Synthetic Probabilistic Temporal Networks

Deep Kiran Inamdar

May 03, 2021

Contents

1. Abstract

An analysis over a real-world network is ideal for spotting vital observations and quoting implications. Most of the time, these are computationally heavy, demanding a small world synthetic representation of the real-world network. The critical expectation here is to reproduce a small-scale snapshot of an existing network preserving the properties of the same. There have been efforts to address this problem and come up with conceptual models of generating synthetic probabilistic networks. The focus of these models is to reproduce the behavior of a specific real-world class of networks. Thus, a network generated using an existing model will serve to analyze a particular set of networks depicting certain behaviors. Moreover, most of the existing synthetic models focus on constructing static networks and not temporal networks.

We hereby propose a model to generate synthetic temporal networks probabilistically - SPTN. Contrary to the existing network generators, SPTN focuses on building networks having suitable properties as per demand instead of being specific to a typical set of networks with stringent properties. SPTN can generate undirected, directed, and weighted graphs. Moreover, the probabilities may not be constant values but are functions dependent on graph indicators. Implementation in R backs the proposed model, and many experiments are performed over this model. The paper explains the concept of SPTN and its implementation. Further, we reproduce a few of the existing probabilistic models and do static and temporal analysis to verify the results.

2. Introduction

In the real world, the evolution of the network is very dynamic. There are tons of dependencies on which the behavior of the network changes with time. The dependencies vary depending on the domain of the network. For instance, a social network has sociological dependencies, whereas a pandemic has dependencies over the nature of the virus. Having these various dependencies to fit in a synthetic model is impractical and challenging. Most of the existing graph generator models neglect these dependencies and have constant probabilistic values that limit the reproduction of synthetic real-world networks. The well-known Erdos Renyi model can produce a network with a Poisson degree distribution different from an Internet network. Hence, Erdos Renyi cannot be used to analyze the Internet network. Here is where Barabasi or R-MAT come into the picture. Apart from well-known networks with typical network properties, other networks might have unique and vague combinations of properties. The existing models simply cannot reproduce these networks. Hence, there is a need for a model which generates graphs as per demand. While we achieve this flexibility, there should be a generalized structure that incorporates all types of networks. In this paper, we propose, build, and compare our proposed model for the same.

The rest of the paper is organized as follows – section 3 discusses the background study and related work on this topic. Section 4 explains in detail the proposed model and its implementation. Section 5 provides empirical data of simulating Erdos Renyi Barabasi Albert, R-MAT networks using the proposed model. We do static and temporal analysis on these generated networks to check the correctness of the results. We also reproduce Planted Cliques and Signed networks, do a theoretical study, and propose incorporating them in the proposed model. The paper ends with conclusions and future scope in sections 6 and 7, respectively.

3. Background and Related Work

There exists plenty of static graph generators such as Erdos Renyi, R-MAT, and BRITE. Erdos Renyi provides single parameter flexibility, which is edge creation probability. This single probability taken as input is also a constant. Still, this model is a robust model with many mathematical patterns sited and resembles a few of the many real-world networks. R-MAT addresses the creation of graphs closer to real-world networks having a typical non-Poisson power-law degree distribution. R-MAT produces graphs which resemble Internet network. BRITE comes with a concept of providing a geometric layout of the nodes. Whereas Barabasi Albert comes with the idea of preferential attachment while constructing the network. All these models have a unique procedural strategy to construct graphs. For all these models, the result of the graph is kept under the spotlight. This makes them useful only for static analysis. Networks generated via any of the above models have a typical set of properties studied from various perspectives. The significance of these models is judged by the closeness of the generated graphs to real-world networks. A real-world network is often analyzed to bucket the findings in one of the many synthetic graph generator models. Another branch of research focuses on improving the efficiency of the algorithms used for constructing these models.

Another background study was done in the fields of sampling of networks, synthetic model generation, signed networks, static and temporal analysis of real-world networks.

4. Proposed Model

We need a model that can generate temporal networks and have specific properties as per demand from the background study. The properties may change or be constant for each snapshot of the temporal network. The model should be generic enough to accommodate all types of graphs. The probabilities should not be constant but must be dependent on graph indicators by which our model can simulate real-world networks correctly. To achieve all of this, the control should be delegated to the end-user if our model is supposed to generate a network tailored as per the user's demands. Hence, we introduce a functional approach towards constructing a probabilistic model. These functional inputs to the SPTN are referred to as hyperparameters throughout this paper.

4.1 Conceptual Model

Observing a graph, we know that there are only two entities – Nodes and Edges at ground level. If we have control over them, we can create any sort of graph. When we consider the evolution of a network with time, we will have to consider the following aspects nodes and edges.

a) Node Creation Rate – The number of nodes created per unit time.
b) Node Life – Depicts how long a node is going to stay in the network.
c) Edge creation probability – Given two vertices, the probability they get connected.
d) Edge deletion probability – Given two vertices, the probability they get disconnected.
e) Edge weight – Depending on the status of graph/vertices, the weight of the edge. Relevant for weighted graphs.

These parameters may vary as the network evolves. Typically, in real life, these hyperparameters depend on time, graph properties, node properties, and edge properties. Thus, unlike existing synthetic models, we cannot have constant value inputs for these parameters. Hence, we keep them as functions. Keeping these parameters as functions gives us the ability to tweak them based on their dependencies to simulate networks closer to the real world. The SPTN model we propose takes these five hyperparameters as inputs.

4.2 Model structure

Appendix-A Table-1 illustrates the function call to the SPTN model. Notice that, having the proposed structure, we can input the functions tailored to the domain knowledge of the graph being generated. We can set the functions so that they behave differently as the temporal network evolves with time. We can add custom graph attributes that can be used in the functions. Every node can be treated uniquely, unlike other models where every node is treated alike. Also, we can enter custom logic for each function surpassing the inbuilt logic of the proposed model. We can do this by tweaking the function's return value to the next instance of the graph. If the function's return value is a graph object, the model implicitly knows that the logic is handled by the end-user and skips on the internal implementation. Appendix-A Table-1 explains in detail the working of the function call.

4.3 Internal Working of SPTN

For every time t, ranging from t_start to t_end, the following procedure is followed –

1. deleteNode – Internally calls the user specified functional hyperparameter NLF (G,v,t). This executes the logic written by the end-user overall existing nodes to decide whether a node should be deleted or not. Best Time complexity O (n^2) where n is the number of vertices at given time t.
2. addNode – Internally calls the functional parameter NCR (G, t). This executes the logic written by the end-user to add nodes, and depending on the return value of the same, the required number of nodes are added to the network. Best time complexity O (1).
3. addDeleteEdge – Implicitly calls the edge creation function ECF (G, v1, v2, t) and edge deletion function EDF (G, v1, v2, t) one after the other for every combination of vertices. The end-user and SPTN enter the logic for this to abides by this logic to decide on the existence of an edge. Best time complexity O (n^2).
4. The SPTN temporalNetwork () function call returns the meta-data of temporal graph snapshots for equal frequency-time periods. This meta-data object consists of all the instances of the temporal graphs and static indicators of each graph. This meta-data object is the synthetically generated temporal network dataset on which static/temporal network analysis can be performed.
5. Steps 1 to 3 are repeated for every time instance.

Note that there arise instances when the internal logic needs to be surpassed and be wrapped by user-suggested logic. In this case, the functional hyperparameters should return Graph objects rather than probabilistic values or Integers. This indicates to the SPTN system that the logic is being handled at the user's end, and the internal implementation for that function is skipped. We had to do this to reproduce the R-MAT model.

4.4 Hypothesis

a. We hypothesis that these five hyperparameters are the only necessary and sufficient parameters to be controlled in a temporal network that evolves with time. Any temporal network can be fitted in functional input parameters and can be reproduced synthetically by SPTN.
b. The proposed dependencies of hyperparameter functions are the only dependencies required. E.g., Edge creation will only depend on graph attributes and indicators G, the two vertices v1 and v2, and time t.
c. SPTN can mimic any existing random graph generator by tweaking the hyperparameters.

5. Implementation and analysis

Now that we have a function call to SPTN, we used this model to construct temporal networks having different properties. In this section we try simulating known graph generator models and compare the results doing temporal and static analysis wherever required. Every model which we select to reproduce is explained in the following subsections. Each subsection describes the following –

a. The intention behind generating the specific network.
b. The process to generate the specific model – function call.
c. Static/Temporal analysis to validate results.
d. Implications of achieving the showcased results.

5.1 Example Network

To get familiarized with SPTN, let us try constructing a dummy social network model. The intention of demonstrating the construction of the network is to have an awareness of the capabilities of SPTN and verify the results with respect to the inputs. In this example, we consider a node a person and an edge as a contact/connection. Appendix A – Table 2 mentions the function call hyperparameters used to generate the undirected, unweighted network. Notice that the functions depend on the static indicators such as degree of a node, age of a node, time, etc. These functions can be as complicated or straightforward as possible. The takeaway from the example is the flexibility that SPTN provides for constructing a temporal network.

Doing a temporal analysis on a few of the standard static graph indicators, we get a picture of how the network evolved, abiding by the hyperparameters we gave as input. Appendix B – Figure 1 shows the temporal analysis plots for the example network created. For instance, the curve for node count is exponential since we had put in an exponential function for node creation. The maximum degree increases as the network grows, but the average path length is saturated to 7, which is typically the case for most social networks. Appendix B – Figure 2 shows how the network generated by SPTN evolves with time from t=10 to t=50. Few clusters and multiple components are observed in the network over time.

Now that we have an idea of how SPTN works, we move on to more exciting topics to compare and reproduce existing network generator models using SPTN. This is a topic of interest since it suggests that SPTN can generate existing classic models. It also implies that SPTN is flexible to incorporate the logic of these models, which backs our hypothesis. Hence, the overall study hints towards SPTN being a powerful tool to mimic existing and plausible future models. Mimicking the synthetic models ultimately suggests that the real-world networks showcasing properties of a particular model can be generated via SPTN and used to perform analysis.

5.2 Erdos Renyi

Erdos Renyi is the classic and first-of-its-kind static network generator. The model is adored for its simplicity and is exhaustively studied mathematically to find interesting patterns. One crucial characteristic of the Erdos Renyi random graph is the Poisson degree distribution.

We created an Erdos Renyi undirected, unweighted graph with 1000 nodes and a 0.1 edge creation probability in this experiment. We aimed to develop a temporal Erdos Renyi network using SPTN where at any time t, the snapshot of the graph resembles an Erdos Renyi graph. We ran the SPTN from t=0 to t=50 such that the snapshot at t=50 has 1000 nodes. We compared the generated graph with the actual Erdos Renyi graph generated via the Erdos Renyi game.

Appendix A – Table 3 gives the input functions for the SPTN function call used to reproduce Erdos Renyi graphs. The primary function was the edge creation function, where we gave a probability of 0.1. Since the internal working of SPTN had a scan on every pair of vertices, we did the same in steps for the pairs consisting of at least one new node. This ensured that the overall process followed replicates the Erdos Renyi graph generation process. We did a static analysis on the temporal instances of the network. Appendix A – Table 4 shows the results. We observe that the degree distribution for all the snapshots is Poisson. Also, in the static analysis, the numeric values (Especially the clustering coefficients) match the Erdos Renyi game graph. The patterns for diameter and average path length also have the same values throughout the temporal evolution of the network.

Hence, we conclude that SPTN could successfully reproduce Erdos Renyi graphs. This implies that any real-world networks with the characteristics of Erdos-Renyi graphs can be synthetically modeled using SPTN. Further temporal and static analysis can be performed on the small-world network representation of the same.

We can also reproduce the Erdos Renyi graphs with few changes in the implementation by inputting the number of edges instead of edge creation probability. In this case, we can keep choosing a random index in the square adjacency matrix and add (number of edges/t) number of edges per unit time. This will give us the same result.

5.3 Barabasi Albert

Barabasi Albert is the model which we found close to temporal network generation. Though it is not precisely a temporal network generator as it outputs only one graph, the process mimics temporal procedure. The Barabasi Albert model is known to produce networks that abide by power laws.

In the Barabasi Albert model, at every time t, a new node is added. At every time t, a new node connects with the existing nodes to create exactly kedges. The vertex with which the newly created node connects depends on the power of the preferential attachment (p) and the zero appeal (z). Zero appeal is the attractiveness of a node when the in-degree is 0.

In general, the probability that the new node connects to a node j is –

$\Pi(j) = (p_j*k[j]+z)/\sum(p*k[i]+z)$ …where $\pi(j)$ is the probability the new node connects to node j,

p_j is the power, k[j] is the indegree of node j, & z is zero appeal.

For p=1, the network generated abides power law and is scale-free where the degree distribution is linear on a log-log scale.

For p<1, the degree distribution is sub-linear.

For p>1, the degree distribution is super-linear.

Barabasi Albert generated graphs have vital importance because they resemble the characteristics of a real-world network such as the Internet. But the limitation of Barabasi Albert is that they can only give power laws of exponent 3. In Barabasi networks, at any given time t, the number of nodes = t and number of edges = k*t.

We tried reproducing the Barabasi Albert networks. We aimed to set up the model so that every snapshot of the temporal network will characterize a Barabasi Albert graph. We had to tweak the earlier defined function call for SPTN to include one more parameter, which is fixedEdgeCreation. This parameter will have an integral value that resembles the number of edges added for every time instance. By default, it is -1, which means there is no restriction on the exact number of edges to be added.

We generated 3 Barabasi Albert graphs as follows:

1. 500 nodes, p (power) =1, m (out-degree) =2 and z (zero appeal) =1.
2. 500 nodes, p = 0.5, m =2, z =1.
3. 500 nodes, p = 1.5, m =2, z =1.

We triggered the SPTN model with the same input parameters and wrote the Barabasi logic in the edge creation function to produce corresponding three directed graphs. We plotted the degree distribution on a log-log scale for all these six graphs. Having three separate graphs for varying values of p was to do the preferential attachment analysis. For p<1, p=1, and p>1, we get graphs with sub-linear, linear, and super-linear curves of the degree distribution on a log-log scale. The results which we achieved by doing this experiment are listed below.

Appendix A – Table 5 lists the input functions used in the SPTN function call. Appendix B – Figure 4 shows the degree distribution curves. We can observe that for p=1 the degree distribution is linear for both the SPTN and the Barabasi game generated graphs. Not much visible difference can be noticed for sub-linear and linear graphs, but this is due to the low number of nodes for which we ran the model.

The implications of the reproducing Barabasi Albert graph suggest the power of the SPTN tool to generate a beneficial class of graphs that too temporally. This can be used to analyze some profound real-world networks such as the Internet.

5.4 R-MAT (Recursive Matrix)

The R-MAT model is another attractive synthetic model to generate networks with a unique set of properties that major real-world networks showcase. An R-MAT generated graph aims to have the graph match the degree distribution (power-law or not), the diameter and exhibit the community structure (clustering coefficients and components). Also, the R-MAT focuses on proposing an optimal algorithm to generate the graph.

The concept of the R-MAT model is simple. For a given N x N adjacency matrix, we must add K edges. The logic behind adding the Kedges is probabilistic. The adjacency matrix is divided into four equal sections [a, b, c, d]. Each section is given a probability such that P(a)>P(b)>=P(c)>=P(d) and a + b + c + d = 1. For every edge to be added, a section is chosen probabilistically. Then the chosen section is recursively subdivided into four sections, and the process continues till we land on an index of the adjacency matrix and add the edge. This process is followed for all the kedges. It has been observed that the R-MAT generated graph reproduces power-law degree distribution. For the case when the probabilities are set equal to 0.25, it produces graphs characterizing Erdos Renyi.

We used SPTN to reproduce R-MAT characterized graphs. Appendix A – Table 6 gives the function call for producing R-MAT graphs. We aimed to create a temporal network such that every snapshot of it will resemble the characteristics of R-MAT. We did the degree distribution analysis for probabilities [0.4,0.2,0.2,0.2] to see if the distribution is Power law. Also, we did the degree distribution analysis for all probabilities set to 0.25 to see if it produces Poisson distribution. Appendix B – Figure 5 has the degree distribution plots for both the configurations of the probabilities. It is clear from the plots that the degree distributions obey power law and Poisson in the two plots.

It is observed that the degree distributions for unequal probabilities were power law, and for equal probabilities, it was Poisson. This indicates that SPTN is successfully able to reproduce R-MAT graphs. This implies that SPTN can synthetically simulate the temporal evolution of the Internet network and further analysis. Other networks which showcase power-law distributions can also be studied using SPTN.

5.5 Planted Cliques

In graph theory, a clique is defined as a subset of vertices, or subgraph, in which every vertex is adjacent, also known as complete subgraphs. A planted clique is created from inside a graph by adding edges between all vertices in that subset. Formally, there is a decision process, meaning that this is not a random distribution; however, these graphs can be modeled using our synthesis based on the following algorithm:

1. Generate an Erdos Renyi model, as we did above, but we decide with a 100% probability that an edge will be created between them for each pair of vertices.
2. Over time we will see that each pair of vertices will be connected by an edge, assuming a 0% probability of node deletion. If there is > 0% chance, then we will add a node for each edge deleted and create two edges to replace the disconnected node.

We implemented the planted clique by constructing an Erdos Renyi graph, selecting k random vertices, and connecting these k vertices. Thus, we construct a clique inside a random graph using SPTN.
Appendix – A – Table 7 shows the functions used in the SPTN function call to create planted cliques.

The algorithm for finding all cliques in a graph is NP-complete, fixed-parameter intractable, and is hard to approximate. Thus, we left our experiment by creating a planted clique and did not go in the direction of implementing or studying an algorithm to search for a planted clique. Our empirical finding that a planted clique can be constructed using SPTN is sufficient to claim that the model can analyze these types of graphs in whichever domain they are produced.

Planted Cliques are an essential piece to computer science theory as it is they can be used to find a maximum clique or all cliques in a graph. This can be used to find cliques in a social network and many other applications such as bioinformatics, modeling gene clustering, electrical engineering, design algorithms to compute Boolean functions efficiently, and in chemistry, describe and relate chemicals.

5.6 Signed Networks

Based on our previous research, we wanted to study and analyze temporal signed network generation. These networks are explored through two theories, balanced and status, which relate positive and negative edges and identifies human-related structures such as friends and enemies. Balance theory first emerged in the 1960s, by Heider, which states that any unsigned network can be broken down into subgraphs or triads which follow the guidelines in Appendix A - Table 8. The hypothesis was that relationships are formed to balance the network. Status theory on a signed directed network states that a positive edge from Node A to Node B means that Node B is to be of a higher status than Node A. This does not always agree with balance theory regarding edge prediction of a triad because balance theory was intended to be a model for undirected networks. Suppose there is a triad with a negative edge and two positive edges. Balance theory states that the triad would be underrepresented, but status theory states that the triad is probable.

Previous work had traditionally focused on small networks where social relationships were observed through direct interactions between individuals. However, they discovered in a temporal-directed social network that status theory effectively explains local patterns. It captures more substantial aspects of social behavior and linking mechanisms. These models provide insight into how fundamental theories and principles direct the formation of signed edges applied to social psychology, social computing applications, or other social networks. By creating a probabilistic synthesis of a signed network, we can create "dummy models" of social interactions at any scale or mimic any real networks such as Wikipedia, Reddit, or Facebook and analyze it as it evolves over time.

In our model, we wanted to analyze a temporal synthesis of a signed network using balance theory. While we were able to create a probabilistic signed network, we were unable to develop the algorithm that produces a balanced graph synthetically. However, we do have an algorithm designed. Firstly, the features needed to implement this are a signed graph with an initial configuration of three nodes with all positive, negative, or no edges, a probability function for linking nodes based upon mutual neighbors, and edge growth containment (an upper bound on the number of edges in the network). By having an initial configuration, we can easily create a beneficial distribution to mimic other models or start with a simple triad. A probability function is required to mimic the probability of a signed network based on balance theory - two nodes with multiple positive or negative mutual neighbors are likely to form a positive edge, two disagreeing edges with mutual neighbors are more likely to form a negative edge, and the probability of a new edge forming on a node is dependent on the number of mutual neighbors. And we need edge growth containment for ease of analysis; otherwise, it would grow rapidly and would be useless in terms of visual analysis. Appendix A – Table 9 shows the functional parameters used in the SPTN function call to produce a signed network. Appendix B – Figure 6 shows the SPTN generated signed network evolution snapshots.

Then for each pair of nodes, we identify the mutual neighbors, find their number, and their existing edges. If no edge exists between them, we create a new edge-based on the number of mutual neighbors and their signs as per Heider's balance theory. Another model considered was from a reference paper which stated - there exists a strongly balanced graph with v vertices and e edges iff $1 <= v-1 <= e <= ({}^{V}C_2)$

Our model is highly flexible and can take this inequality as a parameter for edge creation and deletion. If this inequality is true for any time instance t, then we would be able to use this condition to identify the signed network is in a balanced state.

6. Conclusion

- We had started with a hypothesis that SPTN can reproduce any network. In this paper, we scratched the surface by proposing the robust model and testing the hypothesis by replicating existing graph generator models. We were successful in simulating the same.
- We also had an intuitive conceptual justification for our hypothesis that any graph can be generated by controlling the nodes and edges. In this paper, we played around with different mechanisms to control the nodes and the edges, leading to various temporal network evolutions.
- This paper also briefs about the existing synthetic models. We successfully reproduced the temporal forms of the graphs generated by popular synthetic models such as Erdos Renyi, Barabasi, and R-MAT. We were successful in mimicking them and empirically showcase the results.
- Few other standard graph types such as planted cliques and signed networks were also shown to be reproducible. For some, we were only able to do a theoretical analysis up to an extent to feel confident about SPTN's ability to reproduce them.
- We also demonstrated the generation of temporal networks that are undirected and directed, and weighted.
- The hyperparameter dependency on multiple graph indicators was also demonstrated by feeding in complex functions as hyperparameters.
- The meta-map generated by the SPTN function call for temporal network

simulation is helpful for static and temporal analysis. This was demonstrated for the temporal network generated for comparing the Erdos Renyi graph.

7. Future Scope

The main purpose of having SPTN in place was to reproduce real-world networks synthetically, preserving the prominent characteristics. Mimicking existing models though it required a bit of creativity, was doable so that these models already had a procedure that could be incorporated. In return, since we simulated these models, we claim that we can simulate the real-world networks simulated by these models. But our target audience also has most of the random real-world network which do not fit in legacy graph generator models. As we say, we need to be creative in feeding in the functional inputs or hyperparameters for reproducing these networks. So, the question boils down to finding these five functions by observing the patterns of the temporal evolution of these graphs. We need a methodology to highlight the indicators on which these five functions depend. We foresee the most challenging part of applying SPTN to real-world networks. This is one central area where we would focus in the future.

- We envision coming up with a Machine Learning model that would succor in finding the relational dependencies of indicators over the input functions. Having this hand in hand with the already spawned SPTN model will be an exciting task.
- Another improvement we wish to do is reduce the number of parameters in the functional call. No doubt the current function call is complicated, but it has the complexity it deserves and can be improved upon in the coming future.
- The efficiency of the underlying algorithm can be improved multifold. Having the model tested on many more existing models will help us structure the function call better in a robust and scalable way.
- The geographical representative model BRITE was not explored in this paper.
- We need to find limitations of SPTN to fix the scope of the project as sometimes too much flexibility is also a curse.
- Use SPTN to replicate a small-world temporal simulation of a real-world network. This is our aim for which the model needs to be more robust, and we must have ideas to fit any real-world network in the five functional hyperparameters of SPTN.

The SPTN conceptual model seems to be a robust one and can incorporate many more sets of graphs. This may serve as a robust tool to conduct temporal analysis on scaled-down representations of real-world networks.

8. References

1. R-MAT model
2. Barabasi-Albert Model
3. Temporal Analysis of the Wikigraphs
4. Time-Varying Graphs and Social Network Analysis: Temporal Indicators and Metrics
5. Sampling of Large Graphs
6. The evolution of structural balance in time-varying signed networks
7. Signed Networks in Social Media
8. Strongly balanced and Random graphs

Appendix A – Tables

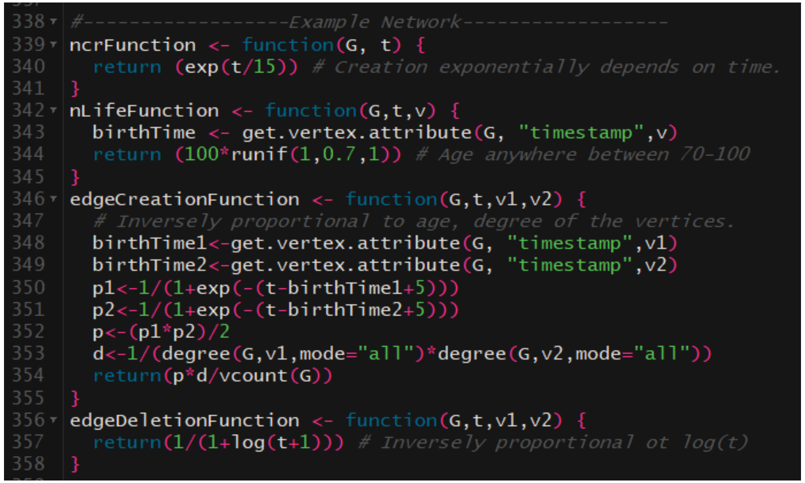
Table 1 – Function Call

Description - This is the function call to trigger a temporal network.

Usage - temporalNetwork(G, ncr(), nlf(), ecf(), edf(), ewf(), meta(…))

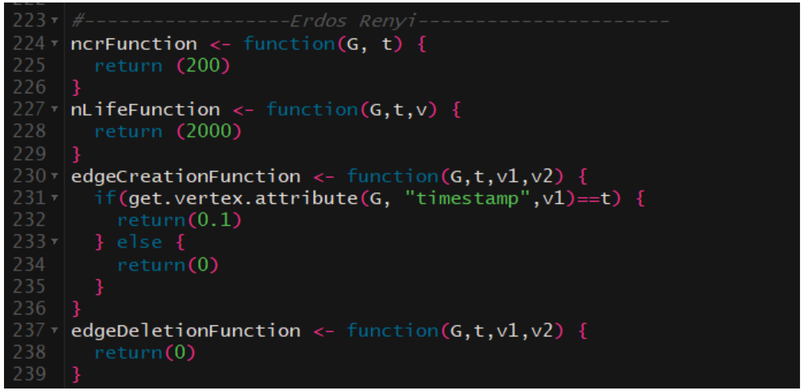
Argument	Explanation
G	Accepts initial state of the Graph object.
Ncr(G, t)	Node creation function. Takes Graph object at previous time instance and time as arguments. Returns – Integer/Graph object.
Nlf(G, v, t)	Node life function. Takes Graph object at previous time instance, vertex, and time as arguments. Returns – Integer/Graph object.
Ecf(G, v1, v2, t)	Edge creation function. Returns Probability. Takes Graph object at previous time instance, two vertices, and time as arguments. Returns - Double/Graph object.
Edf(G, v1, v2, t)	Edge deletion function. Returns Probability. Takes Graph object at previous time instance, two vertices, and time as arguments. Returns - Double/Graph object.
Ewf(G, v1, v2, t)	Edge weight function. Takes Graph object at previous time instance, two vertices, and time as arguments. Returns Double between 0-10/Graph object.
Meta-Data	
T_start	Start time epoch. Positive Integer
T_end	End time epoch. Positive Integer > t_start
Unit_time	Time step size. Positive Integer
selfloops	Boolean. If self-loops are allowed or not.
fixedEdgeSize	Default = -1. Passed when we must create fixed number of edges at every time instance. E.g., Barabasi Network. Accepts a positive integer.

Table 2 – Example Network Function Call



```
#------------------Example Network------------------
ncrFunction <- function(G, t) {
  return (exp(t/15)) # Creation exponentially depends on time.
}
nLifeFunction <- function(G,t,v) {
  birthTime <- get.vertex.attribute(G, "timestamp",v)
  return (100*runif(1,0.7,1)) # Age anywhere between 70-100
}
edgeCreationFunction <- function(G,t,v1,v2) {
  # Inversely proportional to age, degree of the vertices.
  birthTime1<-get.vertex.attribute(G, "timestamp",v1)
  birthTime2<-get.vertex.attribute(G, "timestamp",v2)
  p1<-1/(1+exp(-(t-birthTime1+5)))
  p2<-1/(1+exp(-(t-birthTime2+5)))
  p<-(p1*p2)/2
  d<-1/(degree(G,v1,mode="all")*degree(G,v2,mode="all"))
  return(p*d/vcount(G))
}
edgeDeletionFunction <- function(G,t,v1,v2) {
  return(1/(1+log(t+1))) # Inversely proportional ot log(t)
}
```

Table 3 – Erdos Renyi Function Call



```
#-----------------Erdos Renyi---------------------
ncrFunction <- function(G, t) {
  return (200)
}
nLifeFunction <- function(G,t,v) {
  return (2000)
}
edgeCreationFunction <- function(G,t,v1,v2) {
  if(get.vertex.attribute(G, "timestamp",v1)==t) {
    return(0.1)
  } else {
    return(0)
  }
}
edgeDeletionFunction <- function(G,t,v1,v2) {
  return(0)
}
```

Table 4 – Erdos Renyi Static Analysis

Indicator	**T=200**	**T=400**	**T=600**	**T=800**	**T=1000**	**Erdos Reyni**
Nodes	200	400	600	800	**1000**	**1000**
Edges	3721	11652	23367	39334	**59119**	**50089**
Average Degree	37.21	58.26	77.89	98.335	**118.238**	**100.178**
Maximum Degree	52	81	108	130	**151**	**136**
Components	1	1	1	1	**1**	**1**
Average Path Length	1.81391	1.85418	1.86999	1.87693	**1.88164**	**1.89977**
Diameter	3	3	3	3	**3**	**3**
ALCC	0.18765,	0.15074,	0.13294,	0.12499,	**0.12007,**	**0.11732**
GCC	0.18675	0.15045	0.13283	0.12493	**0.12003**	**0.11632**

Table 5 – Barabasi Albert Function Call

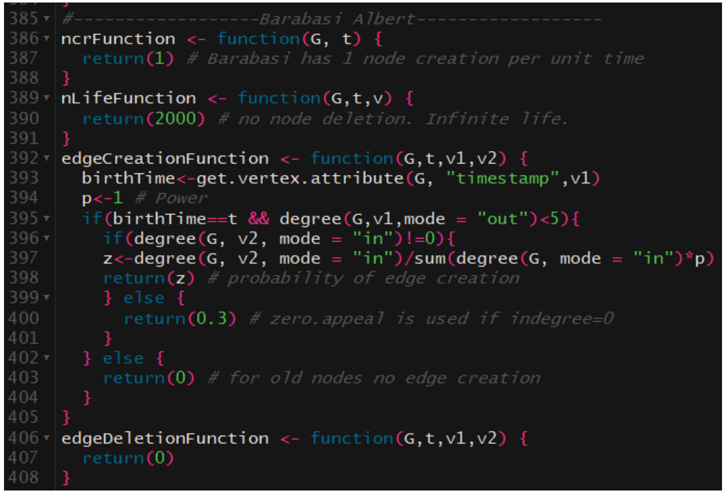
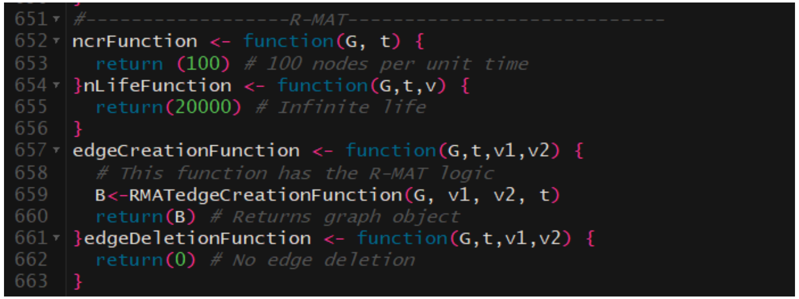
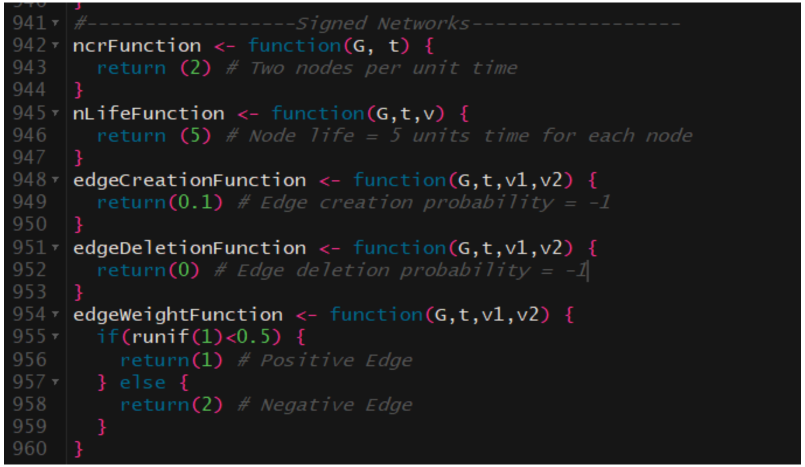

```
#-----------------Barabasi Albert-----------------
ncrFunction <- function(G, t) {
  return(1) # Barabasi has 1 node creation per unit time
}
nLifeFunction <- function(G,t,v) {
  return(2000) # no node deletion. Infinite life.
}
edgeCreationFunction <- function(G,t,v1,v2) {
  birthTime<-get.vertex.attribute(G, "timestamp",v1)
  p<-1 # Power
  if(birthTime==t && degree(G,v1,mode = "out")<5){
    if(degree(G, v2, mode = "in")!=0){
    z<-degree(G, v2, mode = "in")/sum(degree(G, mode = "in")*p)
    return(z) # probability of edge creation
    } else {
      return(0.3) # zero.appeal is used if indegree=0
    }
  } else {
    return(0) # for old nodes no edge creation
  }
}
edgeDeletionFunction <- function(G,t,v1,v2) {
  return(0)
}
```

Table 6 – R-MAT Function Call



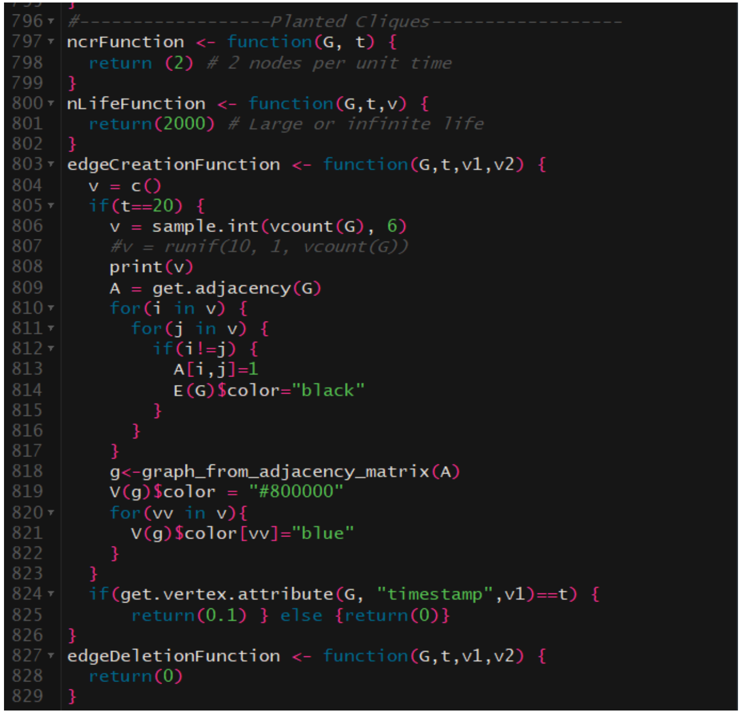
```
#-----------------R-MAT----------------------------
ncrFunction <- function(G, t) {
  return (100) # 100 nodes per unit time
}nLifeFunction <- function(G,t,v) {
  return(20000) # Infinite life
}
edgeCreationFunction <- function(G,t,v1,v2) {
  # This function has the R-MAT logic
  B<-RMATedgeCreationFunction(G, v1, v2, t)
  return(B) # Returns graph object
}edgeDeletionFunction <- function(G,t,v1,v2) {
  return(0) # No edge deletion
}
```

Table 8 – Signed Network function call



```
#-----------------Signed Networks-----------------
ncrFunction <- function(G, t) {
  return (2) # Two nodes per unit time
}
nLifeFunction <- function(G,t,v) {
  return (5) # Node life = 5 units time for each node
}
edgeCreationFunction <- function(G,t,v1,v2) {
  return(0.1) # Edge creation probability = -1
}
edgeDeletionFunction <- function(G,t,v1,v2) {
  return(0) # Edge deletion probability = -1
}
edgeWeightFunction <- function(G,t,v1,v2) {
  if(runif(1)<0.5) {
    return(1) # Positive Edge
  } else {
    return(2) # Negative Edge
  }
}
```

Table 7 – Planted Cliques



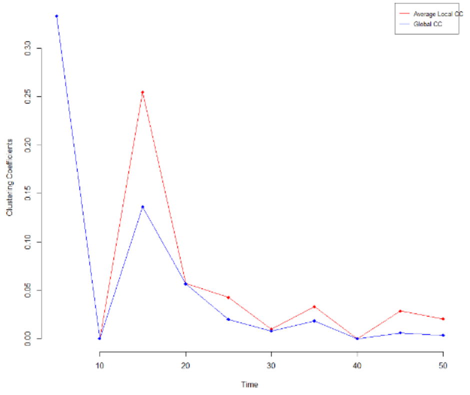
```
#-----------------Planted Cliques-----------------
ncrFunction <- function(G, t) {
  return (2) # 2 nodes per unit time
}
nLifeFunction <- function(G,t,v) {
  return(2000) # Large or infinite life
}
edgeCreationFunction <- function(G,t,v1,v2) {
  v = c()
  if(t==20) {
    v = sample.int(vcount(G), 6)
    #v = runif(10, 1, vcount(G))
    print(v)
    A = get.adjacency(G)
    for(i in v) {
      for(j in v) {
        if(i!=j) {
          A[i,j]=1
          E(G)$color="black"
        }
      }
    }
    g<-graph_from_adjacency_matrix(A)
    V(g)$color = "#800000"
    for(vv in v){
      V(g)$color[vv]="blue"
    }
  }
  if(get.vertex.attribute(G, "timestamp",v1)==t) {
    return(0.1) } else {return(0)}
}
edgeDeletionFunction <- function(G,t,v1,v2) {
  return(0)
}
```

Table 9 – Balanced Network

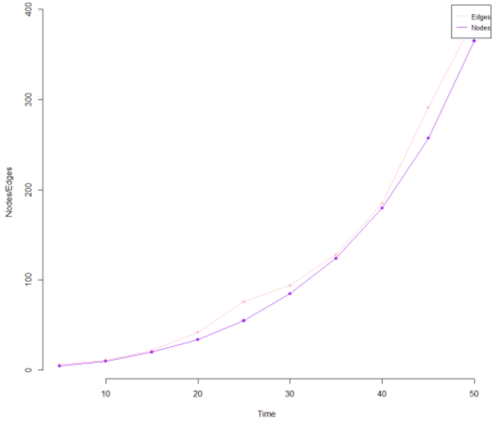
Balanced	WeaklyBalanced	Edge Sign		
Balanced	Balanced	Positive	Positive	Positive
Unbalanced	Unbalanced	Positive	Positive	Negative
Balanced	Balanced	Positive	Negative	Negative
Unbalanced	Balanced	Negative	Negative	Negative

Appendix B – Figures

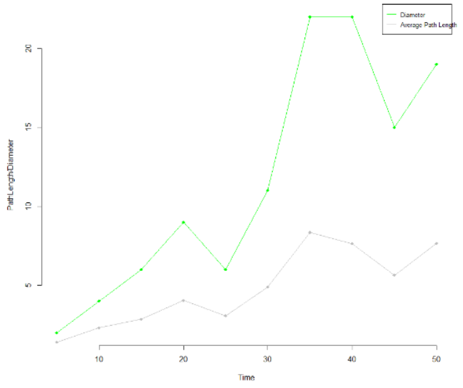
Figure 1 – Temporal Analysis of Example Network



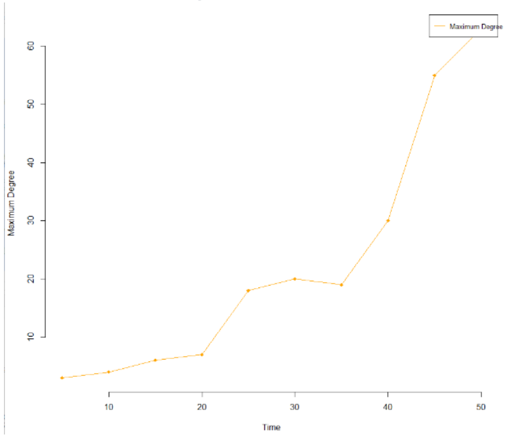
a) Local and Global clustering Coefficients



c) Nodes and Edges

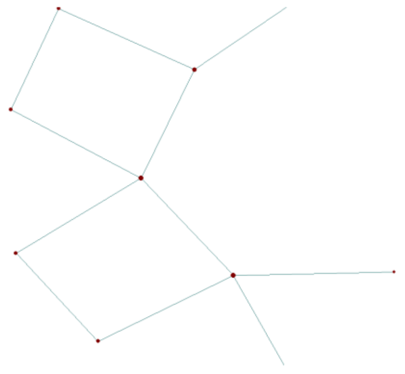


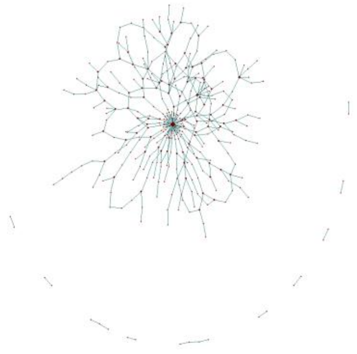
b) Path Length and Diameter



d) Maximum degree

Figure 2 – Evolution of the Example Network


Snapshot at T=10


Snapshot at T=50

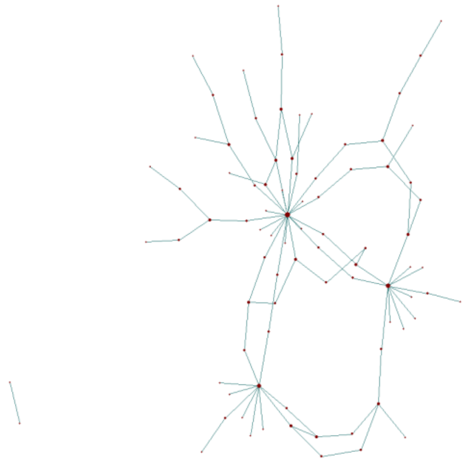
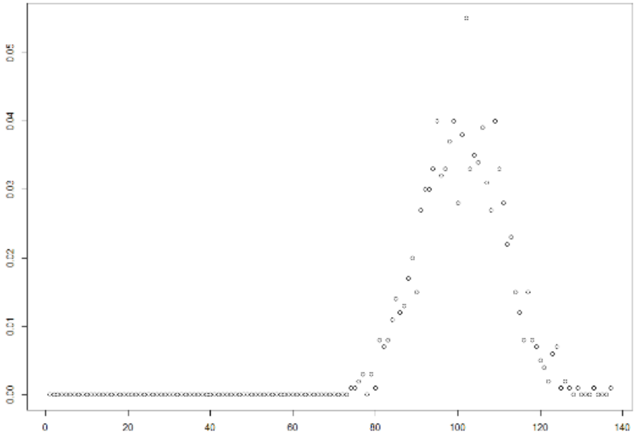
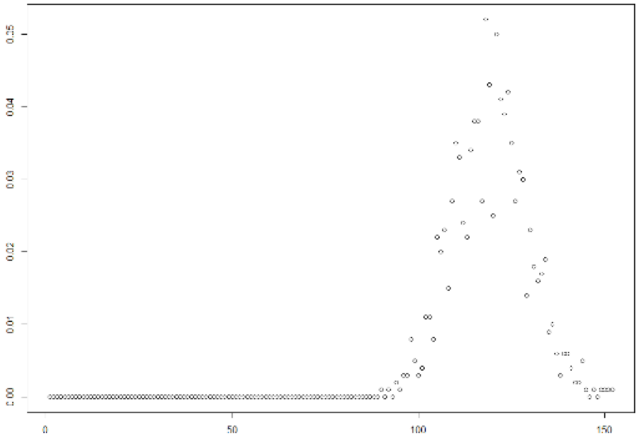

Snapshot at T=30

Figure 3 – Erdos Renyi Degree Distribution

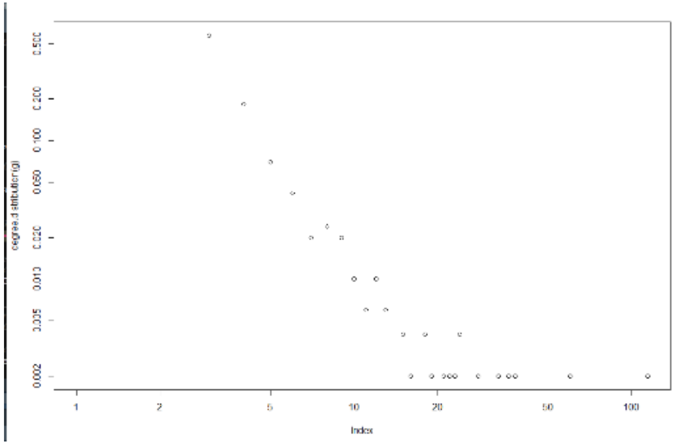


SPTN generated Erdos Reyni Degree distribution - n=1000 & p=0.1

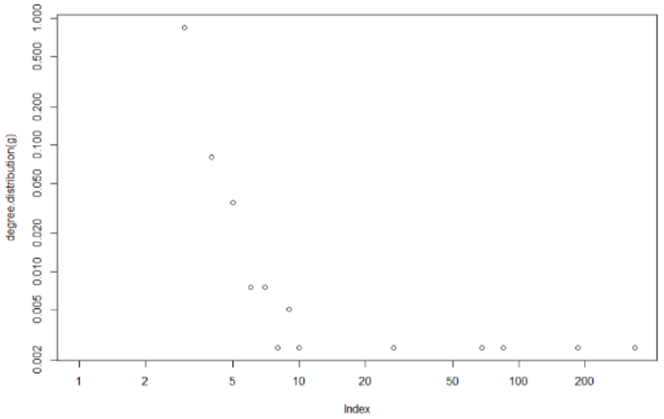


Erdos Renyi Game Degree Distribution - n=1000 & p=0.1

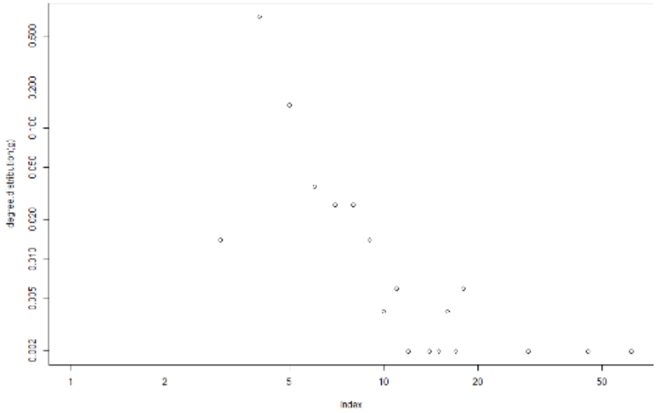
Figure 4 – Barabasi Degree distributions



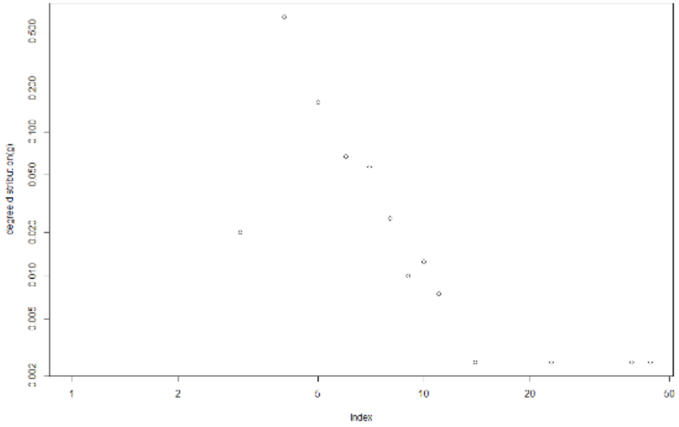
a) Barabasi Game - p=1, n=500, m=2, z=1



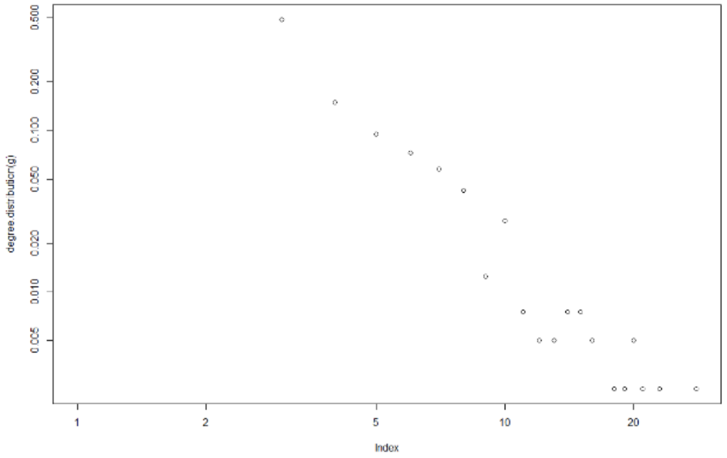
c) Barabasi Game - p=1.5, n=500, m=2, z=1



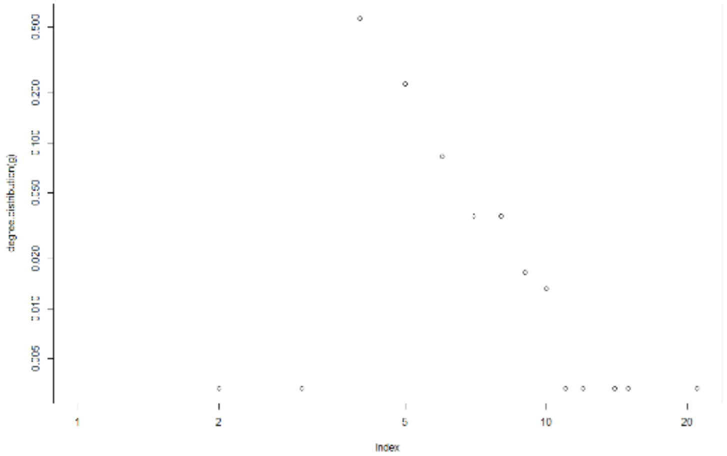
b) SPTN generated Barabasi - p=1, n=500, m=2, z=1



d) SPTN generated Barabasi - p=1.5, n=500, m=2

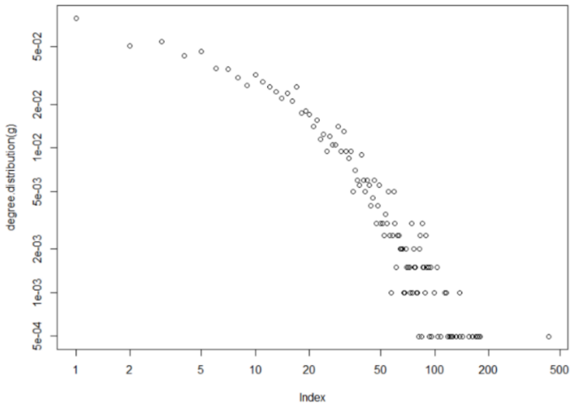


e) Barabasi Game - n=500, p=0.5, m=2, z=1

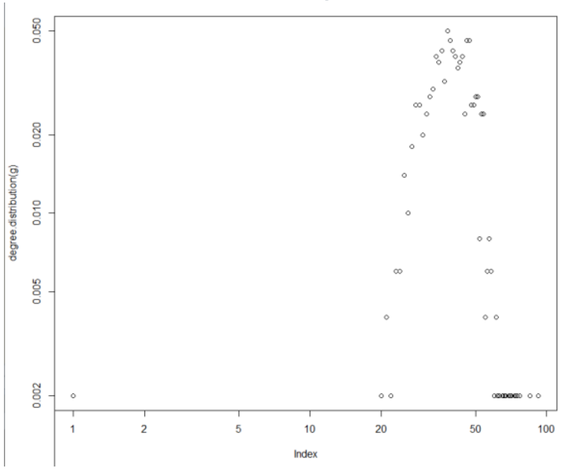


f) SPTN generated Barabasi - n=500, p=0.5, m=2, z=1

Figure 5 – R-MAT Degree distribution

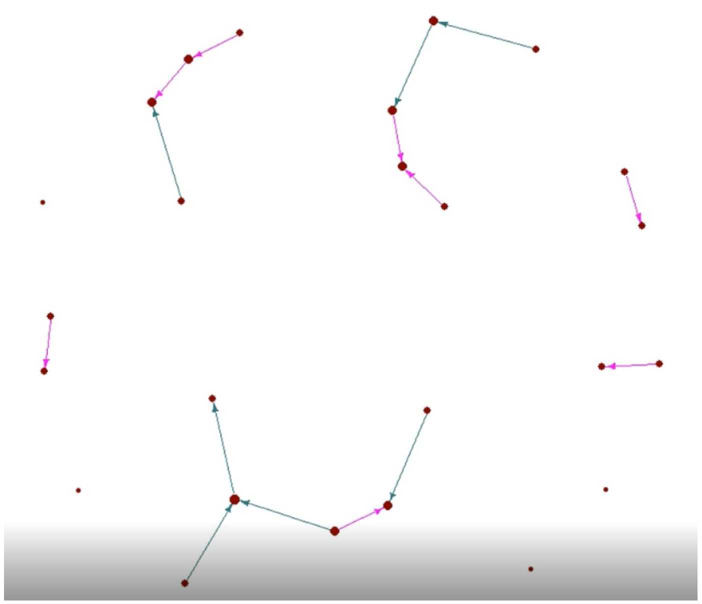


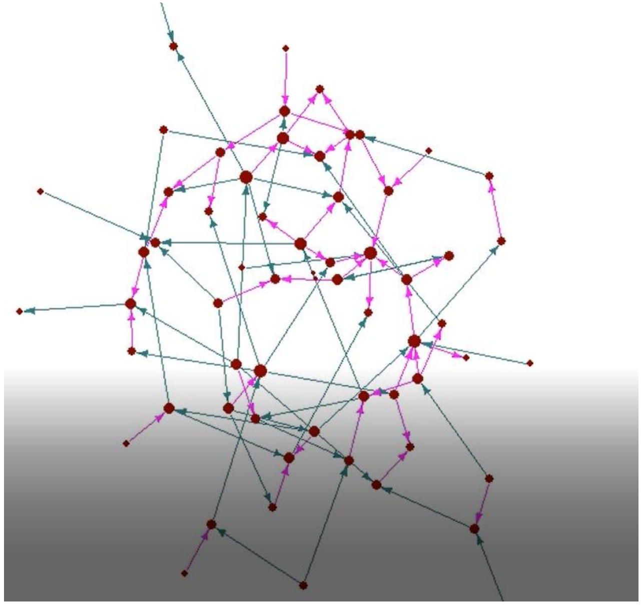
SPTN Generated R-MAT degree distribution n=1000, edges=10000, p= [0.4,0.2,0.2,0.2]



SPTN Generate R-MAT degree distribution for n=500, edges=10000, p= [0.25,0.25,0.25,0.25]

Figure 6 – Signed Network Evolution


SPTN generated Signed Network - T=10


SPTN generated Signed Network - T=30