University of London
Imperial College of Science, Technology and Medicine
Department of Electrical and Electronic Engineering

Propagation of Rumours and Detection of their Sources in Social Networks

Robin Dupont - CID 01277640

First Marker and Project Supervisor: Professor Pier Luigi Dragotti
Second Marker: Mike Brookes

MSc Communications and Signal Processing
2016-2017

Abstract

Online social networks have a gradually increasing importance in our daily lives. They are a fast and reliable way of communication for human beings. Moreover, they provide a simple representation of human communities. Providing humans with fast communication methods is a favourable ground for rumours spreading. If the propagation of the rumours is somehow similar to the spreading of epidemics or diseases, finding the source of the rumours is a whole new problem, which drew attention of researchers recently.
This project tackles the problem of estimating the source of rumours, which are propagating in a social network. Use is made of a small number of monitoring nodes, which are regular nodes reporting information about the time of their infections by rumours.
This report contains explanations about the modelling of the rumour spreading across a network. A detection algorithm providing a ranking of the candidate nodes is presented. It is using two techniques described in the report: the creation of a set of candidates and the ranking of the candidates based on the scattering of the arrival of the rumours.
The performances of this detection algorithm are investigated under different conditions. It follows that there is a good probability of detection for rumours spreading fast enough in small world graphs.

Acknowledgements

I would like to thank my project supervisor, Professor Pier Luigi Dragotti, for his support all along the project, and for giving me the opportunity to tackle a challenging but nevertheless interesting subject. His precious advices and remarks helped me to keep progressing in my project and pushing forward my researches.
I would also thank some friends of mine for a few enlightening discussions we had, as well as their moral support.
Finally, I would like to extend the thank to my parents, for their continuous support and encouragement. They always provided me with excellent working conditions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This report aims to tackle the problem of the modelization of rumour spreading across a social network as well as the identification of the source of those rumour. The propagation of the rumours depends on different parameters which will be discussed. A detection algorithm will be presented with the relevant background theory, an eventually, the proposed solution performances will be analyzed, for different configurations.

The first Chapter of this report presents, from a general point of view, the aims of the project, the motivations and the possible applications. Related work is presented in the chapter two. Some article related to the field of study of this report are presented and the main conclusions are highlighted. Then, the third chapter settle the problem in a mathematical way. It presents how social networks are modelled and expose the contribution of this project. Next, the fourth chapter describe how rumours are propagated and tackles the detection problem by explaining the detection algorithms. The fifth chapter is dedicated to the performance analysis of the algorithms: metrics are established and the performances of the algorithms are investigated. Eventually, conclusions about this project are drawn in chapter six.

1.1 Motivations

Online social networks are an uprising trend. They allow people to communicate and share information all around the world, almost instantly. These networks are particularly interesting in so far as they present a digital version of already existing social communities based on real social interactions. This kind of social media dramatically simplifies the analysis of human communities because friendship and interactions are now binary information: there are either a relationship between two person and interactions or not. Moreover, the graphs obtained from social networks are representing the human communities in a good approximation since a tremendous part of the human interactions are now using such networks. It means that deductions made from the social graph can be extended to the human communities.

Among all the information that are circulating on social networks, rumours are the one which can be the most harmful. A rumour is a piece of information of any kind (a message, a picture, a video, etc ...) that is generally spread as a word of mouth from people to people. Rumours could be harmful because the piece of information they carry is not validated by any kind of authority, and the source is not known. Indeed, it is difficult to determine the source of a rumour because people only know who told them the rumour and to whom they told it.

The aim of this project is to present a method to detect the source of a rumour in a social network. This allows to achieve several objectives. Finding the source of rumours in a social network can be a way to perform early identification of influencers. Influencers are people whose messages have a massive impact on the people connected to them, they influence the other people, hence their name. If the information that a person reveals behave like a rumour, it may be because he can be a potential influencer. Finding the source of rumours can also help the police to identify and arrest people spreading rumours to perform psychological harassment on targets.

However, the detection of the source of rumours can be used for different problems. Identify an infected computer propagating a virus or a worm through a computer network such as internet or on a local network. Or detect the location of a pollution source in a water distribution network.

1.2 Project Aims

This project main objective is to provide an algorithm to detect the sources of rumours in a social network. The major milestones of this work are: a clear definition of the problem as well as the assumption and hypothesis made in order to solve it, a reusable graph generator and rumour spreading simulator to perform the tests, the derivation of an algorithm to determine the source of the rumours and a parallel performance analysis tool to assess the performances of the algorithm.

A literature review has given some insight about common practices in this field of research. The diffusion of a rumour in a social network is very similar to the Susceptible Infected compartmental model (SI model) used in epidemiology. The knowledge of this model, the Markov chain theory as well as some knowledge about graph theory helped to realise the first milestone: the reusable graph generator and rumour spreading simulator. The Python language have been used to guarantee a potential better reusability in the industry domain.

Some mathematical works have been done in the light of the recent research paper related to the subject has led to the derivation of a probability formula, which has been combined with set theory in order to produce a base for the detection algorithm. The detection algorithm yields a ranking of possible candidates, which allows more flexibility than just outputting a best candidate.

To assess the performance of a such algorithm, it is necessary that several tests should be run, in order to eliminate statistical variations, since the simulation is not deterministic. A simulation can be time consuming. Consequently, running tens of it is even more. Thats why it is needed to speed up the computations. Therefore, a parallel performance analysis tool has been developed, allowing to run simulations in parallel.

Chapter 2

State of the Art

2.1 Related Literature Survey

- **"Identifying Infection Sources and Regions in Large Networks"** [11]. In this paper, the authors aim to estimate infection sources in graphs with the knowledge of the graph topology and the state of the nodes: either infected or not. Their estimator is based on the count of all the possible infection sequence. The infection sequence represents the order in which the nodes have been infected. Their estimator can operate when there are 2 sources in a tree graph. The authors also provide a method for estimating the number of rumour sources in the graph. Eventually, if the maximum number of sources is known, their estimator appoints a best candidate a few hops from the true source.

- **"Rumors in a network: Whos the culprit?"** [13]. In this paper, the authors used the Susceptible-Infected model to simulate the propagation of rumours on a network. They built an estimator which requires knowledge of the graph topology, as well as the infected graph. The infected graph is the subgraph formed by all the infected nodes. Their estimator is a maximum likelihood estimator using topological information, which is called rumour centrality. This estimator is able to identify the source exactly, or a best candidate a few hops away from the real source. They focused their study on trees and then investigated the performance of their estimator on both synthetic and real graphs. On line-like growing trees, the probability of detection of their estimator tends to 0, whereas it is not the case for regular trees with a branching factor higher or equal to 3.

- **"Finding Rumour Sources on Random Trees"** [12]. This paper is from the same authors as [13]. They tackle the problem of the line-like growing trees. They kept the same estimator and derived result from it: this estimator achieve a strictly positive probability of source detection for random graph. Furthermore, the authors show that the probability that the distance between the best candidate and the real source is more than k hops decay exponentially with k.

- **"Rumor Centrality: A Universal Source Detector"** [14]. This paper is a slightly modified version of [12], which tackles the same problem and yields the same results.

- **"Identifying rumors and their sources in social networks"** [7].The authors of this paper provide an approach for rumours source detection in graphs as well as a way to classify the information propagate in the network. The classification allows to assess if the information is indeed a rumour or not. The authors used directed graphs to model social networks. They used monitoring nodes which report if they have been infected by the rumours or not. The rumour source estimator used the information provided by the monitoring nodes and combine them with node connectivity information and shortest path information.

- **"Information diffusion in online social networks"** [5]. The authors of this paper aim to summarize methods related to information diffusion in social networks. This paper covers the popular topic detection based on spikes of activity for a specific topic, diffusion models, explanatory models to give understanding about the path taken by the information in the graph, predictive models to give insight about the possible ways a information might spread across a network and eventually methods for possible influencers detections, based on the assessment of their influence on other nodes.

2.2 Overview

Most of the paper reviewed present an estimator for rumour source detection in a graph. The rumour is propagated by a Susceptible-Infected model which spreads rumour across the graph.

The same assumptions are made. The graph topology is known: all the nodes present in the graph as well as all their links to other nodes are known.

Most methods rely on having the knowledge of either a subset of the infected nodes, or the full set of the infected nodes. This last point seems to be a strong assumption, which might not be easily applicable in real life.

Most of the papers focuses their researches on tree graphs, although they tested their estimators on other types of graphs as well.

Chapter 3

Problem Statement and Terminology

In this section, the terminology of this project will be detailed, and the problem will be settled. First, it is explained to what extent this project brings a new contribution in this field of studies. The relevance of the hypothesis and assumptions that are made will be discussed. Then, the terminology that is used later in this report is introduced. Next, the usage of graphs to simulate a social network will be examined, as well as the different types of graph with their advantages and drawbacks. Eventually, the rumour propagation simulation will be presented.

3.1 A new approach

In order to estimate the source of rumours in a social network, time-based information is used. In most research papers, there is no assumption about the time. The start date of the spreading is not known, neither are the epochs or dates at which the nodes were reached by the rumour. However, to counterbalance this, the state of all the nodes, reached or not by the rumour, is known.

In this report, there are some differences with previous works. Time information will be assumed to be known. However, in order not to have a trivial problem, a new approach will be studied, based on monitoring nodes. Monitoring nodes are regular nodes, except that they report at which epoch they have been reach by the rumour. The monitoring nodes represent only a fraction of all the nodes.

Each monitoring node will report at which time step it has been reached by a rumour. With such time-based information, and since there is a single source which is propagating several rumours, it is possible to derive the experimental cumulative distribution function of the probability for a node to be infected after a certain number of steps.

A theoretical probability of infection and its cumulative distribution function can be computed. By comparing this theoretical cumulative distribution function with the experimental one through different metrics, it is possible to rank the possible sources based on the similarity of these two cumulative distribution function.

In order to reduce furthermore the number of candidates, time information gives insight about the possible set of candidates. By combining the information given by each monitoring node, the set of possible candidates is reduced.

These steps are the base of the new approach that will be detailed in this report.

In contrary to other research papers, this report will investigate the influence of the the probability of propagation on the detection performances.

3.2 Assumptions

To settle the problem, some assumptions have been made. Here is the list of them and some insight to justify them when needed.

- **Knowledge of the network topology**: The network topology is the arrangement of nodes and edges between them. Knowing the network topology means that we know to which other nodes is connected each node.
  This assumption can be easily justified, since we are interested by the propagation of rumours on online social networks. Such social medias provide an easy binary representation of relationship that allows to represent the social network as a graph.

- **Single source of rumours**: Only one node in the network is the source of the rumours. This is made in order to make use of the different time of arrival at the monitoring nodes of all the rumours.
  One single source of rumours is an assumption that can be argued. However, for the detection of influencers or people performing psychological harassment, this assumption holds, since this kind of persons tends to spread multiple rumours.

- **Unit threshold**: The threshold is the number of distinct people that has to tell one person the rumour before this person starts to believe in the rumour and start propagating it as well. In order to simplify the problem, the threshold have been set to one.
  This assumption is made only to simplify the problem since usually people might not trust a rumour straight away. However, one could argue that since the rumour is propagating through edges between close people, the threshold can be one because people might trust a rumour straight away if this one come from a trusted person.

- **Undirected edges**: in order to model a bidirectional relationship, the edges are undirected.
  A such assumption is valid in the general case, because people talk and respond to each other. This assumption however might not hold if the detection technique is applied on a graph representing an online social network such as Twitter where the edges between people are mostly directed.

- **SI model for rumour spreading**: The rumours spreads similarly to an epidemic according to the SI model (Susceptible and Infected). A person can either ignore the rumour and stay idle, or aware of the rumour and propagating it. It is assumed that one cannot forget the rumour.
  Rumours spreads generally really fast because it is a very surprising or choking piece of information. Hence, assuming that no one is going to forget about the rumours seems legit.

- **Same number of neighbours**: The number of neighbours for each node in the network is the same. It means that each person is talking to the same number of peoples.
  The degree distribution in a real human community is approximately following a power law. Therefore, a constant number of neighbours seems not appropriate. But only close relatives are only considered. It can be assumed that on average everybody has the same number of close relatives.

- **Same probability of propagation**: The probability of propagation of the rumours is assumed to be constant. Each node has the same probability to propagate the rumour to its neighbours.
  People are very different from one another, and the probability might in reality not be constant. But it is assumed that on average it is the case.

- **Multiple rumours spreading**: The source is spreading several rumours in the network. The number of those rumours is known. With this several rumours, it is possible to establish the experimental cumulative distribution function of the probability of being reach by the rumour after a certain number of steps.

  The people spreading rumours on social network are generally spreading several rumours. They do so because they can earn some money. For example, ads on the website where the rumour facts are described generates money. People performing psychological harassment want to discredit someone, thus they spread several rumours as well.

- **Monitoring nodes and time-based information**: Before the start of the rumour spreading, a few nodes, picked at random, are going to be turned into monitoring nodes (e.g. 10% of the nodes). These nodes will report at which step they have been reach by the rumour.

  One could argue that it is not possible to have precise time information. This assumption is clearly the most ambitious one. If in the general case it might not hold, it is possible sometime to obtain such information. Some rumours are posted on website where there is a time stamp, and the person acting as a monitoring node can report the time at which the rumour have been seen.

3.3 Terminology

Here is a list of the variables commonly used in this repot and their meaning, unless stated otherwise.

G	is the graph
N	is the number of nodes in the graph G

i	is a variable representing the index of a node of the graph G
S	is the rumour source
M	is the number of monitoring nodes in the graph G
W	is the set of all the monitoring nodes
W_i	is the monitoring node which node index in G is i
p	is the probability of propagation of the rumour
v	is the number of neighbours
R	is the number of rumours propagated by the source
I_i^j	is the state of node i with regarding rumour j. This is either equal to 1 (the node is infected), or 0 (the node is not infected.)
k	is a variable representing a number of steps
k_i^j	is the step at which the monitoring node i has been infected by rumour j
D_i	is the maximum distance hop-wise between node i and the source
d_{ij}	is the length of the shortest path between node i and j
P_n^m	is the probability that node m hops away from the source is infected by a rumour by step n
C	is the global set of all the candidates nodes c
C_i	is the set of all the candidates nodes c regarding the monitoring node i

3.4 Modeling a social network trough graphs

In human communities, people are usually talking to a small group of people that can be called friends or relatives. Everyone has a small number of friends, small in comparison to the total number of people in the considered community. People exchange with their friends and relatives much more than with the other peoples. Moreover, they exchange on any kind of subjects, especially personal topics, which is not the case with non-friend peoples. Therefore, since we are interested in the propagation of rumours it can be considered that people only interact with a reduced number of close relatives, which around five [8].

In order to model this kind of interactions, a graph structure will be used. Each person will be represented by a node of the graph. To represent the fact that two people are close relatives, an undirected edge links the two nodes. The edge is undirected to reflect the fact that the interactions between individuals are bidirectional.

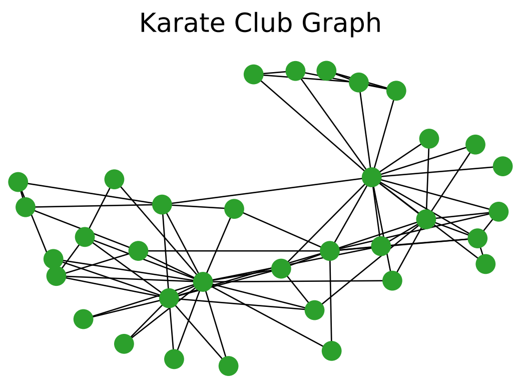

Figure 3.1: Example of real social network : The karate Club network as presented by Zachary[16]

Figure 3.1 shows an example of a graph representing a real human community. This is the famous karate club graph as described by Zachary [16].

3.4.1 Three types of graphs

To model human communities, several types of graphs will be used. Each type of graph has its own advantages and drawbacks, depending on the way the nodes are organised and links to one another. But the core principle (nodes represent people and edges represent friendship) is the same. Tree types of graph will be used:

- The Watts and Strogatz Model [15]. This model yields random graph with small-worlds properties.
- Scale-free graphs. These graphs have a power law distribution, which is what can be observe in real social networks.
- Tree graphs. Since a lot of research papers used that kind of graph for rumour detection.

Graphs yielded by the Watts and Strogatz model are known for showing good small world properties that mimic the real social networks properties. Like the random graphs, the average shortest path is small, which is what is expected from reality [10]. Moreover, and in contrary to random graphs, Watts and Strogatz small world graph show a high clustering coefficient. The global clustering coefficient is the number of closed triads (which can be seen as all the complete subgraphs of 3 nodes) over the number of all triads (which can be seen as all the subgraphs of 3 nodes). The clustering coefficient is the probability of having closed triads in the network.

A high clustering coefficient means that, for three nodes A, B and C, if the edges AB and BC exists, there is a high probability that the edge AC exist. From a social point of view, it can be translated as the following sentence: if A and B are friend, and B and C are friend, there is a high probability that A and C are friend as well. However, that is what can be observed most of the time in social networks. Hence, with a high clustering coefficient, the graphs yield by the Watts and Strogatz model are a good approximation of real social networks.

The main feature of scale free networks is that the degree distribution of that kind of graph follows a power law. The degree distribution is simply the histogram of the degree against the number of nodes of that degree. The degree is simply the number of neighbours of a node. In social networks, we also observe a power law distribution. It means that there are a lot of people with a small number of connections, and fewer people with a lot of connections. Since this is what can also be observed in real life social networks, scale free networks are to some extent useful for modelling social networks. The powerlaw distribution is particularly visible on some nodes with a lot of connections on Figure 3.3

The detection algorithm will be tested as well on tree graphs because this graph structure has been studied in previous papers. Moreover, even though this structure is not really suited for the modelling of social networks, it can model well a hierarchy structure, and therefore model social networks that can appear in companies for example.

Since the Watts and Strogatz graphs exhibits some properties that allow them to model real social networks with a good approximation, most of the study will be done on these graphs, and the other types of graphs will be used during the result analysis section, in order to compare the result obtained with such graphs to the ones with small world graphs.

Graphs construction

Small World Graphs

In order to generate the small world graph, all the nodes are positioned on a circle. Then each node is linked to its $\frac{v}{}$ neighbours on his left and its $\frac{v}{2}$ neighbours on its right (it can potentially be linked to $\lfloor \frac{v}{2} \rfloor + 1$ neighbours on one side and $\lfloor \frac{v}{2} \rfloor$ on the other if v is odd), where v is the number of neighbours. Then for each node, each edge can be rewired with a certain probability to another node picked at random in the graph, which is not necessary a neighbour. An example of Small World graph is shown on Figure 3.2.

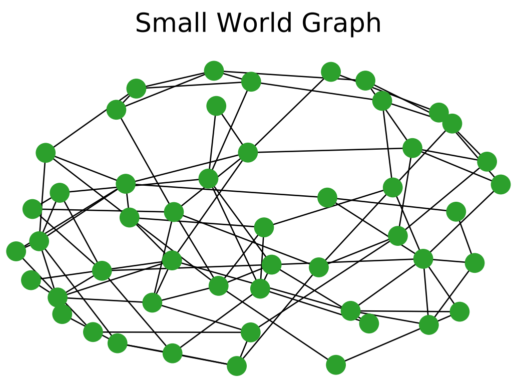

Figure 3.2: Example of a Small World Graph with 50 nodes, 5 neighbours per node, with a probability of rewiring of 0.3

Scale Free Graphs

The scale free graph is generated with the model provided by [1]. This model yields a directed graph, which is then converted to undirected. First a graph G_0 is created with a certain amount of edges and a random of nodes.Then the methods goes iteratively : G_{n+1} is created from G_n with 3 rules.

With probability α a new node is added with an edge from this new node to an older one, chosen based on is degree.

With probability β, a new edge is added from an old node to another old node. They are chosen independently according to their degree.

With probability γ, a new node is added with an edge from an old node to this new node. The old node being chosen according to his degree.

Where α, β and γ are probabilities such as : $\alpha + \beta + \gamma = 1$.

An example of scale free graph is shown on Figure 3.3.

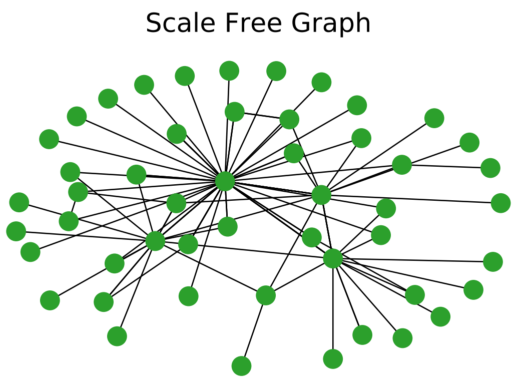

Figure 3.3: Example of a Scale Free Graph with 50 nodes

Regular Tree Graphs

A regular tree graph can be generated by taking a root node. Then this root node is given v neighbours called leaf nodes. Each of these v leaf nodes gets v additional leaf nodes as well, and so on, until a stopping criterion is met. The stopping criterion can either be the total number of nodes, or the depth of the graph. The depth of the tree can be seen as the length of the shortest path between the first root node and the last leaf node. An example of a Balanced Tree Graph is shown on Figure 3.4.

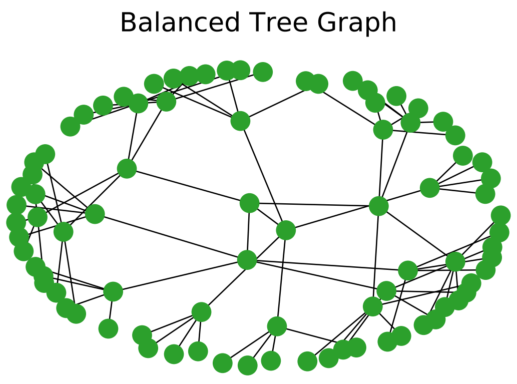

Figure 3.4: Example of a Balanced Tree Graph with 85 nodes, 4 neighbours per node and a depth of 3

Since small world graphs exhibit valuable properties when it comes to modeling human communities, the study will focus on small world graph, although the detection algorithm will be tested on other graphs as well.

3.5 Markov chains and 1D random walk

The problem tackled in this report can be seen as a discrete-time Markov Process. A discrete-time Markov process is a process where the probability of the next state happening is only determined by the current state and not by the previous states.
In mathematical terms we have :

$$P\{X_{n+1} = j \mid X_n = i_n, X_{n-1} = i_{n-1}, ..., X_1 = i_1, X_0 = i0\} = P\{X_{n+1} = j \mid X_n = i_n\}$$

Where $i_0, i_1, ..., i_n, j$ are all the possible states and X_k is the state of the process at the step k. E is the state space, containing all the possible states that X_k can have.
Therefore, going from state i to j can be written this way :

$$P_{ij} = P\{X_{\{}n + 1\} = j \mid X_n = i\}$$

If ones considers the propagation of a rumour along the shortest path, the problem is similar to a Markov chain or a 1D random walk with the particularity that it is not possible to go back.

Chapter 4

Propagation of the rumours and identification of the source

In this chapter, the creation of the graph and the propagation of the rumours in the graph will be described. Then, a method based on sets intersection will be explained. Another method based on similarity between the theoretical and experimental cumulative distribution of time arrival will be presented. Eventually, a detection algorithm combining the two methods will be presented and detailed.

4.1 Graph creation

The graph is created thanks to the utilities in the script `networkUtils.py` (see appendices). A few parameters can be changed to model different configurations. Here is a list of the most important parameters:

- N, the number of nodes
- M, the number of monitoring nodes

- S, the node which is the source of the rumours. The source is chosen randomly among all the N nodes of the network. The function to initialise the source in the graph is `initSourceNode` in `networkUtils.py`
- W, the set of all the monitoring nodes. The monitoring nodes are chosen randomly among the N nodes of the network. There are some constraint, though. All the W_i should have a distinct i. It is to make sure that $\text{Card}(E) = M$. Moreover, $\forall i, W_i \neq S$: a monitoring node cannot be the source.
- R, the number of rumours to propagates. Typically, R is chosen between one to a few tens.
- P, the probability of propagation. The higher p is, the faster the rumours are going to spread in the graph.
- T_i the threshold per node. Always set to one.

N.B.: The threshold is the number of nodes that have to infect a specific node before this nodes become infected. This is made to model the fact that a person will usually wait to hear a rumour from different persons before giving credit to it, and eventually starting to spread it. However, even though setting a threshold randomly as an uniform distribution $U(1, v)$ would model well the different sensitivity of the people, the threshold is set to one in order to simplify the problem.

4.2 Propagation of the rumours

4.2.1 Propagation algorithms

In order to understand how the detection algorithm works, it is important to understand how the propagation of the rumours occurs first. For each rumour a node has two possible states: either infected or not, which is represented by I_i^j. A node which is not infected stays idle. An infected node propagates the rumour as described in Algorithm 1. Considering the node i and the rumour j :

Algorithm 1 Propagates the rumour j at node i one step forward

```
for all neighbours $v$ of node $i$ do
    Draw random number $r$, $r \sim U(0,1)$
    if $r \leq p$ then
        Infect neighbour $v$ with rumour $j$
        next $v$
    else
        next $v$
    end if
end for
```

In order to process all the nodes of the network it is necessary to iterate over i and j. The function in charge of it is `infectionForward` in `networkUtils.py`. The global algorithm can be written as Algorithm 2.

Algorithm 2 Global propagation of the rumours one step forward in the graph

```
for all the nodes $i$ in the Graph $G$ do
    for all the rumours $j$ do
        if $I_i^j$ then
            Algorithm_1(node=$i$,rumour=$j$)
        end if
    end for
end for
```

Every time the function `infectionForward` is called, the rumours propagate a hop further. Every iteration is called a step. The initial state is the step 0. The propagation algorithm keeps track of the number of steps, to inform the node of their step of infection. The propagation algorithm ends when all the nodes are infected by all the rumours. The function `isAllInfected` from `networkUtils.py` checks if all the nodes are infected.

4.2.2 Rumour propagation on different types of graphs

Here are examples of different networks and infection steps. The green dots are the nodes. The nodes circled in red are infected by some of the rumours. The nodes fully red are infected by all the rumours. The nodes circled in light blue are the monitoring nodes. The node in dark blue is the source of rumours.

- **Small World Graph:** with $N = 50$, $M = 5$, $R = 3$ and $p = 0.5$.

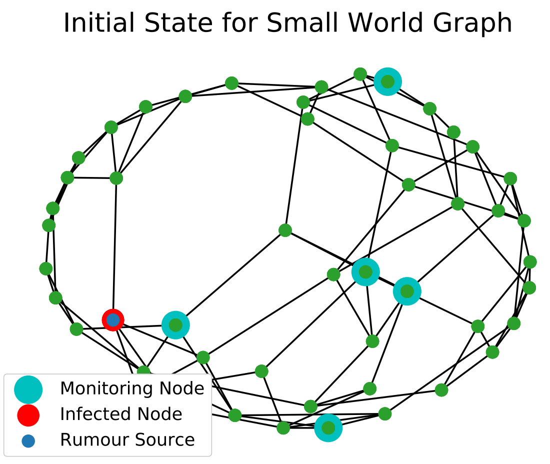

Figure 4.1: Example of a Small World Graph with the different types of nodes represented

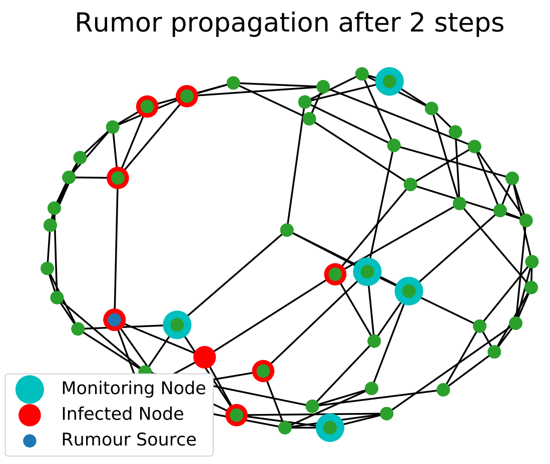

Figure 4.2: Same graph as Figure 4.1, after 2 steps of rumours propagation

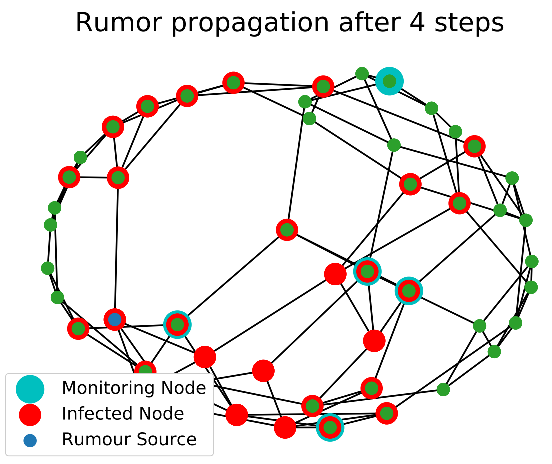

Figure 4.3: Same graph as Figure 4.1, after 4 steps of rumours propagation

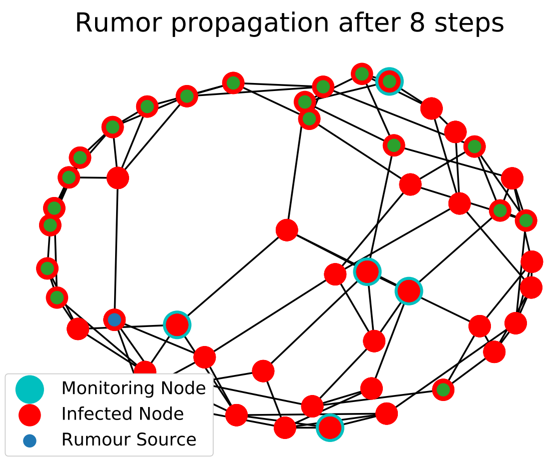

Figure 4.4: Same graph as Figure 4.1, after 8 steps of rumours propagation

- **Balanced Tree Graph:** with $N = 85$, $M = 5$, $R = 3$ and $p = 0.5$.

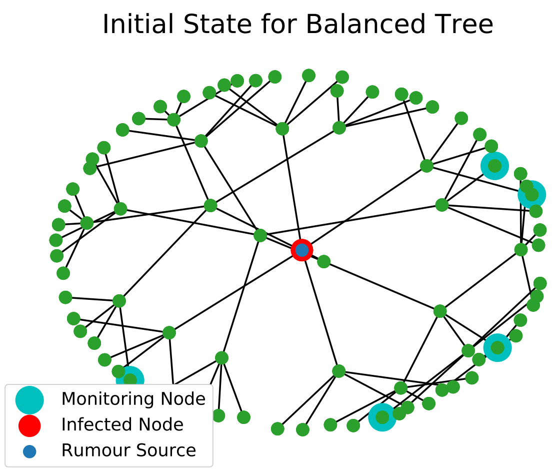

Figure 4.5: Example of a Balanced Tree Graph with the different types of nodes represented

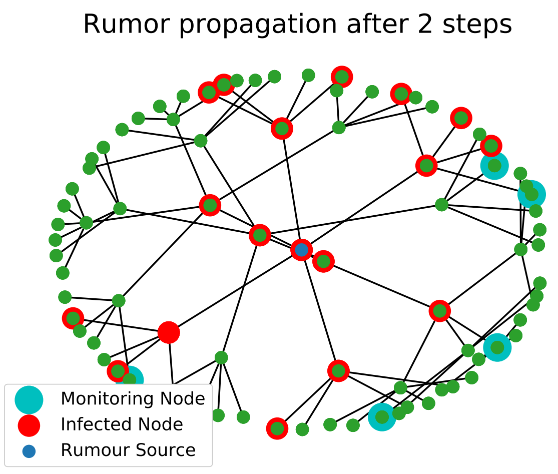

Figure 4.6: Same graph as Figure 4.5, after 2 steps of rumours propagation

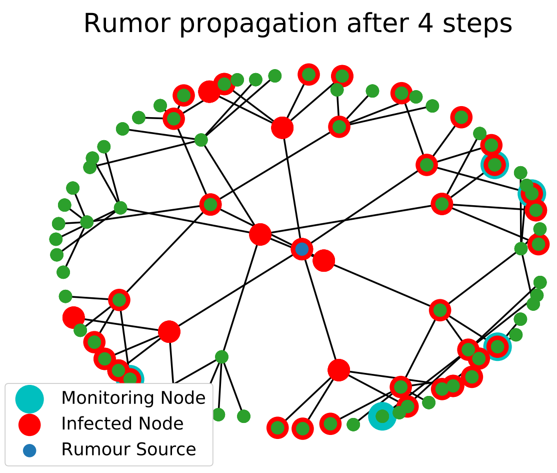

Figure 4.7: Same graph as Figure 4.5, after 4 steps of rumours propagation

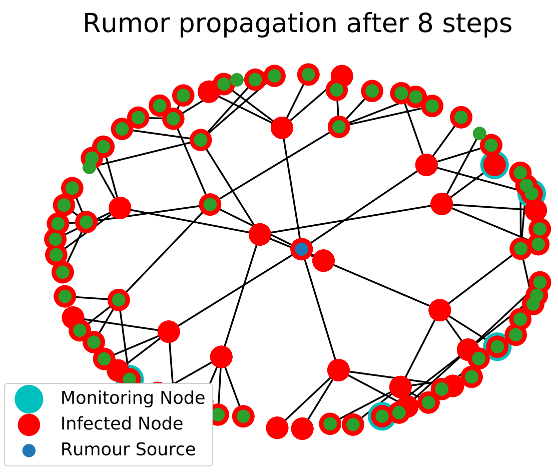

Figure 4.8: Same graph as Figure 4.5, after 8 steps of rumours propagation

- **Scale Free Graph:** with $N = 50$, $M = 5$, $R = 3$ and $p = 0.5$.

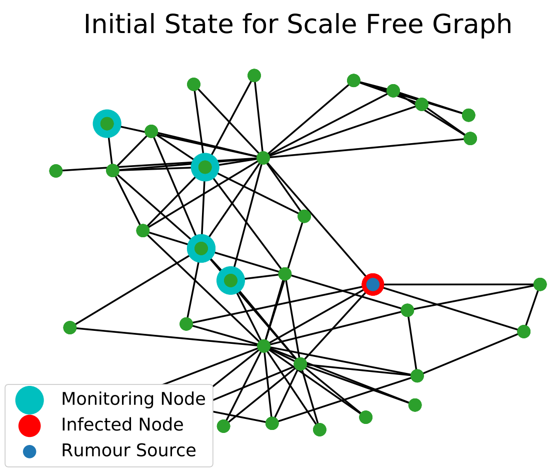

Figure 4.9: Example of a Scale Free Graph with the different types of nodes represented

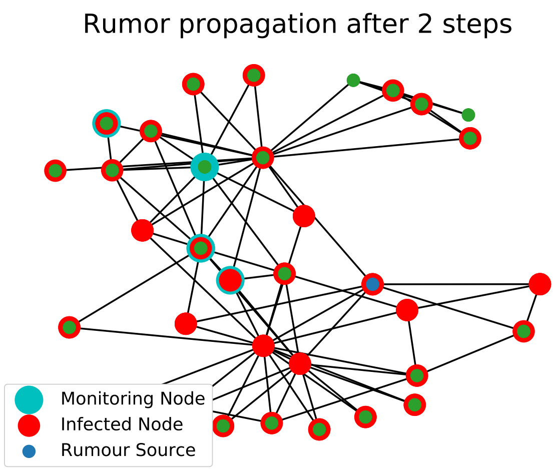

Figure 4.10: Same graph as Figure 4.9, after 2 steps of rumours propagation

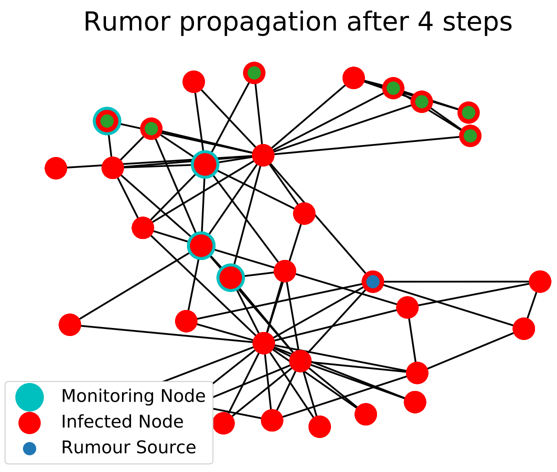

Figure 4.11: Same graph as Figure 4.9, after 4 steps of rumours propagation

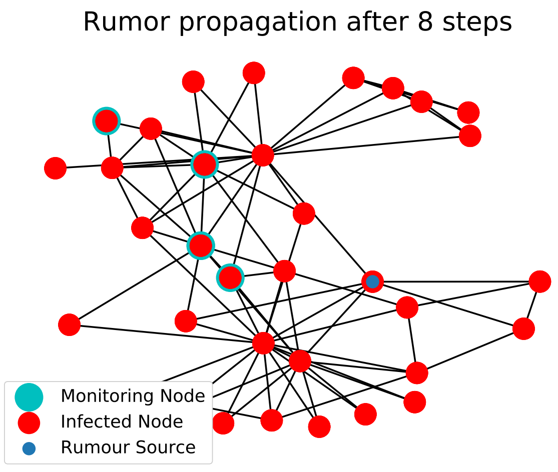

Figure 4.12: Same graph as Figure 4.9, after 8 steps of rumours propagation

4.2.3 Monitoring node reporting

The monitoring nodes are reporting information, which are helpful to detect the source. Each monitoring node report the step of arrival for each rumour. If a monitoring node i, is reached by a rumour j after 4 steps of rumour propagation, then this monitoring node reports $k_i^j = 4$. The information is collected at the end of the simulation. The data can be represented as on Table 4.1.

	rumour 1	rumour 2	...	rumour R
W_1	k_1^1	k_1^2	...	k_1^R
W_2	k_2^1	k_2^2	...	k_2^R
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$
W_M	k_M^1	k_M^2	...	k_M^R

Table 4.1: Example of the representation of the information yielded by monitoring nodes

4.3 The detection

This section will describe two methods to detect sources and then, the two methods will be combine create an efficient detection algorithm, providing a ranking of the candidates nodes.

4.3.1 Method based on first arrival time and on sets intersections

As described in Section 4.2.3, Each monitoring node is reporting the step of its infection by each rumour. Such information can be used to infer the maximum distance to the rumour source. Once the minimum distance found, a set of possible candidates can be established. Then a subset of candidates can be established from these sets.

First step

First, an estimate of the maximum distance between the rumour source and the monitoring node is needed. To achieve this, the smallest step of arrival per monitoring node is identified. This is the step at which the monitoring node have been infected for the first time by a rumour. This information can be found by selecting the minimum value of each row of Table 4.1.

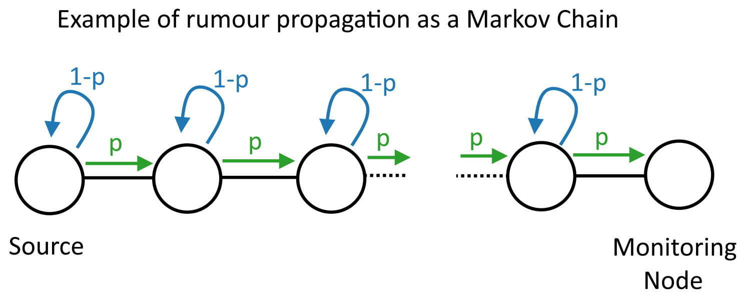

Figure 4.13: Example of rumour propagation as a markov chain along the shortest path between the source node and the monitoring node

The rumour propagation from the source to the monitoring node can be seen as a Markov chain, as shown on Figure 4.13, along the shortest path. It is assumed that the rumour will propagate along the shortest path and at each step, the rumour can either:

- propagate to the next node with propability p
- stay at the current node with the probability $1 - p$

Hence, it can be established that : If the node is at least k hops away from the monitoring node, the first rumour is going to reach the monitoring node in at least k steps. Material implication gives the following rules :

$$A \Rightarrow B \iff \neg B \Rightarrow \neg A \tag{4.1}$$

Where A and B are logical proposition, and $\neg A$ and $\neg B$ are their negations. The symbol $\Rightarrow$ is the material implication, and $\iff$ is the equivalence.

Thus, in our case we have :

- A : The source is at least k hops away from the monitoring node
- B : The first rumour is reach the monitoring node in at least k steps

Hence, we can deduct from the Equation (4.1) that : if the first rumour reach the monitoring node in at most k steps, the source is at most k hops away from the monitoring node. Topologically speaking, the maximum distance in the graph between the rumour source and the monitoring node is known. For each monitoring node i, this distance is:

$$D_i = \min_j(k_i^1, k_i^2, \ldots, k_i^j, \ldots, k_i^R) \tag{4.2}$$

Second step

With the knowledge of D_i, all the nodes which are farther than D_i from monitoring i can be eliminated: there cannot be candidate to be the rumour source. Hence, all the remaining nodes are candidates (except the monitoring node itself).

In order to create the set of candidates per monitoring node i C_i, all the nodes c which satisfies $d_{ic} \leq D_i$ should be found. Mathematically speaking, the set described in Equation (4.3) should be created.

$$C_i = \{\ c \mid d_{ic} \leq D_i\ \} \tag{4.3}$$

A natural approach to create the set C_i is to gather all the neighbours of node i and then their neighbours, and so on, until nodes that are D_i hops away are reached. However, due to the data structure of the graph (mostly determined by the programming language or the library used), it is not always possible to perform like this. In this case another approach has been used.

For each node in the graph, the distance from that node to the considered monitoring node is computed thanks to the Dijkstra shortest path algorithm [2]. If this distance is smaller than D_i, then the node is added to the candidate set, otherwise, it is not. This operation is then repeated to obtain a collection of all the C_i: $\{C_1, C_2, \ldots, C_i, \ldots, C_M\}$.

Third step

Since there is a single source for the rumours, the source is the same for all the monitoring nodes. Consequently, the source have to be in all the candidates sets. The mathematical formlation is described in Equation (4.4).

$$S \in C_i, \quad \forall i \in [\![0; M]\!] \tag{4.4}$$

In order to identify all the candidates satisfying Equation (4.4), the intersection of all the is took (see Equation (4.5)). It yields a subset C of all the nodes which are presents in all the sets C_i.

$$C = C_1 \cap C_2 \cap \cdots \cap C_i \cap \cdots \cap C_M \tag{4.5}$$

The subset C is the final subset of the candidates. It is guarantee that all the candidates in C are in all the C_i and therefore are possible sources. The Figure 4.14 shows an example of set intersection with two monitoring nodes, one reporting the source 1 hop away and the other one 2 hops away. The boundaries represent all the nodes which are candidates for a single monitoring node, or in other world a set C_i. The intersection of the two boundaries is the final set C.

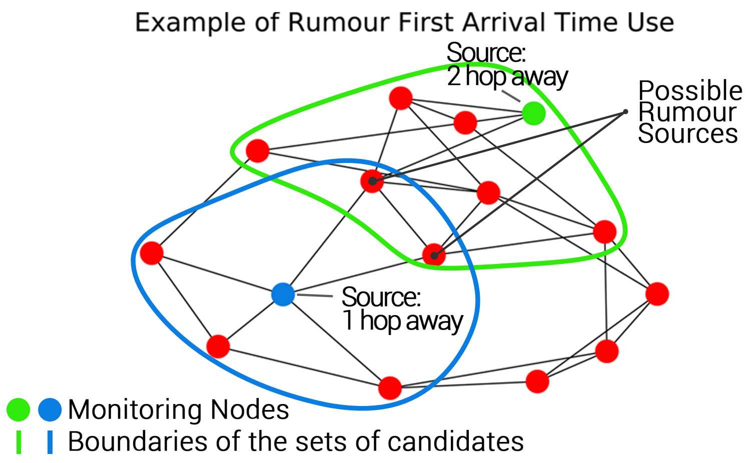
Example of Rumour First Arrival Time Use
Source:
2 hop away
Possible
Rumour
Sources
Source:
1 hop away
Monitoring Nodes
Boundaries of the sets of candidates

Figure 4.14: Example of usage of first arrival time information to determine candidates per node and then possible sources.

Formal algorithm

The global algorithm presenting all the steps to generate the subset of candidates C is presented in Algorithm 3. The function responsible for the generation of this set is `findPossibleSets2` in `networkUtils.py`.

Algorithm 3 Generation of C, the subset of all the possible sources

```
all_sets = {}
for all the monitoring nodes $i$ do
    $C_i = \{\}$
    $D_i = \min_j(k_i^j)$
    for all the nodes $c$ in $G$ do
        $d_{ic}$= compute_shortest_path$(i, c)$
        if $d_{ic} \leq D_i$ then
            $C_i = C_i \cup c$
        end if
    end for
    all_sets = { all_sets, $C_i$}
end for
$C = \bigcap$all_sets
```

Advantages

This method allows to simply reduce the number of candidates and ensure that the real source node is necessary in the final subset. It allows to eliminate a certain number of nodes and therefore simplify the problem by limiting the number of candidates to investigates.

The higher the number of rumours is, the more accurate the result is, because it is more likely that the first rumour will arrive in the minimal number of steps permitted. However, the algorithm could be run with only one rumour and still produce a useful output.

Under certain circumstances, if the rumours propagates fast enough, this method can isolate one candidate only. This happens when rumours reach two different monitoring nodes in the minimal amount of steps permitted.

Drawbacks

This method is computationally expensive, since the distance from all the monitoring nodes to all the other nodes has to be computed. The worst case complexity is $O(MN^3)$, Where M is the number of monitoring nodes, and N the number of nodes. This complexity is partly due to Dijkstra's Algorithm, which complexity is $O(N^2)$.

The size of the final subset C is influenced by the probability of propagation p. If p is low, the rumours will take a lot of steps to reach the monitoring nodes. Hence, the D_i will be high and consequently, the sets C_i will cover a lot of nodes. Therefore, if p is low the size of set C will contain almost all the nodes and the reduction of the number of possible candidates will not be significant.

Possible optimizations

Another implementation of Dijkstra's algorithm whith a complexity in $O(E + N\log(N))$ can be used[4], where E is the number of edges in the network. However it is necessary to convert the network edges to directed edges with non negative weights.

To speed up the process, loops can be run in parallel, since every loop iteration is independent from the others. Although this method allows to speed up the process, it is limited by the number of threads which can be run simultaneously.

4.3.2 Method based on similarity of cumulative arrival time distributions

In the cases considered, the source is propagating several rumours. Although the probability of propagation is the same for every rumour and in all the graph, the different rumours does not spread exactly the same way. Some might experience a bit more delay than others, by staying at the same node for a step or more. Some might experience less delay by propagating farther at almost every step. The delay is introduced if the rumour stays at the same node for at least one step. This occurs with a probability $1-p$ for every neighbours, and for every step, as shown on Figure 4.13.

Hence, the step of arrival (or time of arrival) of the rumours at a given node is different, depending on the rumour. If the total number of rumours received by a node is plotted against the step number, an experimental cumulative distribution of time of arrival can be obtained, as shown on Figure 4.15.

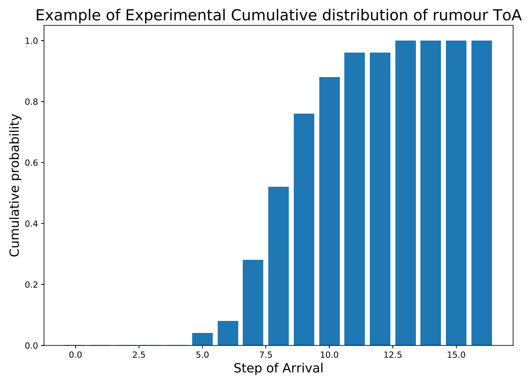

Figure 4.15: Example of an experimental cumulative distribution of the Time of Arrival (ToA) for a monitoring node in the graph show on fig. 4.16

This distribution is the cumulative distribution of the time of arrival of a monitoring node from the graph show on Figure 4.16. This graph has 200 nodes and 20 monitoring nodes. The source propagates 25 rumours with a probability p of 0.5

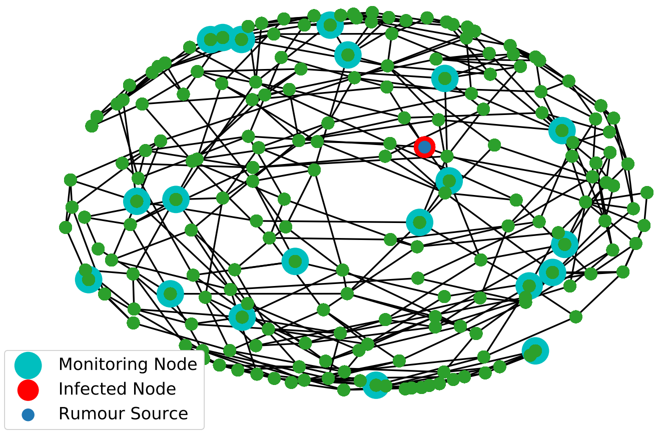

Figure 4.16: Example of a Small World Graph, used for simulation.

The main idea of this method is to take advantage of the scattering of the time of arrival to compare it with a theoretical distribution. If a node, which is supposed to be the source, yields a theoretical distributions similar to the experimental ones for every monitoring nodes, there are good chances that this node is the real source.

Theoretical explanations

In order to compare experimental and theoretical distribution, the theoretical distribution have to be derived. A formula which gives the probability for a node i at d_i hops from the source to be infected by step n is needed. As explained in Section 4.3.1 a rumour propagation along the shortest path can, at each step, either stay at the same node with probability $1 - p$, or propagate to the next node with probability p. Therefore, on its way to a monitoring node, a rumour can stay at the same node once, several times or not at all. Figure 4.17 shows an example of the delays possible from 1 up to 3 delays, for a target node 2 hops away from the source. Consequently, for a given number of delays, it exists several paths with different delays repartitions satisfying this number of delays.

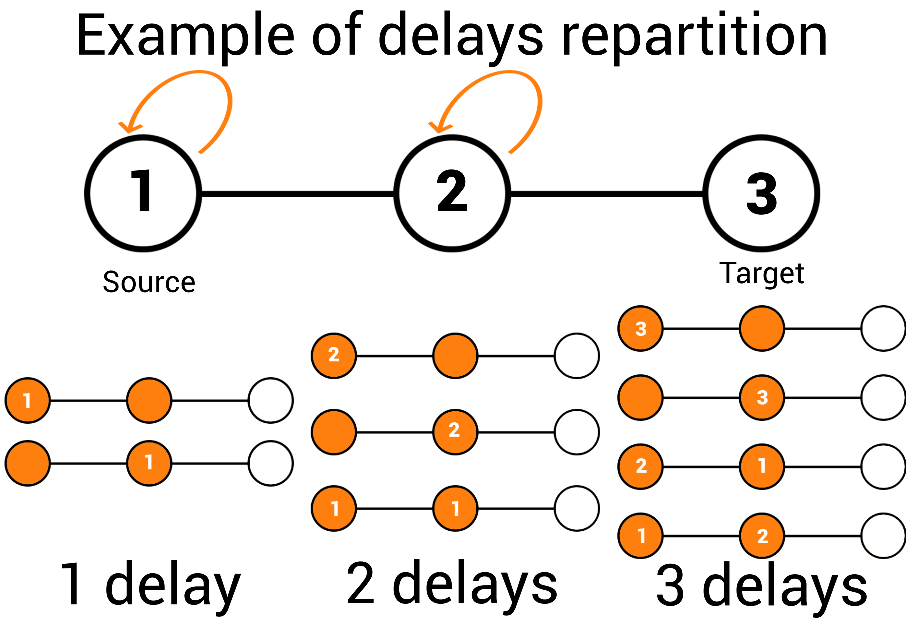

Figure 4.17: Example of delays repartition for a rumour propagating to a target node 2 hops away from the source. The repartition is shown only up to 3 delays

For a rumour moving from the source to a node k hops away, the probability that the rumour arrive at step n with a specific delay repartition, is presented in Equation (4.6).

$$P(\text{node infected by step } n \text{ by one path} \mid d_i = k) = p^k(1-p)^{n-k} \tag{4.6}$$

However, this is not the probability that the node is infected by step n. To derive this probability we have to take into account that several paths can have the same number of delays. Moreover, as shown on Figure 4.17, several delays can occur at the same node. To take this effect into account a slightly changed binomial coefficient is used. The correct coefficient counting the correct number of path l hops long with m delays is written in Equation (4.7), which can be seen as the number of m combinations in a $m+l-1$ elements set.

$$\Gamma_l^m = \binom{l+m-1}{m} \tag{4.7}$$

Where Γ_l^m is the number of m combinations in a l elements set with replacement. All in all, the final probability formula can now be establish by summing on all the possible delays. For a node k hops away from the source, the delays can be between 0 (the rumours propagates straight to the node) to $n-k$, where n is the number of steps elapsed since the beginning of the rumour spreading. The probability that a node k hops away from the source is infected by step n, P_n^k, is given in Equation (4.8).

$$P_k^n = \sum_{i=0}^{n-k} \binom{k+i-1}{i} p^k (1-p)^i \tag{4.8}$$

Practical Implementation

With the Formula 4.8, it is possible to obtain the experimental distribution of the rumours step of arrival. The formula should be applied for different steps ranging from 0 to the maximum number of steps took by the slowest rumour to reach the last monitoring node. This value is obtained by computing $\max\limits_{i,j}(k_i^j)$, where k_i^j are the steps definied in Section 4.2.3, Table 4.1.

The distributions are discrete, since the x-axis is representing the number of steps. Therefore, the distribution H can be considered as a vector, where H_i is the value of the cumulative distribution for the step i, or the i^{th} coordinate of the vector. Hence, we can define the two metrics for the distribution, which are :

- **The L_2 norm**: The distance D_{PQ} between to vectors P and Q using the L_2 norm is :

$$D_{PQ} = \sqrt{\sum_i (P_i - Q_i)^2}$$

- **The χ^2 distance**: The distance D_{PQ} between to vectors P and Q using the χ^2 distance is :

$$D_{PQ} = \sqrt{\frac{1}{2}\sum_i \frac{(P_i - Q_i)^2}{(P_i + Q_i)}}$$

  Experimentally, the χ^2 is supposed to show more robustness to outliers and to be a better metric than L_2[9]. However it will be shown in Chapter 5 that, experimentally and for this problem, L_2 outperforms χ^2.

Figure 4.18 shows a comparison between experimental and theoretical distribution. On the left, the theoretical distribution is the one yielded by the source, which is very similar to the experimental one. On the right, one can see that the theoretical distribution from a node that is not the source is, significantly different from the experimental one.

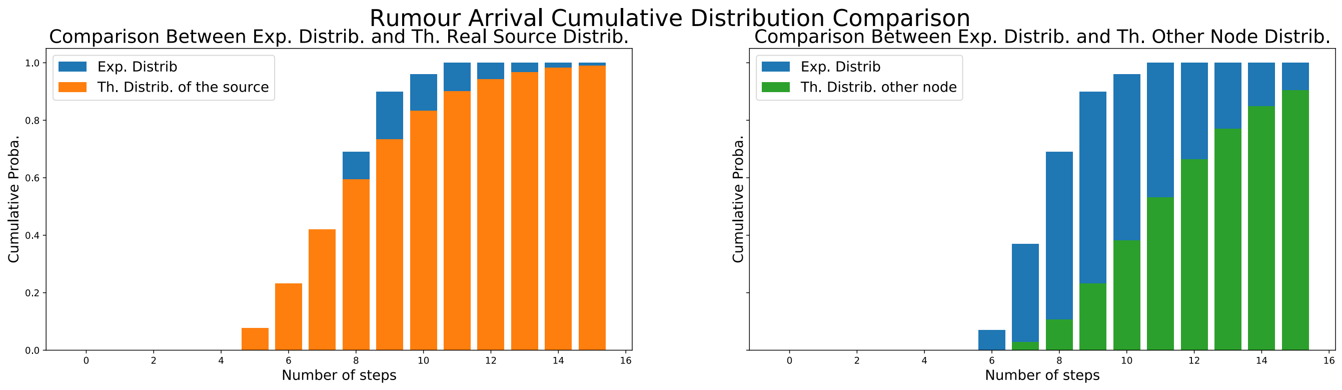

Figure 4.18: Comparison of cumulative distributions. On the left, the theoretical distribution at a monitoring node yield by the real source (in orange) and the experimental one at the same monitoring node (in blue). On the right, the same experimental distribution (in blue) and a distribution yield at the same monitoring node by a node which is not the source.

Once all the distances are computed, each monitoring node yields a ranking of the candidate nodes. The first one is the node which is the most likely to be the source according to the considered monitoring node. In order to combine the results from all the monitoring nodes, a weighting system leading to the final ranking is set.

All the nodes starts with a score of 0. Then, for each node i, each monitoring node adds to the score of i its ranking for the considered monitoring node. Then the node are sorted globally according to their weight. The node with the smalled weight is globally the best candidate to be the source.

Figure 4.19 sums up the process. **Step 1**: The theoretical distribution at each monitoring node is computed for each node, as if this node was the source. **Step 2**: Each monitoring node ranks the candidate nodes. **Step 3**: All the ranks of a node are added together, forming its weight. **Step 4**: The nodes are globally sorted according to their weight.

Finally, all the candidates are ranked. The one with the smalled weight is the best candidate according to the algorithm.

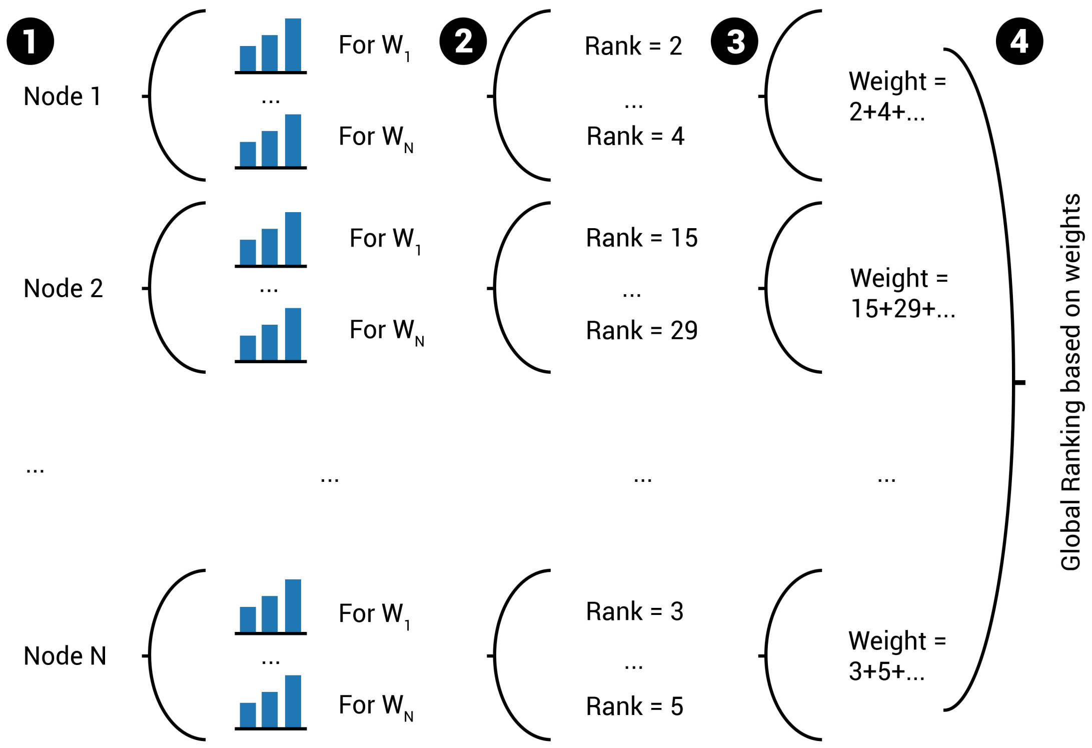

Figure 4.19: Schema explaining the process of ranking the nodes

Advantages

In contrary to the first method, this method provides a ranking of the nodes. Moreover, this method is less computationally expensive than the previous one. The operations made for each node can be considered as constant in time, except the ranking. The ranking in Python is in $O(n \log(n))$[6]. Since the ranking is performed by every monitoring node, and once at the end, the global complexity is in $O((M + 1)N \log(N))$, where N is the number of nodes in the network and M is the number of monitoring nodes.

Drawbacks

One of the major drawback of this method is that all the nodes have to be tested. Which takes some time and is useless for some nodes which cannot be the source, as shown in Section 4.3.1. Moreover, in order to establish the time of arrival distribution, several rumours are needed. Furthermore, this method works better for lower probability of propagation p. The lower p is, the more scattered the steps of arrival are, and therefore the more useful the distribution is.

Possible optimizations

As mentioned in the drawbacks, all the nodes needed to be tested. An optimization would be to pre-select the candidate nodes in order to limit the computations and have less candidates.

4.3.3 Global detection algorithm

The remark in the possible optimizations section of Section 4.3.2 suggests that the second method should be run on a limited set of candidates. In order to achieve this, a natural idea is to combine the two methods described in Section 4.3. Hence, this new method simply combine the previous two methods in two steps :

- **1$^{\text{st}}$ Step**: The number of candidates is reduced to a smaller subset of all the nodes thanks to the first method.
- **2$^{\text{nd}}$ Step**: The candidates from the subset are ranked thanks to the second method.

Eventually a smaller and more reliable ranking is obtained, combining the advantages of both methods. The two methods are complementary, since they both work best in different conditions. The first method yields good result for high probability of propagation, whereas the second one benefits from low probability of propagation. The performances of this global algorithm are investigated in Chapter 5.

Chapter 5

Results analysis

5.1 Settings of the performance analysis

In order to assess the performances of the algorithm described in Section 4.3.3, it is needed to run tens of simulation to obtain average performances. A performance test has been set up with the following parameters. They remain the same across all the tests, except if stated otherwise.

- Small World Graph have been used, except in Section 5.2.4 where different types of graphs are tested. Hence, the performances analysis are run on synthetic data.
- 250 nodes in the graph
- 10% of monitoring nodes, except in Section 5.2.3 where the influence of the number of monitoring node is investigated
- Probability of propagation p ranging between 0.2 and 0.9
- 20 rumours propagating in the network.
- 250 simulations with the same parameters have been run each time and the results are averaged over these 250 trials.

In order to speed up the performances analysis, test have been run in parallel. The file `Bench_parallel.py` starts the benchmarking made by `runHistoSimulation` in the file `benchmark_histo.py`. Each instance of `runHistoSimulation` is a unique simulation. These simulations are made in parallel on several cores. The number of cores depend on the machine. In this case, test have been run on the `batch2` server from the Department of Computing, using a total of 30 cores.

In this section, different aspect of the detection will be investigated and assessed:

- **The probability of detection**. This probability represents the probability that the best candidate appointed by the algorithm is indeed the real source. In Section 5.2.2, this probability will be compared with the probability of the real source being in the first 5 candidates as well as the first 10.
- **The rank of the real source** S. The rank of the real source is the rank of the real source S yield by the algorithm, averaged over the 250 simulations for each set of parameters.
- **The distance from the 1st candidate to the real source**. This is the distance in terms of hops, in the graph, from the first candidate to the real source.

5.2 Performances analysis

5.2.1 Metrics comparison

The Figure 5.1 shows the probability of detection for standards parameters ($N = 250$, $M = 25$, $R = 20$, small world graph, with 250 simulations). One can see that the L_2 norm outperforms the χ^2 distance. This is the case for all the tests made. Therefore, starting from now and until the end of the report, only the results obtained with the L_2 norm will be shown.

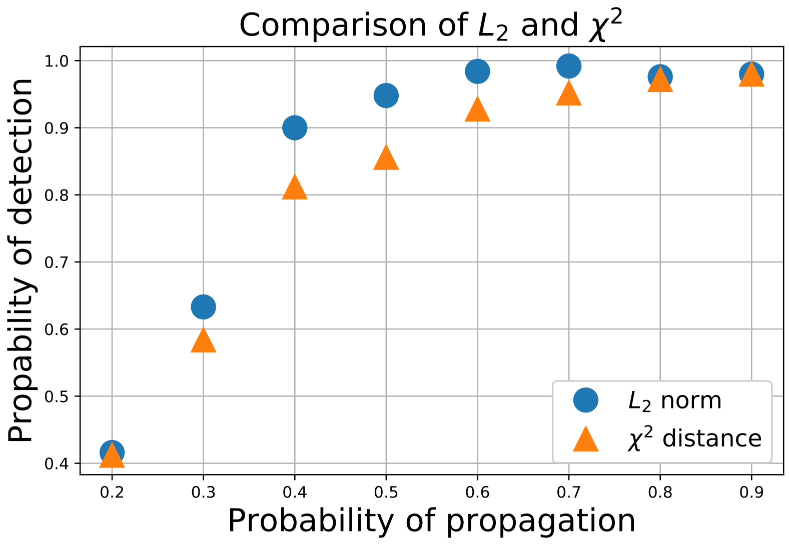

Figure 5.1: Comparison between the two metrics with standards parameters

This experimental result is quite surprising in so far as the χ^2 distance is supposed to be better suited for distribution comparison than the L_2 norm[9]. One can see that for low probability of propagation the two metrics perform similarly. However when the probability of propagation increases, the L_2 norm performs better. For high probability of propagation both metrics are performing equally, because the first step of the global algorithm (the reduction of the number of candidates) is doing most of the job by providing a really reduced set of candidates.

In the middle zone, where $p \in [0.3; 0.7]$, χ^2 distance is outperformed. This can be explained by the fact that the probability of propagation is too high to form a smooth distribution. Too much rumours arrive at the same step. Therefore it produces too large bins for χ^2.

5.2.2 Standard parameters results

Probability of detection

This section present the results obtained for standard parameters ($N = 250$, $M = 25$, $R = 20$, small world graph, with 250 simulations). Figure 5.2 show the probability of detection against the probability of propagation if the first candidate is picked, if the first five are considered and eventually the first ten.

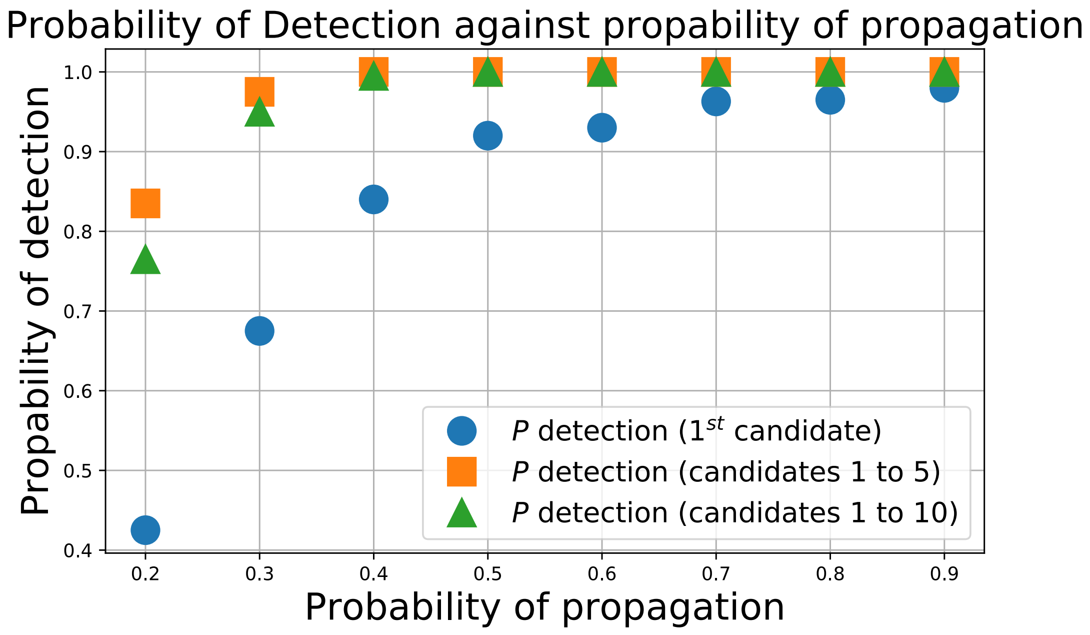

Figure 5.2: Comparison of the different probability of detection for different assumption : the source is the first one, the source is in the first five and the source is in the first ten.

From Figure 5.2, one can see that there is more than 80% of chance to identify the source if the probability of propagation is greater of equal to 0.4. There is a negligible difference between picking the first 5 and the first 10 : around 5% maximum. Hence, picking the first five candidates is the best assumption of the two since it yields comparable results and divide by two the number of possible candidates. Furthermore, the real source is 100% of the time in the first five candidates provided that the probability of propagation is greater or equal to five.

Average rank of the source and distance from the first candidate to the source

Table 5.1 shows the average rank of the real source and distance from the first candidate to the source, with standards parameters, for different probabilities of propagation. The index start at 0, therefore the best candidate has the rank 0, the worse has the rank Card(C).

p	avg. rank of S	dist. from 1st to S
0.2	8.29	1.345
0.3	1.265	0.605
0.4	0.19	0.27
0.5	0.055	0.105
0.6	0.007	0.072
0.7	0.005	0.092
0.8	0.002	0.112
0.9	0.002	0.048

Table 5.1: Average rank of the real source and distance from the first candidate to the source for standards parameters.

It is clear that the rank of the real source quickly tends to the first position (the ranking stats at 0). Furthermore, the distance between the real source and the first candidates reduces as the probability of propagation increases, but it does not exceed 2 hops. It means that the first candidate stays close to the real source, topologically speaking in the graph. Therefore even if the real source has not been identified, the best candidate is likely to be a relative of the real source.

Overall

Overall, the probability of detection increases with the probability of detection. The source is detected at least 90% of the time, if the probability of detection is greater or equal to 0.5. It is possible to isolate a group of five candidates containing the source for a probability grater or equal to 0.3. Even if the probability of propagation is low ($p \leq 0.3$), the best candidates is close, topologically speaking, to the real source, and their are good chances that the two are relatives, or at least in the same friend group.

5.2.3 Influence of the percentage of monitoring nodes

With the standard parameters, 10% of the nodes are acting as monitoring nodes. However, it might not always be possible to get this amount of monitoring nodes. Consequently, the influence of the number of monitoring nodes must be investigated. All the other parameters are standard.

The Figure 5.3 and the Table 5.2 show the results of the performances analysis when there are 10%, 4.8% and 2% of monitoring nodes in the network. Since there are 250 nodes, 5% of 250 is 12.5, which is not an integer. Therefore, 4.8% has been chosen, which represents 12 monitoring nodes for a 250 nodes graph.

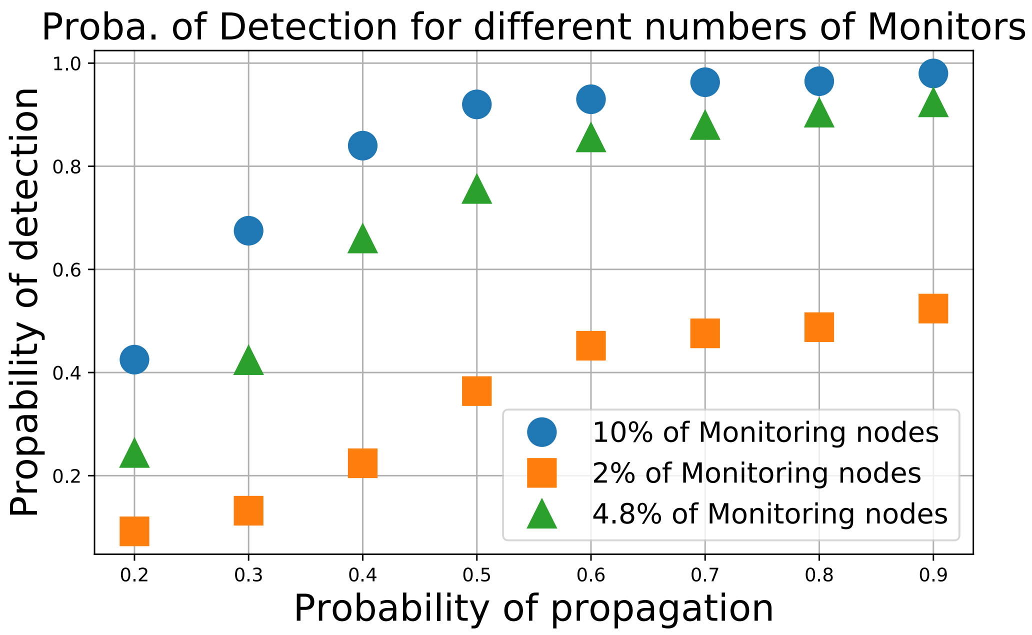

Figure 5.3: Probability of detection for different percentage of monitoring nodes

	10% of monitoring nodes		4.8% of monitoring nodes		2% of monitoring nodes	
p	avg. S rank	d from 1st to S	avg. S rank	d from 1st to S	avg. S rank	d from 1st to S
0.2	8.29	1.345	24.28	2.24	36.48	3.184
0.3	1.265	0.605	4.776	1.448	17.148	3.152
0.4	0.19	0.27	1.16	0.92	9.304	2.816
0.5	0.055	0.105	0.512	0.732	5.2	2.292
0.6	0.007	0.105	0.196	0.492	2.328	1.924
0.7	0.005	0.075	0.208	0.42	2.264	2.004
0.8	0.002	0.069	0.168	0.328	2.276	2.172
0.9	0.002	0.053	0.14	0.184	1.584	1.656

Table 5.2: Average rank of the real source and distance from the first candidate for different percentage of monitoring nodes.

Overall

Having only 2% of the node acting as monitoring nodes yields poor results. Regardless of the probability of propagation, it is never possible to achieve a better probability of detection than 0.55. If 5% of the nodes are monitoring nodes, the probability of detection is significantly better than with 2% of the nodes (around 10 to 15% higher). Moreover, for probability of propagation, higher than 0.5, 5% of monitoring node is performing almost as well as in the case with 10% of monitoring nodes (less than 5% worse). Choosing 5% of monitoring nodes seems to be the ideal trade-off between feasibility and performances, although, for low probability of propagation ($p \leq 0.3$), having more monitoring node is recommended.

5.2.4 Type of graphs

It has been explained in Section 3.4.1 that the study is focussed on Small World Networks due to their valuable properties regarding human communities modelling. However, it is important that the performances of the detection algorithm should be assessed on other types of graph. It shows to what extend the detection algorithm applied on other graphs is reliable.

Figure 5.4 and Table 5.3 show the results of performances analysis for small world graphs, scale free graphs and tree graphs. The tree graphs used are balanced tree graphs of depth 3 and 5 neighbours per node (plus their root node), for a total of 155 nodes.

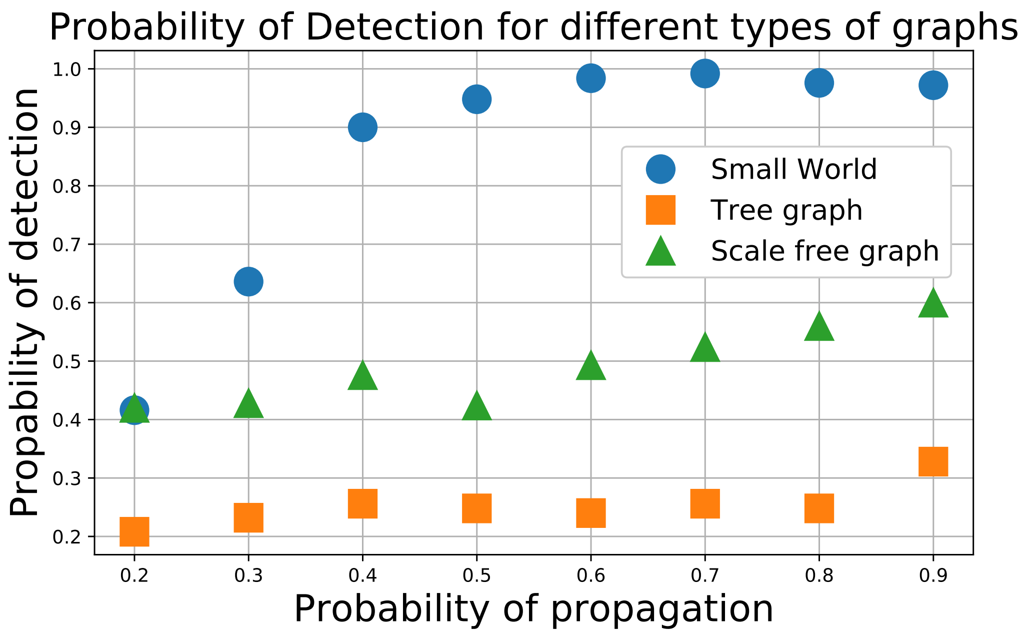

Figure 5.4: Probability of detection for different types of graphs

	Small World Graph		Balanced Tree Graph		Scale Free Graph	
p	avg. S rank	d from 1^{st} to S	avg. S rank	d from 1^{st} to S	avg. S rank	d from 1^{st} to S
0.2	8.29	1.345	16.208	2.204	1.78	0.928
0.3	1.265	0.605	10.04	2.032	1.868	0.84
0.4	0.19	0.27	6.844	1.928	1.52	0.812
0.5	0.055	0.105	7.38	2.0	1.72	0.824
0.6	0.007	0.105	6.628	2.028	1.248	0.748
0.7	0.005	0.075	5.428	2.024	1.172	0.776
0.8	0.002	0.069	5.808	2.044	1.036	0.732
0.9	0.002	0.053	5.168	1.98	1.04	0.7

Table 5.3: Average ranking of the real source S and distance from the first candidate to the source for different types of graphs

Overall

The detection algorithm performs significantly better on small world graphs than on the other types of graphs (around 30% better at least for $p \geq 0.4$). It is possible to achieve a probability of detection up to 0.6 for the scale free graph, however it requires a very high probability of propagation ($p \sim 0.9$). On tree graphs, the algorithm barely improved the detection of 10% from $p = 0.2$ to $p = 0.9$. Furthermore, it is not possible to achieve a correct detection more than 35% of the time on that type of graph.

Chapter 6

Conclusion

6.1 Overall remarks

This project aimed to provide a solution to the problem of detecting a source of rumour in a social network. This detection is based on the usage of a subset of nodes converted into monitoring nodes which report the steps of their infections by rumours. The key aspect of this problem is the usage of the time based information given by the monitoring nodes, due to several rumours propagating in the network, whereas the previous research papers focused on the knowledge of the full infected node subset, once the rumours had spread.

The proposed solution is based on two different usages of the information given by the monitoring nodes. First, using the earliest time of infection per monitoring node allows to draw conclusions regarding the maximum distance at which the source can be. Then, a subset of candidate nodes can be established. Secondly, the rumours time of arrival scattering allows to establish an experimental cumulative distribution of the rumours steps of arrival. By comparing this distribution to the theoretical one, the candidates can be ranked. The global detection algorithm combine these two technique to produce a global ranking of the candidates.

The detection algorithm have been tested on small world graphs, tree graphs and scale free graphs. It as also been tested in different conditions, with different numerbers of monitoring nodes and different probability of rumour propagation. Overall, the algorithm allows detection more than 80% of the time, provided that the probability of propagation is greater or equal to 0.4 and there are 10% of monitoring nodes. Under the same conditions, the probability that the source is in the firt five candidates is greater than 0.9. The ideal trade-off between feasability and performances regarding the number of monitoring nodes is arround 5%. The detection algorithm works better on small world graph than on the other types of networks.

6.2 Future Work

The simulation of the rumours propagation, either in this report or in the research papers, is made with the Susceptible-Infected model. If this simulation works well for epidemics or desease spreading, this is not the best modelization possible for rumour spreading. Using either triadic closures[3] or setting the threshold described in this report higher than 1 would model the fact that people might wait to hear the rumour from several different persons before stating to trust it and spread it.

The number of sources has been kept to one in this project. However, it might be interesting to perform some research for multiple sources spreading.

The detection algorithm assume that the start of the infection is precisely known, some effort should be put into the source detection when only a relative time of infection is known between the monitoring nodes.

The rumour propagation is discretized. A more realistic approach would consist in a continuous time spreading, based on a Poisson process.

Eventually, performing tests on real datasets instead of synthetic ones could allows to investigate the performances of the detection algorithm on real cases.

Bibliography

[1] Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. "Directed Scale-Free Graphs".

[2] E W Dijkstra. "A Note on Two Problems in Connexion with Graphs". *Numerische Mathematlk*, 271(1):269–271, 1959.

[3] D. Easley and J. Kleinberg. *"Networks, Crowds, and Markets: Reasonning about a Highly Connected World"*. Cambridge University Press, 2010.

[4] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the ACM*, 34(3):596–615, 1987.

[5] Adrien Guille, Hakim Hacid, Cecile Favre, and Djamel A. Zighed. "Information diffusion in online social networks". *ACM SIGMOD Record*, 42(1):17, 2013.

[6] Jonathan Hartley. Timecomplexity - python wiki, 2017.

[7] Wuqiong Luo, Wee Peng Tay, and Mei Leng. "Identifying Infection Sources and Regions in Large Networks". 2013.

[8] Pádraig MacCarron, Kimmo Kaski, and Robin Dunbar. "Calling Dunbar's Numbers". *arXiv preprint arXiv:1604.02400*, 2016.

[9] Krystian Mikolajczyk. Lecture notes in pattern recognition, 2016.

[10] S. Milgram. "The Small World Problem". *Psychology Today*, 1967.

[11] Eunsoo Seo, Prasant Mohapatra, and Tarek Abdelzaher. "Identifying rumors and their sources in social networks". pages 83891I–83891I–13, 2012.

[12] Devavrat Shah and Tauhid Zaman. "Finding Rumor Sources on Random Trees". 2011.

[13] Devavrat Shah and Tauhid Zaman. "Rumors in a network: Who's the culprit?" *IEEE Transactions on Information Theory*, 57(8):5163–5181, 2011.

[14] Devavrat Shah and Tauhid Zaman. "Rumor centrality". *ACM SIGMETRICS Performance Evaluation Review (ACM Digital Library)*, 40(1):199–210, 2012.

[15] D J Watts and S H Strogatz. "Collective dynamics of 'small-world' networks". *Nature*, 393(6684):440–442, 1998.

[16] Wayne W Zachary. "An Information Flow Model for Conflict and Fission in Small Groups". *Journal of Anthropological Research*, 33(4):452–473, 1977.

Appendices

Appendices

networkUtils.py

```
Author : Robin Dupont (robin.dpnt@gmail.com)
Imperial College Longon 2016/2017
MSc Communication and Signal Processing

port matplotlib.pyplot as plt
port networkx as nx
port random
port numpy as np
om scipy.special import comb


de_size_param = 40


f generateGraph(myNumNodes,myLinkProba,myGraphType):

  myGraphType = myGraphType%6

  myGraph = nx.karate_club_graph()

  if myGraphType == 1:
      # Small World
      myGraph = nx.watts_strogatz_graph(myNumNodes, 5, myLinkProba)
  if myGraphType == 2:
      # Tree
      myGraph = nx.balanced_tree(4,3)
  if myGraphType == 3:
      # Random Graph
      myGraph = nx.fast_gnp_random_graph(myNumNodes, 2*1/myNumNodes)
  if myGraphType == 4:
      # Random - power Law
      myGraph = nx.random_powerlaw_tree(myNumNodes)
  if myGraphType == 5:
      # Karate club
      myGraph = nx.karate_club_graph()
  if myGraphType == 6:
      # Scale Free Graph
      myGraph = nx.scale_free_graph(myNumNodes).to_undirected()

  return myGraph

```

```
Initiate the parameters of the graph
f initGraphParam(myG,myNumRum,myThreshMax):
  for i in range(myNumRum):
      nx.set_node_attributes(myG, 'infected'+str(i+1), False)   # Init infection stat
      nx.set_node_attributes(myG, 'counter'+str(i+1), 1)
      for j in myG.nodes():
          myG.node[j]['counter'+str(i+1)] = list()              # Generate infector

  nx.set_node_attributes(myG, 'threshold', 1)
  for i in myG.nodes():
      myG.node[i]['threshold'] = random.randint(1,myThreshMax)    # Init threshold
  return myG

Initiate the source of the rumours
f initSourceNode(myG,myNumRum):
  nodeList = myG.nodes()
  sources = []
  index = random.choice(nodeList)
  for i in range(myNumRum):
      myG.node[index]['infected'+str(i+1)] = 1
      sources.append(index)
  sources = list(set(sources))
  return myG, sources


f initMonitoringNodes(myG,myMonitorNum,mySources,myNumRum):
  nodeList = myG.nodes()
  monitorList = list()

  for i in range(myMonitorNum):
      monitorList.append(random.choice(nodeList))

  while len(list(set(monitorList))) != myMonitorNum and len(set(mySources).intersecti
      monitorList = list()
      for i in range(myMonitorNum):
          monitorList.append(random.choice(nodeList))
  for i in  monitorList:
      for k in range(myNumRum):
          myG.node[i]['detected'+str(k+1)] = False

  return myG,monitorList


  return myG

f colorList(myG,myNumRum):
  infected=[]
  notInfected=[]
  for j in range(myNumRum):
      for i in myG.nodes():
          if myG.node[i]['infected'+str(j+1)]:
              infected.append(i)
          else :
              notInfected.append(i)
  return infected, notInfected


f drawColoredGraph(myG,myPos,myNumRum,mySources,myMonitors=None,myIndex=None):
  infect, notInfect = colorList(myG,myNumRum)
  if myMonitors != None:
      nx.draw_networkx_nodes(myG, myPos, myMonitors, node_color='c',
                              node_size=25*node_size_param,label='Monitoring Node')
  nx.draw_networkx_nodes(myG, myPos, infect, node_color='r', node_size=15*node_size_pa
  nx.draw_networkx_nodes(myG, myPos, notInfect, node_color='#2ca02c', node_size=5*node
  nx.draw_networkx_nodes(myG, myPos, mySources, node_color='#1f77b4',
                         node_size=5*node_size_param,label='Rumour Source')
  nx.draw_networkx_edges(myG, myPos, width=2.0,alpha=0.3)
  plt.legend(scatterpoints=1,fontsize=20,loc=(0,0))
  plt.axis('off')
  if myIndex != None:
      plt.savefig('reportFigs/figT' + str(myIndex) + '.eps')
      plt.savefig('reportFigs/figT' + str(myIndex) + '.png')

  plt.show()
  #return myFig
  return None


f drawColoredGraph2(myG,myPos,myNumRum,mySources,myMonitors,myDetected):
  infect, notInfect = colorList(myG,myNumRum)
  labs = {}
  for node in myG.nodes():
      if node in myDetected:
          labs[node]=node
  nx.draw_networkx_nodes(myG, myPos, infect, node_color='r', node_size=15*node_size_pa
  nx.draw_networkx_nodes(myG, myPos, notInfect, node_color='g', node_size=5*node_size_
  nx.draw_networkx_nodes(myG, myPos, myMonitors, node_color='c', node_size=40*node_siz
  nx.draw_networkx_nodes(myG, myPos, myDetected, node_color='y', node_size=50*node_siz
  nx.draw_networkx_nodes(myG, myPos, mySources, node_color='b', node_size=5*node_size_
  nx.draw_networkx_edges(myG, myPos, width=1.0,alpha=0.1)
  nx.draw_networkx_labels(myG, pos=myPos,labels=labs)
  plt.axis('off')
  plt.savefig('./TestFigs/figEnd.png')
  plt.show()
```

```
    return None





f infectionForward(myG, myProba, myNumRum):
    myG2 = myG

    for k in range(myNumRum):
        for i in myG.nodes():
            if myG.node[i]['infected'+str(k+1)]:
                neiList = myG.neighbors(i)
                for j in neiList:
                    if random.random() < myProba:
                        myG2.node[j]['counter'+str(k+1)].append(i)
                        myG2.node[j]['counter' + str(k + 1)] = list(set(myG2.node[j]['c
        for l in myG2.nodes():
            if len(myG2.node[l]['counter'+str(k+1)]) >= myG2.node[l]['threshold']:
                myG2.node[l]['infected' + str(k + 1)] = True

    return myG2


f infectionForwardSingle(myG, myProba, myNumRum):
    myG2 = myG

    for k in range(myNumRum):
        for i in myG.nodes():
            if myG.node[i]['infected'+str(k+1)]:
                neiList = myG.neighbors(i)
                j = random.choice(neiList)
                if random.random() < myProba:
                    myG2.node[j]['counter'+str(k+1)].append(i)
                    myG2.node[j]['counter' + str(k + 1)] = list(set(myG2.node[j]['count
        for l in myG2.nodes():
            if len(myG2.node[l]['counter'+str(k+1)]) >= myG2.node[l]['threshold']:
                myG2.node[l]['infected' + str(k + 1)] = True

    return myG2


f getInfectedList (myG,myNumRum):

    infected = []
    for j in range(myNumRum):
        for i in myG.nodes():
            if myG.node[i]['infected'+str(j+1)]:
                infected.append(i)
    infected = list(set(infected))

    return infected

  f isAllInfected(myG,myNumRum):
    myBool = True
    for j in range(myNumRum):
        for i in myG.nodes():
            if not myG.node[i]['infected'+str(j+1)]:
                myBool = False
    return myBool

  f estimateInfected(myG,myNumRum,myRumId):
    counter = 0

    for i in myG.nodes():
        if myG.node[i]['infected'+str(myRumId)]:
            counter +=1

    return counter


  f findNeighDegN(myG, mySource, myDeg):
    path_lengths = nx.single_source_dijkstra_path_length(myG, mySource)
    neigh = [node for node, length in path_lengths.items() if length == myDeg]
    return  neigh




  f findPossibleSets(myG,mySources,myTrig,myNumRum):
    possibleSets = []
    for s in myTrig:
        curDeg = s[2]
        while curDeg > 1 :
            curSet = findNeighDegN(myG,s[0],curDeg)
            curDeg -= 1
        possibleSets.append(curSet)
    print(possibleSets)


    initSet = set(possibleSets[0])
    for i in range(len(possibleSets)-1):
        initSet = initSet.intersection(possibleSets[i+1])
    return list(initSet)
```

```
f findPossibleSets2(myG,mySources,myTrig,myNumRum):
  possibleSets = []
  for s in myTrig:
      curDeg =s[2]
      while curDeg>1:
          curSet = findNeighDegN(myG,s[0],curDeg)
          curDeg-=1
          if len(curSet)>0:
              possibleSets.append(curSet)
  print(possibleSets)

  array = np.zeros((len(myG.nodes()),1))
  for i in possibleSets :
      for j in i:
          array[j]+=1
  arrayAsList = array.tolist()
  maxVal = max(arrayAsList)
  print(array)
  print('Source : ',mySources[0],' Max : ',max(arrayAsList), ' value : ',arrayAsList[m
  posList = list()
  for i in range(len(array)):
      if maxVal == array[i]:
          posList.append(i)
  return posList


f findSet2(myG, myCenterNode, myMaxDeg):
  # Based on set intersection (not circle)
  curSet = set()
  curDeg = myMaxDeg
  while (curDeg > 0):
      tempoSet = set(findNeighDegN(myG, myCenterNode, curDeg))
      curSet = curSet.union(tempoSet)
      curDeg -= 1
  return curSet


f calculProba(myPropagProba, myDist, mySteps):
  # Compute the probability of a node at k steps of the source  being infected by s
  p = myPropagProba
  k = myDist
  n = mySteps
  result = 0
  for i in range(0, n - k + 1):
      result += comb(k + i - 1, i, exact=True) * pow(p, k) * pow(1 - p, i)
  return result
```

simulation.py

```
Author : Robin Dupont (robin.dpnt@gmail.com)
Imperial College Longon 2016/2017
MSc Communication and Signal Processing

om networkUtils import *


f generateGraphReady(myNumNodes,myLinkProba,myMaxThreshold,myNumRumors,myNumMonitors,
  # mGraph generation
  print("Starting Simulation")
  mGraph = generateGraph(myNumNodes, myLinkProba, 1)
  # Getting layout
  pos = nx.spring_layout(mGraph)
  # Setting default attributes
  mGraph = initGraphParam(mGraph, myNumRumors, myMaxThreshold)
  # Choosing the source
  mGraph, myRumorSources = initSourceNode(mGraph, myNumRumors)
  # Choosing the monitoring nodes
  mGraph, myMonitorsList = initMonitoringNodes(mGraph, myNumMonitors, myRumorSources,
  return mGraph, pos, myRumorSources, myMonitorsList

f updateMonitorTrig(curStep,myMonitorTrigger,myMonitorList,myG,myNumRum):
  for k in myMonitorList:
      for l in range(myNumRum):
          if myG.node[k]['infected' + str(l + 1)] and not myG.node[k]['detected' + st
              myMonitorTrigger.append((k, l + 1, curStep))
              myG.node[k]['detected' + str(l + 1)] = True
              # Sort by monitoring node and by rumour index
      myMonitorList = sorted(myMonitorTrigger, key=lambda x: (x[0], x[1]))
  return myMonitorList


f printMonitorTrig(myMonitorTrig):
  print('\nMonitoring Nodes :')
  prev = 0
  for i in myMonitorTrig:
      if prev == 0 or i[0] != prev:
          print('Monitoring node number : ', i[0], '\n\tinfected by rumor : ', i[1],
          prev = i[0]
      else:
          print('\tinfected by rumor : ', i[1], '\tat step : ', i[2])
          prev = i[0]
  return None

f findSet(myG,mySourceNode,myRadius):
  curSet = findNeighDegN(myG,mySourceNode,myRadius)
  return  curSet



 __name__ == '__main__':

  # Parameters definition
  numRumors = 4
  maxThreshold = 1
  numMonitors = 10
  propagProba = 1
  numNodes = 50
  linkProba = 0.2
  monitorTrigger = list()
  numStep = 100


  Graph, Pos, rumorSources, monitorsList = generateGraphReady(numNodes,linkProba,maxTh
  drawColoredGraph(Graph, Pos, numRumors, rumorSources, monitorsList)
  print("Starting infection")
  infections = [[] for n in range(numRumors)]

  for j in range(5):
      print("========================================\n\n\n\n\n\n\n\n================
      for i in Graph.nodes():
          if i in monitorsList:
              print("Node ",str(i),"\t",Graph.node[i])
      Graph = infectionForward(Graph, propagProba, numRumors)
      monitorTrigger = updateMonitorTrig(j,monitorTrigger,monitorsList,Graph,numRumors
      print(monitorTrigger)
      printMonitorTrig(monitorTrigger)
      drawColoredGraph(Graph, Pos, numRumors, rumorSources, monitorsList)
  plt.show()
```

detectionUtils.py

```
Author : Robin Dupont (robin.dpnt@gmail.com)
Imperial College Longon 2016/2017
MSc Communication and Signal Processing
```

```

om networkUtils import *


f chiDist(histo1, histo2):
  PmQ2 = np.power(histo1 - histo2, 2)

  # prevent 0 values in division
  histo1[histo1 == 0] = np.finfo(float).eps
  histo2[histo2 == 0] = np.finfo(float).eps
  PpQ = histo1 + histo2
  vectRes = np.divide(PmQ2, PpQ)
  res = np.sum(vectRes)
  return np.sqrt(res)


f createHistoForMonitor(monitorTrigger2,monitorsList,numRumors):
  ################ STEP 2
  # Create the histogram of rumor reception step for each monitoring node
  # Find the maximum number of steps :
  maxStep = -1
  for step in monitorTrigger2:
      if step[2] > maxStep:
          maxStep = step[2]

  # create array for each monitoring node
  HistoDict = dict()

  # Fill in the array
  for monitor in monitorsList:
      HistoDict[monitor] = np.zeros((maxStep + 1, 1))
      for elem in monitorTrigger2:
          if monitor == elem[0]:
              HistoDict[monitor][elem[2]] += 1
      HistoDict[monitor] = np.cumsum(HistoDict[monitor]) / numRumors

  return HistoDict,maxStep



f findAllPossibleCandidates(monitorTrigger,Graph):
  ################ STEP 3
  # Find all possible candidates based on set intersections
  setList = []

  for i in monitorTrigger:
      nodeSet = findSet2(Graph, i[0], i[2])
      setList.append(nodeSet)

  finalSet = set.intersection(*setList)
  finalList = list(finalSet)

  return finalList


f dictOfHistoForPossibleSourcesPerMonitor(monitorsList,finalList,maxStep,propagProba,(
  ################ STEP 4
  # Create the histogram for each monitoring node, for each possible source
  # i.e. Step 1: the whole graph, Step 2: only the possible sources determined with
  DictOfPossibleHistPerMonitor = {}
  for monitor in monitorsList:
      sourceHisto = {}
      monitorToTest = monitor
      for source in finalList:
          if source not in monitorsList:
              sourceHisto[source] = np.zeros((maxStep + 1, 1))
              for i in range(0, maxStep + 1):
                  sourceHisto[source][i] = calculProba(propagProba,
                                                       len(nx.shortest_path(Graph, sou
      DictOfPossibleHistPerMonitor[monitor] = sourceHisto

  return DictOfPossibleHistPerMonitor


f computeScores(finalList,monitorsList,DictOfPossibleHistPerMonitor,HistoDict,rumorSou

  scoreDictL2 = {}
  scoreDictChi2 = {}
  for i in finalList:
      scoreDictL2[i] = 0
      scoreDictChi2[i] = 0

  for monitorToTest in monitorsList:
      monitorRankingL2 = []
      monitorRankingChi2 = []

      for source in finalList:

          if source not in monitorsList:

              d = np.linalg.norm(DictOfPossibleHistPerMonitor[monitorToTest][source]
              d2 = chiDist(DictOfPossibleHistPerMonitor[monitorToTest][source],
                           np.transpose(HistoDict[monitorToTest][np.newaxis]))
              monitorRankingL2.append((source,d))
```

```
                monitorRankingChi2.append((source,d2))

        monitorRankingL2 = sorted(monitorRankingL2, key=lambda x: x[1])
        monitorRankingChi2 = sorted(monitorRankingChi2, key=lambda x: x[1])

        monitorRankingL2 = [ elem[0] for elem in monitorRankingL2]
        monitorRankingChi2 = [ elem[0] for elem in monitorRankingChi2]

        for i in finalList :
            if i not in monitorsList:
                scoreDictL2[i] += monitorRankingL2.index(i)
                scoreDictChi2[i] += monitorRankingChi2.index(i)

    scoreListL2 = sorted( scoreDictL2.items(),key=lambda x: x[1])
    scoreListChi2 = sorted( scoreDictChi2.items(),key=lambda x: x[1])
    scoreListL2 = [ elem[0] for elem in scoreListL2]
    scoreListChi2 = [elem[0] for elem in scoreListChi2]


    try:
        if rumorSources[0] in scoreListL2 :
            return scoreListL2.index(rumorSources[0]),scoreListChi2.index(rumorSources[(
        else :
            return len(finalList)-1,len(finalList)-1,len(finalList),scoreListL2[0]
    # Catching the error
    except :
        print("ERROR ###################")
        print(len(scoreListL2))
        if rumorSources[0] in scoreListL2 :
            return scoreListL2.index(rumorSources[0]),scoreListChi2.index(rumorSources[(
        else :
            return len(finalList)-1,len(finalList)-1,len(finalList),max(rumorSources[0]-
```

benchmark_histo.py

```
Author : Robin Dupont (robin.dpnt@gmail.com)
Imperial College Longon 2016/2017
MSc Communication and Signal Processing

om simulation import *
om detectionUtils import *


f runHistoSimulation(myI, myProba) :
    ################ STEP 1
```

```
     # Generate an graph with rumor spread

 # Parameters definition
 numRumors = 20
 maxThreshold = 1
 numMonitors = 20
 propagProba = myProba
 numNodes = 200
 linkProba = 0.3
 monitorTrigger = list()
 numStep = 100

 j = 0

 Graph, Pos, rumorSources, monitorsList=generateGraphReady(numNodes, linkProba, maxT

 while (not isAllInfected(Graph, numRumors)):
     Graph = infectionForward(Graph, propagProba, numRumors)
     # j+1 because j=0 is step 1
     monitorTrigger = updateMonitorTrig(j + 1, monitorTrigger, monitorsList, Graph, 
     j += 1

 monitorTrigger2 = sorted(monitorTrigger, key=lambda x: (x[0], x[2]))
 HistoDict,maxStep = createHistoForMonitor(monitorTrigger2,monitorsList,numRumors)
 finalList = findAllPossibleCandidates(monitorTrigger,Graph)
 DictOfPossibleHistPerMonitor = dictOfHistoForPossibleSourcesPerMonitor(monitorsList
                                                                        maxStep, pro
 scoreL2, scoreChi2, numCandidat, bestCandidat = computeScores(finalList, monitorsLi
                                                               DictOfPossibleHistPer
 distToSource = len(nx.shortest_path(Graph, source=rumorSources[0], target=bestCandi

 print("Simulation ",myI, "done")

 return scoreL2, scoreChi2, Graph, distToSource


__name__ == '__main__':
 runHistoSimulation(1, 0.2)
```

Bench_Parallel.py

```
Author : Robin Dupont (robin.dpnt@gmail.com)
Imperial College Longon 2016/2017
MSc Communication and Signal Processing

```

```
om benchmark_histo import *
om joblib import Parallel, delayed
port multiprocessing
port os


 os.name == 'nt' :
  from win10toast import ToastNotifier

 __name__ == '__main__':
  if os.name == 'nt':
      toaster = ToastNotifier()

  probas = [0.2,0.3,0.4,0.5,0.6,0.7,0.9]

  num_cores = multiprocessing.cpu_count()

  print("Starting benchmarking on ",num_cores," cores")

  for proba in probas:

      inputs = range(0,3)

      results = Parallel(n_jobs=num_cores-1)(delayed(runHistoSimulation)(i,proba) for

      scoreL2 = [v[0] for v in results]
      scoreChi2 = [v[1] for v in results]
      Graphs  = [v[2] for v in results]
      distsToSource = [v[3] for v in results]

      print("\nGlobal Score for L2 : ",np.mean(scoreL2))
      print("Global Score for Chi2 : ",np.mean(scoreChi2))
      print("dists : ", distsToSource)

      # Compute proba of detection First one
      probaDetectL2 = []
      probaDetectChi2 = []
      # Compute proba of detection first 5
      probaDetect5L2 = []
      # Compute proba of detection first 10
      probaDetect10L2 =[]


      for score in scoreL2 :
          if score == 0 :
              probaDetectL2.append(1)
          else:
              probaDetectL2.append(0)
          if score <= 4:
              probaDetect5L2.append(1)
          else:
              probaDetect5L2.append(0)
          if score <=9:
              probaDetect10L2.append(1)
          else:
              probaDetect10L2.append(0)

      moyProba = np.mean(probaDetectL2)
      moyProba5 = np.mean(probaDetect5L2)
      moyProba10 = np.mean(probaDetect10L2)


      for score in scoreChi2:
          if score == 0:
              probaDetectChi2.append(1)
          else:
              probaDetectChi2.append(0)

      moyProbaChi2 = np.mean(probaDetectChi2)


      moyDist = np.mean(distsToSource)

      print("All proba L2", probaDetectL2)
      print("Global Proba L2: ",moyProba)
      print("Global Proba Chi2: ",moyProbaChi2)
      print(moyDist)
      with open('results.txt','a') as f:
          f.write("Proba = "+str(proba)+"\nGlobal Score for L2 : " + str(np.mean(score
                  "\nGlobal Score for Chi2 : "+ str(np.mean(scoreChi2))+ "\nProba Dete
                  str(moyProba)+ "\nProba Bo5 L2 " + str(moyProba5)+ "\nProba Bo10 L2
                  "\n Mean min dist to real source : " + str(moyDist) +"\n")

  if os.name == 'nt':
      toaster.show_toast("Simulation Over",
                     "P: "+str(proba)+" L2: "+str(np.mean(scoreL2))+" Chi2: "+str(np.m
                     duration=60)
```