MINISTRY OF EDUCATION AND SCIENCE OF RUSSIA
Federal State Budgetary Educational Institution of Higher Education
**«SARATOV NATIONAL RESEARCH UNIVERSITY
STATE UNIVERSITY
NAMED AFTER N. G. CHERNYSHEVSKY»**
Chair of mathematical cybernetics and computer science

STUDY OF THE FRIENDSHIP INDEX OF NODES IN GROWING NETWORKS BASED ON MODELS WITH PREFERENTIAL ATTACHMENT

BACHELORS THESIS

student 4 'th year 411 group
specialization 02.03.02 — Fundamental computer science and information technologies
department CSIT
Kozyrev Yuri Dmitrievich

Academic advisor
head of department,PhD, associate professor _____ S. V. Mironov

Head of the department
Ph. D., associate professor _____ S. V. Mironov

Saratov 2023

TABLE OF CONTENTS

INTRODUCTION 4

1 Theoretical information 6

1.1 The Erdos—Renyi model 6

1.2 Barabasi—Albert model 6

1.3 Bollobash—Riordan model 7

1.4 The Chung-Lu model 8

1.5 Triad closure model 8

1.6 Friendship Index 8

2 Implementation of the models 11

2.1 Implementation of the standard Barabasi—Albert model 12

2.2 Implementation of the Barabasi—Albert model with a random number of edges added at each iteration 13

2.3 Implementation of the triad closure model 14

2.4 Parallelized implementation of the Barabasi—Albert model 15

2.5 Representation of the data for analysis 16

3 Graph analysis of real networks 19

3.1 Script for displaying the distribution of the friendship index 19

3.2 Script for displaying the dynamics of the friendship index 21

3.3 Script for displaying the dynamics of the friendship index in graphs presented in several files 24

4 Analysis 26

4.1 Analysis of the distribution and dynamics of the friendship index in the constructed graphs 26

4.2 Analysis of the distribution and dynamics of the friendship index in real networks 31

4.2.1 Analysis of the distribution of the friendship index in the LiveJournal social network 32

4.2.2 Analysis of the distribution of the friendship index in the Twitter social network 32

4.2.3 Analysis of the distribution and dynamics of the friendship index in the citation network of articles in the field of high energy physics phenomenology 34

4.2.4 Analysis of the distribution and dynamics of the friendship index in the Reddit forum network . . . 35
4.2.5 Analysis of the distribution and dynamics of the friendship index in the AskUbuntu Q&A system . . . 36
4.2.6 Analysis of the distribution and dynamics of the friendship index in the SuperUser Q&A system . . . 40
CONCLUSION . . . 44
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ . . . 45
Приложение A Text of the program . . . 47
Приложение B Friendship index distribution graphs in the constructed graphs . . . 53
Приложение C Graphs of the dynamics of the average friendship index in the constructed graphs . . . 58
Приложение D Script text for displaying the friendship index distribution . . . 64
Приложение E Script text for displaying the dynamics of the friendship index . . . 66
Приложение F Script text for displaying the dynamics of the friendship index in graphs presented in several files . . . 69
Приложение G Graphs of the distribution and dynamics of the friendship index in real networks . . . 72

INTRODUCTION

In everyday life, random graphs are often used to solve many problems. Random graphs have found practical application in all areas where complex networks simulating is required. There is a large number of random graph models that reflect various types of complex networks in various fields. Random graphs are used for modeling and analyzing biological and sociological systems, networks, and also for solving many NP class problems.

Random graphs were first defined by the Hungarian mathematicians P. Erdos and A. Renyi in the 1959 book «On Random Graphs» [1] and independently by the American mathematician, E. Hilbert, in his paper «Random graphs» [2].

Random graph is a general term for the probability distribution over graphs [3]. They can be described simply by a probability distribution or by a random process that creates these graphs.

Random graph theory is at the intersection of combinatorics, graph theory, and probability theory. It is based on the idea that the powerful tools of modern probability theory should contribute to a more accurate understanding of the nature of the graph, and are designed to help solve many combinatorics and graph-theory problems.

From a mathematical point of view, random graphs are necessary to answer the question about the properties of typical graphs. For this purpose are used the Erdos—Renyi, Barabashi—Albert models, the triad closure model, the Bianconi-Barabashi model, and others. All models are based on various properties of social networks.

This thesis considers some of the most widely used and well-studied models: the Barabasi-Albert model and the triad closure model.

One of the tools widely used for studying and analyzing random graph models, analyzing social phenomena and networks, communities and their interactions is the friendship index. It is used in sociology and defined as the ratio of the average degree of neighbors to the degree of the object itself.

The purpose of this study is to analyze the friendship index of networks constructed according to the Barabashi—Albert model. To achieve this goal, the following objectives must be accomplished.

— research Barabasi—Albert and triadic closure algorithms for building a random graph;
— implement the classic Barabasi—Albert algorithm, and its modification, in

which the initial degree of each new node is defined as a random variable given by the Poisson distribution;

— implement the triad closure algorithm.

— explore the possibility of multi thread implementation for building a random graph model and calculating the friendship index.

— conduct a series of experiments of building random graphs based on the implemented models.

— analyze the friendship index distribution in the constructed graphs.

1 Theoretical information

There are many different models for constructing random graphs. Let's review some of them.

1.1 The Erdos—Renyi model

The Erdos—Renyi model is one of the first random graph models. A graph constructed according to this model is a pair of the set of vertices $V = \{1, ..., n\}$ and the set of edges E, which consists of the edges of the complete graph K_n built on the set V chosen according to the Bernoulli scheme. Thus, a random graph $G = (V, E)$ is formed. Formally, we have the probability space

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

in which: n — is the number of vertices, p — is the probability of a new edge, F_n — is a sigma set,Ω_n — is the set of possible edges ($|\Omega_n\| = 2^n$), $P_{n,p}(G) = p^{|E|}(1-p)^{\binom{n}{2}-|E|}$ — probability measure. Thus, in the Erdos—Renyi model, each edge included in a random graph with probability p independently of other edges . The Erdos—Renyi model is currently the most studied model of random graphs [4].

1.2 Barabasi—Albert model

The Barabasi—Albert model is one of the first web graph models. A web graph is a directed multigraph vertices of which represent any specific structural units of the Internet, for example: pages, sites, hosts, owners, and so on. For definiteness, we assume that the web graph vertices represent sites. And edges connect vertices that have links between them.

In their model A.-L. Barabashi and R. Albert proposed the strategy of preferential attachment [5]. Its main idea is that the probability of connecting a particular vertex to a new vertex with an edge is proportional to the degree of this vertex. Hereafter, the degree of a vertex $v_i \in V$ of a graph $G = (V, E)$ is the number of vertices directly connected to a given vertex, i.e.

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

The Barabasi—Albert algorithm for building a random network is as follows.

1. Initially a complete graph of m vertices is created, where m — is a model

parameter.

2. At each iteration of network growth, one new vertex is added, which is connected by m edges to the existing ones according to principle of preferential attachment.

1.3 Bollobash—Riordan model

One of the most successful and frequently used models of preferential attachmenting is the Bollobash—Riordan model. There are two main and, in fact, identical modifications of this model. One gives a dynamic and the other a static description of the randomness [6].

In a dynamic modification, when adding an n-th vertex, n new edges are drawn, and the edges can be multi-edges, as well as loops. When creating a graph with a single vertex, a loop is drawn to that point [4]. Thus, the probability of an edge $(n, ß)$, $i \in [0, n-1]$ is $\frac{deg_i}{2n-1}$, where deg_i — is the degree of the vertex i.

A static model (LCD model) is based on an object called a linear chord diagram (LCD). To construct this object, you need to fix $2n$ points $1, ..., 2n$ on the abscissa axis, split them into pairs, and connect the elements of each pair with an arc lying in the upper half-plane. The number of different charts is

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

For each diagram, a graph with n vertices and n edges is constructed using the following algorithm:

— Go from left to right along the abscissa axis until you meet the right end of an arc, let the position of this point be i_k

— The sequence $i_{k-1} + 1$, i_k is declared as an adjacency list for the k-th vertex, $i_0 = 0$

— If $k < n$, k is incremented by 1, move to step (1).

When constructing an LCD model, one of the possible LCDs is randomly selected and the probability of each diagram is $\frac{1}{l_n}$, where l_n is the total number of diagrams. Graphs constructed using this model have the same properties as graphs constructed using the dynamic modification of the Bollobash—Riordan scheme.

1.4 The Chung-Lu model

Let's assume that we are given a finite set of vertices $V = v_1, ..., v_n$ and the degree of each vertex d_i, $i = \overline{1, n}$. The graph $G = (V, E)$ is generated as follows:

— we form a set L consisting of $i \cdot d$ copies of $i \cdot v$ for each i from 1 to n;

— setting random pairs from the L set;

— for vertices u and v from Vs, the number of edges in the graph G connecting them is equal to the number of matches between copies of u and v in L [7].

The graph generated in this way corresponds to the power model $P(a, b)$, which describes graphs for which:

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

1.5 Triad closure model

In addition to the preferential attachment strategy, the triad closure model uses a triad formation strategy. Triad closure is a property of social systems that consists in the fact that if there is a relationship between the vertices (A, B) and (A, C) in some social network, then the probability of forming a triad from these three vertices is high, i.e., the probability of connection (B, C) is high [8]. In the triad closure model, network growth occurs as follows.

1. Initially a complete graph of m vertices is created, where m — is a model parameter.
2. At each iteration of networks growth, one new vertex is added, which is connected by m edges to the existing ones according to the following rules:
   — in accordance with the principle of preferential attachmenting, the vertex to which the first edge is drawn is selected.
   — with probability p, where p is a parameter of the model, the triad formation strategy with an random neighbor of the vertex connected by the first edge is chosen, or, with probability $(1 - p)$, the strategy of preferential attachment to an random vertex of the graph.

1.6 Friendship Index

It hasn't been long since the advent of social networks — Facebook, Vkontakte, LiveJournal, Instagram, LinkedIn, MySpace, etc. - but they are already firmly embedded in the daily lives of many people.

Surveys show that 76% of Internet users in Russia (according to PRT agency data as of January 2014) and approximately 73% of residents of the United States are active users of social networks, and the figure is growing [9].

In today's society, social networks are becoming a huge database of information that scientists and employers are increasingly using to solve specific tasks, whether it's scientific research or evaluating a candidate for a particular position.

Scientific interest in the study of users of social networks is growing rapidly. At the moment, a large amount of empirical material has been accumulated regarding the characteristics of users of social networks, which requires systematization and comprehension.

In social networks, you can often find a phenomenon called the friendship paradox: on average, any person's friends have more friends than the person themselfs. It was discovered in 1991 by Scott Feld, a sociologist at the State University of New York.

To study the friendship paradox, we should introduce several notations. At time t for vertex v_i in the graph $G(t) = (V(t), E(t))$ the sum of the degrees of all neighbors v_i is:

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

average degree of neighbor0 vertices v_i:

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

and the friendship index $\beta_i(t)$ is defined as the ratio of the average degree of neighbors v_i to the degree of v_i itself:

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

Thus, if the average degree of neighbors is greater than the degree of v_i and the friendship paradox holds, then $\beta_i(t) > 1$ [10].

For example, the social networks Facebook and Github confirm the friendship paradox, as shown in Figure 1 [11]. Here, the Oy axis plots the number of network nodes for which the friendship index β_i falls within the range plotted on the Oxaxis. As you can see, a significant majority of graph vertices have the frienship index value

greater than one.

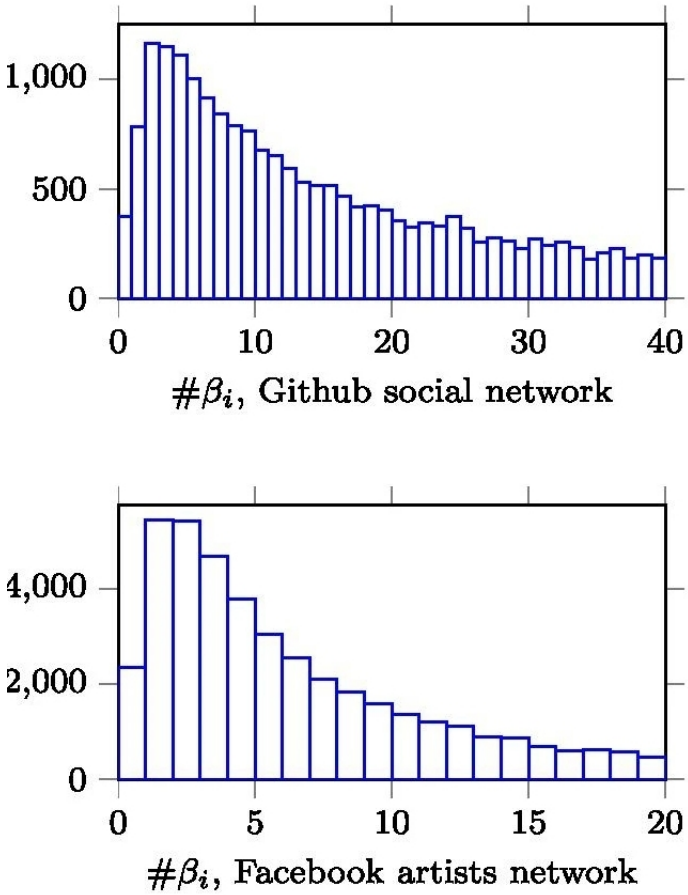

Figure 1 – Friendship index distributions in
phone call network and Amazon delivery network

2 Implementation of the models

In the course of the thesis, the following models were implemented: the standard Barabasi—Albert model, the Barabasi—Albert model with a Poisson distribution of initial degrees, and the triad closure model. All calculations were performed on a computer with an Intel core i5-8265U processor and 16 GB of RAM. The models were implemented in Python 3.9.1. The following libraries were used in the implementation: json, multiprocessing, random, and numpy. random.

All three implemented models are growing random graph models using the preferential attachment mechanism. In the implemented modifications, one vertex and a certain number m of edges are added at each step. The m parameter and the process of selecting neighbors of the new vertex depend on the model used. The triad closure model has an additional parameter p.

During the experiments we iterate over different models and different parameter values. In each experiment, a graph of 100,000 vertices is constructed. Since the resulting graphs are random, each graph is constructed ten times during the experiment, after which a histogram over the average ranges of values of the friendship index and the dynamics of the average friendship index of the graph are plotted.

In the implementation of models, the graph is represented by a pair of arrays `degrees` and `neibours`, which store the degree and the list of neighboring vertex numbers for each vertex of the graph, respectively. Each model is implemented in the form of function

`model_name(n, args, funcs, dts)`,

where the parameter `n` sets the size of the final graph, the array `args` contains the model arguments, `funcs` is an array of metrics to be applied to the graph, and `dts` — frequency of applying metrics.

Metrics are also represented by functions of the form

`metric_name(degrees, neibours)`,

in which `degrees` is an array of degrees of graph vertices, and `neibours` is an array of arrays of neighboring vertices. In the course of our work, we used four metrics:

- `s(degrees, neibours)` — returns an array of sums of neighbor degrees for graph vertices.
- `alfa(degrees, neibours)` — returns an array of average degrees of neighbors for graph vertices.
- `beta(degrees, neibours)` — returns an array of friendship indices for

graph vertices.

— `mean_beta(degrees, neibours)` — returns the average friendship index of the entire graph

2.1 Implementation of the standard Barabasi—Albert model

The implementation of the standard Barabash model—Albert is as follows.

The model is implemented as a function

`my_bag(n, args, funcs, dts)`,

as described above.

The null element of the `args` array is taken as the$m$ parameter of the model.

The implementation uses the array `ans`, which will store the values of the metrics of the constructed graph. Initially it is filled with empty arrays by the number of metrics:

`ans = [[] for _ in range(len(dts))]`.

A complete graph of $m + 1$ vertices is created. To do this, the array `degrees` is filled with `m + 1` values `m` and the array `neibours` is formed from `m + 1` lists, each of which contains numbers of all currently existing vertices of the graph, except for the current one.

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

Next, two helper arrays are created: array `nodes` containing a list of the numbers of all vertices currently belonged to the graph, and `used` — an array that contains the vertices of the graph that are already attached to the new node.

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

Then in a loop over $n - (m + 1)$ indexes new vertices are added to the graph. A list `conections` is formed to store the neighbors of the new node. The `random.choices` function selects m unique vertices that are written to `conections`. The uniqueness of vertices is determined by the list `used`. The principle of preferential attachment is maintained by passing the degrees of vertices as an array of weights to `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
```

In the next loop through the array elements `connections` in the `used` array, all `True` are replaced with `False`. In addition, the values of the `neibours` list are supplemented with the links formed at this step, and the `degrees` array is changed accordingly.

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

The last step in appending a new vertex in this algorithm is to collect graph metrics in the current state. For each function in the list `funcs`, the algorithm determines whether to measure this metric at the current step, based on the divisibility of `i` by the corresponding value `dts`. If the current metric is requested by the user at this step, the result of its calculation is added to the corresponding cell of the `ans` array.

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

The results of the entire `my_bag` function are placed in `ans`.

2.2 Implementation of the Barabasi—Albert model with a random number of edges added at each iteration

The implementation of the Barabasi—Albert model with a Poisson distribution of degrees of new vertices repeats the standard Barabasi—Albert model in many aspects. The first distinction lies in the contents of the parameter list `args`: instead of the m parameter, this model accepts the λ parameter for the Poisson distribution.

This parameter is extracted to the variable `m` and used to generate the value `m0` — the number of vertices of the initial complete graph.

```
    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)]
```

The initial degrees of each new vertex are also determined according to the Poisson distribution. Since the degree of a new vertex can be neither greater than the number of nodes in the graph, nor less than one, then the number of neighbors of the new vertex must be subjected to appropriate restrictions.

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

2.3 Implementation of the triad closure model

The triad closure model is also implemented as a modification of the standard Barabash model—Albert. Unlike other implemented models, the argument list contains two parameters: the fixed degree of new vertices m and the probability of triad formation p. These parameters are extracted into variables with corresponding names.

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

Certain changes are also applied to the process of attaching new vertices: in this variation, there are two ways to select a candidate `choosen` for the role of a new neighbor of the current node. The choice based on the principle of preferential attachment is made either with probability $1 - p$ from the set of all vertices of the graph, or from the number of neighbors of the vertex connected first. The first node is also attached according to the first option.

```
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 Parallelized implementation of the Barabasi—Albert model

To speed up the program, a multithreaded version of the algorithm was implemented using the multiprocessing and numpy libraries: the `run` function creates threads, distributes tasks and collects results, and the `run_thread` function performs experiments in each thread separately and returns their results.

The function `run` takes seven arguments: the number of random graphs to construct, `N` ; the number of threads `thread` , the algorithm for constructing a random graph `model` ; the size of the graph `n` ; array of model arguments `args` ; list of metrics `funcs` and the frequency of their measurment `dts` . First, an object of the `multiprocessing.Manager` class is created, it will manage created threads. A reference to the dictionary with calculation results - `manager.dict()` is stored in the variable `res` . Then in a loop for each of `thread` cores `multiprocessing.Process` is created that will execute the corresponding thread. All processes are added to the list of running threads and started using the `p.start` method. Next, we wait for all threads to execute, collect all the results in the `ans` list, and return it as the result of the function.

At the first step in `run_thread` initializes an empty list `ans` . Then the specified number of random graphs is constructed, the specified metric is calculated for each of them, and the result is written to the `ans` list.

```
1  def run_thread(N, i, res, model, n, args, funcs, dts):
2      ans = []
3      for j in range(N):
4          ans.append(model(n, args, funcs, dts))
5      res[i] = ans
6  def run(N, treads, model, n, args, funcs, dts):
7      procs = []
8      manager = multiprocessing.Manager()
9      res = manager.dict()
10     for i in range(treads):
11         curn = min(N, math.ceil(N / (treads - i)))
12         p = multiprocessing.Process(target=run_thread, args=(curn, i, res,
           ↪  model, n, args, funcs, dts))
13         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 Representation of the data for analysis

This implementation contains a code snippet responsible for displaying data about the constructed graph for further analysis. The matplotlib, matplotlib.pyplot [12] and numpy libraries are used for this purposes.

During the experiments, the average friendship index of the graph vertices was measured for each model every 100 iterations, and after the graph was constructed, the friendship index for each vertex was calculated in order to display the distribution of its values.

First, the data of the final friendship index is processed and presented as a three-dimensional array: for each constructed graph, for each measurment cycle, and for each vertex, there are values of the friendship index. Since data is sampled only once, the measurement axis can be removed, leaving only elements at null position. Then, a histogram is constructed for each graph, and the values of the histogram columns are averaged over the set of graphs. The average histogram is used to create a plot and the result is saved.

```
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()
```

Similarly, information about the dynamics of the average friendship index is transformed. It is stored in the array `ans` under the index `1` and placed in the list `mean_beta_data`. Graph axis and tact axis are swaped. Then the data is averaged over the graphs, and a plot is constructed and saved from them.

```
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()
```

This sequence is repeated for each model of the graph, for each combination of parameters.

```
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()
```

In addition, you need to build and save one copy of each graph to a text document for further study. The json format was chosen as the text storage format for the graph, due to its structure, human readability, and ease of reading and writing. In the json file, the graph is represented as a list of dictionaries. Each element of the list represents a vertex of the graph. A vertex is represented by three dictionary entries:

— `index` — index of a vertex in the graph,

— `degree` — vertex degree,

— `neibours` — list of node neighbors.

Since graph construction algorithms do not return the entire graph, but only the values of the specified metrics, graph data must be extracted using the pseudometrics `d`, `neibours` and `index`. They return an array of degrees, a matrix of neighbors, and a list indexes of the graph, respectively. These values are saved to a file.

```
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()
```

The result is a json file of the following format for each graph:

```
[{"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]}]
```

For the full program code, see A.

3 Graph analysis of real networks

During the course of the thesis, several scripts were written for reading from a text file and analyzing graphs of real networks:

— script for efficiently reading large networks (up to several million vertices and tens of millions edges) and calculating the distribution of the friendship index in them.

— program for reading graphs with information about the time when an edge appeared and tracking the dynamics of changes in the average friendship index.

— script that reads two files: a file with edges and a file containing data about the time when a vertex was added, and also displays the dynamics of the average friendship index of the graph.

3.1 Script for displaying the distribution of the friendship index

Since the script for displaying the distribution of the friendship index must process large graphs with a limited amount of memory, it becomes impossible to save all the information about the graph. However, to calculate the friendship index, we need data about the neighbors of each vertex, because the graph is constantly changing during execution of the algorithm and it becomes impossible to calculate the friendship index when adding a vertex or updating it dynamically. This problem was solved by using two passes through the read file: one for counting the finite degrees of all vertices, and the second for transmitting this data to neighbors, since this algorithm does not imply storing data about the edges of the graph.

First, the script opens the necessary file for reading, creates a list for the values of the friendship index — `res` and a dictionary `degrees` for storing vertex degrees.

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

Then a read loop is started in which a line from the file is written to the variable `line`. If the line is equal to `null`, the loop is interrupted. Splitting the string `line` by a space character creates an edge `edge`. For each of the two ends of an edge, 1 is added to the number of neighbors if there is no entry in `degrees` for the vertex yet then it is created and initialized 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
```

The cursor is then moved to the beginning of the file. A dictionary `sum` is created for the sum of the neighbors of each vertex. And the second reading cycle starts, similar to the first one, with the only difference being that it counts the sums of neighbors, and not the degrees of vertices.

```
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]]
```

At the end of the program, the friendship index is determined for each vertex, the file is closed, and then a histogram of the distribution of the friendship index in the specified graph is constructed and saved.

```
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()
```

The full script code is provided in the Appendix D.

3.2 Script for displaying the dynamics of the friendship index

The script for displaying the dynamics of the friendship index must be able to repeatedly calculate the friendship index on the graph being read, so it saves not only lists of vertex neighbors, but also sums of neighbor degrees.

The first steps are similar to the previous algorithm: the file is opened for reading and a reading cycle is started. However, data is written to the list `edge_list` almost without processing — the string is simply split by the character `\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()
```

Then the elements `edge_list` are converted to triples of numbers containing the number of the first vertex of the edge, the number of the second vertex, and the unix time when the face was added. All data from the file is read and it closes.

```
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()
```

The resulting array is sorted using the `list.sort()` method in ascending timestamp

order. The following variables are initialized:

— time measurement step `step`,

— time reference point `base`,

— list for storing results `res`,

— dictionaries that will contain the processed graph:

`degrees` — dictionary of graph degrees;

`neibours_degrees` — contains sums of degrees of neighbors.

`neibours` — sets of the vertex neighbors themselves.

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

The next step in the loop through the sorted array is to check each vertex if it is present in the already described part of the graph. If there is no vertex yet, then entries in all three dictionaries are initialized for it.

```
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 the vertex is a loop, then the degree of one of the ends increases by 1, the degree of the vertex itself is added to the sum of the degrees of the neighbors of this vertex, and the sum of the degrees of the neighbors of all adjacent vertices increases by 1. Finally, an attempt is made to add a new vertex to the set of neighbors (obviously, the attempt will fail if the set already contains the node).

```
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])
```

Othervise, the metioned steps are performed twice (for each end of the face).

```
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])
```

When more than `step` seccond have passed since `base`, `base` is updated to the lowest value that exceeds the current time. The list `ans` records the index values of all graph vertices. These values are averaged using `numpy.mean()` and the average is added to `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))
```

The `res` values are used to create and save a graph.

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

Since the script was not applied to very large graphs, there are still enough resources left during its execution to simultaneously process multiple networks in multithread mode.

```
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']])
```

The full script code is provided in the Appendix E.

3.3 Script for displaying the dynamics of the friendship index in graphs presented in several files

This program is a modification of the previous one. The main difference is that in this case, the time of adding vertices is read from a separate file in the dictionary `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]
```

In addition, the degree array `edge_list` is sorted using a lambda function that maps the vertex of the edge origin to the time it was added.

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

The full script code is provided in the Appendix F.

4 Analysis

In order to verify that the implemented models can act as models of social processes in the real world, it is necessary to show that the generated graphs have some properties of real networks. The friendship index was chosen as a characteristic for analysis

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

The thesis considers the distribution of the friendship index and the dynamics of its average value.

4.1 Analysis of the distribution and dynamics of the friendship index in the constructed graphs

The graph shown in Fig. 2 shows the distribution of the friendship index in a graph constructed in accordance with standard Barabasi— Albert model with the parameter $m = 3$. From this diagram, we can conclude that for the most vertices of the graph, the value of the friendship index is over 1. Therefore, the friendship paradox holds for this model.

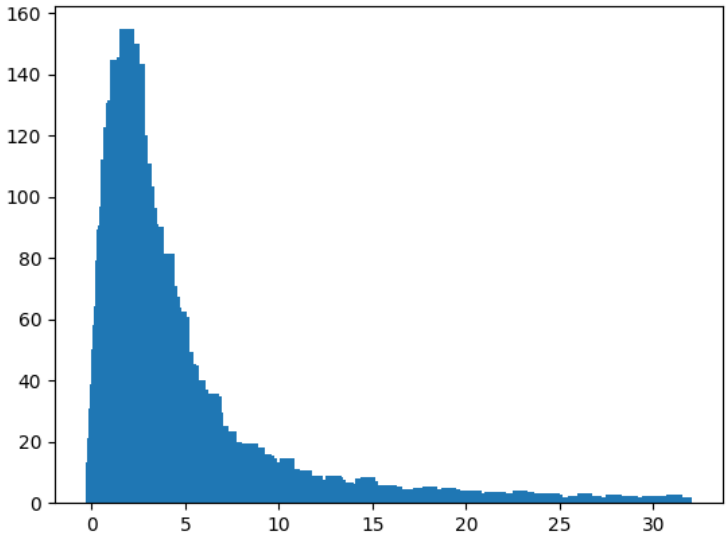

Figure 2 – Friendship index distribution in standard Barabasi— Albert model with $m = 3$

Fig. 3 shows a graph of the distribution of the friendship index in a graph

constructed using the modified Barabasi— Albert model with the parameter $m = 4$. Based on this graph, we can say that the friendship paradox is fulfilled in this model.

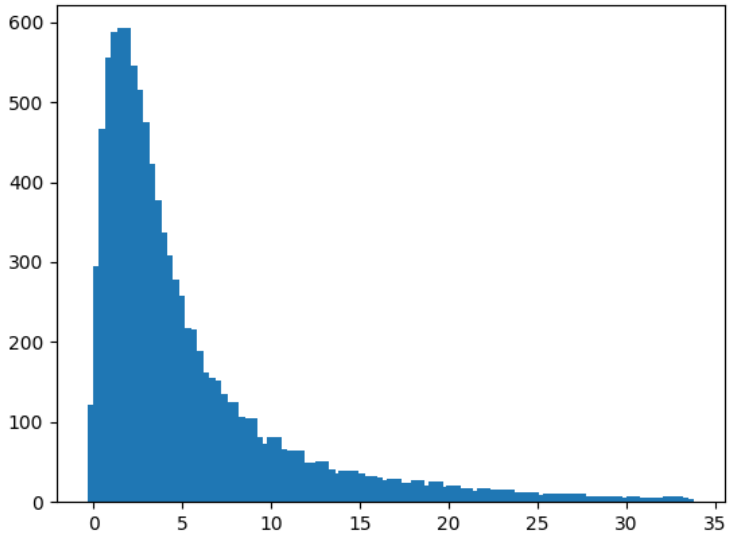

Figure 3 – Distribution of the friendship index in modified Barabasi— Albert model with $m = 3$

The chart in Fig. 4 shows the distribution of the friendship index in the graph, which is based on the triad closure model. This graph also confirms the presence of the friendship paradox in the model under consideration.

In Fig. 5, **??**, 7 output the same values, but on the logarithmic scale. In accordance with the graph, we can hypothesize that the distribution of the values of the friendship index occurs in accordance with a power function

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

Graphs of the friendship index distribution in these models with other parameters m and p are presented in Appendix B.

Figure 8 contains a diagragram in logarithmic scale of the dynamics of the average friendship index in the Barabasi— Albert model at $m = 3$ in comparison with the plot of a power function

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

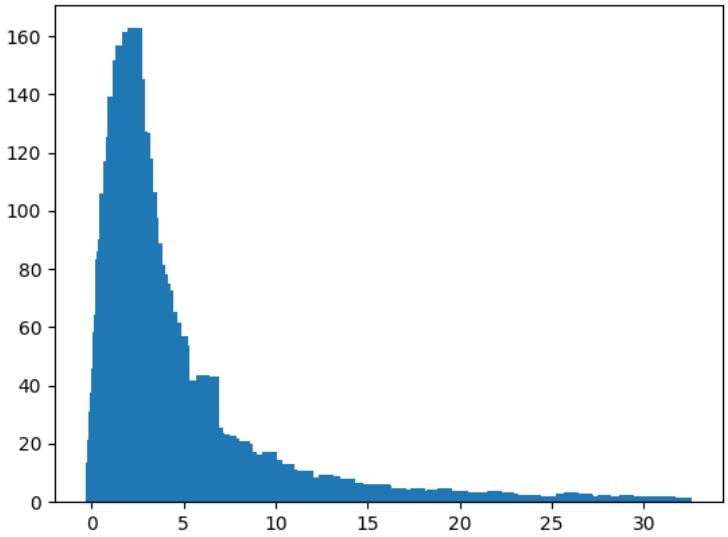

Figure 4 – Friendship index distribution in
triad closure model
with $m = 3$ and $p = 0.25$

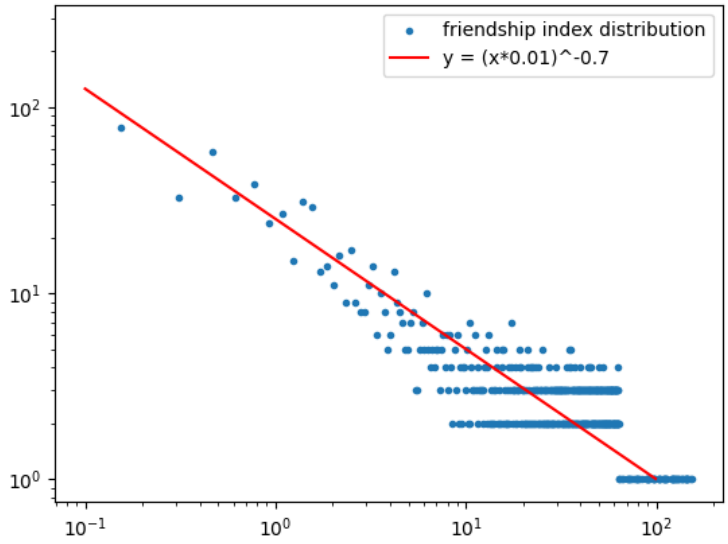

Figure 5 – Distribution of the friendship index in the
standard Barabasi— Albert model
with $m = 3$, on the logarithmic scale

Based on this graph, we can assume that the average value of the friendship index in the models under consideration grows in accordance with the power function.

The chart of the dynamics of the average friendship index in the Barabasi— Albert model with the Poisson distribution of the initial degrees of vertices with $m = 4$ on the logarithmic scale, is shown in Fig. **??**, coincides with the plot of the

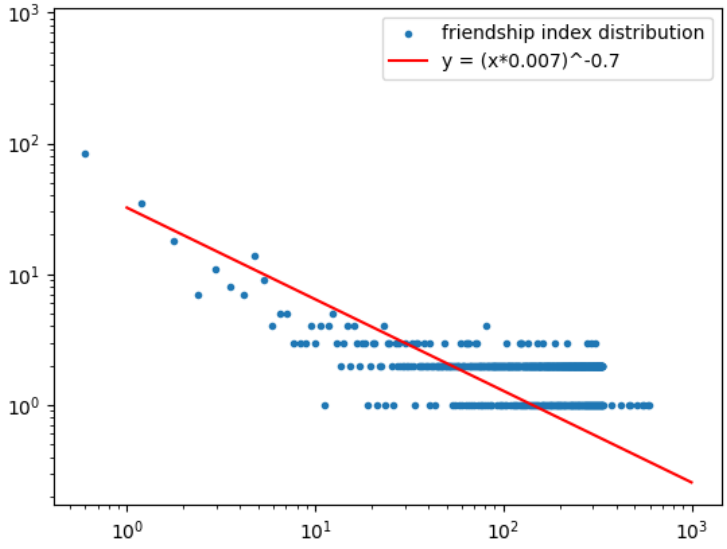

Figure 6 – Distribution of the friendship index in modified Barabasi— Albert model with $m = 4$, on the logarithmic scale

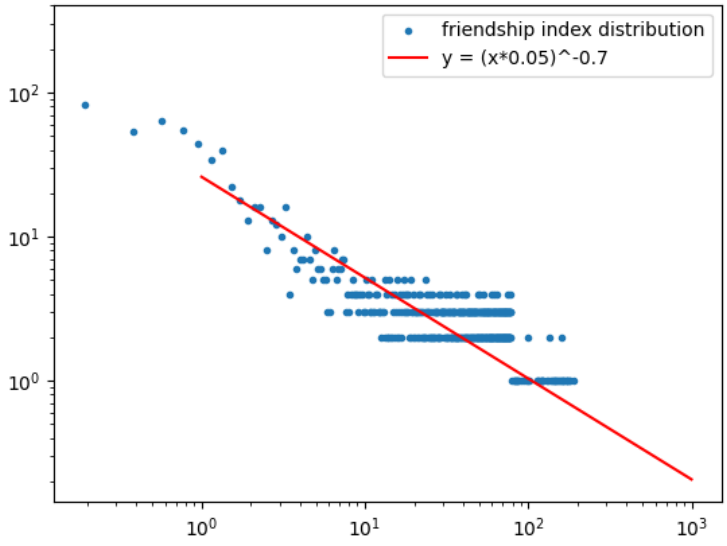

Figure 7 – Friendship index distribution in triad closure model with $m = 3$ and $p = 0.25$, on the logarithmic scale

power function

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

Figure 10 shows a plot of the dynamics of the average friendship index in

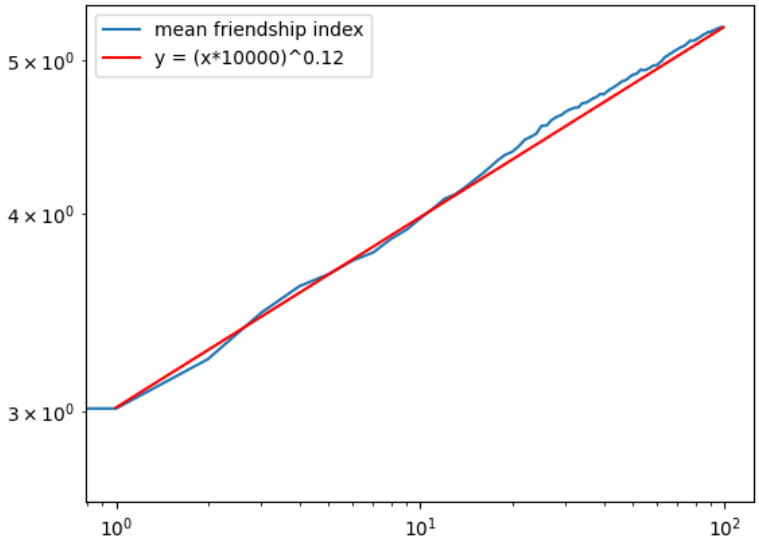

Figure 8 – Dynamics of the average value of the friendship index in the standard Barabasi— Albert model with $m = 3$, on the logarithmic scale

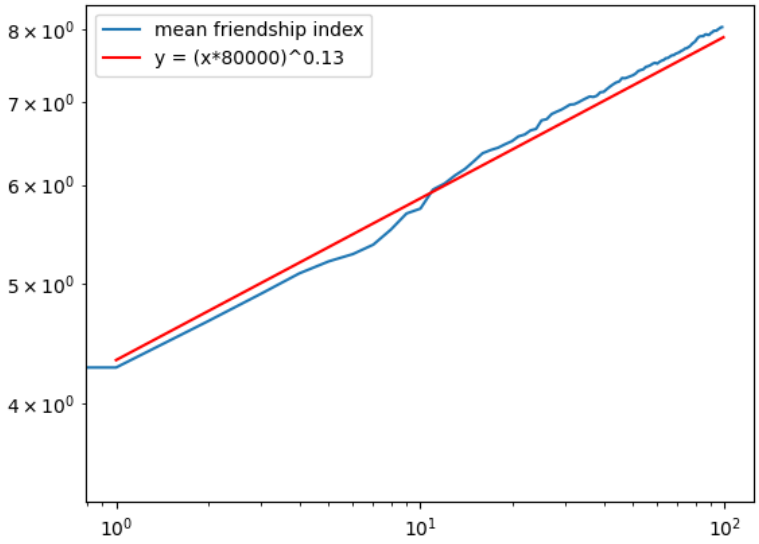

Figure 9 – Dynamics of the average value of the friendship index in modified Barabasi— Albert model with $m = 4$, on the logarithmic scale

the triad closure model at $m = 3$ and $p = 0.25$ on the logarithmic scale, which is identical to the power function:

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

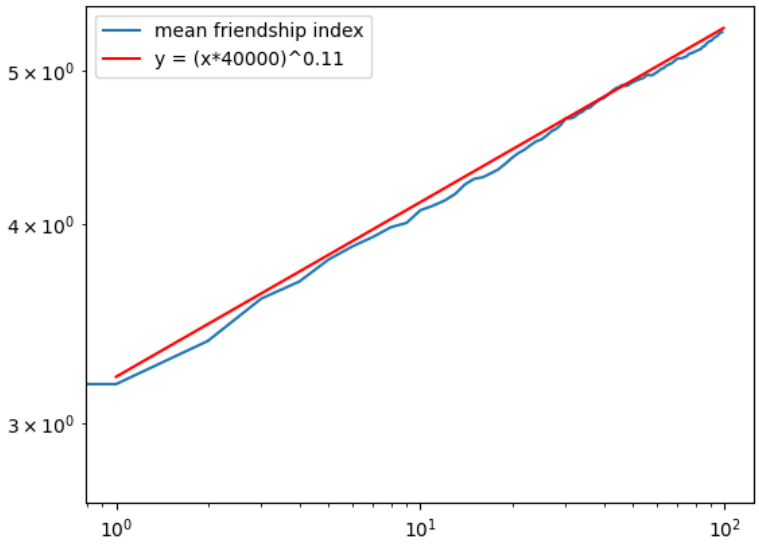

Figure 10 – Dynamics of the average value of the friendship index in the triad closure model
with $m = 3$, on the logarithmic scale

Diagrams of the friendship index dynamics in other models are presented in the appendix C.

4.2 Analysis of the distribution and dynamics of the friendship index in real networks

In the course of the tesis, graphs of the following networks from the real world were considered:

- LiveJournal
- Twitter
- Reddit
- Google+
- askubuntu
- mathoverflow
- superuser
- citation network for articles in the field of high energy physics phenomenology
- internal social network of the University of California, Irvine

4.2.1 Analysis of the distribution of the friendship index in the LiveJournal social network

LiveJournal (LJ) is a blog platform for keeping online diaries (blogs), as well as a separate personal blog hosted on this platform. Provides an opportunity to publish your own and comment on other people's posts, run collective blogs ("communities"), add other users to friends ("friend") and follow their posts in the "friends feed"("friendlent").

The thesis contemplates a dataset based on the state of the LiveJournal platform in 2009 and presented in [13]. In the graph, users act as vertices, and the presence of an edge between the vertices indicates a friendship relationship between the corresponding users. The graph contains $4\,847\,571$ nodes and $68\,993\,773$ edges, which makes it too large to consider the growth dynamics of the average friendship index, but on the other hand, it allows us to study the distribution of the friendship index in a large graph.

Fig. 11 shows a diagram of the distribution of the friendship index in the described graph. We can assume that the friendship index is distributed according to the power law

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

Note that the similarity with the power law of distribution of values is also observed in graphs constructed in accordance with the models of random graphs considered earlier in this paper.

4.2.2 Analysis of the distribution of the friendship index in the Twitter social network

Twitter is an American microblogging service and social network where users post messages known as "tweets"and interact with them. Users interact with Twitter through a browser, mobile app, or API. Until April 2020, the services were available via SMS. The service is provided by Twitter, Inc., which is based in San Francisco, California, and has more than 26 offices worldwide. In 2013, it entered the top ten most visited sites and was named "SMS of the Internet". By the beginning of 2019, Twitter had more than 330 million monthly active users. In practice, the vast majority of tweets are written by a minority of users.

Data about this network is taken from [14]. The graph describes the relationship of friendship between confirmed users belonging to certain circles such as: «artists»,

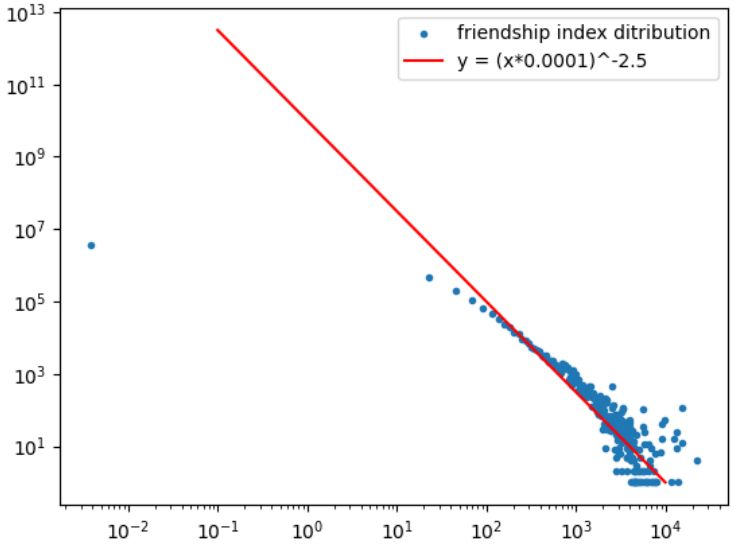

Figure 11 – Distribution of the friendship index in
social network «LiveJournal»
on the logarithmic scale

«politicians», etc. It contains 81 306 vertices connected by 1 768 149 edges. On the graph shown in Fig. 12, it can be found that the distribution of the friendship index in this network does not correspond well to the power function.

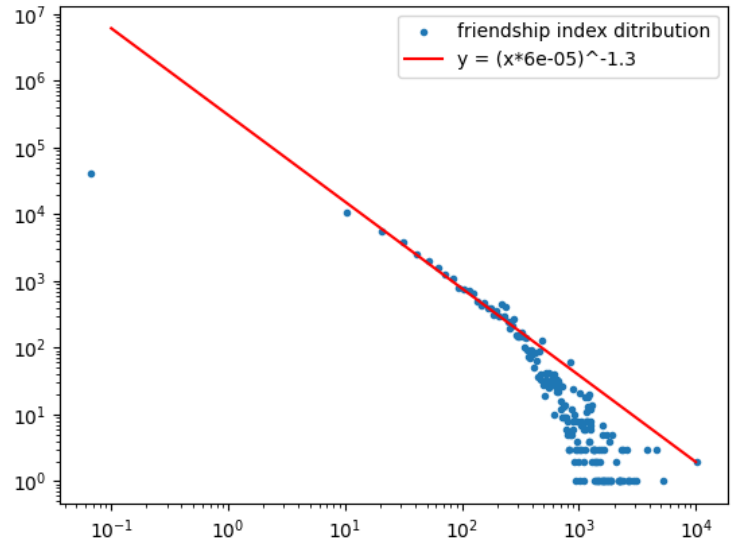

Figure 12 – Distribution of the friendship index in
social network «Twitter»
on the logarithmic scale

In this case dynamics of the friendship index also was not possible to be

calculated.

4.2.3 Analysis of the distribution and dynamics of the friendship index in the citation network of articles in the field of high energy physics phenomenology

The data set contains information on $34\,546$ high-energy physics articles published from January 1993 to April 2003 on arXiv.org — electronic archive with open access for scientific articles and preprints in physics, mathematics, astronomy, computer science, biology, electrical engineering, statistics, financial mathematics and economics. The graph contains $421\,578$ edges. The dataset was introduced in [15].

Figure 13 shows the distribution of the friendship index in this network. From this plot, we can assume that the friendship index in the citation network, like the friendship index in the reviewed random graphs, is distributed according to the power law

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

.

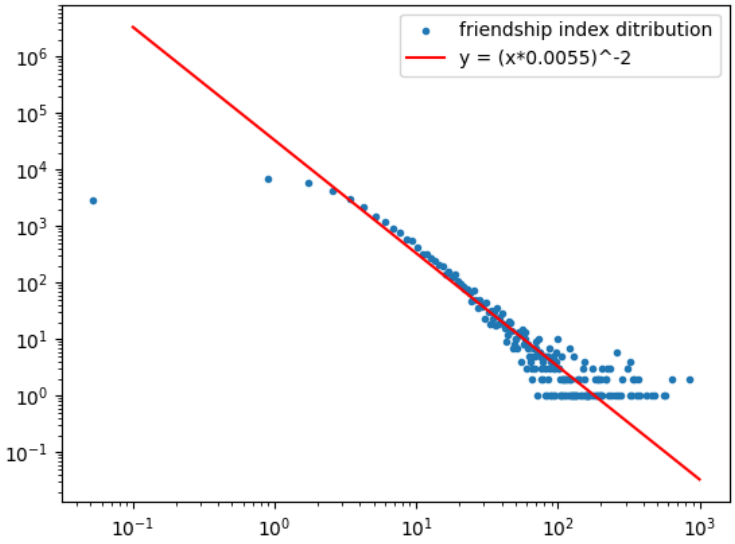

Figure 13 – Distribution of the friendship index in the citation network
on the logarithmic scale

A diagram of the dynamics of the friendship index in this network is shown in Fig. 14. We can hypothesize that the average value of the friendship index in the

graph grows in accordance with the power function

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

This, in terms of this assumption, corresponds to the rate of growth of the friendship index in the networks of the Barabasi— Albert and triad closure models.

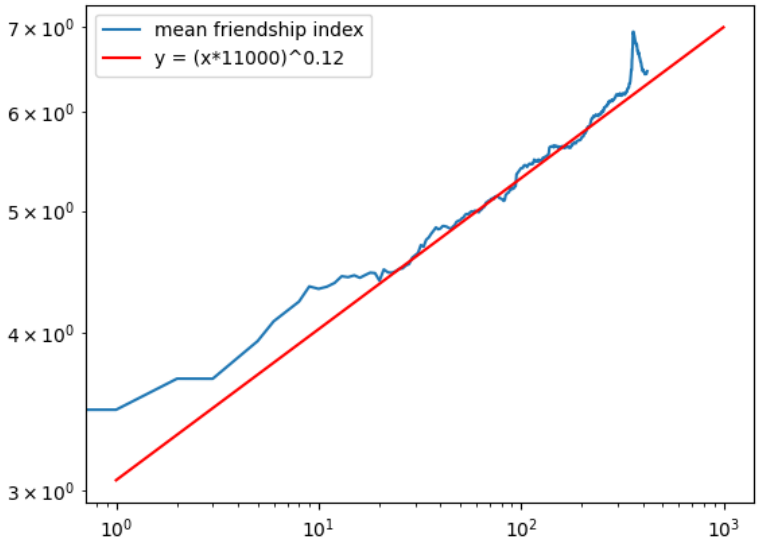

Figure 14 – Dynamics of the friendship index in the citation network
on the logarithmic scale

4.2.4 Analysis of the distribution and dynamics of the friendship index in the Reddit forum network

Reddit is a site that combines the features of a social network and a forum, where registered users can post links to any information they like on the Internet and discuss it. Like many other similar sites, Reddit supports a voting system for liked posts — the most popular ones appear on the main page of the site. One of the most popular sites in the world — 20th place in terms of traffic according to SimilarWeb.

In this case, we consider a network of hyperlinks between subreddits (communities on «Reddit») data was collected over two and a half years: from January 2014 to April 2017. The graph consists of 55 863 nodes connected by858 490 edges. The data was submitted in [16].

Figure 15 shows a chart of distribution of the friendship index in the hyperlink graph between subreddits. From the graph, we can assume that the distribution of the

friendship index in this network also corresponds to a power function, which in this case looks like this:

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

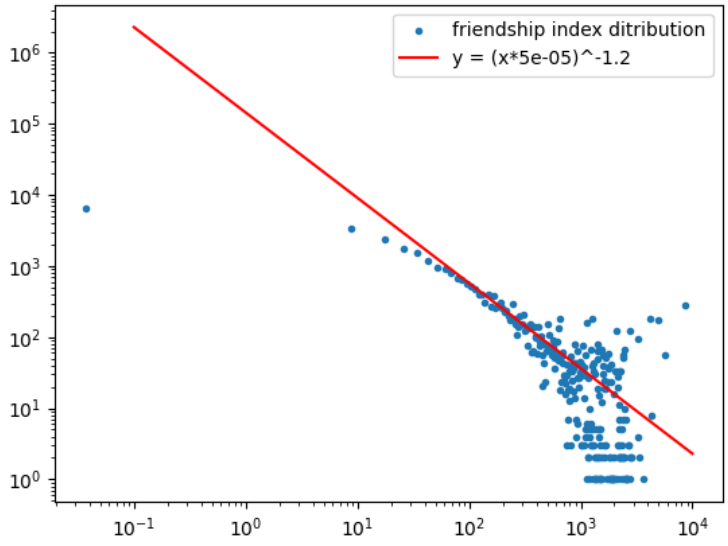

Figure 15 – Distribution of the friendship index in Reddit forum network on the logarithmic scale

Figure 16 indicates that the diagram of the friendship index dynamics on the Reddit forum is similar to the graph of the power function

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

4.2.5 Analysis of the distribution and dynamics of the friendship index in the AskUbuntu Q&A system

AskUbuntu — a website for working with questions and answers that arise when using the Ubuntu Linux distribution. Sites allow users to ask and answer questions. By taking into account activities, voting takes place for questions and answers, positive and negative. You can edit questions and answers in wiki mode. All materials published online are available under the free CC BY-SA license.

4 datasets were reviewed:

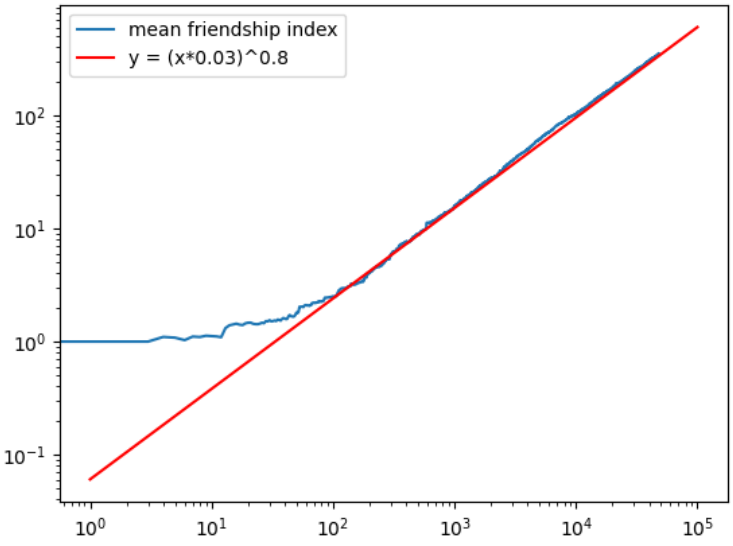

Figure 16 – Dynamics of the friendship index in the Reddit forum network on the logarithmic scale

— graph of answers to the questions, which consists of137 517 vertices representing users, and 280 102 connections between them. the connection between users A and B is that user A answered the user's question B;

— a question citation graph containing 79 155 nodes and 327 513 edges, where a directed edge is drawn between the user and the author of the question that they cited.

— response citation graph consisting of 75 555 and 356 822 vertices and edges, respectively, an edge is drawn from the user who referred to the answer to the user who wrote it.

— and a general graph containing all users and all edges of all previous graphs, with 159 316 nodes and 964 437 connections.

Data for these graphs have been collected for more than seven years, and they were presented in the article [17].

The properties of this network do not fully correspond to the behavior of the models studied in the thesis: in Fig. 17, you can see that the distribution of the friendship index does not correspond well to the power function and, therefore, does not correspond to the constructed models.

However, the average value of the friendship index, as in models of random

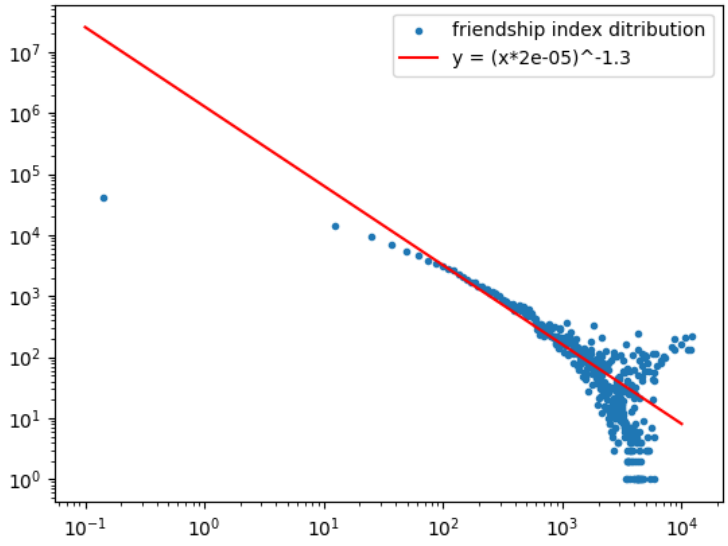

Figure 17 – Distribution of the friendship index in the AskUbuntu network on the logarithmic scale

graphs, grows rather accordingly to the power law

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

as shown in the chart 18.

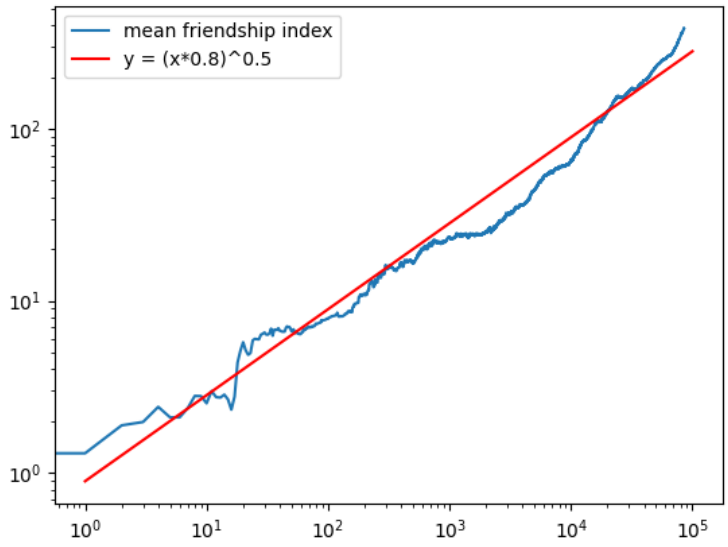

Figure 18 – Dynamics of the friendship index in the AskUbuntu network on the logarithmic scale

These patterns are most clearly shown in the graph of the network of answers to questions. Plots of the distribution and dynamics of the index for this network are shown in Fig. 19 and 20, respectively.

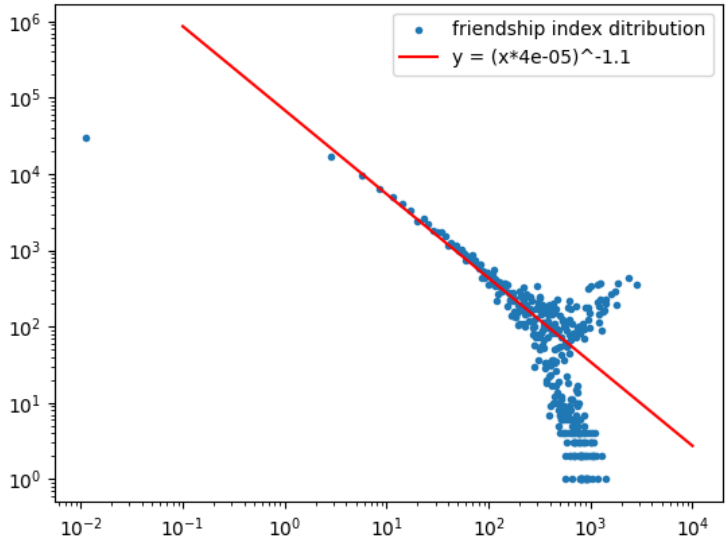

Figure 19 – Distribution of the friendship index in The AskUbuntu question answer network on the logarithmic scale

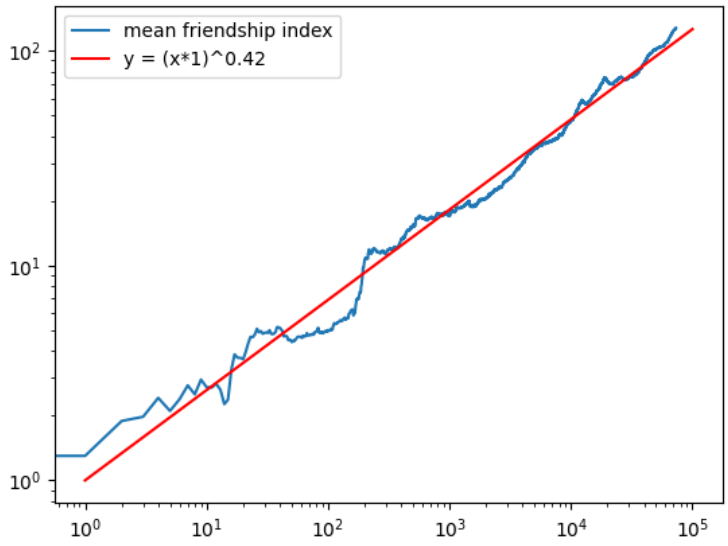

Figure 20 – Dynamics of the friendship index in the network of answers to questions on AskUbuntu on the logarithmic scale

Graphs of other subnets are shown in the Appendix G

4.2.6 Analysis of the distribution and dynamics of the friendship index in the SuperUser Q&A system

SuperUser, like AskUbuntu, is a Q&A platform, and unlike the previous network, it is dedicated to computer enthusiasts. Like AskUbuntu, it acts as one of the sections of StackExchange and it follows the same basic network rules: anyone can ask a question, anyone can answer it, and anyone can vote for the answer or question.

Three subnets were also allocated for this network:

— question answers network
— question citation network
— response citation network

The data were also presented in [17].

The network of the SuperUser platform does not follow the patterns identified in random graphs in the course of this work. In Fig. 21, it can be seen that the plot of the distribution of the friendship index rather poorly coincides with the power function, as well as the diagram of the dynamics of the friendship index presented in Fig. 22.

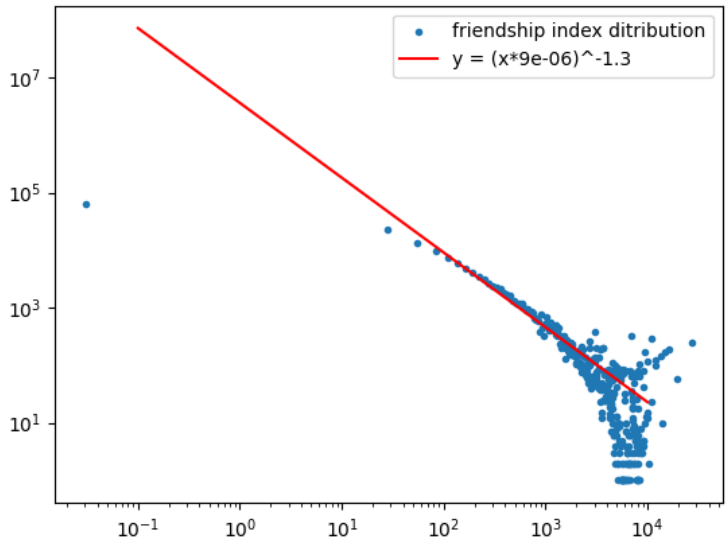

Figure 21 – Distribution of the friendship index in SuperUser network on the logarithmic scale

In the question citation network, this deviation only increases, as can be seen in the graphs 23 and 24.

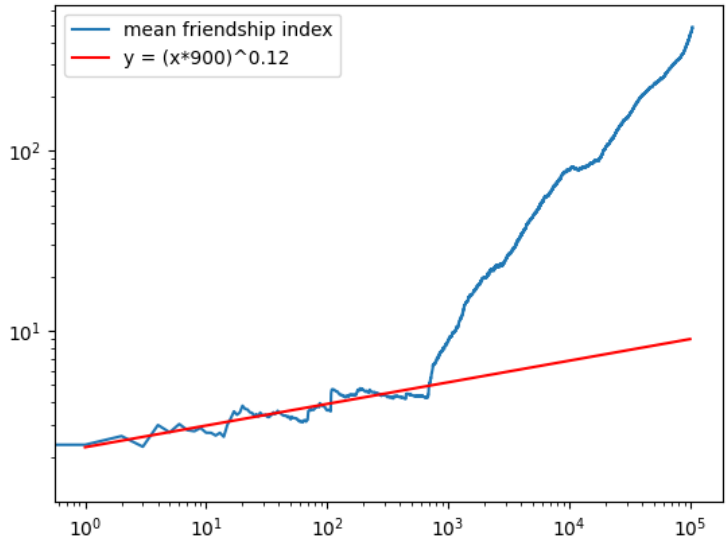

Figure 22 – Dynamics of the friendship index in SuperUser network on the logarithmic scale

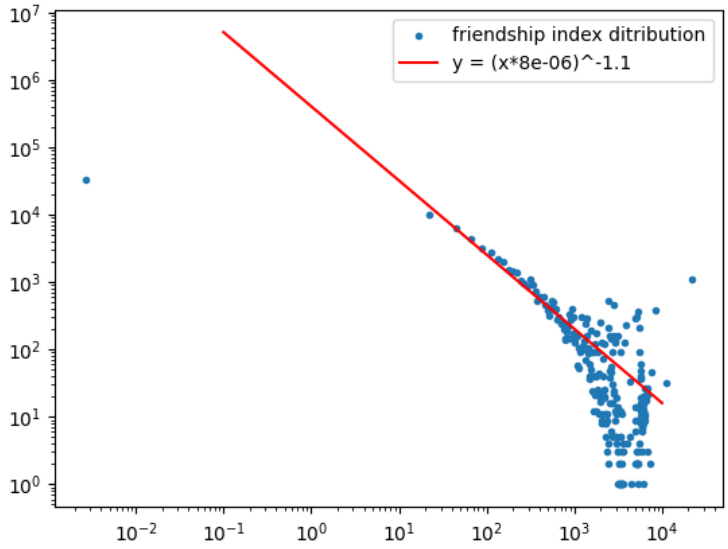

Figure 23 – Distribution of the friendship index in The SuperUser question citation network on the logarithmic scale

At the same time, in the sub-network of answers to questions, the dynamics of the friendship index still probably obeys the power law, which can be observed in Fig. 25.

Graphs for the remaining subnets are presented in the G app. There are also graphs of the distribution and dynamics of the friendship index in Google+,

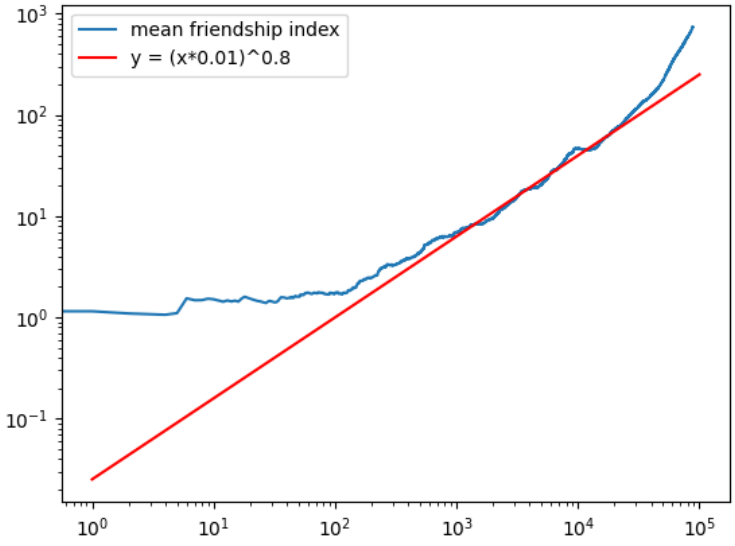

Figure 24 – Dynamics of the friendship index in
The SuperUser question citation network
on the logarithmic scale

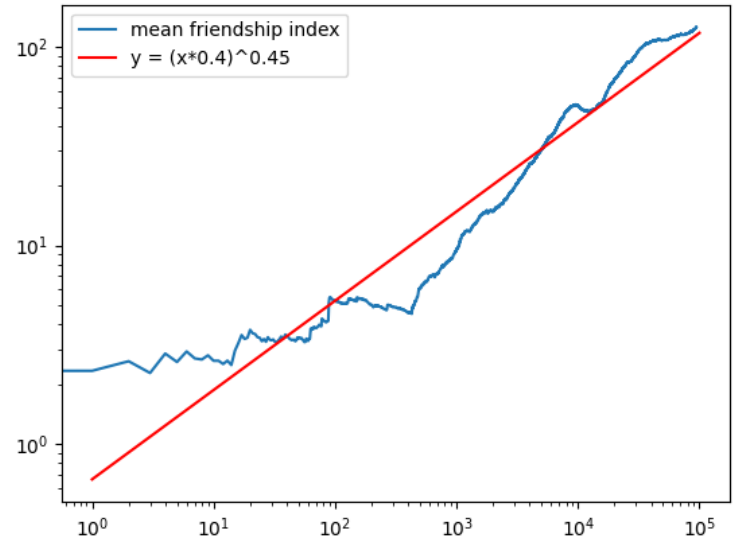

Figure 25 – Dynamics of the friendship index in
The SuperUser question answer network
on the logarithmic scale

MathOverflow, and student messages network of the University of California in Irvine, data for which were taken from the works [14], [17], and [18]. In the**??** a comparative analysis of all the considered networks is presented.

Table 1 – Summary table of experimental results

Network	Friendship Index Distribution Function	γ_{dist}	Friendship Index Dynamics Function	γ_{dyn}
Barabasi— Albert model at $m = 3$	power-law	-0.7	power-law	0.15
Barabasi— Albert model at $m = 3$	power-law	-0.7	power-law	0.15
Barabasi— Albert model with Poisson distribution of initial degrees at $m = 4$	power	-0.5	power	0.15
Barabasi— Albert model with Poisson distribution of initial degrees at $m = 5$	power	-0.5	power	0.15
Barabasi— Albert model with Poisson distribution of initial degrees at $m = 6$	power	-0.45	power	0.15
Triad closure model for $m = 3$ and $p = 0.25$	power	-0.7	power	0.15
Triad closure model for $m = 3$ and $p = 0.5$	power-law	-0.7	power-law	0.15
Triad closure model for $m = 3$ and $p = 0.75$	power	-0.4	power	0.12
Triad closure model for $m = 5$ and $p = 0.25$	power-law	-0.6	power-law	0.13
Triad closure model for $m = 5$ and $p = 0.5$	power-law	-0.6	power-law	0.13
Triad closure model for $m = 5$ and $p = 0.75$	power	-0.55	power	0.12
Twitter	unknown function	N/A	N/A	N/A
Google+	power-law	-1.7	N/A	N/A
Reddit	power	-1.2	power	0.8
AskUbuntu	unknown function	N/A	power function	0.5
MathOverflow	power-law	-1.2	power-law	0.45
SuperUser	unknown function	N/A	exponential	N/A
Citation Network	power	-2	power	0.12
LiveJournal	Power	-2.5	N/A	N/A
Student Message Network	Power	-0.9	Power	0.5

CONCLUSION

During the course of the thesis, various models of generating growing networks were studied. And the following random graph generation models were implemented:

- standard Barabasi—Albert model;
- Barabasi—Albert model with Poisson distribution of initial degrees of nodes;
- the triad closure model.

A series of experiments was conducted in which random graphs were constructed according to the implemented models, and the dynamics of the average value of the friendship index in the network and the distribution of the values of the friendship index in the final graph were studied. In accordance with the results obtained, it is hypothesized that the distribution of the friendship index of the vertices of the final graph and the growth of the average value of the friendship index during the formation of the network occurs according to the power law.

The values of the distribution of the friendship index and the growth dynamics of the average value of the friendship index in a number of real networks were studied. It is concluded that for most networks, the behavior of the friendship index is similar to its behavior in networks growing according to the studied models.

The models were implemented and experiments were performed using Python with the json, multiprocessing, pyplot, numpy, and random libraries.

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

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

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

7 *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.

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

9 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.

10 *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.

11 *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.

12 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.

13 *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.

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

15 *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.

16 *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.

17 *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.

18 *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.

APPENDIX A
Text of the program

This appendix contains the full text of the implementation of the Barabasi—Albert model.

```
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
```

```
35              neibours[i + 1].append(j)
36              neibours[j].append(i)
37              degrees[j] += 1
38          degrees.append(mi)
39          nodes = np.append(nodes, nodes.shape[0])
40          for j in range(len(dts)):
41              if i % dts[j] == 0:
42                  ans[j].append(funcs[j](degrees, neibours))
43      return ans
44
45  def my_triad(n, args, funcs, dts):
46      m = args[0]
47      p = args[1]
48      ans = [[] for _ in range(len(dts))]
49      degrees = list(np.full(m + 1, m))
50      neibours = [list(np.delete(np.arange(m + 1), i)) for i in np.arange(m +
        ↪ 1)]
51      nodes = np.arange(m + 1)
52      used = np.full(n, False)
53      for i in range(m, n):
54          conections = []
55          j = 0
56          while j < m:
57              if j == 0 or np.random.rand() > p:
58                  choosen = random.choices(nodes, weights = degrees, k = 1)[0]
59              else:
60                  choosen = random.choices(neibours[conections[0]], k = 1)[0]
61              if not used[choosen]:
62                  j += 1
63                  conections.append(choosen)
64                  used[choosen] = True
65          neibours.append([])
66          for j in conections:
67              used[j] = False
68              neibours[i + 1].append(j)
69              neibours[j].append(i)
70              degrees[j] += 1
71          degrees.append(m)
72          nodes = np.append(nodes, nodes.shape[0])
73          for j in range(len(dts)):
74              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()
```

APPENDIX B

Friendship index distribution graphs in the constructed graphs

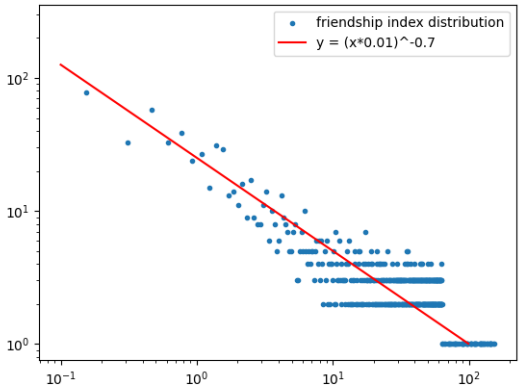

Figure 26 – Distribution of the friendship index in the Barabasi— Albert model with $m = 3$

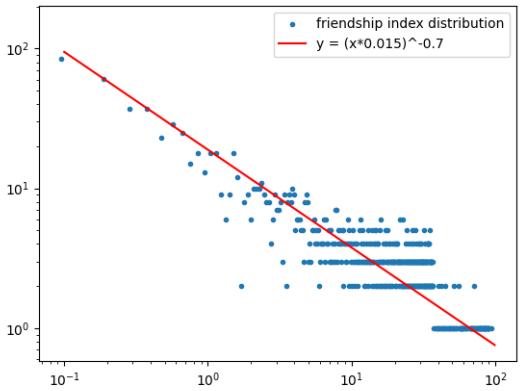

Figure 27 – Distribution of the friendship index in the Barabasi— Albert model with $m = 5$

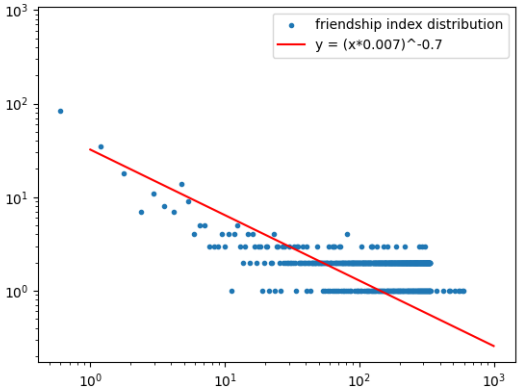

Figure 28 – Distribution of the friendship index in the Barabasi— Albert model with a Poisson distribution of degrees of new vertices for $\lambda = 4$

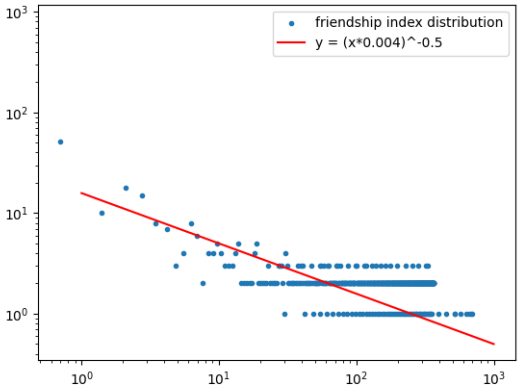

Figure 29 – Distribution of the friendship index in the Barabasi— Albert model with a Poisson distribution of degrees of new vertices for $\lambda = 5$

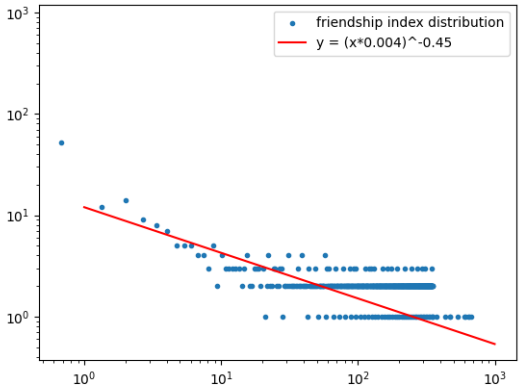

Figure 30 – Distribution of the friendship index in the Barabasi— Albert model with a Poisson distribution of degrees of new vertices for $\lambda = 6$

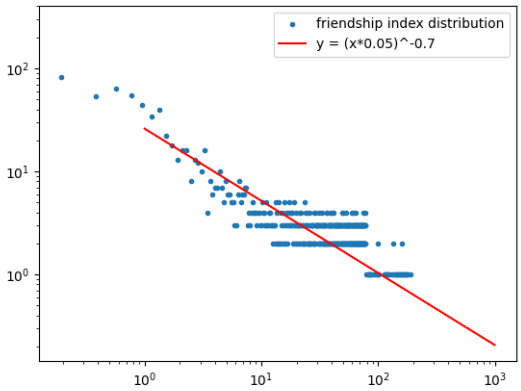

Figure 31 – Friendship index distribution in triad closure model with $m = 3$ and $p = 0.25$

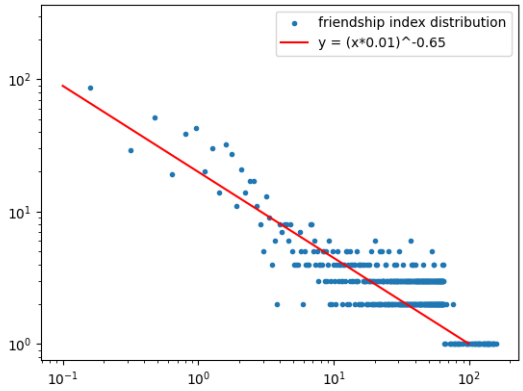

Figure 32 – Friendship index distribution in triad closure model with $m = 3$ and $p = 0.5$

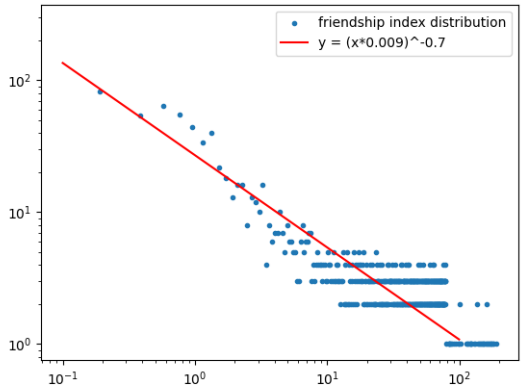

Figure 33 – Friendship index distribution in triad closure model with $m = 3$ and $p = 0.75$

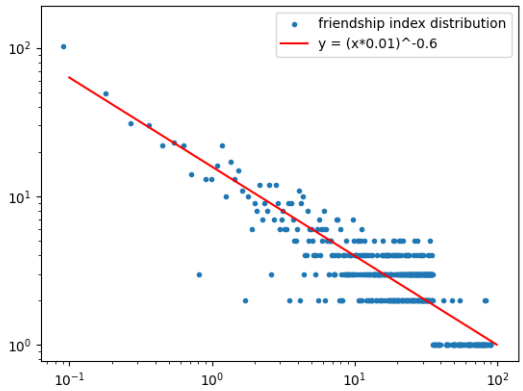

Figure 34 – Friendship index distribution in triad closure model with $m = 5$ and $p = 0.25$

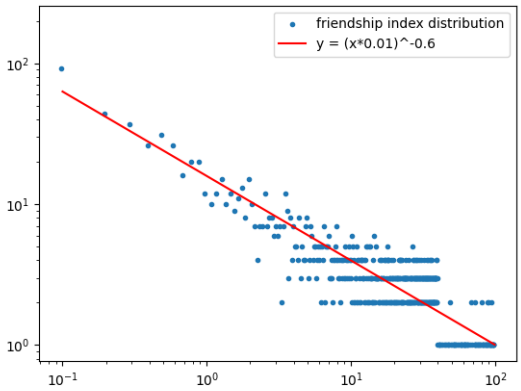

Figure 35 – Friendship index distribution in triad closure model with $m = 5$ and $p = 0.5$

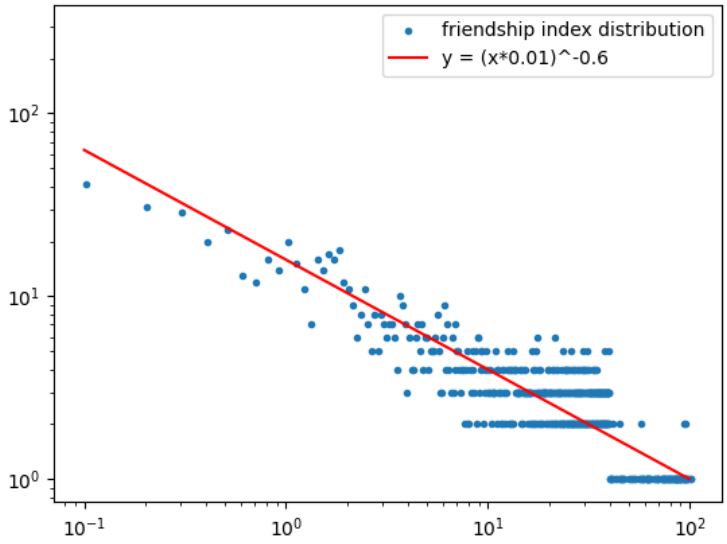

Figure 36 – Friendship index distribution in triad closure model with $m = 5$ and $p = 0.75$

APPENDIX C

Graphs of the dynamics of the average friendship index in the constructed graphs

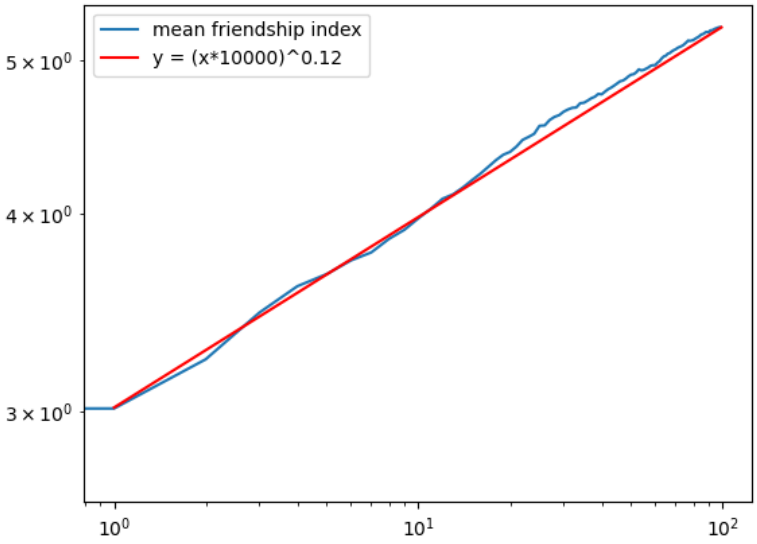

Figure 37 – Dynamics of the average friendship index in the Barabasi— Albert model with $m = 3$

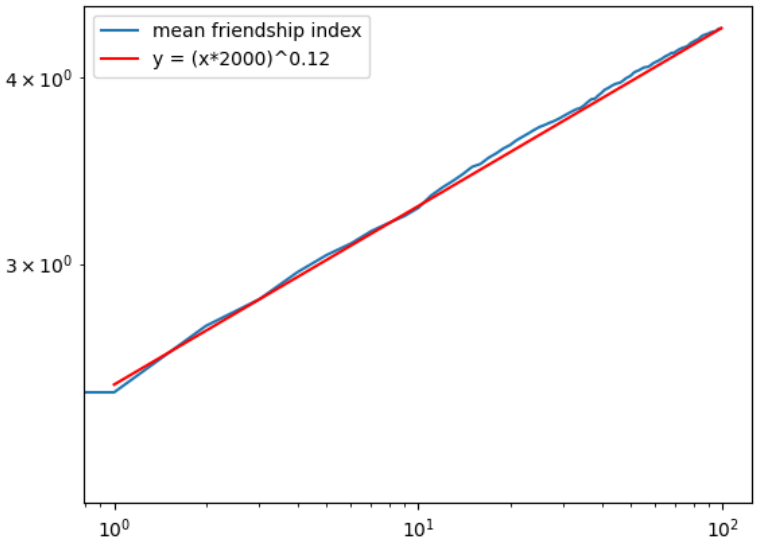

Figure 38 – Dynamics of the average friendship index in the Barabasi— Albert model with $m = 5$

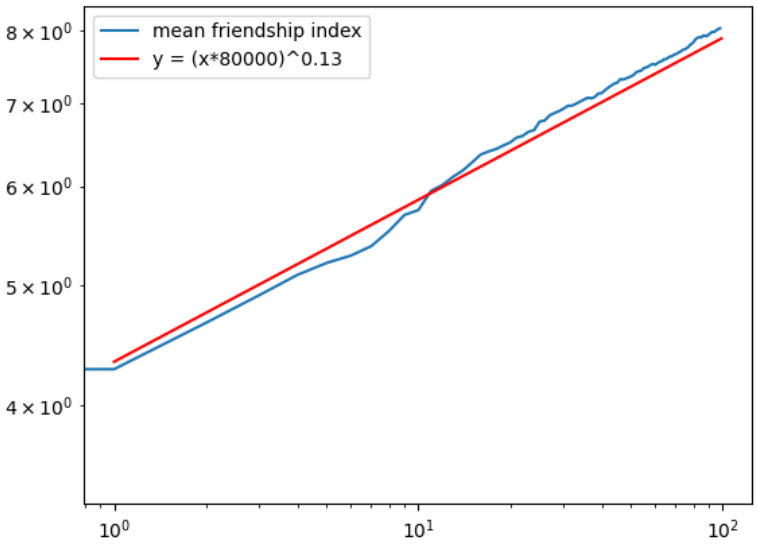

Figure 39 – Dynamics of the average friendship index in the Barabasi— Albert model with Poisson distribution degrees of new vertices with $\lambda = 4$

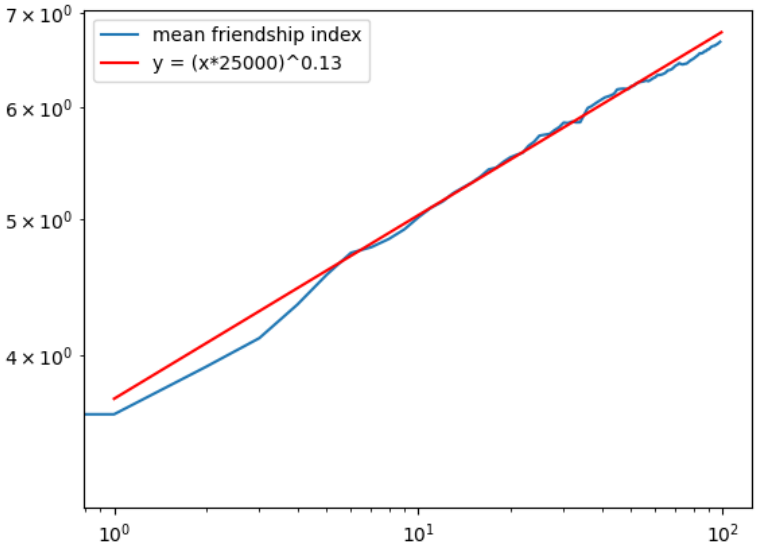

Figure 40 – Dynamics of the average friendship index in the Barabasi— Albert model with Poisson distribution degrees of new vertices with $\lambda = 5$

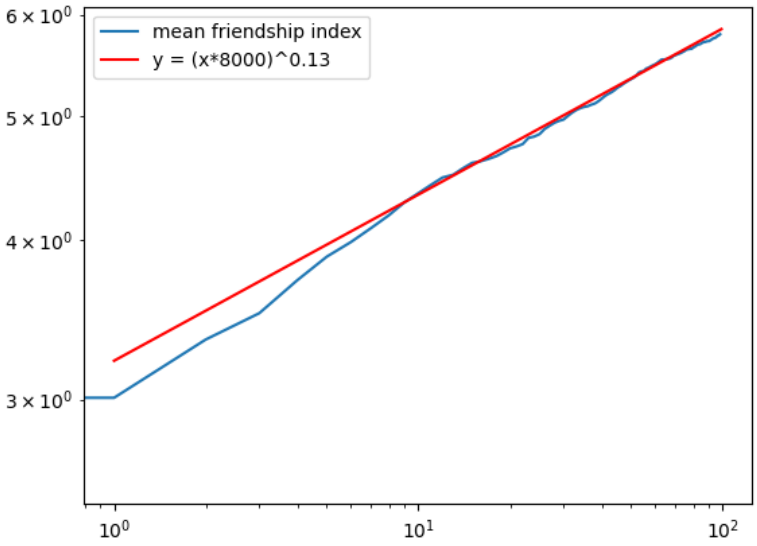

Figure 41 – Dynamics of the average friendship index in the Barabasi— Albert model
with Poisson distribution
degrees of new vertices
with $\lambda = 6$

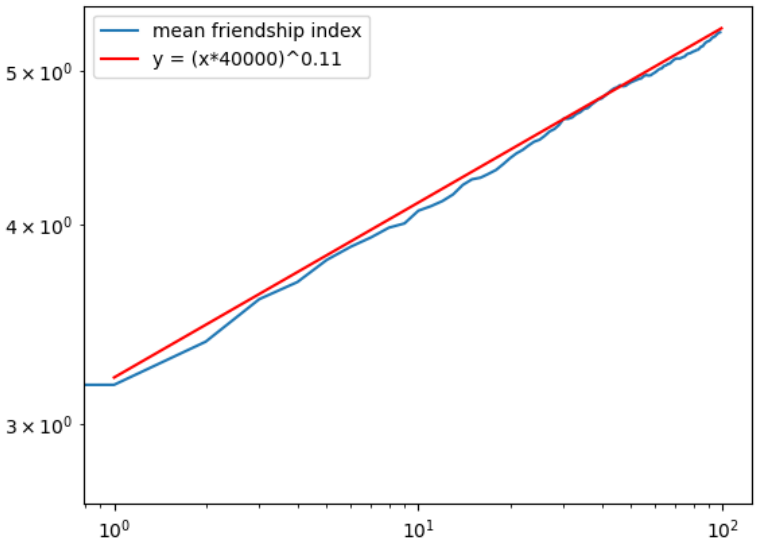

Figure 42 – Dynamics of the average friendship index in the triad closure model
with $m = 3$ and $p = 0.25$

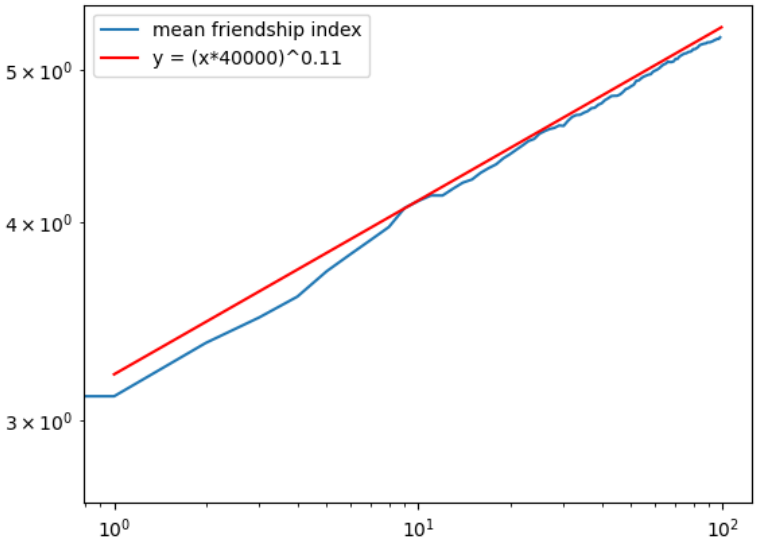

Figure 43 – Dynamics of the average friendship index in the triad closure model
with $m = 3$ and $p = 0.5$

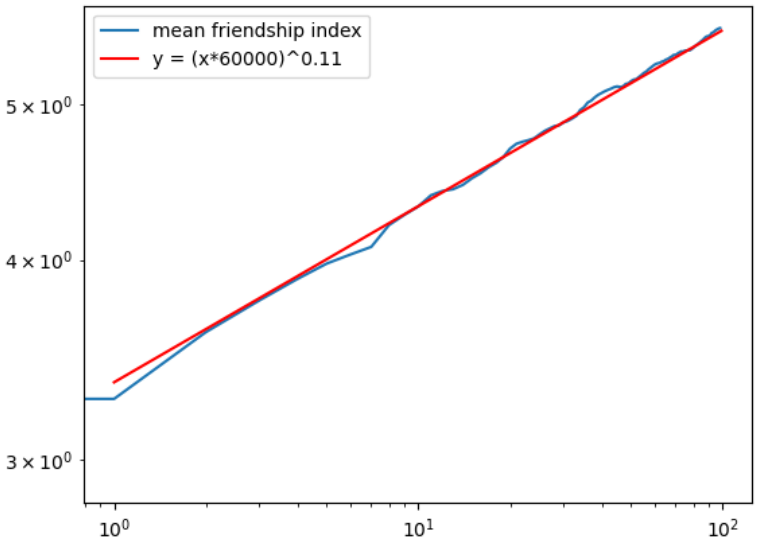

Figure 44 – Dynamics of the average friendship index in the triad closure model
with $m = 3$ and $p = 0.75$

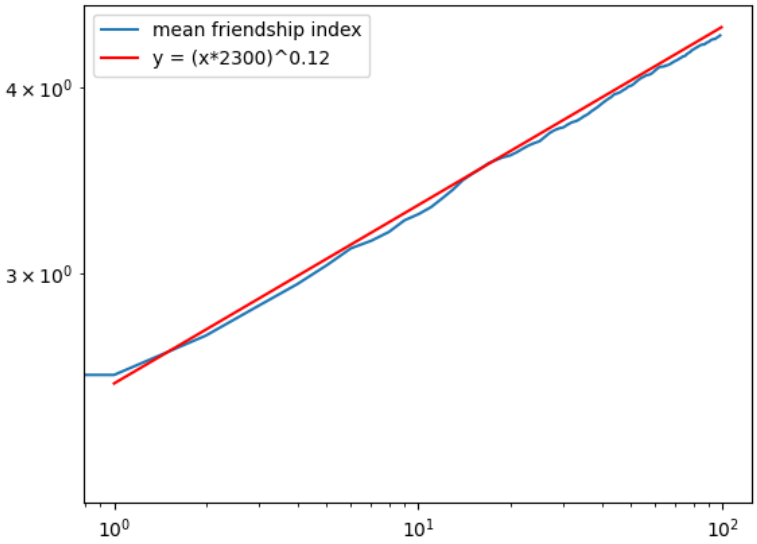

Figure 45 – Dynamics of the average friendship index in the triad closure model
with $m = 5$ and $p = 0.25$

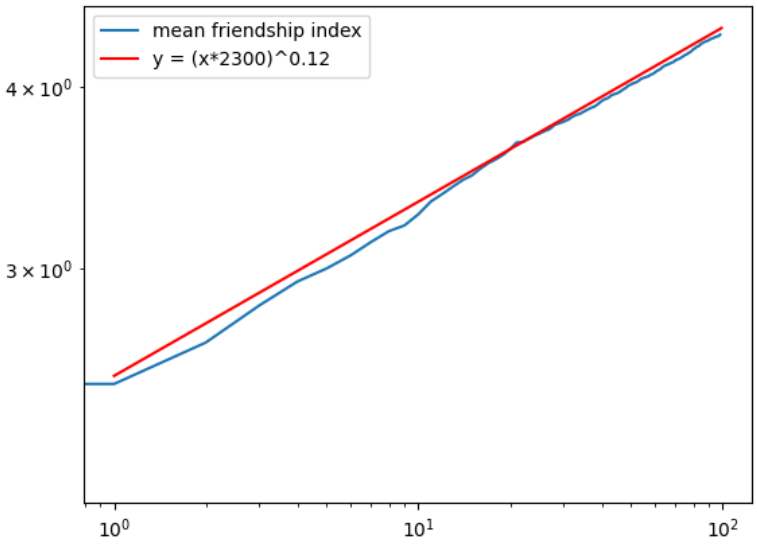

Figure 46 – Dynamics of the average friendship index in the triad closure model
with $m = 5$ and $p = 0.5$

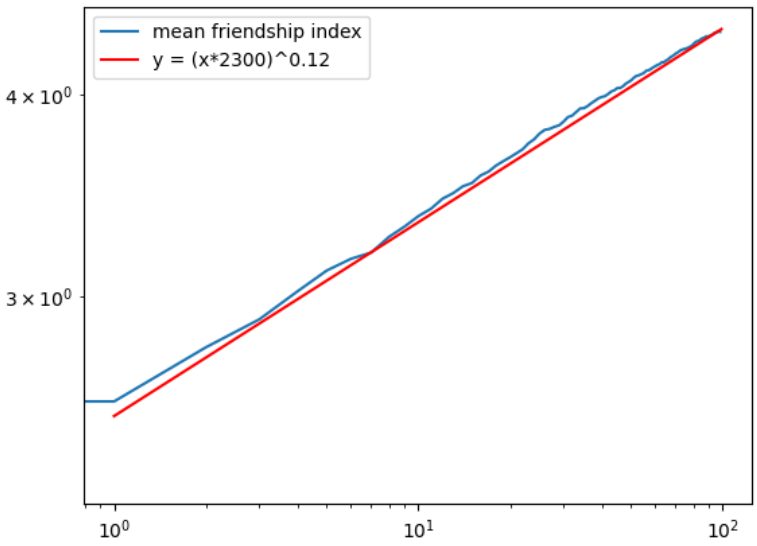

Figure 47 – Dynamics of the average friendship index in the triad closure model
with $m = 5$ and $p = 0.75$

APPENDIX D
Script text for displaying the friendship index distribution

```
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')
```

APPENDIX E
Script text for displaying the dynamics of the friendship index

```
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')
```

APPENDIX F

Script text for displaying the dynamics of the friendship index in graphs presented in several files

```
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')
```

APPENDIX G

Graphs of the distribution and dynamics of the friendship index in real networks

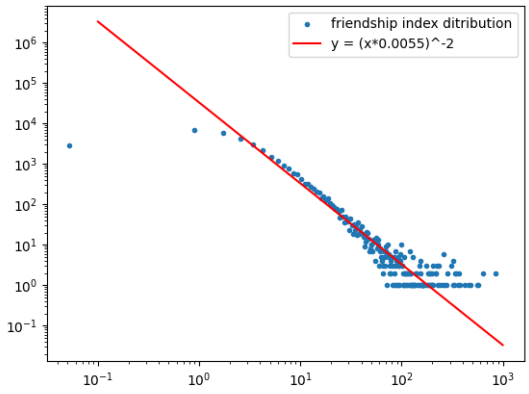

Figure 48 – Distribution of the friendship index in the citation network of articles in the field of high energy physics phenomenology on the logarithmic scale

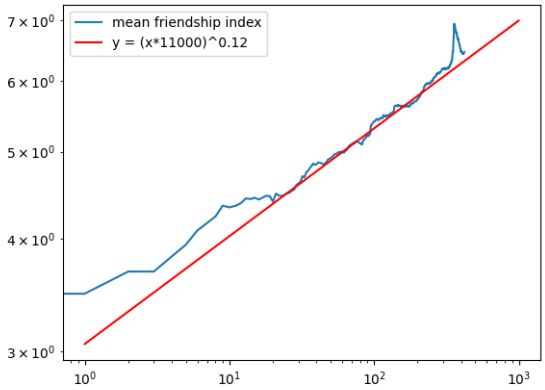

Figure 49 – Dynamics of the friendship index in the network of citations of articles in the field of phenomenology of high energy physics on the logarithmic scale

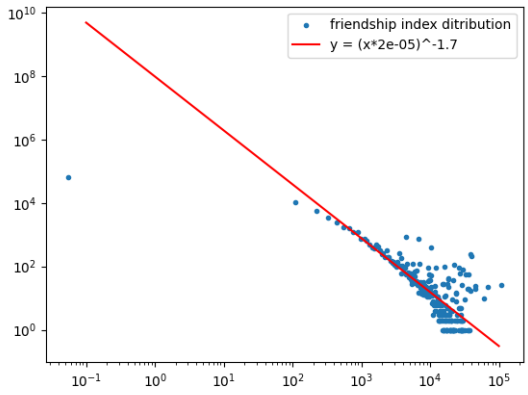

Figure 50 – Distribution of the friendship index in Google+ social network on the logarithmic scale

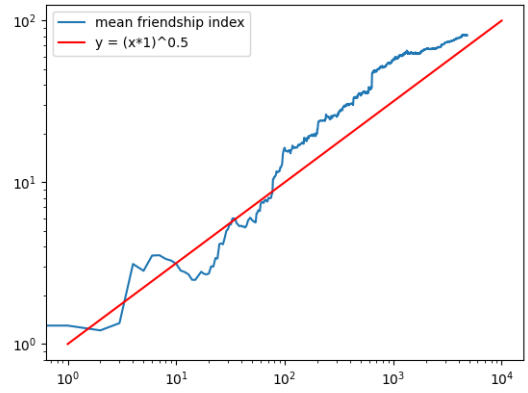

Figure 51 – Dynamics of the friendship index in the message network of students at the University of California, Irvine on the logarithmic scale

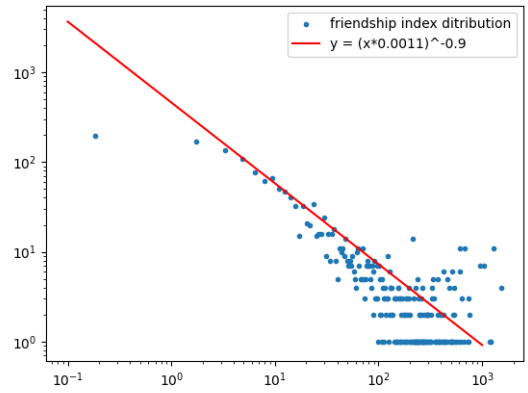

Figure 52 – Distribution of the friendship index in the UC Irvine student message network on the logarithmic scale