Universiteit Leiden

Master Computer Science

Risk score propagation algorithms for domain-domain networks

Name:	Prashand Ramesar	
Student ID:	s1439642	
Date:	26/04/2019	
Specialisation:	Computer Science and Advanced Data Analytics	
1st supervisor:	dr. Frank Takes	
2nd supervisor:	dr. Daniël Worm	[TNO]
Other supervisor:	ir. Alex Sangers	[TNO]
Other supervisor:	ir. Harm Schotanus	[TNO]

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Organizational computer networks are at risk of being compromised by making connections to malicious domains, which can lead to sensitive information being stolen by cyber criminals. It can therefore be helpful to have an estimation of the risk level of domains, expressed as a risk score, before connections are made. To this end, we explore network science methods to construct domain-domain networks from passive DNS data. We propose three score computation algorithms that can calculate risk scores for domains. Two of our proposed algorithms (LISC and NORMLISC) are capable of modelling complex interactions between nodes in the network. However, these algorithms have the unwelcome properties that lower risk scores can be computed for a node than for its neighbours which have higher scores in the initial state (i.e., 'swapping') and that the position of the node in the network is not taken into account for score computation. In contrast, our third algorithm (MAXDIF) has fewer free parameters, takes the position of nodes in the network into account and does not suffer from 'swapping', but gives no freedom in modelling complex interactions between nodes. All in all, computing risk scores is now possible using our proposed algorithms.

Acknowledgements

First and foremost I would like to thank Frank for giving me the green light to start my graduation internship at TNO and being a very involved supervisor throughout the entire project. Despite having a busy schedule, Frank made sure to take the time to meet in person frequently, providing me with great advice and feedback. When meeting in person was not possible, Frank was always available via mail to help me out with any questions I had.

Of course, many thanks go out to Daniël, Alex and Harm for guiding me throughout the project as well. Alex was a very great weekly supervisor in all dimensions. Not only did Alex provide guidance, he was also an amazingly pragmatic listener who managed to take everyone's interests at heart and succeeded in finding a good consensus between everybody's expectations regarding this project. Besides this, Alex also made sure that I got a proper introduction to TNO and helped me with any other practicalities. Harm was also involved weekly since the beginning of my project, providing me with much needed cyber security insights and facilitating access to the TNO malware data so that I could use it for my research. Harm even took the time to revise parts of my thesis regarding cyber security and DNS despite being on leave. Daniël has also been involved since the first stages of my project and promised to pick up the roles as weekly TNO supervisor and as second reader for the last two months of my project, which he did splendidly. I cannot be thankful enough for all the time and energy that has been spent by everyone into ensuring a successful completion of the project.

I would also like to express my gratitude to TNO department Cyber Security & Robustness as a whole for making this very interesting project more fun. I can honestly say that my time at TNO was very pleasant and that I enjoyed coming to the office to work on the project. I have had very pleasant and insightful chats with people from CSR, without whom my project would have certainly been a little less fun.

Lastly, I would like to thank Mnemonic for providing me with a prepared passive DNS data set that I was allowed to use throughout my entire project.

Thank you all,
Prashand

Contents

1 Introduction

Organizations such as financial institutions, insurance companies and governments guard sensitive information in the interest of their stakeholders. Computers within IT networks of such organizations make connections to domains outside of their network, of which some are potentially malicious in the sense that they host malware or any other type of damaging content. An example of how infection can happen is by *domain hijacking*, which entails an unauthorized takeover of a domain name, which can then lead to impersonation and phishing practices [1]. If this type of damaging content infects one or more computers in the organizational network, we speak of a compromised system. Having compromised systems may lead to serious consequences such as significant financial damage [2] or theft of private data [3]. The stolen data can be used for malicious ends and cause unwelcome impact on the organization and its stakeholders.

In malicious domain detection, the main challenge is to identify domains that host malicious content, given a data set of domain data. Often, this is attempted through analyzing Domain Name System (DNS) data [4]. This is usually conducted by labeling domains as malicious or benign and designing a system that aims to correctly predict maliciousness of unlabeled domains. Various methods, ranging from machine learning to expert knowledge-based methods, have been applied to compute domain labels or scores.

The approach in this thesis is similar to what is usual in malicious domain detection, however the focus here lies on computing 'risk scores' of domains, given dynamically evolving networks constructed from DNS data, where each domain is assigned an initial score. A risk score provides an indication of the perceived risk level of making connections to an external domain. The idea here is that 'topologically close' domains (i.e., domains that are relatively close to each other) within such networks are likely to exhibit similar behaviour. As such, they are deemed similarly risky to visit, e.g., due to hacked web servers

with more domains hosted on one device. Should a domain be close to one or more risky domains in the network, then it is more likely that the domain in question is suspicious given its ties to the other suspected nodes. Ties between domains evolve constantly, which in turn impacts the risk scores. An example of a change is that a previously non-existent relationship between two domains is suddenly relevant. This means that these domains should now be linked to each other in the constructed network and that their risk scores influence each other. Such changes can suddenly make a domain safe or dangerous to make a connection to. As such, the risk scores of these domains have to be updated in real time, which is a dynamic process. The focus therefore does not lie on computing initial scores, but instead on the computation of risk scores given initially computed risk scores using defined ties.

The approach to compute risk scores will be taken from a network science perspective. Network science is an interdisciplinary scientific field that is mainly concerned with extracting knowledge from networks. Examples of such networks are found in society (e.g., social networks), nature (e.g., cellular networks) and communication systems (e.g., telecommunication networks) [5]. These networks can be created by first modelling all actors as nodes and drawing links between actors if relevant interaction (e.g., two people are friends with each other on a social network or two computers are sending messages to each other) is occurring. For computing risk scores, it may be fruitful to explore similar network science models such as opinion formation models, the PageRank algorithm and 'Guilt by association' models. These models namely all have the property that they compute new scores for entities by taking relevant ties between them into account.

This research project aims to work toward an approach that can take (enriched) DNS data and risk scores from external sources as input in order to build a 'domain-domain network' (where nodes are domains and links denote a type of relationship between the domains), which can subsequently be used to compute and update risk scores for domains with the help of network science algorithms. Computation should be possible on a global and a local scale. This means that it should be possible to compute all scores at once, but also on a local scale for one domain and its neighbours in the event of a newly detected threat from external sources. Since the World Wide Web is estimated to scale to over a billion domains, the approach also needs to be scalable in terms of time and space complexity [6]. The main research question in this thesis therefore is:

How can network science methods be used to efficiently compute risk scores of domains in dynamically evolving domain-domain networks?

This thesis is structured as follows. Chapter 2 introduces the required preliminary knowledge. Chapter 3 presents an overview of published work regarding malicious domain detection and network score computation models that are related to the work presented in this thesis. Chapter 4 describes the data that is used to answer the research question. Chapter 5 presents the approach taken and delves into the full construction of the network that will be used for the experiments, along with the motivation behind choices that are made during

its construction. Furthermore, the requirements for the risk computation algorithm, as well as the designed algorithms themselves, are presented. Chapter 6 presents the experimental setup that will be followed to answer the research question, the experimental results and lastly a discussion of these results. The thesis concludes with Chapter 7, which provides the conclusions drawn from the research, research limitations and future work.

2 Preliminaries

This chapter introduces preliminary concepts from both *network science* (Section 2.1) and the *Domain Name System* (Section 2.2). These preliminaries present the required knowledge to understand the contents of this thesis.

2.1 Network science

A network consisting of nodes and links can be formally defined as $G = (V, E)$, where V denotes the set of nodes and E denotes the set of links. The total number of nodes $|V|$ and links $|E|$ in the network will be represented as n and m, respectively. A network can be *directed* or *undirected*, referring to whether or not the links have a defined direction. The *degree* of a node v (notation: $deg(v)$) denotes the number of neighbour nodes for a node $v \in V$. The *indegree* and *outdegree* of a node v denote the number of incoming and outgoing links for v respectively. The neighbourhood $N(v)$ denotes the set of neighbours of v. A path is a sequence of links between two nodes, whereby the length of the path is the number of links in this sequence. The distance $d(v, w)$ denotes the shortest path length between node pairs $v, w \in V$. The average distance $\bar{d}$ between all node pairs $v, w \in V$ can be calculated as follows:

$$\bar{d} = \frac{1}{n(n-1)} \sum_{v,w \in V} d(v, w) \tag{2.1}$$

The distance distribution for G denotes the total number of nodes that all nodes in V can reach at distance $1, 2, 3, .., max(d(v, w))$. The *average* distance for G denotes the number of nodes a node in V can reach on average at each distance. See Figure 2.1 for an example of an undirected network and the use of the formal notation given above.

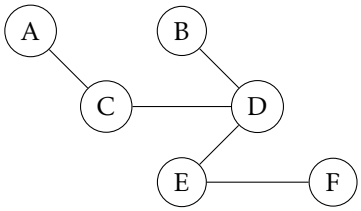

Figure 2.1: An example of an undirected network $G = (V, E)$ with the following properties: $V = \{A, B, C, D, E, F\}$, $E = \{(A, C), (B, D), (C, D), (D, E), (E, F)\}$, $n = |V| = 6$, $m = |E| = 5$, $deg(D) = 3$, $d(A, F) = 4$, $\bar{d}(G) = 2.2$.

In this thesis two types of networks are used, namely *one-mode* networks and *two-mode* networks. Two-mode networks are also known as bipartite networks. This means that the network consists of two different sets of nodes where the links exist only between nodes from two different sets, but not between nodes from the same set. In contrast, one-mode networks, such as the one in Figure 2.1, only consist of one type of nodes. The notation by Latapy et. al [7] will be adopted for these two-mode networks. A two-mode network can be formalized as $G = (\top, \bot, E)$, where $\top$ is the 'top' set of nodes, $\bot$ is the 'bottom' set of nodes and $E \subseteq \top \times \bot$ is the set of links in the network. See Figure 2.2 for an example.

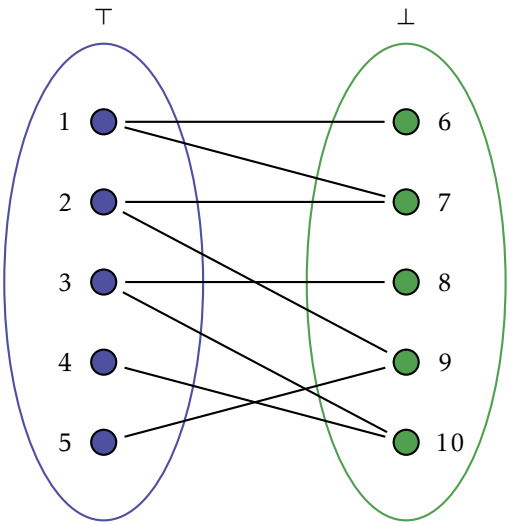

Figure 2.2: An example of an undirected two-mode network $G = (\top, \bot, E)$, with the properties: $n = 10$, $m = 8$, $deg(1) = 2$, $d(1, 5) = 4$, $\top = \{1, 2, 3, 4, 5\}$, $\bot = \{6, 7, 8, 9, 10\}$, $E = \{(1, 6), (1, 7), (2, 7), (2, 9), (3, 8), (3, 10), (4, 10), (5, 9)\}$.

Latapy et al. also define projection methods, with which two-mode networks can be projected to one-mode networks. A $\top$-projection keeps all nodes in $\top$ and places links between nodes if they are linked to the same node in $\bot$ in the two-mode network. For the $\bot$-projection, the rules are analogously applied. In this thesis, we assume that these projected networks are unweighted. Un-

weighted networks carry no additional information as weights on the links. An example of a weighted network could be a road map network, that shows the distance between destinations on the links.

In this thesis, certain network science algorithms will be used as inspiration for the models that will be proposed later. Some of these algorithms, such as PageRank, compute scores iteratively and can be run using Markov chains [8]. It is helpful to know what Markov chains are in order to understand how these algorithms and the proposed models work. Markov chains require a vector $\vec{P}^t$ of size n, where the elements $\vec{P}^t_1$, $\vec{P}^t_2$,..,$\vec{P}^t_n$ are the scores, representing the likelihood of the element being in a certain state, for nodes $1, 2, .., n$ at iteration t. The other requirement is an $n \times n$ transition matrix T, where $T_{v,w}$, with $v = 1, 2, .., n$ and $w = 1, 2, .., n$, is the 'influence' which node v exercises on w. Scores can then be updated iteration-wise using Formula 2.2. This formula is an example of a solution method called the power method.

$$\vec{P}^t = \vec{P}^{(t-1)}T \tag{2.2}$$

2.2 Domain Name System

The Domain Name System (DNS) is a system which enables users to surf the Internet by resolving *domain names* (e.g., *www.google.com*) to the correct *IP addresses* (e.g., 172.217.19.206). The DNS is structured in hierarchical levels. At the top, there is the root level (operated by *root name servers)*, which DNS uses to direct requests to the right top-level-domain (TLD) server. TLDs are the first distinguishable level of domain names which occur at the end of the domain name (e.g., .com, .edu and .org). The next levels after the TLD are, in order, the second level domain (SLD), which is a subdomain of TLD, and other lower level domains that act as subdomains to the domain that precedes them [9]. See Figure 2.3 for an illustration of the domain name hierarchy.

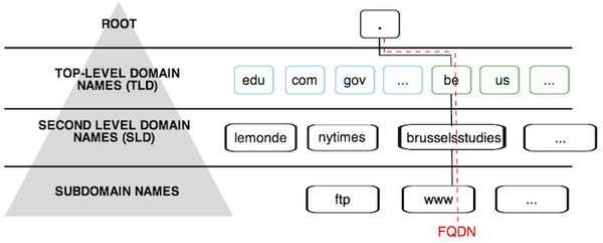

Figure 2.3: An overview of the leveled DNS hierarchy. Illustration from [10].

IP addresses refer to the specific network addresses of the hosts that store the domains. Once the DNS has found the correct IP address for the queried domain, the IP address is sent to the user whose machine can subsequently use it to make a connection to the right server. IP addresses exist in two protocols, namely IPv4 (32-bit long address) and IPv6 (128-bit long address).

Requests to a DNS server are sent as a *query* (e.g., a domain name) and responses are given by DNS as an *answer* (e.g., an IP address). There are different kinds of DNS resource records (which map IP addresses to domains or domains to domains) [11], of which four are relevant :

1. `A`-requests: The DNS is queried for a domain and returns one or multiple IPv4 address.
2. `AAAA`-requests: The DNS is queried for a domain and returns one or multiple IPv6 address.
3. `PTR`-requests: The DNS is queried for an IP-address and returns a domain name.
4. `CNAME`-requests: The DNS is queried for a domain name with an alias and returns a canonical name.

These four types of resource records are relevant in Chapters 4 and 5, because they will play a role in defining the actors (i.e., the domains) and the relationships (i.e., the links) for the network that will be constructed from the DNS data.

3 Related work

For score computation we apply knowledge and methods from malicious domain detection and network science. For malicious domain detection, many different kinds of approaches are used. In a survey on malicious domain detection, Zhauniarovich et al. categorize these steps as *DNS data collection*, *data enrichment*, *feature extraction*, *algorithm design* and *validation of results* [4]. The aim is to roughly follow this framework in order to approach the research question formulated in Chapter 1. For the algorithm design step, network science algorithms are evaluated based on their properties to determine the requirements to successfully calculate and update scores within a network. This chapter will first introduce the step-by-step malicious domain detection framework as defined in [4], as well as relevant publications within this field in Section 3.1. Next, Section 3.2 will provide an overview of relevant network science methods. Lastly, the main contributions that extend the related state-of-the-art publications are discussed in Section 3.3

3.1 Malicious domain detection

When malicious domain detection models are built, usually DNS data sets are used, which can be collected passively or actively. *Passive DNS data* records are collected by simply listening to DNS requests and storing a summary, e.g., as used by Choi et al. [12]. *Active DNS data* consists of actively querying domain names at servers and recording the records [13].

The data can then be enriched with different data sources to detect malicious domains, such as the geo-location [14], the ASN [15], registration records [16], resource records [17] and other types of network activity data [18].

Once the required data has been collected, Zhauniarovich et al. note that many

different types of features can be extracted [4]. These features can be:

1. *Internal*: features directly taken from DNS data, such as average TTL [19].
2. *Contextual*: features computed from DNS data and other data sources, such as the number of ASNs (Autonomous System Number) in the network [20].
3. *DNS data set dependent*: features that are highly dependent on the chosen data set, such as the IP addresses which are linked to a domain [21]. These change over time and depend on the time period in which the data are collected.
4. *DNS data set independent*: features that do not depend on the information collected in the DNS data set, such as the 2 or 3-gram distribution within a domain name [22].
5. *Mono domain*: features calculated for every single domain independently, such as all countries in which a specific domain is hosted [15].
6. *Multi domain*: features determined for a set of domains rather than for all of them separately, such as the number of shared ASNs [23].

Zhauniarovich et al. distinguish two methods to use these features for malicious domain detection, namely *knowledge based methods* and *machine learning based methods* [4]. Knowledge based methods are mostly derived by building systems or frameworks that look for certain patterns in the data. These patterns are based on observations made by individuals who possess domain knowledge. An example would be the work of Sato et al., where they count the co-occurrences of malicious and other domains in simultaneous queries based on the idea that a lot of malicious domains are queried together [24]. Machine learning based approaches, in turn, can be divided into *supervised learning*, *unsupervised learning* and *semi-supervised learning*. Supervised approaches require completely labeled training and testing data. This can be done with the help of blacklisted and whitelisted domains in order to train the machine learning model on the data set. For instance, Fukuda et al. make use of Classification and Regression Trees, Random Forests and Support Vector Machines in their approach [25]. With semi-supervised learning approaches, both known and unknown domains are used in the method. This can for instance be done using a graph-based approach, where unknown domains can be identified based on propagation [26] or clustering techniques [27]. In unsupervised learning approaches, the goal is to build a set of features and relationships between domains in order to separate benign domains from malicious domains without requiring any labeled data. Usually clustering approaches are taken [28]. In hybrid approaches such as Notos [22], a blend of different techniques and algorithms is used to detect malicious domains.

Validation is usually done using a *ground truth*, i.e., malicious and benign domains are determined using black and whitelists, respectively. Blacklists are lists that provide an overview of domains that are known to be malicious. Whitelists provide an overview of trusted domains. This allows one to label the domains accordingly and validate the results using metrics such as True/False

Positive Rates (e.g., as done in Haddadi et al. [29]) and True/False Negative Rates (e.g., in Chiba et al. [20]). Furthermore, other metrics such as Precision (e.g., proposed in Lee et al. [30]), Classification Accuracy (e.g., as seen in Hsu et al. [31]), F1-score (e.g., in Fukuda et al. [25]) and Area Under Curve (e.g., in Manadhata et al. [32]) can also be used to determine performance.

3.2 Network science methods

There are multiple ways to achieve computation of domain scores in many different settings. Here we discuss algorithms performing tasks similar to our task. Namely, we will discuss four related techniques derived from opinion formation (as explained in Subsection 3.2.1), link analysis (see Subsection 3.2.2) and Guilt by Association modelling (in Subsection 3.2.3). These techniques contain the main ingredients required for the design of the models for our task. Additionally, the first two models make use of Markov chains, as explained in Section 2.1.

3.2.1 Opinion formation models

Opinion formation is a widely researched area, starting with a leading theory by Katz and Lazarsfeld, who show that individuals' opinions are formed by peer influence from opinion leaders (i.e., individuals who are influential), and not just by the media [33]. This stimulated research for models that show how opinions form and converge within influence networks. Leskovec et al. discuss two kinds of 'diffusion models' that attempt to model individuals' product purchases based on the behaviour of their peers, namely *threshold models* and *cascade models* [34]. Threshold models are run on weighted networks where an individual will adopt an idea (e.g., the idea to buy a product) if the weighted average of all direct neighbours with this idea exceeds a predefined threshold. Cascade models are also run on weighted networks, but are probabilistic in nature where an individual will adopt an idea with a probability, dependent on whether a neighbour also adopts the idea.

Another form of opinion formation in networks are models that attempt to model *consensus*, i.e., if members of a network end up agreeing with each other. DeGroot introduced a model [35], where each individual v has an initial opinion F_v about an idea, where $0 \leq F_v \leq 1$. These opinions are stored in a vector $\vec{F}$, such that $\vec{F} = (F_1, F_2, .., F_n)$. There also exists a $n \times n$ transition matrix P with elements P_{vw} denoting individual v's influence on node w's opinion in the network, with $0 \leq P_{vw} \leq 1$. The opinion vector is updated iteratively using a Markov chain (see Formula 3.1):

$$\vec{F}^{(t)} = \vec{F}^{(t-1)}P \tag{3.1}$$

Here t denotes the time step for which $\vec{F}$ is calculated. This model has been studied extensively. For instance, Golub et al. [36] show that convergence in this model only happens when the most influential individual becomes less influential as the total number of individuals k increases in the network. In

another related work, Heidergott et al. argue that communications between individuals occur randomly and not constantly with all peers [37]. This means that opinions for each individual need to be updated stochastically by taking random draws from the set of peers instead of updating for all peers in each iteration. They also show that large societies do not converge when everybody has equal influence on each other.

3.2.2 The PageRank algorithm

The PageRank algorithm [38] belongs to the class of *link analysis* algorithms, which are mostly used for information retrieval purposes, such as the ranking of documents by relevance. Link analysis refers to the study of relationships between entities, such as documents or web pages, that refer to each other [39]. This ranking is achieved by linking documents in a network and calculating the 'authority' or 'presence' for each document in the network [40].

PageRank is a method to compute scores iteratively while taking the node's importance in the network into account. PageRank works based on the principle of a *random surfer* surfing the web, starting at a document. This surfer then chooses an outgoing link to another document with a given probability and keeps visiting documents iteratively. To prevent the surfer from being trapped at a document without outgoing links, the surfer can 'teleport' to a random document in the network with probability 1 if he is trapped, and with probability δ if otherwise. Visiting outgoing links hence occur with probability $1-\delta$. After many iterations, the 'importance' or 'PageRank' is reflected by how often the surfer has visited each node in the network. With the PageRank algorithm, being linked to more important nodes is beneficial for the individual's PageRank score, since more important nodes give the neighbours a higher probability of being visited by the random surfer. PageRank can be calculated using a Markov chain and starts out with a vector $\vec{X}^t$ of size n documents, where values $\vec{X}^t_1$, $\vec{X}^t_2$,..,$\vec{X}^t_n$ correspond to the respective PageRank value for each node $v \in V$. These values can be initially set to a value between 0 and 1, e.g., $\frac{1}{n}$, which is initialized equally for every node. $\vec{X}^t$ is iteratively multiplied with an $n \times n$ transition matrix P, which contains the probabilities P_{vw}, with $v = 1, 2, .., n$ and $w = 1, 2, .., n$ at which the surfer chooses to visit node w from node v. Often, the PageRank algorithm is implemented with a parameter ϵ, which acts as a stopping condition and indicates the threshold for the total amount of change in values between iterations. Once this threshold is no longer met, the algorithm terminates and returns the PageRank for each node in the network. All in all, this iteration process boils down to Formula 3.2 for a certain P:

$$\vec{X}^t = \vec{X}^{(t-1)}P \tag{3.2}$$

Here, P is built by determining all possible hyperlinks $h_1, h_2, .., h_h \in H$ that a surfer can click from a given web page v. Then, the probability of visiting a random web page in $N(v)$ from v is $(1-\delta) \cdot \frac{1}{|H|} + \delta \cdot \frac{1}{n}$ and the probability of visiting any web page through teleportation is $\delta \cdot \frac{1}{n}$. The vector $X^{(t-1)}$ is then iteratively multiplied with P until the total change in values is below ϵ.

3.2.3 Guilt by association modelling

In biological research, ideas about the unknown functions of a protein are often derived from connected proteins for which the functions are known [41]. This method is called Guilt by association (GBA). This can be done with protein-protein interaction (PPI) networks, where the set of nodes consists of proteins and the links exist between proteins that interact with each other.

Qian et al. take such a GBA approach in a PPI network [42]. The PPI network $G = (V, E, w)$, where w denotes the weights of an edge $e \in E$ which reflect the strength of association between the proteins. They also define two functions $Y : V \to \mathbb{R}_{\geq 0}$ and $F : V \to \mathbb{R}_{\geq 0}$, which are functions of prior and posterior evidence respectively. $Y(v)$ assigns a higher score to a node v if it is believed beforehand that the protein is linked to a disease and a lower score if it is not. $F(v)$ calculates the posterior score for each node after an iteration in which information from neighbours is propagated to neighbour nodes. $N(v)$ is defined as the set of direct neighbours and $\alpha \in (0, 1)$ is a parameter that reflects the importance of the information absorbed from the set of neighbours $N(v)$. Lastly, if the edge weights are defined as $w'_{v,u} = w_{v,u}/\sqrt{deg(v) \times deg(u)}$, then the posterior scores can be calculated using Formula 3.3:

$$F(v) = \alpha[\sum_{u \in N(v)} F(u)w'_{v,u}] + (1 - \alpha)Y(v) \tag{3.3}$$

3.3 Main contributions

We discussed several graph-based approaches, such as clustering of traffic data [27] and belief propagation techniques that are probability-based [26]. In contrast, we bring knowledge from malicious domain detection and network science together by first exploring methods to link domains to each other through established relationships using passive DNS data. After the resulting domain-domain network is constructed, we provide the domains with initial risk scores and design algorithms which are inspired by certain elements of the algorithms discussed in this section to efficiently compute their risk scores. After experimentation, an analysis of the benefits and drawbacks of each proposed model is given, followed by a discussion on the feasibility of each model in different use cases. As far as we know, such an approach has never been taken to determine the risk level of domains.

4 Data description

The data that will be used in this work comprises two data sets, namely a *passive DNS* data set supplied by Mnemonic [43] (discussed in Section 4.1) and a *malicious* data set supplied by TNO (see Section 4.2), which is a list of domain names that were malicious at a point in time, as derived from public blacklists. Lastly, Section 4.3 is devoted to the data quality of these data sets.

4.1 Passive DNS data set

The passive DNS data set contains 84,599,816 records, with data accumulated from 2012-05-20 until 2018-01-05. Each row is a unique combination of *query*, *RR Type* and *answer*. A sample overview of the records can be found in Table 4.1. Notable columns are:

(i) *Query*: the requested domain.

(ii) *Answer*: One or more answers in the form of an IP address, domain name or text depending on the RR Type.

(iii) *RR Type*: the type of record. `A`- and `AAAA`-records are made from a domain to an IP address. `PTR`-records are connections made in reverse, i.e., from an IP address to a domain. Lastly, `CNAME`-records are connection made directly between two domains (see Section 2.2).

(iv) *Answered*: the number of times a specific DNS request has been answered.

(v) *First seen*: the time stamp at which the relation was first seen.

(vi) *Last seen*: the time stamp at which the relation was last seen.

ID	Query	RR type	Class	Answer	TTL	Answered	First seen	Last seen
1292 3169	just-sh4ring. blogspot.com.br	CNAME	IN	blogspot.l. googleuser content.com	600	2	2017-12-25 15:32:14	2017-12-25 15:32:14
1292 3239	www. googleseo.eu	A	IN	188.114. 252.52	7200	1	2017-12-29 14:26:54	2017-12-29 14:26:54

Table 4.1: A sample overview of the passive DNS data supplied by Mnemonic.

4.2 Malicious data set

The malicious data set contains 19,035,818 records with data accumulated from 2016-11-17 until 2018-10-16. The data is collected from dozens of blacklist sources. The entries in this data set are from domains that also exist in the same time frame as in the Mnemonic data set. A sample overview of the data can be found in Table 4.2. Notable columns are:

(i) *Domain*: the domain that is flagged as malicious (e.g., as malware). It is not recommended to visit these domains!

(ii) *Source*: the source blacklist from where the data is retrieved.

(iii) *Reason*: the reason why the domain was detected as malicious. There are different kinds of reasons, namely:

- *DGA (Domain Generation Algorithm)*: DGA domains are domains that are created in large numbers and can act as communication points that send and receive updates for malicious ends, such as creating botnets or more DGA domains.
- *CNC (Command and Control)*: CNC domains can actively send commands to compromised systems, for instance to steal information.
- *Malware*: Malware is hosted on the domain, which can be used to compromise as many systems as possible. Once compromised, the malware can cause damage, for instance by stealing information.
- *Phishing*: Phishing domains prompt the visitor to enter sensitive private information in an attempt to steal it.
- *Fraud*: Fraud domains are domains that pretend to be legitimate, which can in turn be abused to perform phishing attempts.
- *Spam*: Spam domains are domains that send many unsolicited messages to internet users, often in a phishing attempt.
- *Spyware*: Spyware domains attempt to spread spyware, which is software that monitors system activity, which is subsequently reported to the creators of the spyware.
- *Mining*: Mining domains can install software on systems, which prompts the systems to start unauthorized mining of cryptocurrency.

(iv) *First seen*: the time stamp at which the domain was first seen on a public blacklist.

(v) *Last seen*: the time stamp at which the domain was last seen on a public blacklist.

ID	Domain	Source	Reason	First seen	Last seen
1297527	amazon.co.uk. security-check.ga	http://mirror1. malwaredomains.com /files/domains.txt	malware	2016-11-17 16:48:24	2018-10-16 06:26:06
1297528	autosegurancabrasil .com	http://mirror1. malwaredomains.com/ files/domains.txt	malware	2016-11-17 16:48:24	2018-10-16 06:26:06

Table 4.2: A sample overview of the malicious data supplied by TNO.

4.3 Data quality

Arguably, the most important aspects of data quality are *correctness* and *completeness*. Correctness refers to the degree to which the provided data entries contain correct information and completeness refers to the degree to which the data is complete. More incorrect data entries translate to poorer data correctness. Similarly, the more incomplete data entries there are, (e.g., data entries contain fields that are specified as null) the poorer the data completeness.

Considering the size of the data, a practical solution to assess correctness is by taking samples and manually checking whether the samples align with general expectations. For instance, among the most answered DNS requests in the passive DNS data set are the domains api.facebook.com, star.c10r.facebook.com, googleapis.com, google.com and apple.com. It is fair to say that this is in line with general expectations. While this test is restricted, it provides a certain element of trust in the validity of a part of the data. For now, we assume that the rest of the data is also correct.

In order to assess data completeness, columns in the data set can be queried for their distinct values and counts. If these counts sum up to the total number of entries in the data, then it can be said that the data for that specific column is complete. Here, this is done for the *RR Type* column in the passive DNS data set and for the *Reason* column in the malicious data set. Their respective distributions can be found in Figures 4.1 and 4.2. For both columns, the sum of the counts of all unique values is equal to the number of data entries in the entire data set, which indicates completeness for these respective columns. However, it should be noted that this data set may not be complete, due to some malicious domains not being detected and blacklisted.

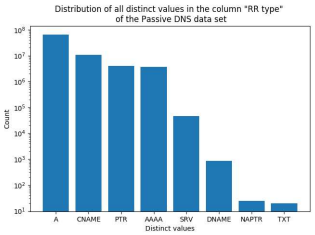

Figure 4.1: The distribution of the unique values for the 'RR Type' column in the passive DNS data set.

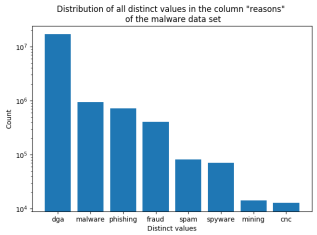

Figure 4.2: The distribution of the unique values for the 'Reason' column in the malicious data set.

5 Approach

The outline of the approach is first given in Section 5.1. Section 5.2 will explain how the domain-domain network is constructed and which research design choices were made for it. Section 5.3 delves into the different algorithms that are designed for risk score computation, given domain expert requirements and general model requirements, as well as a theoretical time and space complexity analysis of the proposed algorithms.

5.1 Outline of the approach

In summary, the approach follows the next steps:

1. Data collection (Sections 4.1, 4.2)
2. Preprocessing and feature extraction (Section 5.2.2)
3. Construction two-mode network (Section 5.2.3)
4. Projection and construction one-mode network (Section 5.2.4)
5. Scoring the domains (Section 5.2.5)
6. Requirements engineering (Section 5.3.1)
7. Design risk computation algorithms (Sections 5.3.2, 5.3.3, 5.3.4)

In Step 1, passive DNS data and malicious data are collected since they provide the necessary data to answer the research question. In the second step, relationship features are defined from the data, with which a two-mode network is constructed in Step 3. Step 4 discusses how this two-mode network is projected to a one-mode network (or a 'domain-domain network'). Additionally, the domains in the network are assigned initial scores based on a defined set

of rules in the fifth step. Furthermore, requirements for the risk computation algorithms are determined in Step 6. These algorithms are finally proposed in Step 7. A full overview of the network construction process (Steps 1 through 5) can be seen in Figure 5.1.

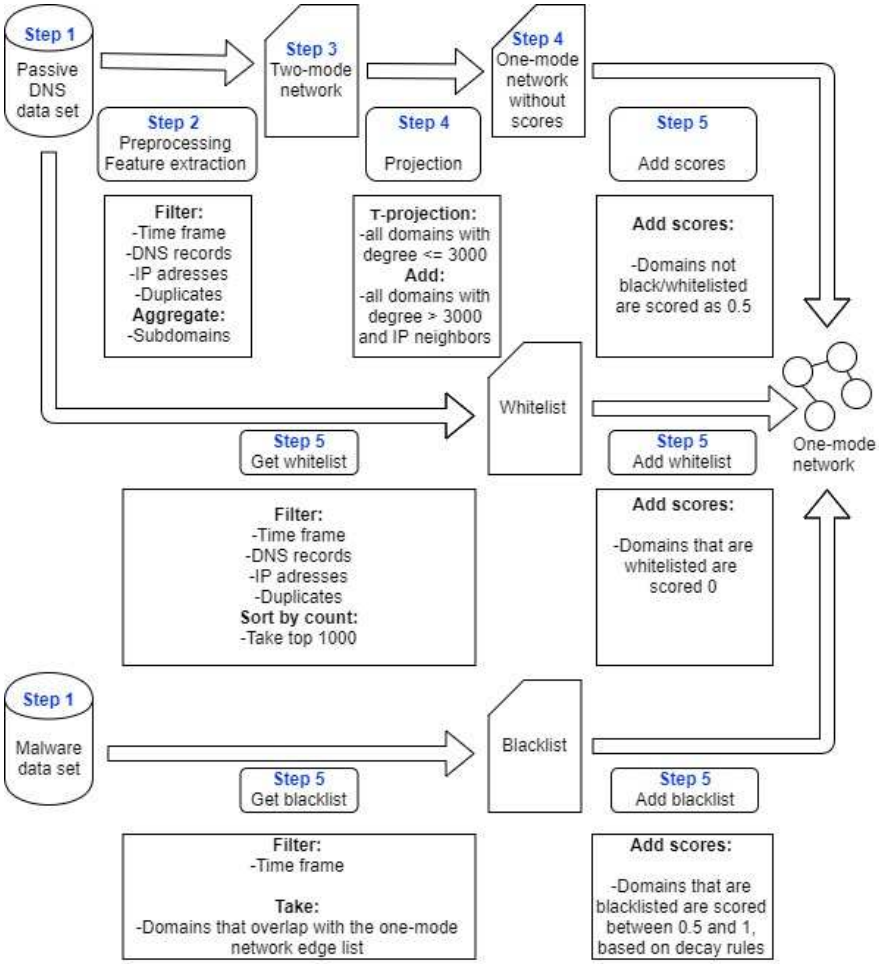

Figure 5.1: The entire construction process of the final network.

5.2 Network construction

This section concerns itself with the construction of the two-mode and one-mode network, as well as a motivation for the implementation choices that were made during network construction.

5.2.1 Overview network construction

Before we undergo the construction process, a few implementation choices are made that solve certain issues related to the network construction. These is-

sues and solutions are explained in this section.

First, domains are only temporarily flagged as malware, for instance by a recovered hostile takeover (*domain hijacking*) which prompted a domain to spread malware. This means that domains that are malicious today can be safe tomorrow. The passive DNS data set spans from 2012 until 2018, which is a time period in which the risk scores of domains may have fluctuated too heavily. It is therefore fruitful to work with a smaller snapshot of the time period in order to get a more stable picture of domains and their risk scores. If domains are malicious only once in a time period, then we can accurately estimate their initial risk score in order to compute their perceived risk score. We choose to take a month of data for the network construction, because we determine that one month is a feasible time frame in which a node's risk score is stable enough and contains enough data to build a network with, without nodes being flagged malicious multiple times.

The second issue is that a projection from a two-mode to a one-mode network may lead to an exploding number of links in the projection. This is because, in a two-mode network, all nodes that share the same connection with a node in the other set are connected to each other in the projected one-mode network. We choose to solve this by setting an upper limit for the node degree of nodes that can be projected to one-mode in order to keep the network size manageable with respect to the number of links. To do so, we also choose to apply preprocessing steps to the two-mode network in order to filter nodes out with a large degree (e.g., large public IPs) as much as possible.

The third issue is that there are no domains for which it can be said with 100% certainty that they are safe. Therefore, we create such a whitelist by taking the top k most frequently requested and answered domains that are not flagged as malware in the passive DNS data set. We check this list (we choose $k = 1000$) manually in order to see whether these k domains are indeed commonly requested and deemed safe in the public eye, such as google.com. All dubious domains are to be removed from the whitelist. With dubious, we mean all domains that we are not familiar with that seem as if they should not be requested frequently. Here, k depends on the risk that one is willing to take to introduce false positives to the data set. The larger the value of k, the larger the probability that this occurs. The malicious nodes are only temporarily flagged as malware within the specified time frame. The risk score depends on when the domain was last flagged as malware. If the domain in question is not again flagged as malware in the following days, then the risk score should lower as a function of the time passed.

5.2.2 Preprocessing steps for the passive DNS data set

We choose to take the most recent month in the data sets as our time frame, i.e., all A, AAAA, PTR and CNAME records that occur between December 5th, 2017 and January 5th, 2018 as our snapshot to work with. Records that contain public IP addresses with a large number of hosted domains are filtered from the network. To reduce the scale of the network, some private IP networks which host a high number of domains are also filtered from the

data. In this construction, the following (public) IP ranges (using CIDR notation [44]) are filtered: '127.0.0.0/8', '224.0.0.0/8 0'-to-239.0.0.0/8 (Class D), '192.168.0.0/16','10.0.0.0/8', '172.16.0.0/12' and '0.0.0.0/8'. Another preprocessing choice that is made is to aggregate all (sub)domains to second-level domain (SLD) and top level domain (TLD) nodes. For example, if there is a record that contains the domain `liacs.leidenuniv.nl`, it will be aggregated to `leidenuniv.nl`. There are exceptions, (e.g., domains ending with .co.uk) which are aggregated to the general country code TLD/SLD format (e.g., bbc.co.uk). This choice is made in order to keep the network relatively small and to prevent an exploding number of links when the network is later projected to a one-mode network. The benefit of this choice is a smaller computation time, but the drawback is that the risk spread in the network is then also limited to TLD/SLD, which means that the risk of individual subdomains cannot be assessed. From this point onward, when domains are mentioned they will be in this SLD/TLD format. This is the part of the domain name that can be registered by a private person or consumer.

5.2.3 Constructing the two-mode network

After taking all the necessary preprocessing steps, a two-mode (or 'domain-IP' network [45]) can be constructed from the remaining data by linking all domains and IPs together. An overview of the number of nodes and links of this network can be found in Table 5.1. Without filtering the IP ranges from the data and aggregating the domains, the domain-IP network would have counted more nodes and links. While this does not lead to a very significant reduction in size of the two-mode network, it does result into a significantly smaller projected network.

Nodes	Links
Domain-IP network (without preprocessing)	
13,988,326	14,362,751
Domain-domain network (without preprocessing)	
9,609,112	17,158,221,218+
Domain-IP network (with preprocessing)	
9,936,525	13,693,977
Domain-domain network (with preprocessing)	
7,140,913	746,713,939

Table 5.1: The number of nodes and links for the domain-IP and domain-domain networks.

5.2.4 Constructing the one-mode network

The next step is to create a one-mode network (or domain-domain network) from the domain-IP network using projection. A simple example of projection on a network is given in Figure 5.2.

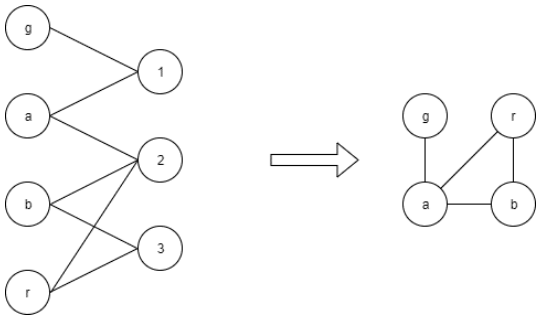

Figure 5.2: An example of a projection from two-mode to one-mode. On the left, we see our two-mode network with $\top = \{a, b, g, r\}$ and $\bot = \{1, 2, 3\}$. Here we define our projection rule to only draw a link between two elements in $\top$ when they share a connection with an element in $\bot$. The resulting projected one-mode network can be found on the right.

A link between two domain names can be drawn if and only if indirect (e.g., by shared IPs) and direct (i.e., by CNAME) connections between these domains exist. We define the domain-domain network as the network with all domains as nodes and a link between two domains if the connection between them is determined sufficiently strong. With sufficiently strong, we mean that we have to determine a rule that says when it is appropriate to place a link between two domains in the projection. Despite the filtering of IP addresses with a large degree, there are still domains and IPs with high degrees, which results into a network consisting of over 10^9 links. This demonstrates that such domain-domain networks can quickly become unmanageable for score computation.

Ideally, our algorithms are capable of computing a risk score for all domains in the network. Therefore, we try a projection method heuristic to preserve as much information as possible while keeping the number of links manageable in the projection. This heuristic starts off by setting cut-off points and projecting nodes and links only when certain conditions are met. The conditions can be set as follows. First, only take nodes with $< u$ links into consideration for projection. Then, project a link between two domains:

1. if domain x is linked to at least β number of IPs, of which α are shared with domain y
2. if domain x is linked to $< \beta$ links, but x is sharing s% of its neighbour links with y

After experimentation (see Section 6.1.1 for more information), our projection rule is to set $u = 3000$, $\alpha = 1$, $\beta = 1$ and $s = 100\%$ due to the limits of the scope of this research. A cut-off point of 3000 keeps the number of links below a billion in the domain-domain network. This size is in agreement with the expectations of cyber security experts for the scale of real-world domain-domain networks made from organizational DNS traffic data. All nodes with a degree below 3000 are projected fully to the domain-domain network. All IPs with a degree at or above 3000 are not projected, but added to the network in 'two-mode' as domain/IP pairs, i.e., just like how they exist in the domain-IP

network. These IPs are kept in the network and treated as if they are domains. The justification is that it may be useful to include IPs with a high degree, because some IPs may be hosting a high portion of malicious domains, which should automatically make other hosted domains with a lower risk score more suspicious of being malicious. This means that these IPs are also assigned initial scores which will be updated according to the scores of their neighbours and also that no post-processing will be required.

These IP address nodes will be given scores for computation and will be treated just like a 'regular' domain. The final step in the creation of the domain-domain network is the addition of CNAME records as links between nodes. The number of nodes and links of the domain-domain networks projected from both domain-IP networks are presented in Table 5.1. With the preprocessed domain-IP network, it results into a manageable network counting roughly 746.7 million links. However, without the explained preprocessing steps, the projected network is too large to compute. In an attempt to construct this network, the disk space occupation neared 2TB and the projected network counted 17,158,221,218 links before the attempt was broken off.

5.2.5 Scoring the domains in the network

The final step in the construction of the network is the assignment of risk scores to the domains. Whitelisted domains are in this case chosen by first filtering the Passive DNS data set on the chosen time frame (i.e., December 5^{th} to January 5^{th}), RR type (A/AAAA/PTR/CNAME) and the right IP ranges. Then we take the top k (we choose 1000). A frequency distribution for the 'Count' column is presented in Figure 5.3.

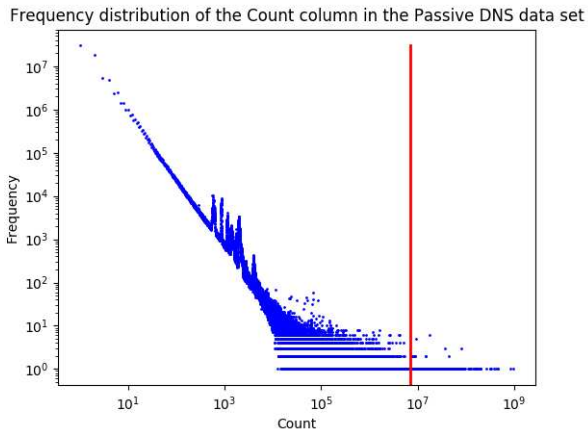

Figure 5.3: The frequency distribution for the 'Count' in the passive DNS data set. The red line shows the cut-off point if $k = 1000$.

It can be seen in Figure 5.3 that the frequency distribution follows a power law and that its 'tail' contains all the records with the highest count. We find that taking a top k of 1000 extracts a sizeable number of records from the tail,

namely all records with a count of at least 6,916,433. We also find that this cut-off point is right at the point before the distribution becomes denser and starts including significantly more records. It should be noted that the value of k depends on the risk that one is willing to take to introduce false positives into the data, accompanied with the trade-off that is made when manually checking each domain to see if they are truly safe domains that are commonly visited publicly. After this process is completed, each node is assigned a score of 0 (safe) for the domain-domain network.

The malicious nodes are taken from the malicious data set by first filtering the records on the chosen time frame. Next, all malware domains that also occur in the domain-domain network (3,819 in total) are added to the blacklist. Since a domain's risk score can vary a lot over a larger time window, it is chosen to score newly detected malware with a 1 (unsafe). A decay parameter is added to the score over time, starting from the day it is not seen as malware anymore (i.e., the 'last seen' entry in the record). In the view of cyber security experts, a node is not deemed as unsafe but 'neutral' after 7 days of its last appearance on public blacklists. This is a rough estimation which needs to be researched in practice, but is a sufficient starting point for this work. Therefore, it is chosen to set the decay parameter to $-\frac{1}{14}t$, with t being the number of days between the domain's last seen time stamp in the malicious data set and 'now' (also a time stamp), i.e., 'last seen' - 'now'. The score will keep decreasing by $\frac{1}{14}$ until 7 days have passed, after which the domain's score is 0.5 (neutral). For this snapshot of the data set, it is chosen to set 'now' to December 15^{th}. More formally, we abbreviate 'first seen' and 'last seen' to FS and LS respectively. The absolute difference in days between FS, LS and now can then be calculated in order to decide the scores of malware nodes:

$$\text{Score} = \begin{cases} \max(0.5, \frac{|LS-FS|}{|now-FS|} \cdot (1 - \frac{|now-LS|}{14})) & \text{if LS} < \text{now} \\ 1 & \text{if (FS} < \text{now} \wedge \text{LS} > \text{now)} \vee \text{FS} = \text{now} \vee \text{LS} = \text{now} \end{cases}$$

A sketch of the equation above is drawn in Figure 5.4, which provides a rough overview of how the risk scores are determined.

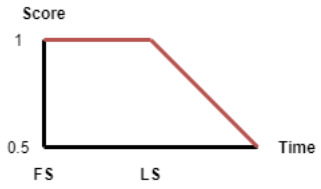

Figure 5.4: A sketch of the function shown in Equation 5.2.5 that shows how risk scores are determined. Given that FS and LS are fixed, 'now' can be placed alongside the horizontal axis in order to obtain the right risk score.

All other domains that are not white- or blacklisted are neutral, which means that nothing can be inferred about the initial risk of the domain. These other domains are therefore given a risk score of 0.5. This concludes the final step of the network construction.

5.3 Algorithm design

In this section, the algorithm requirements are presented in Subsection 5.3.1. Next we propose three different algorithms, LISC, NORMLISC and MAXDIF, in Subsections 5.3.2, 5.3.3 and 5.3.4, respectively. In each of these subsections, first the algorithms are introduced along with the underlying concepts that these algorithms work on. After this introduction, an explanation is given of the computation steps of each algorithm, followed by the algorithm in pseudocode. Next, an iteration of each algorithm is worked out on an example network in order to show how each algorithm works. This will all be discussed in the *static* case, i.e., the case where all nodes in the network are updated at once. Section 5.3.5 then presents how these algorithms will work in the *dynamic* case, i.e., how we think each algorithm would run in real-time by applying local updates next to global updates shown in pseudocode. Additionally, a local update step example is applied to the same network. Finally, Section 5.3.6 discusses the theoretical time and space complexity of each algorithm in the worst-case.

5.3.1 Algorithm requirements

The requirements can be split up into two categories, namely *domain expert requirements* and *model requirements*. Domain expert requirements are formulated by experts and reflect characteristics that the algorithm ideally possesses. Model requirements, on the other hand, state the required behavioural characteristics of the algorithm with respect to convergence. An overview of these requirements is provided below.

The domain (D) expert requirements:

1. **Static risk requirement**: Domains with a high risk score need to remain high.
2. **Influence requirement**: Nodes with a more certain score (i.e., scores closer to 0 and 1) are more influential than nodes with a more neutral score during computation.
3. **Susceptibility requirement**: Nodes with a more uncertain score (i.e., scores closer to 0.5) are more susceptible to change towards the score of its neighbours.
4. **Guilt by association (GBA) requirement**: Computation must be seen as a predominantly local problem: score computation is between neighbours. A node's risk should not affect nodes more than three steps away, dampening the risk influence with every step further.
5. **Risk spreading requirement**: Risk needs to spread from high-risk nodes to lower-risk nodes. After the algorithm is finished, 'risky' neighbourhoods/communities need to be distinguishable in the network.
6. **Update requirement**: In the event of a newly detected threat, it needs to be possible to update the score of a single node and its neighbours quickly.

The model (M) requirements:

1. **Convergence requirement**: The algorithm needs to converge within a few iterations to a state where risky neighbourhoods can ideally be distinguished from the rest in the network.

2. **No swapping requirement**: The algorithm should not compute a lower score for a node than for its neighbours which have higher scores in the initial state.

3. **Limited parameter requirement**: There should be a balance in the trade-off between the number of free parameters and guarantees about the model performance in different settings or networks.

4. **Degree requirement**: A node's position in the network matters and should be included in the computation. If a node has a high risk score, then automatically all the nodes that it can reach are also risky. Central nodes are therefore more impactful in the network.

5. **Scalability requirement**: The algorithm needs to be linearly scalable in terms of practical time and space complexity. It is acceptable if the theoretical worst-case time and space complexity are more complex than linear.

5.3.2 Local Influence and Susceptibility Computation

In this section we propose LISC, the Local Influence and Susceptibility Computation algorithm. This algorithm is inspired on DeGroot's model of reaching consensus (see Subection 3.2.1), which also makes use of a vector and a transition matrix. Normally, the transition matrix contains the level of *influence* that each node has on its neighbours. However, we extend this model by including a level of *susceptibility* for each node towards other nodes. The model aims to make the differences in risk score between neighbours smaller using influence and susceptibility.

Introduction LISC and example iteration

First we define special influence and susceptibility functions $f(s(v))$ and $g(s(v))$. These functions can be defined freely and take as input the risk score of a node and return the respective influence and susceptibility values for that node's risk score. While many variations of $f(s(v))$ and $g(s(v))$ are possible, we limit ourselves to linear functions of which the exact parameters are determined by experts such that they are in line with the stated requirements. These choices are therefore motivated by practical considerations of domain experts (see Section 6.3). For now, we assume that these functions return a constant '1', no matter the input.

Iteratively, scores can be updated by taking a domain's own score $s(v)_t$ at time step t, and the scores of its neighbours $s(w)_t$, for $w \in N(v)$. For LISC, the general update rule for a node v is formalized in Formula 5.1.

$$s(v)_{t+1} = s(v)_t + g(s(v)_t) \cdot \frac{\sum_{w \in N(v)} f(s(w)_t) \cdot (s(w)_t - s(v)_t)}{|N(v)|} \tag{5.1}$$

Here, $s(v)_{t+1}$, i.e., the score for v in the next time step (or iteration) is calculated by adding the average of the score differences between v and w to $s(v)_t$. This score difference is weighted by the individual influence of each neighbour w on v and is multiplied by v's susceptibility to adapt to its neighbours.

One iteration of LISC consists of applying Equation 5.1 for each domain in the network. In the overall model, the score vector $\vec{P}^t$ of size n consists of the scores of all domains at time step t. The $n \times n$ transition matrix T^t, with the elements T_{vw}, for $v, w = 1, 2, .., n$ consists of $f(s(v)_t)$ and $g(s(w)_t)$ for all $v, w \in V$ at time step t. Another requirement is an $n \times n$ incidence matrix I_{rc}, with element $P_{rc} = 1$ if there exists a link between v and w, and $P_{rc} = 0$ if otherwise. The algorithm runs by not only updating $\vec{P}^t$ per iteration, but also $\vec{T}^t$, since a node's influence and susceptibility changes every time its score changes if $f(s(v)_t)$ and $g(s(v)_t)$ return variable outputs. Let $\vec{e}_{|\vec{P}|}$ be a vector of ones of size $|\vec{P}| = n$. An overview of the complete algorithm can be found in Algorithm 1. For efficiency purposes, it is encouraged to implement the algorithm using sparse matrices.

Algorithm 1 LISC

Input: Score vector $\vec{P}^t$, Incidence matrix I, iterations i, step size α
Output: Score vector $\vec{P}^{t+1}$

procedure LISC
  for $t \leftarrow 0$ to i **do**
    $T^{t+1} \leftarrow F(\vec{P}^t) \otimes G(\vec{P}^t) \circ I$
    $D^{t+1} \leftarrow \vec{P}^t \ominus \vec{P}^t$
    $\vec{P}^{t+1} \leftarrow \vec{P}^t + \alpha(T^{t+1} \circ D^{t+1})^\top \cdot \vec{e}_{|\vec{P}|} \oslash (I \cdot \vec{e}_{|\vec{P}|})$
  return $\vec{P}^{t+1}$

In Algorithm 1, T^t is computed by applying $f(s(v))$ and $g(s(v))$ to all elements of $\vec{P}$. Let $F(\vec{V})$ and $G(\vec{V})$ be the respective functions that can calculate the influence and susceptibility values for all elements of a vector. Then we take the outer product ($\otimes$) between these vectors in order to compute all pairwise influence and susceptibility scores which will be spread in the network. A Hadamard product operation ($\circ$) with I is then applied to the resulting matrix in order to keep all influence and susceptibility scores between the node pairs which are linked to each other. Then we calculate all pairwise score differences by taking the outer subtraction ($\ominus$) of $\vec{P}^t$ with itself, stored as matrix D^t. The final step is then to take the Hadamard product between T^t and D^t, taking the transpose and multiplying the result by $\vec{e}_{|\vec{P}|}$. This results into a vector which contains all weighted score differences for all $v \in V$. Adding a portion of this vector (represented as the step size α), divided row wise ($\oslash$) by the total number of neighbours for every node ($I \cdot \vec{e}_{|\vec{P}|}$), to $\vec{P}^t$ results into a complete update of all nodes in $\vec{P}^{t+1}$.

An example of an iteration of Algorithm 1 can be found below. In this example, we choose Formulas 6.1 and 6.2 for $f(s(v))$ and $g(s(v))$ respectively. See Figure 5.5 for an example of a network. Numbers are rounded to three decimals.

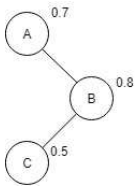

Figure 5.5: An example of a scored network of size $n = 3$.

In order to run the algorithm, first $\vec{P}^{(t)}$, I and $\vec{e}_n$ are initialized as:

$$P^{(t)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}, I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \vec{e}_{|P^{(\vec{t-1})}|} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

T^{t+1} is then calculated as $F(P^{(t)}) \otimes G(P^{(t)}) \circ I$:

$$T^{t+1} = \begin{bmatrix} 0.4 \\ 0.6 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0.6 \\ 0.4 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.24 & 0.16 & 0.4 \\ 0.36 & 0.24 & 0.6 \\ 0.6 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.16 & 0 \\ 0.36 & 0 & 0.6 \\ 0 & 0 & 0 \end{bmatrix}$$

The next step is to calculate $D^{t+1} = \vec{P}^{(t)} \ominus \vec{P}^{(t)}$:

$$D^{t+1} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} \ominus \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 & -0.1 & 0.2 \\ 0.1 & 0 & 0.3 \\ -0.2 & -0.3 & 0 \end{bmatrix}$$

The final step is the update $\vec{P}^{t+1} \leftarrow \vec{P}^{t} + \alpha(T^{t+1} \circ D^{t+1})^{\top} \cdot \vec{e}_{|\vec{P}|} \oslash (I \cdot \vec{e}_{|\vec{P}|})$:

$$\begin{aligned} P^{t+1} &= \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} + 1 \cdot \left(\begin{bmatrix} 0 & 0.16 & 0 \\ 0.36 & 0 & 0.6 \\ 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & -0.1 & 0.2 \\ 0.1 & 0 & 0.3 \\ -0.2 & -0.3 & 0 \end{bmatrix} \right)^{\top} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \oslash \left(\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 & 0.036 & 0 \\ -0.016 & 0 & 0 \\ 0 & 0.18 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \oslash \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0.036 \\ -0.016 \\ 0.18 \end{bmatrix} \oslash \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.036 \\ -0.008 \\ 0.18 \end{bmatrix} = \begin{bmatrix} 0.736 \\ 0.792 \\ 0.680 \end{bmatrix} \end{aligned}$$

Thus nodes A, B and C are updated with the new scores according to the node score update rule specified in Formula 5.1.

5.3.3 Normalized Local Influence and Susceptibility Computation

The following algorithm we propose is Normalized Local Influence and Susceptibility Computation, or NORMLISC. The general concept behind NORMLISC is the same as with LISC, but the risk scores are updated by fixing the degree to which a score adapts to the scores of its neighbours.

Introduction NORMLISC and example iteration

This algorithm is similar to LISC, as it computes risk scores using influence, susceptibility and a step size α. However, the degree to which a node adapts to itself (α) and to the scores of neighbours ($1 - \alpha$) is normalized to sum to 1. This way, the score of a node can be calculated by taking the weighted average between its own score and those of its neighbours. This eliminates the requirement of a matrix D^t that keeps track of the risk score differences between all neighbours. If $f(s(v)_t)$ and $g(s(v)_t)$ return constant values, then the transition matrix does not have to be recomputed which allows the model to be expressed as a Markov chain. Pseudocode for NORMLISC is shown in Algorithm 2.

Algorithm 2 NORMLISC

Input: Score vector $\vec{P}$**, Incidence matrix** I**, iterations** i**, step size** α
Output: Score vector $\vec{P}^t$

1: **procedure** NORMLISC
2: **for** $t \leftarrow 0$ to i **do**
3: $T^{t+1} \leftarrow F(\vec{P}^t) \otimes G(\vec{P}^t) \circ I$
4: $T^{t+1}(\vec{P}^{t+1}) \leftarrow T^{t\top} + \alpha \cdot eye(|\vec{P}|)$
5: $T^{t+1}(\vec{P}^{t+1})' \leftarrow T^{t+1}(\vec{P}^{t+1}) \oslash ((T^{t+1}(\vec{P}^{t+1}) \cdot \vec{e}_{|\vec{P}|}) \otimes \vec{e}_{|\vec{P}|})$
6: $\vec{P}^{t+1} \leftarrow T^{t+1}(\vec{P}^{t+1})' \cdot \vec{P}^t$
7: **return** $\vec{P}^{t+1}$

Here T^{t+1} is calculated exactly like in LISC, to which an identity matrix (or 'eye') of size $|\vec{P}|$ is added, multiplied with α (line 4). Here, we choose the same $F(\vec{V})$ and $G(\vec{V})$ as for LISC. The rows of this matrix $T^{t+1}(\vec{P}^{t+1})$ are then normalized to sum up to 1 by dividing it row-wise by the sums of its rows (line 5). Lastly, the score vector $\vec{P}$ is updated by multiplying it iteratively with the computed matrix $T^{t+1}(\vec{P}^{t+1})'$. Here, α is not chosen as the step size, but rather as a parameter that indicates how 'strongly' each node should converge towards its neighbours. If α is set to a low value (e.g., 0.1), then each node will mostly take on the values of its neighbours, and if a high value is chosen (e.g., $\alpha = 5$), then the node scores will converge very slowly. An example iteration of Algorithm 2 can be found on the next page. See Figure 5.6 for an example of a network. Numbers are rounded to three decimals.

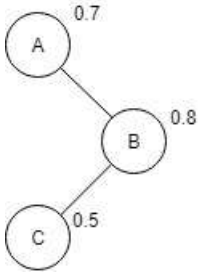

Figure 5.6: An example of a scored network of size $n = 3$.

In order to run the algorithm, first $\vec{P}^{(t)}$, I and $\vec{e}_{|\vec{P^{(t)}}|}$ are initialized as:

$$P^{(t)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}, I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \vec{e}_{|\vec{P^{(t-1)}}|} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

T^{t+1} is then calculated as $F(P^{(t)}) \otimes G(P^{(t)}) \circ I$:

$$T^{t+1} = \begin{bmatrix} 0.4 \\ 0.6 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0.6 \\ 0.4 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.24 & 0.16 & 0.4 \\ 0.36 & 0.24 & 0.6 \\ 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.16 & 0 \\ 0.36 & 0 & 0.6 \\ 0 & 0 & 0 \end{bmatrix}$$

The next step is to then add the 'eye' multiplied with $\alpha = 0.1$ ($T^{t+1}(\vec{P}^{t+1}) = T^{t\top} + \alpha \cdot eye(|\vec{P}|)$):

$$\begin{aligned} T^{t+1}(\vec{P}^{t+1}) &= \begin{bmatrix} 0 & 0.36 & 0 \\ 0.16 & 0 & 0 \\ 0 & 0.6 & 0 \end{bmatrix} + 0.1 \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0.36 & 0 \\ 0.16 & 0 & 0 \\ 0 & 0.6 & 0 \end{bmatrix} + \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \\ &= \begin{bmatrix} 0.1 & 0.36 & 0 \\ 0.16 & 0.1 & 0 \\ 0 & 0.6 & 0.1 \end{bmatrix} \end{aligned}$$

Next, the rows of $T^{(t+1)}(\vec{P}^{t+1})$ are normalized to sum up to 1 ($T^{t+1}(\vec{P}^{t+1})' = T^{t+1}(\vec{P}^{t+1}) \oslash ((T^{t+1}(\vec{P}^{t+1}) \cdot \vec{e}_{|\vec{P}|}) \otimes \vec{e}_{|\vec{P}|})$):

$$
\begin{aligned}
T^{(t+1)}(\vec{P}^{t+1})' &= \begin{bmatrix} 0.1 & 0.36 & 0 \\ 0.16 & 0.1 & 0 \\ 0 & 0.6 & 0.1 \end{bmatrix} \oslash \left(\left(\begin{bmatrix} 0.1 & 0.36 & 0 \\ 0.16 & 0.1 & 0 \\ 0 & 0.6 & 0.1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \\
&= \begin{bmatrix} 0.1 & 0.36 & 0 \\ 0.16 & 0.1 & 0 \\ 0 & 0.6 & 0.1 \end{bmatrix} \oslash \left(\begin{bmatrix} 0.46 \\ 0.26 \\ 0.7 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \\
&= \begin{bmatrix} 0.1 & 0.36 & 0 \\ 0.16 & 0.1 & 0 \\ 0 & 0.6 & 0.1 \end{bmatrix} \oslash \begin{bmatrix} 0.46 & 0.46 & 0.46 \\ 0.26 & 0.26 & 0.26 \\ 0.7 & 0.7 & 0.7 \end{bmatrix} \\
&= \begin{bmatrix} 0.217 & 0.783 & 0 \\ 0.615 & 0.385 & 0 \\ 0 & 0.857 & 0.143 \end{bmatrix}
\end{aligned}
$$

Lastly, the update step happens ($\vec{P}^{t+1} = T^{t+1}(\vec{P}^{t+1})' \cdot \vec{P}^{t}$):

$$
\vec{P}^{t+1} = \begin{bmatrix} 0.217 & 0.783 & 0 \\ 0.615 & 0.385 & 0 \\ 0 & 0.857 & 0.143 \end{bmatrix} \cdot \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.778 \\ 0.738 \\ 0.757 \end{bmatrix}
$$

Thus nodes A, B and C are updated with the new scores according to the node score update rule specified in Formula 5.1.

5.3.4 Maxima-based Diffusion

The third algorithm that we propose is Maxima-based Diffusion, or MAXDIF.

Introduction MAXDIF and example update

MAXDIF relies on the PageRank algorithm for convergence. The first difference with the previous algorithms is that it takes a directed network as input. All links between nodes are drawn from the node with the higher score (source) to the node with the lower score (target). If two adjacent nodes have the same score, then the link is bidirectional. The incidence matrix I' can be drawn with a helper function DRAW_INCIDENCE($\vec{P}, I$), which requires as input the $\vec{P}$ and the undirected incidence matrix I. Furthermore, MAXDIF does not use influence and susceptibility functions and has fewer free parameters. A pseudocode representation of MAXDIF is presented in Algorithm 3.

Algorithm 3 Draw Incidence Matrix I

Input: Score vector $\vec{P}$, Incidence matrix I
Output: Incidence matrix I'

1: **procedure** DRAW_INCIDENCE
2: $I' \leftarrow \vec{P} \ominus \vec{P}$
3: **for** i, j in I' **do**
4: **if** $I'[i,j] \geq 0$ **then**
5: $I'[i,j] = 1$
6: **else**
7: $I'[i,j] = 0$
8: $I' \leftarrow I' \circ I$
9: **return** I'

The idea behind MAXDIF is that it spreads the risk from nodes with higher risk scores to their neighbours with lower risk scores via outgoing links. All score differences are computed between each of such node pairs, whereby each node with outgoing links 'transfers' a percentage of the node pair score difference to its neighbours. These percentages are based on the PageRanks of all of the node's outgoing neighbours, normalized to 1. Nodes with multiple incoming links take the average value of all incoming transferred scores. This ensures that the computation is not only based on score differences between nodes, but also that nodes with a higher PageRank score in the network (i.e., nodes with a higher indegree, or nodes that are surrounded by more nodes with a higher PageRank) are given a higher priority to converge upwards. Another reason for using PageRank is that it ensures that nodes with a lower indegree, but central position in the network (i.e., it has incoming links from one or a few nodes with a high PageRank), are also given priority to converge upwards, as opposed to less central nodes in the network. This idea works like a 'diffusion model' of sorts, where risk is spread outwards and risky 'neighbourhoods' or 'communities' can be distinguished from the rest of the network after a few iterations. The pseudocode for MAXDIF can be found in Algorithm 4.

Algorithm 4 MAXDIF

Input: Score vector $\vec{P}$, Incidence matrix I, iterations i, delta δ, epsilon ϵ
Output: Score vector $\vec{P}^{t+1}$

1: **procedure** MAXDIF
2: **for** $t \leftarrow 0$ to i **do**
3: $I'^t \leftarrow \text{draw\_incidence}(\vec{P}^t, I)$
4: $\vec{R}^t \leftarrow \text{PageRank}(I'^t, \delta, \epsilon)$
5: $PR^t \leftarrow (\vec{R}^t \otimes \vec{e}_{|\vec{P}^t|})^\top \circ I'^t$
6: $D^t \leftarrow \vec{P}^t \ominus \vec{P}^t \circ I'^t$
7: $\vec{T}^t \leftarrow (PR^t \oslash ((PR^t \cdot \vec{e}_{|\vec{P}^t|}) \otimes \vec{e}_{|\vec{P}^t|}) \circ D^t)^\top \cdot \vec{e}_{|\vec{P}^t|}$
8: $\vec{P}^{t+1} \leftarrow \vec{P}^t + \vec{T}^t$
9: **return** $\vec{P}^{t+1}$

MAXDIF starts off by building the directed incidence matrix I' (line 3), which is fed to the PageRank algorithm with a user-defined value for the parameters

δ and ϵ. This returns PageRank vector $\vec{R}$, which contains the PageRank for all nodes in the network (line 4). Matrix PR contains the PageRank values of all neighbours that the nodes have a directed link to (line 5) and matrix D contains all the score differences between these pairs of nodes (line 6). After normalizing the PageRank values to sum up to 1, the Hadamard product is taken between PR and D, after which all row sums are taken of this matrix's transpose (line 7). These sums are then added to the score vector to conclude the first iteration of the algorithm (line 8). An example iteration of Algorithm 4 can be found below. See Figure 5.7 for an example of a network.

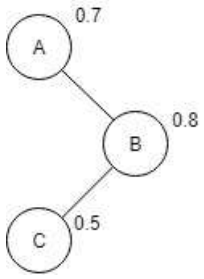

Figure 5.7: An example of a scored network of size $n = 3$.

In order to run the algorithm, first $\vec{P}^t$, I and $\vec{e}_n$ are initialized as:

$$P^t = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}, I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \vec{e}_{|\vec{P}^t|} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The first step of the iteration is to build the directed incidence matrix I' (see Algorithm 3):

$$I' = \vec{P}^t \ominus \vec{P}^t = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} \ominus \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 & -0.1 & 0.2 \\ 0.1 & 0 & 0.3 \\ -0.2 & -0.3 & 0 \end{bmatrix}$$

Next, apply DRAW_INCIDENCE($\vec{P}, I$) (change all fields in $I' < 0$ to 0 and all fields ≥ 0 to 1):

$$I' = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The final step in DRAW_INCIDENCE($\vec{P}, I$) is $I' = I' \circ I$:

$$I' = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Now that I' has been built it is time to feed I' to the PageRank algorithm (for $\delta = 0.85$, $\epsilon = 0.0001$) which returns the values:

$$\vec{R} = \begin{bmatrix} 0.213 \\ 0.574 \\ 0.213 \end{bmatrix}$$

Now, the PageRank values for all neighbours that each node is pointing to need to be stored ($PR = (\vec{R} \otimes \vec{e}_{|\vec{P}|})^\top \circ I'$):

$$
\begin{aligned}
PR &= (\vec{R} \otimes \vec{e}_{|\vec{P}^t|})^\top \circ I' \\
&= \left(\begin{bmatrix} 0.213 \\ 0.575 \\ 0.213 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)^\top \circ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0.213 & 0.575 & 0.213 \\ 0.213 & 0.575 & 0.213 \\ 0.213 & 0.575 & 0.213 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0.213 & 0 & 0.213 \\ 0 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

Additionally, the differences for each directly linked node pair needs to be calculated ($D = \vec{P}^t \ominus \vec{P}^t \circ I'$):

$$
D = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} \ominus \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -0.1 & 0.2 \\ 0.1 & 0 & 0.3 \\ -0.2 & -0.3 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0.1 & 0 & 0.3 \\ 0 & 0 & 0 \end{bmatrix}
$$

The final steps are normalizing the sum of the 'outgoing' PageRank values for each node to 1, multiplying these values row-wise by the difference between each node pair, summing all incoming computation values for each node, with which $\vec{P}^t$ will be added:

$$
\begin{aligned}
\vec{T}^t &= (PR^t \oslash ((PR^t \cdot \vec{e}_{|\vec{P}^t|}) \otimes \vec{e}_{|\vec{P}^t|}) \circ D^t)^\top \cdot \vec{e}_{|\vec{P}^t|} \\
&= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0.213 & 0 & 0.213 \\ 0 & 0 & 0 \end{bmatrix} \oslash \left(\left(\begin{bmatrix} 0 & 0 & 0 \\ 0.213 & 0 & 0.213 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \right. \\
&\left. \circ \begin{bmatrix} 0 & 0 & 0 \\ 0.1 & 0 & 0.3 \\ 0 & 0 & 0 \end{bmatrix} \right)^\top \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0.213 & 0 & 0.213 \\ 0 & 0 & 0 \end{bmatrix} \oslash \left(\begin{bmatrix} 0 \\ 0.426 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \circ \begin{bmatrix} 0 & 0 & 0 \\ 0.1 & 0 & 0.3 \\ 0 & 0 & 0 \end{bmatrix} \right)^\top \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0.213 & 0 & 0.213 \\ 0 & 0 & 0 \end{bmatrix} \oslash \begin{bmatrix} 0 & 0 & 0 \\ 0.426 & 0.426 & 0.426 \\ 0 & 0 & 0 \end{bmatrix} \right. \\
&\left. \circ \begin{bmatrix} 0 & 0 & 0 \\ 0.1 & 0 & 0.3 \\ 0 & 0 & 0 \end{bmatrix} \right)^\top \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ 0.1 & 0 & 0.3 \\ 0 & 0 & 0 \end{bmatrix} \right)^\top \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0.05 & 0 \\ 0 & 0 & 0 \\ 0 & 0.15 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0 \\ 0.15 \end{bmatrix}
\end{aligned}
$$

Thus nodes A, B and C are updated with the new scores:

$$\vec{P}^{t+1} = \vec{P}^t + \vec{T}^t = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.05 \\ 0 \\ 0.15 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0.8 \\ 0.65 \end{bmatrix}$$

5.3.5 Dynamic LISC, NORMLISC and MAXDIF

This section explains how the proposed algorithms can be used in the dynamic case. These implementations enable the algorithms to conform to requirement D6. In the dynamic case, the algorithms run iteratively until new information (e.g., a new threat appearance on a public blacklist) becomes available, in which case the risk scores have to be updated locally, i.e., the score of a node is updated along with those of its neighbours. This dynamic implementation for LISC, NORMLISC and MAXDIF can be found in the following subsections.

Dynamic LISC

Should the update happen during an iteration of the computation algorithm, the algorithm should break off after the current iteration, return the last fully computed $\vec{P}^t$, apply the update step, and continue running the next iteration with the updated $\vec{P}^t$. This can be done by checking a boolean NEWTHREAT if newly detected threats are available from external sources. If this is true, then the required information (i.e., which node n has to be updated with which score s) is requested using the function REQUEST(NODE, SCORE). This external information should be updated continuously. Should there be a newly detected threat, then the local update UPDATE.LISC is run. Should this not be the case, then LISC should run for another iteration. A full implementation of LISC with the update step can be found in Algorithm 5.

Algorithm 5 LISC.DYNAMIC

Input: Score vector $\vec{P}^t$, Incidence matrix I, iterations i, step size α, bool newthreat
Output: Score vector $\vec{P}$

```
1: procedure LISC.DYNAMIC
2:     for t ← 0 to i do
3:         if NEWTHREAT=False then
4:             P⃗^t ←LISC(P⃗^t, I, 1, α)
5:         else
6:             n, s ← REQUEST(NODE, SCORE)
7:             P⃗^t ←UPDATE.LISC(P⃗^t, I, n, s)
8:             t ← t − 1
9:     return P⃗^t
```

It is chosen to design local update methods that update the nodes in a similar fashion as the original algorithm, i.e., in this case by updating a node's score using predefined influence and susceptibility functions (again, we choose Formulas 6.1 and 6.2 for $f(s(v)_t)$ and $g(s(v)_t)$). It is desirable to not only update the score for that single node in $\vec{P}^t$, but also the scores of its neighbours and

the transition matrix T^t for all affected nodes (recomputed in line 3 of Algorithm 1). The required steps to fully update $\vec{P}^t$ are shown in Algorithm 6.

Algorithm 6 Update step LISC

Input: Score vector $\vec{P}$, Incidence matrix I, node n, score(n) s
Output: $\vec{P}$

1: **procedure** UPDATE.LISC ▷ Update $\vec{P}$
2: $\vec{P}_n \leftarrow s$
3: $T' \leftarrow f(s) \cdot I_{n,c}$
4: $T'' \leftarrow g(s) \cdot I^{\top}_{r,n}$
5: $D' \leftarrow (s \cdot \vec{e}_{|\vec{P}|} - \vec{P}^{\top}) \circ I_{s,c}$
6: $D'' \leftarrow (\vec{P}^{\top} - s \cdot \vec{e}_{|\vec{P}|}) \circ I^{\top}_{r,s}$
7: $B' \leftarrow T' \circ D'$
8: $B'' \leftarrow T'' \circ D''$
9: $C' \leftarrow I_{s,c} \circ F(\vec{P})$
10: $C'' \leftarrow I^{\top}_{r,s} \circ G(\vec{P})$
11: $\vec{P} \leftarrow \vec{P} + B' \circ C'' + B'' \circ C'$
12: **return** $\vec{P}$

An example update of LISC (Algorithm 6) can be found in Appendix A.

Dynamic NORMLISC

The dynamic implementation for NORMLISC follows the same logic as with LISC. New information is first checked using the boolean NEWTHREAT, with external information requested by the function REQUEST(NODE,SCORE). The full dynamic implementation of NORMLISC, including the update step can be found in Algorithm 7.

Algorithm 7 NORMLISC.DYNAMIC

Input: Score vector $\vec{P}$, Incidence matrix I, iterations i, step size α, bool newthreat
Output: Score vector $\vec{P}$

1: **procedure** NORMLISC.DYNAMIC
2: **for** $t \leftarrow 0$ to i **do**
3: **if** NEWTHREAT=False **then**
4: $\vec{P}^t \leftarrow$NORMLISC$(\vec{P}, I, 1, \alpha)$
5: **else**
6: $n, s \leftarrow$ REQUEST(NODE, SCORE)
7: $\vec{P}^t \leftarrow$UPDATE.NORMLISC$(\vec{P}, I, n, s)$
8: $t \leftarrow t - 1$
9: **return** $\vec{P}^t$

The update method for NORMLISC is also of the same character as LISC. That is, it updates single nodes based using influence, susceptibility (again, we choose Formulas 6.1 and 6.2 for $f(s(v))$ and $g(s(v))$) and α. Here, an update

also happens after breaking off after the current iteration of NORMLISC, returning the last fully computed $\vec{P}^t$, applying the update step and continuing running the next iteration with the updated $\vec{P}^t$. The update method is shown in Algorithm 8.

Algorithm 8 Update step NORMLISC

Input: Score vector $\vec{P}$**, Incidence matrix** I **,node** n**, score**(n) s**, alpha** α
Output: Score vector $\vec{P}$

procedure UPDATE.NORMLISC ▷ Update $\vec{P}$
  $\vec{P}_n = s$
  $T \leftarrow f(s) \cdot G(\vec{P})$
  $T' \leftarrow T \circ I_{n,c}$
  $T'_n \leftarrow \alpha$
  $T'' \leftarrow T' \oslash ((T'^{\top} \cdot \vec{e}_{|\vec{P}|}) \otimes \vec{e}_{|\vec{P}|})^{\top}$
  $T''_n \leftarrow 0$
  $\vec{P} \leftarrow \vec{P} \circ (\vec{e}_{|\vec{P}|} - T'') + s \cdot T''$
  return $\vec{P}$

An example update of NORMLISC (Algorithm 8) can be found in Appendix B.

Dynamic MAXDIF

The dynamic implementation of MAXDIF, including an update step can be found in Algorithm 9. This method is identical to the methods for LISC and NORMLISC (see Algorithms 5 and 9, respectively).

Algorithm 9 MAXDIF.DYNAMIC

Input: Score vector $\vec{P}$**, Incidence matrix** I**, iterations** i**, delta** δ**, epsilon** ϵ**, bool newthreat**
Output: Score vector $\vec{P}$

procedure MAXDIF.DYNAMIC
  for $t \leftarrow 0$ to i **do**
    if NEWTHREAT=False **then**
      $\vec{P}^t \leftarrow$MAXDIF$(\vec{P}, I, 1, \delta, \epsilon)$
    else
      $n, s \leftarrow$ REQUEST(NODE, SCORE)
      $\vec{P}^t \leftarrow$UPDATE.MAXDIF$(\vec{P}, I, s)$
      $t \leftarrow t - 1$
  return $\vec{P}^t$

The update method for MAXDIF cannot be completely of the same character as MAXDIF due to the incorporation of the PageRank algorithm in it, which makes a quick update infeasible. To remedy this, a node's risk is not spread outwards based on the PageRank of its neighbours, but rather evenly among the risky node's neighbours, i.e., there is no more priority given to nodes with a more central position in the network. For example, if a node n is updated

and has to spread its risk to 3 neighbours a, b, c, then each of the neighbours receives the score difference between itself and n multiplied with $\frac{1}{\text{outdegree}(a)}$. Additionally, a modified version of DRAW_INCIDENCE($\vec{P}, I$) for the update method is given in Algorithm 10.

Algorithm 10 Draw Incidence Update

Input: Incidence matrix I
Output: Incidence matrix I'

procedure DRAW_INCIDENCE_UPDATE
    Initialize Matrix I'
    for i, j in I **do**
        if $I[i, j] \geq 0$ **then**
            $I'[i, j] = 1$
        else
            $I'[i, j] = 0$
    return I'

See Algorithm 11 for the specific MAXDIF update method.

Algorithm 11 Update step MAXDIF

Input: Score vector $\vec{P}$**, Incidence matrix** I**, node** n**, score**(n) s ▷ Update $\vec{P}$
Output: $\vec{P}$

procedure UPDATE.MAXDIF
    $\vec{P}_n = s$
    $I' \leftarrow s - \vec{P}$
    $I'' \leftarrow$ DRAW_INCIDENCE_UPDATE$(I') \circ I_{c,n}$
    $I''' \leftarrow I'' \oslash ((I''^{\top} \cdot \vec{e}_{|\vec{P}|}) \otimes \vec{e}_{|\vec{P}|})^{\top}$
    $\vec{P} \leftarrow \vec{P} + I' \circ I'''$
    return $\vec{P}$

An example update of MAXDIF (Algorithm 11) can be found in Appendix C.

5.3.6 Time and space complexity

The theoretical time and space complexities for one iteration of LISC, NORM-LISC, MAXDIF will be determined in the worst case. This means that the algorithms are run on a complete network, i.e., a network counting n nodes and $\frac{n(n-1)}{2}$ links. Implementation then requires full matrices (which are of size $n \times n$), as opposed to sparse. Additionally, an iteration of MAXDIF relies on i iterations of PageRank in terms of time complexity, which runs in $\mathcal{O}(n^2)$ time per iteration, since it performs a vector-matrix multiplication. An overview of the theoretical time and space requirements for each algorithm can be found in Tables 5.2, 5.3 and 5.4.

LISC	Space	Time
$\vec{P}$	$\mathcal{O}(n)$	
I	$\mathcal{O}(n^2)$	
$\vec{e}_{\lvert\vec{P}\rvert}$	$\mathcal{O}(n)$	
$T^{t+1} \leftarrow F(\vec{P}^t) \otimes G(\vec{P}^t) \circ I$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$D^{t+1} \leftarrow \vec{P}^t \ominus \vec{P}^t$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$\vec{P}^{t+1} \leftarrow \vec{P}^t + \alpha(T^{t+1} \circ D^{t+1})^\top \cdot \vec{e}_{\lvert\vec{P}\rvert} \oslash (I \cdot \vec{e}_{\lvert\vec{P}\rvert})$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Order	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

Table 5.2: Theoretical time and space complexity analysis of LISC.

NORMLISC	Space	Time
$\vec{P}$	$\mathcal{O}(n)$	
I	$\mathcal{O}(n^2)$	
$\vec{e}_P$	$\mathcal{O}(n)$	
$eye(\lvert\vec{P}\rvert)$	$\mathcal{O}(n^2)$	
$T^{t+1} \leftarrow F(\vec{P}^t) \otimes G(\vec{P}^t) \circ I$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$T^{t+1}(\vec{P}^{t+1}) \leftarrow T^{t\top} + \alpha \cdot eye(\lvert\vec{P}\rvert)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$T^{t+1}(\vec{P}^{t+1})' \leftarrow T^{t+1}(\vec{P}^{t+1}) \oslash ((T^{t+1}(\vec{P}^{t+1}) \cdot \vec{e}_{\lvert\vec{P}\rvert}) \otimes \vec{e}_{\lvert\vec{P}\rvert})$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$\vec{P}^{t+1} \leftarrow T^{t+1}(\vec{P}^{t+1})' \cdot \vec{P}^t$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Order	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

Table 5.3: Theoretical time and space complexity analysis of NORMLISC.

MAXDIF	Space	Time
$\vec{P}$	$\mathcal{O}(n)$	
I	$\mathcal{O}(n^2)$	
$\vec{e}_P$	$\mathcal{O}(n)$	
$I'^t \leftarrow \text{draw\_incidence}(\vec{P}^t)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$\vec{R}^t \leftarrow \text{PageRank}(I'^t, \delta, \epsilon)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$PR^t \leftarrow (\vec{R}^t \otimes \vec{e}_{\lvert\vec{P}^t\rvert})^\top \circ I'^t$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$D^t \leftarrow \vec{P}^t \ominus \vec{P}^t \circ I'^t$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
$\vec{T}^t \leftarrow (PR^t \oslash ((PR^t \cdot \vec{e}_{\lvert\vec{P}^t\rvert}) \otimes \vec{e}_{\lvert\vec{P}^t\rvert}) \circ D^t)^\top \cdot \vec{e}_{\lvert\vec{P}^t\rvert}$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Order	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

Table 5.4: Theoretical time and space complexity analysis of MAXDIF.

Realistically, domain networks constructed from DNS data are sparse (and not complete) by nature and can hence be run using a sparse matrix implementation of the algorithms. Nonexistent links are then excluded from the computations. In that case, the algorithms will run as a function of the number of node pairs (direct neighbours) in the network, i.e., the algorithms will have a theoretical time and space complexity in the order of $\mathcal{O}(m)$ instead of $\mathcal{O}(n^2)$.

6 Experiments and results

This chapter will discuss the experiments and results. Section 6.1 explains the experimental setup required to answer the research question. Section 6.2 will discuss the results of the experiments regarding the network properties of the domain-IP and domain-domain networks constructed in Chapter 5. Section 6.3 shows the results of the experiments regarding the proposed algorithms. This chapter concludes with a discussion of the results in Section 6.4.

6.1 Experimental setup

This section is divided into two subsections, namely the experimental setup regarding network properties (as discussed in Subsection 6.1.1) and the experimental setup regarding the algorithm experiments (in Subsection 6.1.2).

6.1.1 Network properties

For the network properties, both the constructed domain-IP and domain-domain network (with preprocessing) will be analyzed. An overview of the number of nodes and links can be found in Table 6.1. In this section, we distinguish two different classes of experiments, namely *projection method experiments* and *network structure experiments*. These experiments show what needs to be taken into account for proper domain-domain network construction.

Projection method experiments

The following set of experiments concerns itself with the effectiveness of projection methods to obtain a domain-domain network. We experiment with several values for α, β, s and u (see Section 5.2.4) in order to see which of these parameters are effective in creating a network of manageable size, while retaining enough information.

Domain-IP network (with preprocessing)	
9,936,525 nodes	13,693,977 links
Domain-domain network (with preprocessing)	
7,140,913 nodes	746,713,939 links
Test network X	
7,140,913 nodes	186,678,440 links
Test network Y	
7,140,913 nodes	373,356,880 links
Small test network	
10 nodes	9 links
Medium test network	
30 nodes	35 links

Table 6.1: The number of nodes and links for the all networks used in the experiments.

Network structure experiments

We create several distributions of the constructed domain-IP and domain-domain network. First, we create a subdomain distribution in order to see how much of the network size (in terms of nodes) is compressed in the preprocessing step. Second, for the domain-IP networks, the degree distributions for all domains and IPs are plotted, followed by the degree distributions for all malicious domains and IPs. The third experiment is to plot the degree distribution of the domain-domain network in order to find out if the chosen projection method has any significant impact on the structure of the network. The final experiment regarding the network properties is to plot the average distance distribution for a random sample of domains, as well as the average distance distribution for all malicious nodes in the domain-domain network.

6.1.2 Algorithm experiments

In this section, we distinguish three different classes of experiments, namely *algorithm complexity experiments*, *algorithm convergence experiments* and *algorithm ranking experiments*. Due to the limits of the scope of this research, we limit ourselves to experiments with respect to the static case.

Algorithm complexity experiments

An important practical aspect of the designed algorithms is that they should terminate in feasible time using finite resources.

The theoretical time and space complexity (see Section 5.3.6) will be compared to the practical run time and space utilization of the algorithms by running them on the domain-domain network. Additionally, the algorithms are run on two test networks X and Y. These networks are constructed by taking the same number of nodes as the domain-domain network and placing links at random between them. Y has double the number of links as X, so that it can be determined in which order the time and space requirements for the algorithms

grow. The number of nodes and links for X and Y can be found in Table 6.1.

The algorithms are implemented in Python 3, using `NumPy` and `SciPy` sparse matrices for the matrix and vector calculations. The algorithms are run on a single thread of an `Intel XEON E5-2630v3` CPU (2.4 GHz) with access to 1.5TB of RAM.

Algorithm convergence experiments

Different parameter settings for the algorithms LISC, NORMLISC and MAXDIF will be run on a small and medium test network in order to find to which degree the algorithms conform to the requirements. The number of nodes and links for these networks can be found in Table 6.1. These experiments may lead to an overall picture that shows how these algorithms perform in comparison to each other with respect to convergence.

Algorithm ranking experiments

The algorithms are run for one iteration on the medium test network and placed in descending order by risk score, in order to find out if the algorithms 'rank' the nodes differently. If this is true, then it can be confirmed that algorithm choice has an impact on the final result and that additional experiments may be required in the future to find out which algorithm ranks the nodes 'best'.

6.2 Results—Network properties

This section concerns itself with the results of the experiments designed in Section 6.1.1.

Projection method experiments

For the projection methods, the small experiment with different values for α, β, s, u and the resulting size, in terms of the number of links, of the domain-domain network is summarized in Table 6.2.

Table 6.2 shows that taking different values for u has the largest impact on network size. However, it is not ideal to filter domains that are potentially risky from the network as it may result in an incomplete picture of the network. It can happen that a malicious domain is filtered from the final network by setting values greater than 1 for α or β, which in turn causes loss of relevant information in the projection. Therefore, as discussed in Section 5.2.4, α, β and s are set to 1 and u to 3000 to keep the number of links in the projection below 1 billion.

Network structure

During preprocessing, the subdomains are aggregated to domains, i.e., the step before the domain-IP network is constructed. The distribution for the number of aggregated domains can be found in Figure 6.1.

α	β	s	u	$\lvert E \rvert$
12	25	50%	1k	~29.3M
12	25	100%	1k	~46.6M
12	25	100%	5k	~107.4M
12	25	100%	10k	~436.4M
10	20	50%	1k	~29.5M
10	20	100%	1k	~45.6M
10	20	100%	5k	~107.9M
10	20	100%	10k	~436.9M
8	15	50%	1k	~30.7M
8	15	100%	1k	~47.2M
8	15	100%	5k	~109.7M
8	15	100%	10k	~438.7M
1	1	100%	3k	~746.7M

Table 6.2: Projecting the domain-IP network to a domain-domain network.

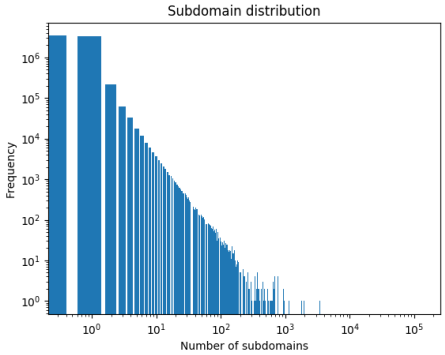

Figure 6.1: The number of subdomains in the network that is aggregated to domains. The leftmost bar indicates the number of domains not aggregated.

In Figure 6.1 we can see how much information is filtered from the network. There exists a downwards trend between the number of subdomains and their occurrence in the data. For instance, there are more than a million domains with one subdomain present in the data, but there are less than 10 domains with a thousand subdomains.

Next, the degree distributions for both domains and IPs can be found in Figures 6.2 and 6.3, respectively. These figures show that the largest part of the network consists of domains and IPs which have a degree below 1000. The 'tail' of the degree distribution contain nodes that have a higher degree and tend to be problematic during projections due to a rapidly increasing number of generated links between nodes. The domain and IP address degree distributions for all malware domains that occur within the same time period (i.e., between December 5th and January 5th) can be found in Figures 6.4 and 6.5 respectively. These figures show that the malware domains follow a similar but not identical degree distribution when compared to the full network.

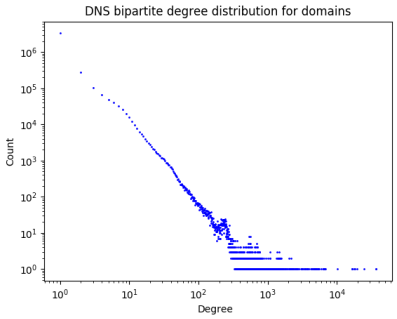

Figure 6.2: The degree distribution for all domains in the domain-IP network.

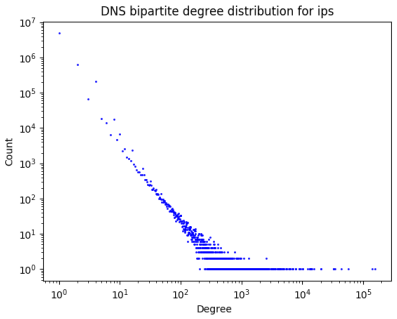

Figure 6.3: The degree distribution for all IP address nodes in the domain-IP network.

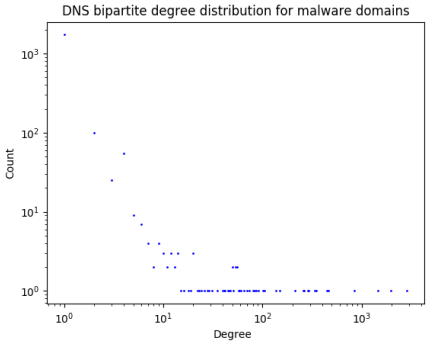

Figure 6.4: The degree distribution for all malware domains in the domain-IP network.

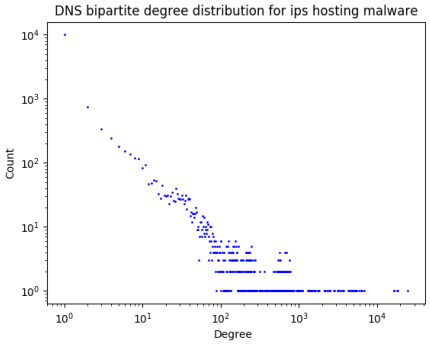

Figure 6.5: The degree distribution for all IP address nodes to which all malware domains are connected in the domain-IP network.

The degree distribution for the domain-domain network is presented in Figure 6.6. The chosen projection method may distort the power law that can be distinguished in Figures 6.2-6.5. However, this is an expected outcome since IPs with a degree of over 3000 in the domain-IP network are not projected, but placed 'as is' in the domain-domain network. This introduces domains in the domain-domain network that are not directly connected to other domains in the network, but via intermittent IPs. These IPs have a high degree (> 3000) but a low frequency. The domains that are not fully projected into the domain-domain network either have one neighbour (the IP in question) or are already projected to one-mode via another IP with a lower degree (< 3000). However, these domains could have had a higher degree in the one-mode projection had the projection rule not been in place. This suggests that many of the nodes in the degree distribution with a 'true' degree of over 3000 now exist with a lower degree in higher numbers, potentially explaining the 'kink' that can be seen at the 10^3 frequency and degree mark in the distribution.

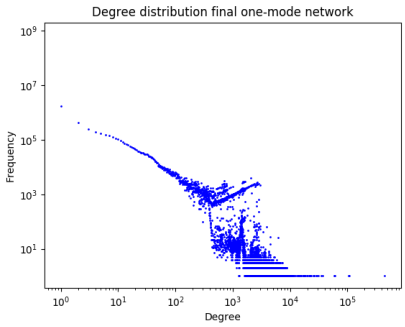

Figure 6.6: The degree distribution of the domain-domain network.

An overview of the average distance distribution for 1000 random nodes and all malicious nodes within the network can be found in Figure 6.7. The similarity of the distributions may indicate that the malware domains are 'hiding in plain sight', meaning that they can occur anywhere in the network. However, it can also be seen that the malicious distance distribution does not line up exactly with the randomly chosen distance distribution, suggesting that it is not ruled out that some global network effects could be in play. For instance, if malicious domains had existed in large clusters, then their average distance and degree distributions would deviate visibly from the distributions of a set of random nodes in the network. In this hypothetical situation, it could mean that it may for instance be possible to improve score computation by using clustering techniques or community detection algorithms, instead of following a purely local approach. This indicates that, based on these findings, one could justify that risk score computation in DNS networks be seen as a predominantly local problem. However, it should be noted that the distribution plots do not serve as hard evidence that score computation is a fully local problem. Note that the error bars in Figure 6.7 are wide due to some parts of the network not being accessible, i.e., the network is disconnected. When nodes are unreachable, the `graph-tool` implementation returns an average distance of 0.

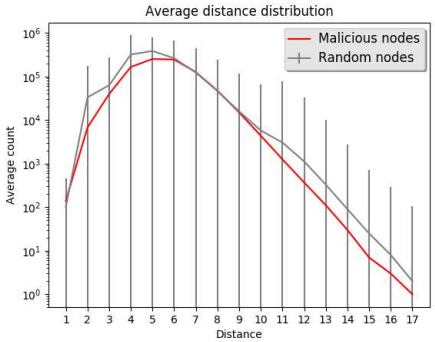

Figure 6.7: An overview of the average distance distribution for the chosen domains and all malicious domains. The error bars show the standard deviation for the randomly sampled domains.

6.3 Results—Algorithm experiments

This section concerns itself with the results of the experiments designed in Section 6.1.2.

An overview of the run time and memory occupation of all three algorithms over 10 iterations on the domain-domain network can be found in Figure 6.8.

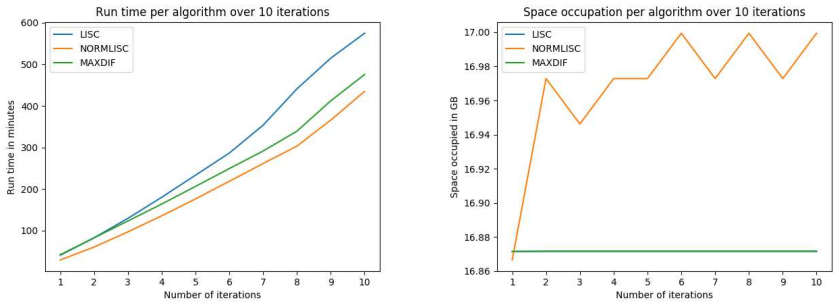

Figure 6.8: Run time and memory occupation of all three algorithms over 10 iterations on the domain-domain network.

Figure 6.8 shows that there is a distinguishable difference in terms of run time. LISC is the most expensive algorithm, NORMLISC is the least expensive algorithm and MAXDIF lies in the middle. It can be seen that the run times follow a linear progression in the first few iterations, after which the run times increase faster. It may be possible that this is due to the node scores having longer floating point numbers after a few iterations, which requires longer processing times in terms of arithmetic. For example, for a CPU it can be a lot easier to compute 101×101 than $00111110011100101010000111 \times 00111111001010001001101101$. A quick experiment verifies this hypothesis. In this experiment, two 10000×10000 matrices are filled randomly and multiplied with each other. This is repeated 10 times in the case where both matrices are randomly filled with 'float16' numbers between 0 and 1, and also 10 times in the case where both matrices are randomly filled with numbers between 0 and 1 with only 1 decimal. The first case has 'longer' numbers and it takes, on average, approximately 1.16 seconds longer to perform a matrix multiplication.

Figure 6.8 also shows that LISC and MAXDIF keep the space occupation stable throughout the iterations and remain at the same memory cost level (approximately 16.87 GB for both algorithms). However NORMLISC has varying space costs (between 16.865 and 17 GB) throughout the iterations, making NORMLISC $\sim 0.77\%$ more expensive than LISC and MAXDIF in this setting. It remains an open question as to why the memory consumption varies for NORMLISC. It could depend on the difference in implementation. However, such differences are very small.

In order to test if the time and space complexity indeed grows linearly with

the number of links in the case of sparse matrices, the algorithms are run on networks X and Y, with X having half the number of links compared to Y. The time and memory consumption of all three algorithms over 10 iterations on X and Y can be found in Figures 6.9 and 6.10 respectively.

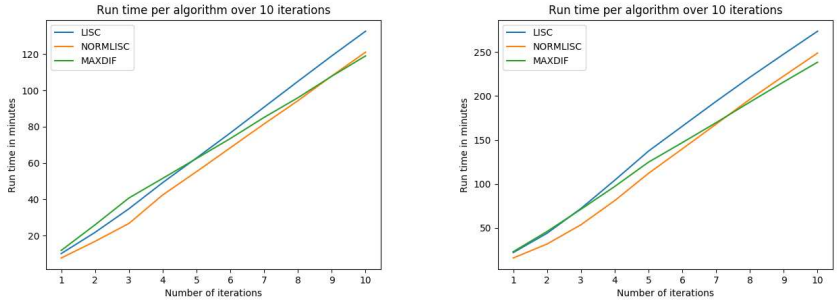

Figure 6.9: Run time of all three algorithms over 10 iterations. Left: X, right: Y.

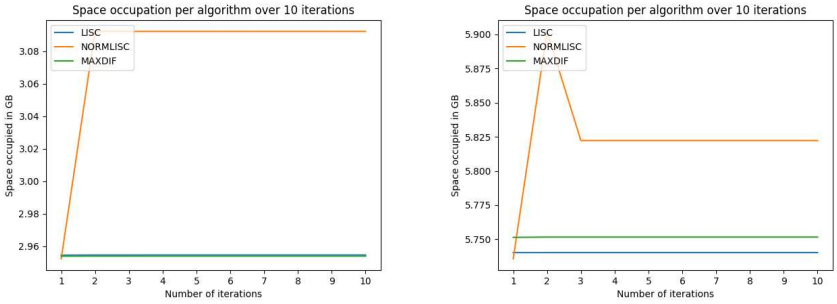

Figure 6.10: Space occupation of all three algorithms over 10 iterations. Left: X, right: Y.

Figure 6.9 shows that the run time per algorithm indeed appears to grow approximately linearly with the number of links in the network. It takes approximately 4 hours for each algorithm to finish running on Y, compared to approximately 2 hours for X. Figure 6.10 shows that the memory occupation also grows approximately linearly with the number of links in the network. For Y, the memory costs are almost double for each algorithm compared to X. In conclusion, it can be reasonably asserted that the practical time and space complexity for each algorithm is $\mathcal{O}(m)$.

Algorithm convergence experiments

The algorithms will be tested on two undirected test networks, namely the small and medium network (see Table 6.1). We choose smaller networks because we can plot the begin and end states in order to manually determine the

behaviour of the algorithms with respect to convergence. Doing this for large networks (e.g., 1000 nodes) is not trivial. These test networks are shown in Figure 6.11.

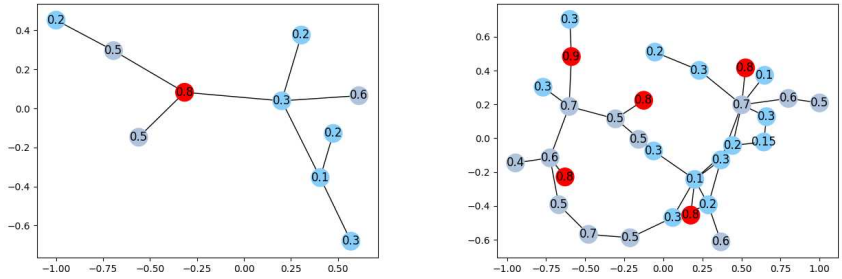

Figure 6.11: The test networks. Small (left) and medium (right).

For each of the experiments introduced in this section, the absolute sum of node score change is plotted per iteration in order to gain insight into the rate of convergence for all algorithms in different settings. Also each network state is plotted after [1, 2, 3, 4, 5, 10, 50, 100, 1000, 10000] iterations for LISC and NORMLISC in order to infer statements about the algorithms' behaviour regarding convergence. Network states for MAXDIF are plotted after 1,2,3 and 4 iterations. The experiments will be run on the small and medium test networks. A summary of all these experiments is provided in Table 6.3, after which an explanation for each of these experiments is given.

Algorithm	F and G	Dampening rate	Weighted mean	Iterations
LISC	F and G active F active, G constant F constant, G active F and G constant	None	No	10000
LISC	F and G active	$\frac{1}{t}$, $\frac{1}{1.05^t}$, $\frac{1}{2^t}$ $\frac{1}{t^2}$, $\frac{1}{t^5}$, $\frac{1}{t^{10}}$	No	10000
LISC	F and G active	None	Yes	10000
NORMLISC	F and G active F active, G constant F constant, G active F and G constant	None	No	10000
NORMLISC	F and G active	1.05^t, 2^t, t^2 t^5, t^{10}	No	10000
NORMLISC	F and G active	None	Yes	10000
MAXDIF	Not applicable	$\frac{1}{t^2}$, $\frac{1}{2^t}$	No	4

Table 6.3: All experiments on the test networks with respect to convergence.

Since LISC and NORMLISC rely on the influence and susceptibility functions f and g, it may prove helpful to not only perform these experiments in the scenario where both functions are 'constant' (i.e., they return '1' as explained in Subsection 5.3.2), but also in the cases where they are 'active'. We will perform experiments where one or both functions are constant or active. This may provide additional insight into how f and g act with respect to convergence.
A graphic representation of the chosen active functions can be found in Figures 6.12 and 6.13.

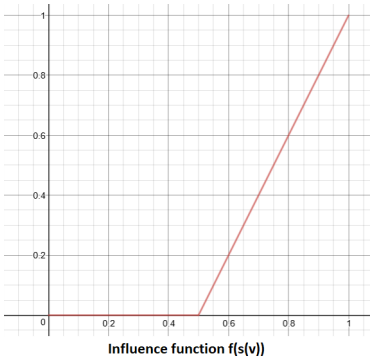

Figure 6.12: The influence function $f(s(v))$.

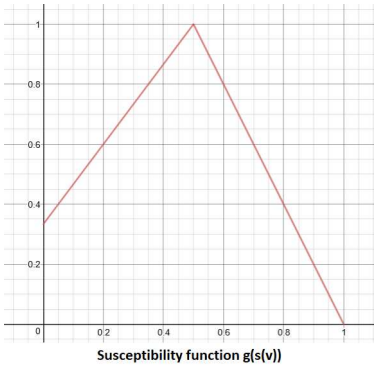

Figure 6.13: The susceptibility function $g(s(v))$.

The influence function is modelled to assign no influence to nodes with a risk score below or at 0.5 and full influence to nodes with a risk score of 1. Furthermore, $g(s(v))$ is modelled to assign full susceptibility to neutral nodes, 'moderate' susceptibility to safe nodes and little to no susceptibility to high risk nodes. The exact formulation of these functions, which correspond to the functions given in Figures 6.12 and 6.13, are given in Formulas 6.1 and 6.2 respectively. With these formulations, we meet requirements D1, D2 and D3 (see Section 5.3.1).

$$f(s(v)) = \begin{cases} 0 & \text{if } 0 \leq s(v) \leq 0.5 \\ 2 \cdot s(v) - 1 & \text{if } 0.5 < s(v) \leq 1 \end{cases} \tag{6.1}$$

$$g(s(v)) = \begin{cases} (4/3) \cdot s(v) + (1/3) & \text{if } 0 \leq s(v) \leq 0.5 \\ -2 \cdot s(v) + 2 & \text{if } 0.5 < s(v) \leq 1 \end{cases} \tag{6.2}$$

Ideally, the algorithm converges to a state where risky communities of nodes can be distinguished from the rest of the network instead of a global value. To remedy this, the convergence can be slowed down by adding a 'dampening rate' as a function of each iteration t in the algorithm. For LISC, this dampening rate is then multiplied with the amount of change that will be added to the score vector $\vec{P}$ for that specific iteration (multiplied with α in Algorithm 1). For instance, if a dampening rate of $\frac{1}{t}$ is chosen for LISC, then for each iteration $t = 1, 2, 3, .., y$, the 'change' vector is multiplied with $1, \frac{1}{2}, \frac{1}{3}, .., \frac{1}{y}$. For NORMLISC, this dampening rate needs to increase with every iteration since

this algorithm works on the basis that convergence slows down if a node itself gets a higher weight. For NORMLISC, dampening rates are subsequently multiplied with α (see Algorithm 2). The choice for these dampening rates are arbitrary and serve to demonstrate their effectiveness in curbing the convergence for each algorithms.

The next possibly informative experiment for LISC and NORMLISC is to take a weighted mean between the old score vector $\vec{P}^{t-1}$ and the new score vector $\vec{P}^{t}$. For each iteration, LISC and NORMLISC will take a weighted mean of $\vec{P} = 0.15 \cdot \vec{P}^{t-1} + 0.85 \cdot \vec{P}^{t}$ to see if the algorithm eventually converges to a steady state. This is inspired by the use of a dampening rate in the PageRank algorithm, for which the same rule is applied.

For the experiments involving MAXDIF, four iterations will be run each time, along with dampening rates since this algorithm also has the property of converging to a global score value for all nodes in the network. Another remark is that it can happen that a node with a low risk score (e.g., 0.2) is linked to a node with an even lower score (e.g., 0.1). In this specific case, MAXDIF must be run for at least 2 iterations. See Figure 6.14 for an example.

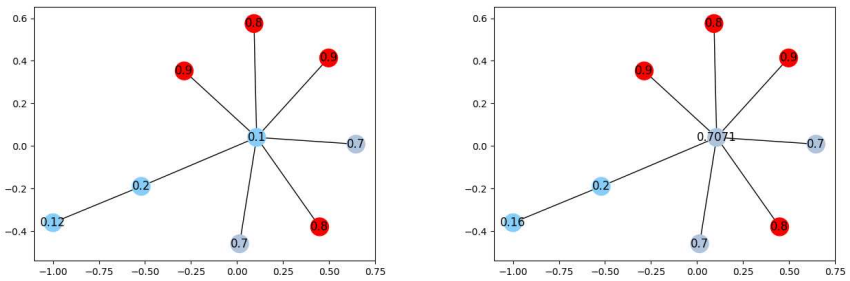

Figure 6.14: Left: initial state of the network. Right: the state of the network after 1 iteration (rate=1).

In Figure 6.14, the node with the lower score is linked to other nodes with very high scores, which causes the node with the lower score to converge upwards to a higher value in the first iteration, but leaving the other node (with a score of 0.2) completely unaffected due to the algorithm only computing node scores over directed links. This issue may be resolved by running the algorithm at least one more time and redirecting the links in the process. It may be possible that this type of problem will persist after two iterations. However, the most immediate cases such as the one depicted in Figure 6.14 are then already addressed. Additionally, with the introduction of a high dampening rate, nodes that are further removed would be affected minimally. In this scenario, the problem would persist even after many iterations of the algorithm.

Finally, the average network score of the algorithms will be plotted per iteration for the small and medium test network in order to gain more insight into

their convergence. This way, it can be verified if the node scores truly converge or diverge.

An overview of all algorithm experiments with respect to varying parameters on the small and medium network can be found in Table 6.4. The experiments range from varying with the dampening rates and using a weighted mean to setting f and g to return constant values in order to see the effects on network convergence.

Algorithm	Parameter values	Appendix
LISC	Varying with functions F and G	D
LISC	Varying with the dampening rate	E
LISC	Using a weighted mean	F
NORMLISC	Varying with functions F and G	G
NORMLISC	Varying with the dampening rate	H
NORMLISC	Using a weighted mean	I
MAXDIF	Varying with the dampening rate	J

Table 6.4: An overview of the parameter experiments for the designed algorithms, and in which Appendix to find the respective results.

For LISC (Appendix D-F), it can be seen that letting influence and susceptibility functions f and g return constant values (1 in these experiments) will lead to a violation of requirement M1. In the 'active case', i.e., when the dampening rate is 1 and f and g are not set to return constant values, it can be seen that LISC needs between 1 and 100 iterations to roughly converge to the same value for all nodes in the network. Ideally, the scores converge locally to the scores of the neighbours in as few iterations as possible. Varying with the dampening rate, given that they are a function of the number of iterations t, leads to quicker convergence of the scores due to the rate of change becoming smaller per iteration. The smaller the dampening rate, the fewer iterations are needed to update the scores locally. For instance, a dampening rate of $\frac{1}{t^{10}}$ still shows a concentration of risky nodes after 10000 iterations. Using a weighted mean ($\vec{P} = 0.15 \cdot \vec{P}^{t-1} + 0.85 \cdot \vec{P}^{t}$) per iteration roughly leads to a marginally quicker rate of convergence as in the standard case. It should be noted that LISC violates requirement M2, because risky nodes end up having a lower risk score than its neighbours after a few iterations.

For NORMLISC, (Appendix G-I). A difference with LISC is that varying with f and g still leads to conformance to requirement M1, even if both functions are constant. This is due to NORMLISC iteratively normalizing the transition matrix T to sum to 1 per row. Computing normalized weighted averages of risk scores between neighbours ensures that the scores do not diverge. Without the normalization step, nodes will 'over- or underparticipate', leading to every node trying to exercise more or less influence than their fair share in the network. Varying with the dampening rate has the same effect as with LISC. In this case, the larger the dampening rate, the fewer iterations are needed to converge the network scores. For instance, a dampening rate of t^{10} still shows a concentration of risky nodes after 10000 iterations, as is the case with

LISC. Using a weighted mean also leads to a marginally quicker convergence for NORMLISC. It should be noted that this algorithm also violates requirement M2.

For MAXDIF (Appendix J), only a few iterations are required due to most of the convergence being determined by the incorporation of the PageRank algorithm, thereby conforming to requirement M1. Adding a dampening rate also speeds up the convergence, leading to a few iterations (< 10) being required to compute risk scores. Additionally, MAXDIF also conforms to requirement M2.

Next, the average score is plotted over 10000 iterations for the small (Figure 6.15) and medium (Figure 6.16) test networks per algorithm.

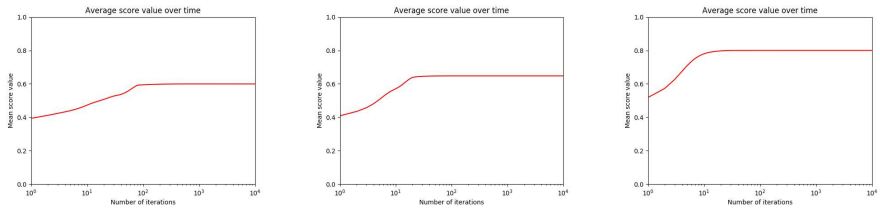

Figure 6.15: Left: mean global score of the small test network. Left: LISC. Middle: NORMLISC. Right: MAXDIF.

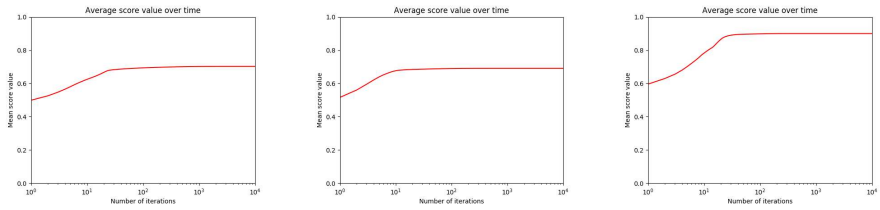

Figure 6.16: Left: mean global score of the medium network. Left: LISC. Middle: NORMLISC. Right: MAXDIF.

Figures 6.15 and 6.16 show that LISC and NORMLISC converge to an average global value that always lies somewhere between the minimum and maximum value of all initial scores in the network. This is due to the definition of f and g in Formulas 6.1 and 6.2, which are formulated in such a way that a node's risk score can lower in a next iteration. Since MAXDIF does not have this property, this means that lower scores can only increase. After a sufficient number of iterations, the entire network converges to the maximum initial score in the network. It can also be seen that the convergence to a global average happens after a low (i.e., $> 10^1$ and $< 10^2$) number of iterations for all algorithms. This indicates that each algorithm may only be run for a few (i.e., $< 10^1$) iterations in order to get a risk score estimation.

Algorithm ranking experiments

Here, the ranking outputs for each algorithm after 1 iteration on the medium test network are shown in Table 6.5. Each node is given an index between 0 and 29. It can be seen that significant mutual overlap exist between the rankings. For instance, the same 9 nodes occur in the top 10 rankings for all algorithms. These happen to be the nodes that initially start with a high score and their neighbours, which are assigned a higher score due to the GBA principle that appears to stay intact for the algorithms. In general, we expect newly detected threats to be assigned a slightly lower risk score than its risky neighbours, due to the algorithms converging properties. This means that the top 10 rankings stay relatively similar before and after a few iterations of any of the proposed algorithms.

TOP	LISC	NORMLISC	MAXDIF
1	6 (0.9 - 0.884)	6 (0.9 - 0.885)	6 (0.9 - 0.9)
2	9 (0.8 - 0.8)	9 (0.8 - 0.8)	5 (0.7 - 0.856)
3	12 (0.8 - 0.8)	12 (0.8 - 0.8)	3 (0.8 - 0.8)
4	3 (0.8 - 0.784)	26 (0.8 - 0.786)	9 (0.8 - 0.8)
5	26 (0.8 - 0.784)	3 (0.8 - 0.785)	12 (0.8 - 0.8)
6	5 (0.7 - 0.742)	5 (0.7 - 0.752)	23 (0.7 - 0.8)
7	23 (0.7 - 0.712)	23 (0.7 - 0.716)	26 (0.8 - 0.8)
8	10 (0.7 - 0.7)	10 (0.7 - 0.7)	4 (0.6 - 0.724)
9	4 (0.6 - 0.664)	4 (0.6 - 0.671)	10 (0.7 - 0.7)
10	2 (0.3 - 0.652)	8 (0.5 - 0.63)	8 (0.5 - 0.681)
11	22 (0.6 - 0.632)	22 (0.6 - 0.624)	13 (0.5 - 0.624)
12	8 (0.5 - 0.63)	18 (0.6 - 0.6)	22 (0.6 - 0.62)
13	18 (0.6 - 0.6)	7 (0.5 - 0.562)	18 (0.6 - 0.6)
14	13 (0.5 - 0.58)	13 (0.5 - 0.557)	25 (0.5 - 0.6)
15	7 (0.5 - 0.55)	2 (0.3 - 0.522)	7 (0.5 - 0.563)
16	25 (0.5 - 0.52)	25 (0.5 - 0.517)	11 (0.3 - 0.549)
17	11 (0.3 - 0.52)	17 (0.5 - 0.5)	0 (0.4 - 0.5)
18	17 (0.5 - 0.5)	11 (0.3 - 0.453)	16 (0.3 - 0.5)
19	0 (0.4 - 0.435)	0 (0.4 - 0.43)	17 (0.5 - 0.5)
20	1 (0.3 - 0.417)	1 (0.3 - 0.391)	2 (0.3 - 0.435)
21	19 (0.3 - 0.417)	19 (0.3 - 0.391)	1 (0.3 - 0.384)
22	24 (0.3 - 0.417)	24 (0.3 - 0.391)	28 (0.3 - 0.381)
23	28 (0.3 - 0.417)	28 (0.3 - 0.391)	14 (0.2 - 0.369)
24	20 (0.2 - 0.32)	16 (0.3 - 0.3)	19 (0.3 - 0.364)
25	16 (0.3 - 0.3)	20 (0.2 - 0.297)	24 (0.3 - 0.362)
26	15 (0.1 - 0.296)	15 (0.1 - 0.253)	29 (0.2 - 0.3)
27	14 (0.2 - 0.248)	14 (0.2 - 0.243)	20 (0.2 - 0.286)
28	27 (0.1 - 0.212)	29 (0.2 - 0.2)	21 (0.15 - 0.238)
29	29 (0.2 - 0.2)	27 (0.1 - 0.194)	15 (0.1 - 0.237)
30	21 (0.15 - 0.15)	21 (0.15 -0.15)	27 (0.1 - 0.166)

Table 6.5: The output rankings of each algorithm by index after 1 iteration on the medium test network. The brackets present the initial score and the score after 1 iteration.

6.4 Discussion

In this section, the experiments and results will be discussed with respect to the research question:

How can network science methods be used to efficiently compute risk scores of domains in dynamically evolving domain-domain networks?

In order to answer the research question, we first discuss our method to create domain-domain networks, along with the benefits and drawbacks of such a method. Next, we discuss the performance of the proposed algorithms based on the performed experiments, the degree to which they conform to the stated requirements and in which use case it is appropriate to use each algorithm. We conclude the section with the limitations with respect to domain-domain network creation and algorithm design.

Network creation

The DNS data set has undergone many preprocessing steps, from filtering IPs and aggregating subdomains to applying projection methods to construct a one-mode (domain-domain) network from DNS data. These steps are required to obtain a workable network with respect to time and space requirements. The network and the designed algorithms go hand in hand in this regard. The 'denser' the network (or, the more links in a network), the longer the algorithms will take to complete an iteration due to the time and space complexity of the algorithms. The drawback to the preprocessing approach is that the network does not take in all the information that might be required for optimal score computation. It could be possible that a subdomain (e.g., li-acs.leidenuniv.nl) starts with a high risk score, but its 'parent' (leidenuniv.nl) is initially recognized as a safe domain. The aggregation step could hence throw away relevant information. One possible method to circumvent this is by the design of an algorithm that is more scalable and can handle dense networks. The problem with this, however, is that Guilt by association modelling requires a computation between a node's score and the scores of its neighbours. The denser a network, the more node pairs exist, leading to an exploding number of calculations. It may be fruitful to split a dense network into smaller networks using a clustering algorithm before computing the scores on the subgraphs. This approach, however, introduces other assumptions and splitting criteria which will lead to other kinds of generalizations on the data. Another possible approach is to explore the feasibility of multithreaded CPU processing and GPU processing implementations with respect to computational speed up.

Algorithm performance, conformance to requirements and use cases

Aside from the preprocessing choices, the computation methods also have to conform to certain domain expert and model requirements (see Section 5.3.1). An overview of the conformance to the requirements for LISC, NORMLISC and MAXDIF can be found in Table 6.6, after which a discussion on each requirement is given. A '+' indicates that the algorithm conforms to the require-

ment and a'-' indicates the opposite. A ± indicates that the algorithm conforms to the requirement under certain circumstances.

Requirement	LISC	NORMLISC	MAXDIF
D1: Static risk requirement	±	±	+
D2: Influence requirement	+	+	-
D3: Susceptibility requirement	+	+	-
D4: GBA requirement	+	+	+
D5: Risk spreading requirement	+	+	+
D6: Update requirement	+	+	+
M1: Convergence requirement	+	+	+
M2: No swapping requirement	-	-	+
M3: Limited parameter requirement	-	-	+
M4: Degree requirement	-	-	+
M5: Scalability requirement	+	+	+

Table 6.6: The degree to which each algorithm conforms to the requirements as defined in Subsection 5.3.1.

D1: A drawback of LISC and NORMLISC is that malicious nodes do not necessarily retain their scores over several iterations, depending on the formulation of influence and susceptibility functions f and g. MAXDIF, however, never lowers a node score.

D2 and D3: The functions f and g provide a certain freedom in determining the behaviour of the model, which enables the end user to tune the model to their desires. This is an advantage which LISC and NORMLISC have over MAXDIF. However, no explicit guarantees can be made with respect to the convergence of the model. Running the model on a different data set or with a different use case will require expert insight and experimentation in order to reformulate f and g.

D4: All three algorithms are designed to model score computation as a predominantly local problem, since computation is fully based on a node's neighbours.

D5: All models also spread risk from risky nodes to less risky neighbours. This happens automatically for MAXDIF, but depends on the formulation of f and g for LISC and NORMLISC.

D6: All three algorithms also have a way to update the single nodes quickly (see Algorithms 6, 8 and 11).

M1: The results from the algorithm convergence experiments show that each algorithm has converging properties. However, this requires running the algorithm using a dampening factor and/or limiting the number of iterations. This is due to their innate property that the network scores converge towards each other. If the algorithms run indefinitely, then eventually all scores in the network converge to the same value.

M2: Another benefit of MAXDIF is the property that a node's score is at most raised with the absolute difference between itself and its neighbours, without the neighbours' score being able to decrease. This prevents the swapping behaviour that LISC and NORMLISC tend to show.

M3: The drawback of f and g is that they introduce more free parameters into the model, which makes tuning the model difficult. Since MAXDIF does not use these functions, it conforms more to the limited parameter requirement than LISC and NORMLISC

M4: A benefit of MAXDIF is that it inherently looks at the PageRank of its neighbours to assign more of its 'share' to nodes that have a more critical position in the network, which is not a property that LISC and NORMLISC possess.

M5: In terms of complexity, all algorithms have a time and space complexity in the order of $\mathcal{O}(m)$. This is not ideal if the desire is to run multiple iterations over a network consisting of billions of links. However, given that networks are sparse enough, the run time and space occupation are manageable since the algorithms do not need to run for many iterations to update the neighbours.

Limitations

All in all, MAXDIF has fewer tunable parameters and spreads can spread risk in a targeted way. The main drawback it has compared to LISC and NORMLISC is that it has less freedom in modelling interactions between high and low scores due to the absence of influence and susceptibility functions, such as low scores being able to 'pull' neighbour scores down. This freedom is immediately the main advantage of LISC and NORMLISC over MAXDIF because it enables the user to model complex interactions between nodes. The main drawback for LISC and NORMLISC is that this freedom introduces free parameters that have to be optimized. The formulation and validation of f and g can be very arbitrary, so guarantees cannot be made either about the convergence. Another drawback that LISC and NORMLISC have is the swapping phenomenon, which could be very undesirable in certain circumstances. Another trade-off that can be considered is whether or not normalization is required in the network to ensure an even impact for each node in the network. NORMLISC and MAXDIF have built-in normalization, whereas LISC does not.

Finally, the conformance to the requirements was mostly tested using test networks instead of real-world networks. Requirements D1-D6 and M3-M4 are about the designed properties of the algorithms (e.g., using influence and susceptibility or considering the position in the network). These properties hold true in any case, therefore they may also hold true for real-world networks. M1 and M2, however, are not evaluated using real-world networks. For M1, it may be reasonably assumed that the converging properties hold true due to the innate design of the algorithms to converge. For M2, it can be assumed that swapping may occur in real-world networks because swapping depends on the formulation on f and g. However, these assumptions about M1 and M2 holding true in a real-world scenario should be confirmed with experiments.

7 Conclusion

In this work multiple algorithms, LISC, NORMLISC and MAXDIF, are designed to compute risk scores for domains using a domain-domain network constructed from passive DNS data. These risk scores reflect the perceived risk of making connections to the domains and can be utilized to preemptively block malicious domains from being accessed in order to protect the IT network.

First, some preprocessing steps are applied before a domain-IP network is constructed by linking domains to IP addresses. Once the construction has happened, domains are projected to a domain-domain network if they share an IP address. However, the projection from domain-IP to domain-domain introduces an exploding number of links. To this end, heuristics such as filtering nodes with a large degree and domain aggregation assist in the creation of a network with a manageable number of links. Nodes with a degree < 3000 are projected directly to the domain-domain network, and IP nodes with a degree ≥ 3000 are placed 'as is' in the network along with their neighbours, in order to limit the number of projected links. Finally domains which are directly linked to each other via CNAME are also added to the one-mode projection.

Furthermore, risk scores of domains can be computed locally by using one of the three designed algorithms. The three algorithms all have a time and space complexity in the order of $\mathcal{O}(m)$. This means that the algorithms' time and space requirements scale linearly with the number of links in the network. In practice, sparse networks finish in a feasible time. However, this becomes increasingly harder when the network is denser. After experimentation, it is determined that the use of a dampening rate and a limited number of iterations has a higher likelihood of ensuring that the algorithms converge locally. If more freedom in modelling complex interactions between nodes is desired, either LISC or NORMLISC can be used, depending on whether or not normalization is required. This approach has the drawback that swapping can occur.

If the number of free parameters needs to stay limited and swapping is an undesired property, then MAXDIF can be used.

Furthermore, given a dampening rate of 1 and an unlimited amount of iterations, LISC and NORMLISC converge to a global value between the minimum and maximum initial score in the network. The exact height of this global score depends on the formulation of the influence and susceptibility functions f and g. MAXDIF, however, converges to the maximum initial score in the network, given the same conditions.

The main limitation of this work is that there is no ground truth to validate the outcome of the algorithms. The discussed algorithms present a method to estimate the risk to make connections to domains, but it cannot be said with certainty which type of estimation is more applicable in a real-world scenario.

The next steps could be to design preprocessing and projection heuristics such that more information is kept for the creation of the domain-domain network. Additionally, more types of relationships may be introduced to the network whereby some relationships are stronger than others. One way to deal with this is by transforming the domain-domain network to a weighted network, where each link has a weight denoting the 'strength' of the relationship. Another way to deal with this is by exploring multiplex networks, where only links are placed between nodes when they are deemed strong enough, depending on a threshold defined by experts. In addition, distributions can be drawn of score differences between node pairs throughout a number of iterations in order to get a clearer picture of the convergence of each algorithm. Also, more exploration could be done with respect to the properties of the domain-IP and the domain-domain networks, e.g., their connectivity, the size of their giant component and their sparseness. Furthermore, another goal is designing an algorithm that can model complex interactions (e.g., through influence and susceptibility functions) whilst also eliminating the swapping phenomenon. In the current case, a trade-off needs to be made about what requirement is more important. If eliminating swapping is more important, then it means that MAXDIF has to be picked, but also that more complex relationships cannot be modelled. Additionally, the experiments in this thesis were conducted in the static case. The algorithms also need to be tested in the dynamic case on real-world networks to see if they still conform to the listed requirements. Another step is to build a validation set refereed by cyber security experts. This validation set should contain the scores for each node in the network for two timestamps. This set can then be used to compare to the outcomes of the algorithms in order to measure algorithm performance. A validation set would also open up more ways to approach the problem of risk score computation, such as a machine-learned ranking problem.

Bibliography

[1] Internet Corporation For Assigned Names and Numbers. Domain name hijacking: Incidents, threats, risks, and remedial actions, 2005. http://archive.icann.org/en/announcements/hijacking-report-12jul05.pdf, Last accessed on 2019-02-28.

[2] Brian Cashell, William D Jackson, Mark Jickling, and Baird Webel. The economic impact of cyber-attacks. *Congressional Research Service Documents, CRS RL32331 (Washington DC)*, 2004.

[3] Lejun Fan, Yuanzhuo Wang, Xueqi Cheng, Jinming Li, and Shuyuan Jin. Privacy theft malware multi-process collaboration analysis. *Security and Communication Networks*, 8(1):51–67, 2015.

[4] Yury Zhauniarovich, Issa Khalil, Ting Yu, and Marc Dacier. A survey on malicious domains detection through dns data analysis. In *Proceedings of the 2018 ACM Conference on Computing Surveys (CSUR 2018)*, pages 67:1–67:36. ACM, 2018.

[5] Albert-László Barabási et al. *Network science*. Cambridge university press, 2016.

[6] Antal Van den Bosch, Toine Bogers, and Maurice De Kunder. Estimating search engine index size variability: a 9-year longitudinal study. *Scientometrics*, 107(2):839–856, 2016.

[7] Matthieu Latapy, Clémence Magnien, and Nathalie Del Vecchio. Basic notions for the analysis of large two-mode networks. *Social networks*, 30(1):31–48, 2008.

[8] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[9] Ron Aitchison. *Pro DNS and BIND 10*. Apress, August 2005.

[10] Margot Waty, Seth van Hooland, Simon Hengchen, Mathias Coeckelbergs, and Max De Wilde. How hot is .brussels? Impact of the uptake of the .brussels top-level domain name extension. *arXiv preprint arXiv:1606.04277*, 2016.

[11] Daniel Cheung. Common dns request types, 2018. https://support.opendns.com/hc/en-us/articles/227986607-Common-DNS-Request-Types, Last accessed on 2019-01-11.

[12] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in dns traffic. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, pages 715–720. IEEE, 2007.

[13] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS 2008)*. The Internet Society, 2008.

[14] Christian Seifert, Ian Welch, Peter Komisarczuk, Chiraag Uday Aval, and Barbara Endicott-Popovsky. Identification of malicious web pages through analysis of underlying dns and web server relationships. In *Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN 2008)*, pages 935–941. Citeseer, 2008.

[15] Matija Stevanovic, Jens Myrup Pedersen, Alessandro D'Alconzo, Stefan Ruehrup, and Andreas Berger. On the ground truth problem of malicious dns traffic analysis. *Computers & Security*, 55:142–158, 2015.

[16] Mark Felegyhazi, Christian Kreibich, and Vern Paxson. On the potential of proactive domain blacklisting. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET'10)*. USENIX Association Berkeley, 2010.

[17] Shuang Hao, Nick Feamster, and Ramakant Pandrangi. Monitoring the initial dns behavior of malicious domains. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (IMC 2011)*, pages 269–278. ACM, 2011.

[18] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, and Wenke Lee. Connected colors: Unveiling the structure of criminal networks. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2013)*, pages 390–410. Springer, 2013.

[19] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS 2011)*. The Internet Society, 2011.

[20] Daiki Chiba, Takeshi Yagi, Mitsuaki Akiyama, Toshiki Shibahara, Takeshi Yada, Tatsuya Mori, and Shigeki Goto. Domainprofiler: Discovering domain names abused in future. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*, pages 491–502. IEEE, 2016.

[21] Xin Hu, Matthew Knysz, and Kang G Shin. Measurement and analysis of global ip-usage patterns of fast-flux botnets. In *Proceedings of the 30th IEEE International Conference on Computer Communications (IEEE INFOCOM 2011)*, pages 2633–2641. IEEE, 2011.

[22] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *Proceedings of the 19th USENIX conference on Security (Security'10)*, pages 273–290, 2010.

[23] Issa Khalil, Ting Yu, and Bei Guan. Discovering malicious domains through passive dns data graph analysis. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 663–674. ACM, 2016.

[24] Kazumichi Sato, Keisuke Ishibashi, Tsuyoshi Toyono, Haruhisa Hasegawa, and Hideaki Yoshino. Extending black domain name list by using co-occurrence relation between dns queries. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET'10)*. USENIX Association Berkeley, 2010.

[25] Kensuke Fukuda and John Heidemann. Detecting malicious activity with dns backscatter. In *Proceedings of the 2015 Internet Measurement Conference (IMC 2015)*, pages 197–210. ACM, 2015.

[26] Yonghong Huang and Paula Greve. Large scale graph mining for web reputation inference. In *Proceedings of the IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP 2015)*, pages 363–369. IEEE, 2015.

[27] Zhiyun Qian, Zhuoqing Morley Mao, Yinglian Xie, and Fang Yu. On network-level clusters for spam detection. In *Proceedings of the 17th Network & Distributed System Security Symposium (NDSS 2010)*. The Internet Society, 2010.

[28] Matthew Thomas and Aziz Mohaisen. Kindred domains: detecting and clustering botnet domains using dns traffic. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)*, pages 707–712. ACM, 2014.

[29] Fariba Haddadi and A Nur Zincir-Heywood. Analyzing string format-based classifiers for botnet detection: Gp and svm. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC 2013)*, pages 2626–2633. IEEE, 2013.

[30] Jehyun Lee and Heejo Lee. Gmad: Graph-based malware activity detection by dns traffic analysis. *Computer Communications*, 49:33–47, 2014.

[31] Ching-Hsiang Hsu, Chun-Ying Huang, and Kuan-Ta Chen. Fast-flux bot detection in real time. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID 2010)*, pages 464–483. Springer, 2010.

[32] Pratyusa K Manadhata, Sandeep Yadav, Prasad Rao, and William Horne. Detecting malicious domains via graph inference. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS 2014)*, pages 1–18. Springer, 2014.

[33] Elihu Katz and Paul F Lazarsfeld. *Personal influence*. New York: Free Press, 1955.

[34] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

[35] Morris H DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974.

[36] Benjamin Golub and Matthew O Jackson. Naive learning in social networks and the wisdom of crowds. *American Economic Journal: Microeconomics*, 2(1):112–49, 2010.

[37] Bernd Heidergott, Jia-Ping Huang, and Ines Lindner. Naive learning in social networks with random communication. *TI Discussion Paper Series*, 2018(018/II), 2018.

[38] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[39] Monika Henzinger. Link analysis in web information retrieval. In *Bulletin of the Technical Committee on Data Engineering (TCDE 2000)*, volume 23, pages 3–9. IEEE, 2000.

[40] Allan Borodin, Gareth O Roberts, Jeffrey S Rosenthal, and Panayiotis Tsaparas. Link analysis ranking: algorithms, theory, and experiments. *ACM Transactions on Internet Technology (TOIT)*, 5(1):231–297, 2005.

[41] Stephen Oliver. Proteomics: guilt-by-association goes global. *Nature*, 403(6770):601, 2000.

[42] Yu Qian, Søren Besenbacher, Thomas Mailund, and Mikkel Heide Schierup. Identifying disease associated genes by network propagation. In *Proceedings of the 12th Asia Pacific Bioinformatics Conference on Systems Biology (APBC 2014)*, volume 8, pages 305–311. BioMed Central, 2014.

[43] Mnemonic. Defining cybersecurity, 2018. `https://www.mnemonic.no/`, Last accessed on 2018-12-14.

[44] Yakov Rekhter and Tony Li. An architecture for IP address allocation with CIDR. Technical report, 1993.

[45] Issa Khalil, Bei Guan, Mohamed Nabeel, and Ting Yu. Killing two birds with one stone: Malicious domain detection with high accuracy and coverage. *arXiv preprint arXiv:1711.00300*, 2017.

Appendices

A Update example LISC

An example update (see Algorithm 6) can be found below. See Figure 5.5 for an example network. Numbers with long tails are rounded to three decimals. See Figure 1 for an example of a network.

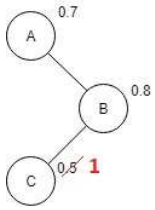

Figure 1: An example of a scored network of size $n = 3$. Node C's risk score is updated to '1' (from 0.5) in line 2.

Before the update is performed, $\vec{P}^{(t-1)}, I, \vec{e}_{|\vec{P}^{(t-1)}|}$, node n (n=2) and score(n) s ($s = 1$) have to be initialized:

$$P^{(t-1)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}, I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \vec{e}_{|\vec{P^{(t-1)}}|} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Now the network scores are updated as follows:

$$\vec{P}_n^{(t-1)} = 1 : \vec{P}^{(t-1)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix}$$

$$T' = f(s) \cdot I_{n,c} = 1 \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$T'' = g(s) \cdot I_{r,n}^{\top} = 0 \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$D' = (s \cdot \vec{e}_{|\vec{P}^{(t-1)}|} - \vec{P}^{(t-1)}) \circ I_{s,c} = \left(1 \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.7 \\ 0.8 \\ 1.0 \end{bmatrix} \right) \circ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.2 \\ 0 \end{bmatrix}$$

$$D'' = (\vec{P}^{(t-1)} - s \cdot \vec{e}_{|\vec{P}^{(t-1)}|}) \circ I_{r,s}^{\top} = \left(\begin{bmatrix} 0.7 \\ 0.8 \\ 0.7 \end{bmatrix} - 1 \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \circ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.2 \\ 0 \end{bmatrix}$$

$$B' = T' \circ D' = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 0.2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.2 \\ 0 \end{bmatrix}$$

$$B'' = T'' \circ D'' = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0 \\ -0.2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C' = I_{s,c} \circ F(\vec{P}^{(t-1)}) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \circ F\left(\begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0.4 \\ 0.6 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.6 \\ 0 \end{bmatrix}$$

$$C'' = I_{r,s}^{\top} \circ G(\vec{P}^{(t-1)}) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot G\left(\begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0.6 \\ 0.4 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.4 \\ 0 \end{bmatrix}$$

$$\begin{aligned} \vec{P}^{t} &= \vec{P}^{(t-1)} + B' \circ C'' + B'' \circ C' \\ &= \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.2 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 0.4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.4 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.08 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0.7 \\ 0.88 \\ 1 \end{bmatrix} \end{aligned}$$

B Update example NORMLISC

See Figure 2 for an example of a network.

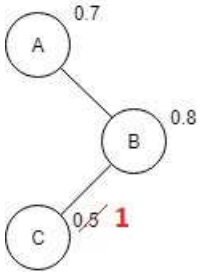

Figure 2: An example of a scored network of size $n = 3$. Node C's risk score is updated to '1' (from 0.5).

Before the network scores are updated, $\vec{P}^{(t-1)}, I, \vec{e}_{|\vec{P}^{t-1}|}$,node n ($n = 2$), score(n) s ($s = 1$) and α ($\alpha = 0.2$) need to be initialized:

$$P^{(t-1)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}, I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \vec{e}_{|P^{(\vec{t-1})}|} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Now the scores are updated as follows:

$$\vec{P}_n^{(t-1)} = s \rightarrow \vec{P}^{(t-1)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix}$$

$$T = f(s) \cdot G(\vec{P}^{(t-1)}) = f(1) \cdot G\left(\begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix}\right) = 1 \cdot \begin{bmatrix} 0.6 \\ 0.4 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.4 \\ 0 \end{bmatrix}$$

$$T' = T \circ I_{n,c} = \begin{bmatrix} 0.6 \\ 0.4 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.4 \\ 0 \end{bmatrix}$$

$$T'_n = \alpha : T' = \begin{bmatrix} 0 \\ 0.4 \\ 0.2 \end{bmatrix}$$

$$
\begin{aligned}
T'' &= T' \oslash ((T'^{\top} \cdot \vec{e}_{|\vec{P}|}) \otimes \vec{e}_{|\vec{P}|})^{\top} \\
&= \begin{bmatrix} 0 \\ 0.4 \\ 0.2 \end{bmatrix} \oslash \left(\left(\begin{bmatrix} 0 & 0.4 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)^{\top} \\
&= \begin{bmatrix} 0 \\ 0.4 \\ 0.2 \end{bmatrix} \oslash \left(0.6 \otimes \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)^{\top} \\
&= \begin{bmatrix} 0 \\ 0.4 \\ 0.2 \end{bmatrix} \oslash \begin{bmatrix} 0.6 \\ 0.6 \\ 0.6 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0.667 \\ 0.333 \end{bmatrix}
\end{aligned}
$$

$$
T''_n = 0 : T'' = \begin{bmatrix} 0 \\ 0.667 \\ 0 \end{bmatrix}
$$

$$
\begin{aligned}
\vec{P}^t &= \vec{P}^{(t-1)} \circ (\vec{e}_{|\vec{P}|} - T'') + s \cdot T'' \\
&= \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} \circ \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.667 \\ 0 \end{bmatrix} \right) + 1 \cdot \begin{bmatrix} 0 \\ 0.667 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 0.333 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.667 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 0.7 \\ 0.933 \\ 1 \end{bmatrix}
\end{aligned}
$$

C Update example MAXDIF

See Figure 3 for an example of a network.

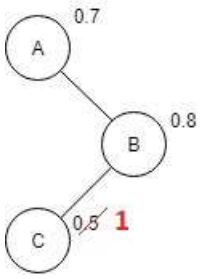

Figure 3: An example of a scored network of size $n = 3$. Node C's risk score is updated to '1' (from 0.5).

Before the update is performed, $\vec{P}^{(t-1)}$, I, $\vec{e}_{|\vec{P}^{(t-1)}|}$, node n (n=2) and score(n) s ($s = 1$) have to be initialized:

$$P^{(t-1)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}, I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \vec{e}_{|\vec{P^{(t-1)}}|} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Now the network scores are updated as follows:

$$\vec{P}_n^{(t-1)} = s \rightarrow \vec{P}^{(t-1)} = \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix}$$

$$I' = s - \vec{P}^{(t-1)} = 1 - \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0 \end{bmatrix}$$

$$I'' = \text{DRAW\_INCIDENCE\_UPDATE}(I') \circ I_{c,n} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$I''' = I'' \oslash ((I''^{\top} \cdot \vec{e}_{|\vec{P}^{(t-1)}|}) \otimes \vec{e}_{|\vec{P}^{(t-1)}|})^{\top}$$

$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \oslash \left(\left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \otimes \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right)^{\top} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \oslash \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\vec{P}^{t} = \vec{P}^{(t-1)} + I' \circ I''' = \begin{bmatrix} 0.7 \\ 0.8 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.3 \\ 0.2 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1 \\ 1 \end{bmatrix}$$

D LISC: Varying with F and G

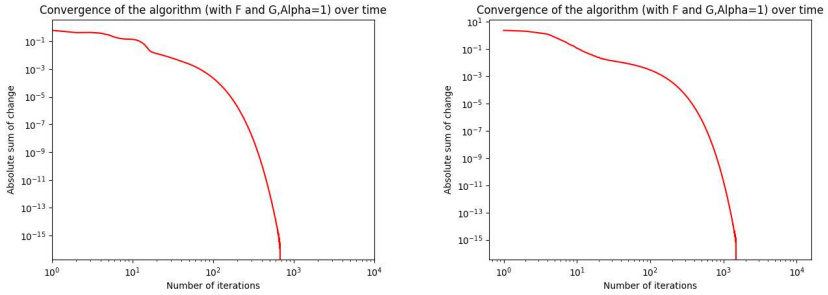

Figure 4: [F and G]: Left: convergence of medium network. Right: convergence of large network

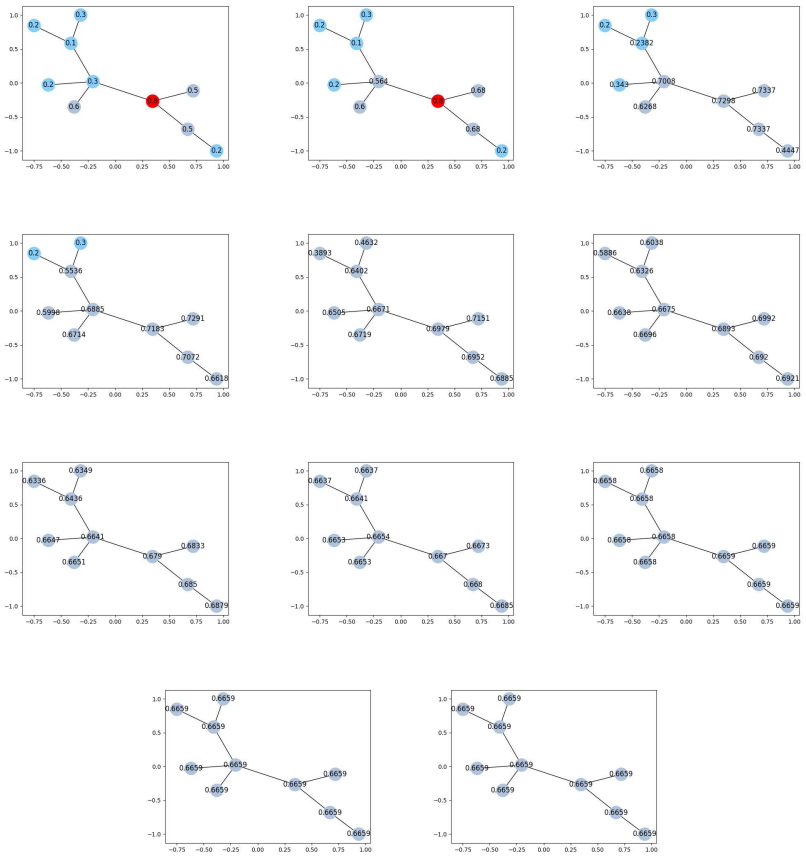

Figure 5: [F and G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

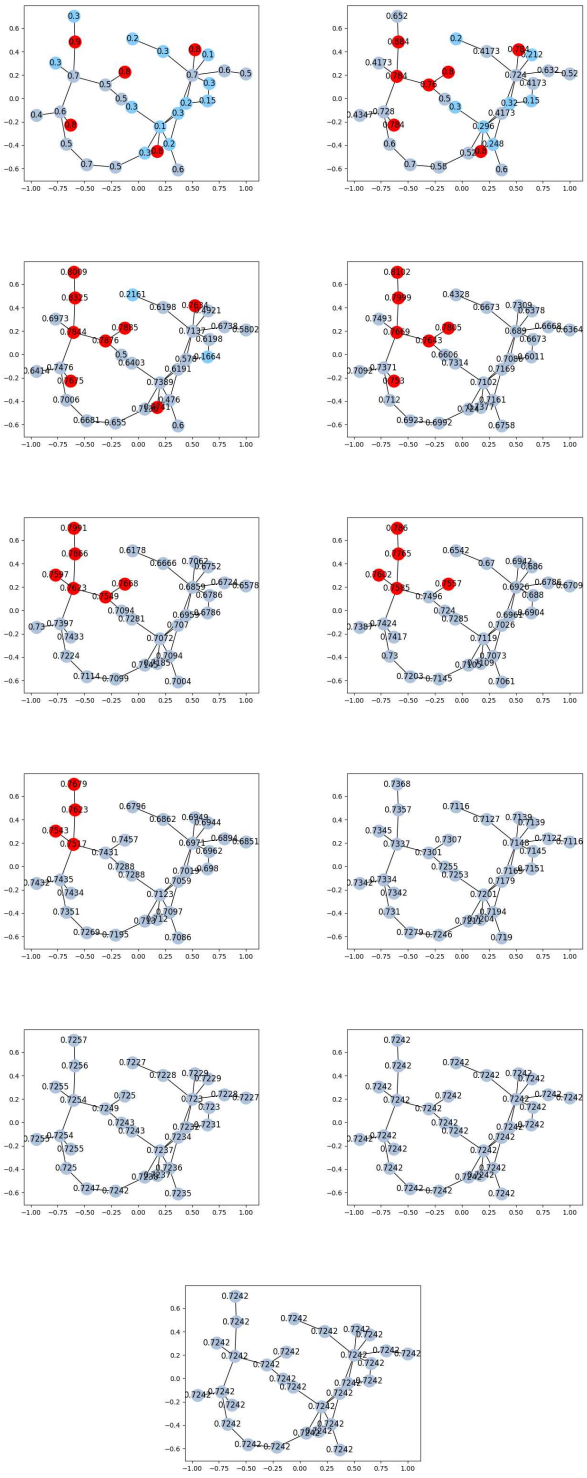

Figure 6: [F and G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

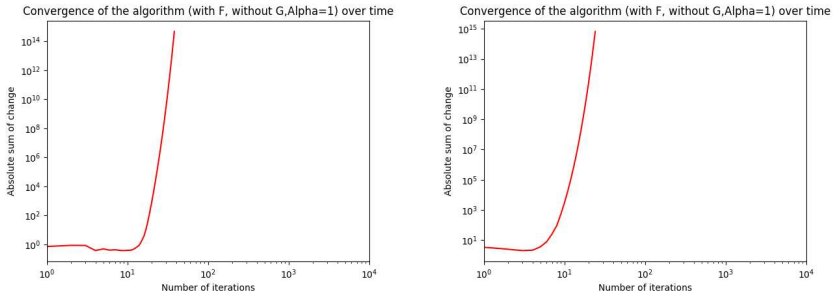

Figure 7: [With F, without G]: Left: convergence of medium network. Right: convergence of large network

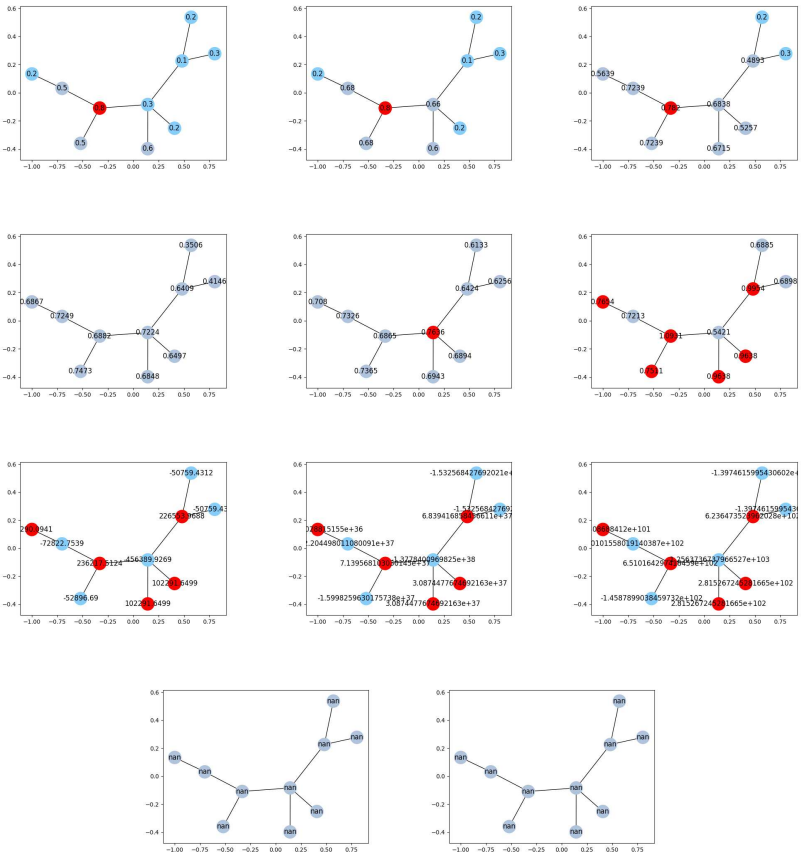

Figure 8: [With F, without G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

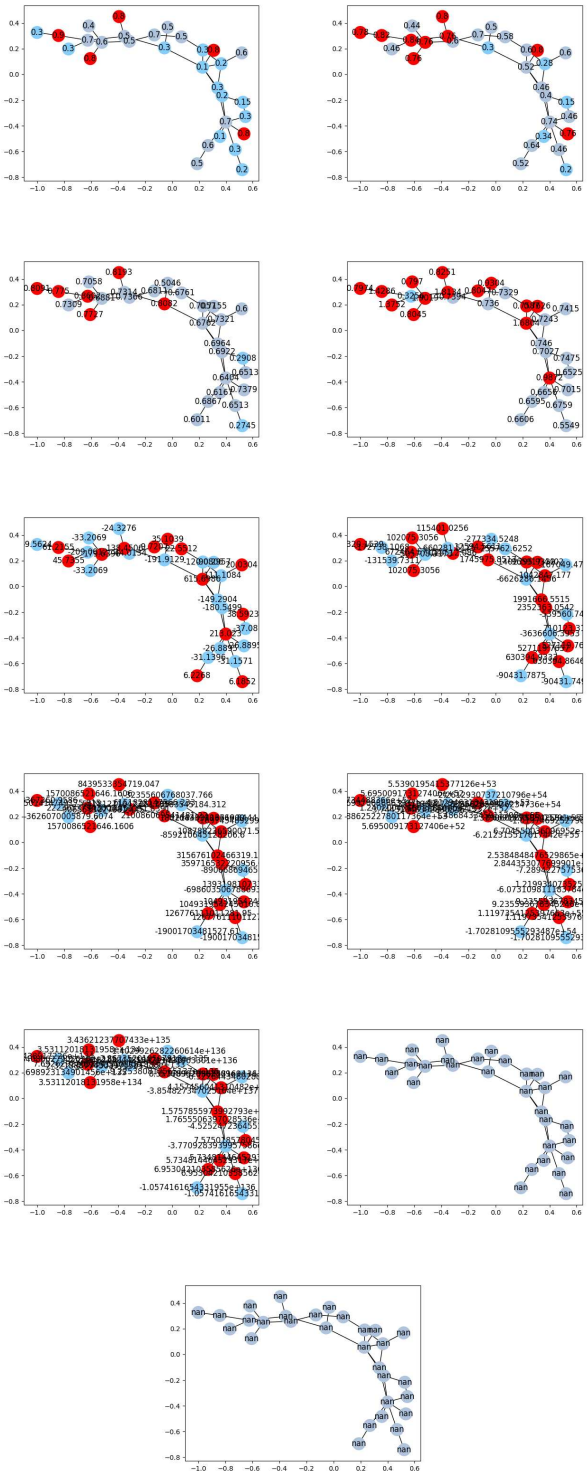

Figure 9: [With F, without G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

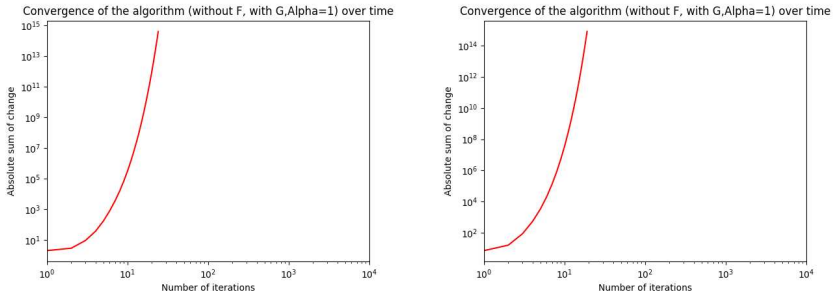

Figure 10: [Without F, with G]: Left: convergence of medium network. Right: convergence of large network

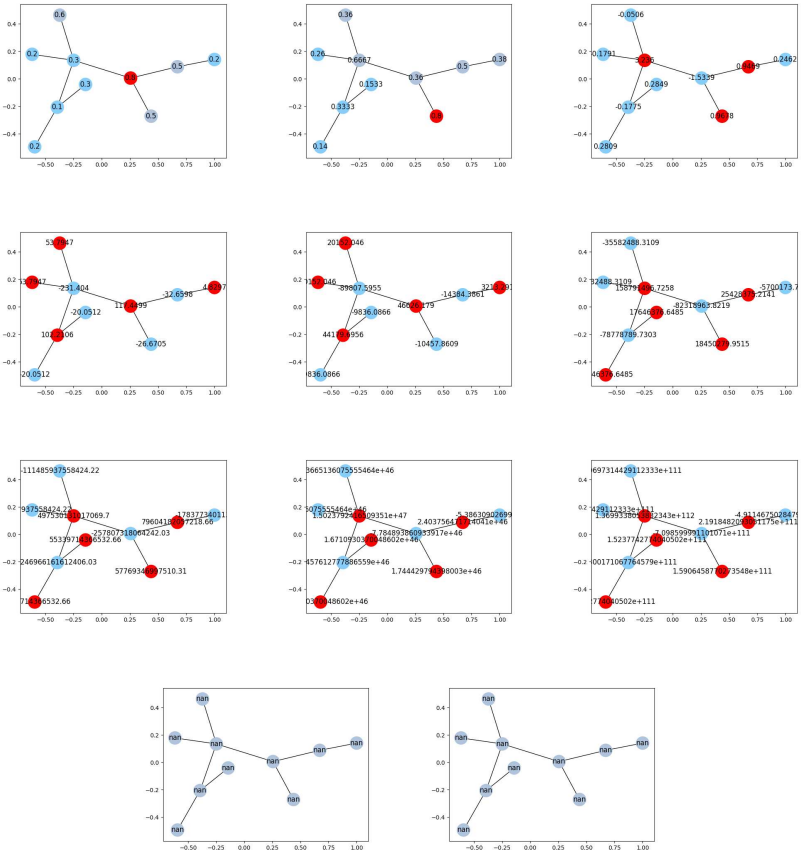

Figure 11: [Without F, with G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

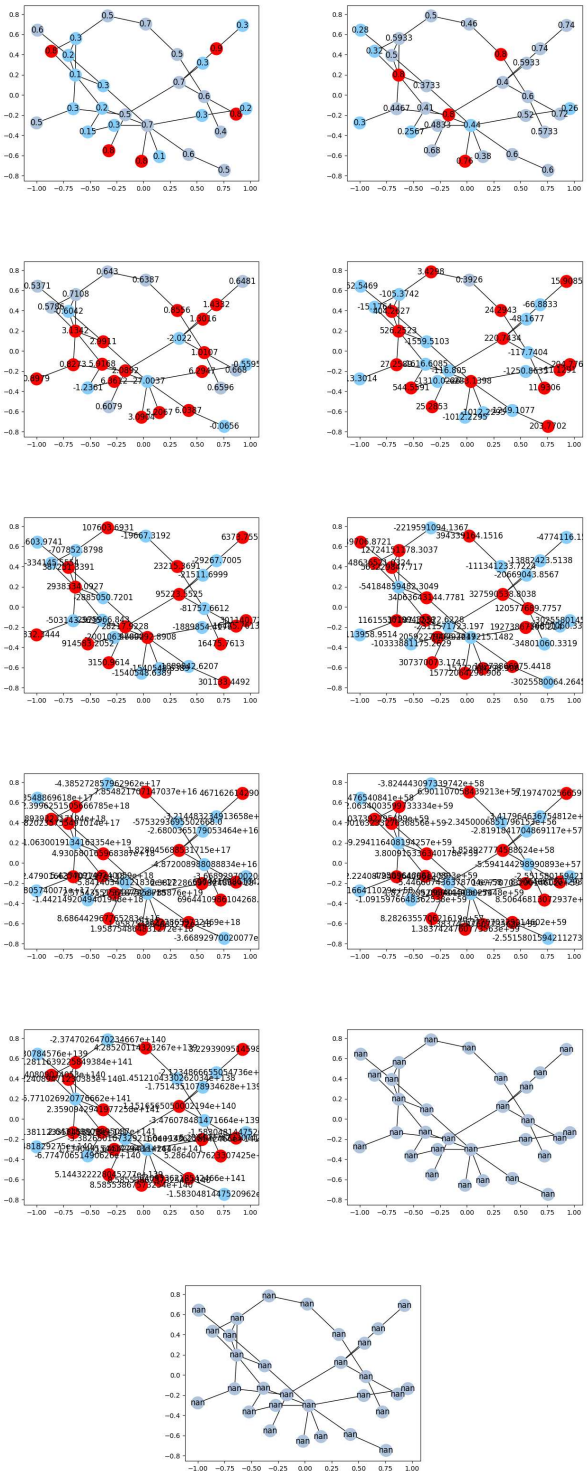

Figure 12: [Without F, with G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

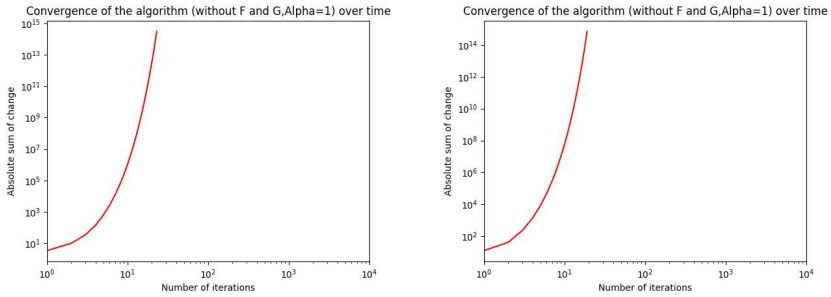

Figure 13: [Without F and G]: Left: convergence of medium network. Right: convergence of large network

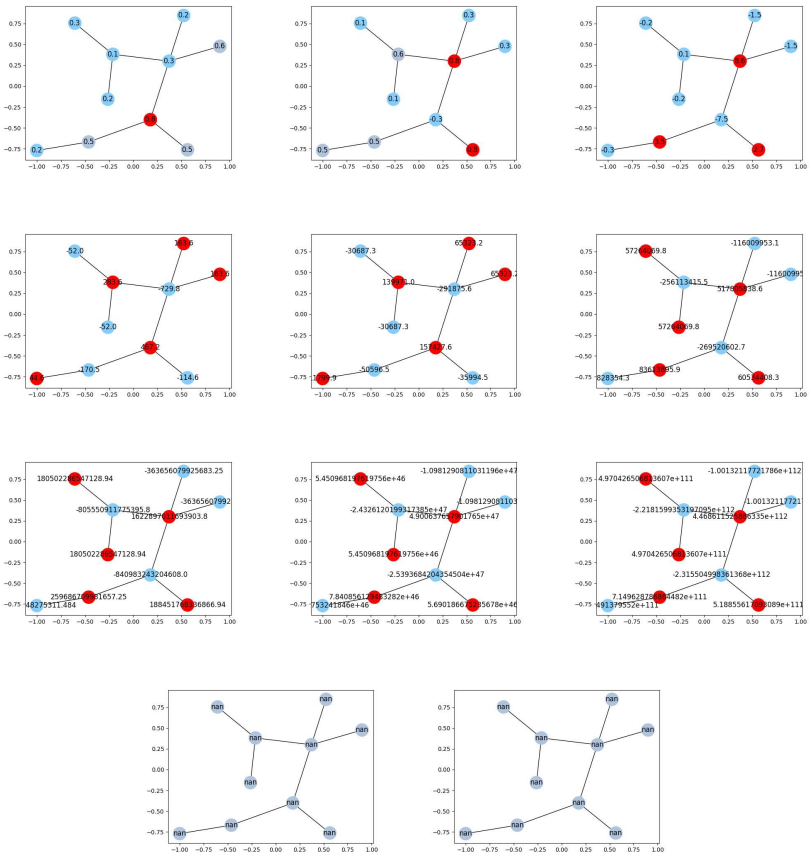

Figure 14: [Without F and G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

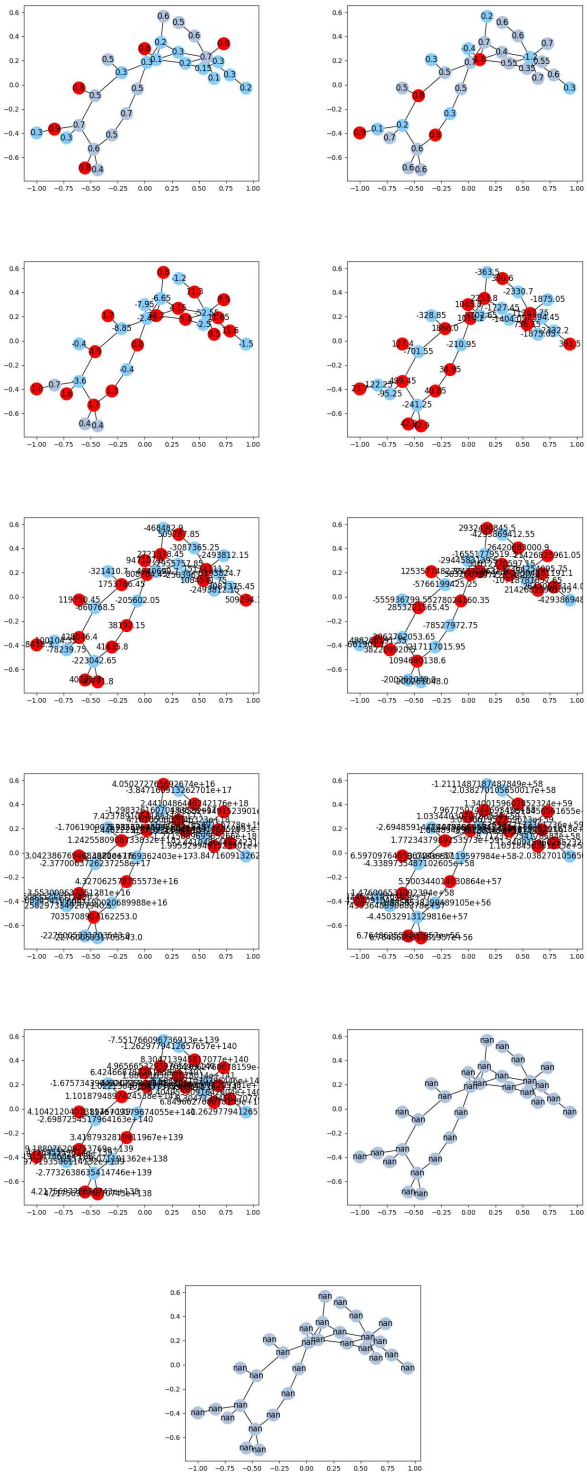

Figure 15: [Without F and G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

E LISC: Varying with the rate

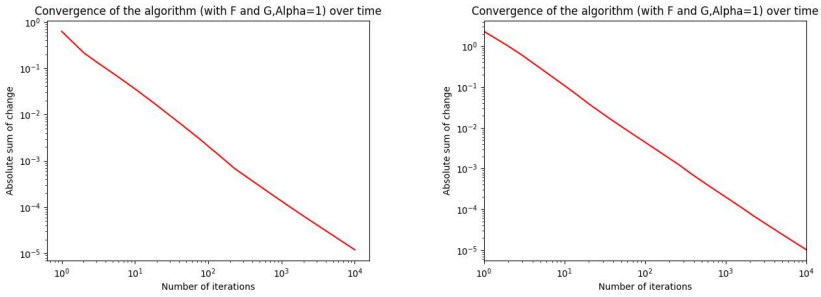

Figure 16: [Rate=$\frac{1}{t}$]: Left: convergence of medium network. Right: convergence of large network

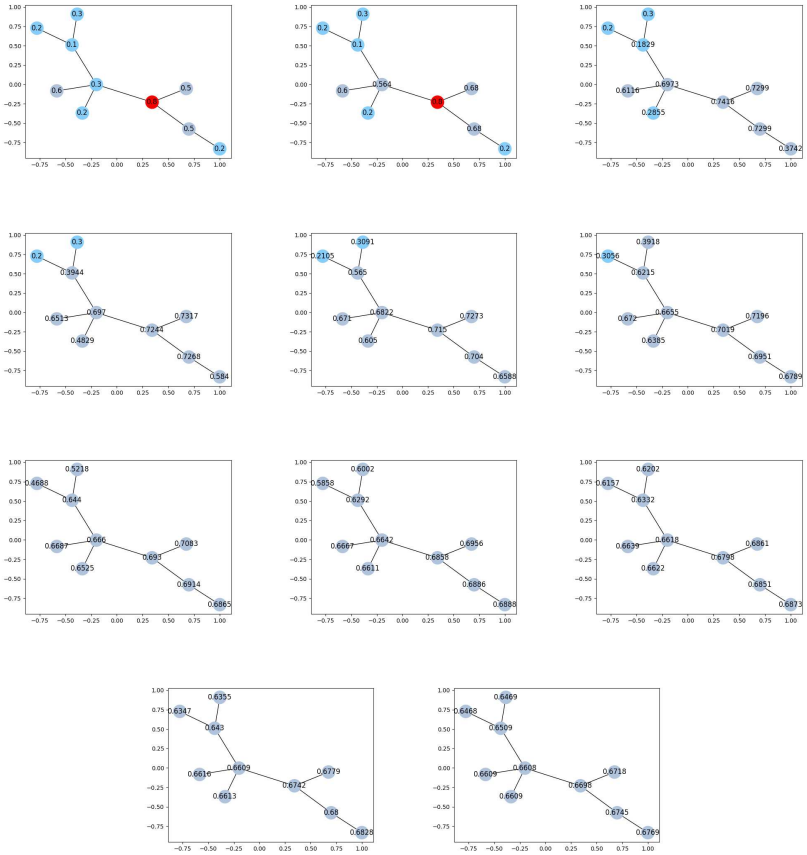

Figure 17: [Rate=$\frac{1}{t}$]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

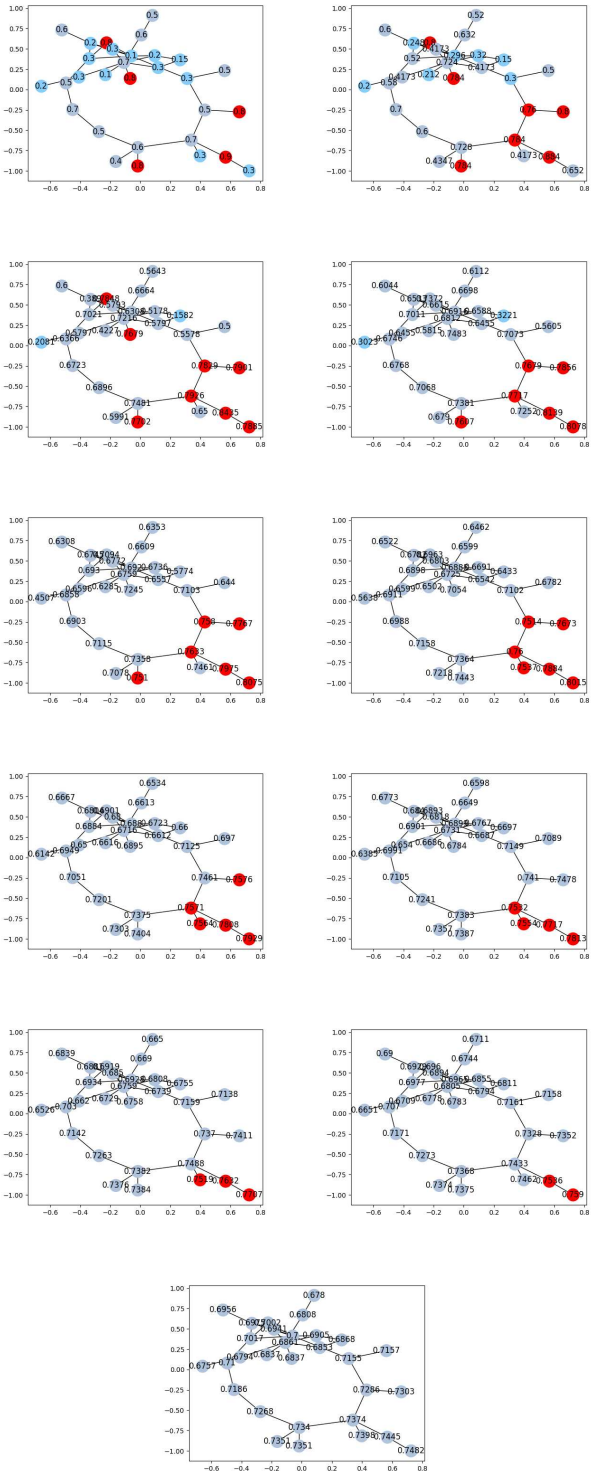

Figure 18: [Rate=$\frac{1}{t}$]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

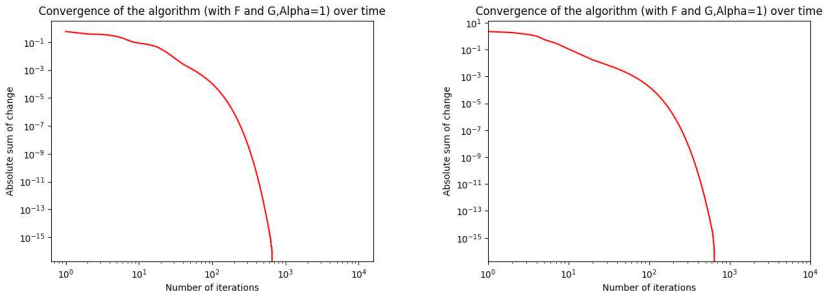

Figure 19: [Rate=$\frac{1}{1.05^t}$]: Left: convergence of medium network. Right: convergence of large network

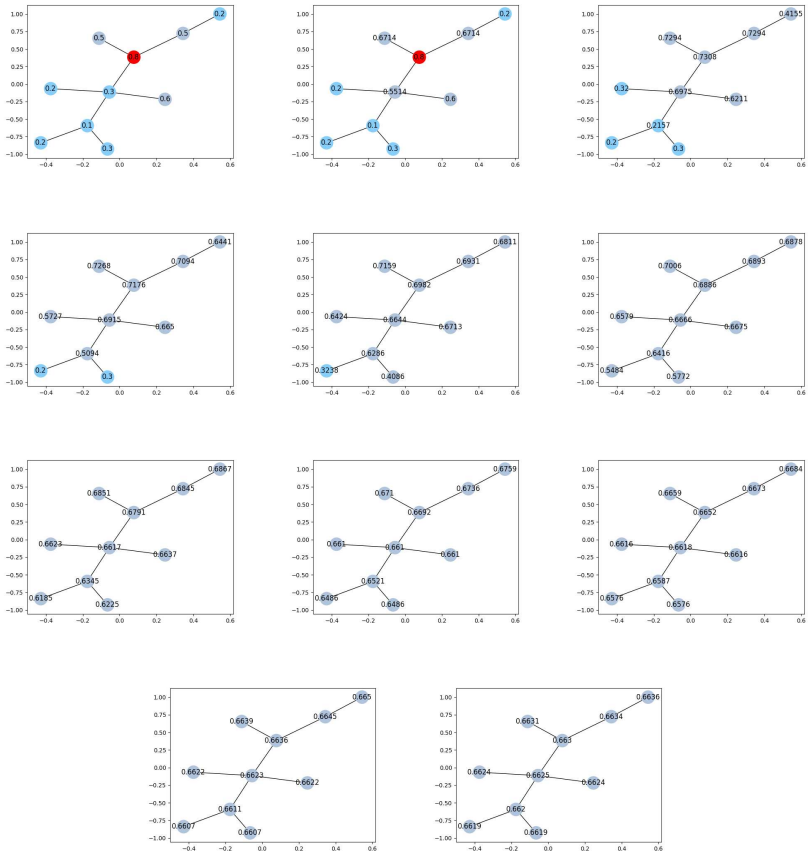

Figure 20: [Rate=$\frac{1}{1.05^t}$]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

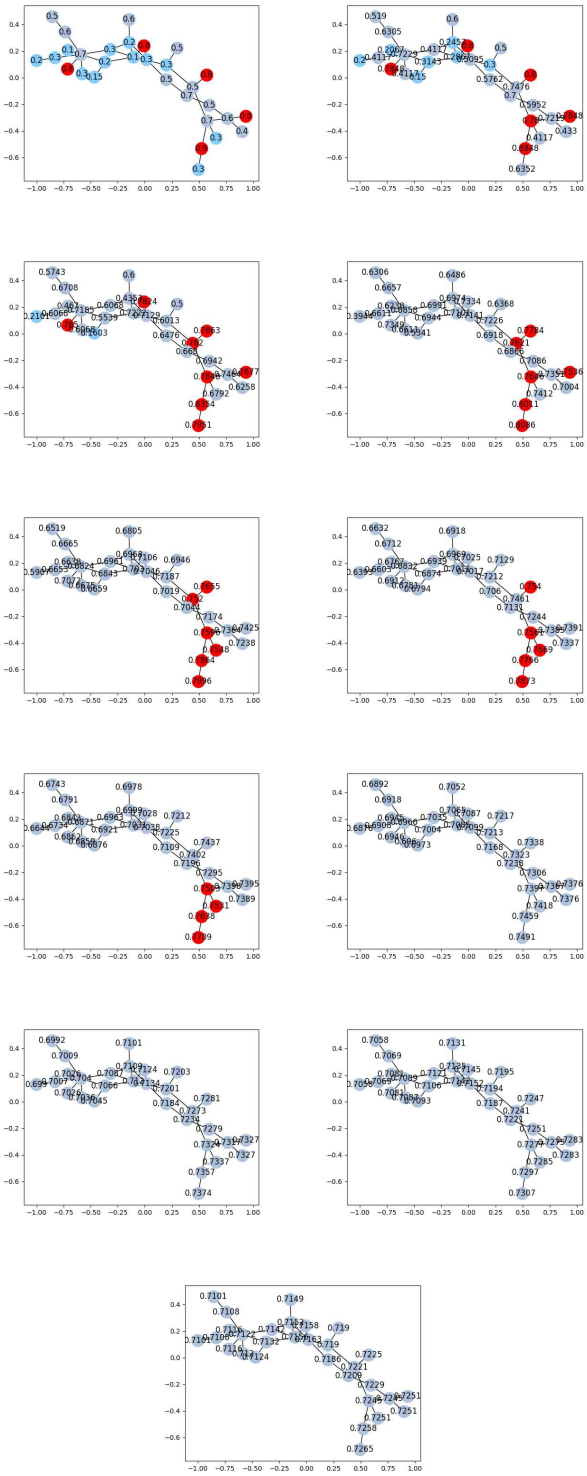

Figure 21: [Rate=$\frac{1}{1.05^t}$]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

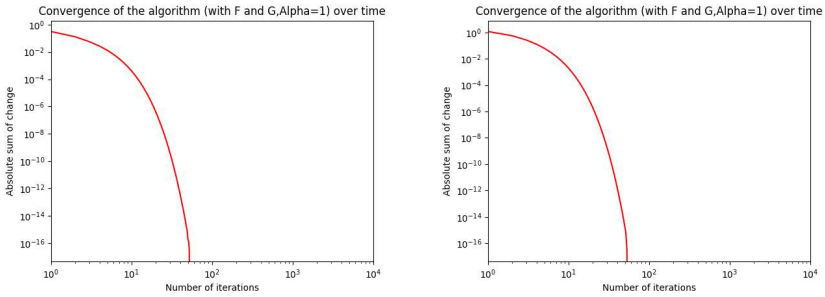

Figure 22: [Rate=$\frac{1}{2^t}$]: Left: convergence of medium network. Right: convergence of large network

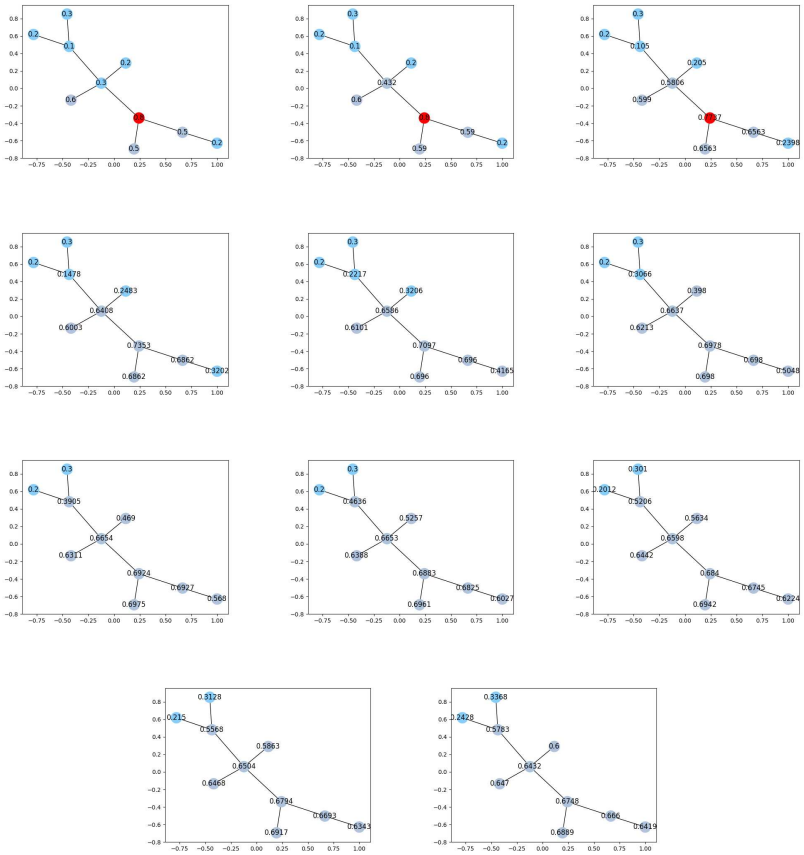

Figure 23: [Rate=$\frac{1}{2^t}$]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

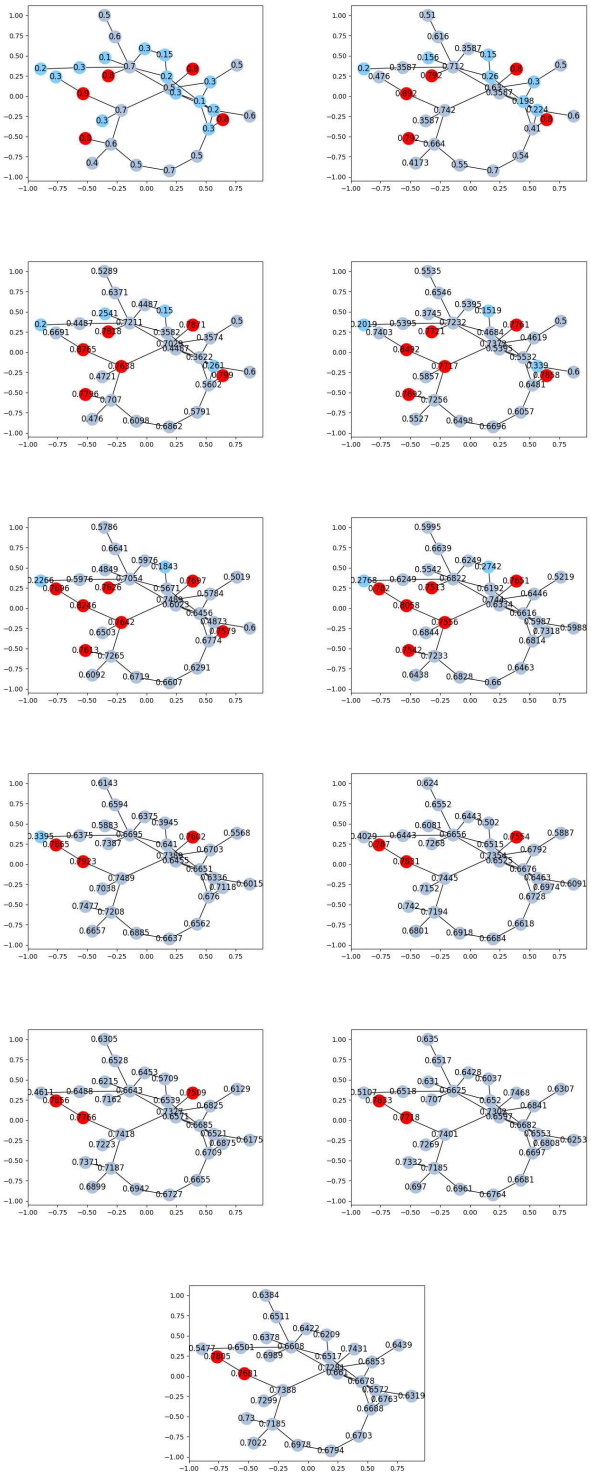

Figure 24: [Rate=$\frac{1}{2^t}$]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

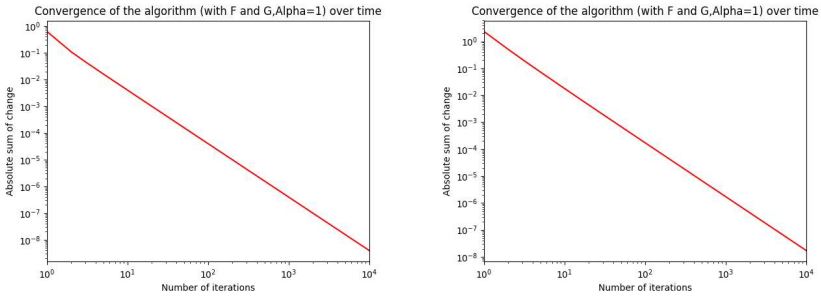

Figure 25: [Rate=$\frac{1}{t^2}$]: Left: convergence of medium network. Right: convergence of large network

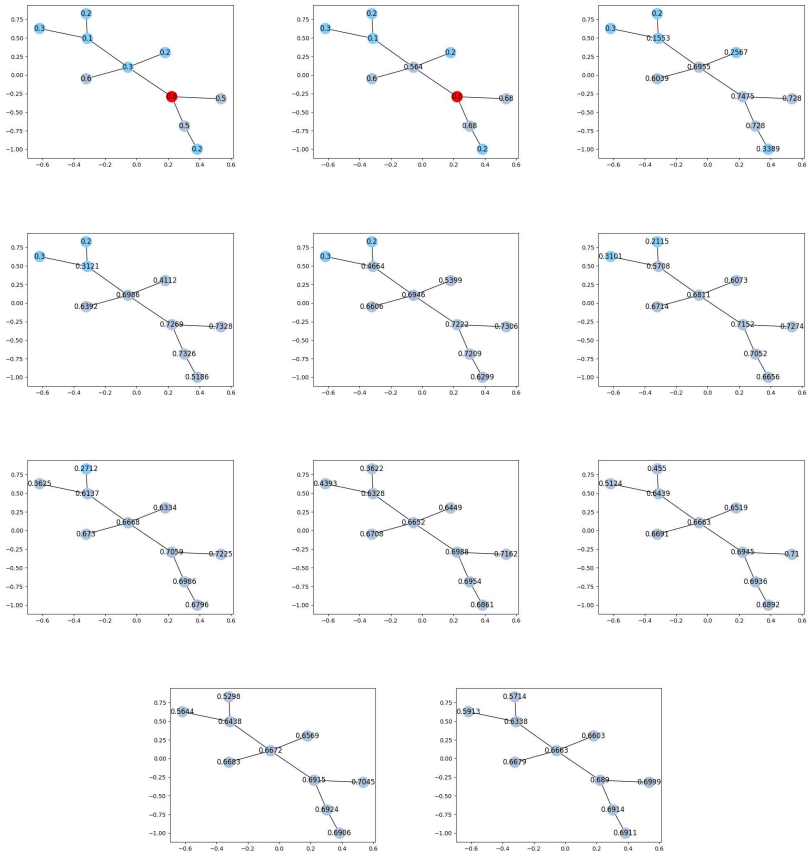

Figure 26: [Rate=$\frac{1}{t^2}$]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

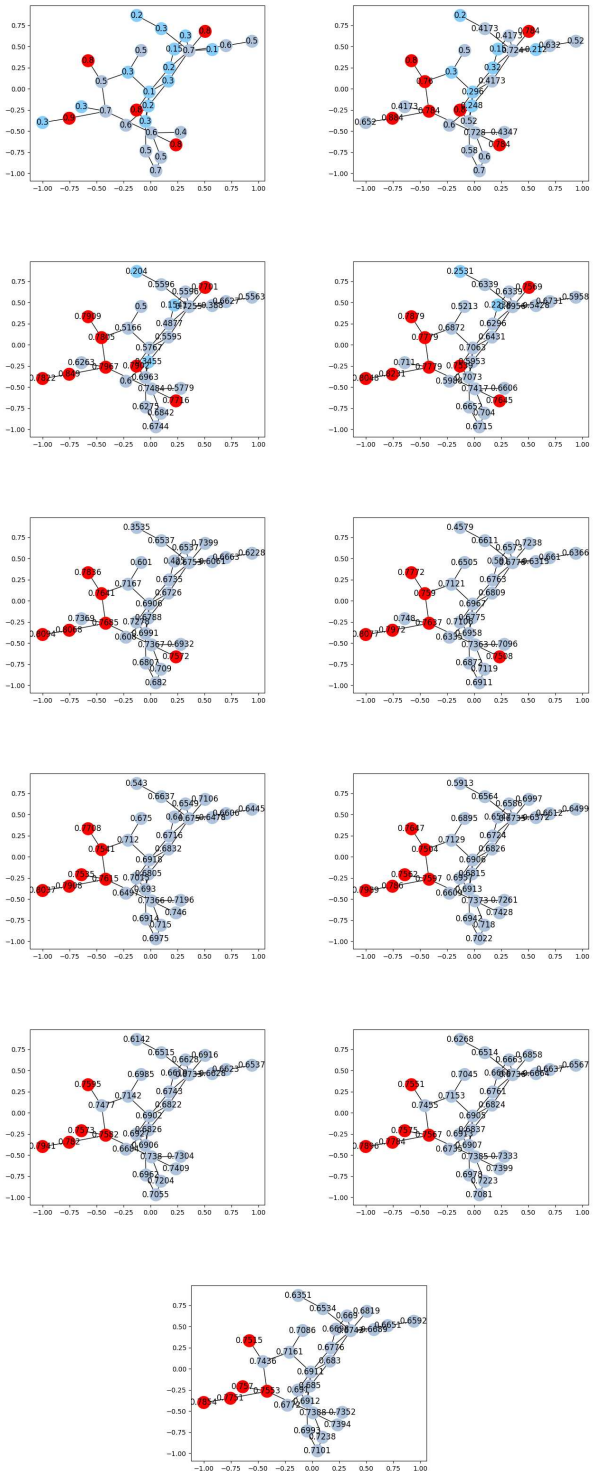

Figure 27: [Rate=$\frac{1}{t^2}$]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

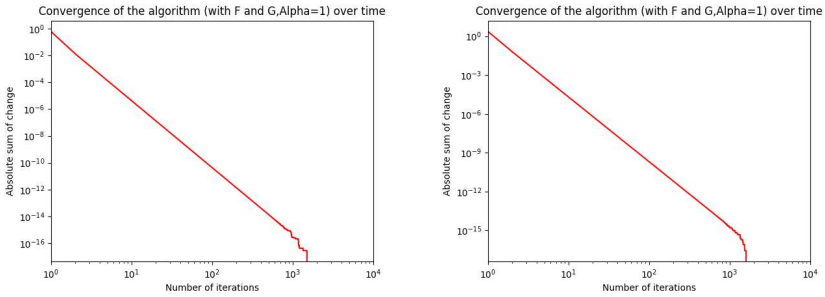

Figure 28: [Rate=$\frac{1}{t^5}$]: Left: convergence of medium network. Right: convergence of large network

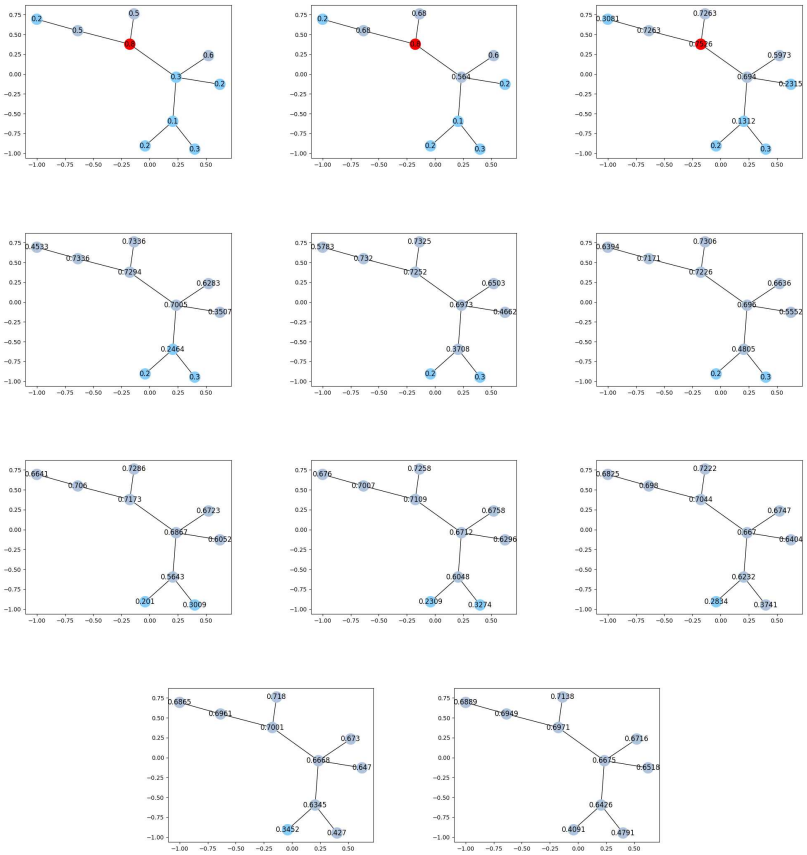

Figure 29: [Rate=$\frac{1}{t^5}$]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

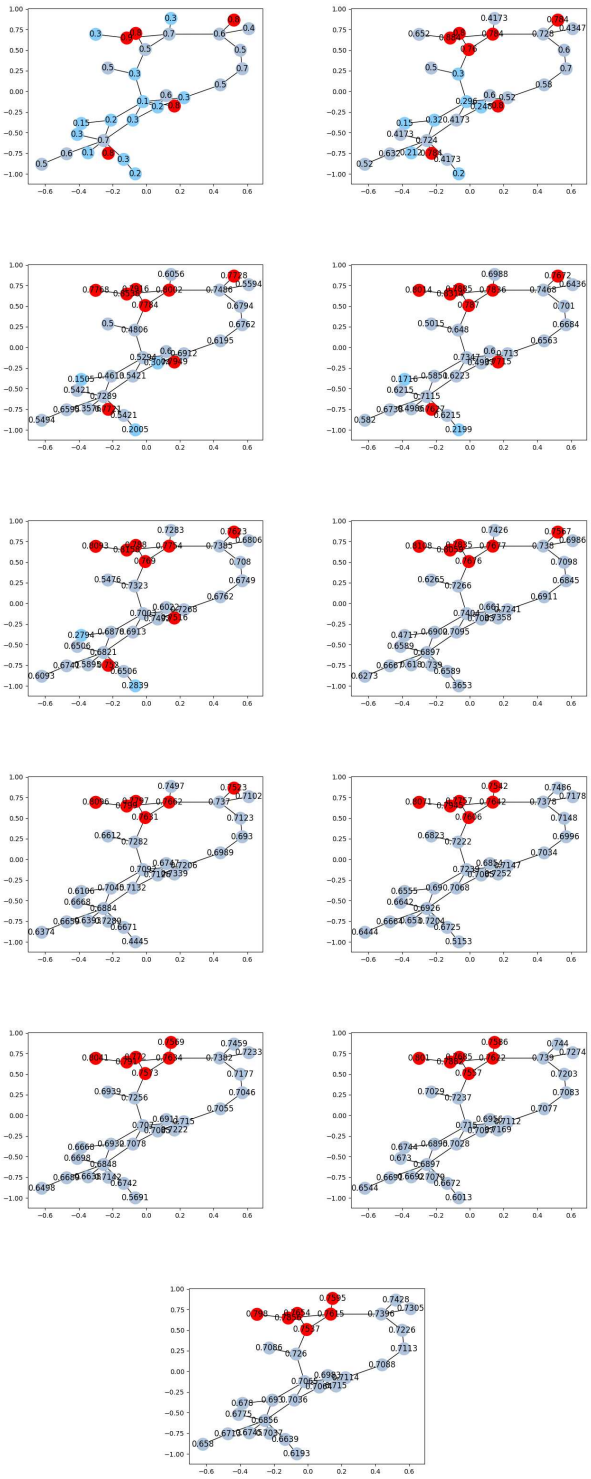

Figure 30: [Rate=$\frac{1}{t^5}$]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

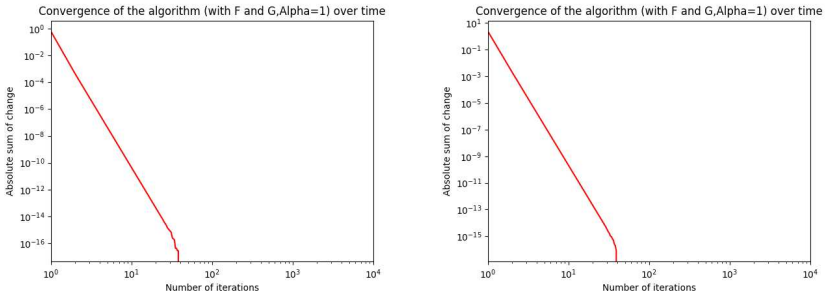

Figure 31: [Rate=$\frac{1}{t^{10}}$]: Left: convergence of medium network. Right: convergence of large network

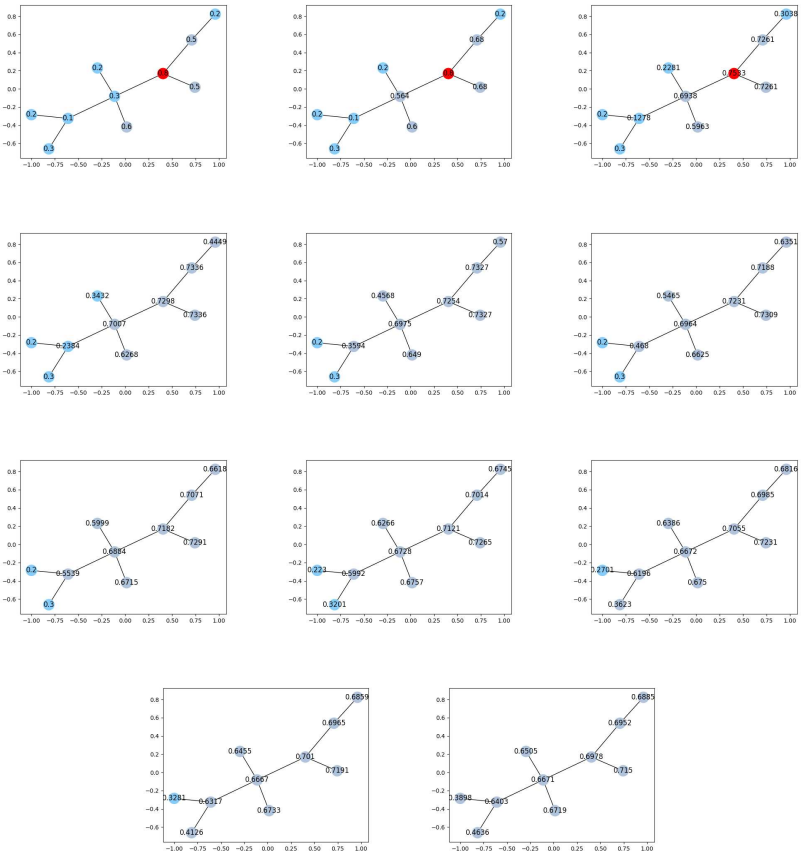

Figure 32: [Rate=$\frac{1}{t^{10}}$]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

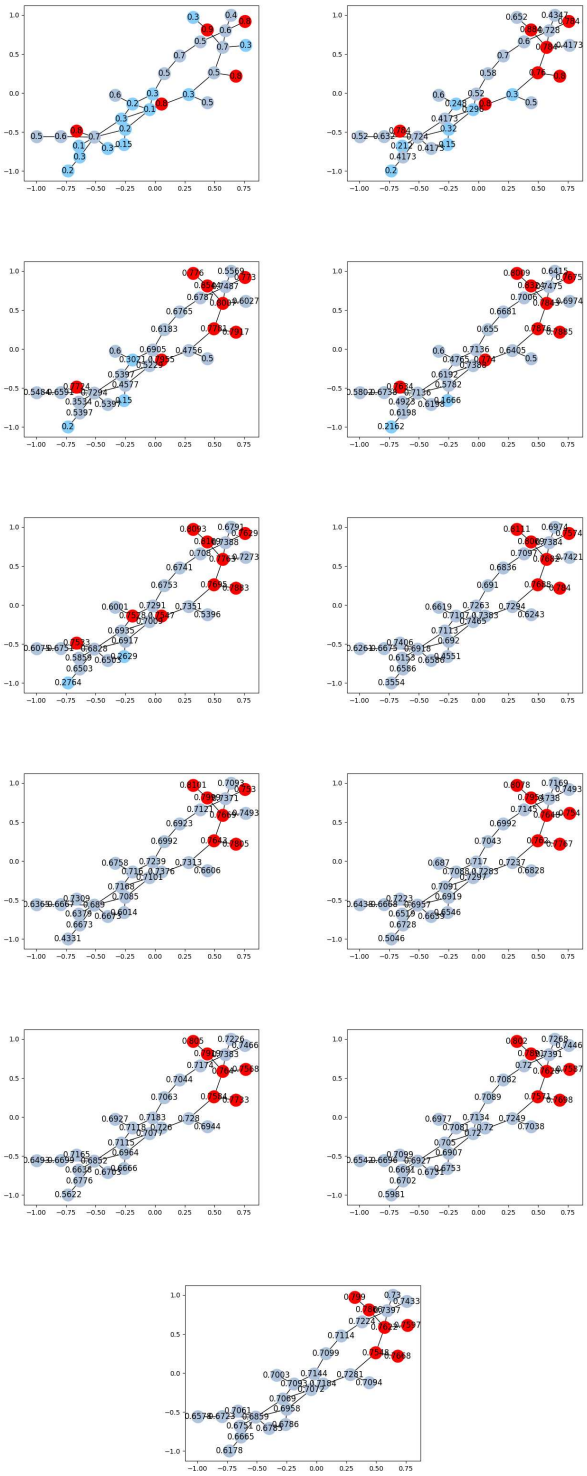

Figure 33: [Rate=$\frac{1}{t^{10}}$]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

F LISC: Weighted mean, rate=1

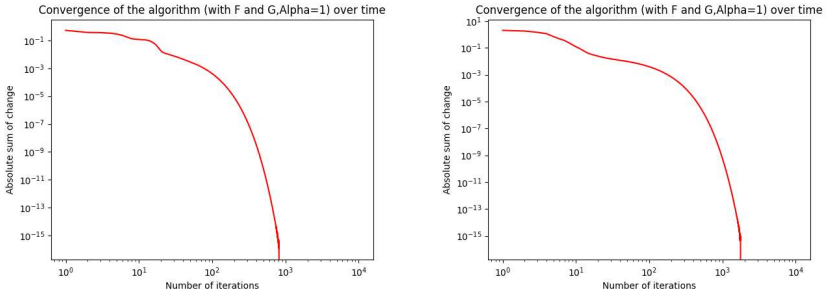

Figure 34: [Rate=1]: Left: convergence of medium network. Right: convergence of large network

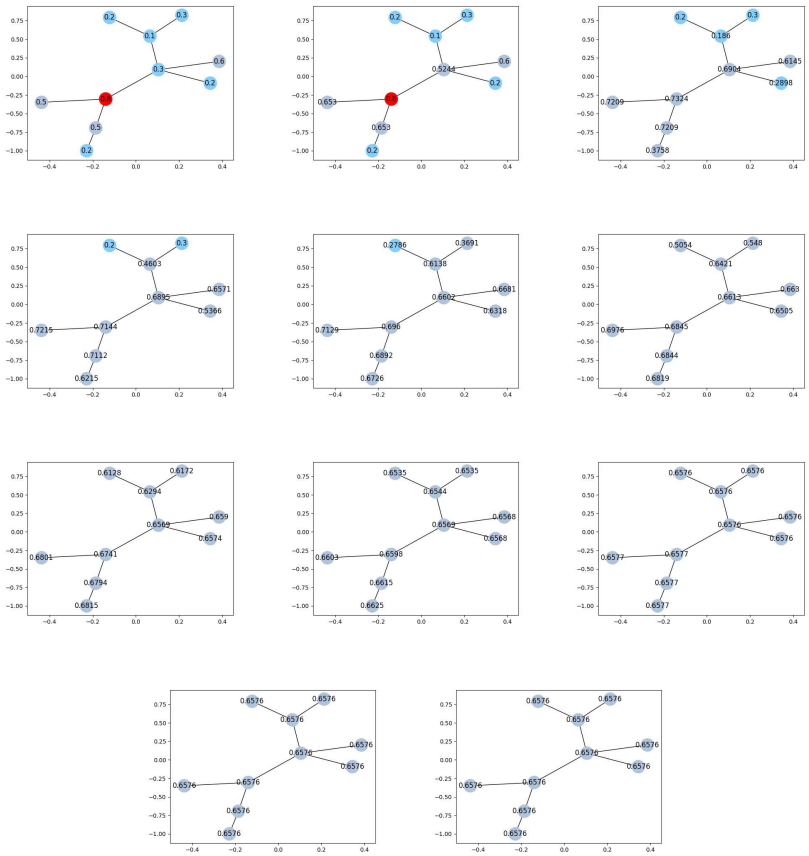

Figure 35: [Rate=1]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

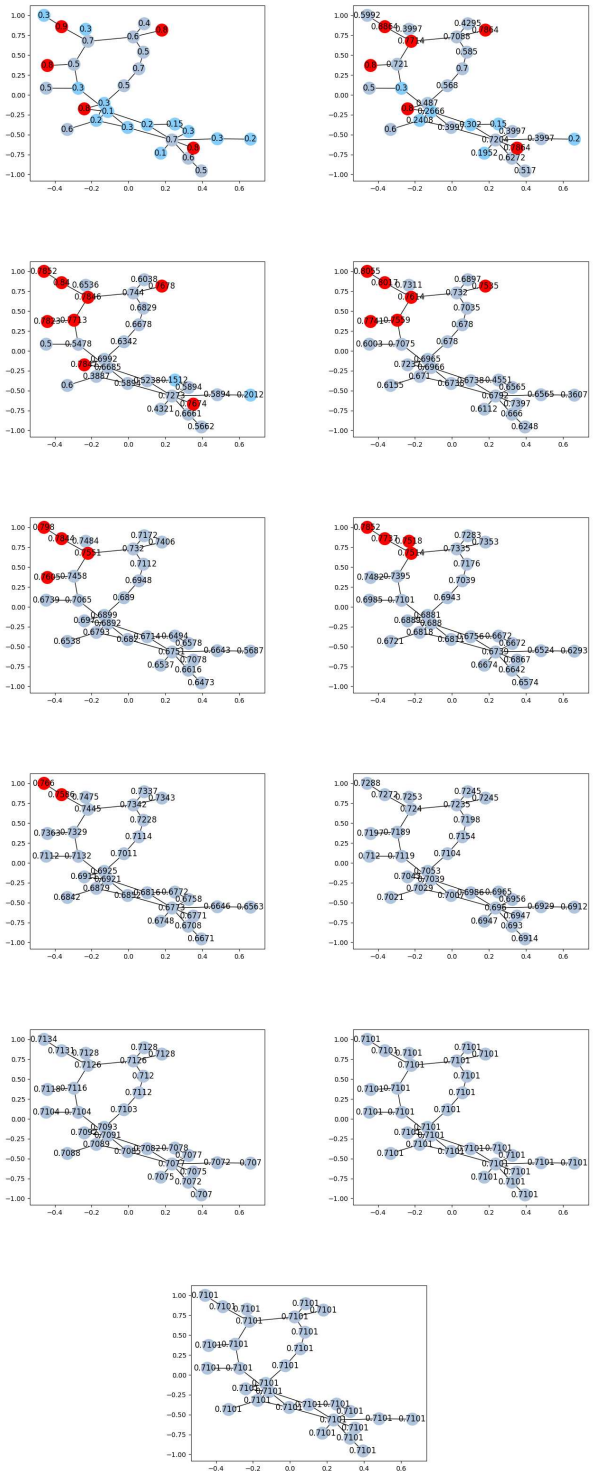

Figure 36: [Rate=1]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

G NORMLISC: Varying with F and G

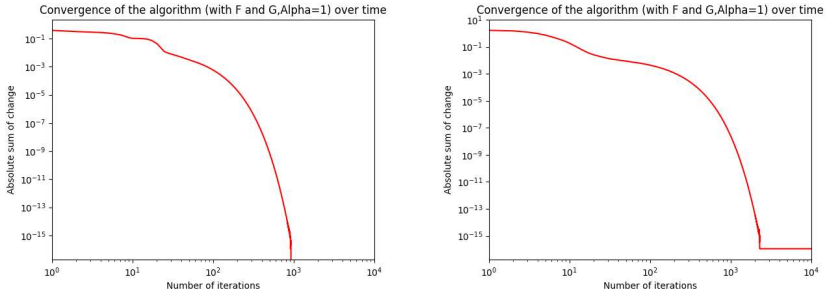

Figure 37: [F and G]: Left: convergence of medium network. Right: convergence of large network

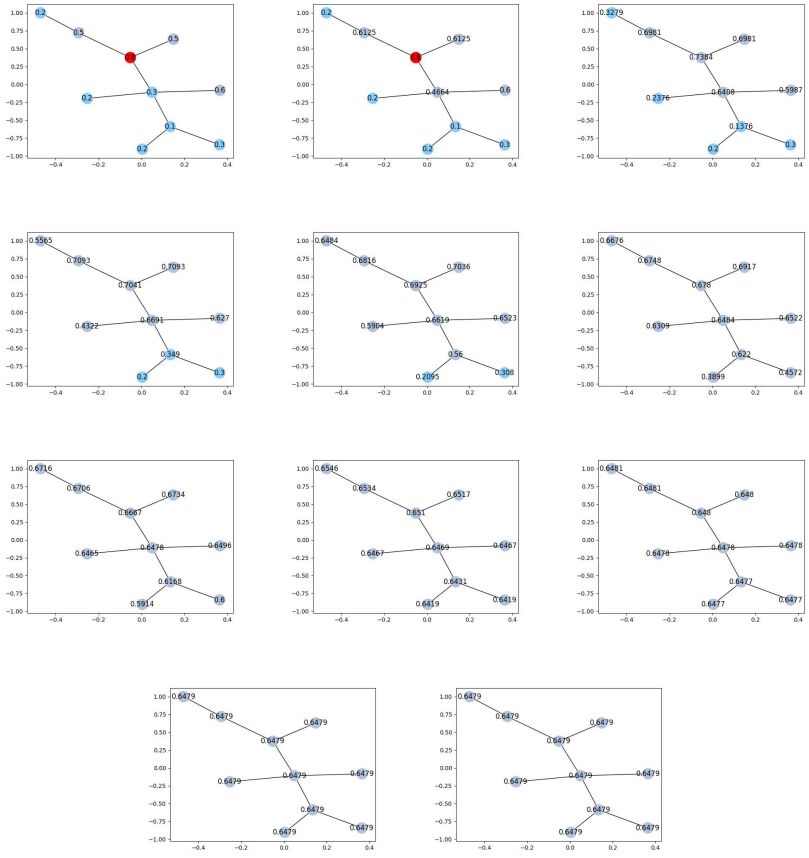

Figure 38: [F and G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

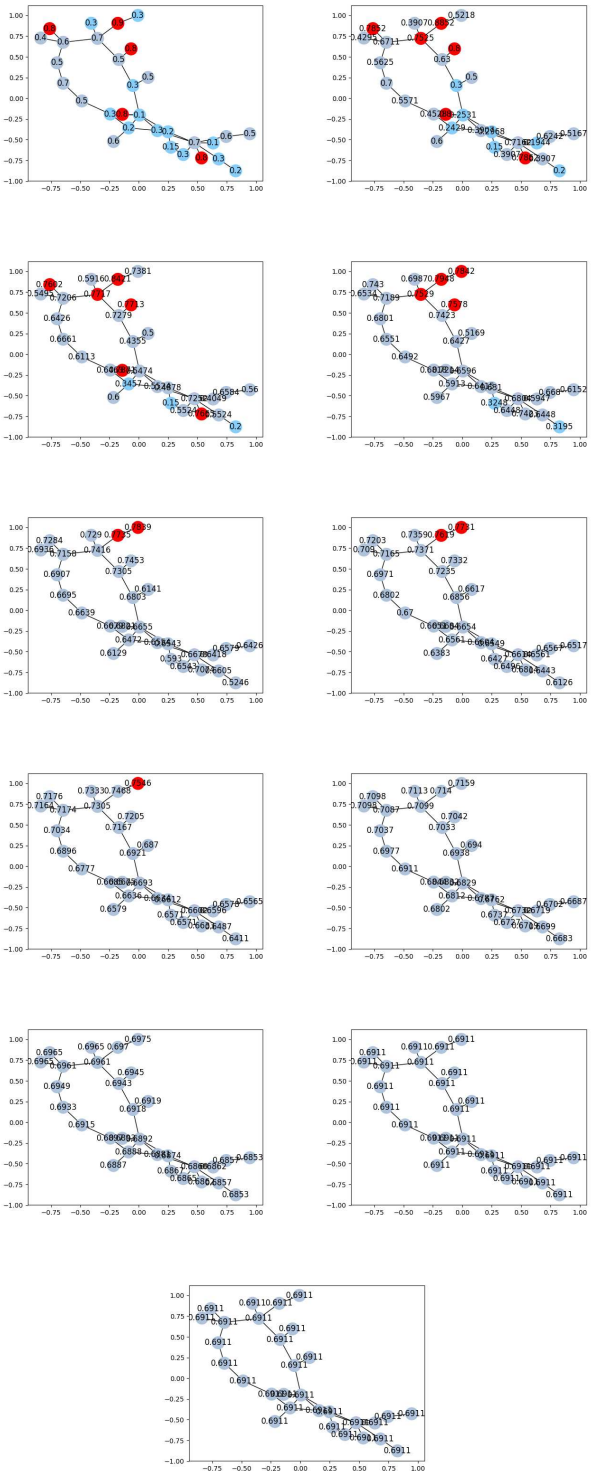

Figure 39: [F and G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

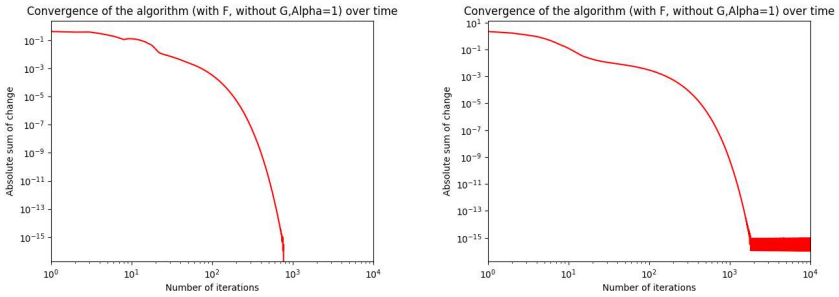

Figure 40: [With F, without G]: Left: convergence of medium network. Right: convergence of large network

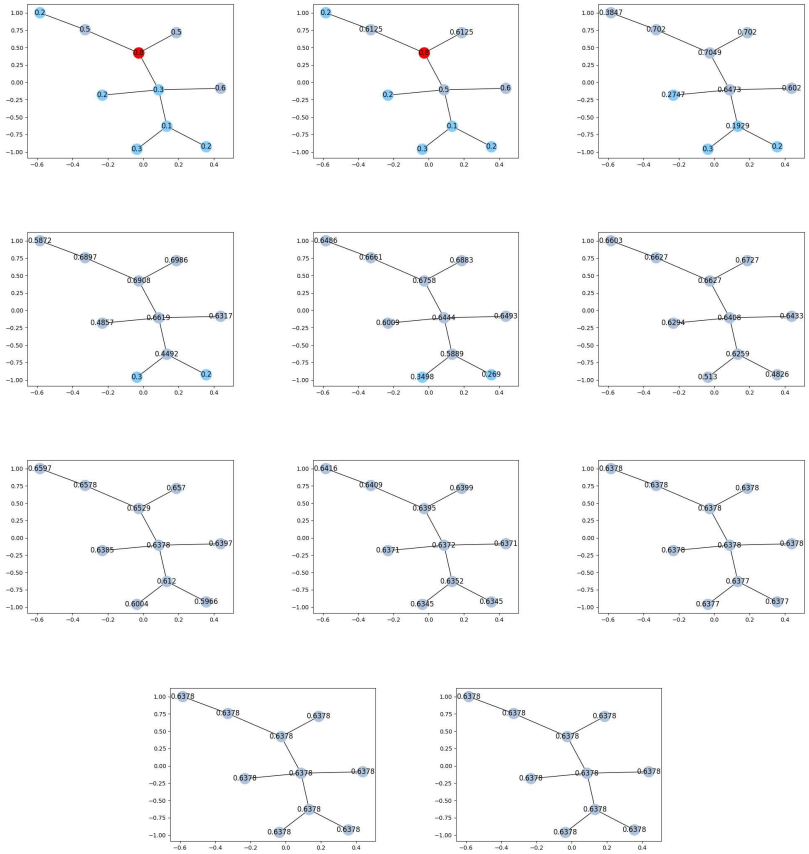

Figure 41: [With F, without G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

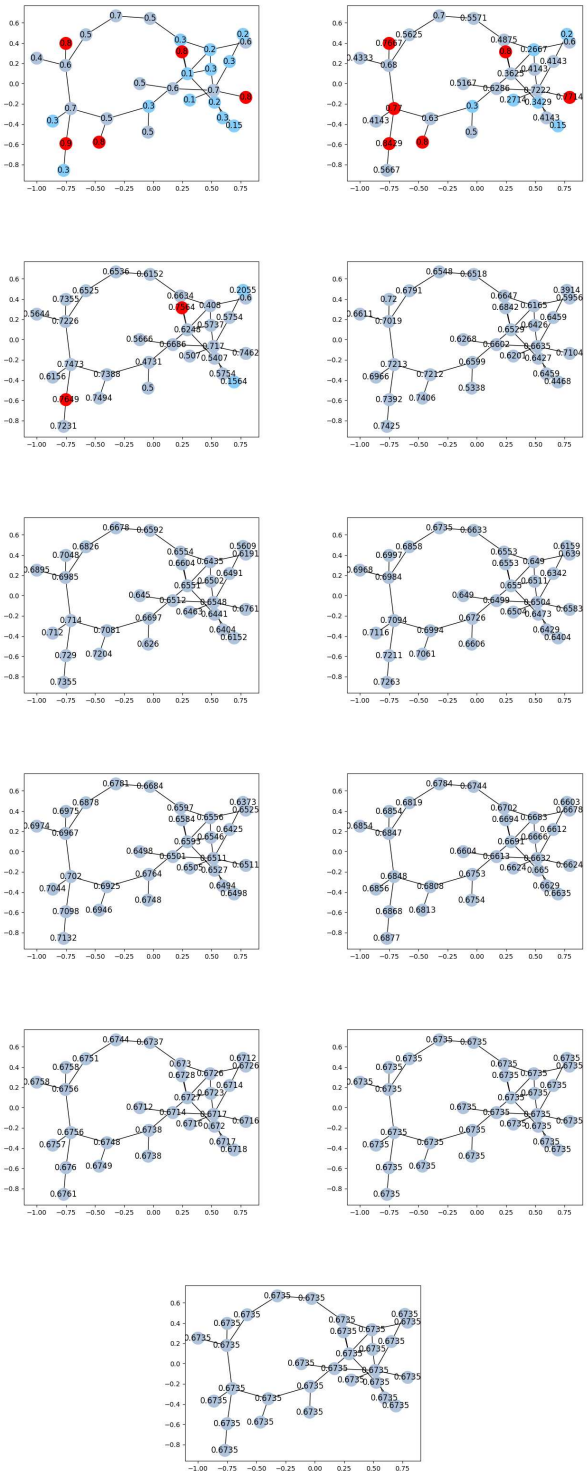

Figure 42: [With F, without G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

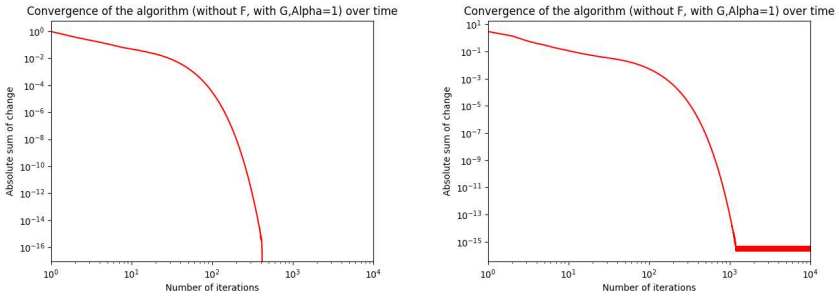

Figure 43: [Without F, with G]: Left: convergence of medium network. Right: convergence of large network

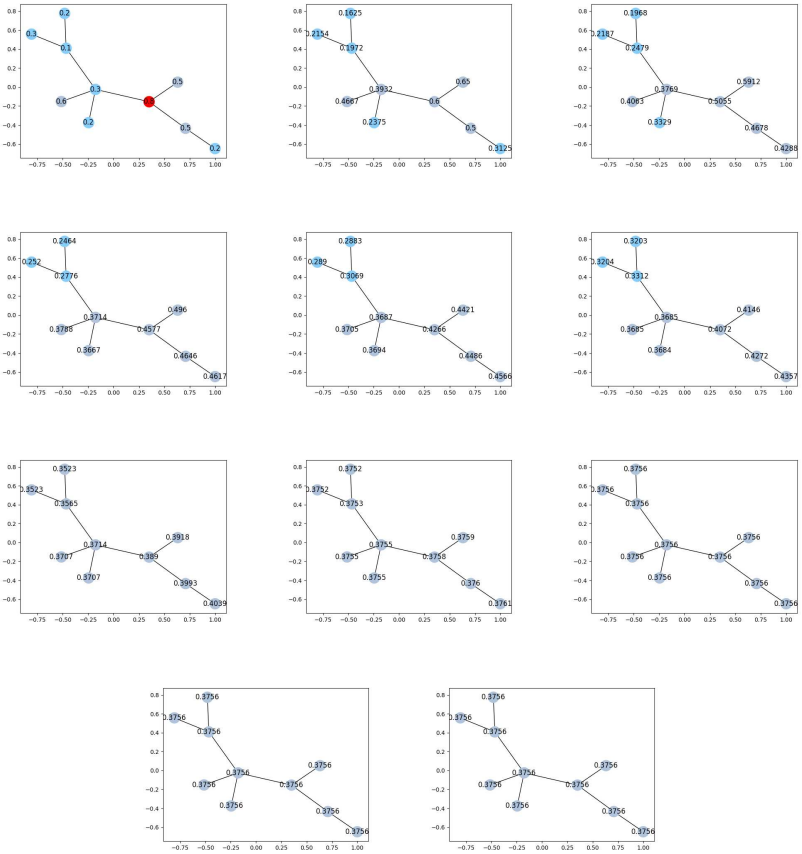

Figure 44: [Without F, with G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

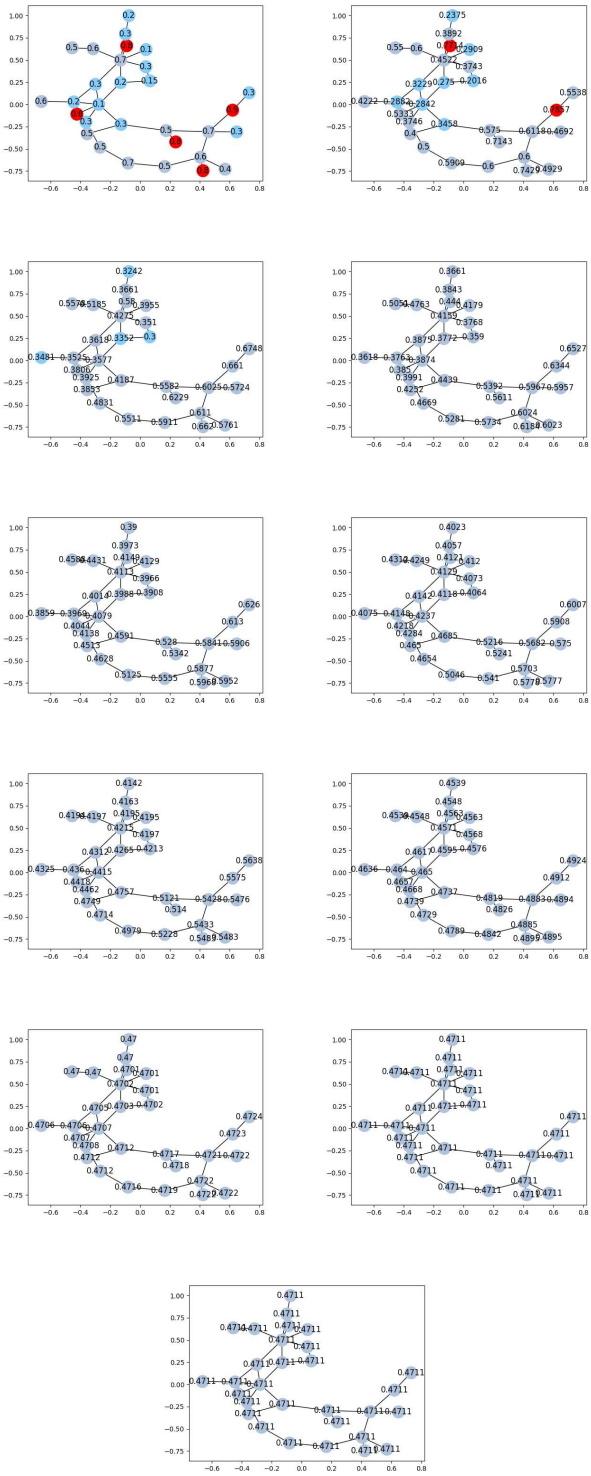

Figure 45: [Without F, with G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

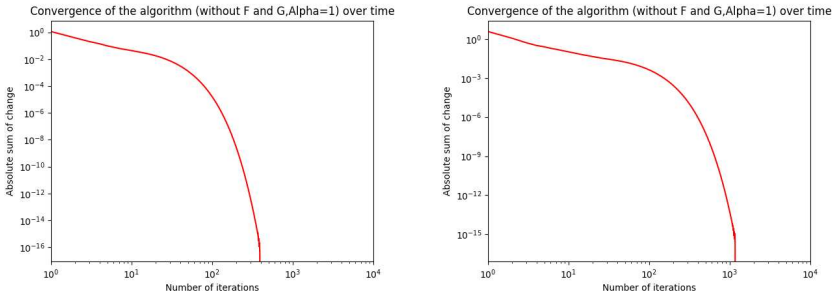

Figure 46: [Without F and G]: Left: convergence of medium network. Right: convergence of large network

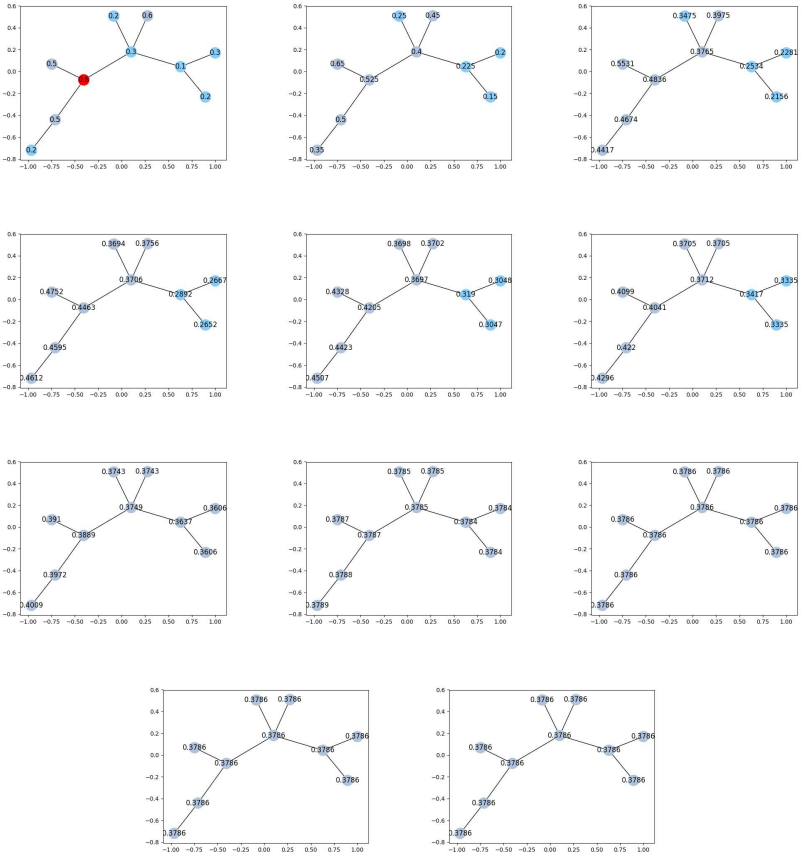

Figure 47: [Without F and G]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

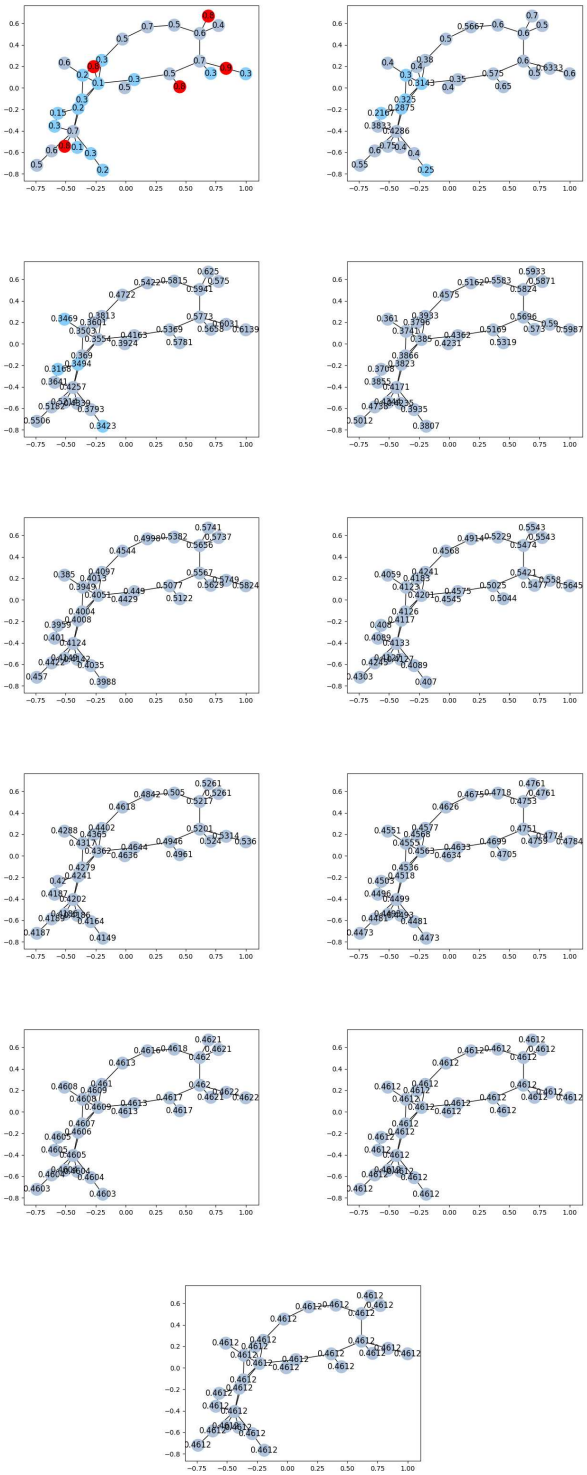

Figure 48: [Without F and G]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

H NORMLISC: Varying with the rate

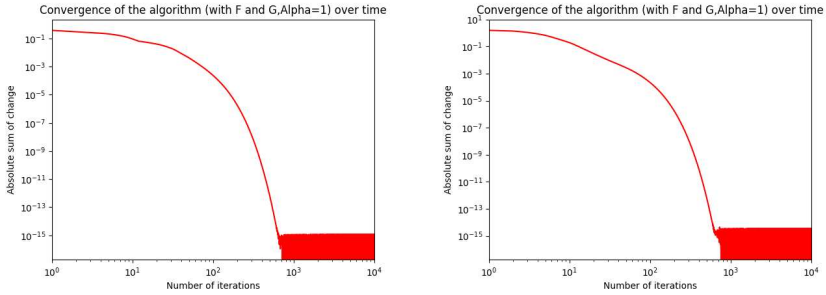

Figure 49: [Rate=1.05^t]: Left: convergence of medium network. Right: convergence of large network

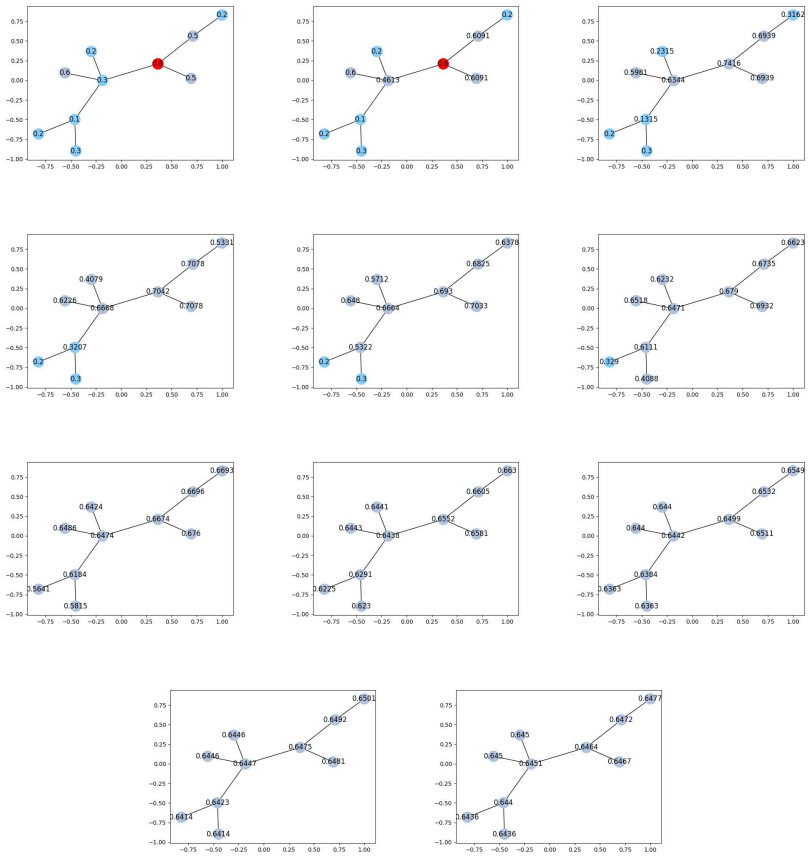

Figure 50: [Rate=1.05^t]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

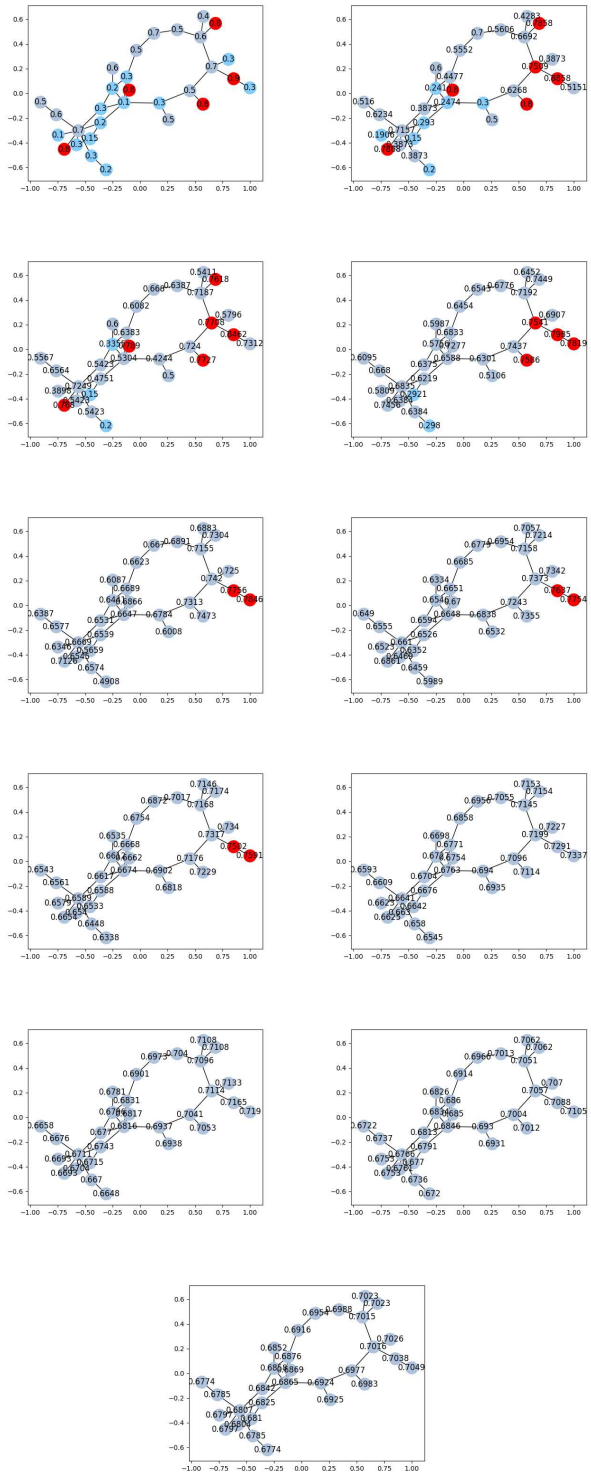

Figure 51: [Rate=1.05^t]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

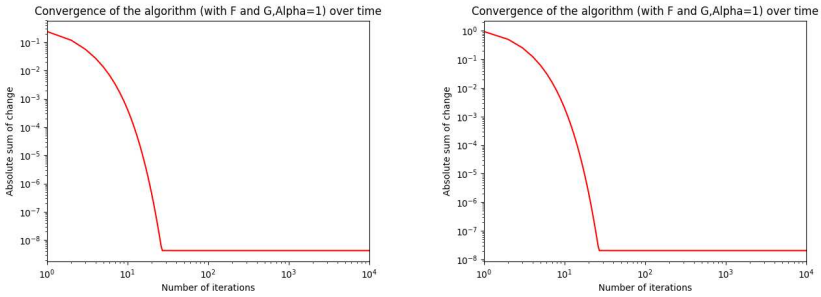

Figure 52: [Rate=2^t]: Left: convergence of medium network. Right: convergence of large network

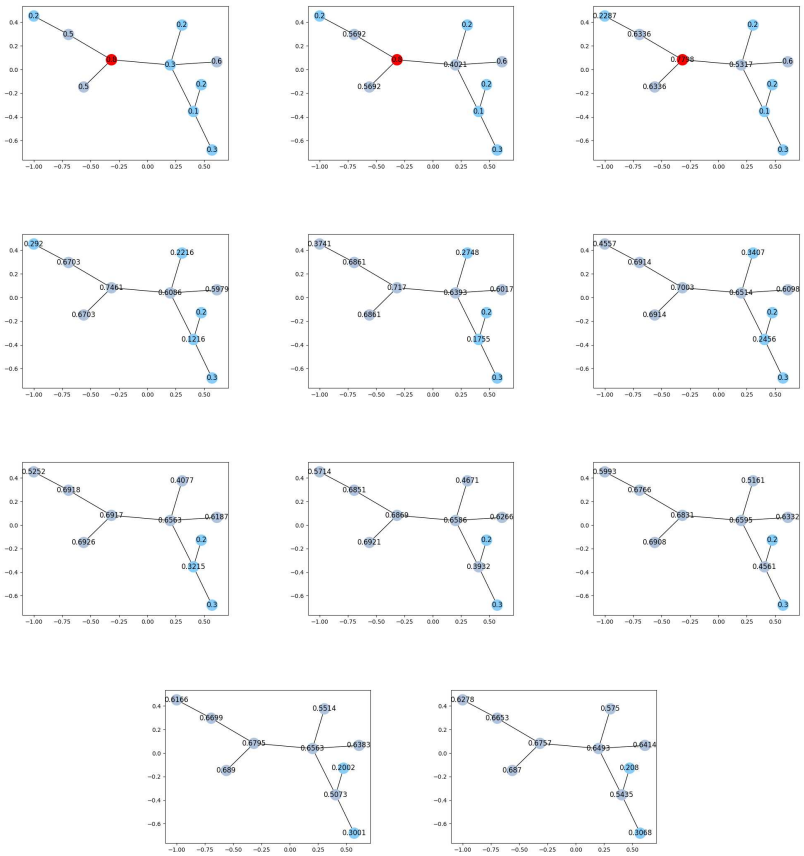

Figure 53: [Rate=2^t]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

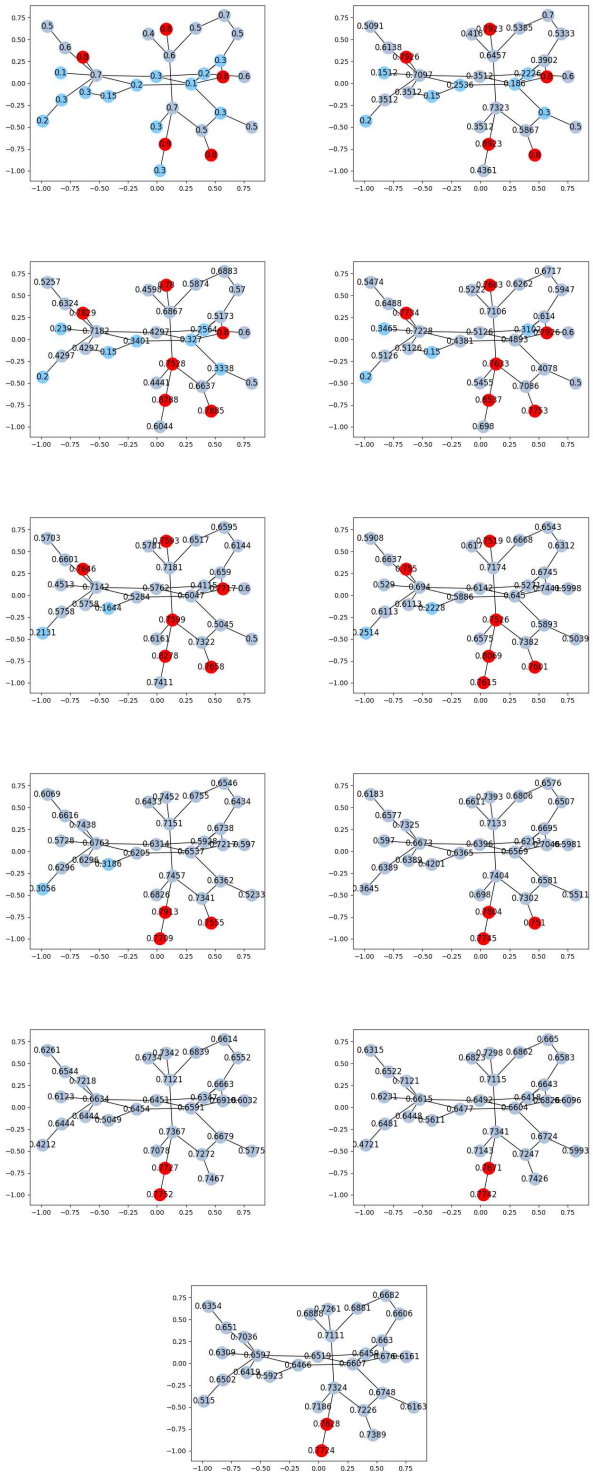

Figure 54: [Rate=2^t]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

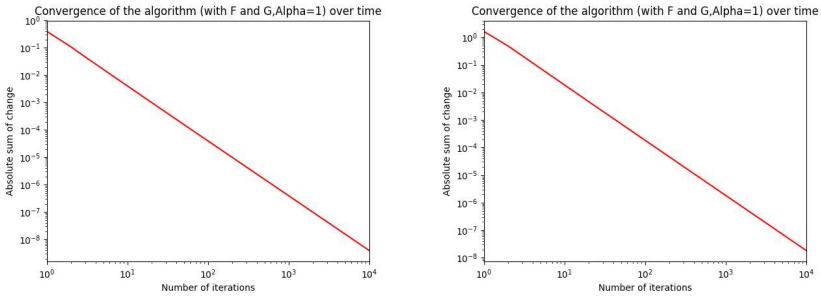

Figure 55: [Rate=t^2]: Left: convergence of medium network. Right: convergence of large network

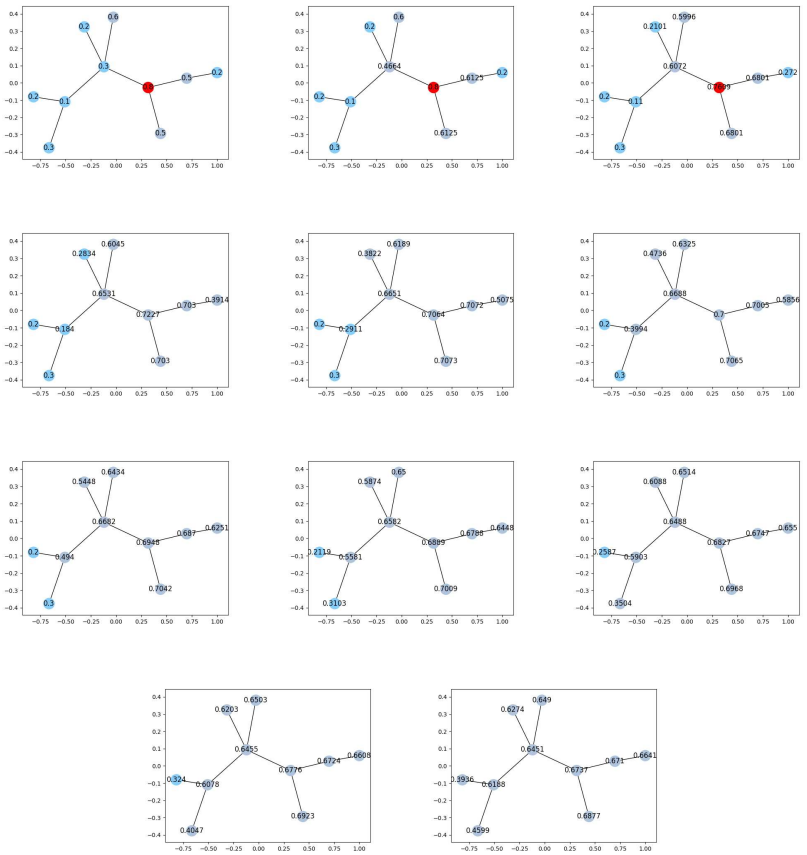

Figure 56: [Rate=t^2]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

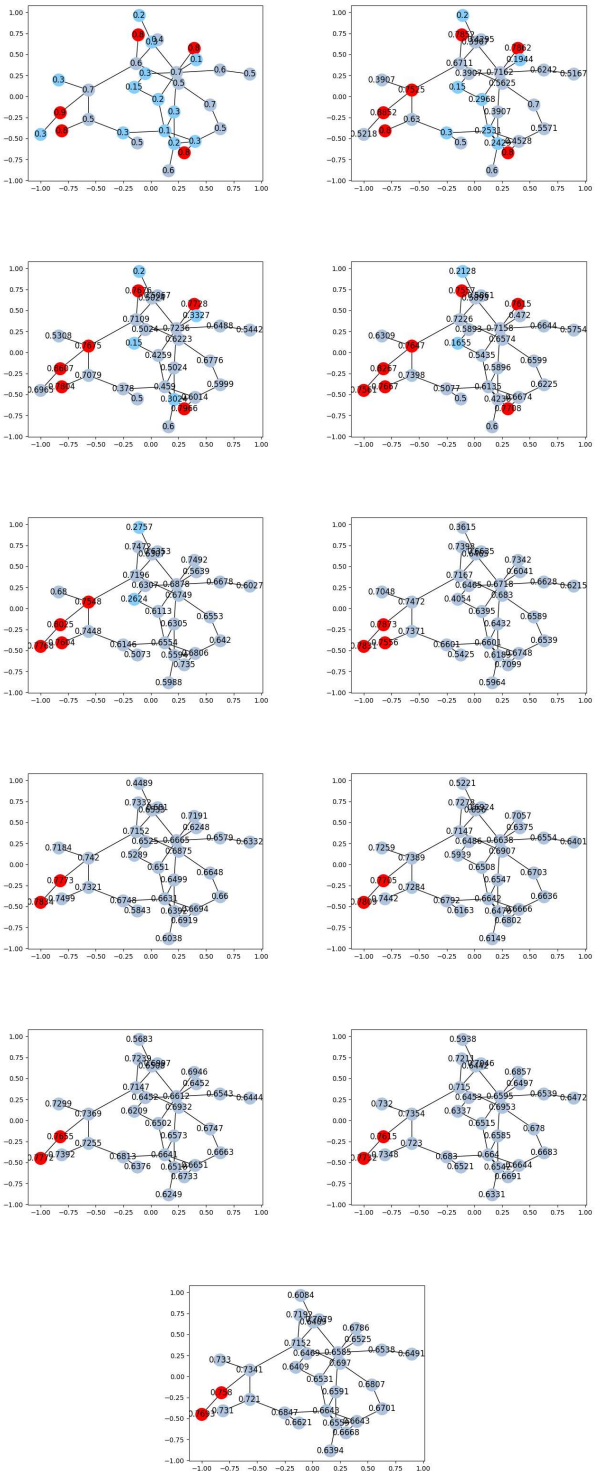

Figure 57: [Rate=t^2]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

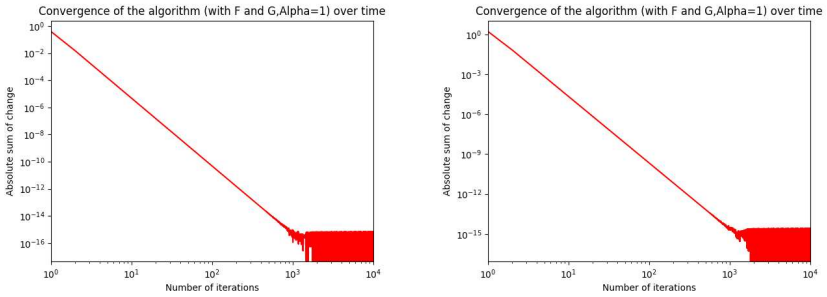

Figure 58: [Rate=t^5]: Left: convergence of medium network. Right: convergence of large network

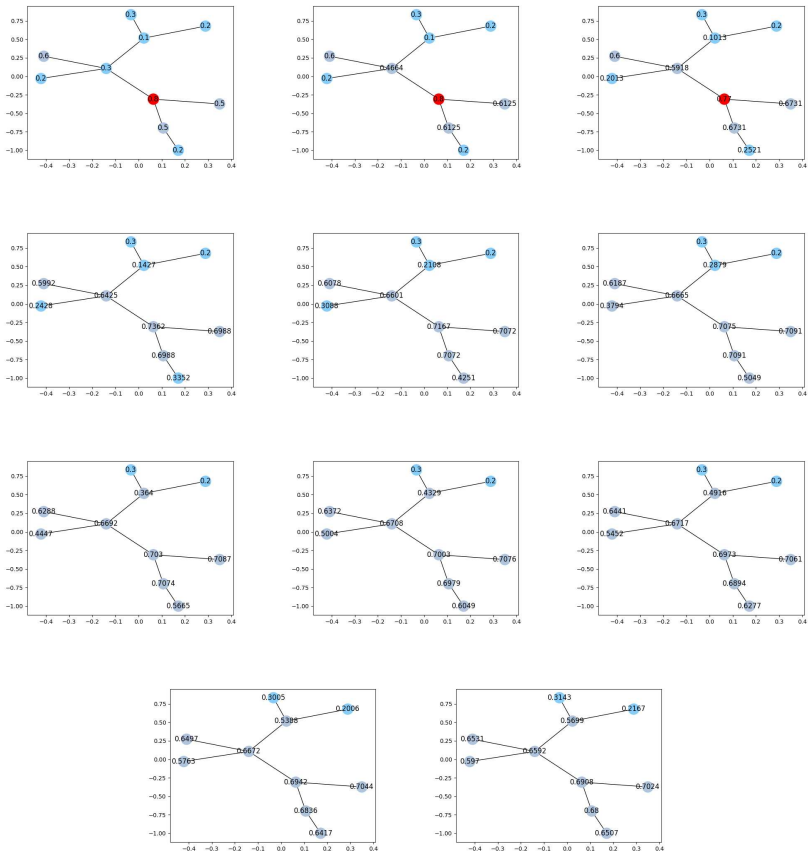

Figure 59: [Rate=t^5]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

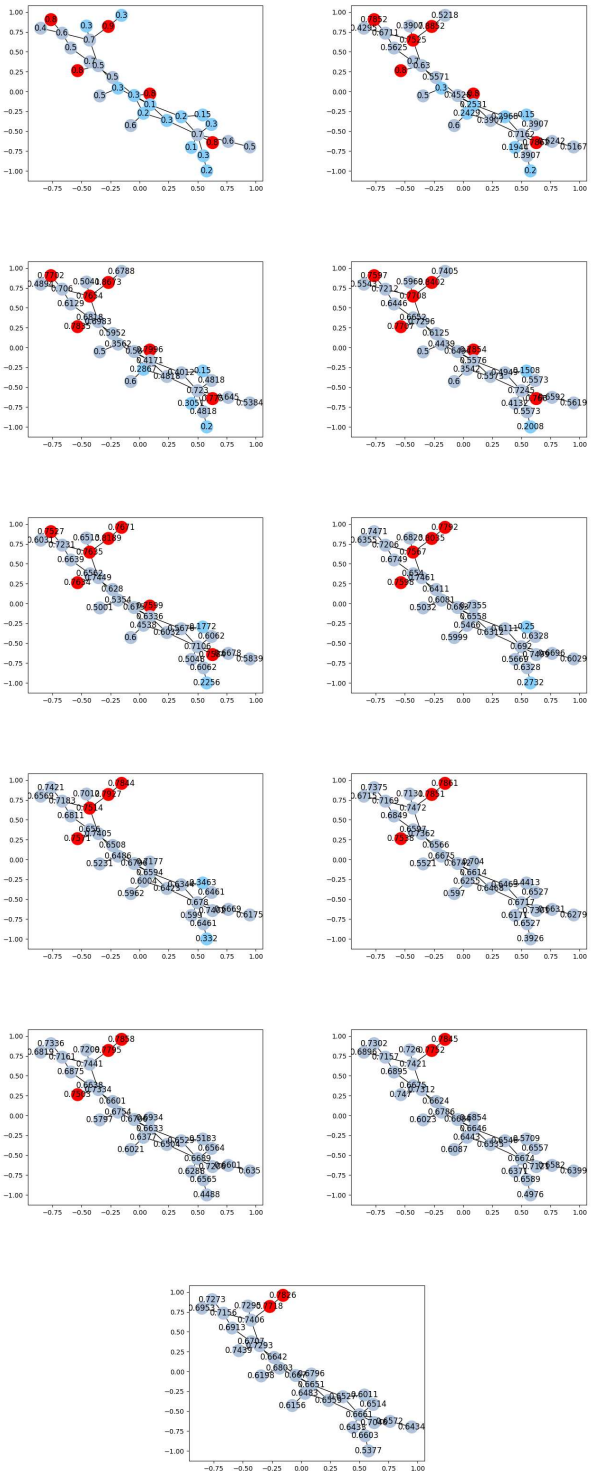

Figure 60: [Rate=t^5]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

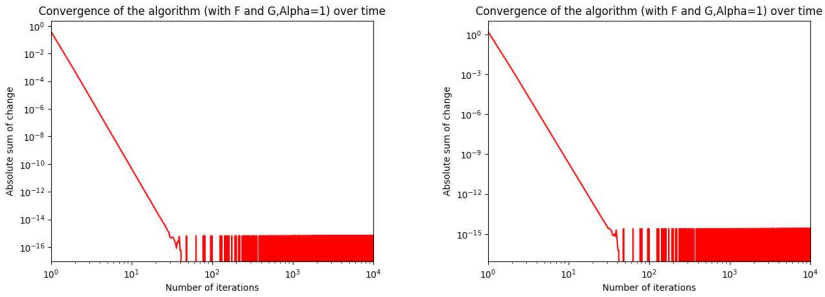

Figure 61: [Rate=t^{10}]: Left: convergence of medium network. Right: convergence of large network

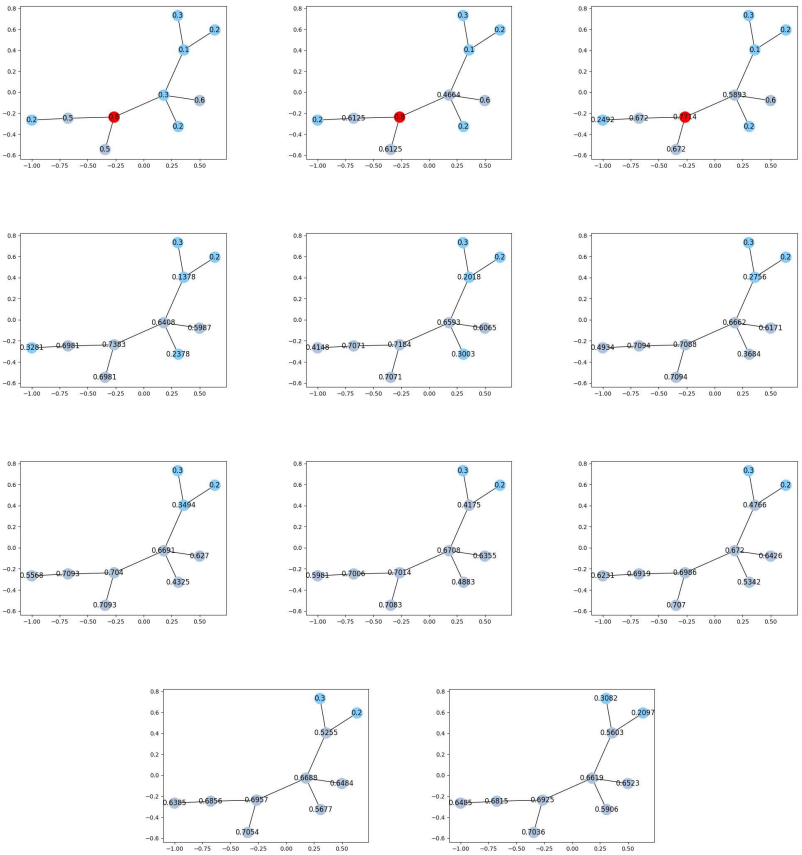

Figure 62: [Rate=t^{10}]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

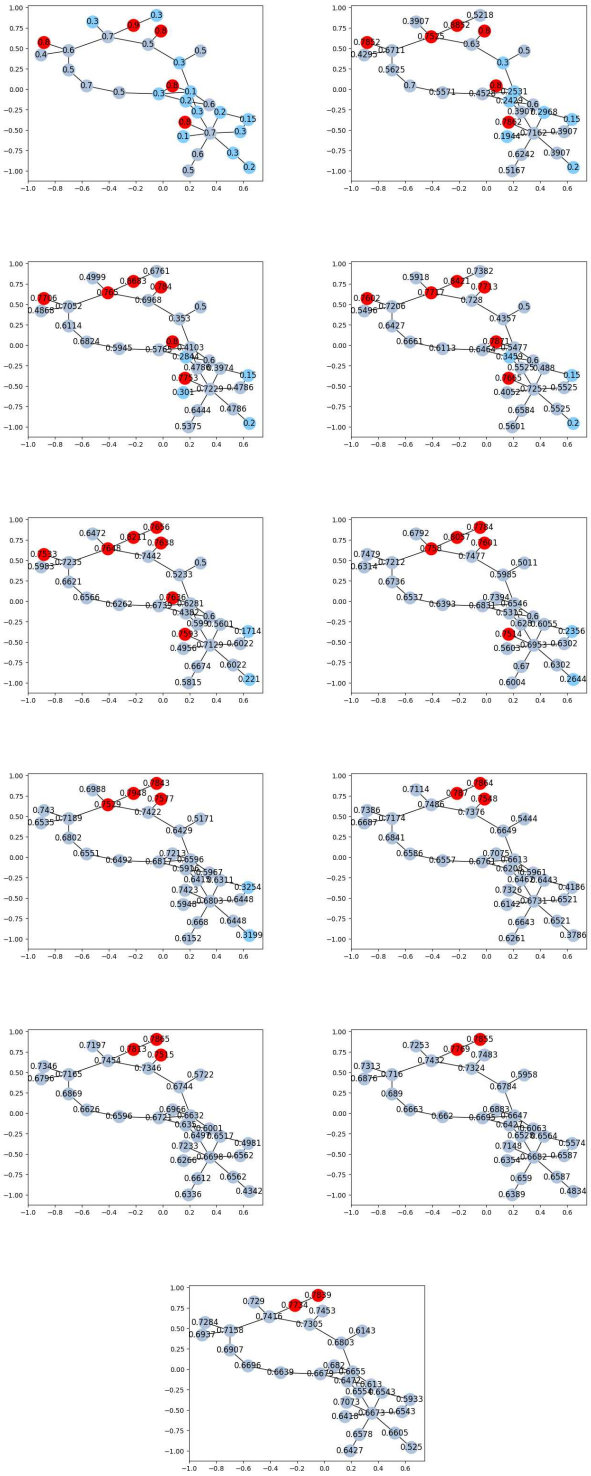

Figure 63: [Rate=t^{10}]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

I NORMLISC: Weighted mean, rate=1

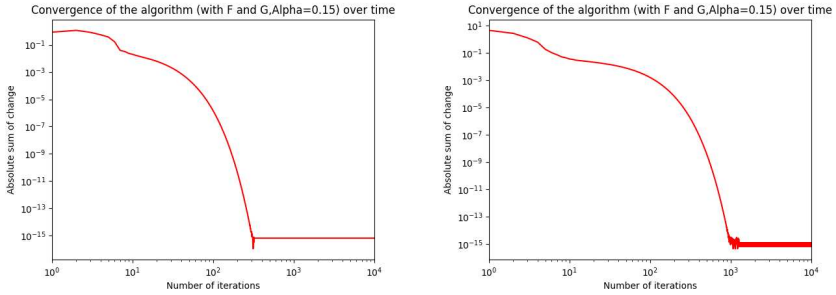

Figure 64: [Rate=1]: Left: convergence of medium network. Right: convergence of large network

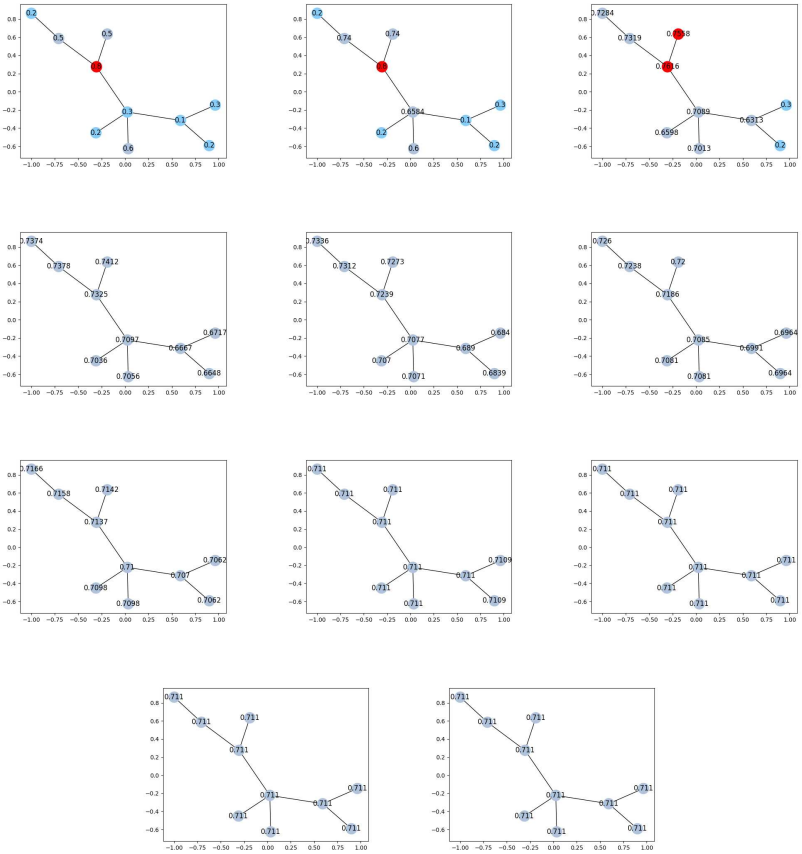

Figure 65: [Rate=1]: Convergence of the medium network over [1,2,3,4,5,10,50,100,1000,10000] iterations

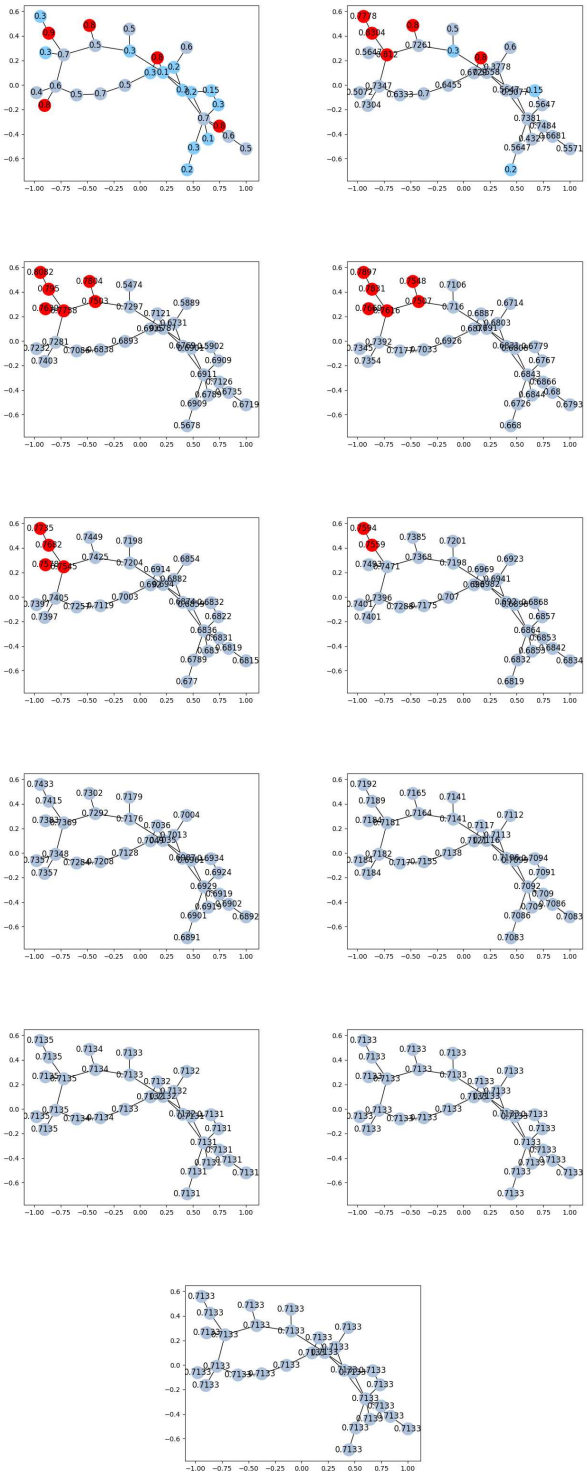

Figure 66: [Rate=1]: Convergence of the large network over [1,2,3,4,5,10,50,100,1000,10000] iterations

J MAXDIF: Varying with the rate

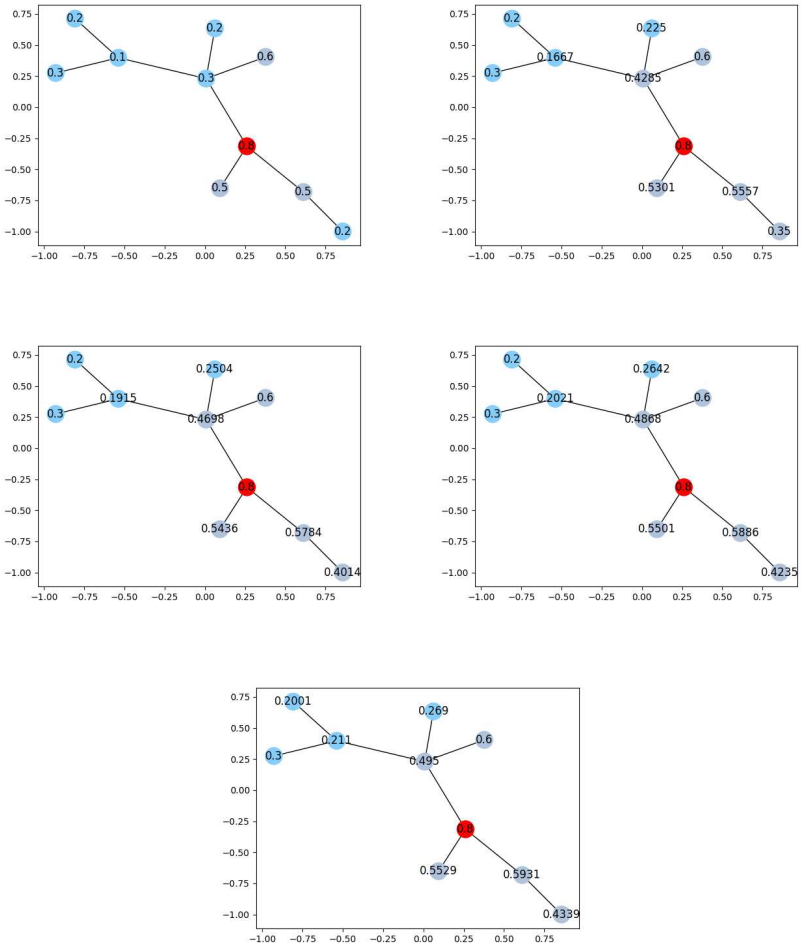

Figure 67: [Rate=$\frac{1}{2^t}$]: Convergence of the medium network over [1,2,3,4] iterations

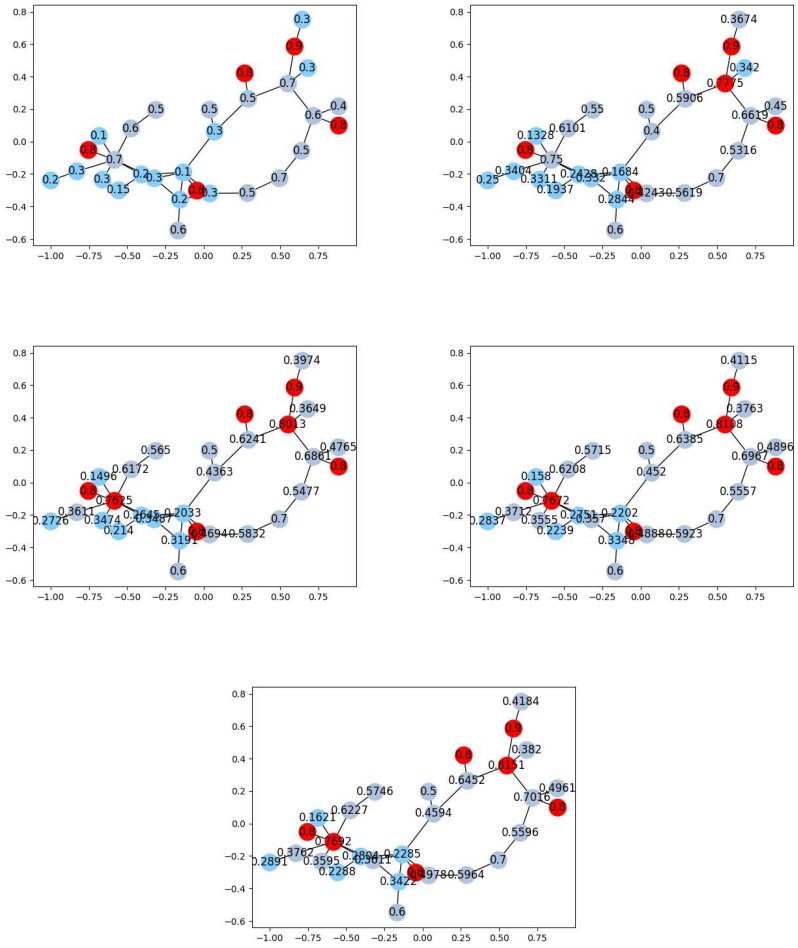

Figure 68: [Rate=$\frac{1}{2^t}$]: Convergence of the large network over [1,2,3,4] iterations

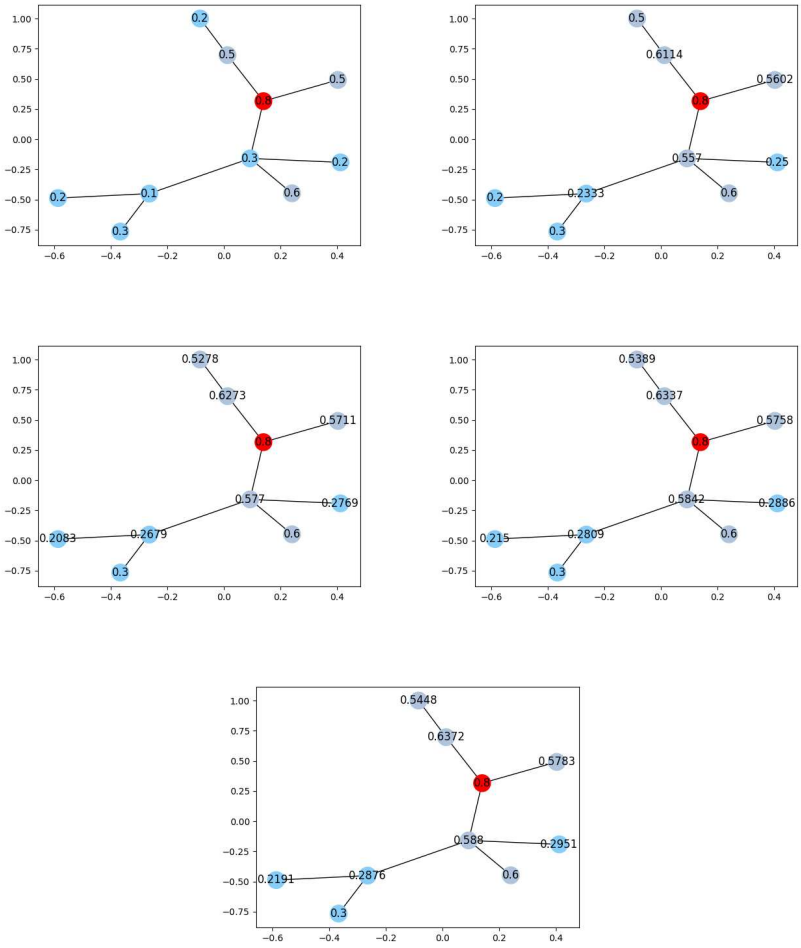

Figure 69: [Rate=$\frac{1}{t^2}$]: Convergence of the medium network over [1,2,3,4] iterations

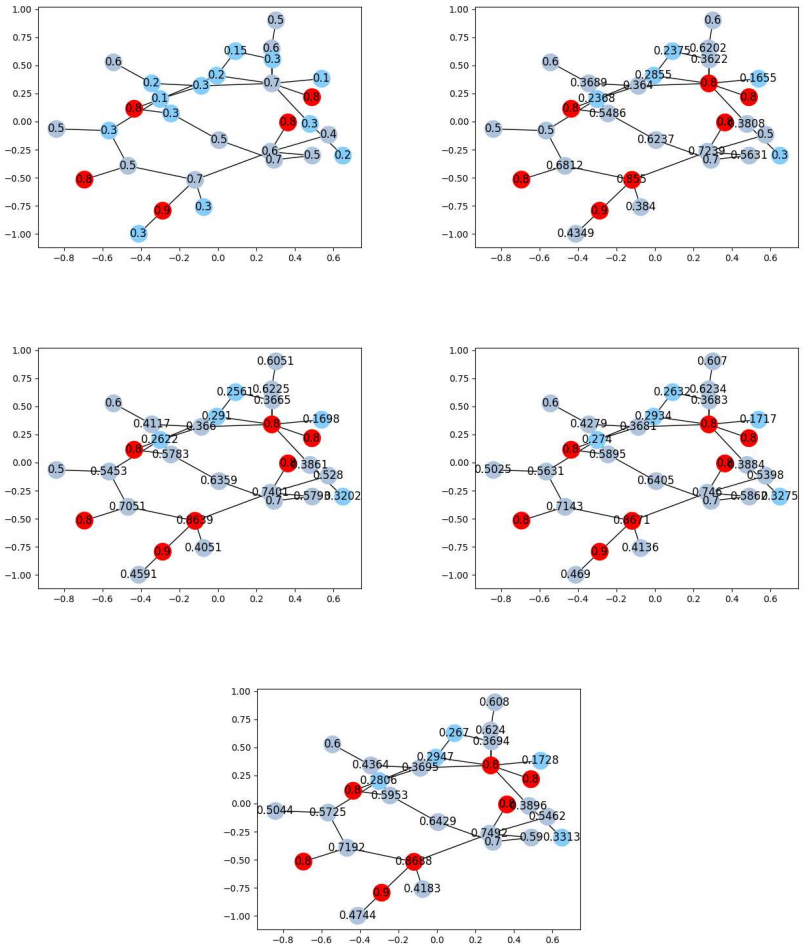

Figure 70: [Rate=$\frac{1}{t^2}$]: Convergence of the large network over [1,2,3,4] iterations