с j1 вершины
3           pairs.append(i) # добавляются в список pairs
```

Выражение [j2] = random.choices(pairs) — говорит о том, что из списка pairs выбирается случайная вершина j2. Далее соединяются ребра j1 с k вершиной, так же как и с j2

```
1       [j2] = random.choices(pairs)
2       G.add_edge(j1, k)
3       G.add_edge(k, j2)
```

Цикл для построения очередного графа окончен.

Построенный граф можно начертить. Для этого зададим набор опций. После этого граф отрисовывается.

```
1   options = {
2           'node_color': 'lightblue',
3           'edge_color': 'lightblue',
4           'node_size': 10,
5           'width': 1,
6           'with_labels': True,
7       }
8       nx.draw_circular(G, **options) # чертим граф
```

Общий массив для всех графов deggs[ind] подсчитывает сколько вершин и с какой степенью присутствуют в графе, создается список степеней, если ее еще не было, то заводиться с 1, если же он есть то он увеличивается на 1.

```
1   for i in range(0, M):
2       ind = deg[i][1]
3       if ind in list(deggs.keys()):
4           deggs[ind] += 1
5       else:
6           deggs[ind] = 1
```

Вывод среднего количества вершин по всем графам спискakey. its = dict(sorted(deggs.items())) — сортирует словарь deggs по значению клю-

ча, `print(its)` — выводит отсортированный словарь на экран.

```
for key in list(degs.keys()):
    degs[key] = degs[key] / T
its = dict(sorted(deggs.items()))
print(its)
```

Вывод графика с логарифмической шкалой:

```
plt.loglog(xs, ys) # Изменение размеров графика и отображение
plt.show() # выводим окно с изображением на экран
```

С полным кодом программы можно ознакомиться в приложении Б.

### 4.3 Построение графов

Было проведено несколько серий экспериментов.

В первой серии экспериментов в реализации программы с использованием библиотеки `networkx` выводились построенные графы для малых значений `M`.

На Рис. 6–13 можно наблюдать процесс формирования графа из 10 вершин на каждой итерации.

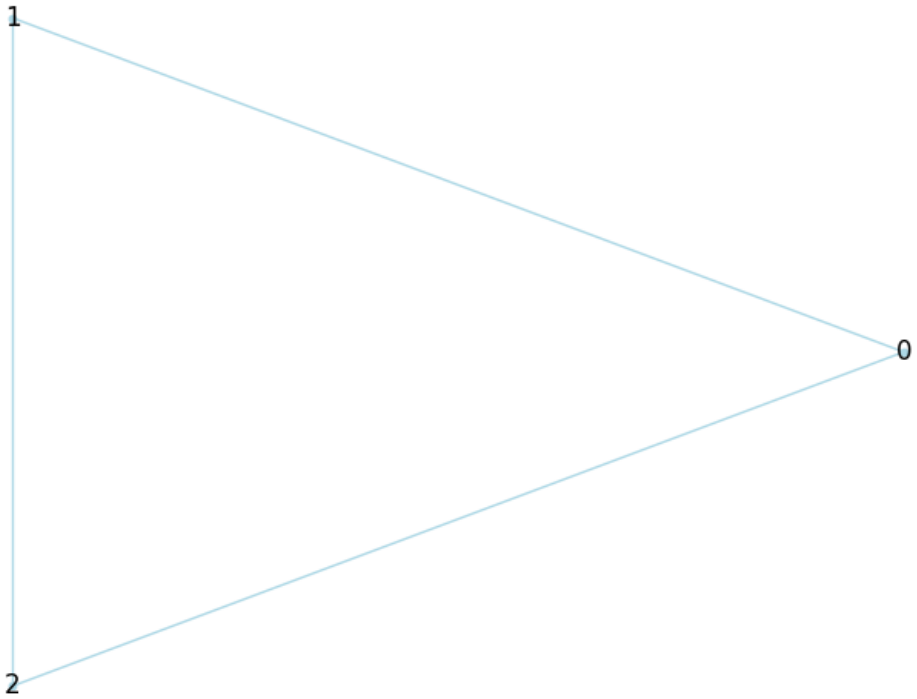


Рисунок 6 – Добавление 3-й вершины

Первый шаг однозначный. После него имеем полный граф.

От второго шага зависит, какие из двух вершин, имеющихся на текущий момент увеличат свою степень. На Рис. 7 такими вершинами оказались вершины с номерами 0 и 1. С большей вероятностью последующие шаги будут выбирать эти вершины, для проведения очередного ребра.

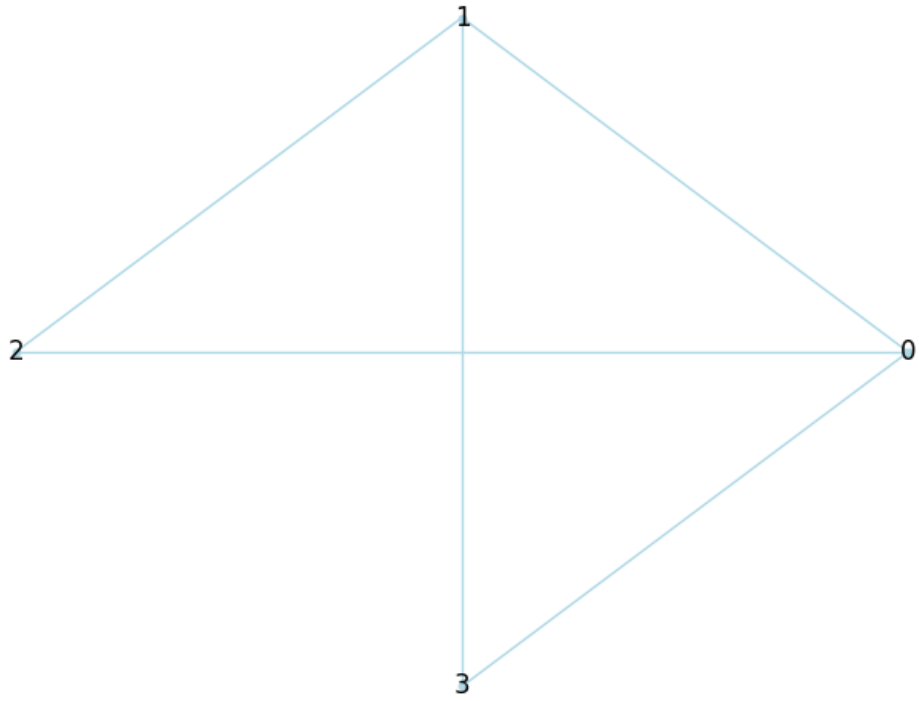


Рисунок 7 – Добавление 4-й вершины

Следующий шаг это подтверждает. Выбрана вершина 1 (см. Рис. 8).

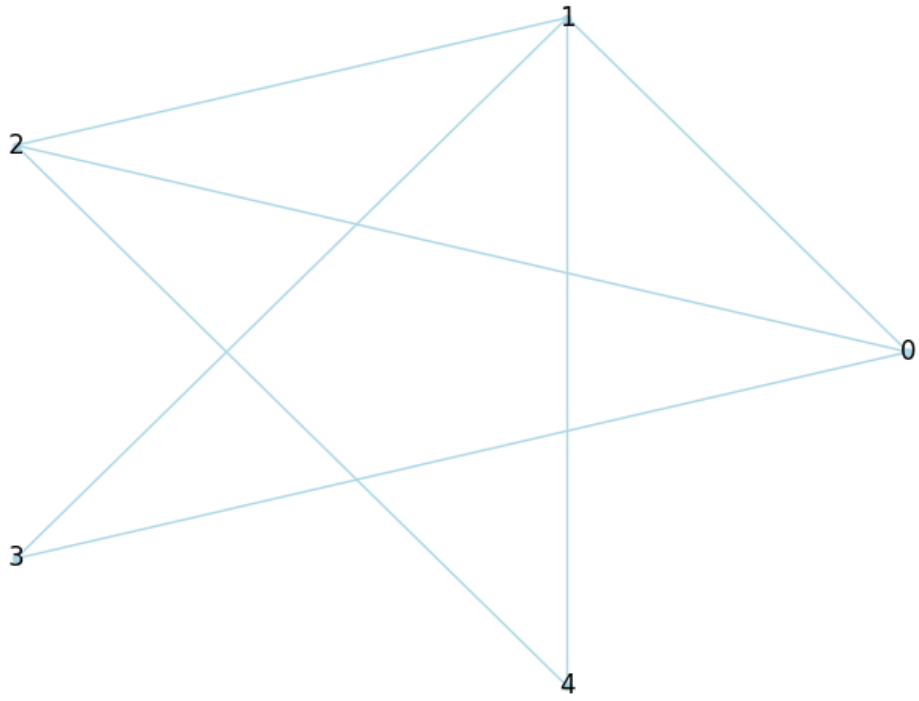


Рисунок 8 – Добавление 5-й вершины

А на следующем выбрана вершина 0 (см. Рис. 9).

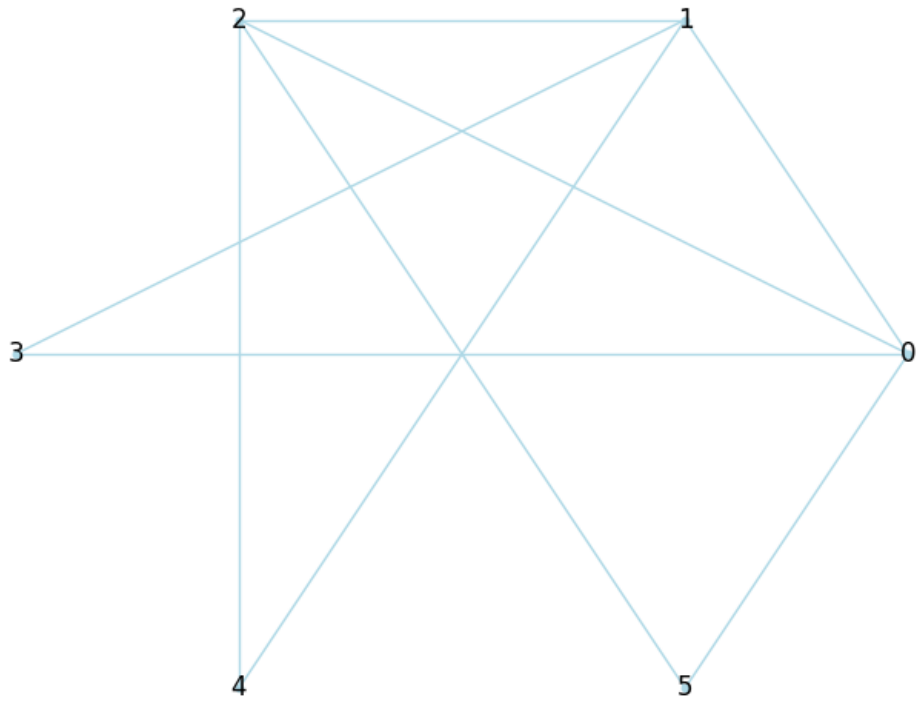


Рисунок 9 – Добавление 6-й вершины

Дальнейшие шаги приводят к росту степени вершины 1 и смежных с ней вершин. Так можно заметить, что несмежная с ней вершина 5 после окончания построения графа так и осталась вершиной со степенью 2.

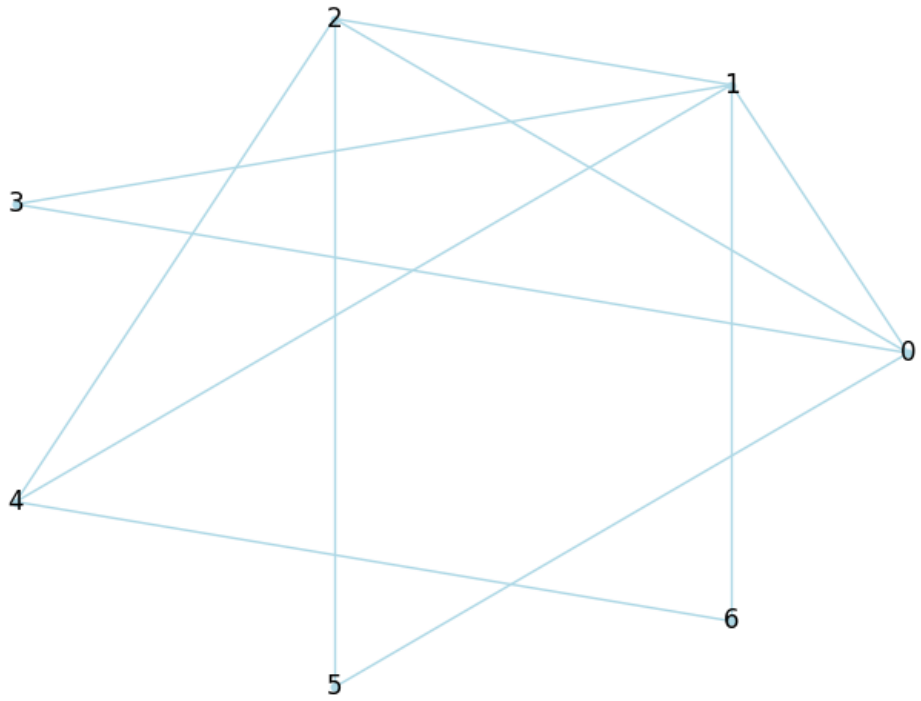


Рисунок 10 – Добавление 7-й вершины

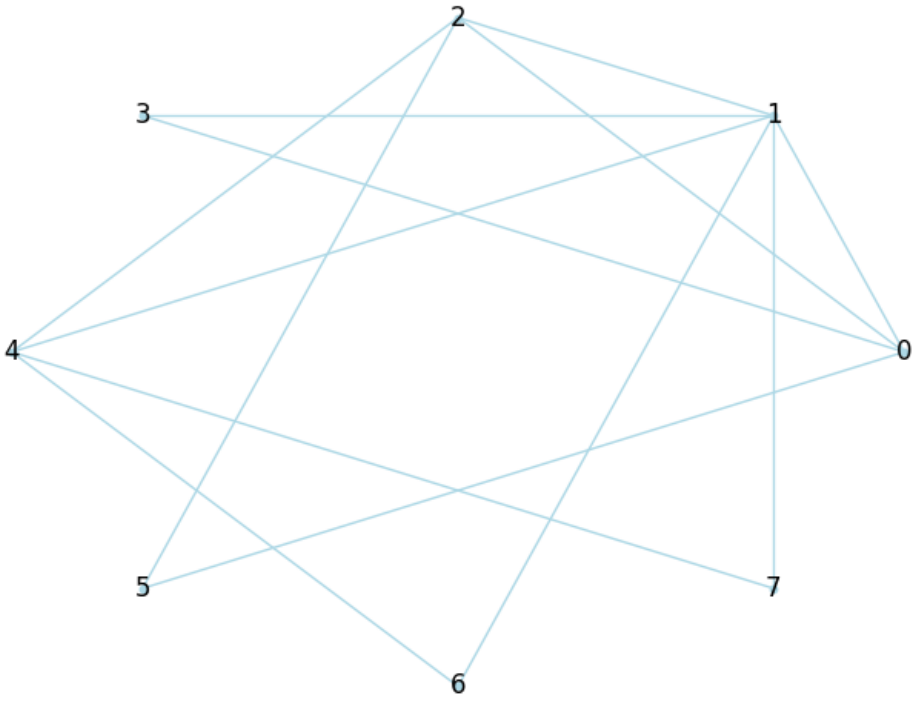


Рисунок 11 – Добавление 8-й вершины

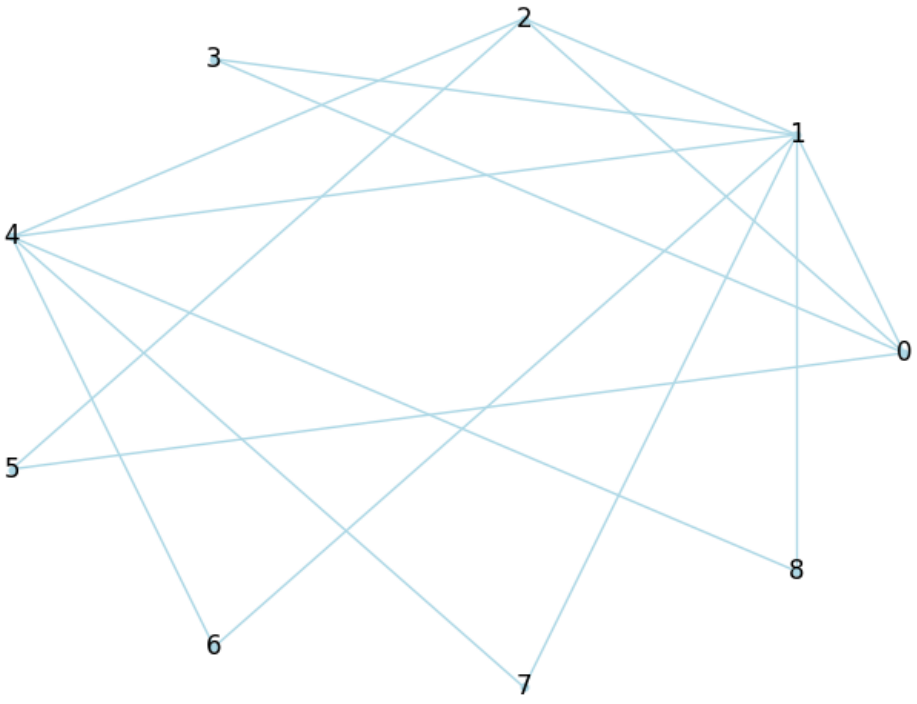


Рисунок 12 – Добавление 9-й вершины

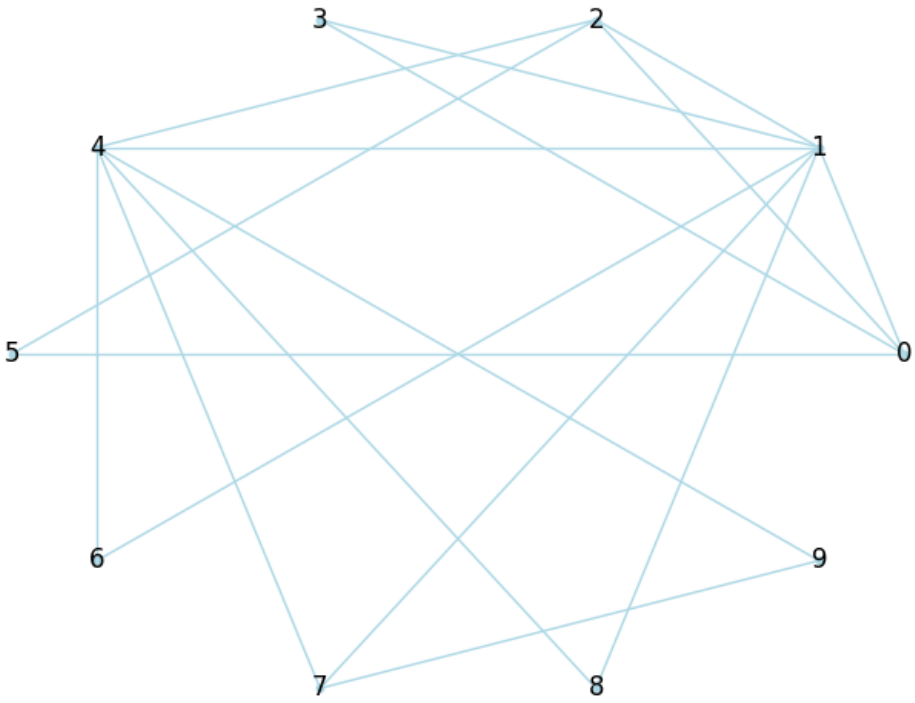


Рисунок 13 – Добавление 10-й вершины

На Рис. 14–18 можно видеть результаты различных построений для разных значений `M`.

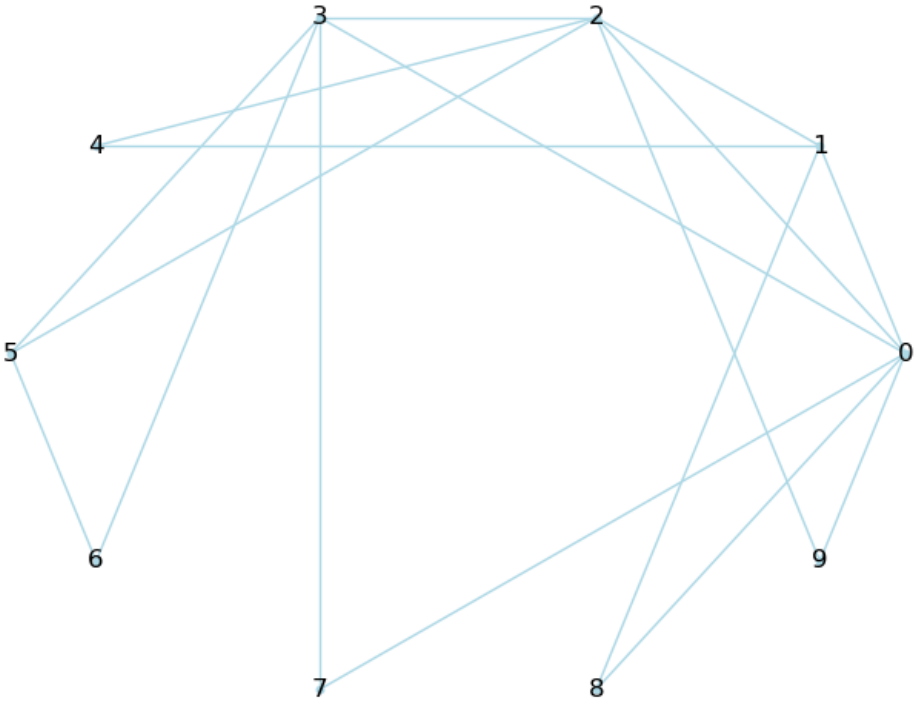


Рисунок 14 – Параметры: T = 1, M = 10

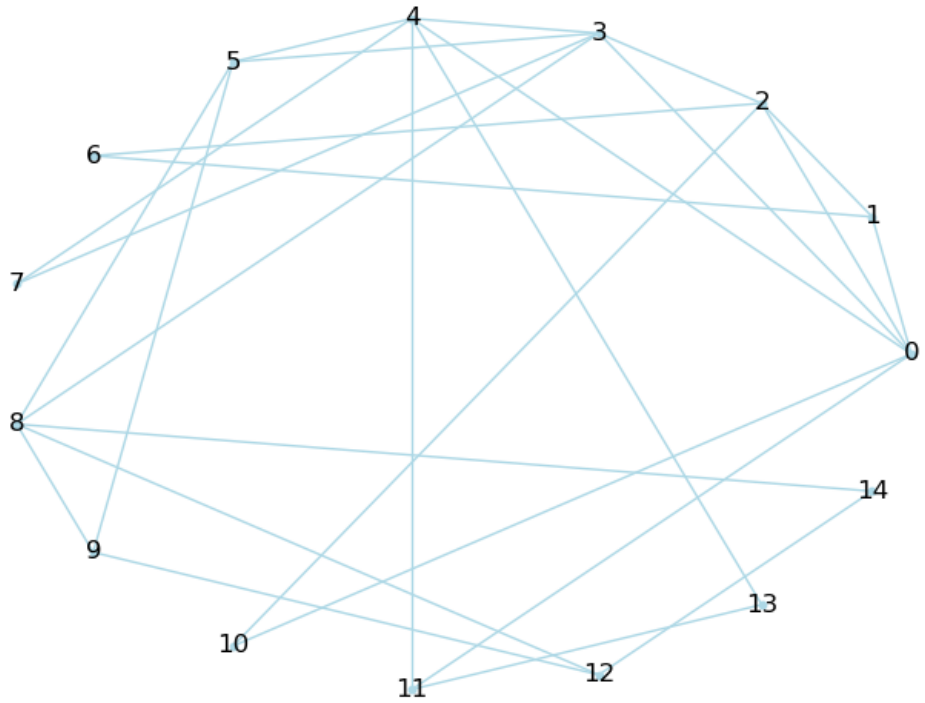


Рисунок 15 – Параметры: T = 1, M = 15

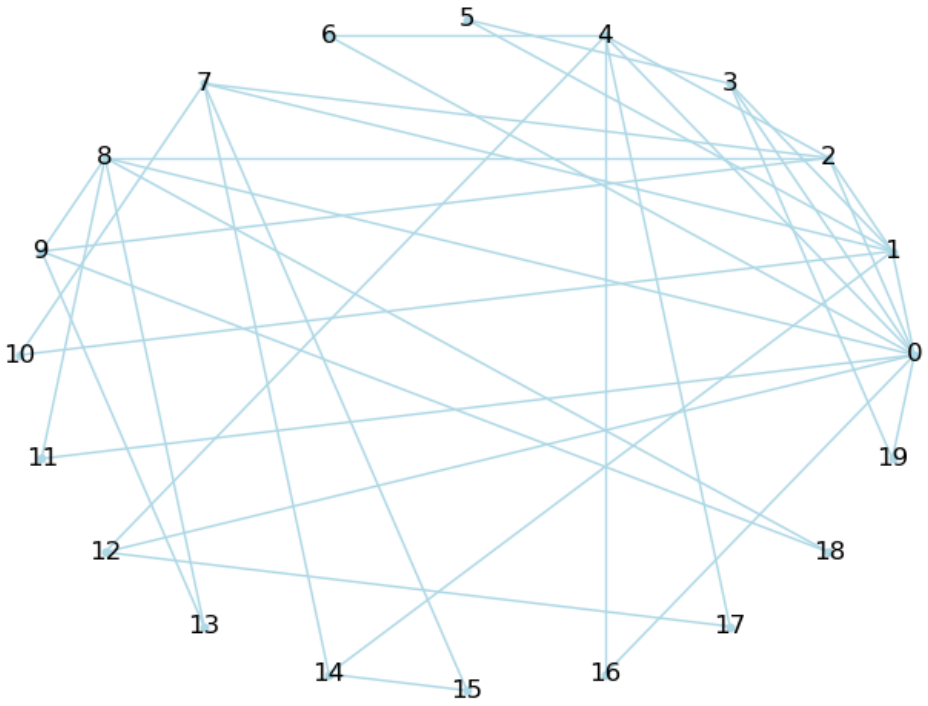


Рисунок 16 – Параметры: T = 1, M = 20

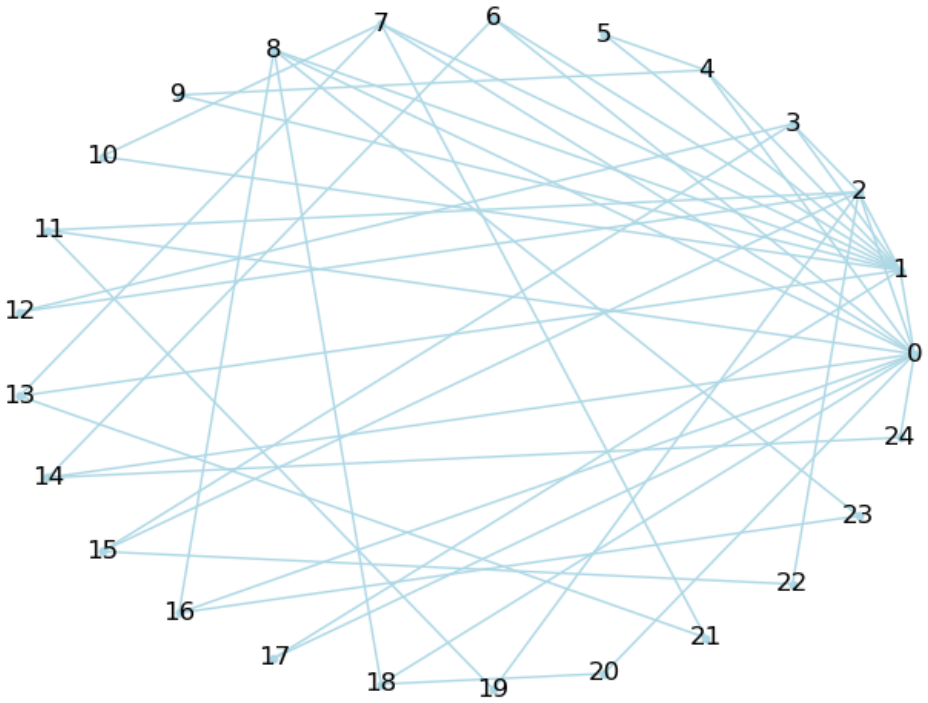


Рисунок 17 – Параметры: T = 1, M = 25

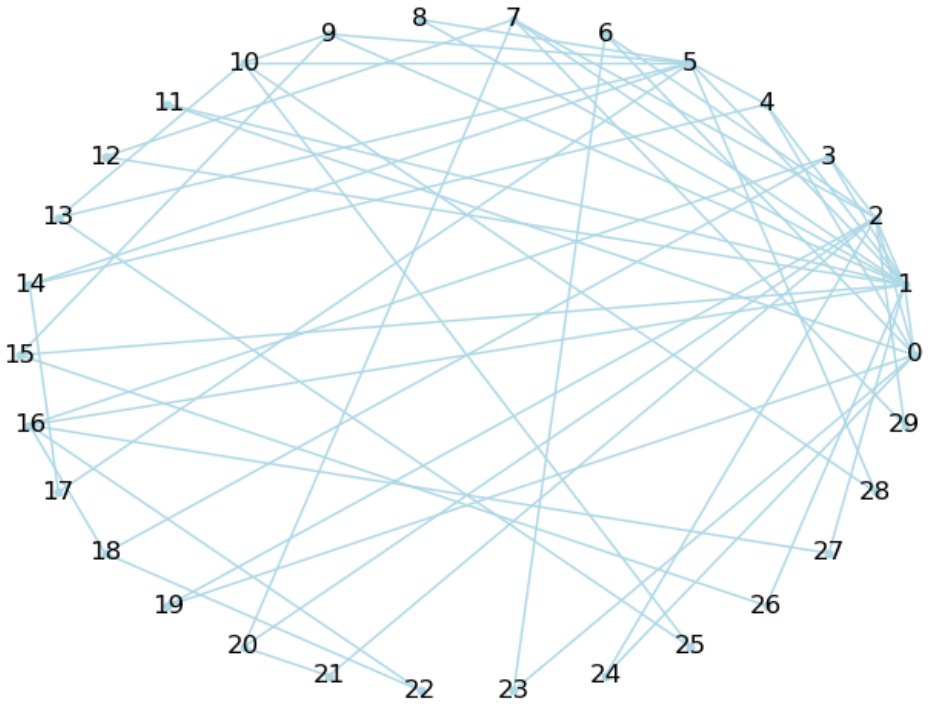


Рисунок 18 – Параметры: T = 1, M = 30

На приведенных графах стоит обратить внимание на факт, что у некоторого множества вершин, добавленных в граф на первых шагах, степень вершин растет значительно. В графе на Рис. 18, например, такими вершинами явля-

ются вершины с номерами 1 и 2, а в графе на Рис. 17 — вершины с номерами 0 и 1.

### 4.4 Сравнение работы двух реализаций

Во второй серии экспериментов программы запускались для больших значений параметра M. Сравнивалось время работы алгоритмов.

Для оценки времени работы программы были использованы средства библиотеки time.

```
1 import time
2 start_time = time.time()
3 print("%s секунд" % (time.time() - start_time))
```

Результаты запусков приведены в таблице.

Таблица 1 – Время работы реализаций
с разными параметрами M и T

| T | M | Время построения набора графов, сек | |
|---|---|---|---|
| | | numpy | networkx |
| 100 | 100 | 0.499 | 0.364 |
| 50 | 200 | 0.868 | 0.639 |
| 20 | 1000 | 7.779 | 6.054 |
| 20 | 10000 | 620.195 | 589.299 |
| 10 | 50000 | - | 8136.447 |
| | | 14322.712 | 13346.453 |

В ходе выполнения экспериментов было выявлено, что выполнение программы с использованием матрицы смежности работа не только выполнялась дольше, чем с использованием библиотеки networkx, но еще и оказалась более требовательным к памяти при больших значениях параметров. Практически все запуски приложений проводились на компьютере с процессором i5 4.8 Ghz и 4-мя ядрами с оперативной памятью 8-ми гигабайт. Но исполнение на этом компьютере приложения, использующего матрицу смежности, при значении параметра M равном 50000, оказалось невозможным. Этим обусловлен прочерк в первой строке таблицы, соответствующей этому параметру. Исполнение программы с использованием библиотеки networkx оказалось успешным.

Вторая строка результатов при M равном 50000 получена на компьютере с процессором i5 2.5 Ghz и 2-мя ядрами.

### 4.5 Результат работы

В ходе выполнения программ собиралась и сохранялась в файл информация о среднем значении количества в построенных графах вершин с определенной степенью. Фрагмент такой информации можно увидеть в приложении В. В конце работы алгоритмов эта информация представляется в виде графиков. На Рис. 19 и 20 представлены такие графики при значении параметров `T = 10` и `M = 50`.

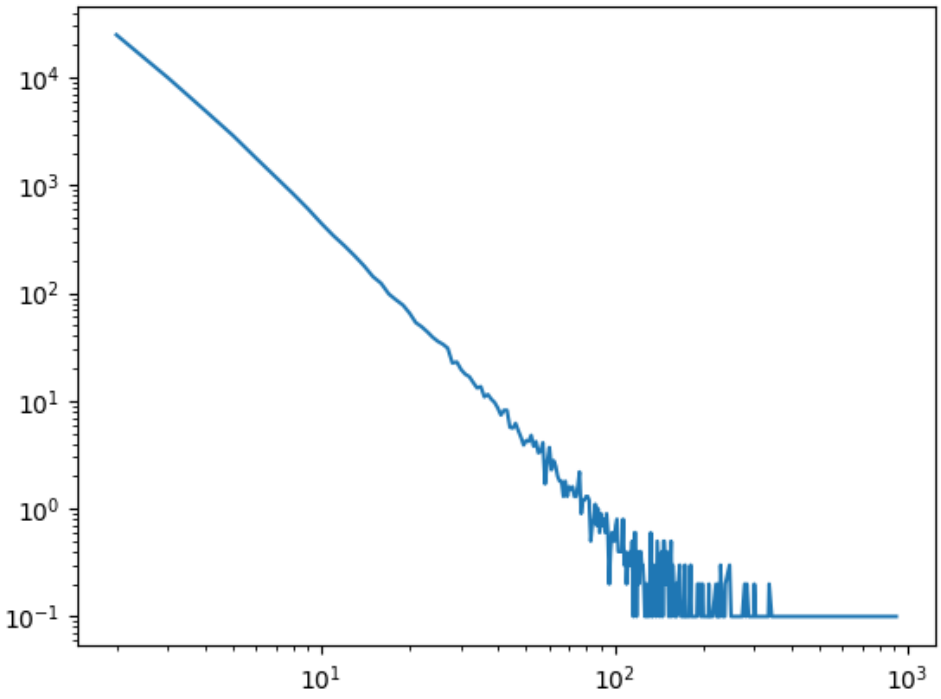


Рисунок 19 – numpy Параметры: T = 10, M = 50000.

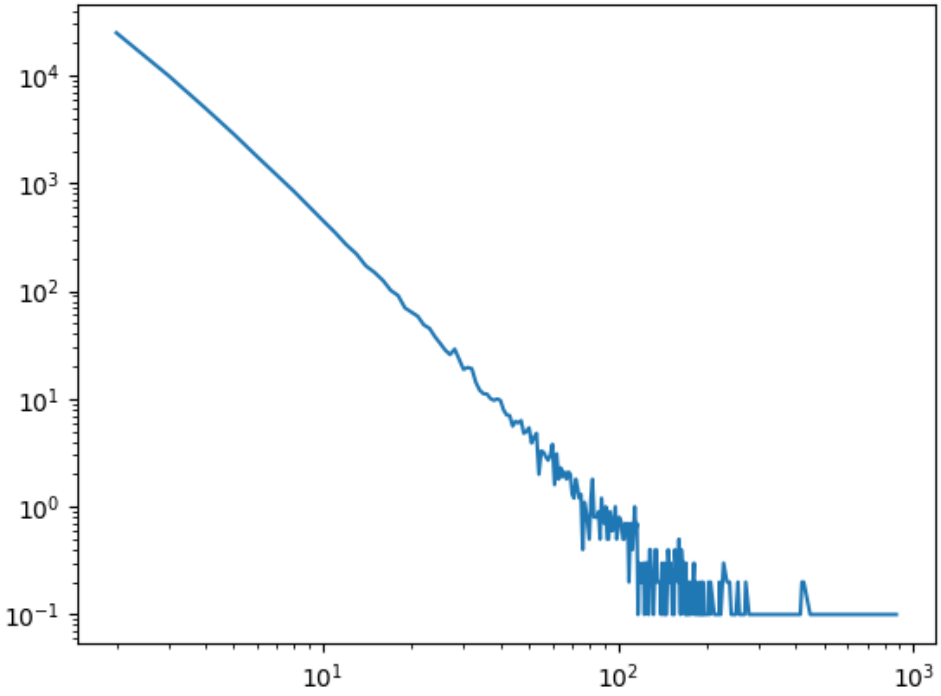


Рисунок 20 – networkx Параметры: T = 10, M = 50000.

Несмотря на то, что графы генерировались случайным образом, графики распределения степеней оказались практически идентичными. Часть графиков от 0 до $10^2$ почти вырождена в отрезок прямой.

Визуализированные результаты вычислений, представленные на Рис. 19 и 20, показывают, что коэффициент угла наклона прямой на логарифмически шкалированном графике распределения степеней равен -3. Это подтверждает теоретические результаты представленные в [2], которые говорят о том, что сети, генерируемые данной моделью являются безмасштабными со степенным законом распределения степеней, экспонента которого имеет значение -3.

# ЗАКЛЮЧЕНИЕ

В данной работе была изучена модель построения генерации случайного графа, основанном на принципах предпочтительного соединения и тройственного замыкания, являющаяся частным случаем модели, предложенной Питтером Холмом и Бом Джун Кимом.

Данная модель вызывает интерес по той причине, что она может быть усложнена, что дает простор для дальнейших исследований. Можно предложить модель, в которой на каждой итерации добавляется не одна вершина, а другое константное случайно выбранное число, причем выбор распределения для этого числа может послужить для появления новой модели. Так же можно поступить по отношению к добавляемому числу ребер.

В ходе работы была проведена реализация на языке Python модели построения случайного графа с использованием матрицы смежности и с использованием библиотеки `networkx` для работы с графовыми структурами. Был сделан вывод, что использование матрицы смежности является более требовательным к памяти компьютера, на котором исполняется программа и приводит к увеличению времени генерации графа.

Полученные эмпирические результаты подтверждают теоретические выводы, касающиеся модели построения графа, полученные в [2].

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1 *Колчин, В. Ф.* Случайные графы / В. Ф. Колчин. — Спб: Физматлит, 2004.

2 *Holme, P.* Growing scale-free networks with tunable clustering / P. Holme, B. J. Kim // *Phys. Rev. E.* — Jan 2002. — Vol. 65. — P. 026107. https://link.aps.org/doi/10.1103/PhysRevE.65.026107.

3 *Bianconi, G.* Triadic closure as a basic generating mechanism of communities in complex networks / G. Bianconi, R. Darst, J. Iacovacci, S. Fortunato. — Vol. 65.: Phys. Rev. E., 2014.

4 *Huang, H.* Triadic closure pattern analysis and prediction in social networks / H. Huang, J. Tang, L. Lio, J. Luo, X. Fu. — Vol. 27, no. 12.: IEEE Transactions on Knowledge and Data Engineering, 2015.

5 *Linyi, Z.* The node influence for link prediction based on triadic closure structure / Z. Linyi. — Pp. 761–766.: 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2017.

6 Will triadic closure strengthen ties in social networks? [Электронный ресурс]. — URL: https://doi.org/10.1145/3154399 (Дата обращения 17.05.2020). Загл. с экр. Яз. англ.

7 The local closure coefficient: A new perspective on network clustering. [Электронный ресурс]. — URL: https://doi.org/10.1145/3289600.3290991. (Дата обращения 13.05.2020). Загл. с экр. Яз. англ.

8 *Райгородский, А. М.* Экстремальные задачи теории графов и анализ данных / А. М. Райгородский. — Мск: Регулярная и хаотичная динамика, 2009.

9 *Barabashi, L. A.* Emergence of scaling in random networks / L. A. Barabashi, R. Albert. — Cambridge: Science, 1999.

10 *Barabashi, L. A.* Scale-free characteristics of random networks: the topology of the world-wide web / L. A. Barabashi, R. Albert, H. Jeong. — Cambridge: Physica, 2000.

11 *Маргулис, Г. А.* Вероятностные характеристики графов с большой связностью / Г. А. Маргулис. — Мск: Проблемы передачи информации, 1974.

12 *Саммерфилд, М.* Программирование на Python 3. Подробное руководство / М. Саммерфилд. — Спб: Символ-Плюс, 2009.

13 Learnpython [Электронный ресурс]. — URL: https://www.learnpython.org/en/Modules_and_Packages (Дата обращения 22.05.2020). Загл. с экр. Яз. англ.

14 *Харари, Ф.* Теория графов / Ф. Харари. — М.: Мир, 1973.

15 *Erd os, P.* On random graphs / P. Erd os, A. Rényi // *Publ. Math. Debrecen*. — 1959. — Vol. 6. — Pp. 290–297.

16 *Barabaśi, L.-A.* Diameter of the world-wide web / L.-A. Barabaśi, R. Albert, H. Jeong // *Nature*. — 1999. — Vol. 401. — Pp. 130–131.

17 *Bollobas, B.* The degree sequence of a scale-free random graph process / B. Bollobas, O. Riordan, J. Spencer, G. Tusnady. — New York: Random Structures Algorithms, 2001.

18 *Bollobás, B.* Mathematical results on scale-free random graphs / B. Bollobás, O. M. Riordan // Handbook of Graphs and Networks. — John Wiley & Sons, Ltd, 2005. — Pp. 1–34.

19 *Райгородский, А. М.* Модели случайных графов / А. М. Райгородский. — М.: МЦНМО, 2011.

20 *Aiello, W.* A random graph model for massive graphs // Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing. — STOC '00. — New York, NY, USA: Association for Computing Machinery, 2000. — P. 171–180. https://doi.org/10.1145/335305.335326.

21 Researchgate [Электронный ресурс]. — URL: https://www.researchgate.net/publication/11497687_Growing_Scale-Free_Networks_with_Tunable_Clustering (Дата обращения 20.05.2020). Загл. с экр. Яз. англ.