МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ИССЛЕДОВАНИЕ ИНДЕКСА ДРУЖБЫ МОДЕЛИ БАРАБАШИ—АЛЬБЕРТ

КУРСОВАЯ РАБОТА

студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Козырева Юрия Дмитриевича

Научный руководитель
зав. каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Теоретические сведения 5
1.1 Модели построения случайного графа 5
1.1.1 Модель Эрдеша—Ренье 5
1.1.2 Модель Барабаши—Альберт 5
1.1.3 Модель Боллобаша—Риодана 6
1.1.3.1 Динамическая модификация 6
1.1.3.2 Статическая модификация, или LCD-модель 6
1.2 Индекс дружбы 7
2 Реализация модели Барабаши—Альберт 10
2.1 Реализация стандартной модели Барабаши—Альберт 10
2.2 Реализация метрик для выявления парадокса дружбы 12
2.3 Паралелизация построения модели Барабаши—Альберт 13
2.4 Вывод и представление данных для анализа 14
3 Анализ 16
ЗАКЛЮЧЕНИЕ 18
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19
Приложение А Текст программы 21

ВВЕДЕНИЕ

Наука о графах — одно из приложений к весьма красивой и интересной науке — науке о случайных графах.

В повседневной жизни для решения многих задач часто используются случайные графы. Случайные графы нашли практическое применение во всех областях, где нужно смоделировать сложные сети — известно большое число случайных моделей графов, отражающих разнообразные типы сложных сетей в различных областях. Случайные графы применяются при моделировании и анализе биологических и социальных систем, сетей, а также при решении многих задач класса NP.

Случайные графы впервые определены венгерскими математиками Эрдёшем и Реньи в книге 1959 года «On Random Graphs» [1] и независимо Гильбертом в его статье «Random graphs» [2].

Случайный граф — общий термин для обозначения вероятностного распределения графов [3]. Их можно описать просто распределением вероятности или случайным процессом, создающим эти графы.

Теория сучайных графов находится на стыке комбинаторики, теории графов и теории вероятностей. В основе ее лежит глубокая идея о том, что мощные инструменты современной теории вероятностей должны поспособствовать более верному осознанию природы графа, призваны помочь решению многих комбинаторных и теоретико-графовых задач [4].

С математической точки зрения случайные графы необходимы для ответа на вопрос о свойствах типичных графов.

Модель Барабаши—Альберт является одной из наиболее популярных и хорошо изученных моделей случайных графов.

Индекс дружбы - один из показателей используемых в социологии при анализе социальных сетей и других социальных явлений, который определяется как отношение средней степени соседей к степени самого объекта.

Целью настоящей работы является анализ индекса дружбы модели случайного графа Барабаши—Альберт. Для достижения этой цели необходимо решить следующие задачи.

— рассмотреть алгоритм Барабаши—Альберт для построения случайного графа;
— реализовать алгоритм Барабаши—Альберт на некотором языке програм-

мирования;

— реализовать параллельное вычисление индекса дружбы для построенного графа;
— провести анализ закона распределения индекса дружбы графа, построенного по алгоритму Барабаши—Альберт.

1 Теоретические сведения

1.1 Модели построения случайного графа

Существуют различные модели построения случайных графов.

1.1.1 Модель Эрдеша—Ренье

Модель Ердеша—Ренье является одной из первых моделей случайного графа. Граф построенный по этой модели представляет собой совокупность множества вершин $V = \{1, ..., n\}$ и множества рёбер E, состоящего из рёбер полного графа K_n построенного на множестве V, выбранных по схеме Бернулли. Таким образом образуется случайный граф $G = (V, E)$. Формально выражаясь, мы имеем вероятностное пространство

$$G(n, p) = (\Omega_n, F_n, P_{n,p}),$$

в котором: n — количество вершин, p — вероятность появления новго ребра, F_n — сигма-множество,$|\Omega_n \mid = 2^N$ — множество возможных рёбер, $P_{n,p}\ (G) = p^{|E|}q^{\binom{n}{2}} - |E|$ — вероятностная мера. 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 [5].

1.1.2 Модель Барабаши—Альберт

Модель Барабаши—Альберт является одной из первых моделей веб-графов. Веб-граф представляет собой ориентированный мульти-граф, вершинами в котором являются какие-либо конкретные структурные единицы в Интернете: речь может идти о страницах, сайтах, хостах, владельцах и пр. Для определенности будем считать, что вершинами веб-графа служат именно сайты. А рёбрами соединяются вершины, между которыми имеются ссылки.

Также Барабаши и Альберт была предложена модель предпочтительного присоединения [6], основная идея которой заключается в том, что при присоединении к графу новой вершины проводится некоторое количество рёбер от добавленной вершины к уже существующим, при этом вероятность появления ребра межу новой вершиной и какой-то конкретной вершиной пропорциональна степени данной вершины(степенью вершины $v_i \in V$ графа $G = (V, E)$ называ-

ется колличество вершин, напрямую связанных с данной, т.е.

$$deg(v_i) = |\{v \in V : (v, v_i) \in E\}|$$

). Однако в своих работах Барабаши и Альберт никак не конкретизировали, какую именно из этих моделей они предлагают рассматривать.

1.1.3 Модель Боллобаша—Риодана

Одной из наиболее удачных и часто используемых моделей предпочтительного присоединения является модель Боллобаша—Риодана. Существуют две основных и, по сути, совпадающих модификации этой модели. В одной дается динамическое, а в другой статическое описание случайности [7].

1.1.3.1 Динамическая модификация В данной модификации при добавлении n-ной вершины проводятся n новых рёбер, при этом рёбра могут быть кратными, а также петлями и даже кратными рёбрами, при создании графа с единственной вершиной проводится петля в этой точке [5]. Таким образом вероятность появления ребра $(n,\ i),\quad i \in [0,\ n-1]$ равна $\frac{\deg\ i}{2n-1}$, где $degi$ — количество уже проведенных рёбер из вершины n в вершину i. Очевидно, что распределение вероятностей задано корректно, поскольку

$$\sum_{i=1}^{n-1} \frac{deg\ i}{2n-1} + \frac{1}{2n-1} = \frac{2n-2}{2n-1} + \frac{1}{2n-1} = 1,$$

где $degi$ - степень (колличество соседей) i-той вершины.

1.1.3.2 Статическая модификация, или LCD-модель Данная модель основывается на объекте называемом линейной хордовой диаграммой (LCD). Для построения данного объекта требуется зафиксировать на оси абсцисс $2n$ точек $1, \ldots, 2n$, разбить их на пары и соединить элементы каждой пары дугой, лежащей в верхней полуплоскости. Количество различных диаграмм равно

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

По каждой диаграмме строится граф с n вершинами и n ребрами по следующему алгоритму:

— Идти слева направо по оси абсцисс пока не встретится правый конец какой-либо дуги, пусть позиция этой точки равна i_k
— Последовательность $i_{k-1}+1, i_k$ объявляется списком смежности для k-той вершины, $i_0 = 0$
— Если $k < n$, k увеличивается на 1, переход на шаг (1).

При построении модели LCD случайно выбирается одна из возможных LCD и вероятность каждой диаграммы равной $\frac{1}{l_n}$, где l_n – общее число диаграмм. Графы построенные по такой модели имеют те же свойства, что и графы построенные по динамической модификации схемы Боллобаша—Риодана. Модель Чунг-Лу Пусть нам задано некоторое конечное множество вершин $V = v_1, \ldots, v_n$ и степень каждой вершины d_i, $i = \overline{1, n}$. Генерация графа $G = (V, E)$ происходит следующим образом:

— Формируем множество L, состоящее из $i \cdot d$ копий $i \cdot v$ для каждого i от 1 до n.
— Задаем случайные паросочетания на множестве L.
— Для вершин u и v из Vs количество ребер в графе G , соединяющее их, равно числу паросочетаний между копиями u и v в L [8].

Сгенерированный таким образом граф соответствует степенной модели $P(a, b)$, описывающей графы, для которых:

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

1.2 Индекс дружбы

С момента появления социальных сетей — Facebook, Vkontakte, LiveJournal, Instagram, LinkedIn, MySpace и т. д. прошло не так много времени, но они уже плотно вошли в повседневную жизнь многих людей.

Опросы показывают, что 76% пользователей Интернета в России (по данным агентства PRT на январь 2014) и примерно 73% жителей Соединенных Штатов являются активными пользователями социальных сетей, и эта цифра растет. [9]

В современном обществе социальные сети становятся огромной базой информации, которую ученые и работодатели все чаще привлекают для решения конкретных задач, будь то научное исследование или оценка кандидата на определенную должность.

Научный интерес к изучению пользователей социальных сетей стремительно растет. На данный момент накоплено большое количество эмпирического материала в отношении характеристик пользователей социальных сетей, который требует систематизации и осмысления.

В социальных сетях часто можно встретить явление именуемое парадоксом дружбы - в среднем соседи некоторого объекта имеют больше соседей, чем сам объект. Оно было обнаружено в 1991 году социологом из государственного университета Нью-Йорка Скоттом Фельдом (англ. Scott L Feld), в процессе изучения социальных сетей [10].

Для наблюдения парадокса дружбы обычно используются несколько метрик. В момент времени t для вершины v_i в графе $G(t) = (V(t), E(t)$ сумма степеней всех соседей v_i равна:

$$s_i(t) = \sum_{j:(v_i,v_j)\in E(t)} deg_j(t),$$

средняя степень соседних вершин $\alpha_i(t)$:

$$\alpha_i(t) = \frac{s_i(t)}{deg_i(t)},$$

а индекс дружбы $\beta_i(t)$ определяется как отношение средней степени соседей v_i к степени самой v_i:

$$\beta_i(t) = \frac{\alpha_i(t)}{deg_i(t)} = \frac{s_i(t)}{deg_i^2(t)} = \frac{\sum_{j:(v_i,v_j)\in E(t)} deg_j(t)}{deg_i^2(t)},$$

таким образом, если средняя степень соседей больше степени v_i и парадокс дружбы выполняется, то $\beta_i(t) > 1$ [11].

Например, социальные сети Facebook и Github подтверждают парадокс дружбы, что показано на рис. 1 [12].

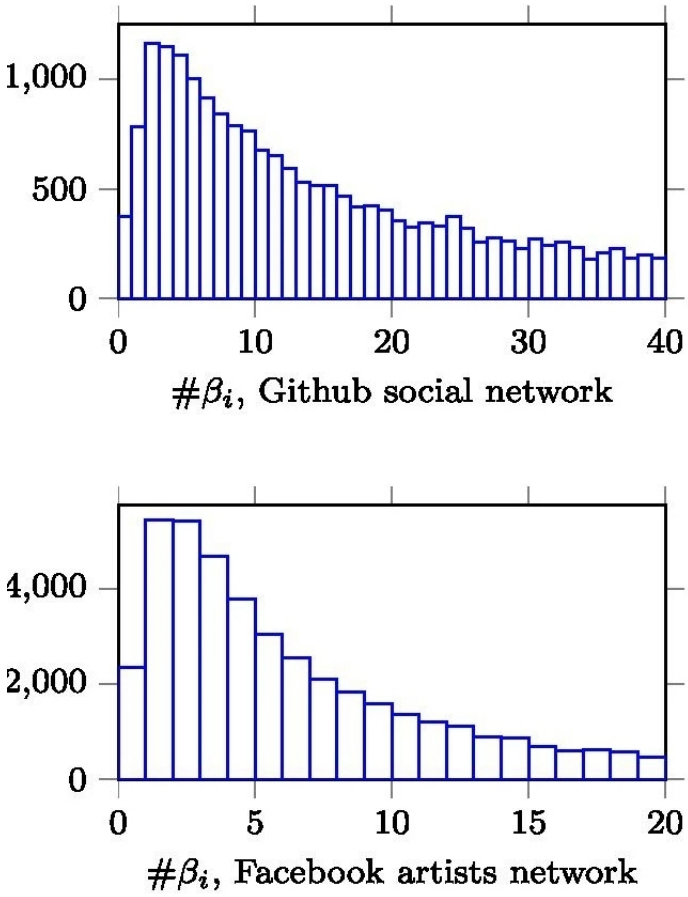

Рисунок 1 – Распределения индекса дружбы в сети телефонных звонков и сети доставки Amazon

2 Реализация модели Барабаши—Альберт

В ходе выполнения курсовой работы была реализована модель Барабаши—Альберт. Все расчёты производились на компьютере с процессором Intel core i5-8265U и 16 ГБ оперативной памяти. Модель реализована на Python 3.9.1 с помощью библиотек networkx [13], multiprocessing, random и numpy.random.

Реализованная модель представляет собой модель растущего случайного графа предпочтительного связывания. В реализованой модификации на каждом шагу добавляется фиксированное количество m рёбер, при этом вероятность появления ребра межу новой гранью и одной из старых пямо пропорциональна колличеству уже существующих соседей старой вершины. В эксперементах в качестве параметра m подставляется число 5, каждый вариант графа строится на 10000 вершин. Так как получаемые графы случайны, то в ходе эксперемента граф сроится шесдесят раз, и строится гистограмма по диапазонам индекса дружбы.

2.1 Реализация стандартной модели Барабаши—Альберт

Реализация стандартной модели Барабаши—Альберт состоит в следующем. Сначала создаётся полный граф из m вершин с помощью команды `nx.complete_graph(m)` . Затем в графе создаются $n - m$ вершин, но рёбра ещё не проводятся. Далее создаются и инициализируются вспомогательные массивы `nodes` , `used` и `degrees` , хранящие список присоединённых к графу вершин, информацию о том использованы они или нет и степени вершин, соответственно. Затем в цикле добавляются ребра, как представлено в следующем коде.

```
    for i in range(m, n):
        if not o :
            conections = []
            j = 0
            while j < m:
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
                    used[choosen] = True
```

Здесь описан цикл по неприсоединённым вершинам (см. стр. 1). Для каждой вершины, с помощью функции `random.choices()`, выбираются m различных вершин с которыми будет соединена новая вершина (см. стр. 6). Для того чтобы не было кратных ребер создаётся массив `connections`, хранящий все выбранные вершины (см. стр. 3, 11).

Следующим шагом с помощью массива `connections` исправляются массивы `used` и `degrees`. И новая вершина добавляется в массив `nodes`.

```
for j in range(m):
  used[conections[j]] = False
  degrees[conections[j] - 1] += 1
  ...
nodeCount += 1
  nodes.append(nodeCount)
  degrees.append(m)
```

Случай добавления первой вершины рассматривается отдельно: если флаг `o` поднят, добавляется грань между нулевым и первым узлами.

```
        else:
            G.add_edge(0, 1)
            o = False
        nodeCount += 1
        nodes.append(nodeCount)
        degrees.append(m)
    return G
```

Далее по такому же алгоритму присоединяются другие вершины. Таким образом, получается алгоритм построения модели сучайного графа:

```
    def my_bag(n, m):
      G = nx.complete_graph(m)
      for i in range(m, n):
          G.add_node(i)
      nodeCount = m
      o = True
      nodes = []
      degrees = []
```

```
    used = []
    for j in range(n):
        used.append(False)
    for i in range(m):
        nodes.append(i)
        degrees.append(m)
    for i in range(m, n):
        if not o :
            conections = []
            j = 0
            while j < m:
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
                    used[choosen] = True
            for j in range(m):
                used[conections[j]] = False
                degrees[conections[j] - 1] += 1
        else:
            G.add_edge(0, 1)
            o = False
        nodeCount += 1
        nodes.append(nodeCount)
        degrees.append(m)
    return G
```

2.2 Реализация метрик для выявления парадокса дружбы

Для проведения эксперементов были реализованы функции `s, alpha, beta` вычисляющие сумму степеней соседей

$$s_i(t) = \sum_{j:(v_i,v_j)\in E(t)} deg_j(t),$$

среднюю степень соседей

$$\alpha_i(t) = \frac{s_i(t)}{deg_i(t)}$$

и индекс дружбы

$$\beta_i(t) = \frac{\alpha_i(t)}{deg_i(t)},$$

соответственно, для массива поданых вершин.

```
def d (G, i):
    # print(i)
    return dict(G.degree)[i]
def s (G, i):
    ans = 0
    for j in G.neighbors(i):
        ans += dict(G.degree)[j]
    return ans
def alfa (G, i):
    return s(G, i) / d(G, i)
def beta (G, i):
    return alfa(G, i) / d(G, i)
```

2.3 Паралелизация построения модели Барабаши—Альберт

Для ускорения работы программы была реализована многопоточная версия алгоритма с помощью библиотек multiprocessing и numpy: функция `run` создаёт потоки, распределяет задания и собирает результаты, а функция `run_thread` проводит эксперементы на каждом ядре отдельно и возвращает их результат.

Сначала в функции `run` создаётся объект класса `multiprocessing.Manager`, который будет управлять в дальнейшем созданными потоками, ссылка на словарь с результатами вычислений - `manager.dict()` хранится в переменной `res`. Затем в цикле для каждого ядра создаётся свой `multyprocessing.Process` который будет выполнять соответсвующий поток, все процессы добавляются в список выполняющихся потоков и запускаются, с помощью метода `p.start`. Далее мы дожидаемся выполнения всех потоков, собираем все результаты в список `ans` и возвращаем его ка результат функции.

Первым шагом в `run` инициализируется пустой список `ans`. Потом поочерёдно строится заданное колличество случайных графов, для каждого из них вычисляется заданная метрика и результат записывается в список `ans`.

```
def run_thread(n, m, N, i, res, args, f):
    ans = []
```

```
    for j in range(N):
        G = my_bag(n, m)
        ans.append(np.apply_along_axis(lambda x: f(G, x[0]),
        ↪ 0,[args]).tolist())
    res[i] = ans
def run(n, m, N, treads, args, f):
    assert N % treads == 0
    global run_thread
    procs = []
    manager = multiprocessing.Manager()
    res = manager.dict()
    for i in range(treads):
        p = multiprocessing.Process(target=run_thread, args=(n, m,
        ↪ math.ceil(N/treads), i, res, args, f))
        procs.append(p)
        p.start()
    for proc in procs:
        proc.join()
    ans = []
    for i in res.values():
        ans += i
    return ans
```

2.4 Вывод и представление данных для анализа

Данная реализация содержит фрагмент кода, отвечающий за отображение данных о построенном графе для дальнейшего анализа. Для этого используются библиотеки matplotlib, matplotlib.pyplot [14] и numpy.

Функция `numpy.histogram` получая на вход массив состоящий из индексов дружбы для всех вершин полученных во всех экспериментах и преобразует его в массив содержащий двенадцать диапазонов и двенадцать счётчиков принадлежащих им индексов. Затем этот массив делится на колличество эксперементов, для нахождения усреднённых результатов, и по нему строится столбчатая диаграмма, с помощью функции `bar` из библиотеки pyplot.

```
ans = np.histogram(run(k, l, 60, 6, np.array(range(k//1)) * 1, beta), bins =
↪ 12)
plt.bar(ans[1][:-1], ans[0] / 60)
plt.show()
```

Полный код программы приведен в приложении А.

3 Анализ

Так как модель Барабаши—Альберт подходит для описания реальных систем, в том числе сети Интернет, то распределение индекса дружбы в этой модели должно соответствовать его распределению в реальных сетях.

Примером могут служить графики распределения индекса дружбы в сети телефонных звонков и сети доставки Amazon, представленные на рис. 2 [15], соответствует гистограмме построенной для алгоритма Барабаши— Альберт (см. рис. 3).

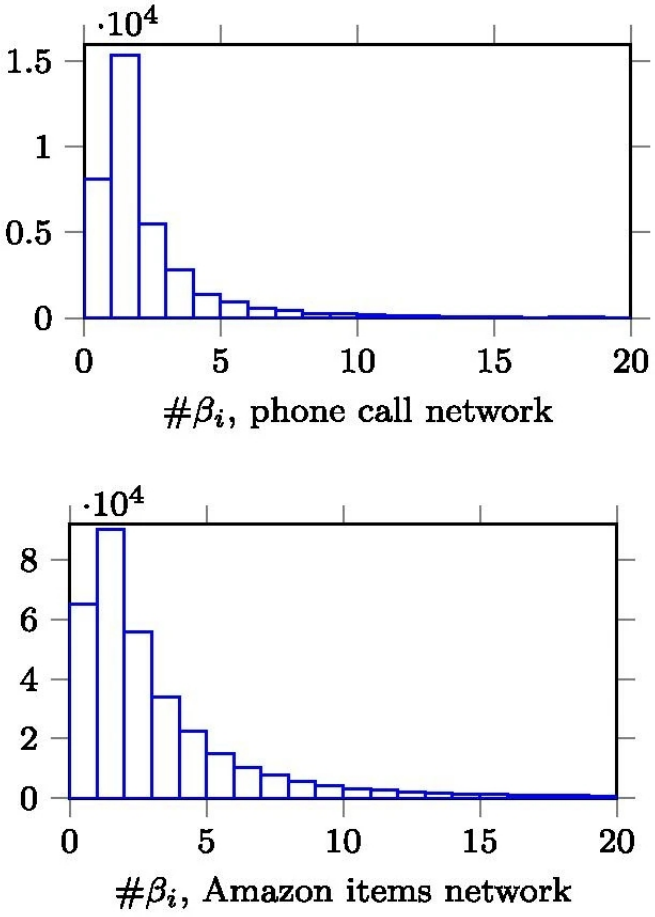

Рисунок 2 – Распределения индекса дружбы в социальных сетях Facebook и Github

Эксперименты, проведенные в ходе выполнения курсовой работы показали, что модель случайного графа Барабаши—Альберт подходит для моделирования процессов, происходящих в социальных сетях и других сложных системах.

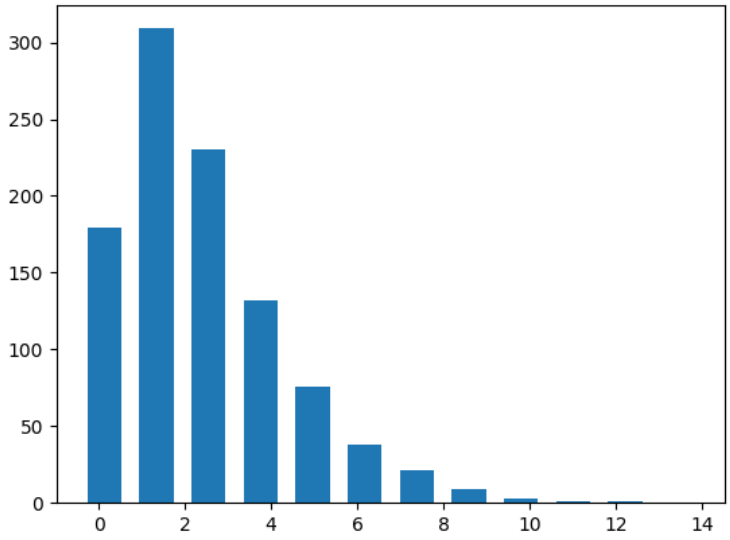

Рисунок 3 – Распределение индекса дружбы в модели Барабаши— Альберт

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены различные модели генерации случайных графов, и проведены исследования стандартной модели Барабаши—Альберт. Проведенные эксперименты показали, модель случайного графа Барабаши—Альберт подходит для моделирования процессов, происходящих в социальных сетях и других сложных системах. Также были изучены различные модули языка программирования Python такие как: networkx(для работы с графами), random и numpy.random(для работы со случайными величинами), multiprocessing(для выполнения паралельных вычислений), а также matplotlib, matplotlib.pyplot и pylab(для построения и отображения графиков).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Erdös, P.* On random graphs i / P. Erdös, A. Rényi // *Publicationes Mathematicae Debrecen*. — 1959. — Vol. 6. — Pp. 290–297.

2 *Gilbert, E. N.* Random graphs / E. N. Gilbert // *Annals of Mathematical Statistics*. — 1959. — Vol. 30, no. 4. — Pp. 1141–1144.

3 Случайные графы [Электронный ресурс]. — URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B5_%D0%B3%D1%80%D0%B0%D1%84%D1%8B (Дата обращения 18.05.2021). Загл. с экр. Яз. рус.

4 ScienceHub 04: Теория случайных графов [Электронный ресурс]. — URL: https://habr.com/ru/company/postnauka/blog/201416/ (Дата обращения 21.05.2021). Загл. с экр. Яз. рус.

5 *Райгородский, А.* Модели случайных графов / А. Райгородский // *ТРУДЫ МФТИ*. — 2010. — Т. 2, № 4. — С. 130.

6 *Albert, R.* Statistical mechanics of complex networks / R. Albert, A.-L. Barabasi // *Reviews of Modern Physics*. — 2002. — Vol. 74, no. 1. — P. 47.

7 *Райгородский, А.* Модели случайных графов и их применени / А. Райгородский. — Москва, М.: Издательство МЦНМО, 2011.

8 *Берновски, М.* Случайные графы, модели и генераторы безмасштабных графов. / М. Берновски, Н. Кузюрин // *Труды Института системного программирования РАН*. — 2012. — Т. 22, № 4. — С. 419.

9 Пользователи социальных сетей: современные исследования [Электронный ресурс]. — URL: https://psychojournal.ru/article/887-polzovateli-socialnyh-setey-sovremennye-issledovaniya.html (Дата обращения 10.03.2022). Загл. с экр. Яз. рус.

10 Парадокс дружбы [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%BE%D0%BA%D1%81_%D0%B4%D1%80%D1%83%D0%B6%D0%B1%D1%8B (Дата обращения 20.04.2022). Загл. с экр. Яз. рус.

11 *Pal, S.* A study on the friendship paradox – quantitative analysis and relationship with assortative mixing. / S. Pal, F. Yu, Y. e. a. Novick // *Appl Netw Sci*. — 2019. — Vol. 74, no. 4. — P. 47.

12 *Sidorov, S.* Friendship paradox in growth networks: analytical and empirical analysis. / S. Sidorov, S. Mironov, A. Grigoriev // *Appl Netw Sci*. — 2021. — Vol. 74, no. 6. — P. 47.

13 NetworkX [Электронный ресурс]. — URL: https://networkx.org/ (Дата обращения 16.05.2021). Загл. с экр. Яз. англ.

14 matplotlib.pyplot [Электронный ресурс]. — URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (Дата обращения 20.05.2021). Загл. с экр. Яз. англ.

15 Network Science by Albert-Laszlo Barabasi [Электронный ресурс]. — URL: http://networksciencebook.com/chapter/4#advanced-a (Дата обращения 20.04.2021). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А
Текст программы

В этом приложении приведён полный текст реализации модели Барабаши—Альберт.

```
from multiprocessing.dummy import current_process
from statistics import mean

import matplotlib as mpl
import matplotlib.pyplot as plt
import networkx as nx
import math
import random
import pylab
import multiprocessing
import datetime

import matplotlib as mpl
import matplotlib.pyplot as plt
import networkx as nx
import math
import numpy as np
import random
import pylab
def my_bag_poisson(n, m):
    m0 = np.random.poisson(m)
    G = nx.complete_graph(m0)
    for i in range(m0, n):
        G.add_node(i)
    nodeCount = m0
    o = True
    nodes = []
    degrees = []
    used = []
    for j in range(n):
        used.append(False)
    for i in range(m0):
        nodes.append(i)
        degrees.append(m0)
    mi = np.random.poisson(m0, n)
```

```
    for i in range(m0, n):
        if not o :
            conections = []
            j = 0
            while j < min(mi[i], nodeCount):
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
                    used[choosen] = True
            for j in range(min(mi[i], nodeCount)):
                used[conections[j]] = False
                degrees[conections[j] - 1] += 1
        else:
            G.add_edge(0, 1)
            o = False
        nodeCount += 1
        nodes.append(nodeCount)
        degrees.append(m)
    return G

def my_bag(n, m):
    # print(n + m)
    G = nx.complete_graph(m)
    for i in range(m, n):
        G.add_node(i)
    nodeCount = m
    o = False
    nodes = []
    degrees = []
    used = []
    for j in range(n):
        used.append(False)
    for i in range(m):
        nodes.append(i)
        degrees.append(m)
    for i in range(m, n):
        if not o :
            conections = []
```

```
            j = 0
            while j < m:
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
                    used[choosen] = True
            for j in range(m):
                used[conections[j]] = False
                degrees[conections[j] - 1] += 1
        else:
            G.add_edge(0, 1)
            o = False
        nodeCount += 1
        nodes.append(nodeCount)
        degrees.append(m)
    return G

def run_thread(n, m, N, i, res, args, f):
    ans = []
    for j in range(N):
        G = my_bag(n, m)
        ans.append(np.apply_along_axis(lambda x: f(G, x[0]),
        ↪ 0,[args]).tolist())
    res[i] = ans
def run(n, m, N, treads, args, f):
    assert N % treads == 0
    global run_thread
    procs = []
    manager = multiprocessing.Manager()
    res = manager.dict()
    for i in range(treads):
        p = multiprocessing.Process(target=run_thread, args=(n, m,
        ↪ math.ceil(N/treads), i, res, args, f))
        procs.append(p)
        p.start()
    for proc in procs:
        proc.join()
    ans = []
```

```
116     for i in res.values():
117         ans += i
118     return ans
119 def d (G, i):
120     # print(i)
121     return dict(G.degree)[i]
122 def s (G, i):
123     ans = 0
124     for j in G.neighbors(i):
125         ans += dict(G.degree)[j]
126     return ans
127 def alfa (G, i):
128     return s(G, i) / d(G, i)
129 def beta (G, i):
130     return alfa(G, i) / d(G, i)
131 if __name__ == "__main__":
132     k = 10000
133     l = 5
134 
135     ans = np.histogram(run(k, l, 60, 6, np.array(range(k//1)) * 1, beta),
        ↪ bins = 12)
136 
137     # ans = np.apply_along_axis(mean, 0, run(k, l, 12, 6,
        ↪ np.array(range(k//1)) * 1, beta))
138     # h =
139     plt.bar(ans[1][:-1], ans[0] / 60)
140     plt.show()
```