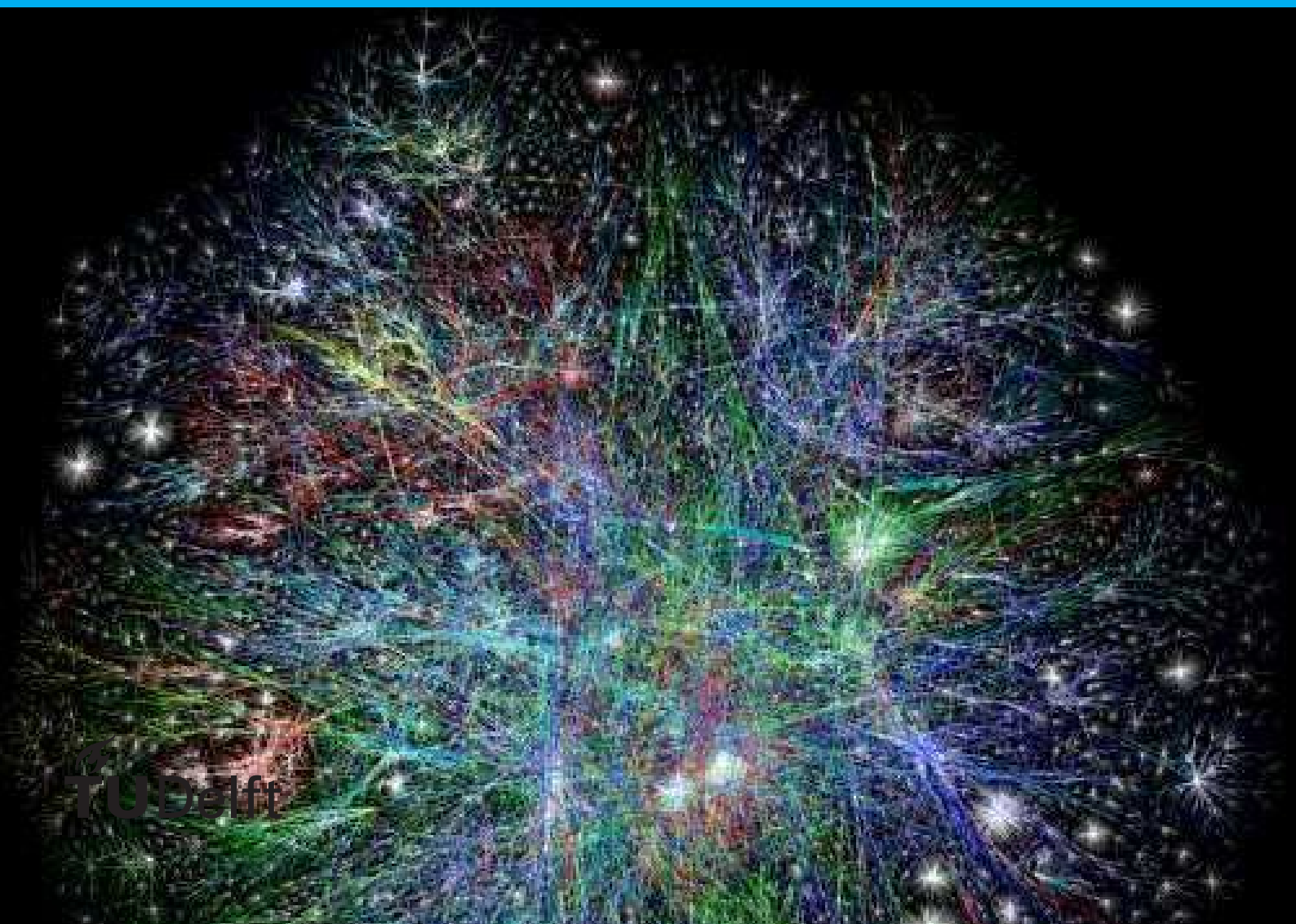
Random Graphs with Time-Varying Fitness

Runsheng Wang

Master Thesis
EEMCS
Applied Mathematics



Random Graphs with Time-Varying Fitness

by

Runsheng Wang

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday July 20, 2020 at 10:00 AM.

Student number:	4806727	
Project duration:	December 1, 2019 – July 20, 2020	
Thesis committee:	Prof. dr. F. H. J. Redig,	TU Delft
	Dr. A. Cipriani,	TU Delft, supervisor
	Dr. H. N. Kekkonen,	TU Delft

This thesis is confidential and cannot be made public until July 31, 2020.

TU Delft

Preface

This report represents my MSc Thesis for the Master of Science in Applied Mathematics degree at Delft University of Technology. I have spent almost eight months on the random graphs and finally made some progress in the theoretical area. The proof and simulations are not part of my ambition, but the step in an unknown section makes me excited.

For the successful completion of this thesis, I was privilleged to be guided by my daily supervisor Dr. Alessandra Cipriani, to whom I want to express my gratitude. Her knowledge, guidance, patience, and positive attitude during the whole project was of utmost importance for the successful realization of the thesis. I would also like to thank my other thesis committee members: Prof. Frank Redig and Dr. Hanne Kekkonen. Their strict requirements are essentially determinant on writing the thesis.

It was a hard time for students and employees at TU Delft, due to the COVID-19. My father passed away in April, and I was not even able to meet him one last time. Also, I was stuck in a third country because of the quarantine policy and took a risk staying with others. I would say thank you to all the medical personnel for the protection from the virus. I would say thank you to my family in China, because you are always there no matter what has happened. I would say thank you to my friends, because it was also you who encouraged me to cheer up to continue my study.

Runsheng Wang
Delft, July 2020

Contents

Chapter 1

Introduction

One of the well-known random graph models is Erdös-Rényi random graph (ER graph) [9], which plays an important role in social network simulations. The ER graph is a random graph with fixed vertices and a probability $p \in (0,1)$ of existence of an edge between any two nodes. ER graph is the first mathematical attempt to model real-life networks. One important feature of a graph is the degree, which represents the popularity of a node. It is proven that the degree sequence of the ER graph does not have a heavy tail, i.e. there is no $\tau > 0$, such that

$$\mathbb{P}(D(i) > k) \sim k^{-\tau}, \quad \text{as } k \to \infty \tag{1.1}$$

where $D(i)$ denotes the degree of node i, and $k \in \mathbb{N}$. Heavy tails are common in empirical networks, so ER graph is a kind of expected model but resembles few natural cases. For example, in search engines and some other applications, only degree is insufficient to express nodes' tendency to get more links. Thus, the fitness is introduced in a random graph to make a node more or less attractive for attachment. Normally, fitness values of each node are assigned by sampling from a fixed distribution. According to different fitness laws, random graphs with fitness can be divided into many types. But most of the random graphs with fitness share a common property: their degree sequences follow some power-law, satisfying the equation (1.1), and different models may have different tail exponents [6]. This is an interesting phenomenon. Another feature of random graph models is the decay of the degree sequence itself, and the degree sequence of a model may obtain a tail exponent of node i, i.e. with some fixed time t, there exist some positive constant c and μ, such that

$$D(i) \sim ci^{-\mu}. \tag{1.2}$$

This paper mainly focuses on the special property of some representative random trees.

The random recursive tree (RRT) and the preferential attachment model (PAM) can be two special models of dynamic random trees. Many researches have been made on both models, especially asymptotic results, so they could be nice benchmarks in this paper. As named, at each time step $t \in \mathbb{N}$, a RRT allows a new node to come into the network and attach itself to any of the old nodes uniformly at random. Pittel in 1994 has proved that the RRT does not have a power-law but an exponential tail of degree sequence [13]. Many people like Berry [2] and Janson [12] tested the same property, and our independent proof of this fact is in the Section 2. Also, another property of RRT was proven by Devroye in 1987 that RRTs have a height of $O(\log(t))$, where t means the number of nodes [10]. PAM was developed from the so-called Yule process, conceived by Yule [16] in 1925. In the standard PAM, also known as Albert-Barabási model [3], the tree branches from one original node and grows in time: in each new step, one new node is attached to any of the old nodes with a probability proportional to their degrees at that time. The PAM has a significant power-law in degree distribution [15]. In particular, the tail of the degree distribution of the Albert-Barabási model decays like k^{-3}, where k means the degree, k large (equation (2.9)). That leads to another main feature that older nodes will have more probability to gain degree by attachment of new nodes, which is the so-called 'rich-get-richer' phenomenon. There are still some restrictions to PAM, one of which could be the fixed tail exponent. In modern applications, it is required to reach various tail exponent of power-law, so PAM needs to be extended to some further models. Again by Barabási and his colleague Bianconi [4], an extra fitness function was added in the

attachment probability in order to vary the tail exponent. A sequence of independent fitness values coming from some fixed distribution are introduced into PAM as weights of the degree. It is proven that the tail exponent of the power-law changes after adding the fitness function, and the number can be influenced by the choices of fitness distribution [4][6]. Moreover, if the fitness sequence follows a recursive formula, the model becomes a recursive fitness model (RFM). This is a new type of random recursive tree on which little research has been done. As named, a RFM assigns a fitness value to a new node as output of a function of the node's father's fitness value. Hence, the fitness sequence is not fixed at the start, but follows a recursive process. Obviously, both of PAM and Bianconi-Barabási Model (BBM) have a time-independent fitness choice, but one can also make the fitness sequence vary over time. Fontanari and Cipriani [7] first introduced the idea of time-dependent fitness sequence. That means for a specific node, the fitness value may change at two different times. Finally, these are so-called random graphs with time-varying fitness. In Section 2 and 4, it is discussed whether the structure is well-defined and if it influences the power-law. There is a huge quantity of random trees in this category, but we enumerate only one special case for research: Preferential Attachment Inverse Model (PAMinv). The PAMinv is invented from the PAM. The fitness value of a node comes from the expectation of its degree sequence in PAM. By definition in Section 2, one could see that the fitness value is not deterministic but varies by time t.

RFM and IM are new models, and IM is even first developed in this paper in order to make an interpolation of tail of degree sequence from PAM toward an exponential tail (RRT). At the same time, the fitness sequence and height of these models are also discussed in detail.

The paper is divided into four main sections except the introduction. In Section 2, definitions of these models are given and some propositions, especially on the tail exponents, are discussed mathematically. Then in Section 3, there are descriptions about the models' generating algorithm and details on the simulation of random trees. After that, in Section 4, the main results from the simulation are displayed and fully discussed, including some intuitive comparison among these models and their own features. Finally, there is a conclusion section to make a summary of all the results, and raise problems that still wait for further exploration.

Chapter 2

Mathematical Definition and Properties

In this section, we mainly introduce the definitions of the models mentioned above, their properties, and mathematical deduction.

2.1 Definitions of Models

Hereby we would like to first set up an algorithm to combine these models together. In particular, we aim to construct a time-evolving random tree branching from only one node with a degree of 2 (a self-loop). After that, each new node has to attach itself to one of the previous nodes except itself according to the probability (2.1).
The construction algorithm is as follows: $\mathcal{F}_t$ is the fitness vector, whose elements are fitness values of graph nodes at time t. G_t means the random graph at time t with its vertices set V_t and edge set E_t. $D_t(j)$ means the degree of vertex j at time t, i.e. the number of edges attached to vertex j. Meanwhile, for all $t \in \mathbb{N}$, the fitness function at time t, $f_t : V_t \to \mathbb{R}^*$, is non-negative.

Algorithm 1: Construction of General Random Trees

Initialization:;
$G_1 = (V_1, E_1) \leftarrow (\{1\},(1,1))$;
$\mathcal{F}_1 \leftarrow \{f_1(1)\}$;
Recursive loop:;
for $t > 1$ **do**
  $\mathcal{F}_t \leftarrow \{f_t(v)\}$ for all $v \in V_{t-1}$;
  node t attached to node $j \in V_{t-1}$ chosen with probability

$$\mathbb{P}(t \to j) = \frac{f_t(j) D_{t-1}(j)}{\sum_{v \in V_{t-1}} f_t(v) D_{t-1}(v)} \tag{2.1}$$

  ;
  $V_t \leftarrow V_{t-1} \cup \{t\}$;
  $E_t \leftarrow E_{t-1} \cup \{(t, j)\}$;
end

The random tree is born with one node with a self-loop, but we still call the model a tree, since the remaining branching process follows the tree definition: so it has no loops. The reason why it is compulsory to put a self-loop here is to prevent a single point with zero degree, and that could lead to no possibility to get a new attachment and therefore, the tree would end. Also, graphs given by the construction algorithm are well-defined, because each vertex is labelled by the step t of its birth, and

by assumption, the time-dependent variable

$$Z_t = \sum_{v \in V_{t-1}} f_t(v) D_{t-1}(v) \tag{2.2}$$

is always positive and increasing.
Obviously, we do not put any special restriction on the fitness functions. Different choices of fitness functions could lead to different models. Following are some special kinds of random tree models.

Definition 2.1.1. (Preferential Attachment Model) Given the construction Algorithm 1, the random tree is called a preferential attachment model if all the fitness functions are constantly equal to one.

In this case, the attachment probabilities become

$$\mathbb{P}(t \to j) = \frac{D_{t-1}(j)}{\sum_{v \in V_{t-1}} D_{t-1}(v)}. \tag{2.3}$$

That indicates that the new node chooses its attachment partner depending only on the partners' degree. This will lead to some interesting results that will be discussed in later parts.

Definition 2.1.2. (Random Recursive Tree) Given the construction Algorithm 1, the random tree is called a random recursive tree if the fitness functions are specially chosen as follows: the fitness of node i at time t is

$$f_t(i) = \frac{1}{D_{t-1}(i)}. \tag{2.4}$$

Normally, one could see another definition of RRT: a random recursive tree is a rooted tree chosen uniformly at random from the trees with a given number of vertices. It is not hard to notice that the two definitions are equivalent, if we apply the fitness back to (2.1):

$$\mathbb{P}(t \to j) = \frac{f_t(j) D_{t-1}(j)}{\sum_{v \in V_{t-1}} f_t(v) D_{t-1}(v)} = \frac{D_{t-1}(j)/D_{t-1}(j)}{\sum_{v \in V_{t-1}} D_{t-1}(v)/D_{t-1}(j)} = \frac{1}{t-1}. \tag{2.5}$$

This means that the choice of the attachment is uniform at random as the second definition says.

Definition 2.1.3. (Bianconi-Barabási Model) Given the construction Algorithm 1, the random tree is called a Bianconi-Barabási model if the fitness functions are i.i.d. random variables sampled from some fixed distribution, i.e. given some probability distribution ν on $[0, +\infty)$, for any $i \leq t$,

$$f_t(i) = f(i) \sim \nu, \quad i.i.d. \tag{2.6}$$

The Bianconi-Barabási model (BBM) is an important modification of the PAM, and becomes more applicable in dynamic areas. A sequence of time-independent fitness functions are introduced to work as weights in the attachment choosing process. Uniform, exponential, and some other distributions ν have been tested by Barabási [4], and different choices of distribution affect the tail exponent of the model's degree sequence. We would now like to extend the BBM as follows.

Definition 2.1.4. (Recursive Fitness Model) Given the construction Algorithm 1, the random tree is called a recursive fitness tree (RFM) if the fitness functions are chosen by the following rule: for time t and node i ($i \leq t$), there is a parent function $\pi : V_t \to V_t$, and $\pi(i)$ gives the parent of node i in a graph. Then there exists some function $F : \mathbb{R}^* \to \mathbb{R}^*$, such that,

$$f(t) = F(f(\pi(t))), \quad f(1) = 1. \tag{2.7}$$

This is a new model that cannot be described by any formulas above. The RFM shares a feature with BBM that the fitness does not vary with time, which means if a fitness value is assigned to a node, it will never change, even though the node's degree might change. However, the RFM is slightly different from BBM, since the fitness comes from a recursive process rather than a fixed distribution.(Normally, the recursive process cannot be described by some distribution.)

Some RFM examples can be listed here.

- Plus-1 Model: one of the simplest RFM. The fitness of a new node is one more than its father's fitness, i.e. The recursive function is $F(x) = x + 1$.
- Plus-i Model: its recursive function is $F(x) = x + \pi(t)$.
- Plus-i^2 Model: its recursive function is $F(x) = x + (\pi(t))^2$.
- Times-2 Model: its recursive function is $F(x) = 2x$.
- Times-i Model: its recursive function is $F(x) = \pi(t)x$.

Those are five models with fitness that comes from a recursive formula. One could set up more complex RFMs according to different recursive functions.

Definition 2.1.5. (Inverse Model) Given the construction Algorithm 1, the random tree is called an Inverse Model (IM), if the fitness functions are specially chosen as follows: for time t and node $i(i \leq t)$

$$f_t(i) = \frac{1}{\mathbb{E}[d_{t-1}(i)]} \tag{2.8}$$

Here d means the degree sequence of the benchmark model (PAM).

The expectation of the degree sequence of the benchmark model (PAM) can be computed[15], since these models have clear fitness-independent attachment choosing process. Thus, the special model can be defined as Preferential Attachment Inverse Model (PAMinv). This model is introduced in order to be compared with PAM. In fact, the attachment probabilities are on average the same as that of PAM.

Generally, random trees with fitness can be divided into 3 main categories according to the choice of fitness sequence: A fixed distribution leads to the BBM, where RRT and PAM can be two special cases of this style. A recursive process leads to the RFM, where the Plus-1 Model can be a special case. Finally, the fitness sequence can also update itself time-dependently, when the IMs can be such a kind. In this paper, we mainly focus on the properties of these kinds of models, especially the power-law of the degree sequence. Thus, such representative models mentioned above will be taken into consideration, although there are still plenty of special models of random trees.

2.2 Models and Notations

Here is a summary of all the models mentioned in this paper and a list of symbols used in the paper.

Table 2.1: MODELS in the paper

Benchmark Model	Preferential Attachment Model (PAM)
	Random Recursive Tree (RRT)
Bianconi-Barabási Model (BBM)	Bianconi-Barabási Model with exponential(2) distribution (BBM-exp2)
	Bianconi-Barabási Model with standard uniform distribution (BB-Muni)
Recursive Fitness Model (RFM)	Plus-1 Model (Plus-1)
	Plus-i Model (Plus-i)
	Plus-i^2 Model (Plus-i^2)
	Times-2 Model (Times-2)
	Times-i Model (Times-i)
Inverse Model (IM)	Preferential Attachment Inverse Model (PAMinv)

Table 2.1 summarizes all the random trees in the paper, and the words in brackets refer to the short names of models.

Table 2.2: NOTATIONS

Symbol	Meaning
$i, j, ...$	Nodes of some random graph
(i, j)	The edge between nodes i and j of some random graph
$D_t(i)$	The degree of node i at iteration time t, $i = 1, 2, ..., t$
$\mathcal{F}_t$	The fitness vector with elements of fitness values of graph nodes at time t
G_t	The random graph at time t
V_t	The vertices set of random graph G_t
E_t	The edges set of random graph G_t
f_t	The fitness function at time t
$f_t(i)$	The fitness value of node i at time t
$\pi(i)$	The parent function, indicating the parent of node i in some random graph
$F(x)$	The recursive function of some RFM
$N_k(t)$	The number of nodes with degree k at time t in some random graph
$N_{\geq k}(t)$	The number of nodes with degree no less than k at time t in some random graph
$diam(G)$	The diameter of a random graph G
$dist(i, j)$	The distance between nodes i and j
H_t	The height of some random graph at time t
PAM_t	Sigma algebra generated by PAM at time t
RFM_t	Sigma algebra generated by RFM at time t
RRT_t	Sigma algebra generated by RRT at time t

2.3 Properties of the Models

We are highly interested in the tail exponent of random trees. Some of the results are well-known and proven, but others are still left to detect. In this section, we begin to list the known theorems and then mathematically prove the remaining properties on the power-law of the degree sequence.

2.3.1 Preferential Attachment Model

Theorem 2.3.1. *(Power-law of PAM) For the Preferential Attachment Model defined by Definition 2.1.1, its degree sequence follows a power-law with index -3,i.e.*

$$\lim_{t\to\infty} \frac{N_k(t)}{t} = k^{-3} \tag{2.9}$$

where $N_k(t)$ means the number of nodes with degree k at time t.

A number of scientists, such as Barabási [4] and van der Hofstad [15], have proved the theorem in different ways. This theorem indicates that the degree sequence of PAMs undoubtedly has a heavy tail. For the standard PAM, the limit of expectation of its degree sequence can be mathematically estimated in Theorem 2.3.3, but a pre-knowledge has to be mentioned first.

Theorem 2.3.2. *(Degree of PAM) The total degree of standard PAM at time t equals $2t$.*

Proof. This theorem is a quite intuitive result and easy to prove. Obviously, a standard PAM at time t has t nodes, and each node except the first and the last one has an attachment to its parent and another attachment to its child. For the first node, it has a self-loop, and an attachment with its child. For the last node, it has only one attachment with its parent. Hence, totally the PAM at time t has a degree of $2(t-2)+3+1 = 2t$. □

Theorem 2.3.3. *(Expectation of PAM degree sequence) For the standard PAM, as $t \to \infty$ and a large i,*

$$\mathbb{E}[D_t(i)] \sim (\frac{t}{i})^{\frac{1}{2}} \tag{2.10}$$

The proof is based on the Theorem 8.1 of van der Hofstad [15], and this is a special case with $m = 1$ and $\delta = 0$.

Proof. (follows the proof by van der Hofstad) First, split the expectation and we can compute that for $t \geq i$,

$$\mathbb{E}[D_{t+1}(i)|D_t(i)] = D_t(i) + \mathbb{E}[D_{t+1}(i)) - D_t(i)|D_t(i)] \tag{2.11}$$

$$= D_t(i) + \frac{D_t(i)}{2t+1} \text{(Theorem 2.3.2)} \tag{2.12}$$

$$= D_t(i)\frac{2t+2}{2t+1} \tag{2.13}$$

$$= D_t(i)\frac{2(t+1)}{2t+1}. \tag{2.14}$$

Also, we can compute the following value: for each $i \in \mathbb{Z}^+$

$$\mathbb{E}[D_i(i)] = 1 + \frac{1}{2(i-1)+1} = \frac{2i}{2i-1}. \tag{2.15}$$

Then, it is enough to set up a non-negative martingale over t

$$M_t(i) = D_t(i) \prod_{s=i-1}^{t-1} \frac{2s+1}{2(s+1)}. \tag{2.16}$$

$M_t(i)$ is a martingale over time t because of the following 3 points. First, we denote the sigma algebra generated by the PAM tree configuration at time t as PAM_t. By definition, $M_t(i)$ is PAM_t measurable, because variable $D_t(i) \in PAM_t$ is in the sigma algebra. Second, it follows the conditional expectation property, i.e.

$$\begin{aligned}
\mathbb{E}[M_{t+1}(i)|M_t(i)] &= \mathbb{E}[M_{t+1}(i)|D_t(i)] \\
&= \mathbb{E}[D_{t+1}(i)|D_t(i)] \prod_{s=i-1}^{t} \frac{2s+1}{2(s+1)} \\
&= D_t(i)\frac{2(t+1)}{2t+1} \cdot \prod_{s=i-1}^{t} \frac{2s+1}{2(s+1)} \text{(by equation (2.14))} \\
&= D_t(i) \prod_{s=i-1}^{t-1} \frac{2s+1}{2(s+1)} = M_t(i).
\end{aligned}$$

Third, the expectation of $M_t(i)$ is finite. Moreover, apply equation (2.15), we have

$$\mathbb{E}[M_t(i)] = \mathbb{E}[\mathbb{E}[M_t(i)|D_i(i)]] = \mathbb{E}[D_i(i)]\frac{2(i-1)+1}{2i} = 1 < \infty. \tag{2.17}$$

Then, it is important to compute the constant part of the martingale $M_t(i)$. By using the definition of Gamma function, we have

$$\prod_{s=i-1}^{t-1} \frac{2s+1}{2(s+1)} = \prod_{s=i-1}^{t-1} \frac{s+1/2}{s+1} = \frac{\Gamma(t+\frac{1}{2})\Gamma(i)}{\Gamma(t+1)\Gamma(i-\frac{1}{2})}. \tag{2.18}$$

Apply equation (2.18) back to $M_i(t)$ (2.16), and take the expectation on both side, it is not hard to get the following result:

$$\mathbb{E}[D_t(i)] = \frac{\Gamma(t+1)\Gamma(i-\frac{1}{2})}{\Gamma(t+\frac{1}{2})\Gamma(i)}. \tag{2.19}$$

We will use the Stirling formula of Gamma function [1]. For any real number $x > 0$ (actually it is also true for a complex number z with $R(z) > 0$), we have

$$\ln\Gamma(x) = (x-\frac{1}{2})\ln x - x + \frac{\ln 2\pi}{2} + 2\int_0^\infty \frac{\arctan\frac{y}{x}}{e^{2\pi y}-1}dy \tag{2.20}$$

$$\Gamma(x) = \sqrt{\frac{2\pi}{x}}(\frac{x}{e})^x(1+O(\frac{1}{x})). \tag{2.21}$$

By using (2.21), we have

$$\frac{\Gamma(t+a)}{\Gamma(t)} = \frac{\sqrt{\frac{2\pi}{t+a}}(\frac{t+a}{e})^{t+a}(1+O(\frac{1}{t+a}))}{\sqrt{\frac{2\pi}{t}}(\frac{t}{e})^t(1+O(\frac{1}{t}))} = \frac{(t+a)^a(t+a)^{t-\frac{1}{2}}}{t^{t-\frac{1}{2}}}(1+O(\frac{1}{t})) = t^a(1+O(\frac{1}{t})). \tag{2.22}$$

Applying (2.22) back to (2.19), we can get the result mentioned by van der Hofstad: the expectation of $D_t(i)/t^{\frac{1}{2}}$ converges almost surely to $\frac{\Gamma(i-\frac{1}{2})}{\Gamma(i)}$. If we apply (2.22) again, we can get the final result for case $m = 1$

$$\mathbb{E}[D_t(i)] \sim (\frac{t}{i})^{\frac{1}{2}}. \tag{2.23}$$

□

Again, we are also interested in the height of PAM. Dommers and van der Hofstad [8] have proven that the diameter of PAM is $O(\log t)$. The definition of a diameter of a graph and the theorem are as follows.

Definition 2.3.1. (Diameter of graph) The diameter of a graph G is defined as follows:

$$diam(G) = \max_{i,j\in V}\{dist(i,j)|dist(i,j) < \infty\} \tag{2.24}$$

where $dist(i,j)$ denotes the graph distance between node i and j.

Definition 2.3.2. The height of a graph G is defined as follows:

$$H(G) = \max_{j\in V}\{dist(i_0,j)|dist(i_0,j) < \infty, i_0 \text{ is the origin}\} \tag{2.25}$$

With Definition 2.3.1, the upper bound of the diameter of PAM can be estimated in the following theorem.

Theorem 2.3.4. *(Diameter of PAM)(Dommers et al., 2009) There exists a constant c_0, such that with high probability, the diameter of standard PAM at time t is at most $c_0 \log t$.*

The proof can be referred to Dommers and van der Hofstad's paper [8]. By Definition 2.3.1 and 2.3.2, diameter includes height of a graph, which means that the height of PAM increases by a logarithm function.

2.3.2 Bianconi-Barabási Model

Since PAM is a special case of the BBM, properties of Bianconi-Barabási Models can be extended from the PAM's features. However, the introduction of fitness sequence brings some differences. According to the behavior of degree sequence, the BBMs can be divided into two main phases: The innovation-pays-off phase and the fit-get-richer phase. First, one BBM follows the innovation-pays-off phase if the

fitness may play a role overwhelming the degree sequence in connecting the new node, which would lead to condensation [5][4]. Condensation is another important phenomenon of random trees, but that is not in the scope of this paper. One could refer to the theorems and conjectures in other papers [4][7]. Second, the fit-get-richer phase describes the most cases of BBMs, including the PAM: a node with a specific fitness value will get more attachments if it already has many edges. The following theorem [4] shows the conditions and tail exponents of BBMs in fit-get-richer phase.

Theorem 2.3.5. *(Fit-get-richer Phase) Suppose Ω to be the set of fitness values from which f can be chosen. With all the assumptions of BBM, let $T = |\Omega|$ and $1 \leq T \leq +\infty$ indicate the size of the fitness function set, $h = \sup_{1\leq t\leq T} f(t) < +\infty$, $q(t) \in Q$ is the probability of $f(t)$, where Q admits a strictly positive continuous density on $(0, h)$. If*

$$\sum_{t=1}^{T} \frac{f(t)q(t)}{h - f(t)} > 1$$

then, the BBM falls in the fit-get-richer phase with a tail exponent of $\lambda_0 f^{-1}(t)$, where λ_0 is the largest solution of the following equation

$$\sum_{t=1}^{T} \frac{f(t)q(t)}{\lambda_0 - f(t)} = 1 \tag{2.26}$$

Theorem 2.3.5 gives a mathematical estimation of the tail exponent of some BBM, which varies over different choices of fitness distribution ν. It is easy to see that BBM is fit-get-richer if Q is a single element or uniform distribution. The proof is in detail given by Mr. Borgs and his colleagues [5] by using Pólya urn process [11]. Here we state the definition of the process, which will be useful for the latter proof in Section 4.5.

Definition 2.3.3. (Pólya Urn Process) There are $r < +\infty$ bins. Each bin $i \leq r$ is assigned a fixed activity a_i, $0 \leq a_i < +\infty$. For $n \geq 0$, let

$$X_n = (X_{n,1}, \dots, X_{n,r})$$

where $X_{n,i}$ is the number of balls in bin i at time n. The initial load is given by X_0, which may be random or deterministic. The i^{th} bin also has a random vector $\xi_i = (\xi_{i,1}, \dots, \xi_{i,r})$ with independent integer elements $\xi_{i,j}$. The process is defined as follows. At time n, we pick one bin. Bin i is chosen with probability proportional to $a_i X_{n-1,i}$, a is some fixed sequence with r terms. If bin i is picked, we draw an independent copy ξ_i^n of ξ_i and update $\{X_n\}_{n\geq 0}$ according to

$$X_n = X_{n-1} + \xi_i^n$$

Normally, balls are added or at least not withdrawn from the system, and this refers to the non-extinction condition:

$$\xi_{ij} \geq 0, \quad i \neq j \tag{2.27}$$

$$\sum_{j=1}^{r} \xi_{ij} \geq 0. \tag{2.28}$$

In Pólya urn process, a transformation matrix A plays an important role, and the matrix A describes the expected operation among each bin, i.e. the transformation from X_n to X_{n+1} for any $n > 0$.

Definition 2.3.4. (Matrix A given by Janson 2004) Given Definition 2.3.3, the transformation matrix A is a $r \times r$ square matrix

$$A := \{a_j \mathbb{E}[\xi_{ji}]\}_{i,j=1}^{r}.$$

In our case, r mainly denotes the largest degree number plus 1 in our models, i.e. $r = \max(D(j)) + 1$. Obviously, if for all $j = 1, 2, \dots, r$, $a_j = 1$, then $A_{ij} = \mathbb{E}[\xi_{ji}]$ and the j^{th} column of A is the expected change when balls in bin j is drawn. With conditions (2.27),(2.28) and Perron-Frobenius Theory, A has a largest real eigenvalue λ_1, such that every other eigenvalue λ satisfies: $Re\lambda < \lambda_1$. The largest eigenvalue and its corresponding eigenvector play a central role in the following theorem.

Theorem 2.3.6. *(Janson 2004) With assumption of (A1)-(A6) and the condition on essential non-extinction, $\frac{X_n}{n} \to \lambda_1 v_1$, where λ_1 and v_1 denote the largest eigenvalue and its corresponding eigenvector of branching matrix A related to process X_n.*

The proof of this theorem is extremely complex, and can be found in Janson's paper [11]. After that, Borgs and his colleagues use this theorem to prove the asymptotic properties of Bianconi-Barabási Model [5].

2.3.3 Recursive Fitness Model

Similar to the PAM, there are two steps to get the power-law of the RFM degree sequence. First, it is essential to prove that for each node, its degree should converge toward the expectation value. Then, we can somehow compute the expectation of the degree sequence. So the first step is the following theorem. The proof is mainly based on van der Hofstad chapter 8 [15]. We first state Hoeffding's inequality [14], and it is used to prove the theorem, but we do not prove it here.

Lemma 2.3.7. *(Hoeffding's Inequality) Let $X_1, \dots, X_n$ be independent random variables bounded by the interval $[a, b]$. We define the empirical mean of these variables by*

$$\bar{X}_n = \frac{1}{n}(X_1 + \cdots + X_n).$$

Then we have

$$\mathbb{P}(\bar{X}_n - \mathbb{E}[\bar{X}_n] \geq t) \leq e^{-\frac{2t^2}{(b-a)^2}}$$

where $\mathbb{E}[\bar{X}_n]$ denotes the expectation of $\bar{X}_n$. The inequalities can be also stated in terms of the sum $S_n = X_1 + \cdots + X_n$.

$$\mathbb{P}(S_n - \mathbb{E}[S_n] \geq t) \leq e^{-\frac{2t^2}{n^2(b-a)^2}}$$

Theorem 2.3.8. *(Convergence of RFM) In any RFM, for any constant $C > \sqrt{8}$, as $t \to \infty$,*

$$\mathbb{P}(\max_k |N_k(t) - \mathbb{E}[N_k(t)]| \geq C\sqrt{t \log t}) = o(1) \tag{2.29}$$

Proof. First of all, we denote the sigma algebra generated by the RFM tree configuration at time t as RFM_t.
Notice that when $k > t + 1$, we have $N_k(t) = 0$. Thus,

$$\mathbb{P}(\max_k |N_k(t) - \mathbb{E}[N_k(t)]| \geq C\sqrt{t \log t}) = \mathbb{P}(\max_{k \leq t+1} |N_k(t) - \mathbb{E}[N_k(t)]| \geq C\sqrt{t \log t}) \tag{2.30}$$

$$\leq \sum_{k=1}^{t+1} \mathbb{P}(|N_k(t) - \mathbb{E}[N_k(t)]| \geq C\sqrt{t \log t}) \tag{2.31}$$

Then, it is enough to prove that for any $k \leq t + 1$,

$$\mathbb{P}(|N_k(t) - \mathbb{E}[N_k(t)]| \geq C\sqrt{t \log t}) = o(t^{-1}) \tag{2.32}$$

We establish a Doob Martingale, for $n = 0, \dots, t$:

$$M_n = \mathbb{E}[N_k(t)|RFM_n]. \tag{2.33}$$

First, by definition, there is always a number $N_k \geq 0$ denoting the number of vertices with degree k at time t for each model of $n = 0, \dots, t$, which means $N_k(t)$ is measurable with respect to RFM_n. Then it is essential to prove the finite expectation.

$$\mathbb{E}[|M_n|] = \mathbb{E}[M_n] = \mathbb{E}[\mathbb{E}[N_k(t)|RFM_n]] = \mathbb{E}[N_k(t)] \leq t < \infty. \tag{2.34}$$

Notice that the RFM_{n+1} tree is obtained from the RFM_n model, which means that RFM_n is a subset of RFM_{n+1}, we have

$$\mathbb{E}[M_{n+1}|RFM_n] = \mathbb{E}[\mathbb{E}[N_k(t)|RFM_{n+1}]|RFM_n] \tag{2.35}$$
$$= \mathbb{E}[N_k(t)|RFM_n] \text{ (Tower Property)} \tag{2.36}$$
$$= M_n \tag{2.37}$$

By (2.34) and (2.37), we conclude that $\{M_n\}_{n=0}^t$ is a martingale with respect to the models $\{RFM_n\}_{n=0}^t$.

The above was the core part of the proof, and the key point is the establishment of the martingale M_n.

Then, we can again get

$$N_k(t) - \mathbb{E}[N_k(t)] = M_t - M_0 \tag{2.38}$$

And due to the branching process of RFM model, we have the inequality: for $n = 1, \dots, t$

$$|M_n - M_{n-1}| \leq 2 \tag{2.39}$$

Then, Hoeffding's inequality Lemma 2.3.7 can be applied to (2.32), for any $c > 0$

$$\mathbb{P}(|N_k(t) - \mathbb{E}[N_k(t)]| \geq c) \leq 2e^{-\frac{c^2}{8t}} \tag{2.40}$$

Take $c = C\sqrt{t \log t}$ in (2.40) and $C^2 > 8$, we get

$$\mathbb{P}(|N_k(t) - \mathbb{E}[N_k(t)]| \geq C\sqrt{t \log t}) \leq 2e^{-\log t \frac{C^2}{8}} = 2t^{-\frac{C^2}{8}} = o(t^{-1}) \tag{2.41}$$

□

When it comes to the second step to compute the expectation of degree sequence, Hofstad manages to find a time-independent sequence p_k and proves the convergence of $\mathbb{E}[N_k(t)]$ toward p_k over time iteration. However, for the RFM, a major problem is that we do not have a method to calculate the probability distribution of the fitness sequence because of the recursive process of the fitness. Although it is impossible to get the analytical solution, the numerical solution in the Results part may show some directions.

2.3.4 Random Recursive Tree

Similar to the PAM, RRT is a well-known model with plenty of features. We first give a formula of the expectation of the degree sequence, and then state the tail behavior.

Theorem 2.3.9. *In the RRT model with only one root vertex, we set $D_i(t)$ as the degree of node i at iteration time t, then its expectation is a harmonic number, or more precisely for large t*

$$\mathbb{E}[D_i(t)] = 1 + \sum_{j=i}^{t-1} \frac{1}{j} \sim \ln \frac{t-1}{i}. \tag{2.42}$$

Proof. First of all, we notice the following relation

$$D_i(t+1) = \begin{cases} D_i(t) + 1, & w.p. \quad \frac{1}{t} \\ D_i(t), & w.p. \quad 1 - \frac{1}{t} \end{cases}. \tag{2.43}$$

Here, w.p. means 'with probability'. Then we can compute the following conditional expectation by using (2.43):

$$\mathbb{E}[D_i(t+1)|D_i(t)] = D_i(t) + \mathbb{E}[D_i(t+1) - D_i(t)|D_i(t)] \tag{2.44}$$
$$= D_i(t) + (1 \cdot \frac{1}{t} + 0) \tag{2.45}$$
$$= D_i(t) + \frac{1}{t}. \tag{2.46}$$

After that, apply the conditional property to (2.46), we get

$$\mathbb{E}[D_i(t+1)|D_i(t-1)] = \mathbb{E}[\mathbb{E}[D_i(t+1)|D_i(t)]|D_i(t-1)] \tag{2.47}$$

$$= \mathbb{E}[D_i(t)|D_i(t-1)] + \frac{1}{t} \tag{2.48}$$

$$= D_i(t-1) + \frac{1}{t-1} + \frac{1}{t}. \tag{2.49}$$

Continue to use (2.49) for $t-1$ times on $\mathbb{E}[D_i(t)|D_i(t-1)]$, we get part of our result:

$$\mathbb{E}[D_i(t)] = \mathbb{E}[D_i(t)|D_i(i)] = D_i(i) + \sum_{j=i}^{t-1} \frac{1}{j} = 1 + \sum_{j=i}^{t-1} \frac{1}{j}. \tag{2.50}$$

Here by definition, $D_i(i) = 1$. Meanwhile, according to the logarithm increments property of Harmonic series, we have

$$1 + \sum_{j=i}^{t-1} \frac{1}{j} \sim \ln \frac{t-1}{i}. \tag{2.51}$$

□

Then, we define $N_{\geq k}(t) = \sum_{i=1}^{t} \mathbb{1}_{D_i(t)\geq k}$, which means the number of vertices with degree at least k at time t. Then, with (2.51) we have

$$N_{\geq k}(t) \sim \sum_{i=1}^{t} \mathbb{1}_{\mathbb{E}[D_i(t)]\geq k} \sim \sum_{i=1}^{t} \mathbb{1}_{\ln \frac{t-1}{i} \geq k} = \mathbb{1}_{i \leq (t-1)e^{1-k}} \sim te^{1-k}. \tag{2.52}$$

Above is a short explanation of the exponential tail of the RRT model. To prove it mathematically, we need the following definition and theorems.

First of all, define a random proportion of vertices with degree k at time t as follows:

$$P_k(t) = \frac{1}{t} \sum_{i=1}^{t} \mathbb{1}_{D_i(t)=k}. \tag{2.53}$$

Then, we immediately have the following equations

$$N_k(t) = \sum_{i=1}^{t} \mathbb{1}_{D_i(t)=k} = tP_k(t), \tag{2.54}$$

$$\mathbb{E}[N_k(t)] = \mathbb{E}[tP_k(t)]. \tag{2.55}$$

Meanwhile, we write the sigma algebra generated by RRT model at time t as RRT_t.

With these definitions, we want to prove the following theorem about the degree sequence in RRT.

Theorem 2.3.10. *For RRT model, there exists a unique sequence $\{p_k\}_{k=1}^{\infty}$ with respect to degree k, and a constant $C > 0$, such that, as $t \to \infty$*

$$\mathbb{P}(\max_k |P_k(t) - p_k| \geq C\sqrt{\frac{\ln t}{t}}) = o(1) \tag{2.56}$$

As a branching process, RRT model still has the special Proposition 2.3.8, and by the relation (2.54) and (2.55), to prove Theorem 2.3.10, it is enough to prove the following proposition:

Proposition 2.3.11. *In the RRT model, there exists a constant C such that for all $t \geq 1$ and all $k \in \mathbb{N}$,*

$$|\mathbb{E}[N_k(t)] - tp_k| \leq C \tag{2.57}$$

Then, combine Proposition 2.3.8 and 2.3.11 together, we can easily get Theorem 2.3.10. But at the beginning, suppose that Proposition 2.3.11 is true, which means that $\mathbb{E}[N_k(t)] \approx tp_k$, we first prove Theorem 2.3.10.

Proof. (Theorem 2.3.10) Again, we split the conditional expectation, which makes a good preparation for induction.

$$\mathbb{E}[N_k(t+1)|RRT_t] = N_k(t) + \mathbb{E}[N_k(t+1) - N_k(t)|RRT_t] \tag{2.58}$$

By definition, $N_k(t+1)$ could be larger, smaller, or equal to $N_k(t)$. Thus, the problem is again split into three different conditions:

- $k > 1$. The incoming vertex attaches to a node with degree $k-1$, which means its degree rises to k at time $t+1$, and $N_k(t+1)$ grows by one. This happens with probability $\frac{N_{k-1}(t)}{t}$.
- $k > 1$. The incoming vertex attaches to a node with degree k, which means its degree rises to $k+1$ at time $t+1$, and $N_k(t+1)$ decreases by one. This happens with probability $\frac{N_k(t)}{t}$.
- $k = 1$. $N_k(t+1)$ will grow by one, except the case that the vertex attached to the new node also has a degree 1. This happens with probability $1 - \frac{N_1(t)}{t}$.

We can combine the three conditions together. Setting $p_0 = 1$, then

$$\mathbb{E}[N_k(t+1) - N_k(t)|RRT_t] = \mathbb{1}_{k>1}\frac{N_{k-1}(t) - N_k(t)}{t} + \mathbb{1}_{k=1}(1 - \frac{N_1(t)}{t}) \tag{2.59}$$

$$= \frac{N_{k-1}(t) - N_k(t)}{t}. \tag{2.60}$$

Then, apply the tower property of conditional expectation to (2.60),

$$\mathbb{E}[N_k(t+1)] = \mathbb{E}[N_k(t)] + \mathbb{E}[N_k(t+1) - N_k(t)] \tag{2.61}$$

$$= \mathbb{E}[N_k(t)] + \mathbb{E}[\mathbb{E}[N_k(t+1) - N_k(t)|RRT_t]] \tag{2.62}$$

$$= \mathbb{E}[N_k(t)] + \mathbb{E}[\frac{N_{k-1}(t) - N_k(t)}{t}] \tag{2.63}$$

$$= \frac{t-1}{t}\mathbb{E}[N_k(t)] + \frac{1}{t}\mathbb{E}[N_{k-1}(t)] \tag{2.64}$$

And with assumption of p_k, we can get the following equation with respect to (2.64)

$$p_k = -p_k + p_{k-1} \tag{2.65}$$

Then, we obtain the recursive formula defining the sequence $\{p_k\}_{k=1}^{\infty}$. With the assumption that $p_0 = 1$, we have for any $k \in \mathbb{N}$

$$p_k = 2^{-k} \tag{2.66}$$

The recursive solution is unique. □

After that, we can turn to the proof of the Proposition 2.3.11.

Proof. (Proposition 2.3.11) First define a sequence of variables ϵ_k:

$$\epsilon_k(t) = \mathbb{E}[N_k(t)] - tp_k, \quad k \geq 1. \tag{2.67}$$

Then, Proposition 2.3.11 is equivalent to proving that there exists a constant C such that

$$\max_k |\epsilon_k(t)| \leq C. \tag{2.68}$$

To prove (2.68), induction in t is performed.

First of all, according to (2.65), we have the equation below

$$(t+1)p_k = tp_k + p_k = tp_k - p_k + p_{k-1} \tag{2.69}$$

Then, make a subtraction between (2.64) and (2.69), we get

$$\epsilon_k(t+1) = \frac{t-1}{t}\epsilon_k(t) + \frac{1}{t}\epsilon_{k-1}(t) \tag{2.70}$$

When $t = 1$, by definition, the RRT model starts from one node with a self-loop, so

$$\mathbb{E}[N_k(1)] = \mathbb{1}_{k=2} = 2\mathbb{P}(k=2) \leq 2 \tag{2.71}$$

And we know that $0 < p_k \leq 1$, so we can get the initialization that at $t = 1$, and for all $k \geq 1$

$$|\epsilon_k(1)| = |\mathbb{E}[N_k(1)] - p_k| \leq \max\{\mathbb{E}[N_k(1)], p_k\} \leq 2 \tag{2.72}$$

This means that claim (2.57) is true when $C \geq 2$ for $t = 1$.

Then, suppose the claim (2.57) is true for any $t \leq T$, we want to prove that it is still true for $t = T+1$. Again apply triangular inequality to (2.70), we get for $T \geq 1$

$$|\epsilon_k(T+1)| = |\frac{T-1}{T}\epsilon_k(T) + \frac{1}{T}\epsilon_{k-1}(T)| \tag{2.73}$$

$$\leq |\frac{T-1}{T}\epsilon_k(T)| + |\frac{1}{T}\epsilon_{k-1}(T)| \tag{2.74}$$

$$\leq \frac{T-1}{T} \cdot C + \frac{1}{T} \cdot C \tag{2.75}$$

$$= C \tag{2.76}$$

So by induction, for any $t > 0$, and $k \in \mathbb{N}$, the claim (2.57) is true. □

Then, combining the two propositions together, we can get Theorem 2.3.10. After that, we can say mathematically for RRT model, the expectation of the number of vertices with degree k at time t has a light tail.

The next theorem by Devroye in 2012 [10] decribes the convergence of height of RRT.

Theorem 2.3.12. *(Height of RRT) Let H_t denote the height of RRT at time t, then*

$$\frac{H_t}{\log t} \to e \quad a.s. \tag{2.77}$$

Chapter 3

Model Simulation

In this section, the simulation process of models above is introduced. According to the definitions of these random trees and pseudo code 1, it becomes key points to establish the fitness sequence. Python is used to make the simulation, and codes are attached in the appendix. It is clear that the process is divided into three parts: initialization, loops, and visualization. We mainly focus on details of the loops in this section. Additionally, intuitive images of random tree models are also shown in this section. Most images are composed of 1000 vertices colored red and several edges colored blue. A thousand could be a proper quantity, because more nodes are too crowded in an image, and fewer nodes cannot show well the scale-free trend.

First, some packages are used in the simulation and visualization process. Numpy is used for handling data, Matplotlib is used for visualization, Seaborn is used to generate empirical density curves, Collections is used for counting, Math and Scipy can be used for complex computation, Random is used to generate random numbers from some specific distribution, and finally Networkx is used to build the graph. Then, some important variables in the program are also introduced in advance. In all programs in the appendix, T denotes the largest iteration steps, deg denotes the degree vector based on the nodes of random graph at time t, fit denotes the fitness vector based on the nodes of random graph at time t, $prob$ denotes the vertices' weight vector which indicates the possibility of each node to get a new edge, $index$ gives the node to which the new vertex attaches itself, and G means the whole graph. By using these variables, we can describe the models above.

3.1 Selection Function

One of the key points of the simulation procedure can be the choice of new edges. We write a single file A in the appendix with 2 selection functions inside. Both of the functions aim to select a node as the parent of a new node to construct a new edge according to the probability sequence.

The first function in line 12 is named random-pick with two variables: a list, *sequence*, and an array, *probabilities*. Here the elements in *sequence* are nodes of the random tree, and elements of *probabilities* are the probabilities of these nodes to be selected as the parent of a new node, i.e. if $i \in V_t$ is a node of some random tree, and p is its corresponding element in *probabilities*, then $p = f(i)D_i$. Let P denote the sum of the *probabilities* vector. We first pick a random value x uniformly from the interval $[0, P]$ and initialize a parameter, *cumulative probability*, as 0. In the loop, *item* is a variable to run through the list *sequence*, and meanwhile, *item-probability* spans the selection array *probabilities*. Then, the loop can be described as follows:

$$cumulative \quad probability = \sum_{j=1}^{item} f(j)D_j.$$

Once the *cumulative probability* is larger than the random value x, then the output is the *item* where the loop stops.

The second function in line 23 is named random-picks with the same variables. The function makes random selection in another way, where the choice function from the random package is used. We establish a table by traversing the elements in list *sequence* and array *probabilities*. As shown in line 24, for each element x in list *sequence*, there is a natural number $y \in probabilities$, which denotes denotes the multiplicity of $x \in sequence$ in the *table*. Then, by using choice function, we uniformly pick an element from the *table*. Compared to first function, there is a drawback of the algorithm that the *probabilities* vector should only contain positive integers.

3.2 Preferential Attachment Model

The program in Appendix B simulates the standard PAM with Definition 2.1.1. The figure 3.1 below provides an intuitive view of a PAM with 1000 vertices.

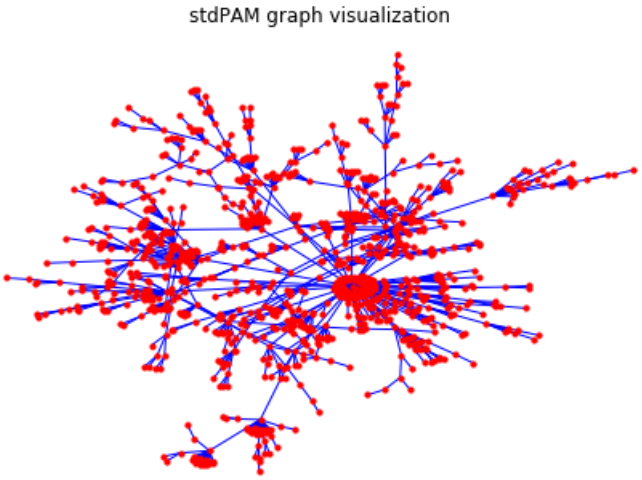


Figure 3.1: Simulation of standard Preferential Attachment Model

It is clear to find out an origin and some 'important' nodes with higher degrees than other nodes, which indicates a significant scale-free feature. A new node tends to attach itself to an old node with larger degree rather than an end of a branch. The figure comes from the following process.

In the program line 22, we first initialize the degree vector as zeros with a size of T. By algorithm 1, the degree of the first node should be 2 (self-loop). Then, we set up a blank graph. In line 30, the fitness sequence is initialized as ones, by Definition 2.1.1. After that, the probability sequence is also initialized as zeros, and the first element is again assigned as 2, the result of fitness times degree.
In the following part, line 41 defines some useful variables, including a vector ind, the distance between each node and the origin. Here, the maximun of ind shows the height of the random tree.

Then it turns to the loop in line 44, the most important part of the simulation procedure. First, it is essential to transfer the array $range(t+1)$ into a list in order to be inserted into the random-picks function. Then it follows a condition structure to find out the $index$ by using the random selection functions. We have to specially consider the $t = 0$ condition, simply because in this case, $prob[0]$ is a single element, and we should convert it into an array manually. Now that we have got the $index$, the selected parent of this loop. According to Algorithm 1, the degree of the chosen node increases by one, and the degree of the new node becomes one. Then in line 54 and 55, the new node and edge are added to the graph G. And the height mark ind of a new node is marked one more than its parent's value. Again in line 57, a sub-loop is conducted to update the $prob$ vector as the multiplication of degree values and corresponding fitness values.

The output of the program includes deg, the degree sequence of the random tree, fit, the fitness sequence of the random tree, and $max(ind)$, the height of the tree. It also outputs a number of figures, but those are left for the discussion section.

3.3 Random Recursive Tree

The program in Appendix C simulates the standard PAM with Definition 2.1.2. The figure 3.2 below provides an intuitive view of a RRT with 1000 vertices.

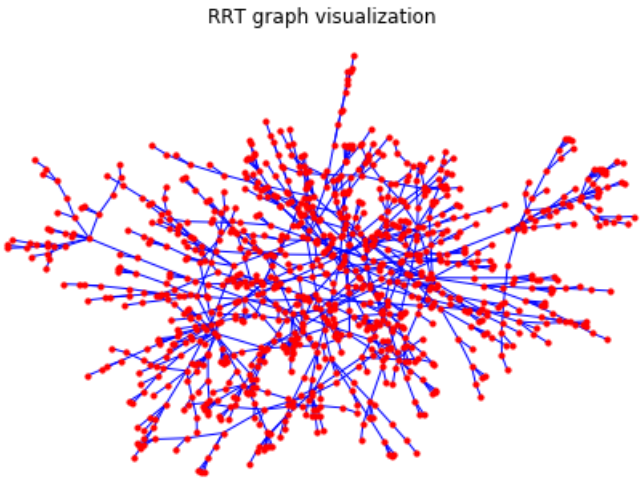


Figure 3.2: Simulation of Random Recursive Tree

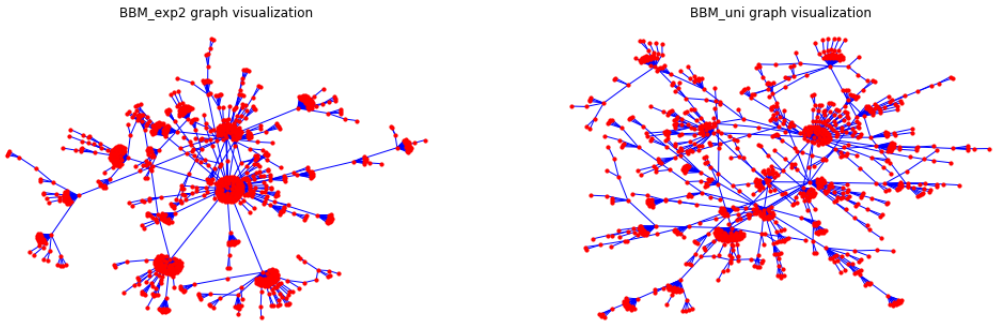
In the figure 3.2, it is easy to notice some branches structure, but it is hard to find the origin. That means RRT almost loses the scale-free feature. By definition, a new node has equal probability to attach it self to any old node. The figure is simulated from the following procedure.

In the program, it is clear that most steps are the same as PAM. Actually, by definition, the only difference among the random tree models in this paper is the generating method of fitness sequence. In line 29, the selection probability sequence $prob$ is defined as ones and deterministic, which means all the vertices share the same possibility to connect the new node. We write down the vector fit in line 33 in order to make a record of the fitness sequence and its trend. Comparison is made in the next section. It is worthy to mention that in RRT, the fitness values become decimals, so that we use a data type $float64$ for storage(line 33).

3.4 Bianconi-Barabási Model

The BBM is a kind of model with a fitness sequence following some fixed probability distribution ν. In this paper, we take the exponential distribution with parameter 2 and standard uniform distribution as examples of ν to show intuitive images and common features of the BBM. The simulation programs in Appendix D are almost the same, so we combine them together to make comments as follows.

As mentioned, the difference from the PAM is the fitness settings. According to Definition 2.1.3 and equation (2.6), $\nu_1 \sim Exp(2)$ and $\nu_2 \sim Uniform[0,1]$. Thus, in line 30, random function in the Numpy package is used to generate a sequence of random numbers with the probability distribution mentioned above. Then, it follows the same procedure as PAM.



(a) Simulation of Bianconi-Barabási Model with Exponential(2) fitness distribution

(b) Simulation of Bianconi-Barabási Model with standard uniform fitness distribution

Figure 3.3: Simulation of Bianconi-Barabási Model

Figure 3.3 gives an intuitive view of both BBMs with 1000 vertices. Compared to the standard PAM figure 3.1, the BBM with uniform distribution seems less scale-free, but the BBM with exponential distribution shows stronger scale-free phenomenon. Details are not discussed in this paper, but we can give some intuitive mathematical explanation. Suppose $f_{exp} \sim \nu_1$ and $f_{uni} \sim \nu_2$. According to Theorem 2.3.5, the tail exponent of BBM is proportional to $f^{-1}(t)$. Although f_{exp} and f_{uni} have the same expectation i.e. $\mathbb{E}[f_{exp}] = \mathbb{E}[f_{uni}] = \frac{1}{2}$, f_{exp} has more probability to choose a value close to zero than f_{uni}. By reverse, the tail exponent of BBM with exponential distribution tends to become larger than that of BBM with uniform distribution. That coincides with Bianconi and Barabási's results that the distribution strongly influences the power-law of degree sequence [4].

3.5 Recursive Fitness Model

Similar to the BBM, the RFM also represents a kind of random trees which have a recursive fitness sequence. In this part, we would like to take three special models as examples to show the simulation process, as well as the intuitive features. The first model is the Plus-1 Model. Given the Definition 2.1.4, the recursive function of Plus-1 Model is

$$f(t) = f(\pi(t)) + 1. \tag{3.1}$$

The second tree is the Plus-i Model, which has a recursive law as

$$f(t) = f(\pi(t)) + \pi(t). \tag{3.2}$$

And the last tree is the Times-2 Model, which has a recursive law as

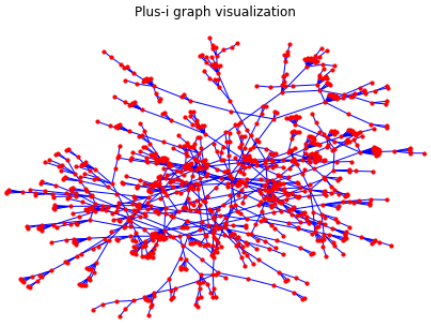
$$f(t) = 2f(\pi(t)). \tag{3.3}$$

The Plus-1 and Plus-i models are recursively additive processes, but the Times-2 model is a recursively multiplicative process. Also, Plus-1 and Times-2 models use a constant in the recursive formula, but the Plus-i model uses a variable instead. In a nutshell, the three models can be the representatives of RFM, and show some common features. Even though the recursive processes of the three models are quite simple, it is impossible to analytically find out a mathematical formula of the degree sequence of these models. Hence, it is important to make such simulation as follows.

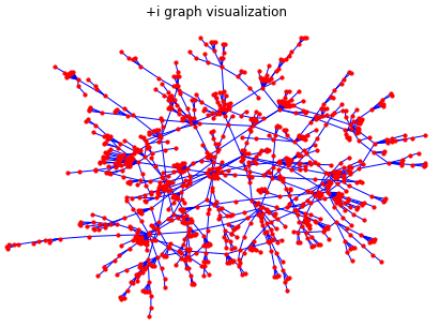
From Appendix E, one could find the programs of RFMs. It can be seen that all the procedures are the same as PAM except the assignment of fitness values. Around line 57 or 59 in the loop of the three programs, the fitness of Plus-1 Model increases by one according to equation (3.1), the fitness of Plus-i Model increases by the number of $index$ according to (3.2), and the fitness of Times-2 Model doubles according to (3.3).

Figure 3.4 shows the intuitive images of the three models. The Times-2 image consists of only 170 vertices because not of time complexity but the computer memory. The running computer is a HP Zbook g5 with a 6-core Intel i7-8750H 2.20GHz CPU, 16GB RAM, and x64 system framework. However, the computation requirement of the Times-2 Model increases to an exponent power. If 1000 nodes is set, the largest fitness value could be estimated between 2^{100} and 2^{1000}, both of which require a cache space more than 100. That is far more than the ability of the computer.

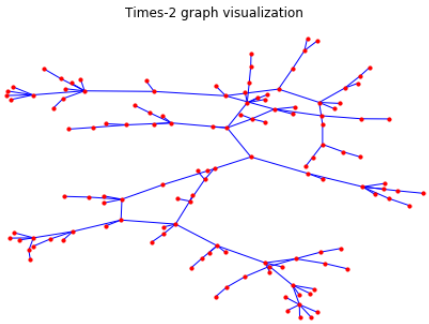
Look back at the figures. It is clear that three models are still scale-free, but the Plus-i Model seems to be the least scale-free and close to the RRT figure 3.2. Details are discussed in the next Section.



(a) Simulation of Plus-1 Model



(b) Simulation of Plus-i Model



(c) Simulation of Times-2 Model

Figure 3.4: Simulation of Recursive Fitness Model

3.6 Inverse Model

In this paper, the Inverse Model mainly refers to the PAMinv. By Definition 2.1.5, the fitness values of PAMinv are the reciprocals of expectation of the degree values of PAM. By doing so, we aim to find a model with some smaller tail exponents than benchmarks. In Section 2, the expectation of the degree sequence of standard PAM over time t has been calculated in Theorem 2.3.3. It is easy to combine them together and get the formula of fitness sequence.

For the PAMinv, the fitness value of node j at iteration time t is as follows.

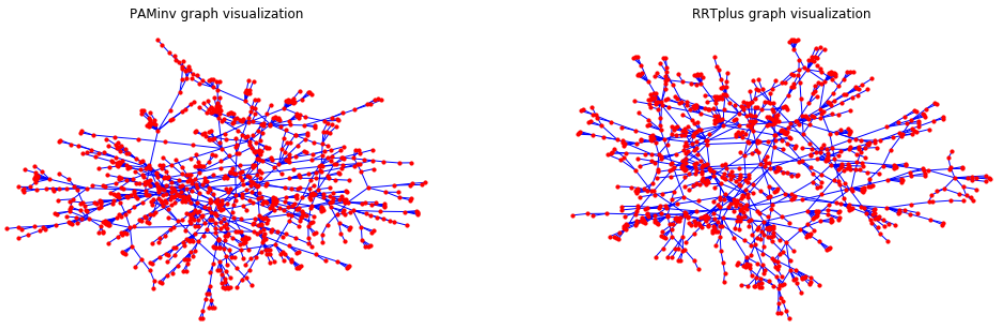
$$f_t(j) = \frac{\Gamma(t+0.5)\Gamma(j)}{\Gamma(t+1)\Gamma(j-0.5)}. \tag{3.4}$$

One can find the corresponding expression in the program in Appendix F line 61. Because of the reference rule of Python, we use $t = t+1$ and $j = j+1$ in the code. In the same line, we use a special function $gammaln$ from the Scipy package instead of the direct function $gamma$. The reason to do so is still on the time complexity. It is well acknowledged that Gamma function (or factorial function)

consumes greatest time in numerical computation, and that is why the running computer breaks down when the iteration goes to around 200 by using $gamma$ function. However, the $gammaln$ function uses the asymptotic expansion (2.20) to reduce the time complexity, and hence we can handle the case with over 10000 iterations. Then, the formula transfers to

$$f_t(j) = \exp(\ln\Gamma(t+0.5) + \ln\Gamma(j) - \ln\Gamma(t+1) - \ln\Gamma(j-0.5)). \tag{3.5}$$

The equation (3.5) is equivalent to the expression in the line 61.



(a) Simulation of Preferential Attachment Inverse Model (b) Simulation of Random Recursive Inverse Tree

Figure 3.5: Simulation of Inverse Models

Figure 3.5 shows the intuitive images of the PAMinv. Hardly can we observe the scale-free feature from the images. But the clustering trend still shows differences from the RRT model, which means the tail exponents of the two models might be larger than RRT but smaller than the PAM. More evidence would be discussed in the Section 4 to support the conjecture.

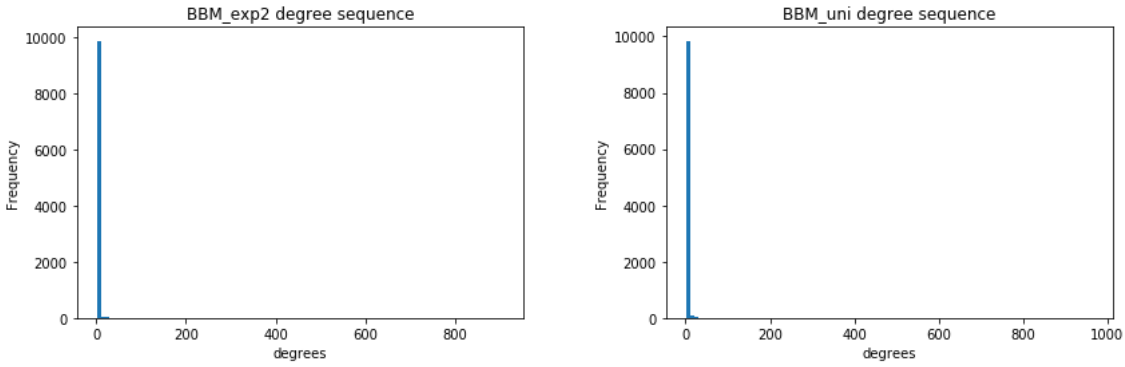
Chapter 4

Results and Discussion

A person can be described by his/her weight, shape, hair color, and so on. Similarly, a random tree also has its characteristics. In this section, we mainly focus on four features of random trees, including the degree sequence, the fitness sequence, the N_k sequence, and the height sequence. The behavior of the four sequences largely determines the shape of a random tree, whether it is scale-free or not. Plenty of figures are put up in order to give an intuitive explanation of the convergence, and comparisons are conducted between different models with benchmark models as RRT and PAM.
Most figures are plotted from models with 10000 iteration steps. In other words, here are results from random graphs with 10000 vertices, and the sample could be large enough to support the following conclusions. However, there are still some exceptions in the RFM cases. For the same reason in the Section 3, the Plus-i Model can be simulated only up to 2000 nodes, and the Times-2 Model has only 130 nodes. 130 is a relatively small number to show some unique features of Time-2 Model, but that is enough to show some common characteristics of the RFMs together with Plus-1 and Plus-i Model.

4.1 Degree Distribution

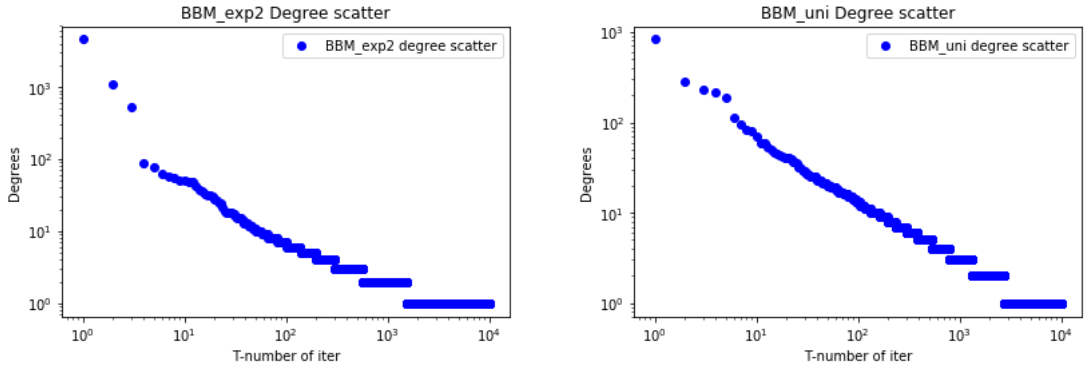
The degree distribution mainly refers to the distribution of degrees of nodes of random graphs at time T ($T = 10000$ unless otherwise stated). Deep researches on benchmark models in this area have been almost fully conducted, so results related to the PAM and RRT do not appear in this part. We are highly interested in the degree sequence behavior of models with dynamical fitness values, including the Bianconi-Barabási Models, the Recursive Fitness Models, and the Inverse Models. Results in this part respond to the mathematical deductions in Section 2 whether a model's degree sequence follows a power-law. Obviously, the degree of a random graph $D_t(j)$ has two variables: the iteration time t and j, the node of graph. Normally, the degree distribution in this part refers to the equation (1.2). However, the degree sequence over time t for some fixed node is sometimes also important and shows some properties of random graph.



(a) Degree Histogram of Bianconi-Barabási Model with exponential(2) fitness distribution

(b) Degree Histogram of Bianconi-Barabási Model with standard uniform fitness distribution

Figure 4.1: Degree Histogram of Bianconi-Barabási Model

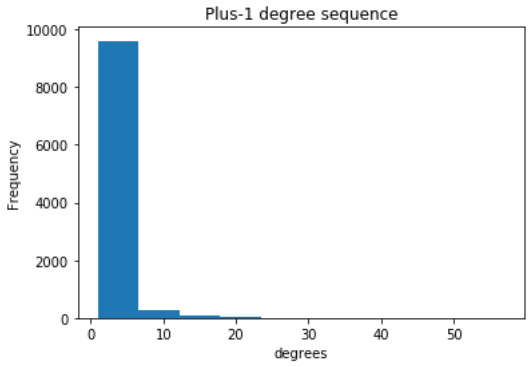


(a) Degree Scatter of Bianconi-Barabási Model with exponential(2) fitness distribution

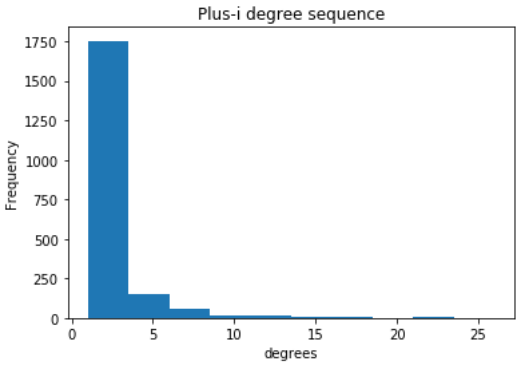
(b) Degree Scatter of Bianconi-Barabási Model with standard uniform fitness distribution

Figure 4.2: Degree Scatter of Bianconi-Barabási Models

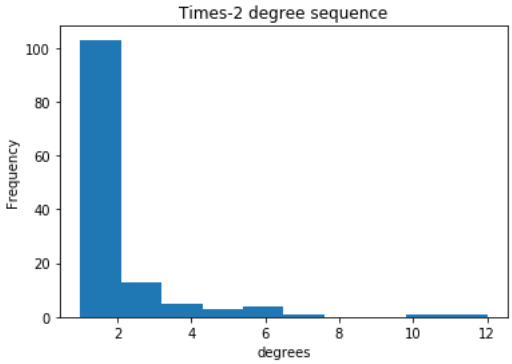
Figure 4.1 and Figure 4.2 show basic features of Bianconi-Barabási Model. We first plot the frequency histograms of two BBMs in Figure 4.1. From both pictures, it is easy to see that nodes in BBM tend to have small degrees rather than a large degree, which means a node in BBM has only few neighbors. That is nice, but it is not enough to say the BBM escapes the power-law. Comparing the two figures in Figure 4.1, one can also notice that the largest degree of BBM with uniform distribution is much smaller than that of the BBM with exponential distribution. We also see that there is one node in BBM with exponential distribution with more than 4000 children, and this is a sign of scale-free feature. Then, it is necessary to look at the trend of the degree sequence. We make a log-log plot of degree sequences of both BBMs over the nodes (base is 10). In log-log plot, it is clear to see a tail exponent as the slope of a line, i.e. a power function $y = x^a$ is transformed into a linear function $\log y = y' = ax' = a \log x$ in a log-log figure. In both figures of Figure 4.2, it is clear that the scatter points of degrees shape as a line, which means the two BBMs' degree sequences follow some power-law. And this results coincides with the conclusion by Barabási [4]. However, it is not essential to find out the precise tail exponent of these models. By Theorem 2.3.5, in BBMs, the tail exponent depends on the reciprocal of the fitness value, $f^{-1}(j)$, which means it may vary from node to node.



(a) Degree Histogram of Plus-1 Model (10000 nodes)

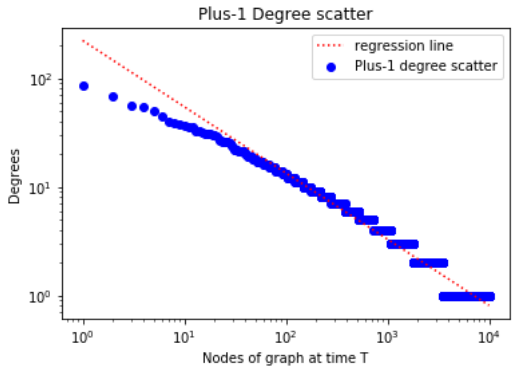


(b) Degree Histogram of Plus-i Model (2000 nodes)

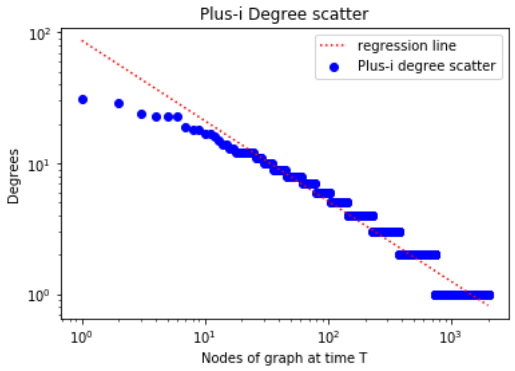


(c) Degree Histogram of Times-2 Model (130 nodes)

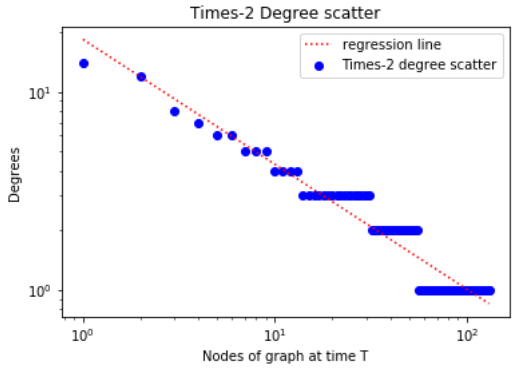
Figure 4.3: Degree Histogram of 3 examples of Recursive Fitness Models



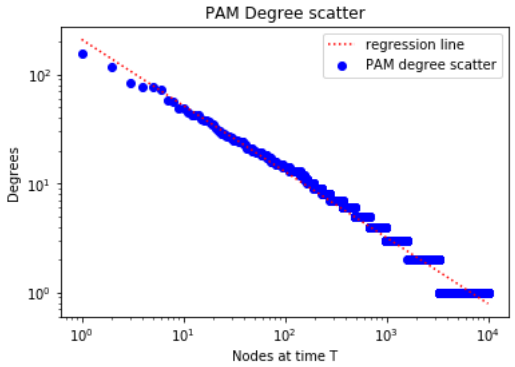
(a) Degree Scatter of Plus-1 Model (10000 nodes) with regression slope -0.6099864496694771



(b) Degree Scatter of Plus-i Model (2000 nodes) with regression slope -0.6129365038829095



(c) Degree Scatter of Times-2 Model (130 nodes) with regression slope -0.6059477221731087



(d) Degree Scatter of Preferential Attachment Model with regression slope -0.6064811008550601 (10000 nodes)

Figure 4.4: Degree Scatter of 3 examples of Recursive Fitness Models and Preferential Attachment Model as benchmark.

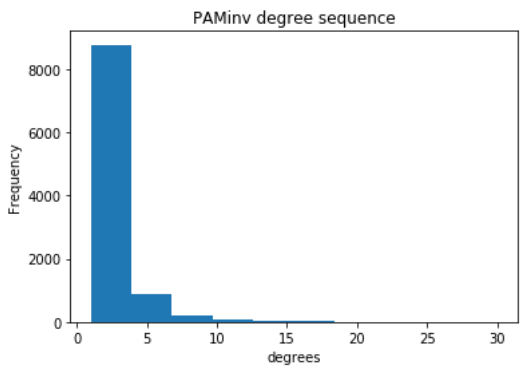
At first, it is essential to mention that with the proof of Theorem 2.3.8, it is worthy to put energy on the degree sequence. Otherwise in case of deviation of degree sequence, we cannot find the trend of models. Figure 4.3, and 4.4 show basic features of Recursive Fitness Model. Similar to the BBMs, the nodes of 3 RFMs at time T also tend to have small degrees. However, the largest degree of RFMs becomes smaller than BBMs, despite the smaller sample size. This common phenomenon means for RFMs, most nodes attach themselves to few neighbors, but a small quantity of vertices would have larger degrees. This directly leads to a scale-free feature. The point will be tested again by the N_k sequence.

Then, we plot the log-log Degree-Nodes scatter figures of the RFMs and PAM as benchmark in Figure 4.4. Regression lines are added to these figures to make analyses on the tail exponents and asymptotic results of RFMs. Obviously, one could find the slope of regression line of PAM is around -0.6, but it should be -0.5 according to Theorem 2.3.3. This is not a contradiction, simply because it is mentioned as an expected value in Theorem 2.3.3 and the $(\frac{t}{i})^{\frac{1}{2}}$ is also an asymptotic result over iteration time t. Figure 4.7 and 4.4 are plotted from only 10000 or fewer nodes, and the sample could be too few to reach the limitation. Also, we make the regression line based on all the nodes of RFMs and PAM, which means some outliers may make sense. But in a nutshell, the regression lines produce a glance of the power-law of the degree sequence of the RFMs. Compared to PAM, the slope of the regression lines of RFMs seems to have little fluctuation. This means the degree sequence of RFM would have a power-law almost the same as PAM, i.e. $D_t^{RFM}(j) \sim j^{-\mu_{PAM}}$.
Additionally from the Figure 4.4, the degree sequences of RFMs are plotted from 10^2 nodes to 10^4 nodes, but get almost same regression slope. Since we do not know a lot about the Recursive Fitness Models, it is reasonable to give the following conjecture:

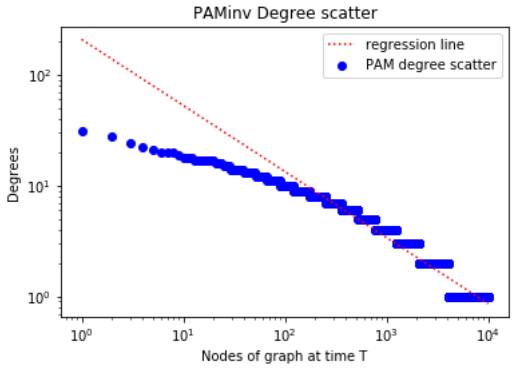
Conjecture 4.1.1. *For all Recursive Fitness Models with an increasing recursive function, their degree sequence follows some power-law, i.e. there exists some positive constant $c = c_m(Model)$ depending on the RFMs and a positive constant $\mu = \mu_{RFM}$, such that for some fixed time t*

$$D_t^{RFM}(j) \sim c_m j^{-\mu}$$

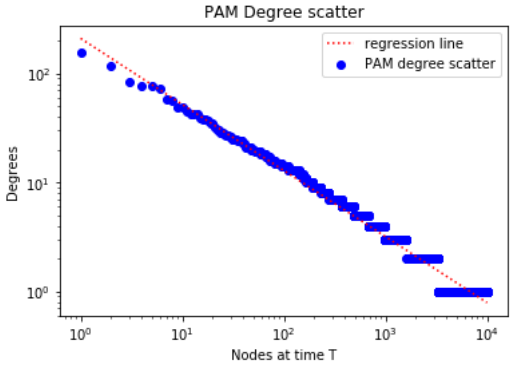
This result shows great difference from BBM. All the results except for the conjecture will be discussed in N_k sequence part as well.



(a) Degree Histogram of Preferential Attachment Inverse Model



(b) Degree Scatter of Preferential Attachment Inverse Model with regression slope -0.5949249127652172



(c) Degree Scatter of Preferential Attachment Model with regression slope -0.6064811008550601

Figure 4.5: Degree Figures of Preferential Attachment Inverse Model and benchmark

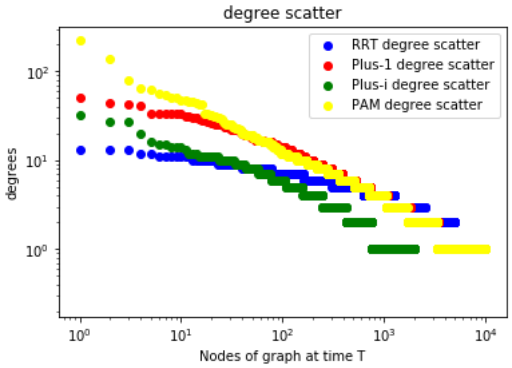
Figure 4.5 shows basic features of Preferential Attachment Inverse Model with PAM here as a benchmark. Figure 4.5a indicates the common phenomenon that most nodes in PAMinv get a small degree, and the higher degree values are, the fewer nodes get the degree. Moreover, the largest degree becomes significantly smaller than the PAM and RFMs, which could be an important hint on the shape of a tree. There should be at least a core in a tree with a large number of neighbors, if the tree has a big maximum degree value. But it is still far from saying about the scale-freeness.

We then plot the log-log Degree-Nodes scatter figures of Preferential Attachment Inverse Model and PAM together with regression lines in Figure 4.5. The regression slopes indicate that the asymptotic degree sequence of PAMinv might be the same as the asymptotic of PAM, since both have the same regression slope. However, as one can notice in Figure 4.6b, the degree sequence of PAMinv does not agree with the degree sequence of PAM for the first several nodes. This contributes a deviation against the regression line. The scatter points cannot form a straight line but a curve in the log-log plot, which means that there might be some other addition in the relation of $D_t(j)$, i.e. there exist a positive parameter $c_{PI} = c_{PI}(T) > 0$ depending on iteration time T, and a positive constant $\mu > 0$, such that for

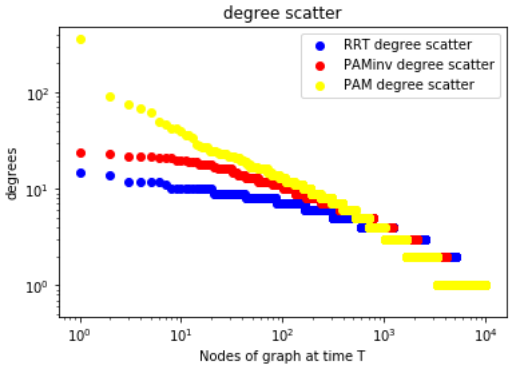
fixed time T, the degree sequence of PAMinv $\{D_T(j)\}_{j=1}^{T}$ satisfies

$$D_T(j) \sim c_{PI}(T) j^{-\mu} + o(j^{-\mu}). \tag{4.1}$$

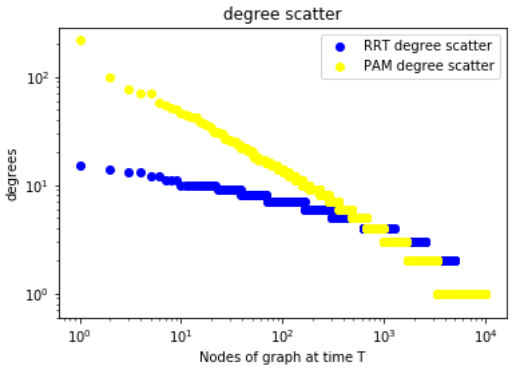
In RFMs, considering Figure 4.4, it appears that the second term $o(j^{-\mu})$ turns zero for a large node, so the scatter points form a straight line in log-log plot figures, but for PAMinv, the final element makes great sense to escape from the line into a curve. However, in asymptotic cases, the $o(j^{-\mu})$ becomes negligible, and shows the regression slope as above.



(a) Degree Scatter of Recursive Fitness Models and benchmarks



(b) Degree Scatter of Preferential Attachment Inverse Model and benchmarks



(c) Degree Scatter of benchmark models

Figure 4.6: Degree Sequence Comparisons

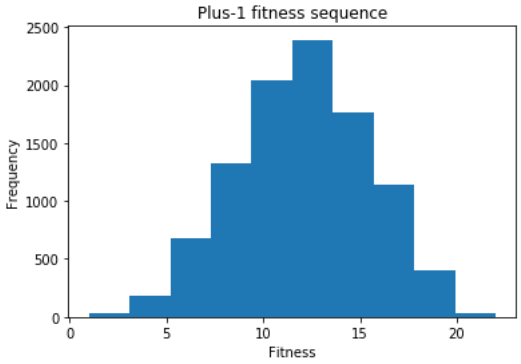
In Figure 4.6, comparisons are made among different kinds of models, and PAM and RRT as benchmark models are also plotted together. Regression lines do not appear in this figure in order to make it clear. One can refer to previous figures to check the asymptotic results. The BBMs also do not appear in this comparison part, simply because the degree sequence properties of BBM have already been adequately discussed in other papers, such as the one by Bianconi and Barabási [4].

Basically, as shown in Figure 4.6a, marked blue, the degree sequence of RRT has an exponential tail, which is different from any other model mentioned in this paper. There is an intersection between RRT and PAM in Figure 4.6c, and the two models also have different asymptotic properties. In Figure 4.6a, two representatives of RFM, Plus-1 and Plus-i, are shown as red line and green line. Putting them together, it is clear that the yellow line almost covers the red line after the first some nodes, which means the degree sequence of Plus-1 Model should have similar tail exponent as PAM. One could notice that the green line is a bit lower than the red and yellow lines, but follows almost the same trace. This is because of the fewer nodes to plot the Plus-i Model due to the computer memory. Neither of these RFMs goes close to the RRT.
In Figure 4.6b, PAMinv is plotted to show the difference. It is not very clear that the tail of PAMinv lies between RRT and PAM or covered by the PAM. But the initial part of red line imitates the trend of blue

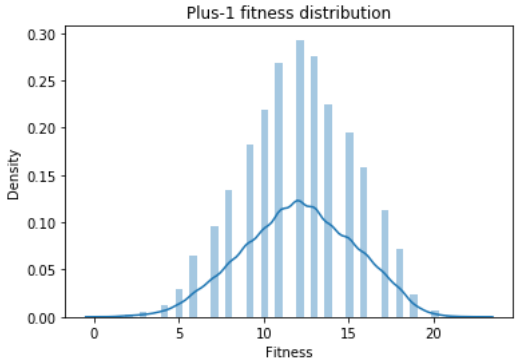
line, which means for initial nodes, the attachment selection of PAMinv should be almost uniformly at random from previous nodes. But later PAMinv gets close to the PAM process.

4.2 Fitness Distribution

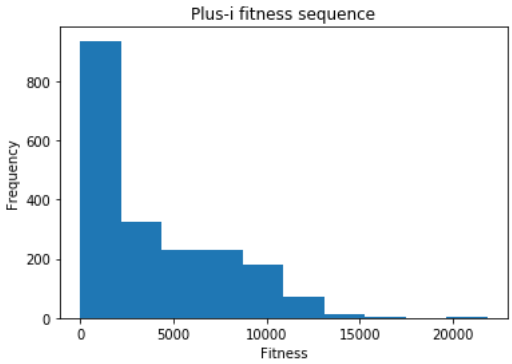
In the Section 3, it is introduced that the program also outputs a fitness sequence *fit*. Normally, the *fit* denotes the fitness values of each node at time T ($T = 10000$ in most cases). Depending on the models, the fitness values of a node become a constant drawn from some fixed distribution or a recursive function, and will not change overtime or vary over iteration steps. There are a lot of researches on constant (PAM) and fixed distribution (BBM) cases, so we focus on the models with time-varying fitness. Thus, results from RFM and PAMinv are put up in this part, and we ignore the PAM, RRT, and BBM results here.



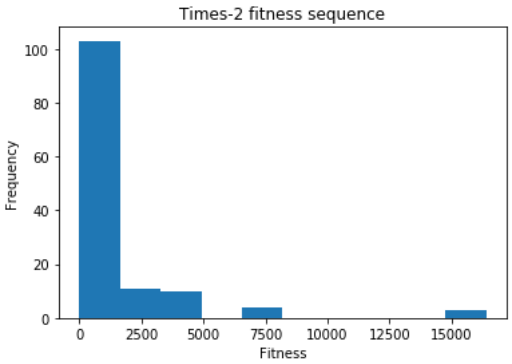
(a) Fitness Histogram of Plus-1 Model (10000 nodes)



(b) Fitness Distribution of Plus-1 Model (10000 nodes)



(c) Fitness Histogram of Plus-i Model (2000 nodes)



(d) Fitness Histogram of Times-2 Model (130 nodes)

Figure 4.7: Fitness Histogram of 3 examples of Recursive Fitness Models

Figure 4.7 are 3 histograms and 1 distribution of fitness sequence of 3 examples of RFMs at time T. We plot the frequency of fitness values of the 3 models. Fitness of Plus-1 Model suggests a Gaussian distribution with a median of around 12, while in the other two models, the fitness values follow some other distributions with a high density towards zero. In the Plus-1 Model, the phenomenon means most nodes get a height of 11, and fewer nodes get a height around 11. This coincides with the height distribution in Section 4.4. Then, for the Plus-i model, one can notice a second peak around fitness value 7500. According to the model, this means quite a few nodes tend to attach themselves to some specific parent with some special fitness choice. This phenomenon still links to a scale-free feature. Combining Figure 4.7b,c together, we can see that most nodes have small fitness values (less than 1000). By Definition 2.1.4, this means new nodes still tend to surround the original node (with a fitness value 1), and could be the reason why RFMs are still scale-free models.

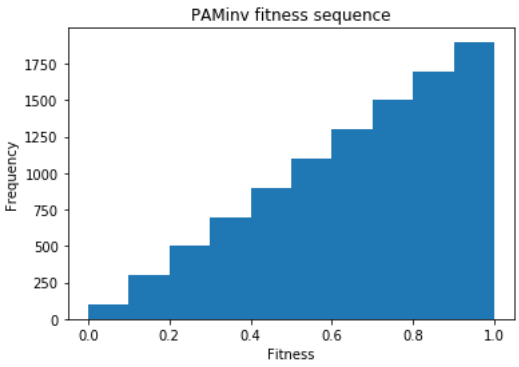


Figure 4.8: Fitness Histogram of Preferential Attachment Inverse Model at time t

Figure 4.8 provides all the information about PAMinv fitness sequence. As Definition 2.1.5, the PAMinv has a time-varying fitness sequence. The left figure shows the fitness sequence of each node at time T, and the right figure shows the fitness sequence of original node from time $t = 1$ to $t = T$. We plot the frequency of fitness values in the left figure, where one can clearly notice a 'triangle' between 0 and 1. Since the reciprocal of the expectation of $D_t(i)$ is applied in this model, the distribution strictly coincides with the equation (2.19), if we compute $i = 1, 2, \ldots, T$.

4.3 N_k Sequence

$N_k(t)$ refers to the number of vertices of a random trees with degree number k at time t. The following equation gives the relation between N_k and $D(j)$ at time t.

$$N_k = \sum_{j=1}^{t} \mathbb{1}_{\{D_t(j)=k\}} \tag{4.2}$$

By Theorem 2.3.1, the N_k sequence of PAM has a tail exponent of -3. For BBMs, RFMs, and even the PAMinv, the sequence of $\frac{N_k(t)}{t}$ can be simulated, and the limiting behavior of $\frac{N_k(t)}{t}$ can also inferred from the simulations. Many researches have been done on the BBMs, so in this part, we mainly focus on the N_k sequence behavior of RFMs and PAMinv. The N_k sequence comes from the degree sequence, and as a consequence, the N_k sequence responds to properties of models shown by the degree sequence.

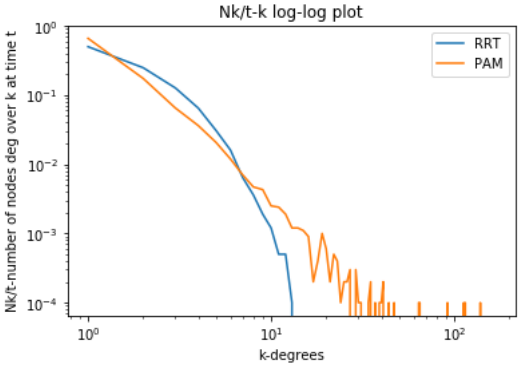
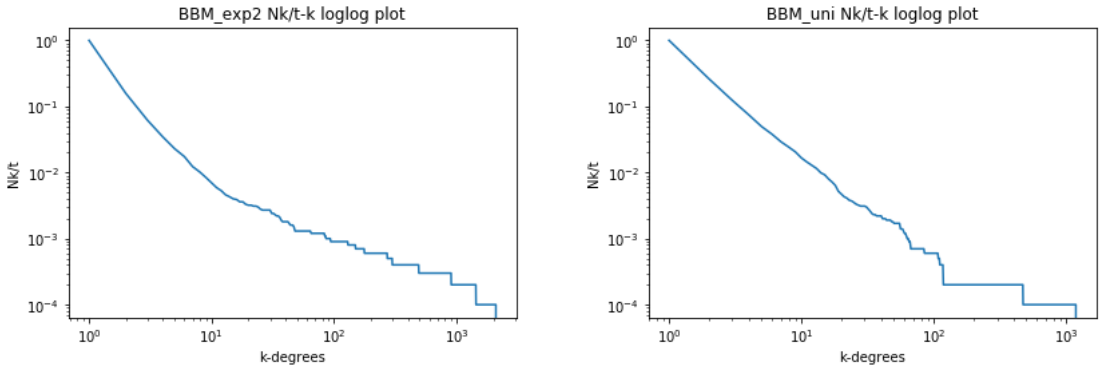


Figure 4.9: N_k Distribution log-log plot of PAM and RRT

Although it is not the key point in this paper, we still put the benchmark models here to show some standard data. Similar to the degree sequence, the RRT outputs the blue curve in the log-log plot. On the other hand, the orange line represents the PAM and shows a regression slope around -2.5 out of the same reason in the degree sequence part. -3 should be an asymptotic value for a large iteration

time t and node j, but apparently 10000 is not large enough, Due to the computer memory, it is not applicable to run a huge amount of iteration numbers, so we just set the slope as the basement value of comparison.

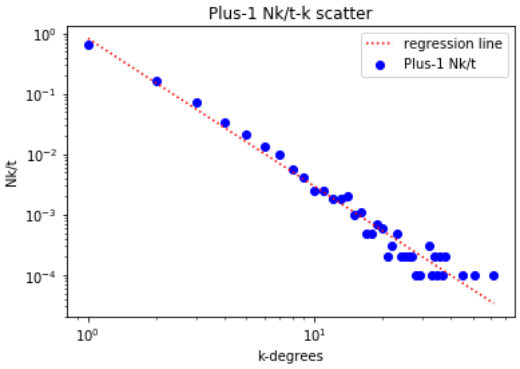


(a) N_k Distribution log-log plot of Bianconi-Barabási Model with exponential(2) fitness distribution

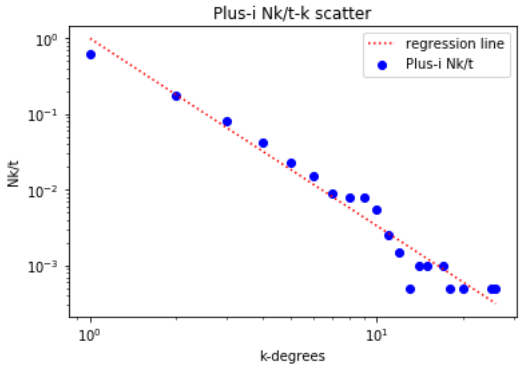
(b) N_k Distribution log-log plot of Bianconi-Barabási Model with standard uniform fitness distribution

Figure 4.10: N_k Distribution log-log plot of 2 examples of Bianconi-Barabási Models

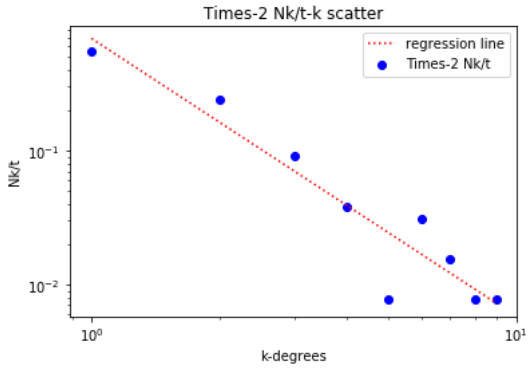
As shown in Figure 4.10, N_k sequences of both BBMs with exponential and standard uniform fitness distribution have a heavy tail. Compared to the BBM-stduni, the figure of BBM-exp shows a convex curve rather than a straight line, which indicates a larger tail exponent. It is unclear to make comparison through the two figures, Figure 4.10a and b. Thus, we turn to the Figure 4.14a. This picture shows that the tail exponent of N_k sequence of BBMs strongly resembles that of PAM. However, significant different could be seen between two BBMs. Bias toward PAM become clear for the BBM with exponential fitness distribution. As shown in Bianconi and Barabási's paper [4], the tail exponent of BBMs varies with the fitness distribution.



(a) N_k Scatter of Plus-1 Model (10000 nodes) with regression slope -2.4843674772247915

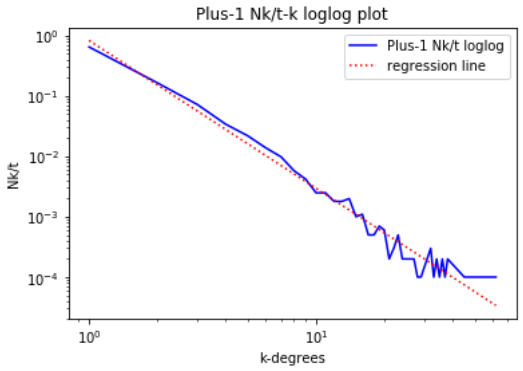


(b) N_k Scatter of Plus-i Model (2000 nodes) with regression slope -2.3760571002731883

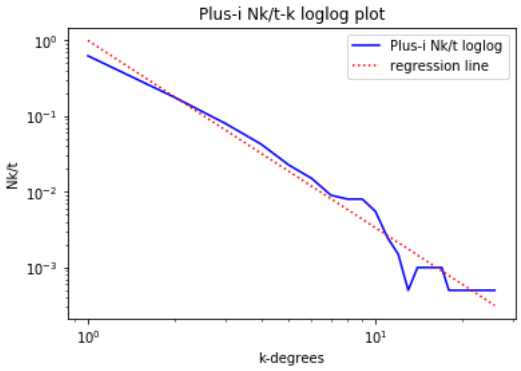


(c) N_k Scatter of Times-2 Model (130 nodes) with regression slope -2.0720278195132433

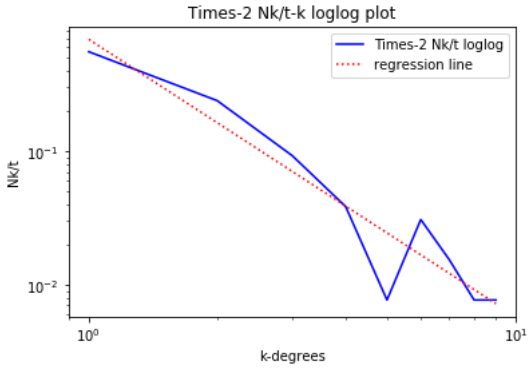
Figure 4.11: N_k Scatter of 3 examples of Recursive Fitness Models



(a) N_k Distribution log-log plot of Plus-1 Model (10000 nodes)



(b) N_k Distribution log-log plot of Plus-i Model (2000 nodes)



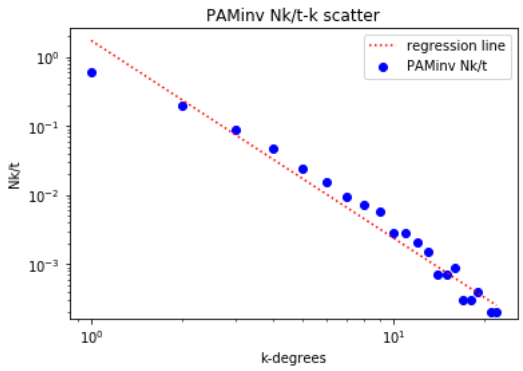
(c) N_k Distribution log-log plot of Times-2 Model (130 nodes)

Figure 4.12: N_k Distribution log-log plot of 3 examples of Recursive Fitness Models

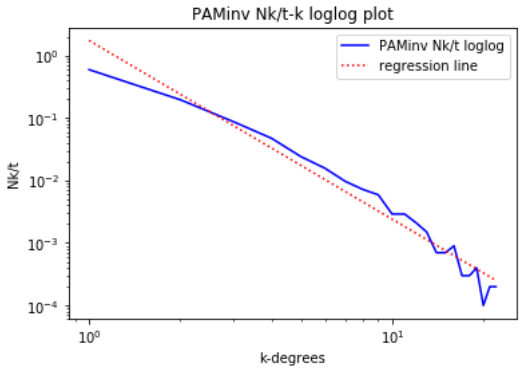
Figure 4.12 and 4.11 show some basic features of RFMs. In both figures, we plot the regression line and the slopes are labelled in the captions. As for the slopes, the Times-2 model has a large fluctuation period, if running programs for several times. The reason might be the small quantity of simulation points. Thus, the slope of Times-2 Model will not be taken into consideration. But even looking at the remaining two models, the regression slopes are quite close to the PAM, but a bit smaller. This highly coincides with the results shown in degree sequence part: the RFMs have almost the same tail exponent as PAM. But the RFMs might have a heavier tail than PAM, according to the regression slopes in Figure 4.11. The result is reinforced in the comparison part by Figure 4.14.
Still in Figure 4.11, together with Figure 4.12, some new results can be reached. One could notice that the regression slopes of RFMs might change a lot when choosing a totally different iteration number t. As we use the N_k/t as the dependent variable, this means that the variable $N_k(t)$ can be split into a function of t and a function of k, where the function of t has a power over or below 1. In other words, we can make such conjecture:

Conjecture 4.3.1. *For all the Recursive Fitness Models with a non-constant increasing recursive function, the number of nodes with degree k at time t can be defined as the composition of two independent functions l and g:*

$$N_k(t) = l(t)g(k), \text{ where } l(t) = t^{\alpha}, \quad \alpha > 0, \alpha \neq 1$$



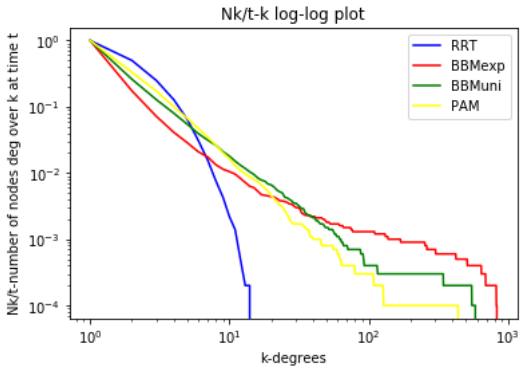
(a) N_k Scatter plot of Preferential Attachment Inverse Model with regression slope -2.834284686143112



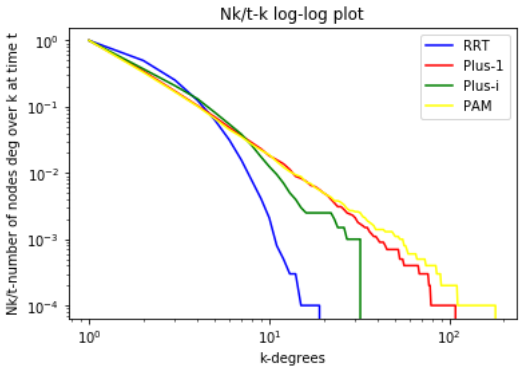
(b) N_k Distribution log-log plot of Preferential Attachment Inverse Model

Figure 4.13: N_k sequence plot of Preferential Attachment Inverse Model

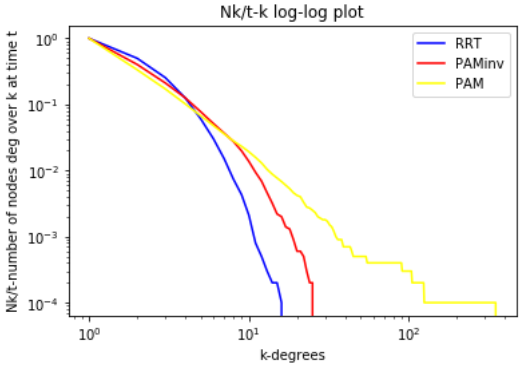
Figure 4.13 shows some basic features of PAMinv. First, it is clear that the regression slope is a bit larger than the PAM, which means the N_k sequence of PAMinv would have a lighter tail than PAM. This also coincides with the results in degree sequence part. Second, one may notice that the slight concavity of the curve in the log-log plot of PAMinv. Compared to the regression line, the blue curve is rather concave. The special phenomenon leads to a polynomial of N_k function, i.e. the expression of equation (4.1) is once more hinted at.



(a) N_k Distribution log-log plot of Bianconi-Barabási Models and benchmarks



(b) N_k Distribution log-log plot of Recursive Fitness Models and benchmarks



(c) N_k Distribution log-log plot of Preferential Attachment Inverse Model and benchmarks

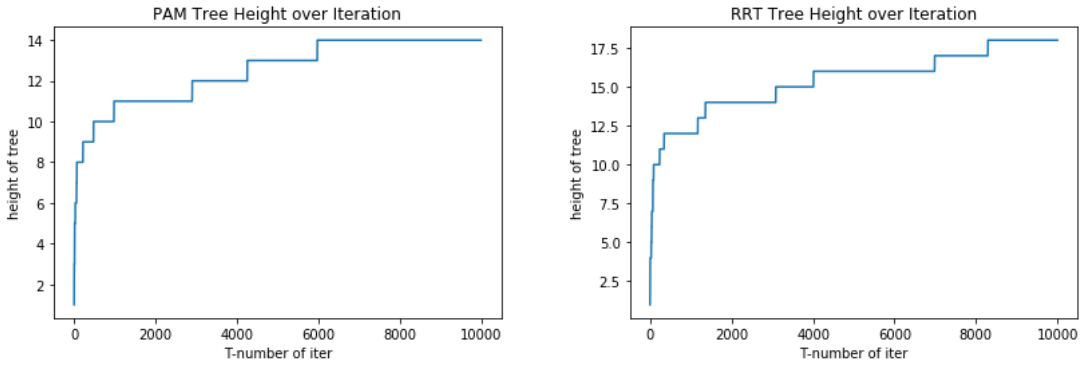
Figure 4.14: N_k Distribution log-log plot Comparisons

In Figure 4.14, comparisons are made among different kinds of models, and PAM and RRT as benchmark models are also plotted together. In Figure 4.14a, the two BBMs lie in each side of PAM line, which shows that the N_k sequence of these two BBMs has a heavy tail almost the same as PAM. In Figure 4.14b, the Plus-1 and [lus-i Models largely imitate the path of PAM line and end up to the left. This also indicates a heavy tail of RFMs and approximately equal tail exponent to PAM. Due to the fact that different models may have different tree sizes, for example, the Plus-i Model, N_k sequence cannot be compared directly. However, the division of N_k by t makes the element comparable among models despite different sample sizes. In Figure 4.14c, the PAMinv line lies between RRT line and PAM line from beginning to end. This could be an interesting phenomenon which can be summarized in the following conjecture. These conclusions highly coincide with the results given by degree sequence in Section 4.1.

Conjecture 4.3.2. *The tail exponent of Preferential Attachment Inverse Model (PAMinv) lies between that of Preferential Attachment Model (PAM) and the exponential tail.*

4.4 Height of Random Trees

Height is another important special feature of a random graph. By Definition 2.3.2, the height H_t describes the length of a tree. Intuitively, from the figures in Section 3, with similar Algorithm 1, a tree with a smaller height tends to have larger branches , which means to be less scale-free. Thus, it is essential to take a view of the increments of the heights of our models. All the results in this section refer to simulations. Some conjectures are also included in this section, but further arguments would be left for later researches.



(a) Height of Preferential Attachment Model

(b) Height of Random Recursive Tree

Figure 4.15: Height of Benchmark Models

The Figure 4.15 shows the height-iterative steps relation for the benchmark models, PAM and RRT. In both figure a and b, we plot the height of PAM and RRT over 10000 iterations, which equals 10000 nodes in the graphs. As shown in both figures, the height of PAM and RRT has staircase increments, which means the models do not prolong themselves during some time. In the first few iterative steps, the height roars fast, but later the height slows down to climb the stairs. This shape indicates a significant logarithm function over the iteration steps, and the result undoubtedly coincides with Theorem 2.3.4 and Theorem 2.3.12. Compared to PAM, the RRT has a maximum height at around 18 with 10000 iterations, which is a bit larger than 14 of PAM. Here we can make a conjecture justified by the previous simulations:

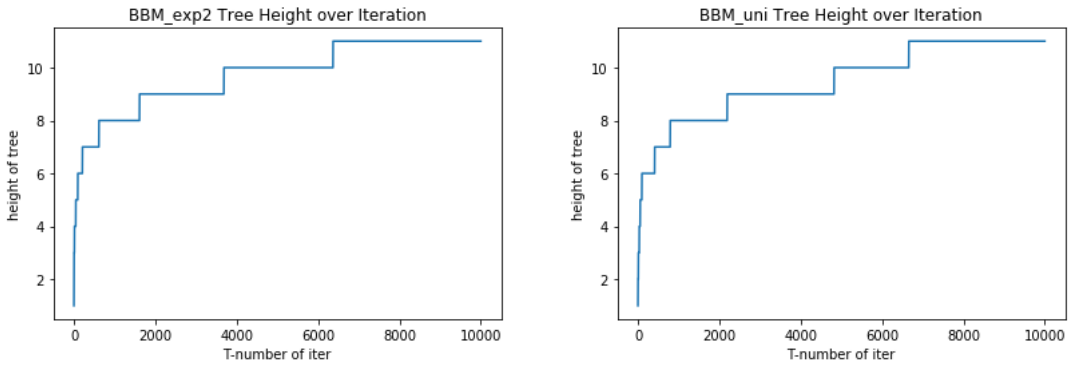
Conjecture 4.4.1. *(Height of PAM Vs RRT) The coefficient c_0 in Theorem 2.3.4 could be smaller than the constant e, i.e. $0 < c_0 < e$.*

Figure 4.16 shows the height sequence of the Bianconi-Barabási Models. We again take the BBM with exponential(2) fitness distribution and the BBM with standard uniform fitness distribution as examples. Similar to the benchmark models, the heights of BBMs still show staircase increments. The

height of both BBMs also lifts up at first and the acceleration becomes smaller and smaller afterwards. So it is reasonable to give the following conjecture.

Conjecture 4.4.2. *(Height of Bianconi-Barabási Model) For any Bianconi-Barabási Model with a fitness distribution ν, there exists a positive constant $c_B = c_B(\nu) > 0$, such that with high probability, the height of the BBM at time t is at most $c_B \log t$.*

Comparing the Figure 4.16a,b, we can hardly find any differences. Both models have the same supreme, and neither of them show some special tendency. Considering the same mean of the fitness distribution of two models, one could also raise a stronger guess based on Conjecture 4.4.2 that we can reinforce the condition of c_B to depend on the expectation of fitness distribution. Moreover, one could still guess if the constant c_B does not need any restrictions.



(a) Height of Bianconi-Barabási Model with exponential(2) fitness distribution

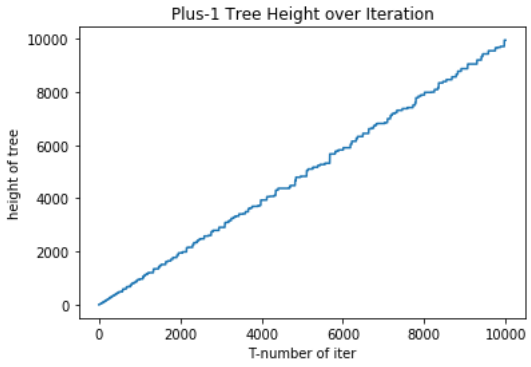
(b) Height of Bianconi-Barabási Model with standard uniform fitness distribution

Figure 4.16: Height of Bianconi-Barabási Models

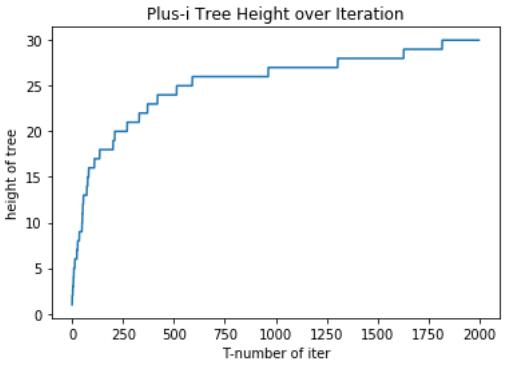
Figure 4.17 shows the height sequence of 3 examples of Recursive Fitness Models. It is really unexpected that the height of the Plus-1 Model significantly increases linearly over the iteration steps, which escapes the logarithm height functions: that is,

$$H_t^{Plus-1} \approx t.$$

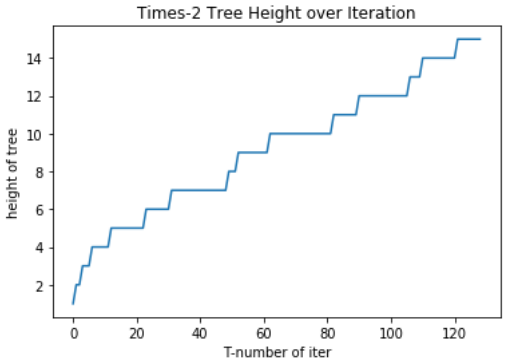
A linear increment means that almost every new node attaches itself to some 'main' branch and forms a string. However, the intuitive Figure 3.4a does not show such obvious trend. A mathematical reason cannot be covered in this paper, but Plus-1 Model has some interesting properties with respect to the height sequence and its fitness distribution.



(a) Height of Plus-1 Model (10000 nodes)



(b) Height of Plus-i Model (2000 nodes)



(c) Height of Times-2 Model (130 nodes)

Figure 4.17: Height of Recursive Fitness Model

Compared to the Plus-1 Model, the remaining two models seem to have a logarithmic height. Due to the computer memory restrictions, only 2000 nodes and 130 nodes are simulated for the two models. That might be a bit insufficient to reveal the whole picture of the height behavior of Plus-i and Times-2, but one can already notice a familiar curve in Figure 4.17b,c. Moreover, apart from the Plus-1 Model, the rest RFMs get a larger height than PAM, BBMs, or even PAMinv. Thus, we can get the following two conjectures.

Conjecture 4.4.3. *For any Recursive Fitness Model, except Plus-1, with a non-constant fitness recursive function F, there exists a positive constant $c_R = c_R(F) > 0$, such that with high probability, the height of the RFM at time t is at most $c_R \log t$.*

Conjecture 4.4.4. *There is a non-constant real recursive function F, such that F induces a Recursive Fitness Model $\mathcal{M}$, and $\mathcal{M}$ forms a random tree G with only $O(1)$ nodes outside the backbone of n nodes.*

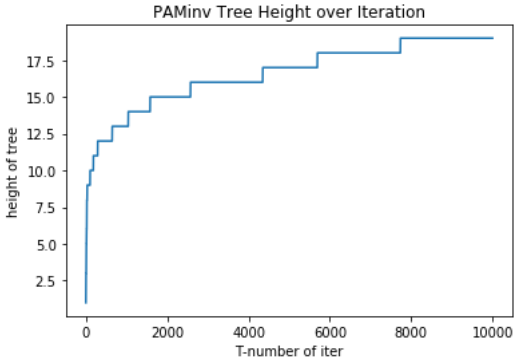


Figure 4.18: Height of Preferential Attachment Inverse Model

The Figure 4.18 shows the height sequence of Preferential Attachment Inverse Model. Compared to the PAM, the supreme of the PAMinv at time t is larger, and the horn of the curve seems sharper, which means the curvature could be larger. Mathematically, this indicates a positive coefficient in front of time t. One may give the conjecture.

Conjecture 4.4.5. *For the Preferential Attachment Inverse Model, there exists a positive constant $c_R = c_R(F) > 0$ and a positive coefficient $m > 0$, such that with high probability, the height of the RFM at time t is at most $c_R \log mt$.*

4.5 Inverse Model

In this part, we mainly focus on the PAMinv. By definition, the expectation of the degree sequence in the generating model divides the degree in order to offset the heavy tail exponent. PAMinv, as mentioned in Definition 2.1.5, is based on the PAM. To make it clear, some different symbols are used in this section. $d_t(i)$ denotes the degree number of node i at time t for PAM, and $D_t(i)$ denotes the number for PAMinv. Then, the probability of a new node $t+1$ attached to the node j chosen from $\{1, 2, \dots, t\}$ can be described by the following equation

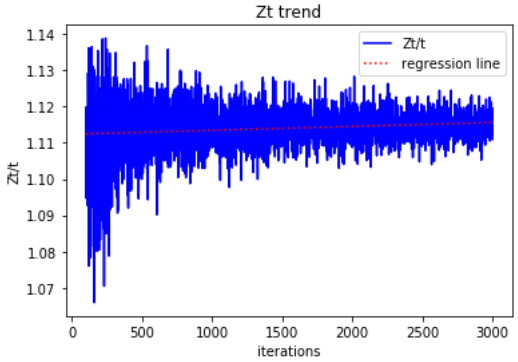
$$\mathbb{P}(t+1 \to j) = \frac{D_t(j)/\mathbb{E}[d_t(j)]}{Z_t}, \text{ where } Z_t = \sum_{l=1}^{t} \frac{D_t(l)}{\mathbb{E}[d_t(l)]}. \tag{4.3}$$

Obviously, the fitness value of a node j becomes $\frac{1}{\mathbb{E}[d_t(j)]}$. We hope that the attachment selection process could highly resemble the one of RRT, which means a uniformly random selection. This relies on the following conjecture.

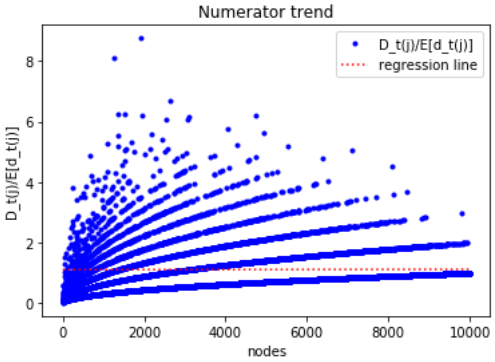
Conjecture 4.5.1. *Given the equation* (4.3)*, for a large t and any node $j = 1, 2, \dots, t$, the probability of a new node $t+1$ attached to node j does not depend on the node j and is almost surely the reciprocal of iteration time, i.e. $\frac{1}{t}$.*

The conjecture can be divided into two parts: First, the denominator in the right part of equation (4.3) divided by time t converges to some constant almost surely, i.e. there exists some constant c, such that $\frac{Z_t}{t} \to c$ a.s. Second, the numerator in the right part of equation (4.3) is independent from node j, i.e. for a large t, for any nodes i_1, i_2, $D_t(i_1)/\mathbb{E}[d_t(i_1)] \approx D_t(i_2)/\mathbb{E}[d_t(i_2)]$.

Evidence shows that it is reasonable to make such conjecture. First of all, in the first two programs in Appendix H, we make a simulation of the behavior of Z_t for $t = 3000$. Restricted to the computer memory, it is hard to set a larger iteration number, but 3000 could be enough to show the trend of Z_t sequence. In Figure 4.19a, it is clear that the Z_t/t sequence fluctuates slightly around the red line with an intercept around 1.11, which indicates the constant c in the conjecture might be 1.11, i.e. $\frac{Z_t}{t} \to 1.11$.



(a) Simulation of $\frac{Z_t}{t}$ of Preferential Attachment Inverse Model with a regression line of $y = 1.1123$ at $t = 3000$



(b) Simulation of $\frac{D_t(j)}{\mathbb{E}[d_t(j)]}$ of Preferential Attachment Inverse Model with a regression line of $y = 1.1116$ at $t = 10000$

Figure 4.19: Asymptotic Simulation of Preferential Attachment Inverse Model

Then, in Figure 4.19b, a regression line is plotted. Generally, it is not a good regression result because of the large variance, i.e. the significant differences between the line and the top points. Also, several arms appear in Figure 4.19b, and this means regression is not an optimal choice for interpolation. However, if we plot the regression line, the intercept coincidentally coincides with that of Figure 4.19a. Although some slips happen here, the red line anyway indicates that the expectation of $D_t(j)/\mathbb{E}[d_t(j)]$ in PAMinv would be a constant for a large number of t, i.e. $\mathbb{E}[D_t(j)/\mathbb{E}[d_t(j)]] \to 1.11$ as $t \to \infty$. If we combine the two results together, it is clear that the expectation of the probability of a new node $t+1$ attached to any old node would be $\frac{1}{t}$, i.e. for any node j in PAMinv,

$$\lim_{t\to\infty} t\mathbb{E}[\mathbb{P}(t+1 \to j)] = \lim_{t\to\infty} t\mathbb{E}[\frac{D_t(j)/\mathbb{E}[d_t(j)]}{Z_t}] = \lim_{t\to\infty} t\frac{1.11}{1.11t} = 1. \tag{4.4}$$

Above are two simulations of the PAMinv, but not a mathematical justification. To give it a proof, a truncation of both denominator and numerator may be introduced. Theorem 2.3.3 and Chapter 8 in van der Hofstad's book [15] show that in PAM, for any node j,

$$\lim_{t\to\infty} d_t(j)t^{-\frac{1}{2}} = \mathbb{E}[d_t(j)]t^{-\frac{1}{2}} = \frac{\Gamma(j-\frac{1}{2})}{\Gamma(j)} = \gamma_j \text{ a.s.} \tag{4.5}$$

Here we write the expression $\frac{\Gamma(j-\frac{1}{2})}{\Gamma(j)}$ as a variable γ_j. Then, we apply the Chebyshev inequality: for any $\eta > 0$

$$\mathbb{P}(|d_t(j) - \mathbb{E}[d_t(j)]| > \eta) \leq \frac{Var(d_t(j))}{\eta^2} \approx \frac{t}{\eta^2}. \tag{4.6}$$

Here, the approximation of variance of $d_t(j)$ over time t comes from the almost surely convergence of $d_t(j)t^{-\frac{1}{2}}$ in equation (4.5). Setting η as $t^{\frac{1}{2}+\delta}$ with an infinitely small term δ, inequality (4.6) becomes

$$\mathbb{P}(|d_t(j) - \mathbb{E}[d_t(j)]| > t^{\frac{1}{2}+\delta}) < t^{-2\delta}\kappa. \tag{4.7}$$

If we consider a large t, the right hand side of (4.7) goes to zero, which means the probability of subtraction between $d_t(j)$ and its expectation is no larger than $t^{\frac{1}{2}}$.

By using this property, we can conduct the truncation of equation (4.3). First, look at the denomi-

nator.

$$Z_t = \sum_{l=1}^{t} \frac{D_t(l)}{\mathbb{E}[d_t(l)]} \tag{4.8}$$

$$= \sum_{l=1}^{t} 1 + \sum_{l=1}^{t} \frac{D_t(l) - d_t(l) + d_t(l) - \mathbb{E}[d_t(l)]}{\mathbb{E}[d_t(l)]} \tag{4.9}$$

$$= t + \sum_{l=1}^{t} \frac{D_t(l) - d_t(l)}{\mathbb{E}[d_t(l)]} + \sum_{l=1}^{t} \frac{d_t(l) - \mathbb{E}[d_t(l)}{\mathbb{E}[d_t(l)]} \tag{4.10}$$

$$\leq t + \sum_{l=1}^{t} \frac{t^{\frac{1}{2}}}{t^{\frac{1}{2}}} + \sum_{l=1}^{t} \frac{D_t(l) - d_t(l)}{\mathbb{E}[d_t(l)]} \text{ (by inequality (4.7))} \tag{4.11}$$

$$= 2t + \sum_{l=1}^{t} \frac{D_t(l) - d_t(l)}{\mathbb{E}[d_t(l)]}. \tag{4.12}$$

At the same time the numerator gives a similar result.

$$\frac{D_t(j)}{\mathbb{E}[d_t(j)]} = 1 + \frac{D_t(j) - d_t(j) + d_t(j) - \mathbb{E}[d_t(j)]}{\mathbb{E}[d_t(j)]} \tag{4.13}$$

$$= 1 + \frac{d_t(j) - \mathbb{E}[d_t(j)}{\mathbb{E}[d_t(j)]} + \frac{D_t(j) - d_t(j)}{\mathbb{E}[d_t(j)]} \tag{4.14}$$

$$\leq 1 + \frac{t^{\frac{1}{2}}}{t^{\frac{1}{2}}} + \frac{D_t(j) - d_t(j)}{\mathbb{E}[d_t(j)]} \text{ (by inequality (4.7))} \tag{4.15}$$

$$= 2 + \frac{D_t(j) - d_t(j)}{\mathbb{E}[d_t(j)]}. \tag{4.16}$$

Obviously, in both equations appears a difficult term $\frac{D_t(j)-d_t(j)}{\mathbb{E}[d_t(j)]}$ which cannot be eliminated by known conditions. Moreover, we want to estimate the probability of a node attaching itself to some previous node, and this is equivalent to the fact that we want to measure the difference between the degree sequence of PAM and PAMinv. However, equations (4.12) and (4.16) indicate that we use the result to estimate the condition, which forms a paradox. Chebyshev inequality might be useless in asymptotic degree sequence of PAMinv.

Another idea may come up. One could complete the proof by means of the Pólya urn process. Theorem 2.3.6 is based on the case that the elements in matrix A are fixed and independent from time t. However, in our case, the fitness sequence varies with iteration time t. It is natural to make such a conjecture:

Conjecture 4.5.2. *(Pólya Urn Process with time-varying fitness) With the same assumption as Theorem 2.3.6, the branching matrices $A = A(t)$ depending on the iteration time $t = 1, 2, \ldots, T$, with the largest eigenvalues $\lambda_1(t)$ and eigenvectors $v_1(t)$. There exists a specific number λ_1 and its corresponding vector v_1, such that*

$$\lim_{t\to\infty} \lambda_1(t) = \lambda_1, \quad \lim_{t\to\infty} v_1(t) = v_1$$

Then, the time-varying Pólya urn process satisfies

$$\frac{X_t}{t} \to \lambda_1 v_1, \quad \text{as } t \to \infty$$

This conjecture could be justified by the following simulation. In the third program in Appendix H, we give the expected format of transformation matrix $A(t)$ over iteration time t. By definition of matrix

A given in Janson's paper [11], we can define the time-depending matrix $A(t)$ as follows. ξ refers to Definition 2.3.3, and $A(t)$ is a $r(t) \times r(t)$ matrix:

$$A(t) := \{a_j(t)\mathbb{E}[\xi_{ji}]\}_{i,j=1}^{r(t)}. \tag{4.17}$$

Here $r(t)$ is the largest degree number plus 1 in the tree at time t, i.e. $r(t) = \max(D_t(j)) + 1$. $A(t)$ is a $r(t) \times r(t)$ sparse square matrix with $r(t)$ rows and columns, which shows the degree number transformation from time t to $t+1$, and the non-zero elements are filled almost diagonal. $\xi_{1,1}$ is 2 because the first segment connects 2 nodes with degree 1, according to Algorithm 1. For the i^{th} row,

$$\xi_{i,i} = -Card\{\text{vertices with degree } i\}.$$

It loses the number of vertices with degree i, when a new node attaches it self to a node with degree i. At the same time,

$$\xi_{i,i+1} = 1 + Card\{\text{vertices with degree } i\}.$$

It increases 1 plus the number of vertices with degree i, when a new node attaches itself to a node with degree i. There is always a new node with degree 1, so $\xi_{j,1} = 1$. Then, $\mathbb{E}[\xi_{ji}]$ has to multiply time-depending weight aa_j, which equals the selection probability sequence $prob$ in PAMinv programs.

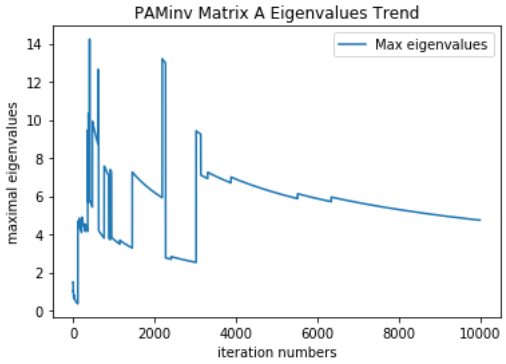


Figure 4.20: Maximal Eigenvalues within 10000 iteration steps converging toward 4.738414198337385

Figure 4.20 gives an intuitive picture of the maximal eigenvalues over 10000 iterations. One could notice that the eigenvalues converge toward around 4.5 despite a fierce fluctuation at first. The result in return strongly support the Conjecture 4.5.2 that the time-varying maximal eigenvalues should converge to some fixed value.

Chapter 5

Conclusion

As mentioned in the introduction section, the paper highly focuses on the power-law of random graphs, including two kinds of newly developed graphs: Preferential Attachment Model (PAM), Random Recursive Tree (RRT) as benchmark models, and Bianconi-Barabási Model (BBM), and two kinds of graphs which first appear in this paper: Recursive Fitness Model (RFM), and Inverse Model. To be precise, two special cases of Bianconi-Barabási Model: exponential(2) distribution and standard uniform distribution are taken into consideration in this paper, and three examples of Recursive Fitness Model are Plus-1 Model, Plus-i Model, and Times-2 Model. Then, setting the reciprocal of expectation of degree sequence of Preferential Attachment Model as a time-varying fitness sequence, we invent a new model to detect its power-law, and the model is called Preferential Attachment Inverse Model (PAMinv). Developed methods can be applied to analyze the degree sequence of PAM and BBM, but they are not fully applicable to the recursive models and models with time-varying fitness. These blanks will be left to fulfill.

In this paper, the tail of RRT is proven, and investigation on the RFM and PAMinv is conducted through both mathematical and empirical ways.
In theory, by using similar methods to prove properties of PAM, the N_k sequence of RFM converges to its expectation over time t. In simulation of the three examples of RFMs, the Plus-1 Model stands out with an almost Gaussian distributed fitness sequence, while fitness values of other models tend to be small for most vertices but have a tail. However, RFMs still follow a similar power-law of degree sequence to PAM, according to simulation results.
Then, through simulation, PAMinv, as expected, shows a different power-law of degree sequence between PAM and RRT, and the tail exponent of N_k sequence of PAMinv lies between that of PAM and RRT, which means PAMinv could be a useful model to interpolate RRT from PAM. Attempts are made to analyze the phenomenon. We simulated the asymptotic behavior of $\frac{Z_t}{t}$ and $\frac{D_t(j)}{\mathbb{E}[d_t(j)]}$ (defined in equation (4.3)), and the simulation results support the phenomenon, but mathematical proof falls into a paradox. After that, we give a conjecture on Pólya urn process with time-varying fitness.

Finally, height of a random tree is another point on which light is thrown. Plus-1 Model, one of RFMs, becomes again the special example, which has an almost linear height distribution over iteration time t, while we make a series of conjectures to say all other models have a logarithmic height distribution over time t. The result is consistent with the simulation figures, but only PAM and RRT are theoretically proven to obtain a logarithmic height, and the proof of other models are left for further investigation.

Appendix A

Code of Random Pick process

Select an element from a weighted sequence according to the weights.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 23 11:03:39 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random

def random_pick(sequence,probabilities):
    x = random.uniform(0,sum(probabilities))
    cumulative_probability=0.0
    #global item, item_probability
    for item,item_probability in zip(sequence,probabilities):
        cumulative_probability+=item_probability
        if x < cumulative_probability:
            break
    return item


def random_picks(sequence,probabilities):
    table = [z for x,y in zip(sequence,probabilities) for z in [x]*y]
    while True:
        return random.choice(table)
```

Appendix B

Code of Preferential Attachment Model

Python code of standard PAM.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  7 14:40:14 2020

@author: wangr
"""

#standard Preferential Attachment model
import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import networkx as nx


##########
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()

#fitness vector
fit = np.ones((T+1,), dtype=np.int)
fit[0] = 1
print(fit)

#probability to choose any connecting point
prob = np.zeros((T+1,), dtype=np.int)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
#prob = prob/np.sum(prob)
print(prob)

ind = np.ones((T+1,), dtype=np.int)
```

```
#loop over T times
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_picks(seq, [prob[0]])
    else:
        index = random_picks(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    ind[t+1]=ind[index]+1
    for j in range(t+1):
        prob[j]=deg[j]*fit[j]
    #prob = prob/np.sum(prob)

print('degree sequence is as follows \n', deg)
print('fitness sequence is as follows \n', fit)
print('height of the tree is as follows \n', max(ind))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.xlabel('degrees')
plt.ylabel('Density')
plt.title('PAM degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.xlabel('Fitness')
plt.ylabel('Density')
plt.title('PAM fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.xlabel('degrees')
plt.ylabel('Frequency')
plt.title('PAM degree sequence')

plt.figure(4)
plt.hist(fit)
plt.xlabel('Fitness')
plt.ylabel('Frequency')
plt.title('PAM fitness sequence')

plt.show()


#hill estimator for gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('PAM degree loglog plot')
```

```
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg), max(deg))
linex3 = []
degj1 = []
degj = np.zeros((max(deg),), dtype=np.float64)
for i in range(1,max(deg)):
    for j in range (0,T):
        if deg[j]==i:
            degj[i-1]=degj[i-1]+1.0
    if degj[i-1]>0:
        linex3.append(i)
        degj1.append(degj[i-1])
degj1=np.array(degj1)
degj2=degj1/T
degj3=np.ones((len(degj1),), dtype=np.float64)
for l in range(1,len(degj1)):
    degj3[l-1]=degj1[l-1]/l

#plt.plot(linex,degj)

plt.figure(7)
plt.hist(degj)
plt.title('PAM Nk-k sequence')

plt.figure(8)
sns.distplot(degj, hist=True, kde=True)
plt.title('PAM Nk-k distribution')

plt.figure(9)
plt.scatter(linex,deg,color='blue',label='PAM degree')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('degrees')
plt.title('PAM degree scatter')

plt.figure(10)
plt.scatter(linex3,degj2,color='blue',label='PAM Nk/T')
logL2=np.log10(linex3)
logB2=np.log10(degj2)
m2, c2 = np.polyfit(logL2, logB2, 1)
y_fit2 = pow(10,m2*logL2 + c2)
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-inf of degrees')
plt.ylabel('Nk/T')
plt.title('PAM Nk/T-k scatter')
print('Nk/T regression slope is ',m2)

plt.figure(11)
plt.scatter(linex,ysort,color='blue',label='PAM degree scatter')
logL=np.log10(linex)
logB=np.log10(ysort)
m1, c1 = np.polyfit(logL, logB, 1)
y_fit = pow(10, m1*logL + c1)
plt.plot(linex, y_fit, ':',color='red',label='regression line')
```

```
164 plt.legend(loc='upper right')
165 plt.yscale('log')
166 plt.xscale('log')
167 plt.xlabel('Nodes at time T')
168 plt.ylabel('Degrees')
169 plt.title('PAM Degree scatter')
170 print('deg regression slope is ',m1)
171
172 plt.figure(12)
173 plt.plot(linex3,degj2,color='blue',label='PAM Nk/T loglog')
174 plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
175 plt.legend(loc='upper right')
176 plt.yscale('log')
177 plt.xscale('log')
178 plt.xlabel('k-degrees')
179 plt.ylabel('Nk/T')
180 plt.title('PAM Nk/T-k loglog plot')
181 #print('Nk/T regression slope is ',m2)
182
183 #visualization of random graph
184 plt.figure(13)
185 nx.draw(G,with_labels=False,edge_color='b',node_color='r',node_size=10)   #
    modification of graph
186 plt.title('PAM graph visualization')
187
188 #height of the tree
189 indh = np.ones((T-1,), dtype=np.int)
190 for m in range(1,T):
191     indh[m-1]=max(ind[0:m])
192 plt.figure(14)
193 plt.plot(indh)
194 plt.xlabel('T-number of iter')
195 plt.ylabel('height of tree')
196 plt.title('PAM Tree Height over Iteration')
197
198
199 plt.figure(16)
200 plt.plot(indh)
201 plt.xscale('log')
202 plt.xlabel('T-number of iter log')
203 plt.ylabel('height of tree')
204 plt.title('PAM Tree Height over Iteration')
205
206 plt.figure(17)
207 plt.scatter(linex,ysort/T,color='blue',label='PAM degree emp scatter')
208 logL1=np.log10(linex)
209 logB4=np.log10(ysort/T)
210 m4, c4 = np.polyfit(logL1, logB4, 1)
211 y_fit4 = pow(10, m4*logL1 + c4)
212 plt.plot(linex, y_fit4, ':',color='red',label='regression line')
213 plt.legend(loc='upper right')
214 plt.yscale('log')
215 plt.xscale('log')
216 plt.xlabel('Nodes of graph at time T')
217 plt.ylabel('Emp Degrees')
218 plt.title('PAM Empirical Degree scatter')
219 print('deg regression slope is ',m4)
220
221
222 plt.figure(19)
223 plt.plot(linex3,degj3,color='blue',label='PAMinv Nk/t loglog')
```

```
logL7=np.log10(linex3[0:10])
logB7=np.log10(degj3[0:10])
m7, c7 = np.polyfit(logL7, logB7, 1)
y_fit7 = pow(10,m7*np.log10(linex3) + c7)
plt.plot(linex3, y_fit7, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('PAMinv Nk/t-k loglog plot')
print('Nk/t-k regression slope is',m7)

plt.show()
```

Appendix C

Code of Random Recursive Tree

Python code of RRT.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 16 21:21:49 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import networkx as nx

##########
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()

#probability to choose any connecting point
prob = np.ones((T+1,), dtype=np.int)
print(prob)

#fitness vector
fit = np.ones((T+1,), dtype=np.float64)
fit[0] = 0.5
print(fit)

ind = np.ones((T+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
```

```
        index = random_picks(seq, [prob[0]])
    else:
        index = random_picks(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    #fit[t+1]=prob[t+1]/deg[t+1]
    fit[index]=prob[index]/deg[index]
    ind[t+1]=ind[index]+1

print('degree sequence is as follows \n', deg)
print('fitness sequence is as follows \n', fit)
print('height of the tree is as follows \n', max(ind))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.xlabel('k-degrees')
plt.ylabel('Density')
plt.title('RRT degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.xlabel('Fitness')
plt.ylabel('Density')
plt.title('RRT fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.xlabel('k-degrees')
plt.ylabel('Frequency')
plt.title('RRT degree sequence')

plt.figure(4)
plt.hist(fit)
plt.xlabel('Fitness')
plt.ylabel('Frequency')
plt.title('RRT fitness sequence')

plt.show()



#hill estimator for gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('RRT degree loglog plot')
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg), max(deg))
linex3 = []
```

```
105 degj1 = []
106 degj = np.zeros((max(deg),), dtype=np.float64)
107 for i in range(1,max(deg)):
108     for j in range (0,T):
109         if deg[j]==i:
110             degj[i-1]=degj[i-1]+1.0
111     if degj[i-1]>0:
112         linex3.append(i)
113         degj1.append(degj[i-1])
114 degj1=np.array(degj1)
115 degj2=degj1/T
116 
117 #plt.plot(linex,degj)
118 
119 plt.figure(7)
120 plt.hist(degj)
121 plt.title('RRT Nk-k sequence')
122 
123 plt.figure(8)
124 sns.distplot(degj, hist=True, kde=True)
125 plt.title('RRT Nk-k distribution')
126 
127 plt.figure(9)
128 plt.scatter(linex,deg,color='blue',label='RRT degree')
129 plt.legend(loc='upper right')
130 plt.yscale('log')
131 plt.xscale('log')
132 plt.xlabel('T-number of iter')
133 plt.ylabel('degrees')
134 plt.title('RRT degree scatter')
135 
136 plt.figure(10)
137 plt.scatter(linex3,degj2,color='blue',label='RRT Nk/T')
138 logL2=np.log10(linex3)
139 logB2=np.log10(degj2)
140 m2, c2 = np.polyfit(logL2, logB2, 1)
141 y_fit2 = pow(10,m2*logL2 + c2)
142 plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
143 plt.legend(loc='upper right')
144 plt.yscale('log')
145 plt.xscale('log')
146 plt.xlabel('k-degrees')
147 plt.ylabel('Nk')
148 plt.title('RRT Nk/T-k scatter')
149 print('Nk/T regression slope is ',m2)
150 
151 plt.figure(11)
152 plt.scatter(linex,ysort,color='blue',label='RRT degree scatter')
153 logL=np.log10(linex)
154 logB=np.log10(ysort)
155 m1, c1 = np.polyfit(logL, logB, 1)
156 y_fit = pow(10, m1*logL + c1)
157 plt.plot(linex, y_fit, ':',color='red',label='regression line')
158 plt.legend(loc='upper right')
159 plt.yscale('log')
160 plt.xscale('log')
161 plt.xlabel('T-number of iter')
162 plt.ylabel('Degrees')
163 plt.title('RRT Degree scatter')
164 print('deg regression slope is ',m1)
165 
```

```
166 plt.figure(12)
167 plt.plot(linex3,degj2,color='blue',label='RRT Nk/T loglog')
168 plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
169 plt.yscale('log')
170 plt.xscale('log')
171 plt.xlabel('k-degrees')
172 plt.ylabel('Nk')
173 plt.title('RRT Nk/T-k loglog plot')
174
175 #visualization of random graph
176 plt.figure(13)
177 nx.draw(G, with_labels=False,edge_color='b',node_color='r',node_size=10)   #
        modification of graph
178 plt.title('RRT graph visualization')
179
180 #height of the tree
181 indh = np.ones((T-1,), dtype=np.int)
182 for m in range(1,T):
183     indh[m-1]=max(ind[0:m])
184 plt.figure(14)
185 plt.plot(indh)
186 plt.xlabel('T-number of iter')
187 plt.ylabel('height of tree')
188 plt.title('RRT Tree Height over Iteration')
189
190 plt.show()
```

Appendix D

Code of Bianconi Barabási Models

Python code of BBM with exponential(2) fitness distribution.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 22 19:33:29 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import networkx as nx

##########
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()


#fitness vector
fit = np.random.exponential(2,T+1)
print(fit)

#probability to choose any connecting point
prob = np.ones((T+1,), dtype=np.float64)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
print(prob)


ind = np.ones((T+1,), dtype=np.int)

#loop over T times
#index = 0
```

```
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_pick(seq, [prob[0]])
    else:
        index = random_pick(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    for j in range(t+1):
        prob[j]=deg[j]*fit[j]
    ind[t+1]=ind[index]+1

print('degree sequence is as follows \n', deg)
print('fitness sequence is as follows \n', fit)
print('height of the tree is as follows \n', max(ind))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.title('BBM_exp2 degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.title('BBM_exp2 fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.title('BBM_exp2 degree sequence')

plt.figure(4)
plt.hist(fit)
plt.title('BBM_exp2 fitness sequence')

plt.show()



#hill estimator for gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('BBM_exp2 degree loglog plot')
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg)+1, max(deg)+1)
degj = np.zeros((max(deg)+1,), dtype=np.int)
for i in range(1,max(deg)+1):
    for j in range (0,T):
        if deg[j]>=i:
            degj[i-1]=degj[i-1]+1

```

```
#plt.plot(linex,degj)

plt.figure(7)
plt.hist(degj)
plt.title('BBM_exp2 Nk-k sequence')

plt.figure(8)
sns.distplot(degj, hist=True, kde=True)
plt.title('BBM_exp2 Nk-k distribution')

plt.figure(9)
plt.scatter(linex,deg,color='blue',label='BBM_exp2 degree')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('degrees')
plt.title('BBM_exp2 degree scatter')

plt.figure(10)
plt.scatter(linex2,degj,color='blue',label='BBM_exp2 Nk')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-inf of degrees')
plt.ylabel('Nk')
plt.title('BBM_exp2 Nk-k scatter')

plt.figure(11)
plt.scatter(linex,ysort,color='blue',label='BBM_exp2 degree scatter')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('Degrees')
plt.title('BBM_exp2 Degree scatter')

plt.figure(12)
plt.loglog(linex2,degj/T,basex=10,basey=10)
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('BBM_exp2 Nk/t-k loglog plot')

#visualization of random graph
plt.figure(13)
nx.draw(G,with_labels=False,edge_color='b',node_color='r',node_size=10)   #
    modification of graph
plt.title('BBM_exp2 graph visualization')

#height of the tree
indh = np.ones((T-1,), dtype=np.int)
for m in range(1,T):
    indh[m-1]=max(ind[0:m])
plt.figure(14)
plt.plot(indh)
plt.xlabel('T-number of iter')
plt.ylabel('height of tree')
plt.title('BBM_exp2 Tree Height over Iteration')

plt.show()
```

Python code of BBM with standard uniform fitness distribution.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 22 19:53:22 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import networkx as nx

#########
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()


#fitness vector
fit = np.random.uniform(0,1,T+1)
print(fit)

#probability to choose any connecting point
prob = np.ones((T+1,), dtype=np.float64)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
print(prob)


ind = np.ones((T+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_pick(seq, [prob[0]])
    else:
        index = random_pick(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    for j in range(t+1):
        prob[j]=deg[j]*fit[j]
    ind[t+1]=ind[index]+1

print('degree sequence is as follows \n', deg)
```

```
print('fitness sequence is as follows \n', fit)
print('height of the tree is as follows \n', max(ind))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.title('BBM_uni degree distributions')

plt.figure(2)
sns.distplot(fit , hist=True, kde=True )
plt.title('BBM_uni fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.title('BBM_uni degree sequence')

plt.figure(4)
plt.hist(fit)
plt.title('BBM_uni fitness sequence')

plt.show()



#hill estimator for gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('BBM_uni degree loglog plot')
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg)+1, max(deg)+1)
degj = np.zeros((max(deg)+1,), dtype=np.int)
for i in range(1,max(deg)+1):
    for j in range (0,T):
        if deg[j]>=i:
            degj[i-1]=degj[i-1]+1

#plt.plot(linex,degj)

plt.figure(7)
plt.hist(degj)
plt.title('BBM_uni Nk-k sequence')

plt.figure(8)
sns.distplot(degj, hist=True, kde=True)
plt.title('BBM_uni Nk-k distribution')

plt.figure(9)
plt.scatter(linex,deg,color='blue',label='BBM_uni degree')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('degrees')
```

```
122 plt.title('BBM_uni degree scatter')
123
124 plt.figure(10)
125 plt.scatter(linex2,degj,color='blue',label='BBM_uni Nk')
126 plt.legend(loc='upper right')
127 plt.yscale('log')
128 plt.xscale('log')
129 plt.xlabel('k-inf of degrees')
130 plt.ylabel('Nk')
131 plt.title('BBM_uni Nk-k scatter')
132
133 plt.figure(11)
134 plt.scatter(linex,ysort,color='blue',label='BBM_uni degree scatter')
135 plt.legend(loc='upper right')
136 plt.yscale('log')
137 plt.xscale('log')
138 plt.xlabel('T-number of iter')
139 plt.ylabel('Degrees')
140 plt.title('BBM_uni Degree scatter')
141
142 plt.figure(12)
143 plt.loglog(linex2,degj/T,basex=10,basey=10)
144 plt.xlabel('k-degrees')
145 plt.ylabel('Nk/t')
146 plt.title('BBM_uni Nk/t-k loglog plot')
147
148 #visualization of random graph
149 plt.figure(13)
150 nx.draw(G,with_labels=False,edge_color='b',node_color='r',node_size=10)   #
      modification of graph
151 plt.title('BBM_uni graph visualization')
152
153 #height of the tree
154 indh = np.ones((T-1,), dtype=np.int)
155 for m in range(1,T):
156     indh[m-1]=max(ind[0:m])
157 plt.figure(14)
158 plt.plot(indh)
159 plt.xlabel('T-number of iter')
160 plt.ylabel('height of tree')
161 plt.title('BBM_uni Tree Height over Iteration')
162
163 plt.show()
```

Appendix E

Code of Recursive Fitness Models

Python code of Plus-1 Model.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 23 10:58:12 2020

@author: wangr
"""

#fitness function +1 model
import pandas as pd
import numpy as np
from numpy import *
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import networkx as nx


#########
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()

#fitness vector
fit = np.ones((T+1,), dtype=np.int)
fit[0] = 1
print(fit)

#probability to choose any connecting point
prob = np.zeros((T+1,), dtype=np.int)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
#prob = prob/np.sum(prob)
print(prob)

ind = np.ones((T+1,), dtype=np.int)
#loop over T times
```

```
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_picks(seq, [prob[0]])
    else:
        index = random_picks(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    fit[t+1]=fit[index]+1
    ind[t+1]=ind[index]+1
    for j in range(t+1):
        prob[j]=deg[j]*fit[j]
    #prob = prob/np.sum(prob)

print('degree sequence is as follows \n', deg)
print('fitness sequence is as follows \n', fit)
print('maximal fitness is as follows \n', max(fit))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.xlabel('degrees')
plt.ylabel('Density')
plt.title('Plus-1 degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.xlabel('Fitness')
plt.ylabel('Density')
plt.title('Plus-1 fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.xlabel('degrees')
plt.ylabel('Frequency')
plt.title('Plus-1 degree sequence')

plt.figure(4)
plt.hist(fit)
plt.xlabel('Fitness')
plt.ylabel('Frequency')
plt.title('Plus-1 fitness sequence')

plt.show()

#hill estimator for gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('Plus-1 degree loglog plot')
#plt.plot(linex,ysort)
```

```
#plt.yscale('log')

linex2 = np.linspace(1,max(deg), max(deg))
linex3 = []
degj1 = []
degj = np.zeros((max(deg),), dtype=np.float64)
for i in range(1,max(deg)):
    for j in range(0,T):
        if deg[j]==i:
            degj[i-1]=degj[i-1]+1.0
    if degj[i-1]>0:
        linex3.append(i)
        degj1.append(degj[i-1])
degj1=np.array(degj1)
degj2=degj1/T

#plt.plot(linex,degj)
plt.figure(6)
#plt.loglog(linex2,degj, basex=10, basey=10)
plt.plot(linex3,degj2)
plt.yscale('log')
#plt.title('Nk loglog plot')
plt.title('Plus-1 Nk-k log plot')

plt.figure(7)
plt.hist(degj)
plt.title('Plus-1 Nk-k sequence')

plt.figure(8)
sns.distplot(degj, hist=True, kde=True)
plt.title('Plus-1 Nk-k distribution')

plt.figure(9)
plt.scatter(linex,deg,color='blue',label='Plus-1 degree')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('degrees')
plt.title('Plus-1 degree scatter')

plt.figure(10)
plt.scatter(linex3,degj2,color='blue',label='Plus-1 Nk/t')
logL2=np.log10(linex3)
logB2=np.log10(degj2)
m2, c2 = np.polyfit(logL2, logB2, 1)
y_fit2 = pow(10,m2*logL2 + c2)
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('Plus-1 Nk/t-k scatter')
print('Nk/T regression slope is ',m2)

plt.figure(11)
plt.scatter(linex,ysort,color='blue',label='Plus-1 degree scatter')
logL=np.log10(linex)
logB=np.log10(ysort)
m1, c1 = np.polyfit(logL, logB, 1)
```

```
y_fit = pow(10, m1*logL + c1)
plt.plot(linex, y_fit, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('Degrees')
plt.title('Plus-1 Degree scatter')
print('deg regression slope is ',m1)

plt.figure(12)
plt.plot(linex3,degj2,color='blue',label='Plus-1 Nk/t loglog')
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('Plus-1 Nk/t-k loglog plot')

#visualization of random graph
plt.figure(13)
nx.draw(G, with_labels=False,edge_color='b',node_color='r',node_size=10)   #
    modification of graph
plt.title('Plus-1 graph visualization')

#height of the tree
indh = np.ones((T-1,), dtype=np.int)
for m in range(1,T):
    indh[m-1]=max(ind[0:m])
plt.figure(14)
plt.plot(indh)
plt.xlabel('T-number of iter')
plt.ylabel('height of tree')
plt.title('Plus-1 Tree Height over Iteration')

plt.figure(15)
plt.scatter(linex,ysort/T,color='blue',label='Plus-1 degree emp scatter')
logL=np.log10(linex)
logB4=np.log10(ysort/T)
m4, c4 = np.polyfit(logL, logB4, 1)
y_fit4 = pow(10, m4*logL + c4)
plt.plot(linex, y_fit4, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('Emp Degrees')
plt.title('Plus-1 Empirical Degree scatter')
print('deg regression slope is ',m4)


plt.show()
```

Python code of Plus-i Model.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar  6 19:33:41 2020

@author: wangr
"""
```

```
7
8  #fitness function +i model
9  import pandas as pd
10 import numpy as np
11 from numpy import *
12 from matplotlib import pyplot as plt
13 import seaborn as sns
14 from random_pick import random_pick, random_picks
15 from collections import Counter
16 import networkx as nx
17
18
19 #########
20 T=2000
21 #degree vector
22 deg = np.zeros((T+1,), dtype=np.int)
23 deg[0] = 2
24 print(deg)
25
26 #initialize the graph
27 G=nx.Graph()
28
29 #fitness vector
30 fit = np.ones((T+1,), dtype=np.int)
31 fit[0] = 1
32 print(fit)
33
34 #probability to choose any connecting point
35 prob = np.zeros((T+1,), dtype=np.int)
36 for i in range(1,T+1):
37     prob[i-1] = deg[i-1]*fit[i-1]
38 #prob = prob/np.sum(prob)
39 print(prob)
40
41 ind = np.ones((T+1,), dtype=np.int)
42
43 #loop over T times
44 #index = 0
45 for t in range(T):
46     seq = list(range(t+1))
47     if t==0:
48         index = random_picks(seq, [prob[0]])
49     else:
50         index = random_picks(seq, prob[0:t])
51
52     print(index)
53     deg[index]=deg[index]+1
54     deg[t+1]=1
55     G.add_node(t+1)
56     G.add_edge(index+1, t+1)
57     fit[t+1]=fit[index]+ index
58     ind[t+1]=ind[index]+1
59     for j in range(t+1):
60         prob[j]=deg[j]*fit[j]
61     #prob = prob/np.sum(prob)
62
63 print('degree sequence is as follows \n', deg)
64 print('fitness sequence is as follows \n', fit)
65 print('height of the tree is as follows \n', max(ind))
66
67 #figures
```

```
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.xlabel('degrees')
plt.ylabel('Density')
plt.title('Plus-i degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.xlabel('Fitness')
plt.ylabel('Density')
plt.title('Plus-i fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.xlabel('degrees')
plt.ylabel('Frequency')
plt.title('Plus-i degree sequence')

plt.figure(4)
plt.hist(fit)
plt.xlabel('Fitness')
plt.ylabel('Frequency')
plt.title('Plus-i fitness sequence')

plt.show()

#hill estimator for gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns
CT = int(T/5)                                             # set the threshold
gamma = 1/(np.mean(np.log(ysort[0:CT]/ysort[CT]))) # get the tail index
print(gamma)

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('Plus-i degree loglog plot')
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg), max(deg))
linex3 = []
degj1 = []
degj = np.zeros((max(deg),), dtype=np.float64)
for i in range(1,max(deg)):
    for j in range(0,T):
        if deg[j]==i:
            degj[i-1]=degj[i-1]+1.0
    if degj[i-1]>0:
        linex3.append(i)
        degj1.append(degj[i-1])
degj1=np.array(degj1)
degj2=degj1/T
n = sum(degj)
degemp=np.ones((len(degj),),dtype=np.float64)
degemp[0]=1-(degj[0]/n)
for i in range(1,len(degj)-1):
    degemp[i]=1-(sum(degj[0:i])/n)

```

```
#plt.plot(linex,degj)
plt.figure(6)
#plt.loglog(linex2,degj, basex=10, basey=10)
plt.plot(linex3,degj2)
plt.yscale('log')
#plt.title('Nk loglog plot')
plt.title('Plus-i Nk-k log plot')

plt.figure(7)
plt.hist(degj)
plt.title('Plus-i Nk-k sequence')

plt.figure(8)
sns.distplot(degj, hist=True, kde=True)
plt.title('Plus-i Nk-k distribution')

plt.figure(9)
plt.scatter(linex,deg,color='blue',label='Plus-i degree')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('degrees')
plt.title('Plus-i degree scatter')

plt.figure(10)
plt.scatter(linex3,degj2,color='blue',label='Plus-i Nk/t')
logL2=np.log10(linex3)
logB2=np.log10(degj2)
m2, c2 = np.polyfit(logL2, logB2, 1)
y_fit2 = pow(10,m2*logL2 + c2)
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('Plus-i Nk/t-k scatter')
print('Nk/T regression slope is ',m2)

plt.figure(11)
plt.scatter(linex,ysort,color='blue',label='Plus-i degree scatter')
logL=np.log10(linex)
logB=np.log10(ysort)
m1, c1 = np.polyfit(logL, logB, 1)
y_fit = pow(10, m1*logL + c1)
plt.plot(linex, y_fit, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('Degrees')
plt.title('Plus-i Degree scatter')
print('deg regression slope is ',m1)

plt.figure(12)
plt.plot(linex3,degj2,color='blue',label='Plus-i Nk/t loglog')
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
```

```
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('Plus-i Nk/t-k loglog plot')

#visualization of random graph
plt.figure(13)
nx.draw(G, with_labels=False, edge_color='b', node_color='r', node_size=10)   #
    modification of graph
plt.title('Plus-i graph visualization')

#height of the tree
indh = np.ones((T-1,), dtype=np.int)
for m in range(1,T):
    indh[m-1]=max(ind[0:m])
plt.figure(14)
plt.plot(indh)
plt.xlabel('T-number of iter')
plt.ylabel('height of tree')
plt.title('Plus-i Tree Height over Iteration')

plt.figure(15)
plt.scatter(linex, ysort/T, color='blue', label='Plus-i degree emp scatter')
logL=np.log10(linex)
logB4=np.log10(ysort/T)
m4, c4 = np.polyfit(logL, logB4, 1)
y_fit4 = pow(10, m4*logL + c4)
plt.plot(linex, y_fit4, ':', color='red', label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('Emp Degrees')
plt.title('Plus-i Empirical Degree scatter')
print('deg regression slope is ',m4)

plt.figure(16)
linex4 = np.linspace(1, len(degemp)-10, len(degemp)-10)
plt.plot(degemp[0:len(degemp)-10], '.', label='Plus-i Degree SF')
logL6=np.log10(linex4)
logB6=np.log10(degemp[0:len(degemp)-10])
m6, c6 = np.polyfit(logL6, logB6, 1)
y_fit6 = pow(10, m6*logL6 + c6)
plt.plot(linex4, y_fit6, ':', color='red', label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('degrees')
plt.ylabel('empirical survival values')
plt.title('Plus-i log-log Degree Survival Function')
print('degemp regression slope is',m6)

plt.show()
```

Python code of Times-2 Model.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar  6 21:54:53 2020

@author: wangr
"""

```

```
#fitness function *2 model
import pandas as pd
import numpy as np
from numpy import *
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import networkx as nx



#########
T=130
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()

#fitness vector
fit = np.ones((T+1,), dtype=np.int)
fit[0] = 1
print(fit)

#probability to choose any connecting point
prob = np.zeros((T+1,), dtype=np.int)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
#prob = prob/np.sum(prob)
print(prob)

ind = np.ones((T+1,), dtype=np.int)


#loop over T times
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_picks(seq, [prob[0]])
    else:
        index = random_picks(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    fit[t+1]=fit[index] * 2
    ind[t+1]=ind[index]+1
    for j in range(t+1):
        prob[j]=deg[j]*fit[j]
    #prob = prob/np.sum(prob)

print('degree sequence is as follows \n', deg)
print('fitness sequence is as follows \n', fit)
print('maximal fitness is as follows \n', max(fit))
print('length of the tree is \n', np.log2(max(fit)))
```

```
print('height of the tree is as follows \n', max(ind))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.xlabel('degrees')
plt.ylabel('Density')
plt.title('Times-2 degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.xlabel('Fitness')
plt.ylabel('Density')
plt.title('Times-2 fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.xlabel('degrees')
plt.ylabel('Frequency')
plt.title('Times-2 degree sequence')

plt.figure(4)
plt.hist(fit)
plt.xlabel('Fitness')
plt.ylabel('Frequency')
plt.title('Times-2 fitness sequence')

ysort = np.sort(deg)[::-1]
#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('Times-2 degree loglog plot')
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg), max(deg))
linex3 = []
degj1 = []
degj = np.zeros((max(deg),), dtype=np.float64)
for i in range(1,max(deg)):
    for j in range(0,T):
        if deg[j]==i:
            degj[i-1]=degj[i-1]+1.0
    if degj[i-1]>0:
        linex3.append(i)
        degj1.append(degj[i-1])
degj1=np.array(degj1)
degj2=degj1/T


#plt.plot(linex,degj)

plt.figure(7)
plt.hist(degj)
plt.title('Times-2 Nk-k sequence')

plt.figure(8)
sns.distplot(degj, hist=True, kde=True)
```

```
130 plt.title('Times-2 Nk-k distribution')
131
132 plt.figure(9)
133 plt.scatter(linex,deg,color='blue',label='Times-2 degree')
134 plt.legend(loc='upper right')
135 plt.yscale('log')
136 plt.xscale('log')
137 plt.xlabel('T-number of iter')
138 plt.ylabel('degrees')
139 plt.title('Times-2 degree scatter')
140
141 plt.figure(10)
142 plt.scatter(linex3,degj2,color='blue',label='Times-2 Nk/t')
143 logL2=np.log10(linex3)
144 logB2=np.log10(degj2)
145 m2, c2 = np.polyfit(logL2, logB2, 1)
146 y_fit2 = pow(10,m2*logL2 + c2)
147 plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
148 plt.legend(loc='upper right')
149 plt.yscale('log')
150 plt.xscale('log')
151 plt.xlabel('k-degrees')
152 plt.ylabel('Nk/t')
153 plt.title('Times-2 Nk/t-k scatter')
154 print('Nk/T regression slope is ',m2)
155
156 plt.figure(11)
157 plt.scatter(linex,ysort,color='blue',label='Times-2 degree scatter')
158 logL=np.log10(linex)
159 logB=np.log10(ysort)
160 m1, c1 = np.polyfit(logL, logB, 1)
161 y_fit = pow(10, m1*logL + c1)
162 plt.plot(linex, y_fit, ':',color='red',label='regression line')
163 plt.legend(loc='upper right')
164 plt.yscale('log')
165 plt.xscale('log')
166 plt.xlabel('Nodes of graph at time T')
167 plt.ylabel('Degrees')
168 plt.title('Times-2 Degree scatter')
169 print('deg regression slope is ',m1)
170
171 plt.figure(12)
172 plt.plot(linex3,degj2,color='blue',label='Times-2 Nk/t loglog')
173 plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
174 plt.legend(loc='upper right')
175 plt.yscale('log')
176 plt.xscale('log')
177 plt.xlabel('k-degrees')
178 plt.ylabel('Nk/t')
179 plt.title('Times-2 Nk/t-k loglog plot')
180
181 #visualization of random graph
182 plt.figure(13)
183 nx.draw(G,with_labels=False,edge_color='b',node_color='r',node_size=10)   #
      modification of graph
184 plt.title('Times-2 graph visualization')
185
186 #height of the tree
187 indh = np.ones((T-1,), dtype=np.int)
188 for m in range(1,T):
189     indh[m-1]=max(ind[0:m])
```

```
190 plt.figure(14)
191 plt.plot(indh)
192 plt.xlabel('T-number of iter')
193 plt.ylabel('height of tree')
194 plt.title('Times-2 Tree Height over Iteration')
195
196 plt.figure(15)
197 plt.scatter(linex,ysort/T,color='blue',label='Times-2 degree emp scatter')
198 logL=np.log10(linex)
199 logB4=np.log10(ysort/T)
200 m4, c4 = np.polyfit(logL, logB4, 1)
201 y_fit4 = pow(10, m4*logL + c4)
202 plt.plot(linex, y_fit4, ':',color='red',label='regression line')
203 plt.legend(loc='upper right')
204 plt.yscale('log')
205 plt.xscale('log')
206 plt.xlabel('Nodes of graph at time T')
207 plt.ylabel('Emp Degrees')
208 plt.title('Times-2 Empirical Degree scatter')
209 print('deg regression slope is ',m4)
210
211
212 plt.show()
```

Appendix F

Code of Inverse Model

Python code of Preferential Attachment Inverse Model.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 10 17:26:11 2020

@author: wangr
"""

import pandas as pd
import numpy as np
from numpy import *
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import scipy.special  as special
import networkx as nx

##########
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#initialize the graph
G=nx.Graph()

#fitness vector
fit = np.zeros((T+1,), dtype=np.float64)
fit[0] = 0.5
print(fit)
fit0 =[] #mark down the fitness sequence over time t of the first node
#probability to choose any connecting point
prob = np.zeros((T+1,), dtype=np.float64)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
#prob = prob/np.sum(prob)
print(prob)

ind = np.ones((T+1,), dtype=np.int)

#loop over T times
```

```
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_pick(seq, [prob[0]])
    else:
        index = random_pick(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    G.add_node(t+1)
    G.add_edge(index+1, t+1)
    #fit[t+1]=prob[t+1]/deg[t+1]
    #fit[index]=prob[index]/deg[index]
    ind[t+1]=ind[index]+1
    for j in range(t+1):
        fit[j]=math.exp(special.gammaln(t+1.5)+special.gammaln(j+1)-special.
            gammaln(t+2)-special.gammaln(j+0.5))
        prob[j]=deg[j]*fit[j]
    fit0.append(fit[0])

print('degree sequence is as follows \n', deg)
print('fitness sequence is as follows \n', fit)
print('height of the tree is as follows \n', max(ind))

#figures
plt.figure(1)
sns.distplot(deg, hist=True, kde=True )
plt.xlabel('degrees')
plt.ylabel('Density')
plt.title('PAMinv degree distributions')

plt.figure(2)
sns.distplot(fit, hist=True, kde=True )
plt.xlabel('Fitness')
plt.ylabel('Density')
plt.title('PAMinv fitness distribution')


#basic histogram
plt.figure(3)
plt.hist(deg)
plt.xlabel('degrees')
plt.ylabel('Frequency')
plt.title('PAMinv degree sequence')

plt.figure(4)
plt.hist(fit)
plt.xlabel('Fitness')
plt.ylabel('Frequency')
plt.title('PAMinv fitness sequence')

plt.show()

#hill estimator for tail index gamma
ysort = np.sort(deg)[::-1]                                  # sort the returns

#log-log plot
linex = np.linspace(1, T+1, T+1)
plt.figure(5)
```

```
plt.loglog(linex,ysort, basex=10, basey=10)
plt.title('PAMinv degree loglog plot')
#plt.plot(linex,ysort)
#plt.yscale('log')

linex2 = np.linspace(1,max(deg), max(deg))
linex3 = []
degj1 = []
degj = np.zeros((max(deg),), dtype=np.float64)
for i in range(1,max(deg)):
    for j in range(0,T):
        if deg[j]==i:
            degj[i-1]=degj[i-1]+1.0
    if degj[i-1]>0:
        linex3.append(i)
        degj1.append(degj[i-1])
degj1=np.array(degj1)
degj2=degj1/T
degj3=np.ones((len(degj1),), dtype=np.float64)
for l in range(1,len(degj1)):
    degj3[l-1]=degj1[l-1]/l

#plt.plot(linex,degj)

plt.figure(7)
plt.hist(degj)
plt.xlabel('k-degrees')
plt.ylabel('Nk number')
plt.title('PAMinv Nk-k sequence')


plt.figure(9)
plt.scatter(linex,deg,color='blue',label='PAMinv degree')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('degrees')
plt.title('PAMinv degree scatter')

plt.figure(10)
plt.scatter(linex3,degj2,color='blue',label='PAMinv Nk/t')
logL2=np.log10(linex3)
logB2=np.log10(degj2)
m2, c2 = np.polyfit(logL2, logB2, 1)
y_fit2 = pow(10,m2*logL2 + c2)
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('PAMinv Nk/t-k scatter')
print('Nk/t regression slope is ',m2)

plt.figure(11)
plt.scatter(linex,ysort,color='blue',label='PAM degree scatter')
logL=np.log10(linex)
logB=np.log10(ysort)
m1, c1 = np.polyfit(logL, logB, 1)
y_fit = pow(10, m1*logL + c1)
```

```
plt.plot(linex, y_fit, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('Degrees')
plt.title('PAMinv Degree scatter')
print('deg regression slope is ',m1)

plt.figure(12)
plt.plot(linex3,degj2,color='blue',label='PAMinv Nk/t loglog')
plt.plot(linex3, y_fit2, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('PAMinv Nk/t-k loglog plot')

#visualization of random graph
plt.figure(13)
nx.draw(G, with_labels=False,edge_color='b',node_color='r',node_size=10)   #
    modification of graph
plt.title('PAMinv graph visualization')

#height of the tree
indh = np.ones((T-1,), dtype=np.int)
for m in range(1,T):
    indh[m-1]=max(ind[0:m])
plt.figure(14)
plt.plot(indh)
plt.xlabel('T-number of iter')
plt.ylabel('height of tree')
plt.title('PAMinv Tree Height over Iteration')



plt.figure(19)
plt.plot(linex3,degj3,color='blue',label='PAMinv Nk/t loglog')
logL2=np.log10(linex3)
logB7=np.log10(degj3)
m7, c7 = np.polyfit(logL2, logB7, 1)
y_fit7 = pow(10,m7*logL2 + c7)
plt.plot(linex3, y_fit7, ':',color='red',label='regression line')
plt.legend(loc='upper right')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t')
plt.title('PAMinv Nk/t-k loglog plot')
print('Nk/t-k regression slope is',m7)

plt.show()
```

Appendix G

Code of Comparisons

Python code of comparisons between PAM and RRT.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 12 23:13:38 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import scipy.special  as special

##########
#RRT model
Tp=10000
#degree vector
degp = np.zeros((Tp+1,), dtype=np.int)
degp[0] = 2
print(degp)


#probability to choose any connecting point
probp = np.ones((Tp+1,), dtype=np.int)
print(probp)

#fitness vector
fitp = np.ones((Tp+1,), dtype=np.float64)
fitp[0] = 0.5
print(fitp)

indp = np.ones((Tp+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(Tp):
    seqp = list(range(t+1))
    if t==0:
        index = random_picks(seqp, [probp[0]])
```

```
    else:
        index = random_picks(seqp, probp[0:t])

    print(index)
    degp[index]=degp[index]+1
    degp[t+1]=1
    fitp[index]=probp[index]/degp[index]
    indp[t+1]=indp[index]+1


#std PAM
Ts=10000
#degree vector
degs = np.zeros((Ts+1,), dtype=np.int)
degs[0] = 2
print(degs)

#fitness vector
fits = np.ones((Ts+1,), dtype=np.int)
fits[0] = 1
print(fits)

#probability to choose any connecting point
probs = np.zeros((Ts+1,), dtype=np.int)
for i in range(1,Ts+1):
    probs[i-1] = degs[i-1]*fits[i-1]
print(probs)

inds = np.ones((Ts+1,), dtype=np.int)
#loop over T times
#index = 0
for t in range(Ts):
    seqs = list(range(t+1))
    if t==0:
        index = random_picks(seqs, [probs[0]])
    else:
        index = random_picks(seqs, probs[0:t])

    print(index)
    degs[index]=degs[index]+1
    degs[t+1]=1
    inds[t+1]=inds[index]+1
    for j in range(t+1):
        probs[j]=degs[j]*fits[j]




#plots
T = max([Tp,Ts])
linex = np.linspace(1, T+1, T+1)
ysortp = np.sort(degp)[::-1]
ysorts = np.sort(degs)[::-1]
#ysortq = np.sort(degq)[::-1]

L = max([max(degp),max(degs)])
linex2 = np.linspace(1,L, L)
degjp = np.zeros((L,), dtype=np.float64)
degjs = np.zeros((L,), dtype=np.float64)

```

```
linexs3 = []
degjs1 = []
for i in range(1,L):
    for j in range (0,T):
        if degs[j]==i:
            degjs[i-1]=degjs[i-1]+1.0
    if degjs[i-1]>0:
        linexs3.append(i)
        degjs1.append(degjs[i-1])
degjs1=np.array(degjs1)
degjs2=degjs1/T


linexp3 = []
degjp1 = []
for i in range(1,L):
    for j in range (0,T):
        if degp[j]==i:
            degjp[i-1]=degjp[i-1]+1.0
    if degjp[i-1]>0:
        linexp3.append(i)
        degjp1.append(degjp[i-1])
degjp1=np.array(degjp1)
degjp2=degjs1/T



plt.figure(2)
plt.loglog(linex2,degjp, basex=10, basey=10, label='RRT')
plt.loglog(linex2,degjs, basex=10, basey=10, label='PAM')
plt.xlabel('log(k)-inf of degrees')
plt.ylabel('log(Nk)-number of nodes deg over k')
plt.legend(loc='upper right')
plt.title('Nk-k log-log plot')

plt.figure(3)
plt.scatter(linex,ysortp,color='blue',label='RRT degree scatter')
plt.scatter(linex,ysorts,color='yellow',label='PAM degree scatter')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Nodes of graph at time T')
plt.ylabel('degrees')
plt.legend(loc='upper right')
plt.title('degree scatter')


plt.figure(5)
plt.plot(linex2,degjp/T, label='RRT')
plt.yscale('log')

plt.plot(linex2,degjs/T, label='PAM')
plt.yscale('log')

plt.xlabel('k-inf of degrees')
plt.ylabel('Nk-number of nodes deg over k')
plt.legend(loc='upper right')
plt.title('Nk/T-k log plot')

plt.figure(6)
plt.loglog(linex2,degjp/T, basex=10, basey=10, label='RRT')
plt.loglog(linex2,degjs/T, basex=10, basey=10, label='PAM')
```

```
plt.xlabel('k-degrees')
plt.ylabel('Nk/t-number of nodes deg over k at time t')
plt.legend(loc='upper right')
plt.title('Nk/t-k log-log plot')

plt.show
```

Python code of comparisons between Benchmarks and Bianconi Barabási Models.

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 30 03:29:20 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math


##########
#RRT model
Tp=10000
#degree vector
degp = np.zeros((Tp+1,), dtype=np.int)
degp[0] = 2
print(degp)


#probability to choose any connecting point
probp = np.ones((Tp+1,), dtype=np.int)
print(probp)

#fitness vector
fitp = np.ones((Tp+1,), dtype=np.float32)
fitp[0] = 0.5
print(fitp)

indp = np.ones((Tp+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(Tp):
    seqp = list(range(t+1))
    if t==0:
        index = random_picks(seqp, [probp[0]])
    else:
        index = random_picks(seqp, probp[0:t])

    print(index)
    degp[index]=degp[index]+1
    degp[t+1]=1
    fitp[index]=probp[index]/degp[index]
    indp[t+1]=indp[index]+1


```

```
#BBMexp2 model
T1=10000
#degree vector
deg1 = np.zeros((T1+1,), dtype=np.int)
deg1[0] = 2
print(deg1)

#fitness vector
fit1 = np.random.exponential(2,T1+1)
print(fit1)

#probability to choose any connecting point
prob1 = np.zeros((T1+1,), dtype=np.float64)
for i in range(1,T1+1):
    prob1[i-1] = deg1[i-1]*fit1[i-1]
print(prob1)

ind1 = np.ones((T1+1,), dtype=np.int)
#loop over T times
#index = 0
for t in range(T1):
    seq1 = list(range(t+1))
    if t==0:
        index = random_pick(seq1, [prob1[0]])
    else:
        index = random_pick(seq1, prob1[0:t])

    print(index)
    deg1[index]=deg1[index]+1
    deg1[t+1]=1
    ind1[t+1]=ind1[index]+1
    for j in range(t+1):
        prob1[j]=deg1[j]*fit1[j]


#BBM std uniform model
Ti=10000
#degree vector
degi = np.zeros((Ti+1,), dtype=np.int)
degi[0] = 2
print(degi)

#fitness vector
fiti = np.random.uniform(0,1,Ti+1)
print(fiti)

#probability to choose any connecting point
probi = np.zeros((Ti+1,), dtype=np.float64)
for i in range(1,Ti+1):
    probi[i-1] = degi[i-1]*fiti[i-1]
print(probi)

indi = np.ones((Ti+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(Ti):
    seqi = list(range(t+1))
    if t==0:
        index = random_pick(seqi, [probi[0]])
    else:
```

```
        index = random_pick(seqi, probi[0:t])

    print(index)
    degi[index]=degi[index]+1
    degi[t+1]=1
    indi[t+1]=indi[index]+1
    for j in range(t+1):
        probi[j]=degi[j]*fiti[j]


#std PAM
Ts=10000
#degree vector
degs = np.zeros((Ts+1,), dtype=np.int)
degs[0] = 2
print(degs)

#fitness vector
fits = np.ones((Ts+1,), dtype=np.int)
fits[0] = 1
print(fits)

#probability to choose any connecting point
probs = np.zeros((Ts+1,), dtype=np.int)
for i in range(1,Ts+1):
    probs[i-1] = degs[i-1]*fits[i-1]
print(probs)

inds = np.ones((Ts+1,), dtype=np.int)
#loop over T times
#index = 0
for t in range(Ts):
    seqs = list(range(t+1))
    if t==0:
        index = random_picks(seqs, [probs[0]])
    else:
        index = random_picks(seqs, probs[0:t])

    print(index)
    degs[index]=degs[index]+1
    degs[t+1]=1
    inds[t+1]=inds[index]+1
    for j in range(t+1):
        probs[j]=degs[j]*fits[j]

#plots
T = max([Tp,Ti,T1,Ts])
linex = np.linspace(1, T+1, T+1)
ysortp = np.sort(degp)[::-1]
ysorti = np.sort(degi)[::-1]
ysort1 = np.sort(deg1)[::-1]
ysorts = np.sort(degs)[::-1]

L = max([max(degp),max(deg1),max(degi),max(degs)])
linex2 = np.linspace(1,L+1, L+1)
degjp = np.zeros((L+1,), dtype=np.int)
degj1 = np.zeros((L+1,), dtype=np.int)
degji = np.zeros((L+1,), dtype=np.int)
degjs = np.zeros((L+1,), dtype=np.int)

for i in range(1,L+1):
```

```
    for j in range (0,Tp):
        if degp[j]>=i:
            degjp[i-1]=degjp[i-1]+1

for i in range(1,L+1):
    for j in range (0,T1):
        if deg1[j]>=i:
            degj1[i-1]=degj1[i-1]+1

for i in range(1,L+1):
    for j in range (0,Ti):
        if degi[j]>=i:
            degji[i-1]=degji[i-1]+1

for i in range(1,L+1):
    for j in range (0,Ts):
        if degs[j]>=i:
            degjs[i-1]=degjs[i-1]+1


plt.figure(2)
plt.loglog(linex2,degjp/Tp,color='blue', basex=10, basey=10, label='RRT')
plt.loglog(linex2,degj1/T1,color='red', basex=10, basey=10, label='BBMexp')
plt.loglog(linex2,degji/Ti,color='green', basex=10, basey=10, label='BBMuni')
plt.loglog(linex2,degjs/Ts,color='yellow', basex=10, basey=10, label='PAM')
plt.xlabel('k-degrees')
plt.ylabel('Nk/t-number of nodes deg over k at time t')
plt.legend(loc='upper right')
plt.title('Nk/t-k log-log plot')

plt.figure(3)
plt.scatter(linex,ysortp,color='blue',label='RRT degree scatter')
plt.scatter(linex,ysort1,color='red',label='BBMexp degree scatter')
plt.scatter(linex[0:len(ysorti)],ysorti,color='green',label='BBMuni degree scatter
    ')
plt.scatter(linex,ysorts,color='yellow',label='PAM degree scatter')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('T-number of iter')
plt.ylabel('degrees')
plt.legend(loc='upper right')
plt.title('degree scatter')

plt.show
```

Python code of comparisons between Benchmarks and Recursive Fitness Models.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  2 16:14:44 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
```

```
16
17
18  #########
19  #RRT model
20  Tp=10000
21  #degree vector
22  degp = np.zeros((Tp+1,), dtype=np.int)
23  degp[0] = 2
24  print(degp)
25
26
27  #probability to choose any connecting point
28  probp = np.ones((Tp+1,), dtype=np.int)
29  print(probp)
30
31  #fitness vector
32  fitp = np.ones((Tp+1,), dtype=np.float32)
33  fitp[0] = 0.5
34  print(fitp)
35
36  indp = np.ones((Tp+1,), dtype=np.int)
37
38  #loop over T times
39  #index = 0
40  for t in range(Tp):
41      seqp = list(range(t+1))
42      if t==0:
43          index = random_picks(seqp, [probp[0]])
44      else:
45          index = random_picks(seqp, probp[0:t])
46
47      print(index)
48      degp[index]=degp[index]+1
49      degp[t+1]=1
50      fitp[index]=probp[index]/degp[index]
51      indp[t+1]=indp[index]+1
52
53
54  #+1 model
55  T1=10000
56  #degree vector
57  deg1 = np.zeros((T1+1,), dtype=np.int)
58  deg1[0] = 2
59  print(deg1)
60
61  #fitness vector
62  fit1 = np.zeros((T1+1,), dtype=np.int)
63  fit1[0] = 1
64  print(fit1)
65
66  #probability to choose any connecting point
67  prob1 = np.zeros((T1+1,), dtype=np.int)
68  for i in range(1,T1+1):
69      prob1[i-1] = deg1[i-1]*fit1[i-1]
70  print(prob1)
71
72  ind1 = np.ones((T1+1,), dtype=np.int)
73  #loop over T times
74  #index = 0
75  for t in range(T1):
76      seq1 = list(range(t+1))
```

```
    if t==0:
        index = random_picks(seq1, [prob1[0]])
    else:
        index = random_picks(seq1, prob1[0:t])

    print(index)
    deg1[index]=deg1[index]+1
    deg1[t+1]=1
    fit1[t+1]=fit1[index]+1
    ind1[t+1]=ind1[index]+1
    for j in range(t+1):
        prob1[j]=deg1[j]*fit1[j]


#+i model
Ti=2000
#degree vector
degi = np.zeros((Ti+1,), dtype=np.int)
degi[0] = 2
print(degi)

#fitness vector
fiti = np.zeros((Ti+1,), dtype=np.int)
fiti[0] = 1
print(fiti)

#probability to choose any connecting point
probi = np.zeros((Ti+1,), dtype=np.int)
for i in range(1,Ti+1):
    probi[i-1] = degi[i-1]*fiti[i-1]
print(probi)

indi = np.ones((Ti+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(Ti):
    seqi = list(range(t+1))
    if t==0:
        index = random_picks(seqi, [probi[0]])
    else:
        index = random_picks(seqi, probi[0:t])

    print(index)
    degi[index]=degi[index]+1
    degi[t+1]=1
    fiti[t+1]=fiti[index]+ index
    indi[t+1]=indi[index]+1
    for j in range(t+1):
        probi[j]=degi[j]*fiti[j]


#std PAM
Ts=10000
#degree vector
degs = np.zeros((Ts+1,), dtype=np.int)
degs[0] = 2
print(degs)

#fitness vector
fits = np.ones((Ts+1,), dtype=np.int)
```

```
fits[0] = 1
print(fits)

#probability to choose any connecting point
probs = np.zeros((Ts+1,), dtype=np.int)
for i in range(1,Ts+1):
    probs[i-1] = degs[i-1]*fits[i-1]
print(probs)

inds = np.ones((Ts+1,), dtype=np.int)
#loop over T times
#index = 0
for t in range(Ts):
    seqs = list(range(t+1))
    if t==0:
        index = random_picks(seqs, [probs[0]])
    else:
        index = random_picks(seqs, probs[0:t])

    print(index)
    degs[index]=degs[index]+1
    degs[t+1]=1
    inds[t+1]=inds[index]+1
    for j in range(t+1):
        probs[j]=degs[j]*fits[j]

#plots
T = max([Tp,Ti,T1,Ts])
linex = np.linspace(1, T+1, T+1)
ysortp = np.sort(degp)[::-1]
ysorti = np.sort(degi)[::-1]
ysort1 = np.sort(deg1)[::-1]
ysorts = np.sort(degs)[::-1]

L = max([max(degp),max(deg1),max(degi),max(degs)])
linex2 = np.linspace(1,L+1, L+1)
degjp = np.zeros((L+1,), dtype=np.int)
degj1 = np.zeros((L+1,), dtype=np.int)
degji = np.zeros((L+1,), dtype=np.int)
degjs = np.zeros((L+1,), dtype=np.int)

for i in range(1,L+1):
    for j in range (0,Tp):
        if degp[j]>=i:
            degjp[i-1]=degjp[i-1]+1

for i in range(1,L+1):
    for j in range (0,T1):
        if deg1[j]>=i:
            degj1[i-1]=degj1[i-1]+1

for i in range(1,L+1):
    for j in range (0,Ti):
        if degi[j]>=i:
            degji[i-1]=degji[i-1]+1

for i in range(1,L+1):
    for j in range (0,Ts):
        if degs[j]>=i:
            degjs[i-1]=degjs[i-1]+1

```

```
199
200 plt.figure(2)
201 plt.loglog(linex2,degjp/Tp,color='blue', basex=10, basey=10, label='RRT')
202 plt.loglog(linex2,degj1/T1,color='red', basex=10, basey=10, label='Plus-1')
203 plt.loglog(linex2,degji/Ti,color='green', basex=10, basey=10, label='Plus-i')
204 plt.loglog(linex2,degjs/Ts,color='yellow', basex=10, basey=10, label='PAM')
205 plt.xlabel('k-degrees')
206 plt.ylabel('Nk/t-number of nodes deg over k at time t')
207 plt.legend(loc='upper right')
208 plt.title('Nk/t-k log-log plot')
209
210 plt.figure(3)
211 plt.scatter(linex,ysortp,color='blue',label='RRT degree scatter')
212 plt.scatter(linex,ysort1,color='red',label='Plus-1 degree scatter')
213 plt.scatter(linex[0:len(ysorti)],ysorti,color='green',label='Plus-i degree scatter
    ')
214 plt.scatter(linex,ysorts,color='yellow',label='PAM degree scatter')
215 plt.yscale('log')
216 plt.xscale('log')
217 plt.xlabel('Nodes of graph at time T')
218 plt.ylabel('degrees')
219 plt.legend(loc='upper right')
220 plt.title('degree scatter')
221
222 plt.show
```

Python code of comparisons between Benchmarks and Inverse Model.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Apr 19 11:25:57 2020
4
5  @author: wangr
6  """
7
8  import pandas as pd
9  import numpy as np
10 import random
11 from matplotlib import pyplot as plt
12 import seaborn as sns
13 from random_pick import random_pick, random_picks
14 from collections import Counter
15 import math
16 import scipy.special  as special
17
18 #########
19 #RRT model
20 Tp=10000
21 #degree vector
22 degp = np.zeros((Tp+1,), dtype=np.int)
23 degp[0] = 2
24 print(degp)
25
26
27 #probability to choose any connecting point
28 probp = np.ones((Tp+1,), dtype=np.int)
29 print(probp)
30
31 #fitness vector
32 fitp = np.ones((Tp+1,), dtype=np.float64)
33 fitp[0] = 0.5
34 #for i in range(1,T+1):
```

```
#     fit[i-1] = prob[i-1]/deg[i-1]
#prob = prob/np.sum(prob)
print(fitp)

indp = np.ones((Tp+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(Tp):
    seqp = list(range(t+1))
    if t==0:
        index = random_picks(seqp, [probp[0]])
    else:
        index = random_picks(seqp, probp[0:t])

    print(index)
    degp[index]=degp[index]+1
    degp[t+1]=1
    fitp[index]=probp[index]/degp[index]
    indp[t+1]=indp[index]+1



#std PAM
Ts=10000
#degree vector
degs = np.zeros((Ts+1,), dtype=np.int)
degs[0] = 2
print(degs)

#fitness vector
fits = np.ones((Ts+1,), dtype=np.int)
fits[0] = 1
print(fits)

#probability to choose any connecting point
probs = np.zeros((Ts+1,), dtype=np.int)
for i in range(1,Ts+1):
    probs[i-1] = degs[i-1]*fits[i-1]
#prob = prob/np.sum(prob)
print(probs)

inds = np.ones((Ts+1,), dtype=np.int)
#loop over T times
#index = 0
for t in range(Ts):
    seqs = list(range(t+1))
    if t==0:
        index = random_picks(seqs, [probs[0]])
    else:
        index = random_picks(seqs, probs[0:t])

    print(index)
    degs[index]=degs[index]+1
    degs[t+1]=1
    inds[t+1]=inds[index]+1
    for j in range(t+1):
        probs[j]=degs[j]*fits[j]


#PAMinv model
```

```
Tm=10000
#degree vector
degm = np.zeros((Tm+1,), dtype=np.int)
degm[0] = 2
print(degm)

#fitness vector
fitm = np.zeros((Tm+1,), dtype=np.float64)
fitm[0] = 0.5
print(fitm)

#probability to choose any connecting point
probm = np.zeros((Tm+1,), dtype=np.float64)
for i in range(1,Tm+1):
    probm[i-1] = degm[i-1]*fitm[i-1]
print(probm)

indm = np.ones((Tm+1,), dtype=np.int)

#loop over T times
#index = 0
for t in range(Tm):
    seqm = list(range(t+1))
    if t==0:
        index = random_pick(seqm, [probm[0]])
    else:
        index = random_pick(seqm, probm[0:t])

    print(index)
    degm[index]=degm[index]+1
    degm[t+1]=1
    indm[t+1]=indm[index]+1
    for j in range(t+1):
        fitm[j]=math.exp(special.gammaln(t+1.5)+special.gammaln(j+1)-special.
            gammaln(t+2)-special.gammaln(j+0.5))
        probm[j]=degm[j]*fitm[j]



#plots
T = max([Tp,Tm,Ts])
linex = np.linspace(1, T+1, T+1)
ysortp = np.sort(degp)[::-1]
ysortm = np.sort(degm)[::-1]
ysorts = np.sort(degs)[::-1]

L = max([max(degp),max(degm),max(degs)])
linex2 = np.linspace(1,L+1, L+1)
degjp = np.zeros((L+1,), dtype=np.int)
degjm = np.zeros((L+1,), dtype=np.int)
degjs = np.zeros((L+1,), dtype=np.int)

for i in range(1,L+1):
    for j in range (0,Tp):
        if degp[j]>=i:
            degjp[i-1]=degjp[i-1]+1

for i in range(1,L+1):
    for j in range (0,Tm):
        if degm[j]>=i:
```

```
156             degjm[i-1]=degjm[i-1]+1
157
158
159 for i in range(1,L+1):
160     for j in range (0,Ts):
161         if degs[j]>=i:
162             degjs[i-1]=degjs[i-1]+1
163
164
165 plt.figure(1)
166 plt.plot(linex2,degjp, label='RRT')
167 plt.yscale('log')
168
169 plt.plot(linex2,degjm, label='PAMinv')
170 plt.yscale('log')
171
172 plt.plot(linex2,degjs, label='PAM')
173 plt.yscale('log')
174
175
176 plt.xlabel('k-inf of degrees')
177 plt.ylabel('log(Nk)-number of nodes deg over k')
178 plt.legend(loc='upper right')
179 plt.title('Nk-k log plot')
180
181 plt.figure(2)
182 plt.loglog(linex2,degjp/Tp,color='blue', basex=10, basey=10, label='RRT')
183 plt.loglog(linex2,degjm/Tm,color='red', basex=10, basey=10, label='PAMinv')
184 plt.loglog(linex2,degjs/Ts,color='yellow', basex=10, basey=10, label='PAM')
185 plt.xlabel('k-degrees')
186 plt.ylabel('Nk/t-number of nodes deg over k at time t')
187 plt.legend(loc='upper right')
188 plt.title('Nk/t-k log-log plot')
189
190 plt.figure(3)
191 plt.scatter(linex,ysortp,color='blue',label='RRT degree scatter')
192 plt.scatter(linex,ysortm,color='red',label='PAMinv degree scatter')
193 plt.scatter(linex,ysorts,color='yellow',label='PAM degree scatter')
194 plt.yscale('log')
195 plt.xscale('log')
196 plt.xlabel('Nodes of graph at time T')
197 plt.ylabel('degrees')
198 plt.legend(loc='upper right')
199 plt.title('degree scatter')
200
201 plt.show
```

Appendix H

Code of Preferential Attachment Inverse Model Simulations

Python code of $\frac{D_t(j)}{\mathbb{E}[d_t(j)]}$ *sequence.*

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 25 20:35:07 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import scipy.special  as special

#########
#numerator simulation PAMinv
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2

#fitness vector
fit = np.zeros((T+1,), dtype=np.float64)
fit[0] = 0.5

#probability to choose any connecting point
prob = np.zeros((T+1,), dtype=np.float64)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]

#loop over T times
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_pick(seq, [prob[0]])
    else:
        index = random_pick(seq, prob[0:t])

```

```
    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    for j in range(t+1):
        fit[j]=math.exp(special.gammaln(t+1.5)+special.gammaln(j+1)-special.
            gammaln(t+2)-special.gammaln(j+0.5))
        prob[j]=deg[j]*fit[j]

#plots
plt.figure(1)
plt.plot(range(1,T+1),prob[0:T],'.',color='blue',label='D_t(j)/E[d_t(j)]')
m1, c1 = np.polyfit(range(1,T+1), prob[0:T], 1)
y_fit = m1*range(1,T+1) + c1
plt.plot(range(1,T+1), y_fit, ':',color='red',label='regression line')
plt.xlabel('nodes')
plt.ylabel('D_t(j)/E[d_t(j)]')
plt.legend(loc='upper right')
plt.title('Numerator trend')
plt.show
print(m1,c1)

plt.figure(2)
plt.plot(range(1,T+1),prob[0:T],color='blue',label='D_t(j)/E[d_t(j)]')
m2, c2 = np.polyfit(range(1,T+1), prob[0:T], 1)
y_fit2 = m2*range(1,T+1) + c2
plt.plot(range(1,T+1), y_fit2, ':',color='red',label='regression line')
plt.xlabel('nodes')
plt.ylabel('D_t(j)/E[d_t(j)]')
plt.legend(loc='upper right')
plt.title('Numerator trend')
plt.show
print(m2,c2)
```

Python code of Z_t sequence.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 24 14:53:29 2020

@author: wangr
"""

import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import scipy.special  as special

#########
#Z_t simulation PAMinv
N = 3000
Zt = np.zeros((N-100,), dtype=np.float64)
Kt = np.zeros((N-100,), dtype=np.float64)

for T in range(100,N):
    #degree vector
    deg = np.zeros((T+1,), dtype=np.int)
    deg[0] = 2
```

```
28
29      #fitness vector
30      fit = np.zeros((T+1,), dtype=np.float64)
31      fit[0] = 0.5
32
33      #probability to choose any connecting point
34      prob = np.zeros((T+1,), dtype=np.float64)
35      for i in range(1,T+1):
36          prob[i-1] = deg[i-1]*fit[i-1]
37
38      #loop over T times
39      for t in range(T):
40          seq = list(range(t+1))
41          if t==0:
42              index = random_pick(seq, [prob[0]])
43          else:
44              index = random_pick(seq, prob[0:t])
45
46          print(index)
47          deg[index]=deg[index]+1
48          deg[t+1]=1
49          for j in range(t+1):
50              fit[j]=math.exp(special.gammaln(t+1.5)+special.gammaln(j+1)-special.
                    gammaln(t+2)-special.gammaln(j+0.5))
51              prob[j]=deg[j]*fit[j]
52
53      Zt[T-100]=sum(prob)
54      Kt[T-100]=Zt[T-100]/T
55
56  #plots
57  plt.figure(1)
58  plt.plot(range(100,N),Kt,color='blue',label='Zt/t')
59  m1, c1 = np.polyfit(range(100,N), Kt, 1)
60  y_fit = m1*range(100,N) + c1
61  plt.plot(range(100,N), y_fit, ':',color='red',label='regression line')
62  plt.xlabel('iterations')
63  plt.ylabel('Zt/t')
64  plt.legend(loc='upper right')
65  plt.title('Zt trend')
66  plt.show
67
68  plt.figure(2)
69  plt.plot(range(100,N),Kt,'.',color='blue',label='Zt/t')
70  m2, c2 = np.polyfit(range(100,N), Kt, 1)
71  y_fit2 = m2*range(100,N) + c2
72  plt.plot(range(100,N), y_fit2, ':',color='red',label='regression line')
73  plt.xlabel('iterations')
74  plt.ylabel('Zt/t')
75  plt.legend(loc='upper right')
76  plt.title('Zt trend')
77  plt.show
```

Python code of maximal eigenvalue sequence and corresponding eigenvectors.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat May  9 19:56:22 2020
4
5  @author: wangr
6  """
7
8  import pandas as pd
```

```
import numpy as np
import random
from matplotlib import pyplot as plt
import seaborn as sns
from random_pick import random_pick, random_picks
from collections import Counter
import math
import scipy.special  as special

#########
#eigen A_ij PAMinv
T=10000
#degree vector
deg = np.zeros((T+1,), dtype=np.int)
deg[0] = 2
print(deg)

#fitness vector
fit = np.zeros((T+1,), dtype=np.float64)
fit[0] = 0.5
print(fit)
#probability to choose any connecting point
prob = np.zeros((T+1,), dtype=np.float64)
for i in range(1,T+1):
    prob[i-1] = deg[i-1]*fit[i-1]
#prob = prob/np.sum(prob)
print(prob)

ind = np.ones((T+1,), dtype=np.int)

#evalues=np.zeros((T+1,), dtype=np.float64)    #set up eigenvalue sequence
#evectors=np.zeros((T+1,), dtype=np.float64)    #set up eigenvector sequence
mevalues=[]                                    #largest eigenvalue
loca = []                                      #location of the largest eigenvalue

#loop over T times
#index = 0
for t in range(T):
    seq = list(range(t+1))
    if t==0:
        index = random_pick(seq, [prob[0]])
    else:
        index = random_pick(seq, prob[0:t])

    print(index)
    deg[index]=deg[index]+1
    deg[t+1]=1
    ind[t+1]=ind[index]+1
    for j in range(t+1):
        fit[j]=math.exp(special.gammaln(t+1.5)+special.gammaln(j+1)-special.
            gammaln(t+2)-special.gammaln(j+0.5))
        prob[j]=deg[j]*fit[j]
    A = np.zeros((max(deg)+1,max(deg)+1),dtype=np.float64)    #set an empty matrix
         for transformation
    aa = np.zeros((max(deg)+1,), dtype=np.float64)    #coefficient times the
        expectation of increment matrix
    Ndeg=np.ones((max(deg)+1,), dtype=np.int)         #set an empty sequence of
        number of bertices with degree k time-updating
    for k in range(max(deg)+1):
        if len(np.where(deg==k)[0])>0:
            for n in np.where(deg==k)[0]:
```

```
                aa[k]=aa[k] + prob[n]
            print(aa[k],k)

        Ndeg[k]=len(np.where(deg==k)[0])

    for i in range(max(deg)+1):
        for j in range(max(deg)+1):
            if i==0:
                A[i,0]=2
            elif i==max(deg):
                A[i,0]=1
                A[i,max(deg)]=1
            else:
                A[i,0]=1
                A[i,i]=-Ndeg[i]
                A[i,i+1]=Ndeg[i]+1
        A[i]=A[i]*aa[i]

    w,v=np.linalg.eig(A)
    print('eigenvalues are \n', w)
    print('eigenvectors are \n', v)
    print('maximal eigenvalue is \n', max(w))
    print('max eigenvalue is located \n', int(np.where(w==max(w))[0][0]))
    print('corresponding eigenvector is \n', v[:,int(np.where(w==max(w))[0][0])])
    mevalues.append(max(w))
    loca.append(int(np.where(w==max(w))[0][0]))


###
print('largest eigenvalues sequence is \n', mevalues)
print('largest eigenvalues locations are \n', loca)


#plots
plt.figure(1)
plt.plot(mevalues, label='Max eigenvalues')
plt.xlabel('iteration numbers')
plt.ylabel('maximal eigenvalues')
plt.legend(loc='upper right')
plt.title('PAMinv Matrix A Eigenvalues Trend')
plt.show
```

Bibliography

[1] Gamma function asymptotic expansions. [EB/OL]. https://dlmf.nist.gov/5.11.

[2] Louigi Addario-Berry and Laura Eslava. High degrees in random recursive trees. *Random Structures & Algorithms*, 52(4):560–575, 2018.

[3] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

[4] Ginestra Bianconi and A-L Barabási. Competition and multiscaling in evolving networks. *EPL (Europhysics Letters)*, 54(4):436, 2001.

[5] Christian Borgs, Jennifer Chayes, Constantinos Daskalakis, and Sebastien Roch. First to market is not everything: an analysis of preferential attachment with fitness. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 135–144, 2007.

[6] Guido Caldarelli, Andrea Capocci, Paolo De Los Rios, and Miguel A Munoz. Scale-free networks from varying vertex intrinsic fitness. *Physical review letters*, 89(25):258702, 2002.

[7] Alessandra Cipriani and Andrea Fontanari. Dynamical fitness models: evidence of universality classes for preferential attachment graphs. *arXiv preprint arXiv:1911.12402*, 2019. https://arxiv.org/abs/1911.12402.

[8] Sander Dommers, Remco Van Der Hofstad, and Gerard Hooghiemstra. Diameters in preferential attachment models. *Journal of Statistical Physics*, 139(1):72–107, 2010.

[9] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[10] Nicolas Fraiman, Omar Fawzi, and Luc Devroye. The height of scaled attachment random recursive trees. *Discrete Mathematics & Theoretical Computer Science*, 2010.

[11] Svante Janson. Functional limit theorems for multitype branching processes and generalized pólya urns. *Stochastic Processes and their Applications*, 110(2):177–245, 2004.

[12] Svante Janson. Asymptotic degree distribution in random recursive trees. *Random Structures & Algorithms*, 26(1-2):69–83, 2005.

[13] Boris Pittel. Note on the heights of random recursive trees and random m-ary search trees. *Random Structures & Algorithms*, 5(2):337–347, 1994.

[14] Robert J Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, pages 39–48, 1974.

[15] Remco Van Der Hofstad. *Random graphs and complex networks*, volume 1. Cambridge university press, 2016.

[16] George Udny Yule. A mathematical theory of evolution, based on the conclusions of dr. jc willis, fr s. *Philosophical transactions of the Royal Society of London. Series B, containing papers of a biological character*, 213(402-410):21–87, 1925.