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 УЗЛОВ РАСТУЩИХ СЕТЕЙ ПОСТРОЕННЫХ ПО МОДЕЛЯМ С ПРЕДПОЧТИТЕЛЬНЫМ ПРИСОЕДИНЕНИЕМ

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 411 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 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В повседневной жизни для решения многих задач часто используются случайные графы. Случайные графы нашли практическое применение во всех областях, где нужно смоделировать сложные сети. Имеется большое число моделей случайных 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 П. Эрдёшем и А. Реньи в книге 1959 года «On Random Graphs» [1] и независимо американским математиком, Э. 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 случайных 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. Для этого используются модели Эрдёша—Реньи, Барабаши—Альберт, модель триадного замыкания, модель Бьянкони-Барабаши и другие. Все модели основываются на различных свойствах социальных сетей.

В дипломной работе рассматриваются одни из активно используемых и хорошо изученных моделей: модель Барабаши-Альберта и модель триадного замыкания.

Для изучения и анализа моделей случайных графов, анализе социальных явлений и сетей, сообществ и их взаимодействий широко применяется индекс дружбы. Индекс дружбы — один из показателей, используемых в социологии, определяется как отношение средней степени соседей к степени самого объекта.

Целью настоящей работы является анализ индекса дружбы сетей, построенных по модели Барабаши—Альберт. Для достижения этой цели необходимо решить следующие задачи.

— рассмотреть алгоритмы Барабаши—Альберт и триадного замыкания для построения случайного графа;
— реализовать классический алгоритм Барабаши—Альберт, и его модификацию, в которой начальная степень каждого нового узла определяется как случайная величина, заданная пуассоновским распределением;
— реализовать алгоритм триадного замыкания;
— рассмотреть возможность параллельной реализации процесса получения случайного графа и вычисления индекса дружбы;
— организовать серии экспериментов, в которых строятся случайные графы по реализованным моделям;
— провести анализ распределения индекса дружбы построенных графов.

1 Теоретические сведения

Существует множество различных моделей построения случайных графов. Рассмотрим некоторые из них.

1.1 Модель Эрдеша—Ренье

Модель Эрдеша—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 нового ребра, F_n — сигма-множество,Ω_n — множество возможных рёбер ($|\Omega_n \mid = 2^n$), $P_{n,p}\ (G) = p^{|E|}(1-p)^{\binom{n}{2}-|E|}$ — 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].

1.2 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.

В своей модели А.-Л. Барабаши и Р. Альберт предложили стратегию предпочтительного присоединения [6]. Её основная идея заключается в том, что вероятность присоединения конкретной вершины ребром к новой вершине пропорциональна степени данной вершины. Здесь и далее степенью вершины $v_i \in V$ графа $G = (V, E)$ называется количество вершин, напрямую связанных с данной, т.е.

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

Алгоритм формирования сети по модели Барабаши—Альберт заключается в следующем.

1. Первоначально берется полный граф из m вершин, где m — параметр модели.
2. На каждой итерации роста сети добавляется одна новая вершина, которая соединяется m ребрами с уже имеющимися в соответствии с принципом предпочтительного присоединения.

1.3 Модель Боллобаша—Риордана

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

В динамической модификации при добавлении n-ной вершины проводятся n новых рёбер, при этом рёбра могут быть кратными, а также петлями. 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 deg_i — степень вершины i.

Статическая модель (LCD-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 диа-

грамм. Графы построенные по такой модели имеют те же свойства, что и графы построенные по динамической модификации схемы Боллобаша—Риордана.

1.4 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:

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

1.5 Модель триадного замыкания

Помимо стратегии предпочтительного присоединения, модель триадного замыкания использует стратегию формирования триад. Триадное замыкание — свойство социальных систем заключающееся в том, что если между вершинами (A, B) и (A, C), в некоторой социальной сети существует взаимосвязь, то велика вероятность формирования триады из этих трёх вершин, т.е., велика вероятность связи (B, C) [9]. В модели триадного замыкания рост сети происходит следующим образом.

1. Первоначально берется полный граф из m вершин, где m — параметр модели.
2. На каждой итерации роста сети добавляется одна новая вершина, которая соединяется m ребрами с уже имеющимися по следующим правилам:
   — в соответствии с принципом предпочтительного присоединения выбирается вершина, к которой проводится первое ребро;
   — с вероятностью p, где p — параметр модели, выбирается стратегия формирования триады с произвольным соседом вершины, присоединенной первым ребром, или, с вероятностью $(1 - p)$, стратегия предпочтительного присоединения к произвольной вершине графа.

1.6 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 [10].

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 друзья любого человека имеют больше друзей, чем он сам. Оно было обнаружено в 1991 году социологом из государственного университета Нью-Йорка Скоттом Фельдом [11].

Для изучения парадокса дружбы следует ввести несколько обозначений. 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 соседей0 вершины v_i:

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

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 $\beta_i(t) > 1$ [12].

Для примера, социальные сети Facebook и Github подтверждают парадокс дружбы, что показано на Рис. 1 [13]. Здесь на оси Oy отложено количество узлов сети, для которых индекс дружбы β_i попадает в диапазон, отложенный на оси Ox. Как видим, значительное большинство вершин графов имеют значение индекса дружбы большее единицы.

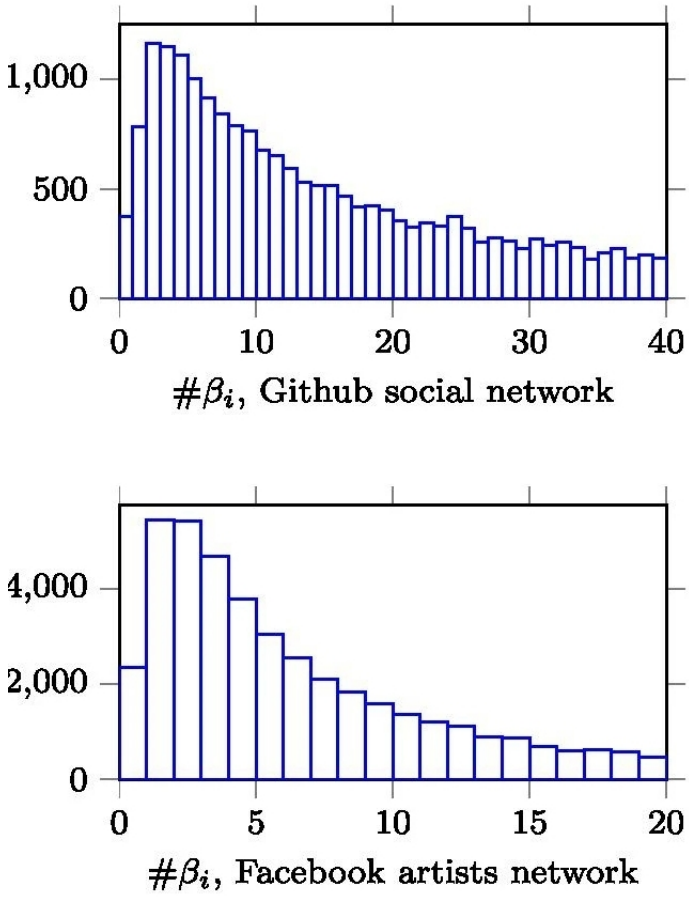

Рисунок 1 – Распределения индекса дружбы в сети телефонных звонков и сети доставки Amazon

2 Реализация моделей

В ходе выполнения дипломной работы были реализованы: стандартная модель Барабаши—Альберт, модель Барабаши—Альберт с пуассоновским распределением начальных степеней и модель триадного замыкания. Все расчёты производились на компьютере с процессором Intel core i5-8265U и 16 ГБ оперативной памяти. Модель реализована на Python 3.9.1. При реализации использовались библиотеки json, multiprocessing, random и numpy.random.

Все три реализованные модели представляют собой модели растущего случайного графа с использованием механизма предпочтительного присоединения. В реализованных модификациях на каждом шаге добавляется одна вершина и некоторое количество m рёбер, при этом параметр m и процесс выбора соседей новой вершины зависят от используемой модели. В модели триадного замыкания используется дополнительный параметр p.

В экспериментах перебираются различные модели и различные значения параметров. В каждом эксперименте строится граф из 100000 вершин. Так как получаемые графы случайны, то в ходе эксперимента граф строится десять раз, после чего строится усреднённая гистограмма по диапазонам значений индекса дружбы и график динамики изменений среднего индекса дружбы графа.

В реализации моделей граф описывается парой массивов `degrees` и `neibours`, хранящие соответственно степень и список номеров соседних вершин для каждой вершины графа. Каждая модель реализована в форме функции `model_name(n, args, funcs, dts)`,
где параметр `n` задаёт размер конечного графа, массив `args` содержит аргументы модели, `funcs` является массивом метрик, которые нужно применить к графу, а `dts` — периодичность применения метрик.

Метрики также представлены функциями вида
`metric_name(degrees, neibours)`,
в которых `degrees` представляет собой массив степеней вершин графа, а `neibours` — массив массивов соседних вершин. В ходе работы были использованы четыре метрики:

- `s(degrees, neibours)` — возвращает массив сумм степеней соседей для вершин графа;
- `alfa(degrees, neibours)` — возвращает массив средних степеней соседей для вершин графа;

— `beta(degrees, neibours)` — возвращает массив индексов дружбы для вершин графа;

— `mean_beta(degrees, neibours)` — возвращает средний индекс дружбы всего графа

2.1 Реализация стандартной модели Барабаши—Альберт

Реализация стандартной модели Барабаши—Альберт состоит в следующем.

Модель реализована в виде функции
`my_bag(n, args, funcs, dts)`,
как описано выше.

В качестве параметра m модели принимается нулевой элемент массива `args`.

В ходе реализации используется массив `ans`, в котором будут хранится значения метрик построенного графа, он заполняется пустыми массивами по количеству метрик:
`ans = [[] for _ in range(len(dts))]`.

Создаётся полный граф из $m + 1$ вершин. Для этого массив `degrees` заполняется `m + 1` значениями `m` и массив `neibours` формируется из `m + 1` списков, каждый из которых содержит номера всех существующих в данный момент вершин в графе, кроме текущей.

```
    degrees = list(np.full(m + 1, m))
    neibours = [list(np.delete(np.arange(m + 1), i)) for i in np.arange(m +
    ↪ 1)]
```

Далее создаются вспомогательные массивы `nodes`, содержащие список номеров всех вершин содержащихся в графа на данный момент, и `used` — массив, в котором отмечаются вершины графа, уже присоединенные к новому узлу.

```
    nodes = np.arange(m + 1)
    used = np.full(n, False)
```

Затем в цикле по $n - (m + 1)$ индексам к графу присоединяются новые вершины. Для хранения соседей нового узла формируется список `conections`. С помощью функции `random.choices` выбираются m уникальных вершин, которые записываются в `conections`. Уникальность вершин определяется по

списку `used`, принцип предпочтительного присоединения соблюдается благодаря передаче степеней вершин в качестве массива весов в `random.choices`.

```
while j < m:
    choosen = random.choices(nodes, weights = degrees, k = 1)[0]
    if not used[choosen]:
        j += 1
        conections.append(choosen)
        used[choosen] = True
```

В следующем цикле по элементам массива `connections` в массиве `used` все `True` заменяются на `False`. Кроме того, значения списка `neibours` дополняются образованными на данном шагу связями, соответственно изменяется массив `degrees`.

```
for j in conections:
    used[j] = False
    neibours[i + 1].append(j)
    neibours[j].append(i)
    degrees[j] += 1
degrees.append(m)
```

Последним этапом добавления новой вершины в данном алгоритме является сбор метрик графа в текущем состоянии. Для каждой функции в списке `funcs` алгоритм определяет, нужно ли снимать данную метрику на данном шаге, основываясь на проверке делимости `i` на соответствующее значение `dts`. Если текущая метрика запрошена пользователем на данном шаге, то результат её обсчёта добавляется в соответствующую ячейку массива `ans`.

```
for j in range(len(dts)):
    if i % dts[j] == 0:
        ans[j].append(funcs[j](degrees, neibours))
```

Результатом выполнения всей функции `my_bag` является `ans`.

2.2 Реализация модели Барабаши—Альберт со случайным числом добавляемых рёбер на каждой итерации

Реализация модели Барабаши—Альберт с пуассоновским распределением степеней новых вершин во многих аспектах повторяет стандартную модель Барабаши—Альберт. Первое отличие заключается в содержимом списка параметров `args`: вместо параметра m данная модель принимает параметр λ для распределения Пуассона. Этот параметр извлекается в переменную `m` и используется для генерации значения `m0` — количества вершин начального полного графа.

```
    m = args[0]
    ...
    m0 = np.random.poisson(m)
    degrees = list(np.full(m0 + 1, m0))
    neibours = [list(np.delete(np.arange(m0 + 1), i)) for i in np.arange(m0 +
    ↪  1)]
```

Начальные степени каждой новой вершины также определяются в соответствии с распределением Пуассона. Так как степень новой вершины не может быть ни больше чем количество узлов графа, ни меньше единицы, то на число соседей новой вершины нужно наложить соответствующие ограничения.

```
    mi = max(1, min(np.random.poisson(m), nodes.shape[0]))
```

2.3 Реализация модели триадного замыкания

Модель триадного замыкания также реализована как модификация стандартной модели Барабаши—Альберт. В отличие от других реализованных моделей список аргументов содержит два параметра: фиксированная степень новых вершин m и вероятность образования триады p, они извлекаются в переменные с соответствующими названиями.

```
    m = args[0]
    p = args[1]
```

Также вносятся определённые изменения в процесс добавления новых вершин: в данной вариации существуют два способа выбора кандидата `choosen` на роль нового соседа текущего узла. Выбор по принципу предпочтительного

присоединения производится либо с вероятностью $1 - p$ из множества всех вершин графа, либо из числа соседей вершины присоединённой первой. Первый узел присоединяется также в соответствии с первым вариантом.

```
if j == 0 or np.random.rand() > p:
    choosen = random.choices(nodes, weights = degrees, k = 1)[0]
else:
    choosen = random.choices(neibours[conections[0]], k = 1)[0]
```

2.4 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 в каждом потоке отдельно и возвращает их результат.

Функция `run` принимает семь аргументов: количество случайных графов, которые требуется построить, `N`; количество потоков `thread`, алгоритм построения случайного графа `model`; размер графов `n`; массив аргументов модели `args`; список метрик `funcs` и частоты их снятия `dts`. Сначала 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 из `thread` ядер создаётся свой `multyprocessing.Process` который будет выполнять соответствующий 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 как результат функции.

Первым шагом в `run_thread` 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, i, res, model, n, args, funcs, dts):
    ans = []
    for j in range(N):
        ans.append(model(n, args, funcs, dts))
    res[i] = ans
```

```
def run(N, treads, model, n, args, funcs, dts):
    procs = []
    manager = multiprocessing.Manager()
    res = manager.dict()
    for i in range(treads):
        curn = min(N, math.ceil(N / (treads - i)))
        p = multiprocessing.Process(target=run_thread, args=(curn, i, res,
        ↪  model, n, args, funcs, dts))
        N -= curn
        procs.append(p)
        p.start()
    for proc in procs:
        proc.join()
    ans = []
    for i in res.values():
        ans += i
    return ans
```

2.5 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.

В ходе проведения экспериментов для каждой модели каждые 100 итераций замерялся средний индекс дружбы вершин графа, а так же после построения графа вычислялся индекс дружбы для каждой из вершин с целью отображения распределения его значений.

Сначала обрабатываются данные финального индекса дружбы, они представлены в виде трёхмерного массива: для каждого построенного графа, для каждого такта снятия метрик, для каждой вершины имеется значения индекса дружбы. Так как данные снимаются лишь однажды то измерение тактов можно убрать оставив от него лишь элементы на нулевых позициях. Затем для каждого графа строится гистограмма, значения столбцов гистограммы усредняются по набору графов. По усреднённой гистограмме строится график и результат сохраняется.

```
ans = run(10, 6, my_triad, n, [m, p], [beta, mean_beta], [n - 1, 100])
beta_data = np.array([i[0] for i in ans])
beta_data = [np.histogram(i[0], bins=100) for i in beta_data]
beta_data = [[i[0], np.delete(i[1], 100)] for i in beta_data]
beta_data = np.array(beta_data)
beta_data = beta_data.transpose((1, 2, 0))
beta_data = np.apply_along_axis(arr = beta_data, axis = 2, func1d = np.mean)
plt.bar(beta_data[1], beta_data[0])
plt.savefig('\\source\\repos\CSW\\diploma_results\\triad_beta_' + str(m) +
    '.jpg')
plt.clf()
```

Схожим образом преобразуется информация о динамике среднего индекса дружбы. Она хранится в массиве `ans` под индексом `1` и выносится в список `mean_beta_data`. Измерение графов и измерение тактов меняются местами. Затем данные усредняются по графам, по ним строится и сохраняется график.

```
mean_beta_data = np.array([i[1] for i in ans])
mean_beta_data = np.apply_along_axis(arr=mean_beta_data, axis=2,
    func1d=lambda x: x[0])
mean_beta_data = mean_beta_data.transpose((1, 0))
mean_beta_data = np.apply_along_axis(arr=mean_beta_data, axis=1, func1d=mean)
plt.bar(mean_beta_data, np.arange(mean_beta_data.shape[0]))
plt.savefig('\\source\\repos\\CSW\\diploma_results\\triad_mean_beta_' +
    str(m) + '.jpg')
plt.clf()
```

Эта последовательность повторяется для каждой модели графа, для каждой комбинации параметров.

```
for m in [3, 5]:
    for p in [0.25, 0.5, 0.75]:
          ans = run(10, 6, my_triad, n, [m, p], [beta, mean_beta], [n - 1,
              100])
      ...
      plt.clf()
for m in [3, 5]:
       ans = run(10, 6, my_bag, n, [m, p], [beta, mean_beta], [n - 1, 100])
      ...
      plt.clf()
```

```
for m in [4, 5, 6]:
        ans = run(10, 6, my_bag_poisson, n, [m, p], [beta, mean_beta], [n - 1,
        ↪ 100])
       ...
       plt.clf()
```

Кроме того, необходимо построить и сохранить в текстовый документ по одному экземпляру каждого графа для дальнейшего изучения. В качестве текстового формата хранения графа был выбран формат json, в силу его структурированности, читабельности для человека, простоты для чтения и записи. В json-файле граф представляется как список словарей, каждый элемент списка представляет вершину графа. Вершина представлена тремя записями словаря:

— `index` — индекс вершины в графе,
— `degree` — степень вершины,
— `neibours` — список соседей узла.

Так как алгоритмы построения графа возвращают не граф целиком, а только значения заданных метрик, то данные графа необходимо извлекать с помощью псевдометрик `d`, `neibours` и `index`, они возвращают массив степеней, матрицу соседей и список индексов графа, соответственно. Эти значения сохраняются в файл.

```
def d (degrees, neibours):
    return degrees
def neibours (degrees, neibours):
    return neibours
def index (degrees, neibours):
    return np.arange(len(neibours))
...
            ans = run(1, 6, my_triad, n, [m, p], [index, d, neibours], [n -
            ↪ 1, n - 1, n - 1])
            ans = ans[0]
            ans = [i[0] for i in ans]
            prejson = [{'index':int(ans[0][i]), 'degree':int(ans[1][i]),
            ↪ 'neibours':[int(j) for j in ans[2][i]]} for i in
            ↪ range(len(ans[0]))]
            with
            ↪ open('\\source\\repos\\CSW\\diploma_results\\triad_graph_'
            ↪ + str(m) + '_' + str(p) + '.json', 'w') as f:
```

```
                json.dump(prejson, f)
                f.close()
```

В результате для каждого графа получается json-файл следующего вида:

```
[{"index": 0, "degree": 799, "neibours": [1, 2, ... , 99033]},
{"index": 1, "degree": 1032, "neibours": [0, 2, 3, ... , 99996]},
{"index": 2, "degree": 418, "neibours": [0, 1, 3, ... , 99864]},
{"index": 3, "degree": 624, "neibours": [0, 1, 2, ... , 99859]},
{"index": 4, "degree": 437, "neibours": [0, 1, 2, ... , 99922]},
{"index": 5, "degree": 174, "neibours": [4, 0, 1, 13, ... , 99982]},
...
{"index": 100000, "degree": 3, "neibours": [22224, 691, 5888]}]
```

Полный код программы приведен в приложении A.

3 Анализ графов реальных сетей

В ходе выполнения дипломной работы были написаны несколько скриптов для считывания из текстового файла и анализа графов реальных сетей:

— скрипт для эффективного считывания больших сетей (до нескольких миллионов вершин и десятки миллионов рёбер) и вычисления распределения индекса дружбы в них;

— программа для чтения графов с информацией о времени появления ребра и отслеживания динамики изменения среднего индекса дружбы;

— скрипт читающий два файла: файл с рёбрами и файл содержащий данные о времени добавления вершины, и также отображающий динамику усреднённого индекса дружбы графа.

3.1 Скрипт для отображения распределения индекса дружбы

Так как скрипт для отображения распределения индекса дружбы должен обрабатывать большие графы при ограниченном количестве памяти, то становится невозможным сохранить всю информацию о графе. Однако для подсчёта индекса дружбы необходимы данные о соседях каждой вершины, потому что в процессе считывания граф постоянно меняется и высчитать индекс дружбы при добавлении вершины или обновлять его динамически становится невозможно. Данная проблема была решена при помощи двукратного прохода по считываемому файлу: один для подсчёта конечных степеней всех вершин, и второй для для передачи этих данных соседям, так данный алгоритм не подразумевает хранения данных о рёбрах графа.

Сначала скрипт открывает на чтение нужный файл, создаются список под значения индекса дружбы — `res` и словарь `degrees` для хранения степеней вершин.

```
1 file = open('\\source\\repos\\CSW\\real_graphs\\' + name + '.txt', 'r')
2 res = []
3 degrees = {}
```

Затем запускается цикл считывания в котором строка из файла записывается в переменную `line`, если строка оказывается равной `null`, то цикл прерывается. Разбивая строку `line` по символу пробела формируется ребро `edge`. Для каждого из двух концов ребра к количеству соседей добавляется 1, если для

вершины ещё не существует записи в `degrees` то она создаётся и инициализируется 0.

```
while True:
    line = file.readline()
    if not line:
        break
    edge = [int(j) for j in line.split(" ")]
    if not edge[0] in degrees:
        degrees[edge[0]] = 0
    degrees[edge[0]] += 1
    if not edge[1] in degrees:
        degrees[edge[1]] = 0
    degrees[edge[1]] += 1
```

Затем курсор переводится в начало файла. Создаётся словарь `sum` для сумм соседей каждой вершины. И запускается второй цикл считывания аналогичный первому, с единственным отличием в том, что он насчитывает суммы соседей, а не степени вершин.

```
file.seek(0)
sums = {}
while True:
    line = file.readline()
    if not line:
        break
    edge = [int(j) for j in line.split(" ")]
    if not edge[0] in sums:
        sums[edge[0]] = 0
    sums[edge[0]] += degrees[edge[1]]
    if not edge[1] in sums:
        sums[edge[1]] = 0
    sums[edge[1]] += degrees[edge[0]]
```

В завершении программы для каждой вершины определяется индекс дружбы, файл закрывается, после чего строится и сохраняется гистограмма распределения индекса дружбы в заданном графе.

```
for i in degrees.keys():
        res.append(sums[i] / degrees[i] / degrees[i])
file.close()
bincnt = 100000
ans = np.histogram(res, bincnt)
x = ans[1]
y = ans[0]
x = np.resize(x, x.size - 1)
plt.bar(x[0:int(bincnt / 100 * 2)], y[0:int(bincnt / 100 * 2)])
plt.savefig("\\source\\repos\\CSW\\diploma_results\\" + name +
↪  "_static.jpg")
plt.clf()
```

Полный код скрипта представлен в приложении Г.

3.2 Скрипт для отображения динамики индекса дружбы

Скрипт для отображения динамики индекса дружбы должен быть способен многократно высчитывать индекс дружбы на считываемом графе, поэтому он сохраняет не только списки соседей вершин, но также и суммы степеней соседей.

Первые шаги напоминают предыдущий алгоритм: файл открывается на чтение и запускается цикл считывания. Однако данные записываются в список `edge_list` практически без обработки — строка просто разбивается по символу `\t`.

```
file = open('\\source\\repos\\CSW\\real_graphs\\' + name, 'r')
    edge_list = []
    while True:
        line = file.readline()
        edge_list.append(line.split("\t"))
        if not line:
            break
    edge_list.pop()
```

Затем элементы `edge_list` преобразуются в тройки чисел содержащие номер первой вершины ребра, номер второй вершины и unix-время добавления грани. Все данные из файла считаны и он закрывается.

```
edge_list = [[i[0],
              i[1],
              datetime.datetime.strptime(i[3], '%Y-%m-%d
               ↪ %H:%M:%S').timestamp()]
             for i in edge_list]
file.close()
```

Полученный массив сортируется с помощью метода `list.sort()` в порядке возрастания временной метки. Инициализируются следующие переменные:

- шаг времени замера `step`,
- точка начала отсчёта времени `base`,
- список для хранения результатов `res`,
- словари которые будут содержать обрабатываемый граф:
  - `degrees` — словарь степеней графа;
  - `neibours_degrees` — содержит суммы степеней соседей;
  - `neibours` — множества самих соседей вершин.

```
edge_list.sort(key = lambda i: i[2])
step = 1000
base = edge_list[0][2]
res = []
degrees = {}
neibours_degrees = {}
neibours = {}
```

Следующим шагом в цикле по отсортированному массиву происходит проверка каждой вершины на наличие в уже описанной части графа, если если вершины ещё нет то для неё инициализируются записи во всех трёх словарях.

```
for i in edge_list:
    cnt += 1
    if not i[0] in degrees:
        degrees[i[0]] = 0
        neibours[i[0]] = set()
        neibours_degrees[i[0]] = 0
    if not i[1] in degrees:
        degrees[i[1]] = 0
        neibours[i[1]] = set()
        neibours_degrees[i[1]] = 0
```

Если вершина является петлёй то степень одного из концов увеличивается на 1, к сумме степеней соседей данной вершины добавляется степень самой вершины, а суммы степеней соседей всех смежных вершин увеличиваются на 1, наконец, происходит попытка добавить новую вершину в множество соседей (очевидно, попытка не удастся, если узел уже содержится в множестве).

```
if i[0] == i[1]:
    degrees[i[0]] += 1
    neibours_degrees[i[0]] += degrees[i[0]]
    for j in neibours[i[0]]:
        neibours_degrees[j] += 1
    neibours[i[0]].add(i[0])
```

В обратном случае к выше указанные шаги проводятся дважды (для каждого конца грани).

```
else:
  degrees[i[0]] += 1
  degrees[i[1]] += 1
  if not i[1] in neibours[i[0]] and not i[0] in neibours[i[1]]:
      neibours_degrees[i[0]] += degrees[i[1]]
      neibours_degrees[i[1]] += degrees[i[0]]
  for j in neibours[i[0]]:
      neibours_degrees[j] += 1
  for j in neibours[i[1]]:
      neibours_degrees[j] += 1
  neibours[i[0]].add(i[1])
  neibours[i[1]].add(i[0])
```

Когда с момента `base` проходит более `step` секунд, `base` обновляется до наименьшего значения превосходящего текущий момент времени. В список `ans` записываются значения индекса дружбы вершин графа. Эти значения усредняются с помощью `numpy.mean()` и среднее добавляется в `res`.

```
if (i[2] - base) >= step:
    base = step * math.ceil(i[2] / step)
    ans = []
    for j in degrees.keys():
```

```
        ans.append(neibours_degrees[j] / degrees[j] / degrees[j])
    res.append(np.mean(ans))
```

По значениям `res` строится и сохраняется график.

```
plt.plot(res)
plt.savefig("\\source\\repos\\CSW\\diploma_results\\" + name +
 ↪ "_iterational_dynamics.jpg")
plt.clf()
```

Так как скрипт не применялся к слишком объёмным графам, то в ходе его выполнения остаётся достаточное количество ресурсов для одновременной обработки нескольких сетей в многопоточном режиме.

```
def run_thread(namegroup, i, res):
    ans = []
    for j in namegroup:
        ans.append(dynamics(j))
    res[i] = ans
def run(names):
    procs = []
    manager = multiprocessing.Manager()
    res = manager.dict()
    for (i, namegroup) in enumerate(names):
        p = multiprocessing.Process(target=run_thread, args=(namegroup, i,
         ↪ res))
        procs.append(p)
        p.start()
    for proc in procs:
        proc.join()
    ans = []
    for i in res.values():
        ans += i
    return ans
run([['askubuntu', 'askubuntu-a2q'], ['askubuntu-c2a', 'askubuntu-c2q'],
 ↪ ['mathoverflow', 'mathoverflow-a2q'],
     ['mathoverflow-c2a', 'mathoverflow-c2q'], ['superuser',
      ↪ 'superuser-a2q'], ['superuser-c2a', 'superuser-c2q']])
```

Полный код скрипта представлен в приложении Д.

3.3 Скрипт для отображения динамики индекса дружбы в графах представленных в нескольких файлах

Данная программа является модификацией предыдущей. Основное отличие заключается в том, что в данном случае время добавления вершин считывается из отдельного файла в словарь `times` .

```
time_file = open('\\source\\repos\\CSW\\real_graphs\\' + name +
↪  '-dates.txt', 'r')
times = {}
while True:
    line = time_file.readline()
    if not line:
        break
    line = line.split("\t")
    line = [int(line[0]), int(line[1].replace('-', ''))]
    times[line[0]] = line[1]
```

Кроме того, сортировка массива степеней `edge_list` проводится с помощью лямбда-функции, сопоставляющей вершину начала ребра с временем её добавления.

```
edge_list.sort(key = lambda i: times[i[0]] if i[0] in times else 100000000)
```

Полный код скрипта представлен в приложении E.

4 Анализ

Для того чтобы проверить, что реализованные модели могут выступать в качестве моделей социальных процессов в окружающем мире, нужно показать, что сгенерированные графы обладают некоторыми свойствами реальных сетей. В качестве характеристики для анализа был выбран индекс дружбы

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

В работе рассматривались распределение индекса дружбы и динамика его среднего значения.

4.1 Анализ распределения и динамики индекса дружбы в построенных графах

На графике, представленном на Рис. 2, изображено распределение индекса дружбы в графе построенном в соответствии с стандартной моделью Барабаши— Альберт с параметром $m = 3$, по этому графику можно сделать вывод о том, что для большинства вершин графа значение индекса дружбы больше единицы. Следовательно в данной модели наблюдается парадокс дружбы.

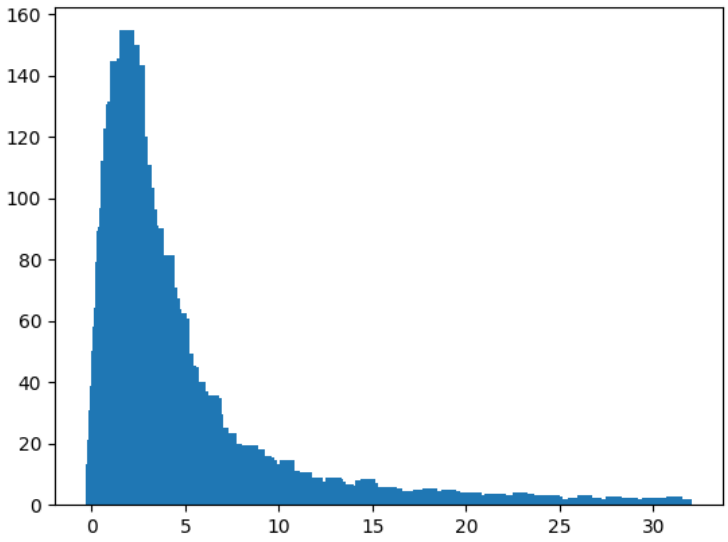

Рисунок 2 – Распределение индекса дружбы в стандартной модели Барабаши— Альберт при $m = 3$

На Рис. 3 показан график распределения индекса дружбы в графе, постро-

енном с помощью модифицированной модели Барабаши— Альберт с параметром $m = 4$. На основании этого графика можно сказать, что в данной модели выполняется парадокс дружбы.

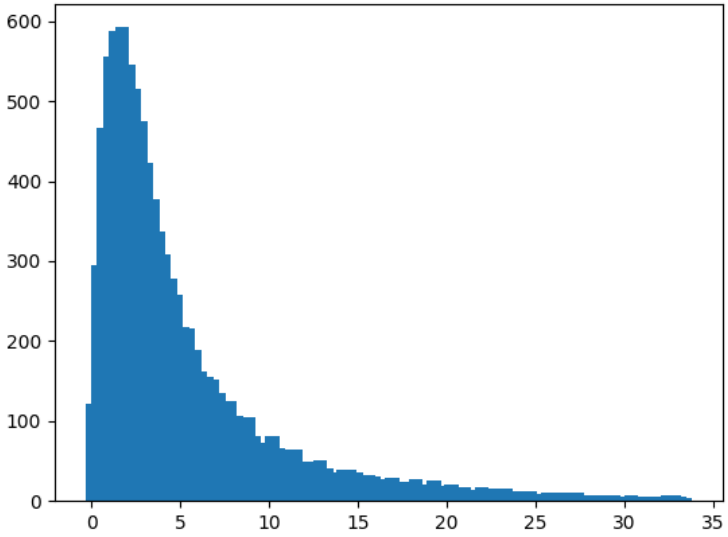

Рисунок 3 – Распределение индекса дружбы в модифицированной модели Барабаши— Альберт при $m = 3$

График, на Рис. 4 показывает распределение индекса дружбы в графе, в основе которого лежит модель триадного замыкания. Этот график также подтверждает присутствие парадокса дружбы в рассматриваемой модели.

На Рис. 5, 6, 7 выведены те же значения, но в логарифмической шкале. В соответствии с графиком, можно выдвинуть гипотезу, что распределение значений индекса дружбы происходит в соответствии со степенной функцией

$$y = (\alpha \cdot x)^{\gamma}.$$

Графики распределения индекса дружбы в этих моделях с другими параметрами m и p представлены в приложении Б.

Рисунок 8 содержит график в логарифмической шкале динамики среднего индекса дружбы в модели Барабаши— Альберт при $m = 3$ в сравнении с графиком степенной функции

$$y = (10000 \cdot x)^{0.12}.$$

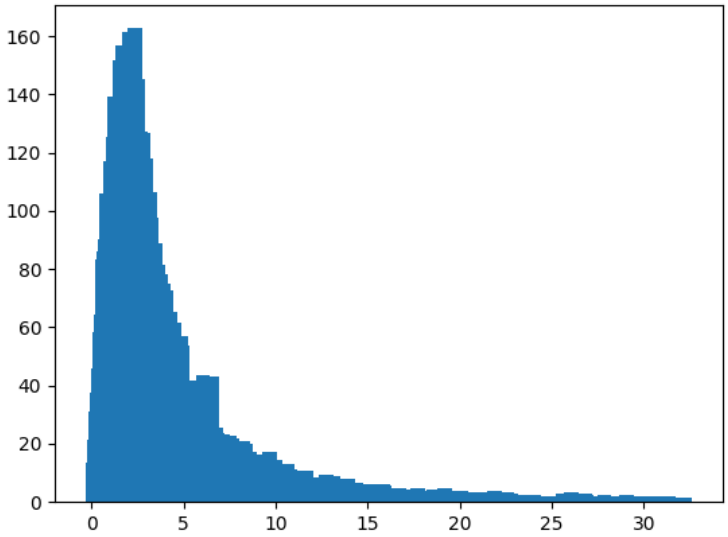

Рисунок 4 – Распределение индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.25$

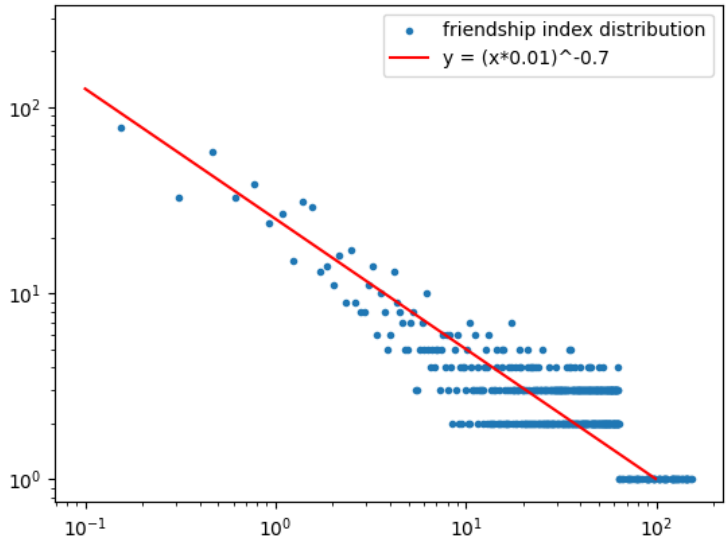

Рисунок 5 – Распределение индекса дружбы в стандартной модели Барабаши— Альберт при $m = 3$, в логарифмической шкале

На основании данного графика можно сделать предположение, что среднее значение индекса дружбы в рассматриваемых моделях растёт в соответствии со степенной функцией.

График динамики среднего индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением начальных степеней вершин при $m = 4$

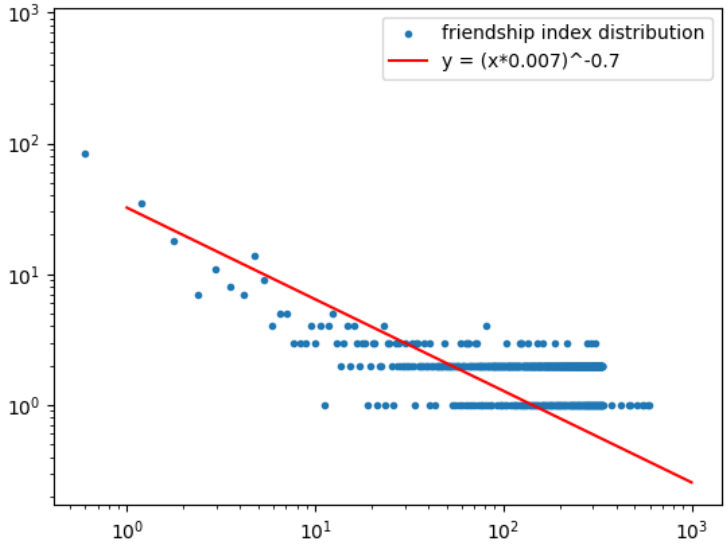

Рисунок 6 – Распределение индекса дружбы в модифицированной модели Барабаши— Альберт при $m = 4$, в логарифмической шкале

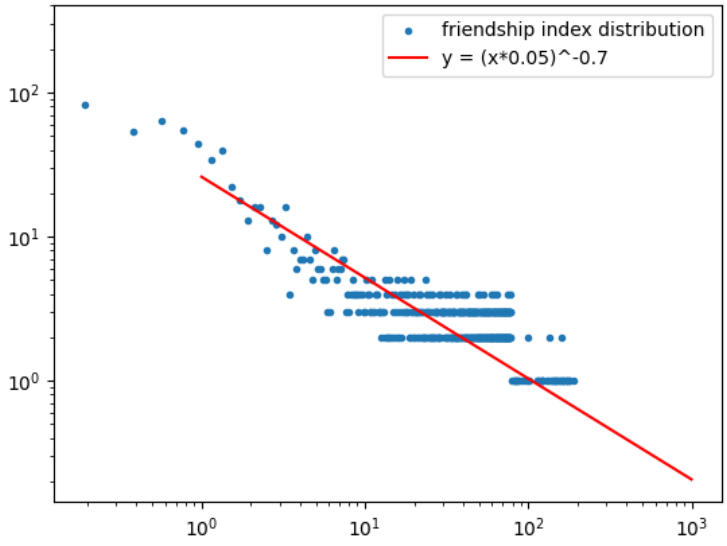

Рисунок 7 – Распределение индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.25$, в логарифмической шкале

в логарифмической шкале, изображённый на Рис. 9, совпадает с графиком степенной функции

$$y = (80000 \cdot x)^{0.13}.$$

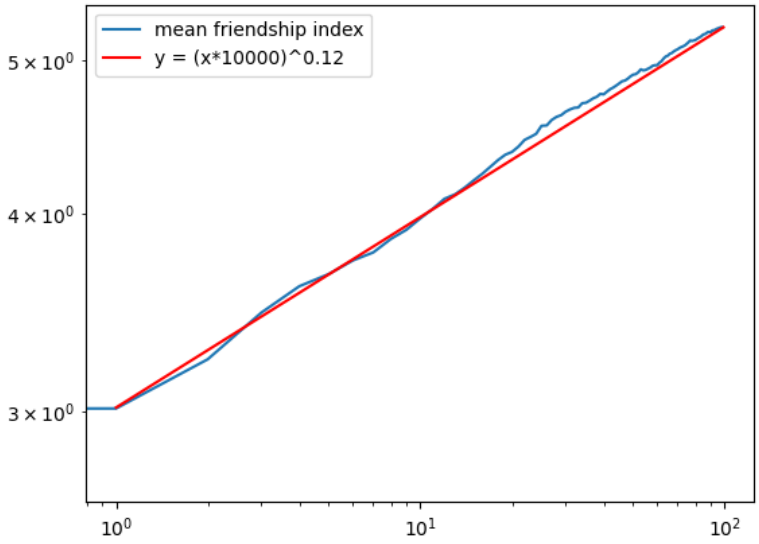

Рисунок 8 – Динамика среднего значения индекса дружбы в стандартной модели Барабаши— Альберт при $m = 3$, в логарифмической шкале

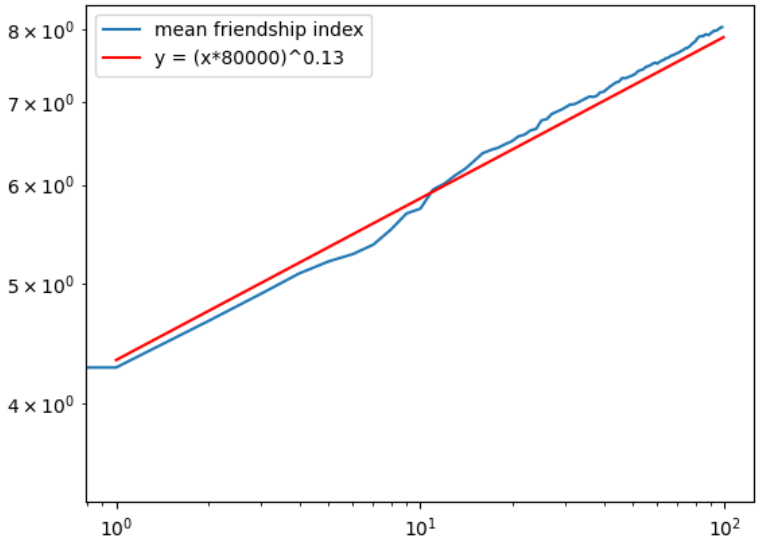

Рисунок 9 – Динамика среднего значения индекса дружбы в модифицированной модели Барабаши— Альберт при $m = 4$, в логарифмической шкале

На Рис. 10 показан график динамики среднего индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.25$ в логарифмической шкале, который идентичен степенной функции:

$$y = (40000 \cdot x)^{0.11}.$$

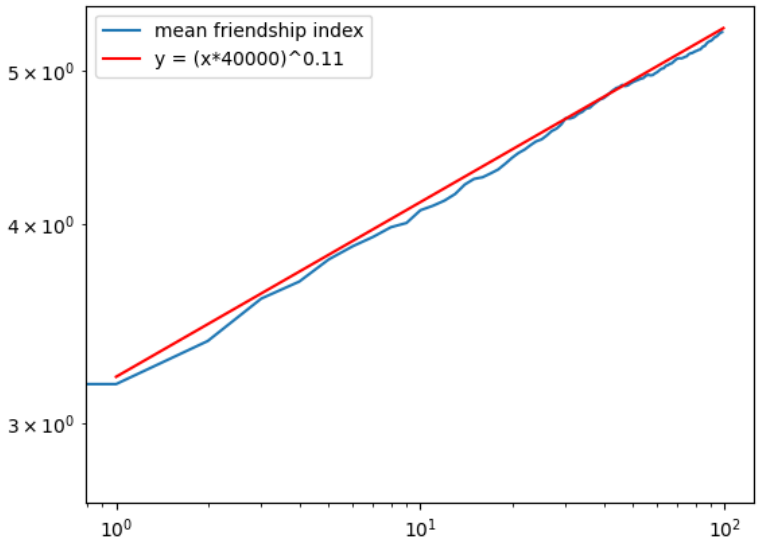

Рисунок 10 – Динамика среднего значения индекса дружбы в модели триадного замыкания при $m = 3$, в логарифмической шкале

Графики динамики индекса дружбы в других моделях представлены в приложении В.

4.2 Анализ распределения и динамики индекса дружбы в реальных сетях

В ходе работы были рассмотрены графы следующих сетей из реального мира:

- LiveJournal
- Twitter
- Reddit
- Google+
- askubuntu
- mathoverflow
- superuser
- сеть цитирования статей в сфере феноменология физики высоких энергий
- внутренняя социальная сеть Калифорнийского университета в Ирвайне

4.2.1 Анализ распределения индекса дружбы в социальной сети LiveJournal

«Живой Журнал», «ЖЖ» (англ. LiveJournal, LJ) — блог-платформа для ведения онлайн-дневников (блогов), а также отдельный персональный блог, размещённый на этой платформе. Предоставляет возможность публиковать свои и комментировать чужие записи, вести коллективные блоги («сообщества»), добавлять в друзья («френдить») других пользователей и следить за их записями в «ленте друзей» («френдленте»).

В работе рассматривался датасет сформированный на основе платформы «Живой Журнал» в 2009 годуи представленный в [15]. В графе в качестве вершин выступают пользователи, а наличие ребра между вершинами указывает на отношение дружбы между соответствующими пользователями. Граф содержит $4\,847\,571$ узлов и $68\,993\,773$ рёбер, что делает его слишком большим для рассмотрения динамики роста среднего индекса дружбы, но с другой стороны позволяет рассмотреть распределение индекса дружбы в крупном графе.

На Рис. 11 изображён график распределения индекса дружбы в описанном графе. Можно сделать предположение, что индекс дружбы распределён по степенному закону

$$y = (0.000002 \cdot x)^{-2}.$$

Отметим, что схожесть со степенным законом распределения значений наблюдается так же и в графах, построенным в соответствии с моделями случайных графов, рассмотренных ранее в работе.

4.2.2 Анализ распределения индекса дружбы в социальной сети Twitter

«Твиттер» (англ. Twitter, от англ. to tweet — «чирикать, щебетать, болтать») — американский сервис микроблогов и социальная сеть, в которой пользователи публикуют сообщения, известные как «твиты», и взаимодействуют с ними. Пользователи взаимодействуют с «Твиттером» через браузер, мобильное приложение или через API. До апреля 2020 года сервисы были доступны через SMS. Сервис предоставляется компанией Twitter, Inc., которая базируется в Сан-Франциско, Калифорния, и имеет более 26 офисов по всему миру. В 2013 году он вошёл в десятку самых посещаемых сайтов и был назван «SMS интернета». К началу 2019 года у «Твиттера» было более 330 миллионов ежемесячных активных пользователей. На практике подавляющее большинство твитов пишет меньшинство пользователей.

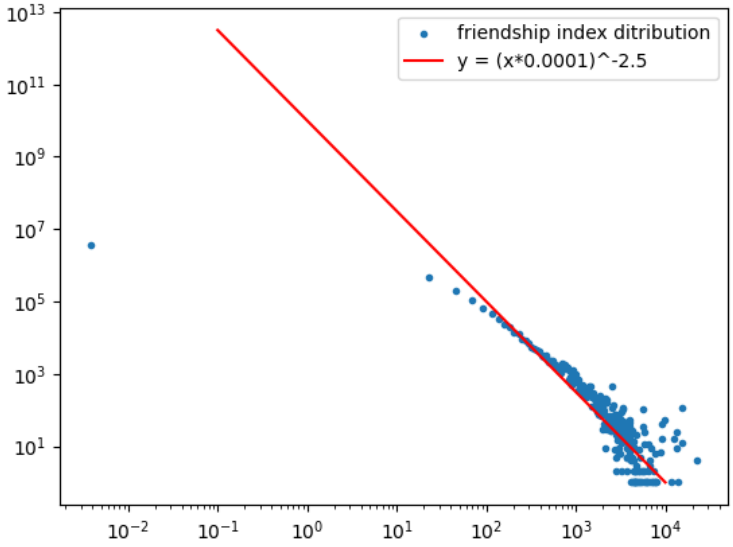

Рисунок 11 – Распределение индекса дружбы в социальной сети «Живой Журнал» в логарифмической шкале

Данные об этой сети взяты из [16]. Граф описывает отношение дружбы между подтверждёнными пользователями, принадлежащим к определённым кругам таким как: «деятели искусства», «политики» и т.д., он содержит 81 306 вершин связанных 1 768 149 рёбрами. На графике, изображённом на Рис. 12 можно обнаружить, что распределение индекса дружбы в данной сети мало соответствует степенной функции.

В данном графе также не удалось проанализировать динамику индекса дружбы.

4.2.3 Анализ распределения и динамики индекса дружбы в сети цитирования статей в сфере феноменология физики высоких энергий

Набор данных содержит информацию о 34 546 статьях на тему физики высоких энергий, опубликованных с января 1993 по апрель 2003 на arXiv.org — электронном архиве с открытым доступом для научных статей и препринтов по физике, математике, астрономии, информатике, биологии, электротехнике, статистике, финансовой математике и экономике. Граф содержит 421 578 рёбер. Датасет был представлен в работе [17].

На Рис. 13 изображено распределение индекса дружбы в данной сети. Из этого графика можно сделать предположение о том, что индекс дружбы в сети цитирования, как и индекс дружбы рассмотренных случайных графах,

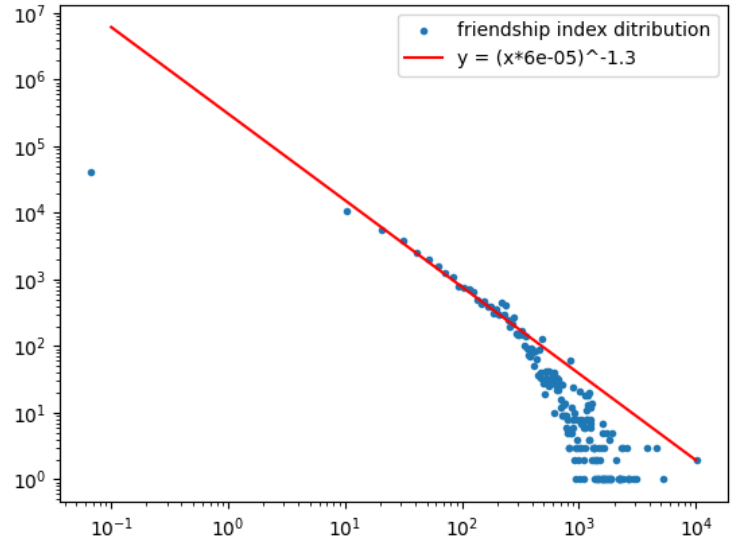

Рисунок 12 – Распределение индекса дружбы в социальной сети «Twitter» в логарифмической шкале

распределён по степенному закону

$$y = (0.0055 \cdot x)^{-2}$$

.

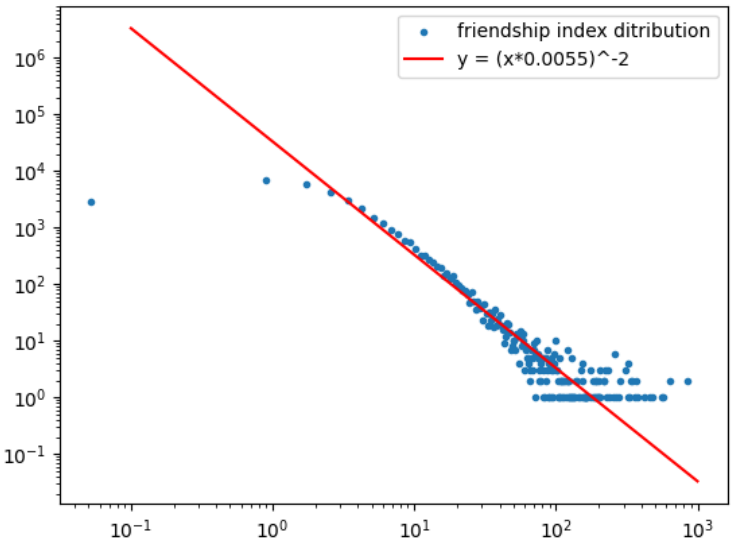

Рисунок 13 – Распределение индекса дружбы в сети цитирования в логарифмической шкале

График динамики индекса дружбы в данной сети представлен на Рис. 14. Можно выдвинуть гипотезу, что среднее значение индекса дружбы в графе растет в соответствии со степенной функцией

$$y = (11000 \cdot x)^{0.12}.$$

Это, в части данного предположения, соответствует росту индекса дружбы в сетях моделей Барабаши— Альберта и триадного замыкания.

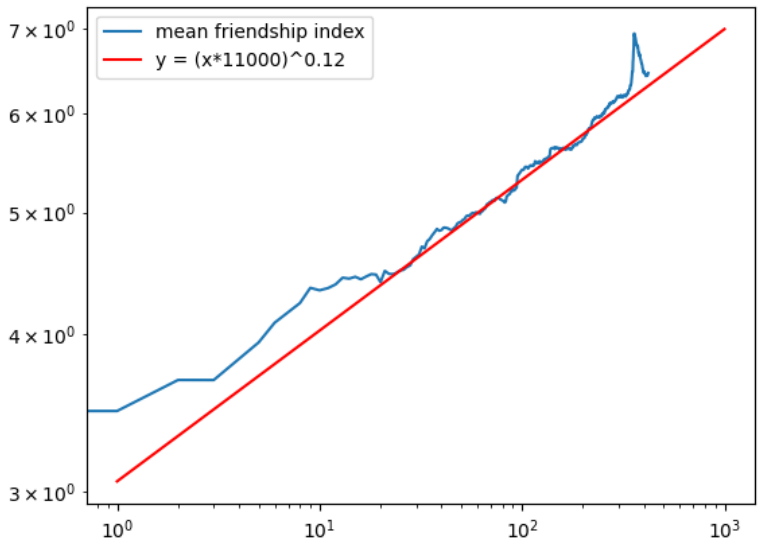

Рисунок 14 – Динамика индекса дружбы в
сети цитирования
в логарифмической шкале

4.2.4 Анализ распределения и динамики индекса дружбы в сети форума Reddit

Reddit (рус. Реддит) — сайт, сочетающий черты социальной сети и форума, на котором зарегистрированные пользователи могут размещать ссылки на какую-либо понравившуюся информацию в интернете и обсуждать её. Как и многие другие подобные сайты, Reddit поддерживает систему голосования за понравившиеся сообщения — наиболее популярные из них оказываются на заглавной странице сайта. Один из наиболее популярных сайтов в мире — 20-е место по посещаемости по данным SimilarWeb.

В данном случае рассматривается сеть гиперссылок между субредитами (сообществами на «Reddit») данные собирались на протяжении двух с половиной

лет: с января 2014 по апрель 2017. Граф состоит из 55 863 узлов, соединённых 858 490 рёбрами. Данные были представлены в [18].

На Рис. 15 представлен график распределения индекса дружбы в графе гиперссылок между субредитами. По графику можно сделать предположение, что распределение индекса дружбы в данной сети также соответствует степенной функции, которая в данном случае выглядит следующим образом:

$$y = (2e^{-6} \cdot x)^{-1}.$$

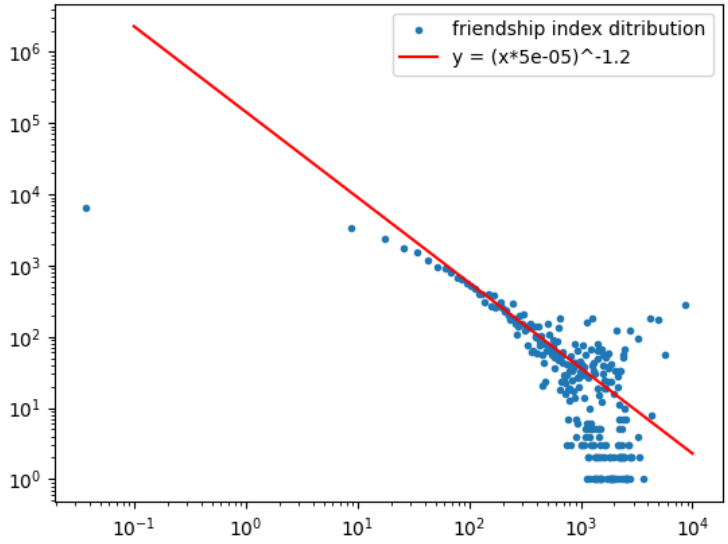

Рисунок 15 – Распределение индекса дружбы в сети форума Reddit в логарифмической шкале

Рис. 16 указывает на сходство графика динамики индекса дружбы на форуме Reddit, с графиком степенной функции

$$y = (0.03 \cdot x)^{0.8}.$$

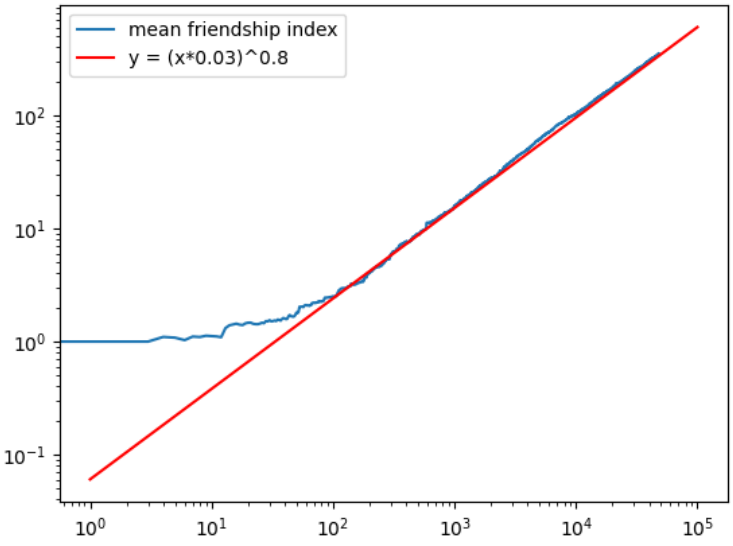

Рисунок 16 – Динамика индекса дружбы в
сети форума Reddit
в логарифмической шкале

4.2.5 Анализ распределения и динамики индекса дружбы в системе вопросов и ответов AskUbuntu

AskUbuntu — веб-сайт для работы с вопросами и ответами возникающими при использовании дистрибутива Linux Ubuntu. Сайты позволяют пользователям задавать вопросы и отвечать на них. Путём учёта активной деятельности, происходит голосование за вопросы и ответы, положительное и отрицательное. Редактировать вопросы и ответы возможно в вики-режиме. Все материалы, публикуемые в сети, доступны под свободной лицензией CC BY-SA.

Были рассмотрены 4 датасета:

— граф ответов на вопросы, который состоит из $137,517$ вершин, представляющих пользователей, и $280\,102$ связей между ними, связь между пользователями A и B заключается в том, что пользователь A ответил на вопрос пользователя B;

— граф цитирования вопросов, содержащий $79\,155$ узлов и $327\,513$ рёбер, где направленное ребро проводится между пользователем и автором вопроса, который он процитировал;

— граф цитирования ответов из $75\,555$ и $356\,822$ вершин и рёбер, соответственно, ребро проводится от пользователя сославшегося на ответ к пользователю его написавшему;

— и общий граф содержащий всех пользователей и все рёбра всех предыдущих графов, в нём $159\,316$ узлов и $964,437$ связей.

Данные для этих графов собирались на протяжении более чем семи лет, они были представлены в статье [19].

Свойства данной сети не полностью соответствуют поведению, изучаемых в дипломной работе моделей: на Рис. 17 можно увидеть, что распределение индекса дружбы плохо соответствует степенной функции и, следовательно, не соответствует построенным моделям.

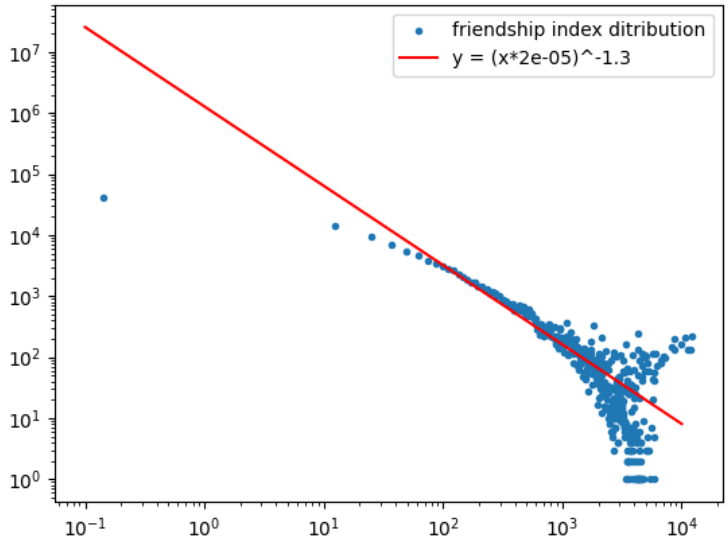

Рисунок 17 – Распределение индекса дружбы в сети AskUbuntu в логарифмической шкале

Однако, среднее значение индекса дружбы, как и в моделях случайных графов, растёт скорее по степенному закону

$$y = (0.8 \cdot x)^{0.5},$$

что показано на графике 18.

Наиболее ярко данные закономерности проявляются в графе сети ответов на вопросы, графики распределения и динамики индекса дружбы в которой изображены на Рис. 19 и 20, соответственно.

Графики других подсетей представлены в приложении Ж

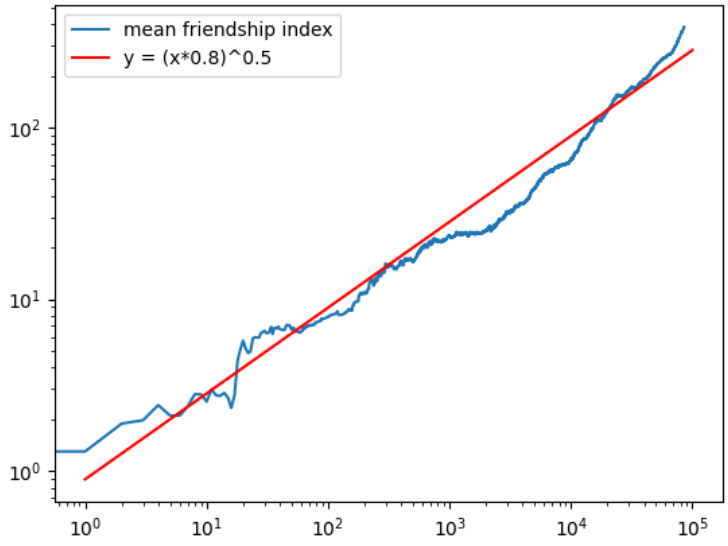

Рисунок 18 – Динамика индекса дружбы в
сети askubuntu
в логарифмической шкале

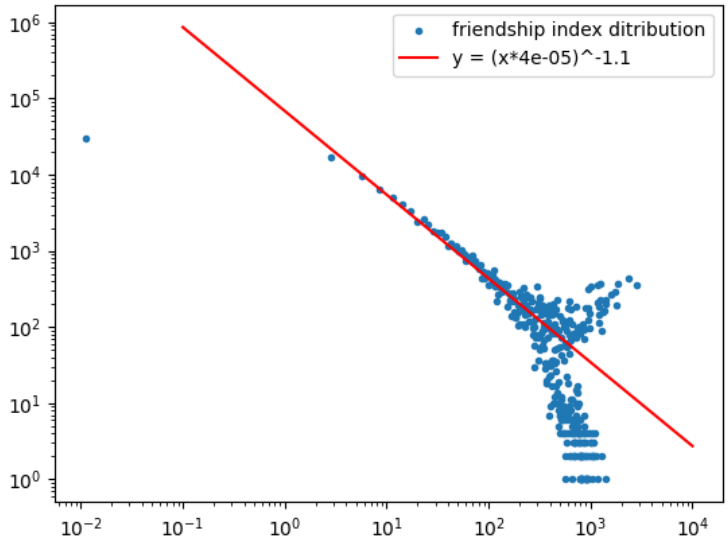

Рисунок 19 – Распределение индекса дружбы в
сети ответов на вопросы на AskUbuntu
в логарифмической шкале

4.2.6 Анализ распределения и динамики индекса дружбы в системе вопросов и ответов SuperUser

SuperUser также как и AskUbuntu является платформой вопросов и ответов, в отличии от предыдущей сети она посвящена компьютерным энтузиастам. Как и AskUbuntu она выступает в качестве одного из разделов StackExchange

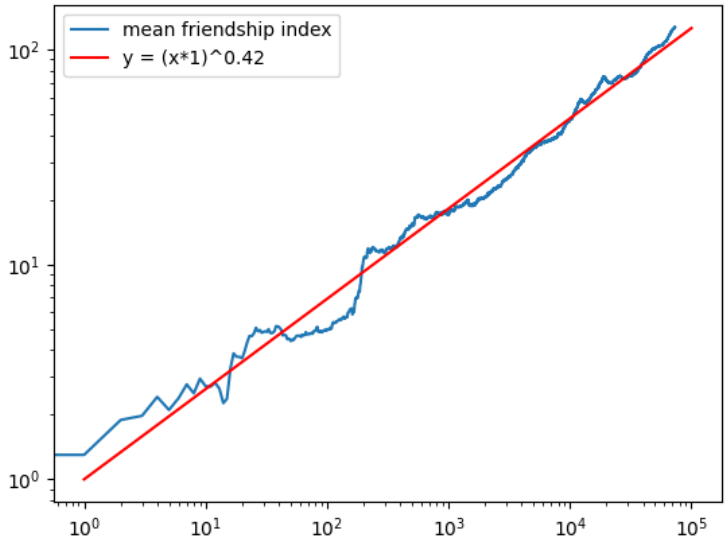

Рисунок 20 – Динамика индекса дружбы в сети ответов на вопросы на AskUbuntu в логарифмической шкале

и она подчиняется тем же базовым правилам сети: любой пользователь может задать вопрос, любой может на него ответить и любой может проголосовать за ответ или вопрос.

Для этой сети были выделены также три подсети:

— сеть ответов на вопросы
— сеть цитирования вопросов
— сеть цитирования ответов

Данные были также представлены в работе [19].

Сеть платформы SuperUser не подчиняется закономерностям выявленным в случайных графах в ходе данной работы. На Рис. 21 можно увидеть, что график распределения индекса дружбы скорее плохо совпадает со степенной функцией, также как и график динамики индекса дружбы представленный на Рис. 22.

В сети цитирования вопросов это отклонение только усиливается, что можно наблюдать на графиках 23 и 24.

В то же время, в подсети ответов на вопросы динамика индекса дружбы всё же, вероятно, подчиняется степенному закону, что можно наблюдать на Рис. 25.

Графики для оставшихся подсетей представлены в приложении Ж, там же расположены графики распределения и динамики индекса дружбы в сетях

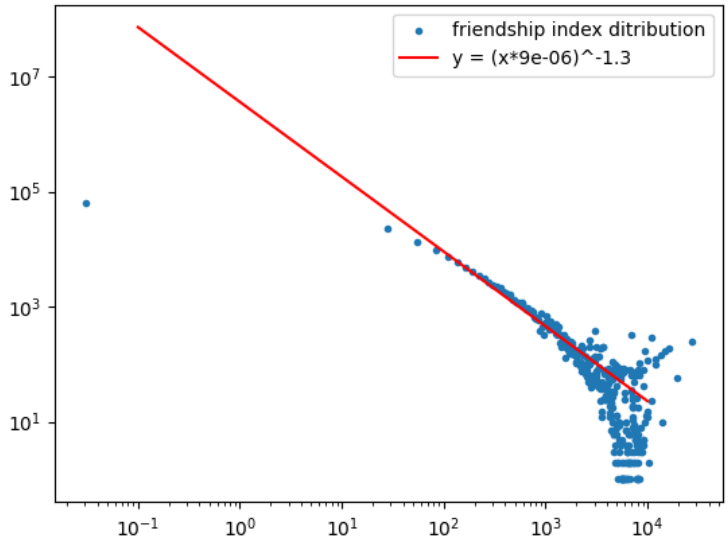

Рисунок 21 – Распределение индекса дружбы в
сети SuperUser
в логарифмической шкале

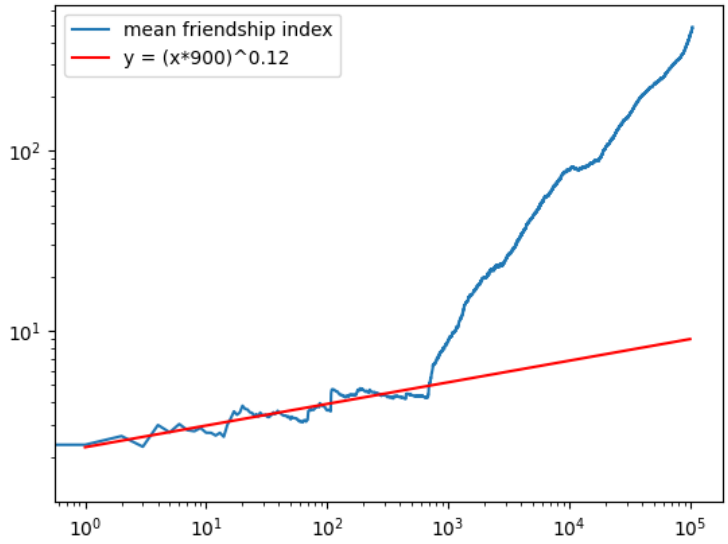

Рисунок 22 – Динамика индекса дружбы в
сети SuperUser
в логарифмической шкале

Google+, mathoverflow и сети сообщений студентов Калифорнийского университета в Ирвайне, данные для которых были взяты из работ [16], [19] и [20]. В таблице1 представлен сравнительный анализ всех рассмотренных сетей.

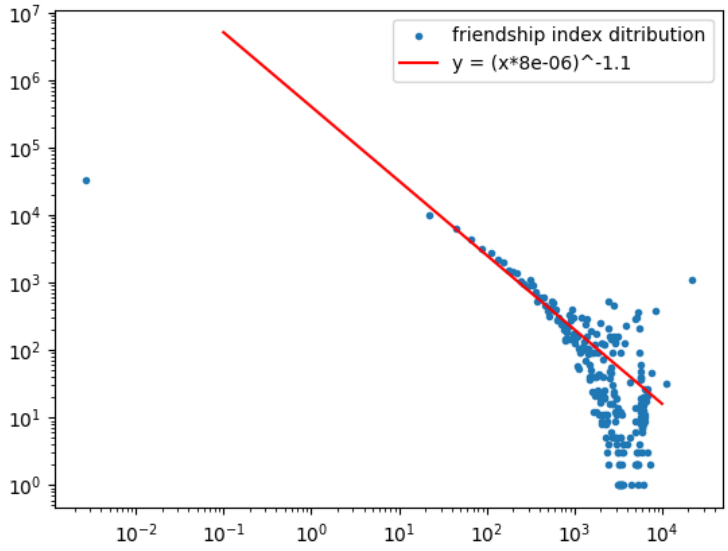

Рисунок 23 – Распределение индекса дружбы в
сети цитирования вопросов на SuperUser
в логарифмической шкале

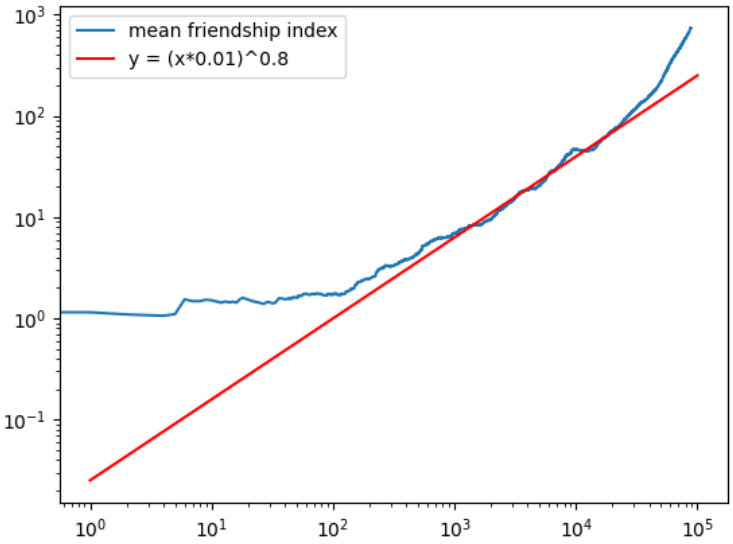

Рисунок 24 – Динамика индекса дружбы в
сети цитирования вопросов на SuperUser
в логарифмической шкале

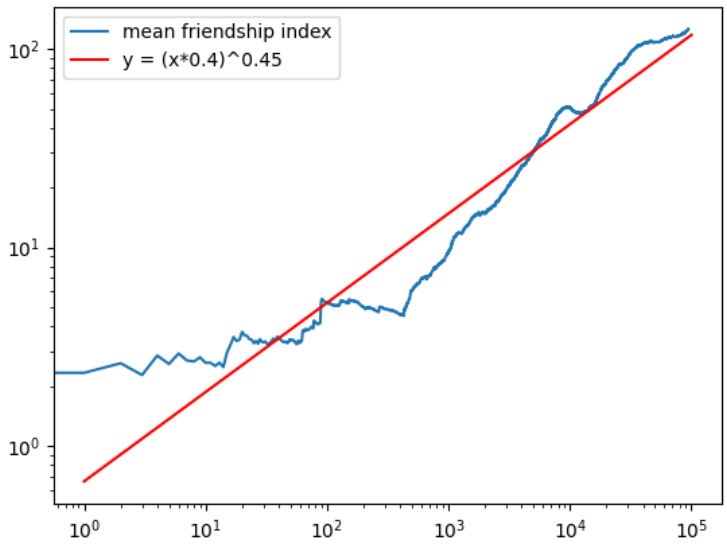

Рисунок 25 – Динамика индекса дружбы в
сети ответов на вопросы на SuperUser
в логарифмической шкале

Таблица 1 – Сводная таблица результатов экспериментов

Сеть	Функция распределения индекса дружбы	γ_{dist}	Функция динамики индекса дружбы	γ_{dyn}
Модель Барабаши— Альберт при $m = 3$	степенная	-0.7	степенная	0.15
Модель Барабаши— Альберт при $m = 3$	степенная	-0.7	степенная	0.15
Модель Барабаши— Альберт с пуассоновским распределением начальных степеней при $m = 4$	степенная	-0.5	степенная	0.15
Модель Барабаши— Альберт с пуассоновским распределением начальных степеней при $m = 5$	степенная	-0.5	степенная	0.15
Модель Барабаши— Альберт с пуассоновским распределением начальных степеней при $m = 6$	степенная	-0.45	степенная	0.15
Модель триадного замыкания при $m = 3$ и $p = 0.25$	степенная	-0.7	степенная	0.15
Модель триадного замыкания при $m = 3$ и $p = 0.5$	степенная	-0.7	степенная	0.15
Модель триадного замыкания при $m = 3$ и $p = 0.75$	степенная	-0.4	степенная	0.12
Модель триадного замыкания при $m = 5$ и $p = 0.25$	степенная	-0.6	степенная	0.13
Модель триадного замыкания при $m = 5$ и $p = 0.5$	степенная	-0.6	степенная	0.13
Модель триадного замыкания при $m = 5$ и $p = 0.75$	степенная	-0.55	степенная	0.12
Twitter	неизвестная функция	N/A	N/A	N/A
Google+	степенная	-1.7	N/A	N/A
Reddit	степенная	-1.2	степенная	0.8
AskUbuntu	неизвестная функция	N/A	степенная	0.5
MathOverflow	степенная	-1.2	степенная	0.45
SuperUser	неизвестная функция	N/A	экспонен-циальная	N/A
Сеть цитирования	степенная	-2	степенная	0.12
LiveJournal	степенная	-2.5	N/A	N/A
Сеть студенческих сообщений	степенная	-0.9	степенная	0.5

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы были изучены различные модели генерации растущих сетей. Были реализованы модели построения случайных графов:

- стандартная модель Барабаши—Альберт;
- модель Барабаши—Альберт с пуассоновским распределением начальных степеней узлов;
- модель триадного замыкания.

В соответствии с реализованными моделями проведена серия экспериментов в которых строились случайные графы и исследовались динамика среднего значения индекса дружбы в сети и распределение значений индекса дружбы в итоговом графе. В соответствии полученными результатами выдвинута гипотеза, что распределение индекса дружбы вершин итогового графа и рост среднего значения индекса дружбы в ходе формирования сети происходит по степенному закону.

Были исследованы значения распределения индекса дружбы и динамики роста среднего значения индекса дружбы в ряде реальных сетей. Сделан вывод, что для большинства сетей поведение индекса дружбы схоже с его поведением в сетях растущих по исследуемым моделям.

Реализация моделей и эксперименты проводились на языке python с использованием библиотек json, multiprocessing, pyplot, numpy, random.

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 *Blum, A.* Foundations of data science / A. Blum, J. E. Hopcroft, R. Kannan. — 2020.

4 *Bollobas, B.* Random Graphs / B. Bollobas. Cambridge Studies in Advanced Mathematics. — 2 edition. — Cambridge University Press, 2001.

5 *Райгородский, А. М.* Модели случайных графов / А. М. Райгородский // *Труды МФТИ*. — 2010. — Т. 2, № 4. — С. 130 – 140.

6 *Albert, R.* Statistical mechanics of complex networks / R. Albert, A.-L. Barabasi // *Reviews of Modern Physics*. — 2002. — Vol. 74, no. 1. — Pp. 47 – 98.

7 *Райгородский, А. М.* Модели случайных графов и их применени / А. М. 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 – 432.

9 *David, E.* Networks, Crowds, and Markets: Reasoning About a Highly Connected World / E. David, K. Jon. — USA: Cambridge University Press, 2010.

10 Пользователи социальных сетей: современные исследования [Электронный ресурс]. — URL: https://psychojournal.ru/article/887-polzovateli-socialnyh-setey-sovremennye-issledovaniya.html (Дата обращения 10.03.2023). Загл. с экр. Яз. рус.

11 *Feld, S. L.* Why your friends have more friends than you do / S. L. Feld // *American Journal of Sociology*. — 1991. — Vol. 96. — Pp. 1464 – 1477.

12 *Pal, S.* A study on the friendship paradox quantitative analysis and relationship with assortative mixing / S. Pal, F. Yu, Y. Novick, A. Swami, A. Bar-Noy // *Applied Network Science*. — 09 2019. — Vol. 4. — Pp. 71 – 118.

13 *Sidorov, S.* Friendship paradox in growth networks: analytical and empirical analysis / S. Sidorov, S. Mironov, A. Grigoriev // *Appl Netw Sci*. — 2023. — Vol. 74, no. 6. — Pp. 47 – 51.

14 matplotlib.pyplot [Электронный ресурс]. — URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (Дата обращения 20.05.2023). Загл. с экр. Яз. англ.

15 *Leskovec, J.* Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters / J. Leskovec, K. J. Lang, A. Dasgupta, M. W. Mahoney // *Internet Mathematics*. — 2009. — Vol. 6, no. 1. — Pp. 29–123. https://doi.org/10.1080/15427951.2009.10129177.

16 *Mcauley, J.* Learning to discover social circles in ego networks / J. Mcauley, J. Leskovec // *NIPS*. — 01 2012. — Vol. 1. — Pp. 539–547.

17 *Leskovec, J.* Graphs over time: Densification laws, shrinking diameters and possible explanations / J. Leskovec, J. Kleinberg, C. Faloutsos // Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. — New York, NY, USA: Association for Computing Machinery, 2005. — KDD '05. — Pp. 177 – 187. https://doi.org/10.1145/1081870.1081893.

18 *Kumar, S.* Community interaction and conflict on the web / S. Kumar, W. L. Hamilton, J. Leskovec, D. Jurafsky // Proceedings of the 2018 World Wide Web Conference. — Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018. — WWW '18. — Pp. 933 – 943. https://doi.org/10.1145/3178876.3186141.

19 *Paranjape, A.* Motifs in temporal networks / A. Paranjape, A. R. Benson, J. Leskovec // *CoRR*. — 2016. — Vol. abs/1612.09259. http://arxiv.org/abs/1612.09259.

20 *Panzarasa, P.* Patterns and dynamics of users' behavior and interaction: Network analysis of an online community / P. Panzarasa, T. Opsahl, K. Carley // *JASIST*. — 05 2009. — Vol. 60. — Pp. 911–932.

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 math
import random
import pylab
import multiprocessing
import datetime
import numpy as np
import json

def my_bag_poisson(n, args, funcs, dts):
    m = args[0]
    ans = [[] for _ in range(len(dts))]
    m0 = np.random.poisson(m)
    degrees = list(np.full(m0 + 1, m0))
    neibours = [list(np.delete(np.arange(m0 + 1), i)) for i in np.arange(m0 + 1)]
    nodes = np.arange(m0 + 1)
    used = np.full(n, False)
    for i in range(m0, n):
        mi = max(1, min(np.random.poisson(m), nodes.shape[0]))
        conections = []
        j = 0
        while j < mi:
            choosen = random.choices(nodes, weights = degrees, k = 1)[0]
            if not used[choosen]:
                j += 1
                conections.append(choosen)
                used[choosen] = True
        neibours.append([])
        for j in conections:
            used[j] = False
```

```
            neibours[i + 1].append(j)
            neibours[j].append(i)
            degrees[j] += 1
        degrees.append(mi)
        nodes = np.append(nodes, nodes.shape[0])
        for j in range(len(dts)):
            if i % dts[j] == 0:
                ans[j].append(funcs[j](degrees, neibours))
    return ans

def my_triad(n, args, funcs, dts):
    m = args[0]
    p = args[1]
    ans = [[] for _ in range(len(dts))]
    degrees = list(np.full(m + 1, m))
    neibours = [list(np.delete(np.arange(m + 1), i)) for i in np.arange(m +
    ↪ 1)]
    nodes = np.arange(m + 1)
    used = np.full(n, False)
    for i in range(m, n):
        conections = []
        j = 0
        while j < m:
            if j == 0 or np.random.rand() > p:
                choosen = random.choices(nodes, weights = degrees, k = 1)[0]
            else:
                choosen = random.choices(neibours[conections[0]], k = 1)[0]
            if not used[choosen]:
                j += 1
                conections.append(choosen)
                used[choosen] = True
        neibours.append([])
        for j in conections:
            used[j] = False
            neibours[i + 1].append(j)
            neibours[j].append(i)
            degrees[j] += 1
        degrees.append(m)
        nodes = np.append(nodes, nodes.shape[0])
        for j in range(len(dts)):
            if i % dts[j] == 0:
```

```
                ans[j].append(funcs[j](degrees, neibours))
    return ans


def my_bag(n, args, funcs, dts):
    m = args[0]
    ans = [[] for _ in range(len(dts))]
    degrees = list(np.full(m + 1, m))
    neibours = [list(np.delete(np.arange(m + 1), i)) for i in np.arange(m +
    ↪  1)]
    nodes = np.arange(m + 1)
    used = np.full(n, False)
    for i in range(m, n):
        conections = []
        j = 0
        while j < m:
            choosen = random.choices(nodes, weights = degrees, k = 1)[0]
            if not used[choosen]:
                j += 1
                conections.append(choosen)
                used[choosen] = True
        neibours.append([])
        for j in conections:
            used[j] = False
            neibours[i + 1].append(j)
            neibours[j].append(i)
            degrees[j] += 1
        degrees.append(m)
        nodes = np.append(nodes, nodes.shape[0])
        for j in range(len(dts)):
            if i % dts[j] == 0:
                ans[j].append(funcs[j](degrees, neibours))
    return ans

def run_thread(N, i, res, model, n, args, funcs, dts):
    ans = []
    for j in range(N):
        ans.append(model(n, args, funcs, dts))
    res[i] = ans
def run(N, treads, model, n, args, funcs, dts):
    procs = []
```

```
115     manager = multiprocessing.Manager()
116     res = manager.dict()
117     for i in range(treads):
118         curn = min(N, math.ceil(N / (treads - i)))
119         p = multiprocessing.Process(target=run_thread, args=(curn, i, res,
            ↪  model, n, args, funcs, dts))
120         N -= curn
121         procs.append(p)
122         p.start()
123     for proc in procs:
124         proc.join()
125     ans = []
126     for i in res.values():
127         ans += i
128     return ans
129 def d (degrees, neibours):
130     return degrees
131 def neibours (degrees, neibours):
132     return neibours
133 def index (degrees, neibours):
134     return np.arange(len(neibours))
135 def s (degrees, neibours):
136     return [sum([degrees[i] for i in j]) for j in neibours]
137 def alfa (degrees, neibours):
138     return [si / di for (si, di) in zip(s(degrees, neibours), d(degrees,
        ↪  neibours))]
139 def beta (degrees, neibours):
140     return [ai / di for (ai, di) in zip(alfa(degrees, neibours), d(degrees,
        ↪  neibours))]
141 def mean_beta (degrees, neibours):
142     return [mean(beta(degrees, neibours))]
143
144 if __name__ == "__main__":
145     n = 100000
146     m = 5
147     p = 0.25
148
149     for m in [3, 5]:
150         for p in [0.25, 0.5, 0.75]:
151             ans = run(1, 6, my_triad, n, [m, p], [index, d, neibours], [n -
                ↪  1, n - 1, n - 1])
```

```
152             ans = ans[0]
153             ans = [i[0] for i in ans]
154             prejson = [{'index':int(ans[0][i]), 'degree':int(ans[1][i]),
                ↪  'neibours':[int(j) for j in ans[2][i]]} for i in
                ↪  range(len(ans[0]))]
155             with
                ↪  open('\\source\\repos\\CSW\\diploma_results\\triad_graph_'
                ↪  + str(m) + '_' + str(p) + '.json', 'w') as f:
156                 json.dump(prejson, f)
157                 f.close()
158
159     for m in [4, 5, 6]:
160         ans = run(10, 6, my_bag_poisson, n, [m, p], [index, d, neibours], [n
            ↪  - 1, n - 1, n - 1])
161         ans = ans[0]
162         ans = [i[0] for i in ans]
163         prejson = [{'index':int(ans[0][i]), 'degree':int(ans[1][i]),
            ↪  'neibours':[int(j) for j in ans[2][i]]} for i in
            ↪  range(len(ans[0]))]
164         with open('\\source\\repos\\CSW\\diploma_results\\bap_graph_' +
            ↪  str(m) + '.json', 'w') as f:
165             json.dump(prejson, f)
166             f.close()
167
168
169     for m in [3, 5]:
170         ans = run(1, 6, my_bag, n, [m, p], [index, d, neibours], [n - 1, n -
            ↪  1, n - 1])
171         ans = ans[0]
172         ans = [i[0] for i in ans]
173         prejson = [{'index':int(ans[0][i]), 'degree':int(ans[1][i]),
            ↪  'neibours':[int(j) for j in ans[2][i]]} for i in
            ↪  range(len(ans[0]))]
174         with open('\\source\\repos\\CSW\\diploma_results\\ba_graph_' +
            ↪  str(m) + '.json', 'w') as f:
175             json.dump(prejson, f)
176             f.close()
177
178
179     for m in [4, 6]:
180         ans = run(1, 6, my_bag_poisson, n, [m], [mean_beta], [100])
```

```
181         mean_beta_data = np.array([i[0] for i in ans])
182         mean_beta_data = np.apply_along_axis(arr=mean_beta_data, axis=2,
            ↪  func1d=lambda x: x[0])
183         mean_beta_data = mean_beta_data.transpose((1, 0))
184         mean_beta_data = np.apply_along_axis(arr=mean_beta_data, axis=1,
            ↪  func1d=mean)
185         plt.plot(np.arange(mean_beta_data.shape[0]), mean_beta_data)
186 
            ↪  plt.savefig('\\source\\repos\\CSW\\diploma_results\\bap_mean_beta_'
            ↪  + str(m) + '.jpg')
187         plt.clf()
188 
189 
190     for m in [3]:
191         ans = run(10, 6, my_bag, n, [m], [mean_beta], [100])
192         mean_beta_data = np.array([i[0] for i in ans])
193         mean_beta_data = np.apply_along_axis(arr=mean_beta_data, axis=2,
            ↪  func1d=lambda x: x[0])
194         mean_beta_data = mean_beta_data.transpose((1, 0))
195         mean_beta_data = np.apply_along_axis(arr=mean_beta_data, axis=1,
            ↪  func1d=mean)
196         plt.plot(np.arange(mean_beta_data.shape[0]), mean_beta_data)
197         plt.savefig('\\source\\repos\\CSW\\diploma_results\\ba_mean_beta_'
            ↪  + str(m) + '.jpg')
198         plt.clf()
```

ПРИЛОЖЕНИЕ Б

Графики распределения индекса дружбы в построенных графах

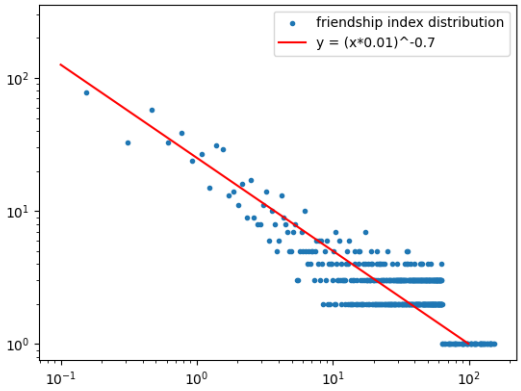

Рисунок 26 – Распределение индекса дружбы в модели Барабаши— Альберт при $m = 3$

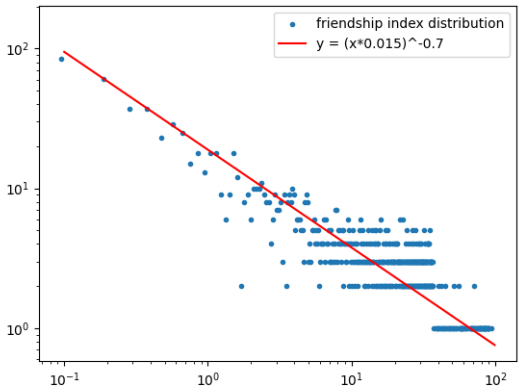

Рисунок 27 – Распределение индекса дружбы в модели Барабаши— Альберт при $m = 5$

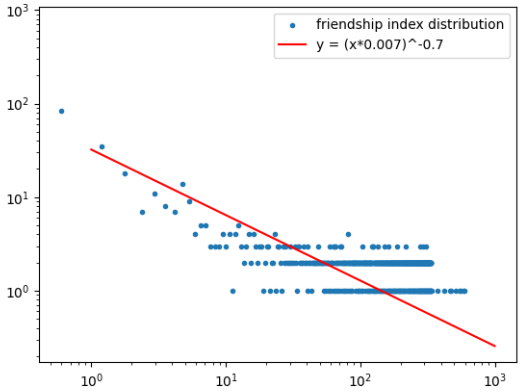

Рисунок 28 – Распределение индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением степеней новых вершин при $\lambda = 4$

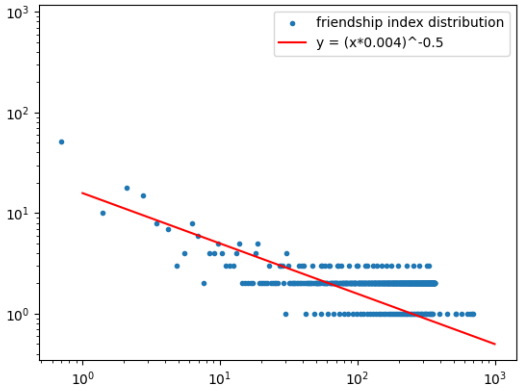

Рисунок 29 – Распределение индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением степеней новых вершин при $\lambda = 5$

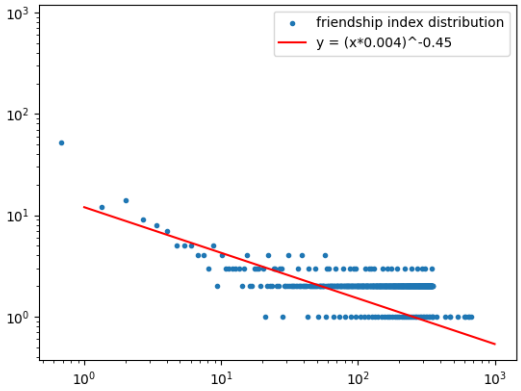

Рисунок 30 – Распределение индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением степеней новых вершин при $\lambda = 6$

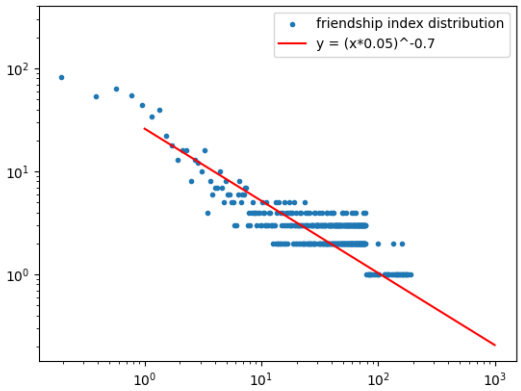

Рисунок 31 – Распределение индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.25$

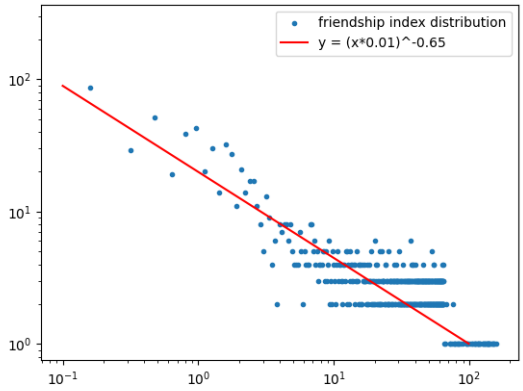

Рисунок 32 – Распределение индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.5$

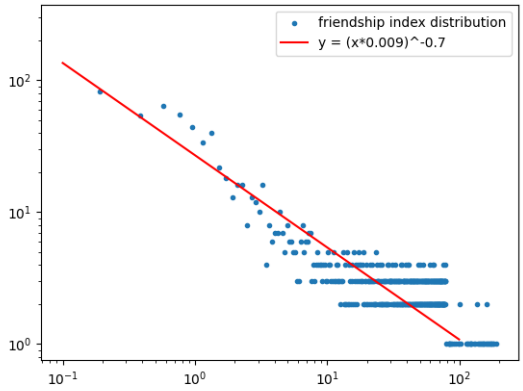

Рисунок 33 – Распределение индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.75$

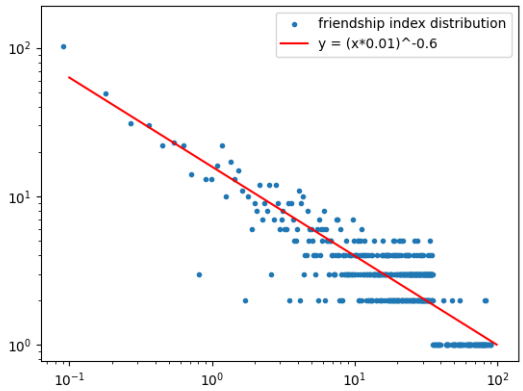

Рисунок 34 – Распределение индекса дружбы в модели триадного замыкания при $m = 5$ и $p = 0.25$

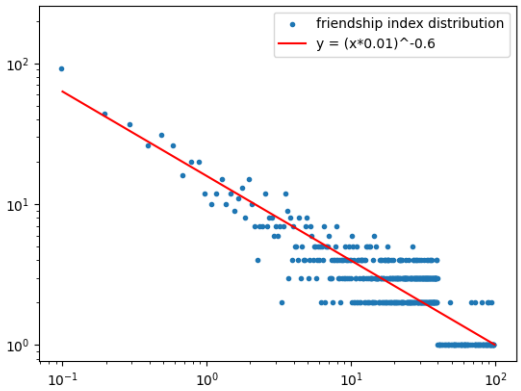

Рисунок 35 – Распределение индекса дружбы в модели триадного замыкания при $m = 5$ и $p = 0.5$

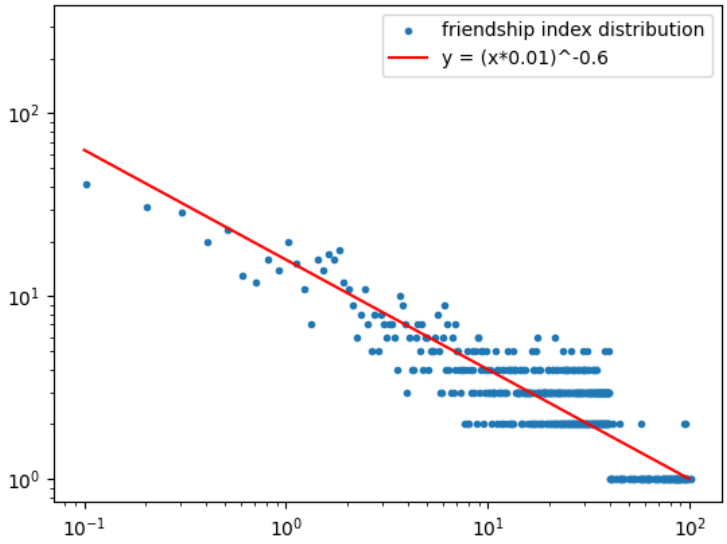

Рисунок 36 – Распределение индекса дружбы в модели триадного замыкания при $m = 5$ и $p = 0.75$

ПРИЛОЖЕНИЕ В

Графики динамики среднего индекса дружбы в построенных графах

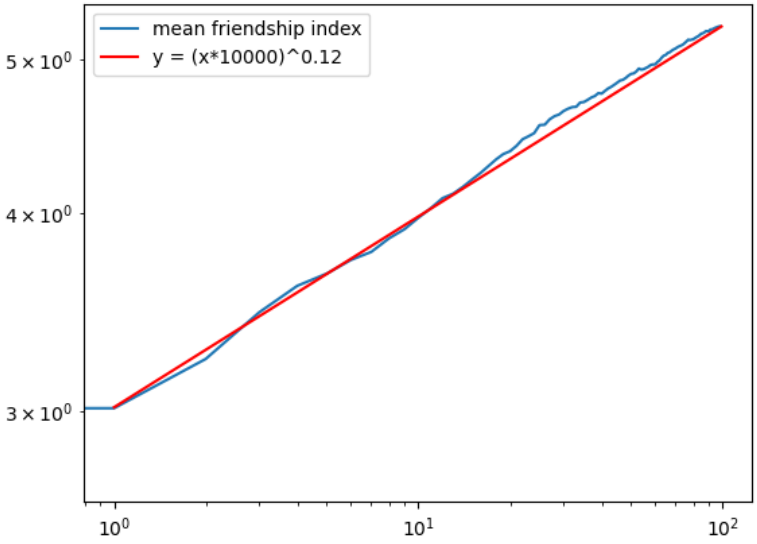

Рисунок 37 – Динамика среднего индекса дружбы в модели Барабаши— Альберт при $m = 3$

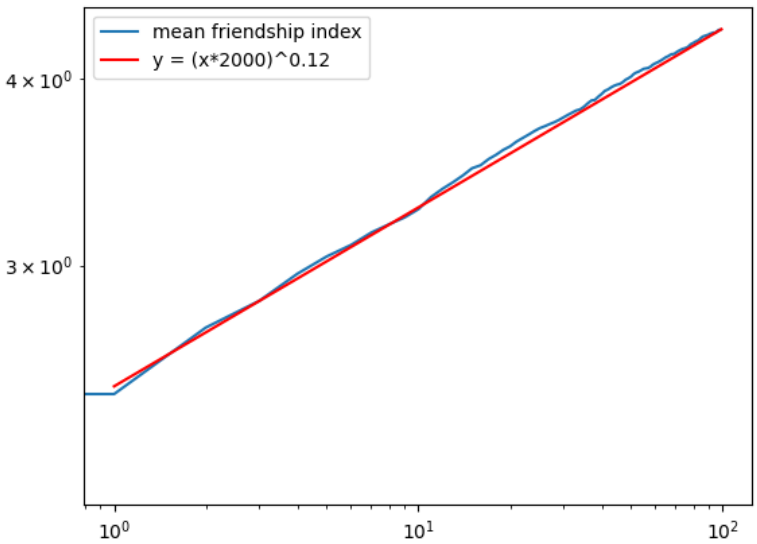

Рисунок 38 – Динамика среднего индекса дружбы в модели Барабаши— Альберт при $m = 5$

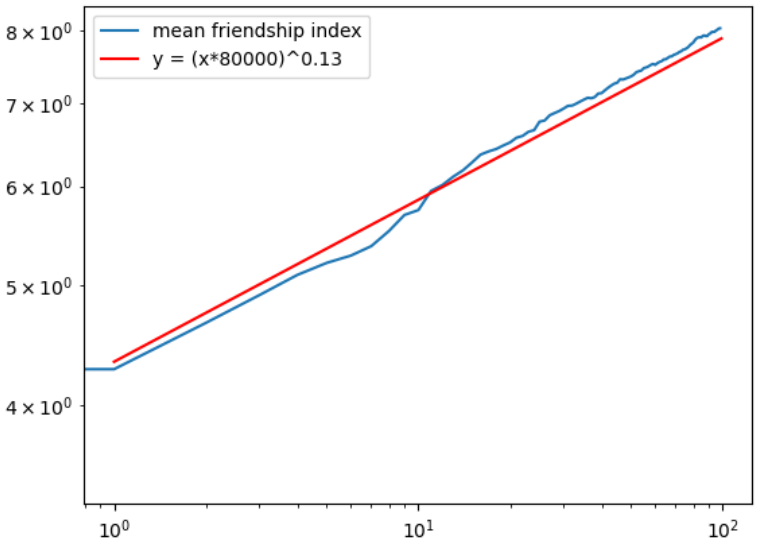

Рисунок 39 – Динамика среднего индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением степеней новых вершин при $\lambda = 4$

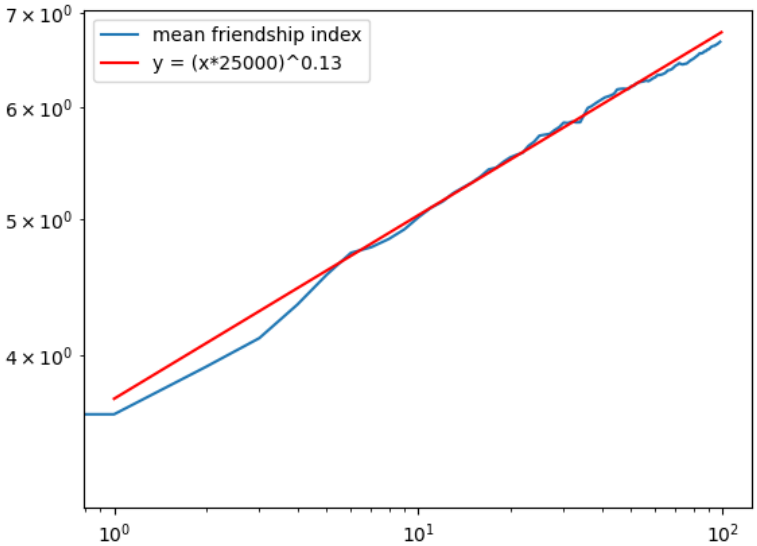

Рисунок 40 – Динамика среднего индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением степеней новых вершин при $\lambda = 5$

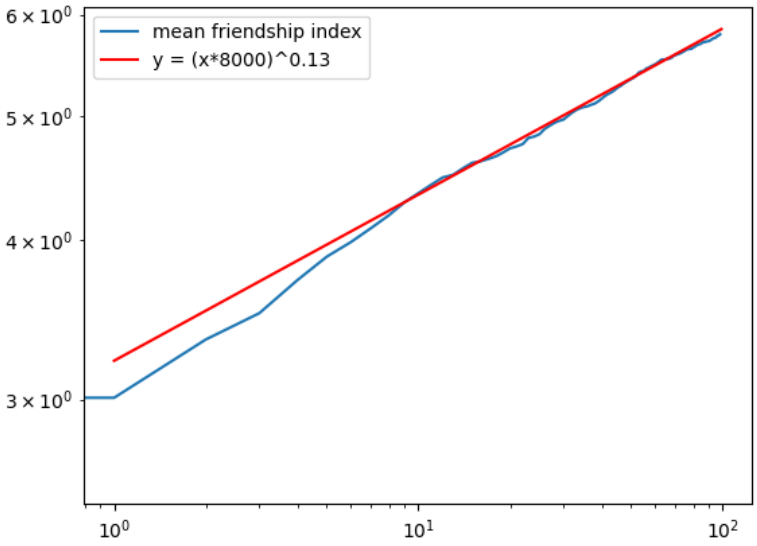

Рисунок 41 – Динамика среднего индекса дружбы в модели Барабаши— Альберт с пуассоновским распределением степеней новых вершин при $\lambda = 6$

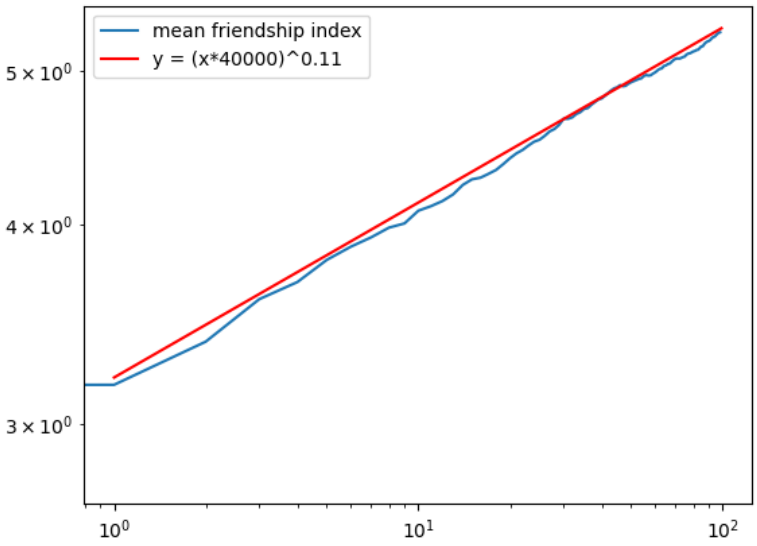

Рисунок 42 – Динамика среднего индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.25$

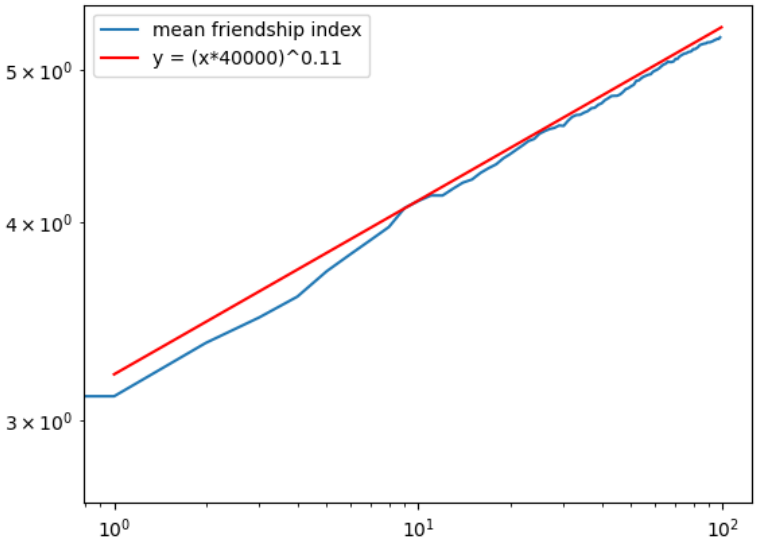

Рисунок 43 – Динамика среднего индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.5$

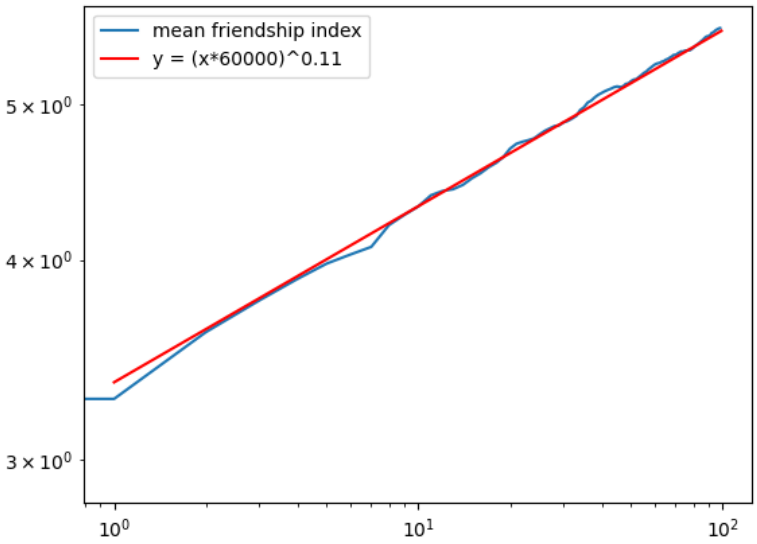

Рисунок 44 – Динамика среднего индекса дружбы в модели триадного замыкания при $m = 3$ и $p = 0.75$

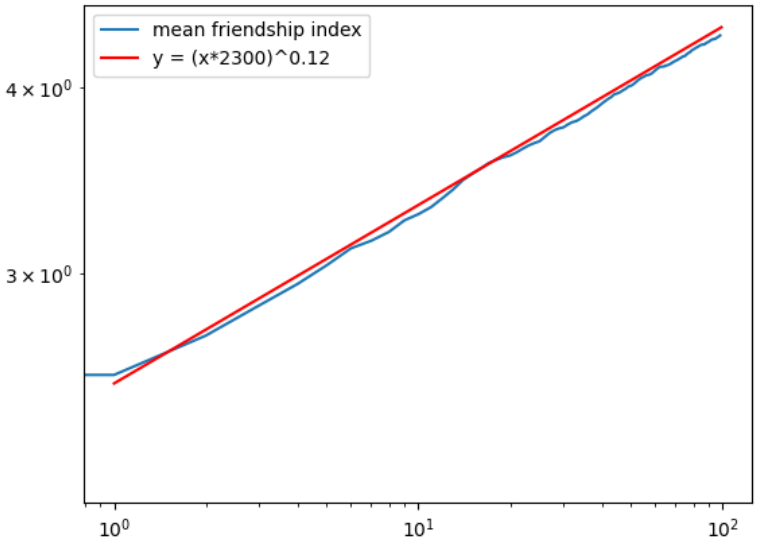

Рисунок 45 – Динамика среднего индекса дружбы в модели триадного замыкания при $m = 5$ и $p = 0.25$

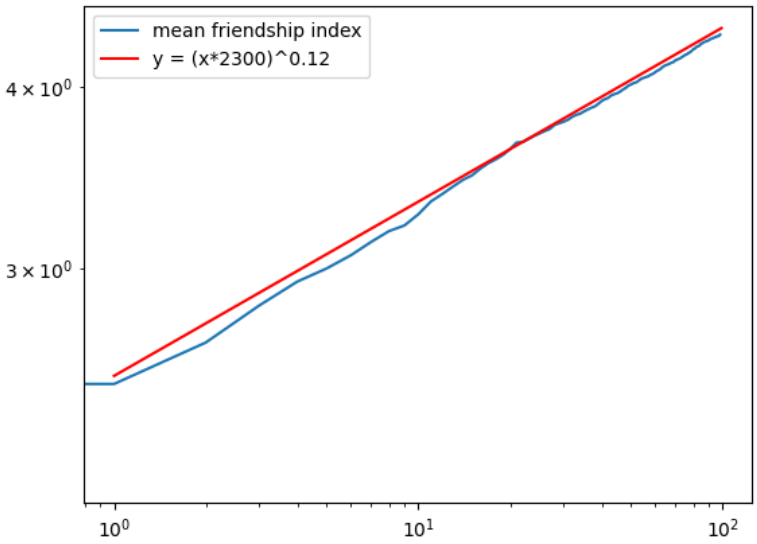

Рисунок 46 – Динамика среднего индекса дружбы в модели триадного замыкания при $m = 5$ и $p = 0.5$

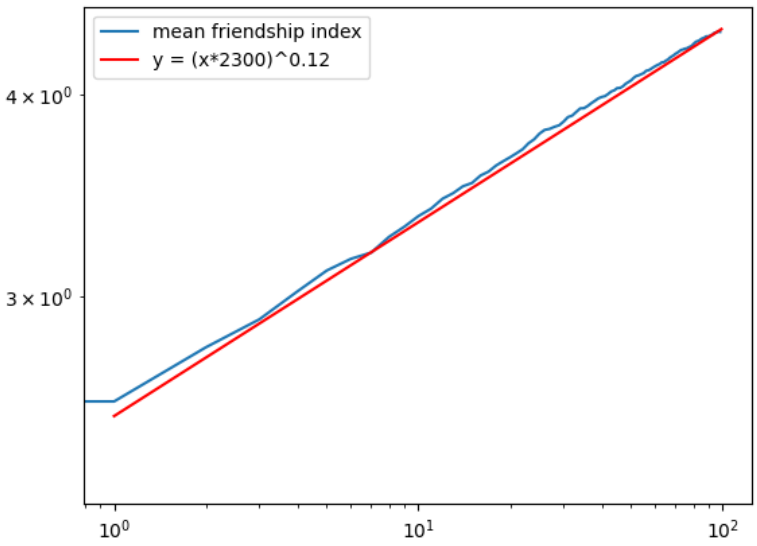

Рисунок 47 – Динамика среднего индекса дружбы в модели триадного замыкания при $m = 5$ и $p = 0.75$

ПРИЛОЖЕНИЕ Г

Текст скрипта для отображения распределения индекса дружбы

```
import igraph
import numpy as np
import matplotlib.pyplot as plt
import os
import multiprocessing
import math
def static(name):
    file = open('\\source\\repos\\CSW\\real_graphs\\' + name + '.txt',
    ↪  'r')
    res = []
    degrees = {}
    while True:
        line = file.readline()
        if not line:
            break
        edge = [int(j) for j in line.split(" ")]
        if not edge[0] in degrees:
            degrees[edge[0]] = 0
        degrees[edge[0]] += 1
        if not edge[1] in degrees:
            degrees[edge[1]] = 0
        degrees[edge[1]] += 1
    file.seek(0)
    sums = {}
    while True:
        line = file.readline()
        if not line:
            break
        edge = [int(j) for j in line.split(" ")]
        if not edge[0] in sums:
            sums[edge[0]] = 0
        sums[edge[0]] += degrees[edge[1]]
        if not edge[1] in sums:
            sums[edge[1]] = 0
        sums[edge[1]] += degrees[edge[0]]
    for i in degrees.keys():
        res.append(sums[i] / degrees[i] / degrees[i])
    file.close()
```

```
38      bincnt = 100000
39      ans = np.histogram(res, bincnt)
40      x = ans[1]
41      y = ans[0]
42      x = np.resize(x, x.size - 1)
43      plt.bar(x[0:int(bincnt / 100 * 2)], y[0:int(bincnt / 100 * 2)])
44      plt.show()
45      plt.clf()
46  if __name__ == "__main__":
47      static('sx-askubuntu')
```

ПРИЛОЖЕНИЕ Д

Текст скрипта для отображения динамики индекса дружбы

```
import igraph
import numpy as np
import matplotlib.pyplot as plt
import os
import multiprocessing
import math
import datetime
def dynamics(name):
    file = open('\\source\\repos\\CSW\\real_graphs\\' + name, 'r')
    edge_list = []
    while True:
        line = file.readline()
        edge_list.append(line.split("\t"))
        if not line:
            break
    edge_list.pop()
    edge_list = [[i[0],
                  i[1],
                  datetime.datetime.strptime(i[3], '%Y-%m-%d
                   ↪ %H:%M:%S').timestamp()]
                 for i in edge_list]
    file.close()
    edge_list.sort(key = lambda i: i[2])
    step = 1000
    base = edge_list[0][2]
    res = []
    degrees = {}
    neibours_degrees = {}
    neibours = {}
    cnt = 0
    for i in edge_list:
        cnt += 1
        if not i[0] in degrees:
            degrees[i[0]] = 0
            neibours[i[0]] = set()
            neibours_degrees[i[0]] = 0
        if not i[1] in degrees:
            degrees[i[1]] = 0
```

```
38              neibours[i[1]] = set()
39              neibours_degrees[i[1]] = 0
40          if i[0] == i[1]:
41              degrees[i[0]] += 1
42              neibours_degrees[i[0]] += degrees[i[0]]
43              for j in neibours[i[0]]:
44                  neibours_degrees[j] += 1
45              neibours[i[0]].add(i[0])
46          else:
47              degrees[i[0]] += 1
48              degrees[i[1]] += 1
49              if not i[1] in neibours[i[0]] and not i[0] in neibours[i[1]]:
50                  neibours_degrees[i[0]] += degrees[i[1]]
51                  neibours_degrees[i[1]] += degrees[i[0]]
52              for j in neibours[i[0]]:
53                  neibours_degrees[j] += 1
54              for j in neibours[i[1]]:
55                  neibours_degrees[j] += 1
56              neibours[i[0]].add(i[1])
57              neibours[i[1]].add(i[0])
58          if (i[2] - base) >= step:
59              base = step * math.ceil(i[2] / step)
60              ans = []
61              for j in degrees.keys():
62                  ans.append(neibours_degrees[j] / degrees[j] / degrees[j])
63              res.append(np.mean(ans))
64          # if cnt % step == 0:
65          #     ans = []
66          #     for j in degrees.keys():
67          #         ans.append(neibours_degrees[j] / degrees[j] / degrees[j])
68          #     res.append(np.mean(ans))
69      plt.plot(res)
70      plt.savefig("\\source\\repos\\CSW\\diploma_results\\" + name +
        ↪ "_iterational_dynamics.jpg")
71      plt.clf()
72  def run_thread(namegroup, i, res):
73      ans = []
74      for j in namegroup:
75          ans.append(dynamics(j))
76      res[i] = ans
77  def run(names):
```

```
78      procs = []
79      manager = multiprocessing.Manager()
80      res = manager.dict()
81      for (i, namegroup) in enumerate(names):
82          p = multiprocessing.Process(target=run_thread, args=(namegroup, i,
            ↪ res))
83          procs.append(p)
84          p.start()
85      for proc in procs:
86          proc.join()
87      ans = []
88      for i in res.values():
89          ans += i
90      return ans
91  if __name__ == "__main__":
92      # run([['askubuntu', 'askubuntu-a2q'], ['askubuntu-c2a',
        ↪ 'askubuntu-c2q'], ['mathoverflow', 'mathoverflow-a2q'],
93      #       ['mathoverflow-c2a', 'mathoverflow-c2q'], ['superuser',
        ↪ 'superuser-a2q'], ['superuser-c2a', 'superuser-c2q']])
94      dynamics('soc-redditHyperlinks-body.tsv')
```

ПРИЛОЖЕНИЕ Е

Текст скрипта для отображения динамики индекса дружбы в графах представленных в нескольких файлах

```
import igraph
import numpy as np
import matplotlib.pyplot as plt
import os
import multiprocessing
import math
import datetime
def dynamics(name):
    file = open('\\source\\repos\\CSW\\real_graphs\\' + name + '.txt',
    ↪  'r')
    time_file = open('\\source\\repos\\CSW\\real_graphs\\' + name +
    ↪  '-dates.txt', 'r')
    times = {}
    while True:
        line = time_file.readline()
        if not line:
            break
        line = line.split("\t")
        line = [int(line[0]), int(line[1].replace('-', ''))]
        times[line[0]] = line[1]
    edge_list = []
    while True:
        line = file.readline()
        edge_list.append(line.split("\t"))
        if not line:
            break
    edge_list.pop()
    edge_list = [[int(j) for j in i] for i in edge_list]
    file.close()
    time_file.close()
    edge_list.sort(key = lambda i: times[i[0]] if i[0] in times else
    ↪  100000000)
    step = 1000
    # base = edge_list[0][2]
    res = []
    degrees = {}
    neibours_degrees = {}
```

```
    neibours = {}
    cnt = 0
    for i in edge_list:
        cnt += 1
        if not i[0] in degrees:
            degrees[i[0]] = 0
            neibours[i[0]] = set()
            neibours_degrees[i[0]] = 0
        if not i[1] in degrees:
            degrees[i[1]] = 0
            neibours[i[1]] = set()
            neibours_degrees[i[1]] = 0
        if i[0] == i[1]:
            degrees[i[0]] += 1
            neibours_degrees[i[0]] += degrees[i[0]]
            for j in neibours[i[0]]:
                neibours_degrees[j] += 1
            neibours[i[0]].add(i[0])
        else:
            degrees[i[0]] += 1
            degrees[i[1]] += 1
            if not i[1] in neibours[i[0]] and not i[0] in neibours[i[1]]:
                neibours_degrees[i[0]] += degrees[i[1]]
                neibours_degrees[i[1]] += degrees[i[0]]
            for j in neibours[i[0]]:
                neibours_degrees[j] += 1
            for j in neibours[i[1]]:
                neibours_degrees[j] += 1
            neibours[i[0]].add(i[1])
            neibours[i[1]].add(i[0])
        # if (i[2] - base) >= step:
        #     base = step * math.ceil(i[2] / step)
        #     ans = []
        #     for j in degrees.keys():
        #         ans.append(neibours_degrees[j] / degrees[j] / degrees[j])
        #     res.append(np.mean(ans))
        if cnt % step == 0:
            ans = []
            for j in degrees.keys():
                ans.append(neibours_degrees[j] / degrees[j] / degrees[j])
            res.append(np.mean(ans))
```

```
76      plt.plot(res)
77      plt.savefig("\\source\\repos\\CSW\\diploma_results\\" + name +
        ↪  "_iterational_dynamics.jpg")
78      plt.clf()
79  if __name__ == "__main__":
80      # run([['askubuntu', 'askubuntu-a2q'], ['askubuntu-c2a',
        ↪  'askubuntu-c2q'], ['mathoverflow', 'mathoverflow-a2q'],
81      #      ['mathoverflow-c2a', 'mathoverflow-c2q'], ['superuser',
        ↪  'superuser-a2q'], ['superuser-c2a', 'superuser-c2q']])
82      dynamics('cit-Patents')
```

ПРИЛОЖЕНИЕ Ж

Графики распределения и динамики индекса дружбы в реальных сетях

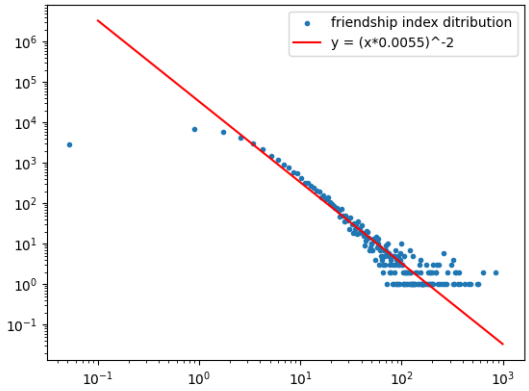

Рисунок 48 – Распределение индекса дружбы в сети цитирования статей в сфере феноменологии физики высоких энергий в логарифмической шкале

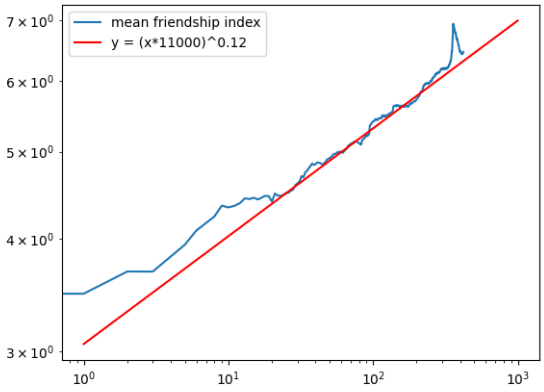

Рисунок 49 – Динамика индекса дружбы в сети цитирования статей в сфере феноменологии физики высоких энергий в логарифмической шкале

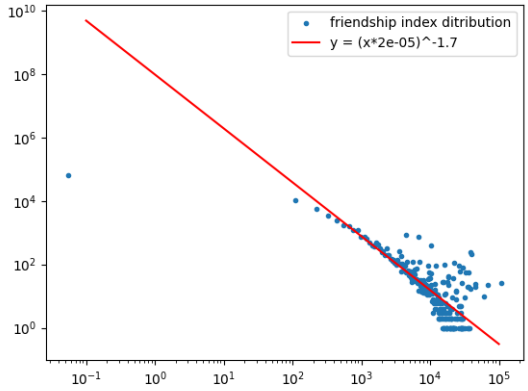

Рисунок 50 – Распределение индекса дружбы в
социальной сети Google+
в логарифмической шкале

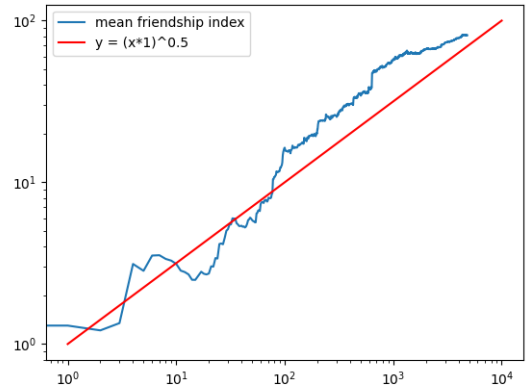

Рисунок 51 – Динамика индекса дружбы в
сети сообщений студентов Калифорнийского университета в Ирвайне
в логарифмической шкале

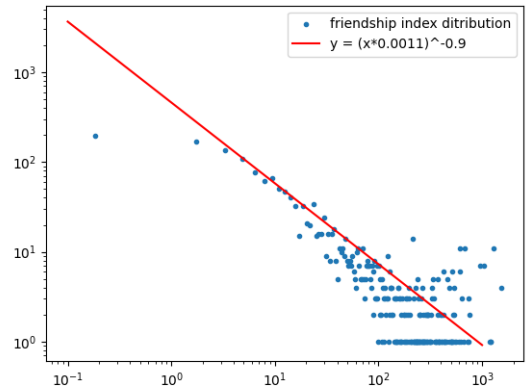

Рисунок 52 – Распределение индекса дружбы в
сети сообщений студентов Калифорнийского университета в Ирвайне
в логарифмической шкале