TEMPORAL CLOSENESS IN KNOWLEDGE MOBILIZATION NETWORKS

by
William Doan

A thesis submitted to
the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of
the requirements for the degree of

MASTER OF COMPUTER SCIENCE

School of Electrical Engineering and Computer Science

at

UNIVERSITY OF OTTAWA

Abstract

In this thesis we study the impact of time in the analysis of social networks. To do that we represent a knowledge mobilization network, *Knowledge-Net*, both as a standard static graph and a time-varying graph and study both graphs to see their differences. For our study, we implemented some temporal metrics and added them to Gephi, an open source software for graph and network analysis which already contains some static metrics. Then we used that software to obtain our results.

Knowledge-Net is a network built using the knowledge mobilization concept. In social science, knowledge mobilization is defined as the use of knowledge towards the achievement of goals. The networks which are built using the knowledge mobilization concept make more visible the relations among heterogeneous human and non-human individuals, organizational actors and non-human mobilization actors.

A time-varying graph is a graph with nodes and edges appearing and disappearing over time. A journey in a time-varying graph is equivalent to a path in a static graph. The notion of shortest path in a static graph has three variations in a time-varying graph: the shortest journey is the journey with the least number of temporal hops, the fastest journey is the journey that takes the least amount of time and the foremost journey is the journey that arrives the soonest. Out of those three, we focus on the foremost journey for our analysis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A social network is a social structure made up of a set of social actors (such as individuals or organizations) and a set of one-to-one ties representing social interactions between actors.

Social network analysis (SNA) is the process of investigating social structures through the use of network and graph theories.

To analyze a social network, we must first convert it to a graph. This is a straightforward process: an actor in a social network is represented by a node in its graph representation, and a tie in a social network is represented by an edge in its graph representation. After the conversion, we can run algorithms often developed for graph theory to analyze the network. This is a classical and widely used method.

In recent years, people started to think about the idea of a temporal graph: a graph with nodes and edges appearing and disappearing during its lifetime. Different formal definitions along with different names were given to temporal graphs, but the base idea is always the same: a graph that changes over time. With the apparition of temporal graphs, temporal metrics were created to analyze them.

Looking back at social networks, people realized that it would be more accurate to represent them using temporal graphs instead of the classical static graphs since a social network often changes during its lifetime, with new actors adding themselves to the network and some actors or ties disappearing. Moreover, the temporal metrics developed could now be applied to real world networks. Thus was born the temporal analysis of social networks.

1.1 Motivations and Objectives

Social networks evolve in time but are often described with a single graph that contains the aggregation of all the temporal connections. Most of the existing work in the field, in

fact, focuses on static representations of social networks.

Recently, there has been more and more interest in incorporating temporal aspects into the analysis of social networks (some recent work is described in Chapter 2) but this area is still largely unexplored.

The main objective of the thesis is to contribute to this study by providing a temporal analysis of a knowledge mobilization network, called *Knowledge-Net*. This network has been already the object of temporal investigation in [2], where some temporal centrality measures have been investigated, comparing the results with the ones obtainable by static analysis. The goal of the thesis is to study Knowledge-Net focusing instead on *temporal closeness*: a measure that indicates the level of "reachability" of the various nodes in the network.

1.2 Contributions

The main contribution of the thesis is the analysis of temporal closeness in Knowledge-Net. In fact, we introduce several definitions of temporal closeness and we compute all of them to compare the results with their static counterpart.

More precisely:

- We introduce different variations of temporal closeness. The first one is the direct temporal adaptation of the definition of static closeness. The second variation solves the problem introduced by the disconnections encountered in a time-varying graph. The third variation solves the problem of the advantage a node can gain from its birthdate in a time-varying graph.

- We devise algorithms to compute the various concepts of closeness in a temporal setting and we include our final protocol into *Gephi*, an open source tool that, up to this point, was providing only static analysis of social networks.

- We focus on a knowledge mobilization network created in a research environment, which describes the relationship among researchers, their projects, their publications and their students. We analyze static and temporal closeness of the actors of this

network and we draw our conclusions regarding the importance of time in the study of this network.

1.3 Overview of the Thesis

In Chapter 2, we present some work done on time-varying and temporal graphs. First, we talk about the work that has been done on dynamic communication networks in general. Then we narrow it down to what has been done on social networks represented as temporal graphs. Finally, we present the temporal analysis done on a knowledge mobilization network.

In Chapter 3, we give the definition of a time-varying graph. We then explain the notion of "journeys" and "distances" in a time-varying graph. Then we show how we got the temporal definition of closeness from its static definition. Finally, we present all the variations of the temporal closeness that we made and explain how they can make the analysis more relevant.

In Chapter 4, we talk about the software Gephi and foremost journeys. We first give a general view of the different components of Gephi's interface. Then we present how to compute foremost journeys and explain our implementation. After that, we list all the algorithms that we added to Gephi along with explaining how to integrate an algorithm to Gephi. Finally, we show how to import a graph and run an algorithm in Gephi.

In Chapter 5, we describe the setup of our experiments. We begin by presenting Knowledge-Net, the knowledge mobilization network that we are studying. Then we explain all the variations of Knowledge-Net that we used for our analysis.

In Chapter 6, we show and explain the results of our analysis of all the variations of Knowledge-Net.

In Chapter 7, we conclude the thesis and give some open problems.

Chapter 2

Related Work

In recent years dynamic graphs have been studied extensively in a variety of different contexts, from social networks, to transportation networks, to computer networks. Most of the existing work is concerned with communication networks in situations where nodes and/or edges can appear and disappear in time (e.g., [7, 18, 21–23, 27]). Recently, some authors have also studied dynamic graphs in the context of social networks (e.g., [2, 3, 20, 26, 28, 31, 33]). In the following, we give a brief overview of the recent work, focusing on work that is particularly relevant to the thesis.

2.1 Dynamic Communication Networks

In [3], evolving graphs (a type of dynamic graph) are used to compute multicast trees with minimum overall transmission time for a class of wireless mobile dynamic networks. The authors show that computing different types of strongly connected components in evolving digraphs is NP-Complete, and then propose an algorithm to build all rooted directed minimum spanning trees in strongly connected dynamic networks.

In [5], the problem of broadcasting with termination detection is studied for time-varying graphs with edges that appear infinitely often but without any known pattern. This is done with respect to three possible metrics: the date of message arrival (foremost), the time spent doing the broadcast (fastest), and the number of hops used by the broadcast (shortest).

In [6], a tool called T-CLOCKS is presented. It is based on a distributed algorithm and allows each node in a delay-tolerant network (a network with a possible absence of end-to-end communication routes at any instant) to track in real-time how "out-of-date" it is with respect to every other. The authors address the case where contacts can have arbitrary durations. The problem is further complicated by the fact that they address continuous-time systems and non-negligible message latencies (time to propagate a single

message over a single link), however this latency is assumed fixed and known.

In [9], stochastic time-dependency in evolving graphs is introduced: starting from an arbitrary initial edge probability distribution, at every time step, every edge changes its state (existing or not) according to a two-state Markovian process with probabilities p (edge birth-rate) and q (edge death-rate). If an edge exists at time t then, at time $t+1$, it dies with probability q. If instead the edge does not exist at time t, then it will come into existence at time $t+1$ with probability p. The speed of information dissemination is investigated in such dynamic graphs.

In [10], the computability and complexity of the exploration problem are studied in a class of highly dynamic graphs: periodically varying (PV) graphs, where the edges exist only at some (unknown) times defined by the periodic movements of carriers.

In [14], a formal classification of dynamic graphs is developed. The authors discuss areas where dynamic graphs arise in computer science such as compilers, databases, fault-tolerance, artificial intelligence, and computer networks. Finally, they propose approaches that can be used for studying dynamic graphs.

In [15], analytical tools are used to derive generic theoretical upper bounds for the information propagation speed in large scale mobile and intermittently connected networks. Then the authors show how their analysis can be applied to specific mobility and graph models to obtain specific analytical estimates.

In [16], a delay-tolerant networking routing problem is formulated. The messages are to be moved end-to-end across a connectivity graph that is time-varying but whose dynamics may be known in advance. The problem has the added constraints of finite buffers at each node and the general property that no contemporaneous end-to-end path may ever exist. The authors then develop several algorithms and use simulations to compare their performance with respect to the amount of knowledge they require about network topology.

In [17], a practical routing protocol for delay-tolerant networks is presented. It only uses observed information about the network. The authors then demonstrate through simulation that their protocol provides performance similar to that of schemes that have global knowledge of the network topology, yet without requiring that knowledge.

In [18], results on two types of problems for temporal networks are provided. First, the authors consider connectivity problems, in which they seek disjoint time-respecting paths between pairs of nodes. They then define and study the class of inference problems, in which they seek to reconstruct a partially specified time labeling of a network in a manner consistent with an observed history of information flow.

In [19], a realistic large scale global delay-tolerant network is studied. The authors explore how messages could be carried between airports based upon scheduled flight connections. They investigate the interaction with different routing protocols, the impact of scheduling uncertainties, and the limiting factors by means of simulations and analysis.

In [22], distributed computation in dynamic networks in which the network topology changes from round to round is investigated. The authors consider a worst-case model in which the communication links for each round are chosen by an adversary, and nodes do not know who their neighbors for the current round are before they broadcast their messages. The model captures mobile networks and wireless networks, in which mobility and interference render communication unpredictable.

In [23], several variants of coordinated consensus in dynamic networks are studied. The authors assume a synchronous model, where the communication graph for each round is chosen by a worst-case adversary. The network topology is always connected, but can change completely from one round to the next. The model captures mobile and wireless networks, where communication can be unpredictable.

In [24], PROPHET, a probabilistic routing protocol for intermittently connected networks, is proposed. In intermittently connected networks there is no guarantee that a fully connected path between source and destination exists at any time, rendering traditional routing protocols unable to deliver messages between hosts.

In [25], an algorithm called DTN Hierarchical Routing (DHR) is proposed. DHR is a routing algorithm for delay-tolerant networks with repetitive mobility which routes on contact information compressed by three combined methods. The authors then use analytical studies and simulation results to show that the performance of their proposed routing algorithm is comparable to that of the optimal time-space Dijkstra algorithm in terms of delay and hop-count.

In [29], a novel framework for the study of dynamic mobility networks is proposed. The authors address the characterization of dynamics by proposing an in-depth description and analysis of two real-world data sets. They show in particular that links creation and deletion processes are independent of other graph properties and that such networks exhibit a large number of possible configurations, from sparse to dense. Then they propose some accurate models that allow to generate random mobility graphs with a temporal behavior similar to the one observed in the experimental data.

In [30], a new routing scheme called Spray and Wait is introduced. It "sprays" a number of copies into the network, and then "waits" until one of these copies meets the destination.

2.2 Social Networks

In [13], a class of models for social networks where the interactions are transient is introduced using evolving graphs with memory dependent edges, which may appear and disappear according to their recent history. In particular the authors show that such networks may continue evolving forever, or else may quench and become static (containing immortal and/or extinct edges). This depends on the existence or otherwise of certain infinite products and series involving age dependent model parameters.

In [21], a number of metrics that can be used to study and explore temporal graphs are presented. The authors then use temporal graphs to analyze real-world data and present the results of their analysis.

In [32], a temporal small world is defined as a time-varying graph in which the links are highly clustered in time, yet the nodes are at small average temporal distances. The small-world behavior is explored in synthetic time-varying networks of mobile agents and in real social and biological time-varying systems.

In the following we describe in more detail some of the recent work in the context of social networks.

2.2.1 Temporal Distance Metrics for Social Network Analysis

In [31] a temporal graph is defined as a graph that can change in time, with nodes and edges appearing and disappearing. It is represented by a sequence of snapshots that show the graph at different time intervals. Formally, a temporal graph $\mathcal{G}_t^w(t_{min}, t_{max})$ with N nodes consist of a sequence of graphs $G_{t_{min}}$, $G_{t_{min}+w}$, ..., $G_{t_{max}}$, where w is the size of each window in some time unit (for example, in seconds). Each G_t consists of a set of nodes N and a set of edges E, such that $i, j \in V$ if and only if there exists a contact between the node i and the node j at time s, noted R_{ij}^S, with $t \leq s \leq t + w$.

Given two nodes i and j, a temporal path $p_{ij}^h(t_{min}, t_{max})$ is a set of paths staring from i and finishing at j that pass through the nodes $n_1^t ... \ n_i^t$, where $t_{min} \leq t \leq t_{max}$ is the time window that the node n is visited and h is the maximum hop within the same window t.

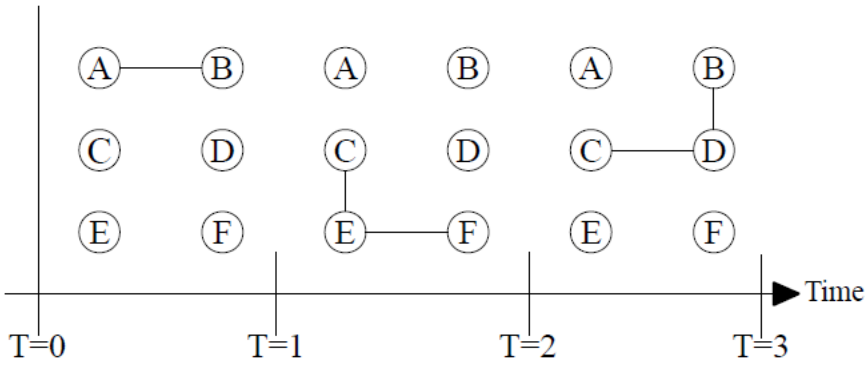
Given two nodes i and j, the shortest temporal distance $d_{ij}^h(t_{min}, t_{max})$ is the shortest temporal path length. Starting from the time t_{min}, it is the path that can connect i to j with the least number of time windows (or temporal hops). The horizon h indicates the maximum number of nodes within each window G_t which information can be exchanged.

Then an algorithm to compute $d_{ij}^h(t_{min}, t_{max})$ is given. The algorithm is based on depth first search and gives for a node i the shortest temporal distance to all the other nodes of the graph. Here is the algorithm of the authors:

"The algorithm assumes global knowledge of the temporal graph and keeps track of two global lists, D and R, indexed by node identifier. D keeps track of the number of temporal hops to reach a node and R keeps track of nodes that are reached. We initialise the value of every nodes of D to 1 and R to $False$. Starting with the first time window, we check that the source node i has been sighted. If so, we perform a depth first search (DFS) to see if any unreached nodes have a path to a node that was reached in a previous window. The maximum depth of DFS is dictated by the horizon h and if there are more than one path we choose the shortest. If a node j is reachable then we set $R[j] = True$ otherwise we increment the distance $D[j]$. If the source node i is not reachable then we increment all $D[j]$ since we cannot establish a transitively connected path from the source. We then repeat this for the next window."

Below is an example to show how the algorithm above works. Consider the temporal graph represented as a series of snapshots of Figure 2.1:

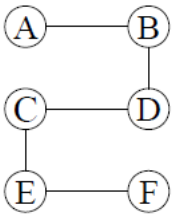
Figure 2.1: Example Temporal Graph, $G_t(0,3)$, $h = 2$ and $w = 1$



From Figure 2.1, we have $\mathcal{G}_t(0,3)$ and $w = 1$. Let us suppose $h = 2$ for this example.

Before starting the example, here is an interesting thing to look at. If we combine all the snapshots into one static graph, we would obtain Figure 2.2:

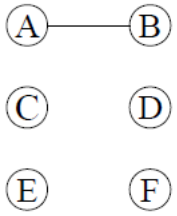
Figure 2.2: Example static graph based on the temporal graph in Figure 2.1



In this static graph, the node A can reach the node F by going through the nodes B, D, C and E, and the node F can reach the node A by going the reverse way. This suggests that the paths are symmetric. But looking at the temporal graph, we can see that this is not the case and that the static graph incorrectly shows that information can spread between the node A and the node F.

Now let us see how the algorithm computes the shortest temporal distance from the node A to all the other nodes of the temporal graph. At time $t = 1$, it is checked if the node A is in the time window (or snapshot). Since it is there, $R[A]$ is set to $True$. Then every other nodes of the time window is checked for reachability (by performing DFS). Since there is a path between A and B, and since A has been reached ($R[A] = True$), $R[B]$ is set to $True$. Nodes C, D, E and F are not connected to any other node, so their values for D are incremented. That is Figure 2.3:

Figure 2.3: Distance and Reachability of Window 1



Initialization

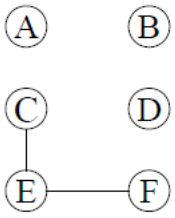
	A	B	C	D	E	F
D	1	1	1	1	1	1
R	F	F	F	F	F	F

Variables

	A	B	C	D	E	F
D	1	1	2	2	2	2
R	T	T	F	F	F	F

At time $t = 2$, all unreached nodes (C, D, E and F) are checked to see if they can be reached by already reached nodes (A and B). There are some connections between the unreached nodes, but no connection between them and either A or B, so their values D are incremented again. We then have Figure 2.4:

Figure 2.4: Distance and Reachability of Window 2

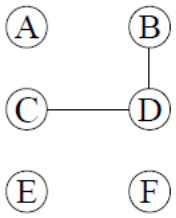


Variables

	A	B	C	D	E	F
D	1	1	3	3	3	3
R	T	T	F	F	F	F

Finally, at time $t = 3$, all unreached nodes (C, D, E and F) are checked again to see if they can be reached by already reached nodes (A and B). Since the node C can reach the node B by going through the node D (this is valid since $h = 2$), $R[C]$ is set to $True$. $R[D]$ is also set to $True$ since the node D can reach the node B. The nodes E and F still can not reach either A or B, so their values D are increased again. We then have Figure 2.5:

Figure 2.5: Distance and Reachability of Window 3



Final

	A	B	C	D	E	F
D	1	1	3	3	4	4
R	T	T	T	T	F	F

At the end, we have $d_{AB} = 1$, $d_{AC} = 3$ and $d_{AD} = 3$. Since $R[E] = False$ and $R[F] = False$, $d_{AE} = \infty$ and $d_{AF} = \infty$.

Some global temporal metrics are then defined:

The temporal efficiency $E_{T_{ij}}$ between the nodes i and j, and from the time interval t_{min} to the time interval t_{max} is defined as:

$$E^h_{T_{ij}}(t_{min}, t_{max}) = \frac{1}{d^h_{ij}(t_{min}, t_{max})}$$

The shortest temporal path length L^h and temporal global efficiency E^h_{glob} for a temporal graph are defined as:

$$L^h(t_{min}, t_{max}) = \frac{1}{N(N-1)} \sum_{ij} d^h_{ij}(t_{min}, t_{max})$$

$$E^h_{glob}(t_{min}, t_{max}) = \frac{1}{N(N-1)} \sum_{ij} E^h_{T_{ij}}(t_{min}, t_{max})$$

Some local temporal metrics are also defined:

$N_i(t_{min}, t_{max})$ is the set of all first-hop neighbors seen by node i at least once in the time interval $[t_{min}, t_{max}]$

$k_i(t_{min}, t_{max})$ is the number of nodes in the set $N_i(t_{min}, t_{max})$

Considering the sequence of subgraphs $G_t^{N_i(t_{min},t_{max})}$, $t = t_{min}$, t_{min+w}, ..., t_{max} where each $G^{N_i(t_{min},t_{max})}$ is the neighbor subgraph of node i, considering only the nodes in $N_i(t_{min}, t_{max})$ and retaining the edges from $G_{t_{min}}$, the clustering coefficient $C_i(t_{min}, t_{max})$ of node i is defined as:

$$C_i(t_{min}, t_{max}) = \frac{\sum_{t=t_{min}}^{t_{max}} \#\text{ of edges in } G_t^{\mathcal{N}_i(t_{min},t_{max})}}{\tau \cdot \frac{k_i(t_{min},t_{max})[k_i(t_{min},t_{max})-1]}{2}}$$

where the maximum time to live of a message is $\tau = (t_{max} - t_{min})$.

The local efficiency of the node i in the time window $[t_{min}, t_{max}]$ is:

$$E_{loc_i}(t_{min}, t_{max}) = E_T\{G_t^{\mathcal{N}_i(t_{min},t_{max})} \quad t \in [t_{min}, t_{max}]\}$$

The characteristic temporal clustering coefficient and the temporal local efficiency are defined as:

$$C(t_{min}, t_{max}) = 1/N \sum_i C_i(t_{min}, t_{max})$$

$$E_{loc}(t_{min}, t_{max}) = 1/N \sum_i E_{loc_i}(t_{min}, t_{max})$$

Then an analysis of some networks is done. The networks studied are: Bluetooth traces of people at the 2005 INFOCOM conference, campus Bluetooth traces of students and staff at MIT and email traces from Kiel University. They refer to these as INFOCOM,

REALITY and EMAIL, respectively. Table 2.1 describes the characteristics of each set of traces:

Table 2.1: Experimental Data Sets

	INFOCOM	REALITY	EMAIL
Start	2005-03-13	2004-07-26	2001-07-29
Duration	4 days	280 days	112 Days
Times	day1:6pm-12pm day2:12am-12pm day3:12am-12pm day4:12am-5pm	12am-12pm	12am-12pm
No. of nodes	41	100	59812
Contacts	avg. 4817	avg. 231	avg. 4000
Granularity	120 secs.	300 secs.	1 sec.

Table 2.2 shows calculations for both the static and temporal clustering coefficient C and path length L for the INFOCOM dataset ($h = max$, $t_{min} = 12am$, $t_{max} = 12pm$, $w = 5min$):

Table 2.2: INFOCOM Static and Temporal Metrics ($h = max$, $t_{min} = 12am$, $t_{max} = 12pm$, $w = 5min$)

			Static				Temporal		
Day	N	$\langle k \rangle$	C	L	C_{rand}	L_{rand}	C	L^*	Disc
1	37	25.7	0.818	1.291	0.764	1.336	0.033	4.090	0.28
2	39	28.3	0.845	1.269	0.824	1.263	0.110	4.556	0.13
3	38	22.3	0.744	1.420	0.644	1.405	0.077	4.003	0.19
4	39	21.4	0.722	1.444	0.541	1.474	0.052	4.705	0.14

The observations that they made were the following: temporal length $L^* \gg$ static length L, and there are much more disconnected node pairs in the temporal version due

to the asymmetry and time ordering of paths. Also, temporal C <static C because the static graph assumes edges always stay there across time, when in fact they come and go.

Below are the results they got for the temporal metrics for all three datasets:

Table 2.3: INFOCOM ($h = 1$, $t_{min} = 12am$, $t_{max} = 12pm$, $w = 5min$, shuffled runs = 50)

Temporal Metrics					Reshuffled		
Day	C	E_{loc}	L	E_{glob}	E_{loc}	L	E_{glob}
1	0.033	0.003	19h 39m	0.003	0.077	5h 29m	0.100
2	0.110	0.020	9h 6m	0.024	0.194	2h 45m	0.239
3	0.077	0.013	10h 32m	0.018	0.114	4h 6m	0.167
4	0.052	0.009	9h 55m	0.013	0.104	3h 3m	0.165

Table 2.4: REALITY ($h = 1$, $t_{min} = 12am$, $t_{max} = 12pm$, $w = 5min$, shuffled runs = 50)

Temporal Metrics					Reshuffled		
Date	C	Eloc	L	Eglob	E_{loc}	L	E_{glob}
08 Sep	0.014	0.000	23h 15m	0.000	0.003	21h 58m	0.010
15 Sep	0.060	0.000	22h 47m	0.001	0.007	19h 55m	0.024
22 Sep	0.061	0.000	22h 53m	0.001	0.007	20h 42m	0.019
29 Sep	0.060	0.001	22h 20m	0.001	0.009	17h 44m	0.037
06 Oct	0.026	0.000	22h 14m	0.001	0.011	16h 23m	0.041
13 Oct	0.038	0.000	21h 37m	0.004	0.013	14h 57m	0.055
20 Oct	0.067	0.001	21h 45m	0.003	0.007	17h 4m	0.031
27 Oct	0.050	0.002	22h 1m	0.001	0.013	15h 19m	0.050
03 Nov	0.051	0.001	21h 6m	0.004	0.012	16h 17m	0.043
10 Nov	0.051	0.000	20h 5m	0.004	0.015	14h 25m	0.061

Table 2.5: EMAIL ($h = 1$, $t_{min} = 12am$, $t_{max} = 12pm$, $w = 5min$, shuffled runs = 50)

Temporal Metrics					Reshuffled		
Date	C	Eloc	L	Eglob	E_{loc}	L	E_{glob}
27Oct	0	$3.1E^{-8}$	86397.94s	$9.3E^{-7}$	$7.7E^{-8}$	86396.91s	$1.6E^{-6}$
28Oct	$3.5E^{-7}$	$4.0E^{-8}$	86399.78s	$1.4E^{-7}$	$4.1E^{-8}$	86399.71s	$1.5E^{-7}$
29Oct	$2.5E^{-7}$	$3.9E^{-8}$	86399.03s	$3.9E^{-7}$	$7.2E^{-8}$	86398.59s	$7.3E^{-7}$
30Oct	0	$5.8E^{-8}$	86398.76s	$5.5E^{-7}$	$6.9E^{-8}$	86398.48s	$7.5E^{-7}$
31Oct	0	$4.7E^{-8}$	86398.92s	$4.9E^{-7}$	$6.5E^{-8}$	86398.64s	$6.9E^{-7}$
01Nov	0	$5.8E^{-8}$	86399.03s	$4.9E^{-7}$	$6.6E^{-8}$	86398.85s	$6.0E^{-7}$
02Nov	0	$4.3E^{-8}$	86398.68s	$5.4E^{-7}$	$6.5E^{-8}$	86398.67s	$6.8E^{-7}$

The "Reshuffled" columns in the three tables above show the metrics calculated on reshuffled temporal graphs for INFOCOM, REALITY and EMAIL, respectively. This is done to destroy any inherent time order. There are no results for the temporal clustering coefficient C since, by definition, it is not affected by the time ordering of windows. As we can see in all three traces, the shuffled network gives a quicker data diffusion time and higher clustering and efficiency. The reason for this is down to the cyclic behavior of humans contacts.

2.2.2 Temporal Indicators and Metrics

In [28], following the definition of [7], a time-varying graph (TVG) is defined as a set of nodes V and a set of edges E connecting the nodes with a presence function ρ which indicates whether a given edge is present at a given time during a time span $\mathcal{T} \subseteq \mathbb{T}$ called the lifetime of the system. Simply put, it is a graph with edges that can appear and disappear across time. (This model is the one used in this thesis, so a more formal definition will be given in another section below)

A journey in a time-varying graph is the temporal extension of the notion of path in a static graph. Journey can be thought of as paths over time from a source to a destination and therefore have both a topological and a temporal length. The topological length of

a journey is the number of hops in the journey. The temporal length of a journey is the duration of the journey.

Since in a time-varying graph there are three distinct measures of distances, there are also three different types of "minimal" journeys. The shortest journey between two nodes is the journey with the least hops. The foremost journey is the journey that arrives to the destination the soonest. The fastest journey is the journey that takes the least time. (Note that the fastest journey is different from the foremost journey since to take the fastest journey, we may have to wait a long time for the appropriate edges to appear, while the foremost journey may start earlier, have a journey slightly longer than the fastest journey, but still arrives at the destination sooner)

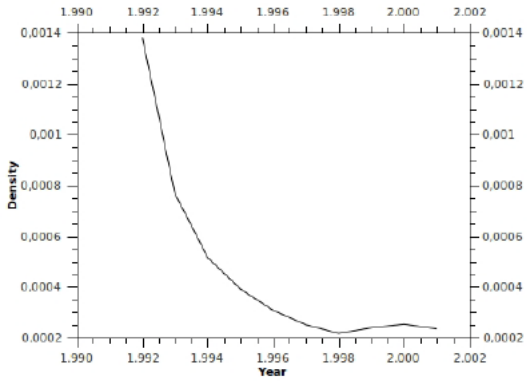
The authors then explain some atemporal parameters and use them to analyze a network. The dataset consists of a collection of papers and their related citations over the period from January 1992 to May 2003. For each paper the set of authors, the dates of on-line deposit, and the references to other papers are provided. There are 352 807 citations within the total amount of 29 555 papers written by 59 439 authors. From the dataset, they extract the network of the most proficient authors - i.e., the authors of papers which received more than 150 citations. In all the example charts, a one-year time window is used.

The density of a graph $G = (V, E)$ is:

$$D = \frac{|E|}{|V| * (|V| - 1)}$$

Figure 2.6 shows the result they obtained for the density of the network of the most proficient authors:

Figure 2.6: Evolution of the Density



The clustering coefficient of a node is:

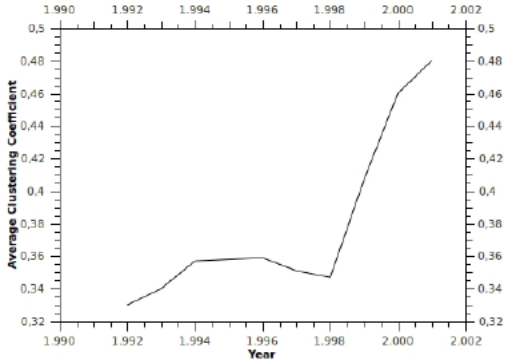
$$C(x) = \frac{|\{(u,v) : u,v \in N(x)\}|}{deg(x)(deg(x)-1)}$$

The average clustering coefficient of a graph can then be defined as the average over all nodes:

$$AC = \frac{1}{|V|}\sum_{x \in V} C(x)$$

Figure 2.7 shows the evolution of the average clustering coefficient of the network of the most proficient authors:

Figure 2.7: Average Clustering Coefficient Evolution

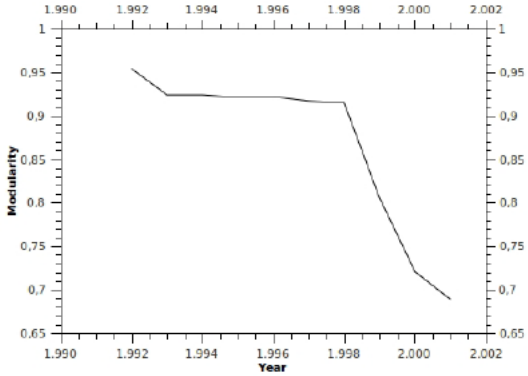


The modularity of a pair of nodes u and v is defined as:

$$M(u,v) = \frac{deg(u) * deg(v)}{2|E|}$$

Figure 2.8 shows the evolution of the average modularity for the network of the most proficient authors:

Figure 2.8: Evolution of the Modularity



Some temporal parameters are then given. In the following formulas, $d(u,v)$ corresponds to the shortest journey between the nodes u and v. In all the formulas below,

$d(u,v)$ can be replaced with $\delta(u,v)$ (foremost journey), or $\hat{\delta}(u,v)$ (fastest journey) depending on which version of the formula we want.

The eccentricity of a node u in a TVG $\mathcal{G}$ is:

$$e(u) = \max\{d(u,v) : v \in V\}$$

The diameter of a TVG $\mathcal{G}$ is:

$$max\{e_i(u) : u \in V\}$$

The betweenness of a node q is:

$$B(q) = \sum_{v \neq u \neq q \in V} \frac{|d'(u,v,q)|}{|d(u,v)|}$$

where $|d'(u,v,q)|$ is the number of shortest journeys between the nodes u and v that pass through q, and $|d(u,v)|$ is the total number of shortest journeys between the nodes u and v.

The closeness of a node u is:

$$TC(u) = \sum_{v \in V \setminus u} \frac{d(u,v)}{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}$$

2.3 Impact of Time in Knowledge Mobilization Networks

In [2], knowledge mobilization (KM) is defined as the use of knowledge towards the achievement of goals.

A knowledge mobilization network (KMN) is a network based on knowledge mobilization and researchers have begun analyzing them using a social network analysis (SNA)

approach. However, this was done the classical way, using static measures. This paper proposes to include time in the calculation of these measures, making them temporal measures. It then shows how a temporal measure can differ from a static measure with an example. The graph used in that example is called Knowledge-Net and the measure used is betweenness.

Knowledge-Net is a network where the nodes are human or non-human actors and the edges represent knowledge mobilization between two actors. (Knowledge-Net is also the main network used in this thesis, so more details about it will be given in another section below) It can be represented as a time-varying graph.

First, some static measures are calculated for Knowledge-Net (reported in Table 2.6).

Table 2.6: Static Measures Computed on Knowledge-Net

	2005	2006	2007	2008	2009	2010	2011
Ave. Degree	1.40	1.32	1.63	1.84	1.98	2.02	2.04
Diameter	4	5	5	6	6	6	6
Density	0.31	0.04	0.04	0.02	0.02	0.01	0.01
#Communities	4	3	6	8	8	15	12
Modularity	0.17	0.52	0.46	0.47	0.46	0.54	0.54
Ave. Clustering Coefficient	0.41	0.06	0.21	0.22	0.20	0.24	0.23
Ave. Path Length	2.04	3.04	3.06	3.26	3.34	3.46	3.50
Ave. Normalized Closeness	0.51	0.33	0.33	0.31	0.30	0.29	0.29
Ave. Eccentricity	3.10	4.41	4.40	4.70	4.80	4.83	4.83
Ave. Betweenness	4.70	58.36	83.53	169.70	234.89	354.23	456.18
Ave. Normalized Betweenness	0.13	0.03	0.02	0.01	0.01	≈ 0	≈ 0
Ave. Page Rank	0.10	0.01	0.01	≈ 0	≈ 0	≈ 0	≈ 0
Ave. Eigenvector	0.52	0.19	0.15	0.10	0.09	0.07	0.05

Although some observations can be made from those results, a static analysis like that cannot provide a deep temporal understanding. So, the authors propose to study Knowledge-Net using a form of temporal betweenness that makes use of time in an explicit manner.

The static betweenness of a node $v \in V$ in a static graph $G = (V, E)$ is defined as:

$$B(v) = \sum_{u \neq w \neq v \in V} \frac{|P(u,w,v)|}{|P(u,w)|}$$

where $|P(u,w)|$ is the number of shortest paths from u to w in G, and $|P(u,w,v)|$ is the number of those passing through v.

Since the number of foremost journeys between two nodes can be exponential and the computation of foremost betweenness is an intractable task, another form of foremost betweenness is considered. Even though that new form of foremost betweenness can have an exponential number of foremost journeys, it is more manageable. The new foremost betweenness $TB_{\mathcal{F}}^{\mathcal{T}}(v)$ for a node v with lifetime $\mathcal{T}$ is then defined as:

$$TB_{\mathcal{F}}^{\mathcal{T}}(v) = \sum_{u \neq w \neq v \in V} \frac{|\mathcal{F}^{\mathcal{T}}(u,w,v)|}{|\mathcal{F}^{\mathcal{T}}(u,w)|}$$

where $|\mathcal{F}^{\mathcal{T}}(u,w)|$ is the number of foremost *increasing journey routes* between u and w during the time frame $\mathcal{T}$ and $|\mathcal{F}^{\mathcal{T}}(u,w,v)|$ is the number of the ones passing through v in the same time frame.

The nodes of Knowledge-Net are then ranked a first time based on their foremost betweenness values and ranked a second time based on their static betweenness values. These two rankings are then compared. The results obtained are reported in Table 2.7.

Table 2.7: Betweenness in Knowledge-Net

Temporal to Static				Static to Temporal		
Actor	Temporal Rank	Static Rank		Actor	Static Rank	Temporal Rank
L1(05)	1	1		L1(05)	1	1
H1(05)	2	2		H1(05)	2	2
A1(06)	3	3		A1(06)	3	3
A2(08)	4	4		A2(08)	4	4
P1(06)	5	8		A5(08)	5	12
A3(07)	6	9		A4(09)	6	7
A4(09)	7	6		P2(08)	7	9
S1(10)	8	115		P1(06)	8	5
P2(08)	9	7		A3(07)	9	6
J1(06)	10	160		P3(10)	10	17
C1(07)	11	223		A6(11)	11	18
A5(08)	12	5		A8(09)	12	36
I1(09)	13	28		P4(10)	13	22
O1(05)	14	45		P5(11)	14	27
S2(05)	15	46		H2(05)	15	44
I2(05)	16	47		A7(09)	16	21
P3(10)	17	10		A9(10)	17	31
A6(11)	18	11		P5(11)	18	69
C2(10)	19	133		P6(10)	19	23
J2(09)	20	182				
A7(09)	21	16				

Note that only the nodes with a high betweenness value are considered in the table above. As can be seen, the four highest ranked nodes are the same for the static and temporal versions. The nodes that have a high static rank also have a high temporal rank, although there are some nodes with a low static rank but a high temporal rank.

Then some new concepts are defined. Rapids are the nodes with high foremost betweenness values. Brooks are the nodes with insignificant foremost betweenness values. Invisible rapids are the nodes whose temporal betweenness rank is considerably higher than their static betweenness rank. Invisible brooks are the nodes whose static betweenness rank is considerably higher than their temporal betweenness rank.

The major invisible rapids found in Knowledge-Net are reported in Table 2.8, and the major invisible brooks in Table 2.9.

Table 2.8: Invisible Rapids in Knowledge-Net

Actor	Time	Temp. Rank	Stat. Rank	Type
P1(06)	[07-11]	5	105	project
S1(10)	[05-11]	8	115	poster
	[06-11]	8	113	
	[07-11]	7	115	
	[08-11]	5	104	
J1(06)	[05-11]	10	160	journal
	[06-11]	10	154	
	[07-11]	10	223	
C1(07)	[05-11]	11	223	citing publication
	[06-11]	11	220	
J2(09)	[06-11]	17	179	journal
	[07-11]	16	182	
C2(10)	[05-11]	19	133	citing poster
	[06-11]	16	132	
	[07-11]	15	133	

Table 2.9: Invisible Brooks in Knowledge-Net

Actor	Time	Stat. Rank	Temp. Rank	Type
J3(08)	[08-11]	9	117	journal
	[09-11]	12	84	
C3(11)	[08-11]	10	191	citing publication
	[09-11]	15	153	
C4(11)	[08-11]	15	105	citing publication
H2(05)	[06-11]	16	118	researcher
	[07-11]	15	134	
A3(07)	[08-11]	16	187	publication
C5(07)	[08-11]	18	158	citing publication

Chapter 3

Time-Varying Graphs

3.1 Definitions

A time-varying graph, as defined in [7], is a graph where each node and each edge comes with a list of time intervals, representing the presence schedule over time, plus sets of weights for the edges, representing length, traversal cost, traversal time, etc.

A journey in a time-varying graphs is equivalent to a path in an usual graph. There are three different quality measures of journeys: the number of hops or length of the journey, the arrival date and the journey time. The length of a journey is similar to the length of a path, while the arrival date and the journey time are new measures introduced with time-varying graphs.

Using these measures, we can define the notion of "distance" in a time-varying graph in three different ways: the shortest journey which is the journey with the minimum number of hops, the foremost journey which is the journey with the earliest arrival date and the fastest journey which is the journey with the minimum journey time.

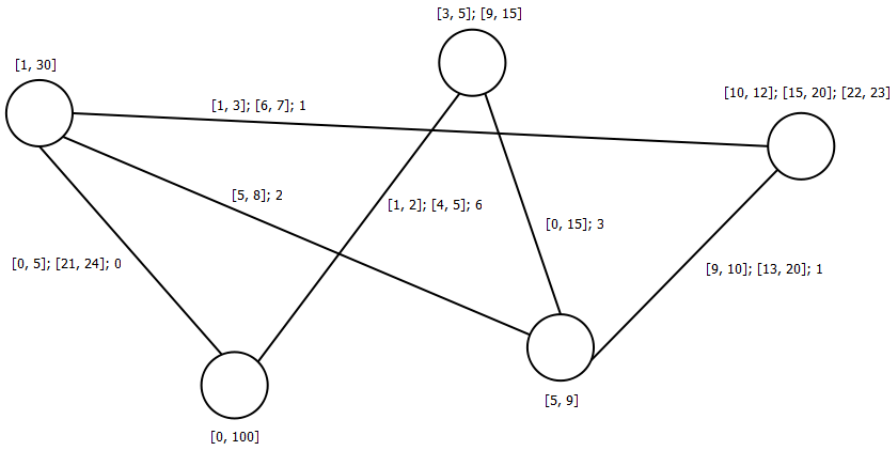
A time-varying graph can be defined as $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, where:

- V is the set of entities (nodes)
- E is the set of relations between the entities (edges)
- $\mathcal{T} \subseteq \mathbb{T}$ is the lifetime of the system
- $\rho : E \times \mathcal{T} \rightarrow \{0, 1\}$, called *presence* function, indicates whether a given edge is available at a given time
- $\zeta : E \times \mathcal{T} \rightarrow \mathbb{T}$, called *latency* function, indicates the time it takes to cross a given edge if starting at a given date (the latency of an edge could vary in time)

This definition can be extended by adding a *node presence function* $\psi : V \times \mathcal{T} \rightarrow \{0, 1\}$ (*i.e.,* the presence of a node is conditional upon time) and a *node latency function* $\varphi : V \times \mathcal{T} \rightarrow \mathbb{T}$ (accounting *e.g.* for local processing times).

For example, Figure 3.1 shows a time-varying graph. Each node has one or more time intervals and exists only within those time intervals. Each edge, like each node, has one or more time intervals and exists only within those time intervals. But for the edges, we also have a number which corresponds to the traversal time of the edge.

Figure 3.1: Example of a Time-Varying Graph



3.2 The Underlying Graph G

Given a TVG $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, the graph $G = (V, E)$ is called *underlying* graph of $\mathcal{G}$. This static graph should be seen as a sort of *footprint* of $\mathcal{G}$, which flattens the time dimension and indicates only the pairs of nodes that have relations at some time in $\mathcal{T}$.

In most studies and applications, it is assumed that G is connected, but in general, this is not the case.

3.3 Journeys

A sequence of couples $\mathcal{J} = \{(e_1, t_1), (e_2, t_2) \ldots, (e_k, t_k)\}$, such that $\{e_1, e_2, ..., e_k\}$ is a walk in G, is a *journey* in $\mathcal{G}$ if and only if $\rho(e_i, t_i) = 1$ and $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ for all $i < k$.

We denote by $departure(\mathcal{J})$, and $arrival(\mathcal{J})$, the starting date t_1 and the last date $t_k + \zeta(e_k, t_k)$ of a journey $\mathcal{J}$, respectively.

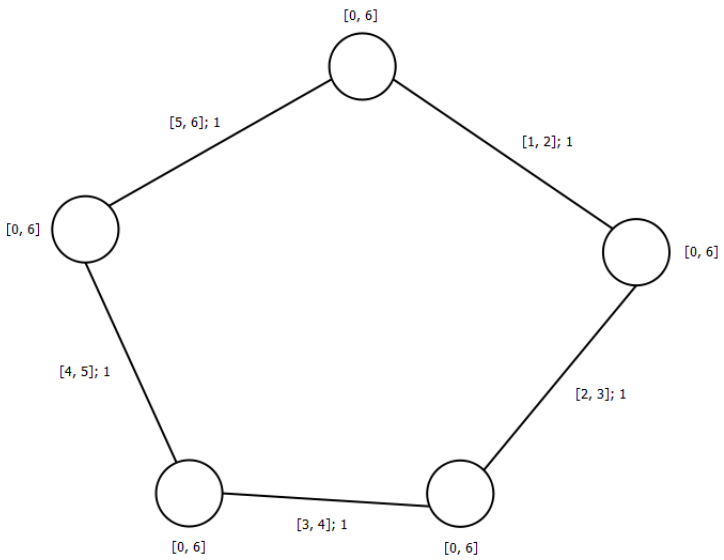
Journeys can be thought of as *paths over time* from a source to a destination and therefore have both a *topological* length and a *temporal* length.

The *topological length* of $\mathcal{J}$ is the number $|\mathcal{J}| = k$ of couples in $\mathcal{J}$ (i.e., the number of *hops*). Its *temporal length* is its end-to-end duration: $arrival(\mathcal{J}) - departure(\mathcal{J})$.

Let us denote by $\mathcal{J}^*_{\mathcal{G}}$ the set of all possible journeys in a time-varying graph $\mathcal{G}$, and by $\mathcal{J}^*_{(u,v)} \subseteq \mathcal{J}^*_{\mathcal{G}}$ those journeys starting at node u and ending at node v. If a journey exists from a node u to a node v, that is, if $\mathcal{J}^*_{(u,v)} \neq \emptyset$, then we say that u can *reach* v, and allow the simplified notation $u \rightsquigarrow v$. Clearly, the existence of journey is not symmetrical: $u \rightsquigarrow v \not\Leftrightarrow v \rightsquigarrow u$; this holds regardless of whether the edges are directed or not, because the time dimension creates its own level of direction. Given a node u, the set $\{v \in V : u \rightsquigarrow v\}$ is called the *horizon* of u.

When a round journey ends, nothing implies the existence of another time schedule allowing to use the same route again. Figure 3.2 is showing that property. If we start at the top node and go clockwise, there is a round journey (a round journey is equivalent to a circuit in an usual graph) that goes through all the other nodes and returns to the top node. But that round journey can only be taken once because of the time dimension.

Figure 3.2: Round Journey in a Time-Varying Graph



3.4 Distances

As seen above, the length of a journey can be measured both in terms of hops or time. This results in two distinct definitions of *distance* in a time-varying graph $\mathcal{G}$:

- The *topological distance* from a node u to a node v at time t, noted $d_{u,t}(v)$, is defined as $Min\{|\mathcal{J}| : \mathcal{J} \in \mathcal{J}^*_{(u,v)}, departure(\mathcal{J}) \geq t\}$. For a given date t, a journey whose departure is $t' \geq t$ and topological length is equal to $d_{u,t}(v)$ is qualified as *shortest*.

- The *temporal distance* from u to v at time t, noted $\hat{d}_{u,t}(v)$ is defined as $Min\{arrival(\mathcal{J}) : \mathcal{J} \in \mathcal{J}^*_{(u,v)}, departure(\mathcal{J}) \geq t\} - t$. Given a date t, a journey whose departure is $t' \geq t$ and arrival is $t + \hat{d}_{u,t}(v)$ is qualified as *foremost*. Finally, for any given date t, a journey whose departure is $\geq t$ and temporal length is $Min\{\hat{d}_{u,t'}(v) : t' \in \mathcal{T} \cap [t, +\infty)\}$ is qualified as *fastest*.

3.5 Temporal Closeness

In the static context, the closeness (or shortest closeness) can be defined as the inverse of the mean of the shortest paths between a node and all the other reachable nodes. More formally, it can be defined as:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{d(u,v)} \tag{3.1}$$

where $d(u,v)$ is the shortest path between the nodes u and v.

With that definition, it is good for a node to have a high closeness value since that means it can reach the other nodes fast.

In the temporal context, we have three different variations of the closeness: temporal shortest closeness (which is different from the static shortest closeness defined above), (temporal) foremost closeness and (temporal) fastest closeness. Since the last two variations do not have a static counterpart, we will omit using "temporal" when talking about them to keep everything simpler.

The temporal shortest closeness is very similar to the static shortest closeness. The only difference is that instead of being "the inverse of the mean of the shortest paths between a node and all the other reachable nodes", it is "the inverse of the mean of the shortest journeys between a node and all the other reachable nodes". We then have the following formula:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{d(u,v)} \tag{3.2}$$

where $d(u,v)$ is the shortest journey between the nodes u and v.

For the definition of foremost closeness, we replace "shortest paths" by "foremost journeys" in the definition of shortest static closeness, which gives us: "the inverse of the mean of the foremost journeys between a node and all the other reachable nodes". The

formula is then:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{\delta(u,v)} \tag{3.3}$$

where $\delta(u,v)$ is the foremost journey between the nodes u and v.

And for the definition of fastest closeness, we replace "shortest paths" by "fastest journeys" in the definition of shortest static closeness, which gives us: "the inverse of the mean of the fastest journeys between a node and all the other reachable nodes". The formula is then:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{\hat{\delta}(u,v)} \tag{3.4}$$

where $\hat{\delta}(u,v)$ is the fastest journey between the nodes u and v.

The definitions above are just a direct translation of the static definition to a temporal context. However, this temporal context introduces a few inconsistencies that must be addressed. First, a connected graph in the static context can become disconnected in the temporal context because of some edges disappearing at a certain point in time. And since the static definition of closeness only involves "reachable nodes", a node that is completely disconnected from the rest of the graph will still have a very high static closeness value. In the static context, this is not a problem since we only have two cases: Either the graph is connected and the computation can be done normally, or the graph is disconnected but stays in that state allowing us to do a particular computation for each of its components. However, in the temporal context, where the number of components varies in time because of the edges appearing and disappearing, this behavior is not wanted, so we decided to multiply the formulas above by a coefficient that takes into account the size of each component of the graph. We then had:

Temporal shortest closeness:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{d(u,v)} \times \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{|v \in V| - 1} \tag{3.5}$$

where $d(u,v)$ is the shortest journey between the nodes u and v.

Foremost closeness:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{\delta(u,v)} \times \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{|v \in V| - 1} \tag{3.6}$$

where $\delta(u,v)$ is the foremost journey between the nodes u and v.

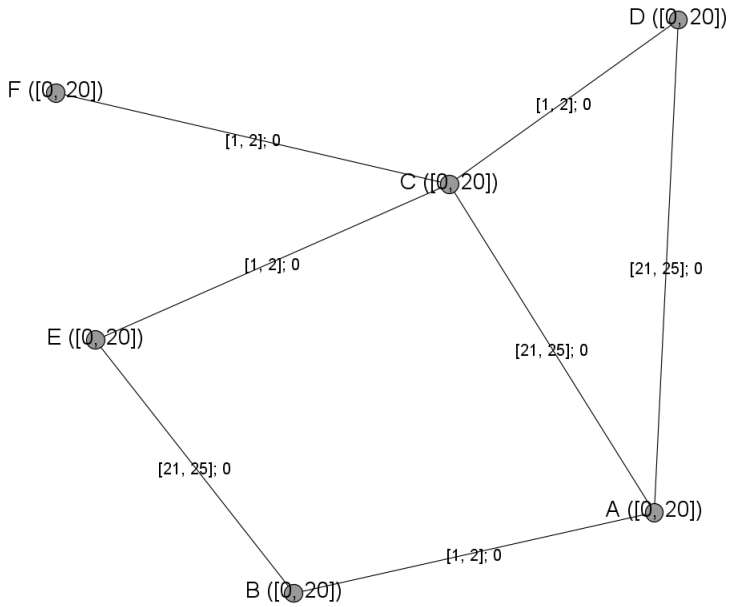
Fastest closeness:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{\hat{\delta}(u,v)} \times \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{|v \in V| - 1} \tag{3.7}$$

where $\hat{\delta}(u,v)$ is the fastest journey between the nodes u and v.

Below is a small example to show how the formulas above work. Figure 3.3 shows a time-varying graph with traversal time = 0:

Figure 3.3: Time-Varying Graph 1 with Traversal Time = 0



In the static context, we have a connected graph. But in the temporal context, the graph is actually disconnected since the edges AC, AD and BE can not be traversed because they appear after the nodes connected to them disappear. So, in the temporal context, during the algorithm's execution, we have two components: one consisting of the nodes A and B, and one consisting of the nodes C, D, E and F. If we apply formula 3.3, we would have the foremost closeness values shown in Table 3.1:

Table 3.1: Foremost Closeness Values for Figure 3.3 Using Formula 3.3

Node	Foremost closeness
A	1
B	1
C	1
D	1
E	1
F	1

Every node has the same value, but by looking at the graph, the component consisting of the nodes C, D, E and F should be more important than the component consisting of the nodes A and B since it is bigger. By applying formula 3.6, this problem get solved since we have the foremost closeness values shown in Table 3.2:

Table 3.2: Foremost Closeness Values for Figure 3.3 Using Formula 3.6

Node	Foremost closeness
A	0.2
B	0.2
C	0.6
D	0.6
E	0.6
F	0.6

Formula 3.6 ensures that nodes that can reach more nodes have a higher foremost closeness value.

We also had the problem that in a time-varying graph, the nodes that are born first have a greater chance to have a higher closeness value than the nodes born later. So we slightly modified the formulas above to take into account the date of birth of the nodes. This gave us our final definitions:

Temporal shortest closeness:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{d(u,v) - max(birth(u), birth(v))} \times \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u,w)\}|}{|v \in V| - 1} \tag{3.8}$$

where $d(u, v)$ is the shortest journey between the nodes u and v.

Foremost closeness:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u, w)\}|}{\delta(u, v) - max(birth(u), birth(v))} \times \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u, w)\}|}{|v \in V| - 1} \tag{3.9}$$

where $\delta(u, v)$ is the foremost journey between the nodes u and v.

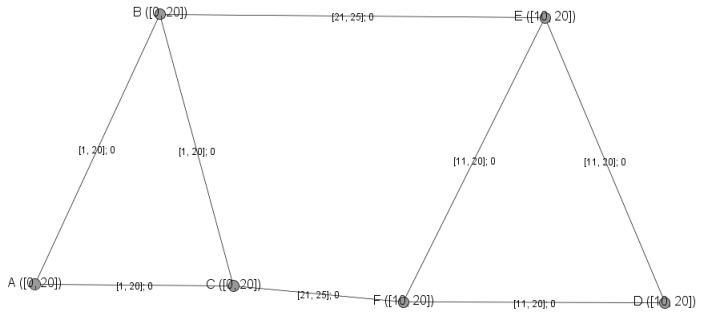
Fastest closeness:

$$C(u) = \sum_{v \in V \setminus u} \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u, w)\}|}{\hat{\delta}(u, v) - max(birth(u), birth(v))} \times \frac{|\{w \in V : \exists \mathcal{J} \in \mathcal{J}^*(u, w)\}|}{|v \in V| - 1} \tag{3.10}$$

where $\hat{\delta}(u, v)$ is the fastest journey between the nodes u and v.

Below is an example showing the utility of the changes we made. Consider the time-varying graph with traversal time = 0 shown in Figure 3.4:

Figure 3.4: Time-Varying Graph 2 with Traversal Time = 0



This graph has two components. One consisting of the nodes A, B and C and one consisting of the nodes D, E and F. The two are connected in the static context but not in the temporal context where the edges BE and CF can not be traversed. The two components are similar, with the only difference being the birthdates of the nodes and

edges. The nodes A, B and C along with the edges connecting them are born before the nodes D, E and F and the edges connecting them. If we apply formula 3.6 to the graph above, we get the results shown in Table 3.3:

Table 3.3: Foremost Closeness Values for Figure 3.4 Using Formula 3.6

Node	Foremost closeness
A	0.4
B	0.4
C	0.4
D	0.03636363636363636
E	0.03636363636363636
F	0.03636363636363636

Because the nodes A, B and C were born earlier, they have higher foremost closeness values. Normally, this would be fine, but for the dataset that we are studying, this behavior is not wanted. We do not want a node to have a higher closeness value just because it was born earlier. Applying formula 3.9, we get what is shown in Table 3.4:

Table 3.4: Foremost Closeness Values for Figure 3.4 Using Formula 3.9

Node	Foremost closeness
A	0.4
B	0.4
C	0.4
D	0.4
E	0.4
F	0.4

The problem is solved since a node born earlier does not have an advantage anymore.

Chapter 4

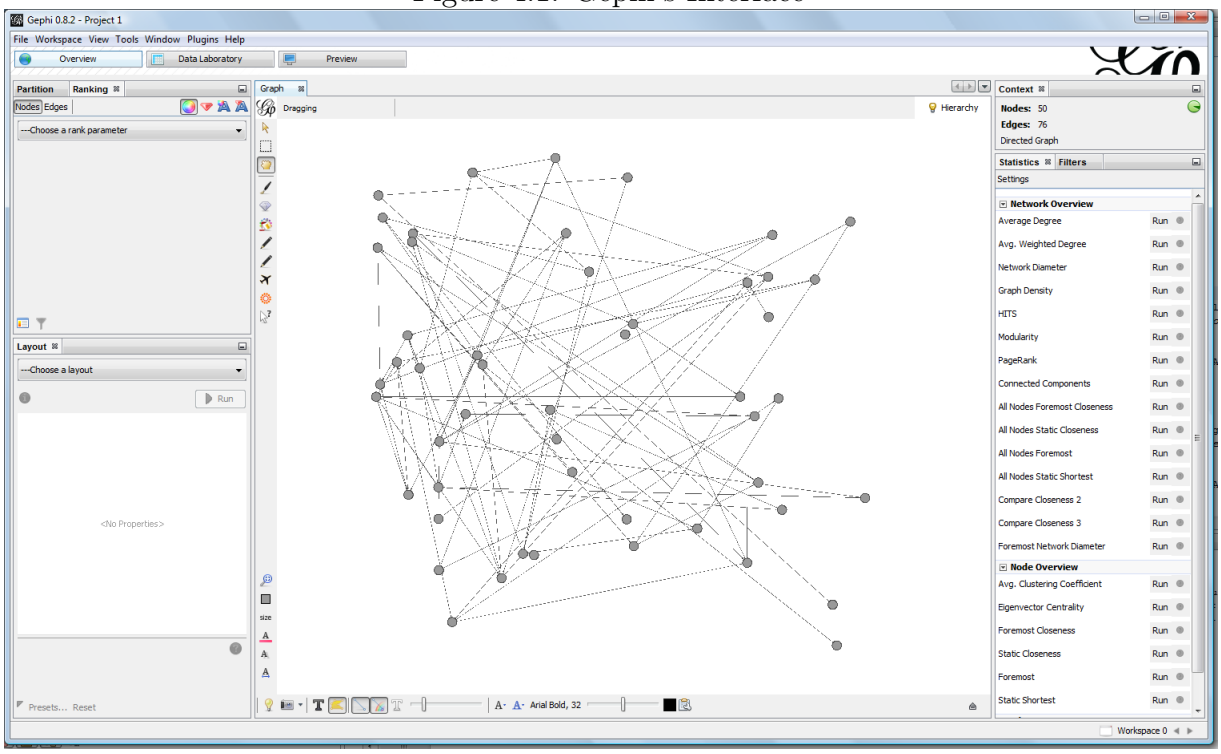
Gephi and Foremost Journeys Implementation

In this Chapter we describe the implementation of an algorithm for computing Foremost Journeys in a time-varying graph and its integration into Gephi.

4.1 Gephi

Gephi is an open source software for graph and network analysis (available at `https://gephi.org/`). It uses a 3D render engine to display large networks in real-time and to speed up the exploration. The interface of Gephi is shown in Figure 4.1:

Figure 4.1: Gephi's Interface



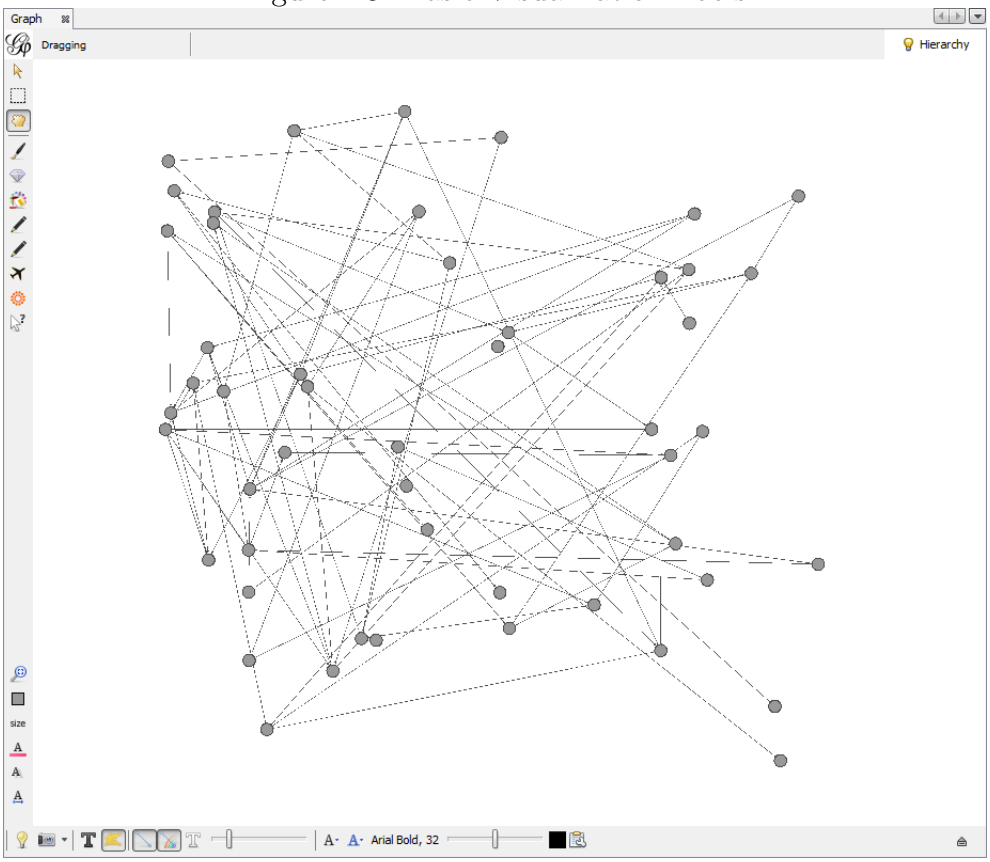
A graph can be created directly in Gephi or it can be imported. Gephi supports several

standard graph file formats. In Figure 4.2 is a table that shows the supported formats along with the features that can be used with each one:

Figure 4.2: Supported Formats

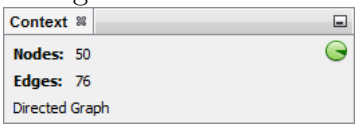
After the graph is created or imported, it will show up in the central part of the interface which contains some basic visualization tools such as changing the color of the nodes or edges, showing the node labels or edge labels, moving the nodes around, etc. (see Figure 4.3)

Figure 4.3: Basic Visualization Tools



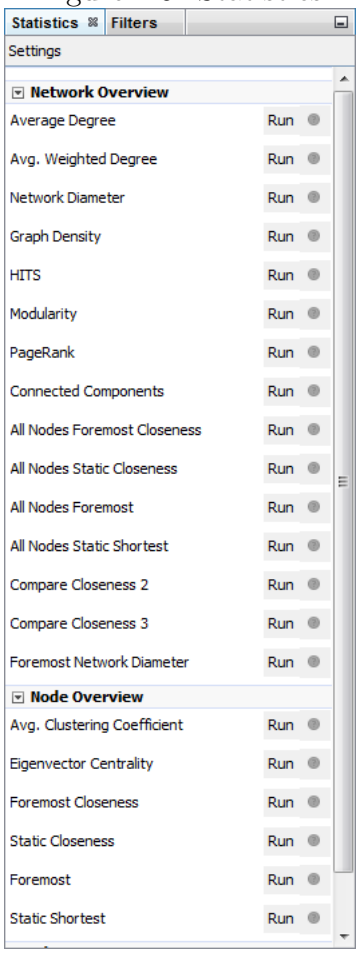
On the top right of the interface, under "Context", one can see the number of nodes and edges in the graph as well as whether the graph is directed or undirected. (see Figure 4.4)

Figure 4.4: Context



On the bottom right of the interface, under "Statistics", there is a list of algorithms that can be run on the graph, such as "Average Degree", "Network Diameter", "Graph Density", etc. (see Figure 4.5)

Figure 4.5: Statistics



4.2 Computing Foremost Journeys

The algorithm to compute the foremost journeys from a source node to all other nodes was presented in [4]. It is reported below:

Input : *An evolving graph $\mathcal{G}$, a vertex $s \in V_{\mathcal{G}}$*
Output : *An array $t_{EAD}[v] \in \mathbb{T}$ which gives for each vertex $v \in V_{\mathcal{G}}$ the Earliest Arrival Date from s; and an array $father[v] \in V_{\mathcal{G}}$ which gives for each vertex $v \neq s \in V_{\mathcal{G}}$ its father in the ubiquitous foremost journey tree.*
Variables : *A min-heap Q of vertices, sorted by the array t_{EAD}. The array t_{EAD} will be updated.*

1. *Initialize $t_{EAD}[s] \leftarrow 0$; and for all $v \neq s \in V_{\mathcal{G}}$, $t_{EAD}[v] \leftarrow \infty$. Initialize Q with only s in the root.*
2. *While $Q \neq \emptyset$ do:*
   (a) *Extract u, the vertex at $root(Q)$, and close it.*
   (b) *Delete $root(Q)$.*
   (c) *Traverse the adjacency list of u, and for each open neighbor v do:*
      i. *Let $t = f((u, v), t_{EAD}[u])$.*
      ii. *If $t + \zeta(u, v) < t_{EAD}[v]$ then*
      Update $t_{EAD}[v] \leftarrow t + \zeta(u, v)$,
      Update $father[v] \leftarrow u$ and
      insert v in the Q if it was not there already.
   (d) *Update Q.*

We will explain how this algorithm works. In a static graph, the shortest paths from one node to all other nodes are computed with the Dijkstra's algorithm. The Dijkstra's algorithm works because prefix paths of shortest paths are also shortest paths. However, prefix journeys of foremost journeys are not necessarily foremost journeys. However, it can be shown that we can find foremost journeys with such a property in a time-varying graph. These foremost journeys are called ubiquitous foremost journeys (UFJ). This greatly helps to compute foremost journeys since it is possible to use an approach similar to the one employed in the Dijkstra's algorithm.

The input for the algorithm is a time-varying graph $\mathcal{G}$ and a node s which will be the node from which we will compute all the foremost journeys.

The output is an array $t_{EAD}[v]$ which gives for each node v the Earliest Arrival Date from s and an array $father[v]$ which gives for each node $v \neq s$ its father in the ubiquitous foremost journeys tree.

The variables used include a min-heap Q of nodes, sorted by the array $t_{EAD}[v]$. The array $t_{EAD}[v]$ will be updated.

At the beginning of the algorithm, $t_{EAD}[s]$ is set to 0, and for all $v \neq s$, $t_{EAD}[v]$ is set to ∞. Variable Q is initialized with only s in the root. The array $father[v]$ is left empty for all v.

Then, we remove node u in the root of Q and close it. For each open neighbor v of u, we check if we have a better earliest arrival date to v by going through u. If it is the case, we update $t_{EAD}[v]$ with the better earliest arrival date we found and we update $father[v]$ with u. We then insert v in Q if it was not there already and update Q. We repeat this process until Q is empty.

The foremost journey is found by backtracking in the array $father[v]$.

The algorithm termination is clear. At each step (a) of the algorithm, one node is closed and we never re-insert a closed node into the heap Q. Thus the loop is repeated at most N times, and the algorithm ends.

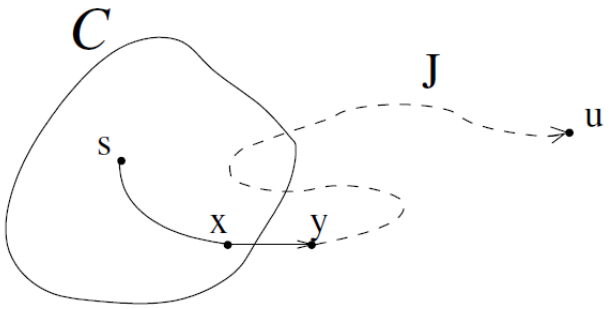
To prove that the algorithm is correct, we must prove that for all nodes u in $V_{\mathcal{G}}$, $t_{EAD}[u] = a(s, u)$ when u is closed. ($a(s, u)$ is the earliest arrival date for the journey that starts at s and ends at u)

Theorem 1. *For all nodes $u \in V_{\mathcal{G}}$, $t_{EAD}[u] = a(s, u)$ when u is closed.*

Proof. We will do that by induction on the set C of closed nodes. At the beginning, $C = s$ and $t_{EAD}[s] = 0 = a(s, s)$. The property holds.

Suppose that at some moment the algorithm has correctly computed C, and a node u is to be closed, i.e., the algorithm is at the moment just before closing u. Thus u has been inserted in the heap Q, so s and u are connected. Let $\mathcal{J}$ be an UFJ from s to u. This journey links the node s inside of C to the node u outside of C. Now let y be the first node in $\mathcal{J}$ which is not in C, and x be the node which immediately precedes y in $\mathcal{J}$ (see Figure 4.6).

Figure 4.6: Theorem 1



Since C has been correctly computed, then $t_{EAD}[x] = a(s, x)$. When x was closed, y was inserted into Q, and since y is before u in journey $\mathcal{J}$, $t_{EAD}[y] \leq t_{EAD}[u]$.

But we said at the beginning that the algorithm is at the moment just before closing u. This means that u was extracted from the root of Q which is sorted by the array t_{EAD}, meaning that $t_{EAD}[u]$ is the smallest in Q, and therefore we have $y = u$ and x is the node that immediately precedes u.

So before u was added to Q, $t_{EAD}[u]$ was updated with $f((x, u), a(s, x)) + \zeta(x, u)$.

Furthermore, we have the following property: Let s and v be two distinct nodes in $\mathcal{G}$, and $\mathcal{J}$ be an UFJ from s to v. Let u be the node which immediately precedes v in $\mathcal{J}$. Then $a(s, v) = f((u, v), a(s, u)) + \zeta(u, v)$. ($f((u, v), a(s, u))$ is the earliest moment after $a(s, u)$ where node u can retransmit a message to its neighbor v, and $\zeta(u, v)$ is the traversal time of the edge (u, v))

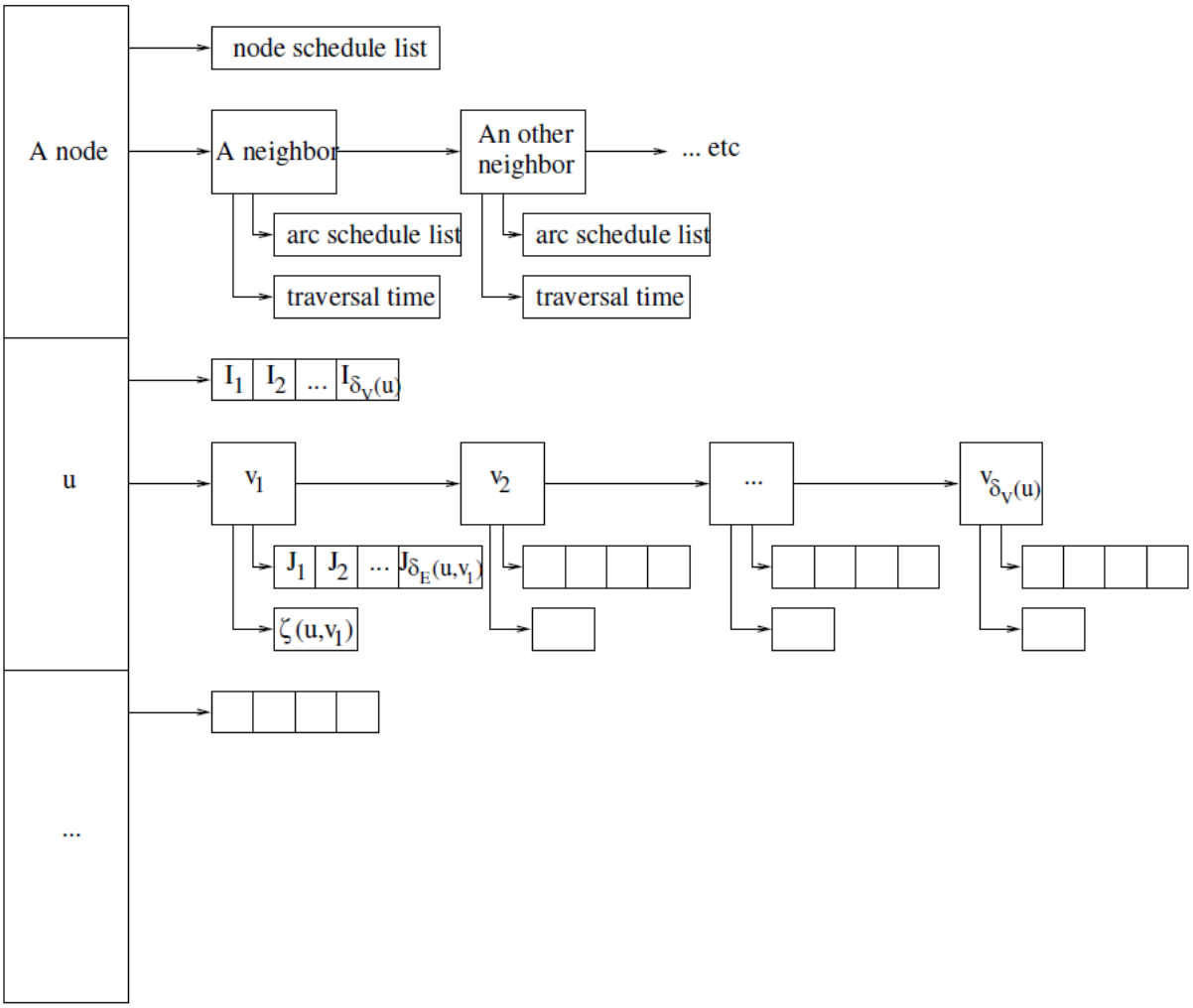
Hence $t_{EAD}[u] = a(s, u)$.

That proves that for all nodes u in $V_{\mathcal{G}}$, $t_{EAD}[u] = a(s, u)$ when u is closed. Therefore the algorithm is correct. □

4.3 Implementation of Foremost Journeys

The algorithm used to compute all foremost journeys from a source node (the one explained in the previous section) was implemented in Java. The data structure used is based on the one shown in Figure 4.7:

Figure 4.7: Data Structure



The data structure was built using multiple Java array lists. First, we have the most general array list that contains a series of array lists, one for each node.

Inside of each node's array list we have three elements. The first element is the "Id" of the node, the second element, called "Time Interval", is the node schedule list and the third element is an array list that contains a series of array lists, one for each of the node's neighbor.

Inside of each node's neighbor's array list we have five elements. The first element is

the "Id" of the neighbor, the second element, called "Time Interval", is the arc schedule list of the neighbor, the third element is the "Traversal Time" to get from the node to the neighbor in the time-varying graph, the forth element, called "Time Interval" (not to be confused with the second element which is also called "Time Interval"), is the node schedule list of the neighbor (This element was added to the original data structure to make the computation simpler) and the fifth element is the "Distance" between the node and its neighbor. The "Distance" corresponds to the weight of the edge between the node and the neighbor in the static graph (This element was added to the original data structure so that it could be used to compute both dynamic and static metrics)

Variable s is a string containing the "Id" of the source node. It is the starting node from where we will compute all the foremost journeys. This node is chosen by the user through the interface.

$t_{EAD}[v]$ was implemented using an array list containing a series of array lists, one for each node. Inside each node's array list we have two elements. The first element is the "Id" of the node and the second element is the current earliest arrival date from the source node s to this node. The second element will be updated throughout the computation.

$father[v]$ was implemented using an array list containing a series of array lists, one for each node. Inside each node's array list we have two elements. The first element is the "Id" of the node and the second element is the "Id" of its current father in the foremost journey from the source node s to it. The second element will be updated throughout the computation.

Q was implemented using an array list containing a series of array lists. Each array list in Q represents a node and consists of two elements. The first element is the "Id" of the node and the second element is the current earliest arrival time from the source node s to this node. The nodes are added and removed from Q throughout the computation.

$close[v]$ was implemented using an array list containing a series of strings. When a node is closed, its "Id" (a string) is added to $close[v]$.

4.4 Algorithms Added to Gephi

The following algorithms are all based on the computation of static shortest paths or foremost journeys and they were all implemented and added to Gephi. We will make use of them in the following Chapter to analyze the knowledge mobilization network *Knowledge-Net*.

- AllNodesClosenessForemost: Computes the foremost birth-adjusted closeness for all the nodes of the graph.
- AllNodesClosenessStatic: Computes the static closeness for all the nodes of the graph.
- AllNodesForemost: Computes all the foremost journeys for all the nodes of the graph.
- AllNodesStaticShortest: Computes all the static shortest paths for all the nodes of the graph.
- ClosenessForemost: Computes the foremost birth-adjusted closeness for the node chosen by the user.
- ClosenessStatic: Computes the static closeness for the node chosen by the user.
- CompareCloseness2: Computes the foremost basic closeness for all the nodes of the graph and ranks them based on this value. Then computes the static closeness for all the nodes of the graph and ranks them based on this value. Finally, compares the two rankings.
- CompareCloseness3: Computes the foremost birth-adjusted closeness for all the nodes of the graph and ranks them based on this value. Then computes the static closeness for all the nodes of the graph and ranks them based on this value. Finally, compares the two rankings.
- Foremost: Computes all the foremost journeys for the node chosen by the user.

- NetworkDiameterForemost: Computes the foremost eccentricity for all the nodes of the graph, the foremost radius of the graph and the foremost diameter of the graph.
- StaticShortest: Computes all the static shortest paths for the node chosen by the user.

4.5 How to Integrate an Algorithm to Gephi

To integrate an algorithm to Gephi, we must first download the source code from Gephi's website as well as the Netbeans IDE from Netbeans's website.

After opening Gephi's source code in the Netbeans IDE, there is a module template that we can use to create modules. A module is a container used to add algorithms to Gephi. It can contain one or more algorithms. In the template there are several files, but in order to add an algorithm, we only need to work on four files that can be found under "Source Packages": `X.java, XBuilder.java, XPanel.java, XUI.java`, where "X" must be changed to the name of the algorithm that is added.

`X.java` is where the algorithm will be implemented. This must be done in the method "public void execute(Graph graph, AttributeModel attributeModel)". The code for the algorithm's output can be written in the method "public String getReport()". `XBuilder.java` is the class that connects the four classes together. `XPanel.java` is where the panel accepting the user's input must be implemented. `XUI.java` is where we must write the code to specify where in the Gephi's interface we want the button to start our algorithm.

4.6 How to Use Gephi

Before explaining how to use Gephi, we will explain how to prepare the files containing the graph that we want to import into Gephi. As seen above, Gephi accepts several standard graph file formats. Among them, the spreadsheet format is one of the simpler ones, so we will use that one.

To import a graph into Gephi using the spreadsheet format, we need two .csv files,

one containing all the nodes and one containing all the edges. (Although .csv files are used here, this is the spreadsheet format and should not be confused with the CSV format which is completely different)

In the nodes file, we must have the following columns:

- Id: The Id of the node.
- Label: The label that will appear on top of the node in Gephi. Usually the Id and Label of a node are the same, but they can be different if wanted.
- Time Interval: The time interval (there can be several time intervals if wanted) during which the node exists in the time-varying graph.

We can add other columns with the names that we want if we want to add other attributes to the nodes.

In the edges file, we must have the following columns:

- Label: The label that will appear on top of the edge in Gephi.
- Source: The source of the edge in a directed graph or one end of the edge in an undirected graph. The data entered into this column must correspond to the Id of the nodes in the nodes file.
- Target: The target of the edge in a directed graph or the other end of the edge in an undirected graph. The data entered into this column must correspond to the Id of the nodes in the nodes file.
- Time Interval: The time interval (there can be several time intervals if wanted) during which the edge exists in the time-varying graph.
- Traversal Time: The traversal time of the edge in the time-varying graph.
- Distance: The distance of the edge in the static graph.

- Type: The type of the graph. The values of this column must be either "Directed", "Undirected" or "Mixed".

We can add other columns with the names that we want if we want to add other attributes to the edges.

Now we will show the steps to import a graph in Gephi and run an algorithm on it:

- Start Gephi
- Click on "New Project"
- Click on "Data Laboratory" (in the top left)
- Click on "Import Spreadsheet"
- Click on the "..." button and select the nodes file
- Click on "Open"
- In the drop-down list under "As table:", select "Nodes table"
- Click on "Next"
- Click on "Finish"
- Click on "Import Spreadsheet"
- Click on the "..." button and select the edges file
- Click on "Open"
- In the drop-down list under "As table:", select "Edges table"
- Click on "Next"
- Click on "Finish"
- Click on "Overview" to see the imported graph
- Click on the algorithm wanted (on the right)

Chapter 5

Experiments Setup

In this Chapter we describe the setting in which we operate and the various parameters related to the experimental study, explaining the choices of our design.

5.1 Knowledge-Net

In [11], knowledge mobilization (KM) is defined as the use of knowledge towards the achievement of goals. It is a concept used for social network analysis (SNA) in science research and innovation. The networks which are built on a knowledge mobilization network approach make more visible the relations among heterogeneous human and non-human individuals, organizational actors and non-human mobilization actors.

Knowledge-Net is one of these networks built on a knowledge mobilization network approach. It is made-up of one class of actors, with three sub-types: individual human and non-human actors, organizational actors, and non-human mobilization actors. These actors are associated according to one relation, "knowledge mobilization", in a one-mode network [11, 12].

Human and non-human individual actors include researchers, students, individual funders, individual policy-makers, nature (i.e., human tissue samples), and collaborators.

Organizational actors include governmental entities (e.g., scientific organizations, departments, and ministries), not-for-profit organizations, businesses, not-for-profit or private funding organizations, and non-governmental scientific organizations.

Non-human mobilization actors, the third type of actors, serve as the "glue that binds" the network actors. It is through mobilization actors that individual, organizational actors and mobilization actors associate. Examples of mobilization actors include laboratories, publications, citing publications, "clear language" research summaries, research projects,

presentations, media events/products, patents, journals, conferences, training opportunities, products (including procedures), new business ventures, and government policies, regulations, legislation, or programs. These mobilization actors are mediators that can enable multiple actors to mobilize explicit and tacit knowledge for a wide range of goals.

More formally, Knowledge-Net is a time-varying graph $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, where:

- $V \in \{$(individual human and non-human), (organizational), (non-human mobilization)$\}$
- E is the set of relations (individual human and non-human, non-human mobilization) or (organizational, non-human mobilization)
- $\mathcal{T} \subseteq \mathbb{T}$ is the lifetime of the system (in a mobilization network, it is expressed in years)
- $\rho : E \times \mathcal{T} \to \{0, 1\}$, called *presence* function, indicates whether a given edge is available at a given time (in a mobilization network, when a new node $v \in V$ is created, all the edges $e \in E$ the connect v to the existing nodes of $\mathcal{G}$ are created at the same time and stay there until the end)
- $\zeta : E \times \mathcal{T} \to \mathbb{T}$, called *latency* function, indicates the time it takes to cross a given edge if starting at a given date (in a mobilization network, it takes 0 unit of time to cross any edge)

5.2 Data Description

Our dataset, the network called Knowledge-Net, can be represented as a time-varying graph. The data was collected from 2005 to 2011. The details of the graph are shown in Table 5.1:

Table 5.1: Details of Knowledge-Net

Actor type	2005	2006	2007	2008	2009	2010	2011
HA	3	22	27	46	51	76	94
NHIA	0	3	6	9	9	9	15
NHMA	7	25	43	87	132	194	248
OA	0	5	5	9	9	9	9
Total	10	55	81	151	201	288	366

The graph starts as a small graph in 2005, and each year more nodes are added to the graph without any being removed. The different actor types are: Human Actors (HA), Non-Human Individual Actors (NHIA), Non-Human Mobilization Actors (NHMA) and Organizational Actors (OA) (which are also non-human). As we can see, there are a lot more non-human actors than human actors. In 2011, there are 272 non-human actors (15 NHIA + 248 NHMA + 9 OA), but only 94 human actors. The non-human actors include conference venues, presentations (invited oral, non-invited oral and poster), articles, journals, laboratories, research projects, websites, and theses. The human actors are composed of principal investigators, highly qualified personnel and collaborators.

5.3 Study Design

To analyze our dataset, we created different versions of it (more details will be given about these versions below) and ran the foremost closeness and static (shortest) closeness algorithms on each of them. We then ranked the nodes by their closeness values (from high to low) in both the foremost closeness and static closeness algorithms for each version. We then took the nodes with the highest foremost closeness ranks and looked for their ranks in the static closeness algorithm, and took the nodes with the highest static closeness ranks and looked for their ranks in the foremost closeness algorithm. The main goal of doing that was to find some special nodes, for example, a node with a high foremost closeness rank but a low static closeness rank, or a node with a high static closeness rank but a low foremost closeness rank.

To obtain the different versions of our dataset, we modified it in various ways. The

original dataset is in the interval [2005; 2011]. It has new nodes adding themselves to the graph each year, but all the nodes stay there from their birthdate until 2011 without ever disappearing and the edges connecting them also stay there without ever disappearing. To make the graph "more dynamic", we decided to make a version where all the edges are active for only one year from their birthdates.

The original dataset ("Full Network" FN) has three very important nodes (LAB-R, Roucou X and Grenier C) that were always in the top of every ranking. They were so important that all the other nodes seem unimportant when compared to them. So we decided to make a version where these three nodes were removed to be able to see the emergence of the other important nodes of the graph ("Most Important Removed" MIR).

We decided to run the foremost closeness algorithm using four different values for the traversal time of the edges. The first value used was 1, the default traversal time in a time-varying graph (using 1 as the traversal time of the edges in a time-varying graph is similar to using 1 as the weight of the edges in a static graph). The second value used was 0 to see if having no traversal time was relevant or not. The third value used was 1/12 (around 0.08). We decided to use that specific value because it was equivalent to one month in the context of our dataset and since our dataset lasted for seven years ([2005; 2011]), one month seemed to be a reasonable traversal time for the edges. The fourth value used was 1/365 (around 0.003). This value is equivalent to one day in the context of our dataset and was used because we wanted a very small traversal time different than 0.

We also decided to use two different definitions of foremost closeness (formula 3.6 and formula 3.9) for the algorithm to see the differences between them. As a reminder, formula 3.6 solves the problem of disconnected components and formula 3.9 solves the problem of the impact of the birthdate.

Table 5.2 shows all the different versions of the dataset on which the foremost closeness and static closeness algorithms were run. In the following, "disappearing" indicates that the edges exist for one year only, "never-disappearing" means that they exist until 2011, "Basic Closeness" indicates that formula 3.6 was used for the foremost closeness algorithm, "Birth-Adjusted Closeness" indicates that formula 3.9 was used for the foremost closeness

algorithm, "FN" means that the full network is considered and "MIR" means that the 3 most important nodes have been removed.

Table 5.2: The Different Settings Studied in the Thesis

Type of Closeness	Traversal Time	Edges Appearance	Network Used
Basic Closeness	0	never-disappearing	FN
Basic Closeness	0	disappearing	FN
Basic Closeness	0	disappearing	MIR
Basic Closeness	1/365	never-disappearing	FN
Basic Closeness	1/12	never-disappearing	FN
Basic Closeness	1	never-disappearing	FN
Birth-Adjusted Closeness	1	never-disappearing	FN

Chapter 6

Analysis of Temporal Closeness

In this Chapter we analyze the closeness of Knowledge-Net specifically during its lifetime from both temporal and static points of view. We decided to use the foremost closeness for the temporal analysis, and the traditional (static) shortest closeness was used for the static analysis. The focus is on the difference between static and temporal views, and on the hidden knowledge that temporal analysis can provide on top of the static analysis. For this purpose, we analyze Knowledge-Net in different classes of zero latency and non-zero latency while the lifetime of the edges varies from 1 year to infinity.

6.1 Basic Closeness with Zero Latency and Never-Disappearing Edges

Let us first consider the case where the latency is zero, and edges remain active for as long as the system exists after they are created. Table 6.1 shows the temporally high ranked actors accompanied by their static ranks in the lifetime $\mathcal{T} = [2005 - 2011]$ of the system:

As a first observation, we notice that, contrary to the static analysis, the temporal results categorize the nodes in very distinct categories. The categories, however, do not necessarily correspond to the static ones. Thus, considering time, we can observe drastic effects to the results of the analysis.

Interestingly, the three highest ranked nodes in the static version correspond to the highest category in the temporal analysis. The most important nodes that are shared in both static and temporal analysis are also the most connected nodes in the graph, both from the static and temporal points of view. Even the betweenness analysis of such nodes shows that they connect the most communities in the graph. Thus, they naturally are close to most nodes of the graph.

At the same time, the highest ranked category in the temporal analysis also corresponds to some nodes that fall far low in the static model. These differences are worth

Table 6.1: List of highest ranked actors according to temporal (resp. static) closeness in the lifetime [2005-2011], with zero latency and never-disappearing edges

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
2005C01	0	0.258865248	1	0.261274159	131
2005C02	0	0.258865248	1	0.261274159	131
2005IOPV01	0	0.258865248	1	0.352998066	46
2005OPC01	0	0.258865248	1	0.353339787	45
2005PPC01	0	0.258865248	1	0.353339787	45
2005V01	0	0.258865248	1	0.261087268	132
Goggin K	0	0.258865248	1	0.378630705	15
Grenier C	0	0.258865248	1	0.4138322	3
LAB-R	0	0.258865248	1	0.542347697	1
Roucou X	0	0.258865248	1	0.501373626	2
2006C01	1	0.257223397	2	0.262024408	128
2006CIP01	1	0.257223397	2	0.283825816	69
2006CIP02	1	0.257223397	2	0.285379203	66
2006IOPV01	1	0.257223397	2	0.352998066	46
2006IOPV02	1	0.257223397	2	0.352998066	46
2006P01	1	0.257223397	2	0.395021645	5
2006P02	1	0.257223397	2	0.354025218	44
2006PPC01	1	0.257223397	2	0.354713314	42
2006V01	1	0.257223397	2	0.261087268	132
2006V02	1	0.257223397	2	0.261087268	132

explaining, and they are more appealing for analysis than the nodes that share similar ranks in both static and temporal versions. Examples of such nodes are: 2005C01, 2005C02, 2005V01. As we know, in the case of either the static or temporal closeness analysis, if a node has a large degree, or if it is close to nodes with high closeness values, it will be ranked higher for closeness. A static observation of 2005C01, 2005C02 and 2005V01 clearly shows that they do not have a large number of links, and those links do not connect them to nodes with very good closeness ranks. Thus, as expected, their static closeness ranks are low. Their temporal closeness ranks, however, are higher because of two major reasons: *a*) the nodes appeared very early in the network, which gives them an advantage to be able to reach other nodes, and *b*) their neighboring nodes also have a very good closeness.

Later in this chapter, we extend our analysis in a progressive lifetime analysis manner, meaning that we compose the lifetime of the system in a way that it removes the effect

of birthdate in the importance of the nodes. In this way, we can analyze the effects of the aforementioned reasoning to understand why nodes become important with regards to closeness. In the meantime, we commence a detailed analysis of individual nodes to understand the specific situations that make them high ranked temporally, yet low ranked statically.

Finer Look Into the Results: To evaluate the results in more details, we focus the analysis on the three nodes 2005C01, 2005C02 and 2005V01, whose temporal and static closeness had a huge gap. Since all of them have exactly the same properties and have similar locations in the structure of the graph, we only consider the detailed analysis of 2005C01.

As time progresses, the edge connecting the node to the rest of graph remains unchanged while the graph grows larger with new edges and nodes added to it. This helps 2005C01 reach more nodes as the system grows over time. However, so far, this does not explain why the (temporal) foremost closeness is much greater than the static closeness. To understand this, we need to look at the graph from the point of view of 2005C01.

When this node is born, it can reach all the nodes that are born in 2005. As new nodes are added to the system at later times, 2005C01 has the ability to reach them, as well as the ability to reach the nodes that were born in 2005. In the case of foremost closeness, the earlier a node is born, the earlier it can reach its neighbors. Thus, a node born in 2005 can reach the nodes born in 2005 in the same year, but a node that is born in 2006 can reach nodes that are born in 2005 only in 2006, which is a year later.

In such a case, the nodes that are added to the system late are "punished" with a small foremost closeness value. Of course, this demotes the value of the temporal analysis as the foremost closeness value of a node can be predicted by the time that it joins the system. Plus, if we look more carefully, such systems, where edges never disappear, do not normally exist in reality. All the temporal networks usually have a determined lifespan for their nodes and edges. Thus, next, we measure the closeness when the edges of the system live for a certain period of time and then disappear. In the next section, let us examine whether limiting the lifetime of the edges changes the results.

6.2 Basic Closeness with Zero Latency and Disappearing Edges

In the concept of Knowledge-Net, it makes sense if we assume that edges normally disappear after a year. The reasoning behind this is that fast scientific achievements and advancement make the old discoveries less attractive and the scientific community tends to mobilize knowledge that is augmented rather than knowledge from old discoveries. Thus, the newer knowledge is more appealing to the scientific community than the older knowledge. We assume the lifetime of the edge to be one year based on the speed of advance in science and technology. The results are shown in Table 6.2:

Table 6.2: List of highest ranked actors according to temporal (resp. static) closeness in the lifetime [2005-2011], with zero latency and disappearing edges

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
2005C01	0	0.258865248	1	0.261274159	131
2005C02	0	0.258865248	1	0.261274159	131
2005IOPV01	0	0.258865248	1	0.352998066	46
2005OPC01	0	0.258865248	1	0.353339787	45
2005PPC01	0	0.258865248	1	0.353339787	45
2005V01	0	0.258865248	1	0.261087268	132
Goggin K	0	0.258865248	1	0.378630705	15
Grenier C	0	0.258865248	1	0.4138322	3
LAB-R	0	0.258865248	1	0.542347697	1
Roucou X	0	0.258865248	1	0.501373626	2
2006C01	1	0.249892873	2	0.262024408	128
2006CIP01	1	0.249892873	2	0.283825816	69
2006CIP02	1	0.249892873	2	0.285379203	66
2006IOPV01	1	0.249892873	2	0.352998066	46
2006IOPV02	1	0.249892873	2	0.352998066	46
2006P01	1	0.249892873	2	0.395021645	5
2006P02	1	0.249892873	2	0.354025218	44
2006PPC01	1	0.249892873	2	0.354713314	42
2006V01	1	0.249892873	2	0.261087268	132
2006V02	1	0.249892873	2	0.261087268	132

With this assumption, we see that the closeness of the system does not change from what we saw in the previous section. The analysis is still in favor of the older nodes rather than the new born ones. Older nodes have a chance to reach the nodes that are of the same age as them while the younger nodes might not be able to reach older nodes. For

example, in the case of 2005C01, within the lifetime of the system, this node has only a single edge which exists at the beginning of the system's lifetime, in 2005. Progressing in time, the node gets disconnected in 2006. The disconnection should clearly affects the closeness of the node in both the static and temporal versions as the node is totally isolated. However, since it is connected to LAB-R in 2005, which, along with a few highly connected nodes, is almost connected to all the other nodes without any intermediary, when 2005C01 reaches Lab-R in 2005, it basically can reach all the nodes of 2005 in one time step, and the rest of the nodes as soon as they are born. Basically, an older node needs to be connected to a node that is connected throughout the lifetime of the system, preferably one of the central nodes, to be able to reach the whole graph even if the node has only one connection.

6.3 Basic Closeness with Zero Latency, Disappearing Edges and the 3 Most Important Nodes Removed

Considering the above-mentioned observation, it is interesting to see whether we will see any differences in the results if we remove all three nodes that are linked to almost all nodes in the graph. The results are shown in Table 6.3:

The results show a drastic change from the previous observations. The first immediate observation is that the high ranked nodes are all in one temporal rank category, and there is only one high ranked node that is born in 2005. The removal of those aforementioned 3 nodes caused the graph to become highly disconnected in 2005, and also lose connectivity in 2006 and later time intervals. This is because most nodes born in 2005 got connected to each other through the 3 important nodes, and without those 3 nodes, the graph of 2005 contains many isolated nodes. Only a few nodes born in 2005 maintain connectivity in the years coming after 2005 and Goggin K is one of them, hence retains its high rank in the temporal version. What is important is that the static closeness gets affected dramatically as well because the nodes lose their shortcut connectivity passage and they have to travel a longer way to reach each other.

Although this analysis provided significant insights into the fact that the structural

Table 6.3: List of highest ranked actors according to temporal (resp. static) closeness in the lifetime [2005-2011], with zero latency, disappearing edges, and the 3 most important nodes removed

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
Goggin K	0	0.222642061	1	0.259363828	28
2006C01	1	0.222642061	1	0.180375753	134
2006CIP01	1	0.222642061	1	0.212206769	82
2006CIP02	1	0.222642061	1	0.215374034	76
2006P01	1	0.222642061	1	0.273673557	14
2006P02	1	0.222642061	1	0.203762083	100
2006PPC01	1	0.222642061	1	0.223249878	69
Bissonnette C	1	0.222642061	1	0.237798746	44
CD4(EGFP)	1	0.222642061	1	0.146430501	159
CIHR	1	0.222642061	1	0.283447612	9
Electron Microscopy Facility	1	0.222642061	1	0.226112056	67
FRSQ	1	0.222642061	1	0.315567918	1
GPI(EGFP)	1	0.222642061	1	0.150099918	156
J-01	1	0.222642061	1	0.219848564	73
J-02	1	0.222642061	1	0.167525766	149
J-06	1	0.222642061	1	0.194881109	112
J-10	1	0.222642061	1	0.173097779	143
LAB-Nichols	1	0.222642061	1	0.178650155	135
LAB-Singh	1	0.222642061	1	0.173476134	140
LAB-Stankova	1	0.222642061	1	0.173476134	140

properties of the graph over time play an important role in the closeness of the nodes, it also yields limited value to the overall analysis as there are still some highly connected nodes in the graph emerging as we remove their higher connected competitors from the analysis.

Considering the above-mentioned observations, when a node can reach all of its close and far neighbors, and neighbors of neighbors instantly (zero latency), its foremost closeness calculation is trivial. This is especially true for highly connected nodes (nodes that can reach all other nodes of the graph). Hence, we will analyze the graph with a latency greater than zero for its edges.

Non-zero latency would make a significant difference, as in addition to the jumps that happen on the nodes connecting nodes born at different times, the nodes that are born in the same time interval will also have some latency when reaching each other.

6.4 Basic Closeness with Non-Zero Latency and Never-Disappearing Edges

In this section, we mirror the analysis done in the previous section, but with a non-zero latency on the edges. In this case, we consider latency values of one day, one month and one year.

6.4.1 1-day Latency

Table 6.4 shows the temporally high ranked actors accompanied by their static ranks in the lifetime $\mathcal{T} = [2005 - 2011]$ of the system, with the latency equal to one day and never-disappearing edges:

Table 6.4: List of highest ranked actors according to temporal (resp. static) closeness in the lifetime [2005-2011], with the latency equal to one day and never-disappearing edges

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
LAB-R	0	0.247566679	1	0.542347697	1
Roucou X	0	0.247565533	2	0.501373626	2
Grenier C	0	0.247564386	3	0.4138322	3
2005OPC01	0	0.247564386	3	0.353339787	45
2005PPC01	0	0.247564386	3	0.353339787	45
2005IOPV01	0	0.247563813	4	0.352998066	46
Goggin K	0	0.247562094	5	0.378630705	15
2005C02	0	0.247559801	6	0.261274159	131
2005C01	0	0.247559801	6	0.261274159	131
2005V01	0	0.247559228	7	0.261087268	132
RP38(2006-2011)	1	0.245686714	8	0.400219298	4
2006P01	1	0.245685585	9	0.395021645	5
2006PPC01	1	0.245673166	10	0.354713314	42
2006P02	1	0.245670344	11	0.354025218	44
2006IOPV01	1	0.24566978	12	0.352998066	46
CD4(EGFP)	1	0.24566978	12	0.352998066	46
GPI(EGFP)	1	0.24566978	12	0.352998066	46
PrP(EGFP)	1	0.24566978	12	0.352998066	46
2006IOPV02	1	0.24566978	12	0.352998066	46
Volkov L	1	0.245658492	13	0.339534884	50

The improvements on the results are apparent even at first glance. First and foremost, we notice a larger number of distinct foremost closeness categories, and fewer nodes in each category. This shows more distinction between the importance of nodes in the temporal

version. The second observation revolves around the birthdates of the nodes that are considered very important. In this model, the birthdates of the important nodes are distributed more evenly. In other words, important nodes consist of nodes that are born in at least two time frames, which is twice as much as the results seen in the previous model.

The reason that we see a clear distinction between the foremost closeness of the nodes in this model can be explained by the use of the latency of the edges in the computation of the foremost closeness values. Let us consider the time interval [2005-2006] and the node 2005C01. This node can reach all the graph of 2005, but its time distance will be more than the time distance of LAB-R since 2005C01 has to reach LAB-R in one time step before it can reach the rest of the graph, while LAB-R can directly reach all the nodes of the graph. This explains the time delay of 2005C01, and hence the distinction between the foremost closeness values of 2005C01 and LAB-R.

Considering such time delays shall create resemblance between the results that we observe for static shortest closeness and temporal foremost closeness since every hop contributes to the distance, temporal or static, that exists between the nodes. However, we still see huge gaps between the static and temporal closeness ranks of nodes like 2005C01, 2005C02 and 2005V01.

Again, due to the similarity of those nodes, we only focus on analyzing 2005C01 since we can use the same analysis for the other nodes. In the static analysis of 2005C01, as explained before, the node reaches the rest of the graph in more steps than most of the other nodes because it has only one edge, which limits its reachability to the rest of the graph. Hence, its static closeness value becomes lower than other nodes of the graph. However, in the temporal view, this node is still an early player in the system, so it benefits from being able to reach more nodes in an early fashion. Thus, similar to the previous model, the early birthdate of the node plays an important role in the foremost closeness computation.

6.4.2 1-month and 1-year Latencies

Repeating the analysis for latencies greater than one day does not create a significant change in the overall results gained from the closeness analysis, while the ranks change only slightly.

Table 6.5 and Table 6.6 show the temporally high ranked actors accompanied by their static ranks in the lifetime $\mathcal{T} = [2005 - 2011]$ of the system, with the latency equal to one month and one year respectively and never-disappearing edges:

Table 6.5: List of highest ranked actors according to temporal (resp. static) closeness in the lifetime [2005-2011], with the latency equal to one month and never-disappearing edges

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
LAB-R	0	0.239571621	1	0.542347697	1
Roucou X	0	0.239538973	2	0.501373626	2
Grenier C	0	0.239506334	3	0.4138322	3
2005OPC01	0	0.239506334	3	0.353339787	45
2005PPC01	0	0.239506334	3	0.353339787	45
2005IOPV01	0	0.239490018	4	0.352998066	46
Goggin K	0	0.239441082	5	0.378630705	15
2005C02	0	0.239375866	6	0.261274159	131
2005C01	0	0.239375866	6	0.261274159	131
2005V01	0	0.239359568	7	0.261087268	132
RP38(2006-2011)	1	0.237499931	8	0.400219298	4
2006P01	1	0.237467845	9	0.395021645	5
2006PPC01	1	0.237115471	10	0.354713314	42
2006P02	1	0.237035532	11	0.354025218	44
2006IOPV01	1	0.237019551	12	0.352998066	46
CD4(EGFP)	1	0.237019551	12	0.352998066	46
GPI(EGFP)	1	0.237019551	12	0.352998066	46
PrP(EGFP)	1	0.237019551	12	0.352998066	46
2006IOPV02	1	0.237019551	12	0.352998066	46
Bissonnette C	1	0.236700375	13	0.385835095	9

Table 6.6: List of highest ranked actors according to temporal (resp. static) closeness in the lifetime [2005-2011], with the latency equal to one year and never-disappearing edges

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
LAB-R	0	0.149379577	1	0.542347697	1
Roucou X	0	0.149104223	2	0.501373626	2
Grenier C	0	0.148829882	3	0.4138322	3
2005OPC01	0	0.148829882	3	0.353339787	45
2005PPC01	0	0.148829882	3	0.353339787	45
2005IOPV01	0	0.14869309	4	0.352998066	46
Goggin K	0	0.148284218	5	0.378630705	15
2006P01	1	0.144832661	6	0.395021645	5
RP38(2006-2011)	1	0.144444716	7	0.400219298	4
2006PPC01	1	0.141167611	8	0.354713314	42
2006P02	1	0.140676593	9	0.354025218	44
2005C01	0	0.140676593	9	0.261274159	131
2005C02	0	0.140676593	9	0.261274159	131
2005V01	0	0.140554372	10	0.261087268	132
2006IOPV01	1	0.140432363	11	0.352998066	46
CD4(EGFP)	1	0.140432363	11	0.352998066	46
GPI(EGFP)	1	0.140432363	11	0.352998066	46
PrP(EGFP)	1	0.140432363	11	0.352998066	46
2006IOPV02	1	0.140432363	11	0.352998066	46
Bissonnette C	1	0.13710007	12	0.385835095	9

6.5 Birth-Adjusted Closeness with 1-year Latency and Never-Disappearing Edges

As explained before, the effects of early birth can be removed from the analysis. For this last case, we rerun the analysis while removing the effects of the early birth of the nodes, the latency is equal to one year and we have never-disappearing edges. While the normalization based on the birthdate has a small effect, it creates more discrepancy between the temporal ranks when compared to the results of the previous model. The birthdate normalization nullifies the effect of being introduced to the graph earlier. Thus, we can measure the real activeness of the nodes by analyzing how much they participate in edges creation and access highly connected nodes. This is very important in the analysis of the activity of nodes. The results are shown in Table 6.7:

Table 6.7: List of highest ranked actors according to temporal (resp. static) birth-adjusted closeness in the lifetime [2005-2011], with the latency equal to one year and never-disappearing edges

Node	Birth date	Foremost closeness value	Foremost closeness rank	Static closeness value	Static closeness rank
LAB-R	0	0.449383562	1	0.542347697	1
Roucou X	0	0.44690078	2	0.501373626	2
Grenier C	0	0.444445281	3	0.4138322	3
2005OPC01	0	0.444445281	3	0.353339787	45
2005PPC01	0	0.444445281	3	0.353339787	45
2005IOPV01	0	0.443227622	4	0.352998066	46
Goggin K	0	0.439614354	5	0.378630705	15
2006P01	1	0.420202811	6	0.395021645	5
RP38(2006-2011)	1	0.41695382	7	0.400219298	4
2006PPC01	1	0.390768314	8	0.354713314	42
2006P02	1	0.387028905	9	0.354025218	44
2006IOPV01	1	0.38518591	10	0.352998066	46
CD4(EGFP)	1	0.38518591	10	0.352998066	46
GPI(EGFP)	1	0.38518591	10	0.352998066	46
PrP(EGFP)	1	0.38518591	10	0.352998066	46
2006IOPV02	1	0.38518591	10	0.352998066	46
2005C01	0	0.378871387	11	0.261274159	131
2005C02	0	0.378871387	11	0.261274159	131
2005V01	0	0.377986173	12	0.261087268	132
2007P01	2	0.370201561	13	0.367943548	24

Within this model, we observe that normalization based on the birth year of the nodes affects the results of the analysis, yet the nodes that appear in the system earlier still have the advantage over the other nodes. This advantage is more affected by the graph structure over time as 2007P01, which is born in 2007, joins the other 2006 and 2005 nodes as an important node. 2007P01 has a very strategic location in the graph as it sits in a place that reaches most connected nodes and has its own connections to other nodes as well. Being connected to all three most important nodes gives 2007P01 an advantage since it falls in between all communities existing in 2007. Hence, it becomes close to all nodes of that time. Being highly connected, compared to its counterparts, makes it more important than the other nodes of the same age group. In the meantime, nodes such as 2005C01 that are important temporally mainly due to appearing early in the system, fall more behind in the birthdate normalized version.

Moreover, although normalizing the birthdate causes the temporal and static closeness

analysis results to become closer in term of values, it helps the temporal analysis to become more realistic and independent of the birthdate of the nodes. Thus, a node with a high closeness value in the temporal version gains its importance from being structurally and temporally well located rather than simply being born earlier.

6.6 Summary

In this Chapter, we proposed the use of a temporal closeness measure to analyze a knowledge mobilization network that had already been studied using classical static parameters and temporal betweenness metrics. Our goal was to see the impact on the perceived static central nodes when employing a measure that explicitly takes time into account. We observed interesting differences. In particular, we witnessed the importance of being introduced early to the system in the temporal version. Our interpretation is that the earlier a node joins the system, the earlier it contributes to the mobilization flow in the network. However, the structure of the network and how it evolves over time play a vital role in increasing the importance of the nodes that appear later in the network. Such nodes, which are younger, but timely and structurally important, can remain undetected when the analysis is performed statically. The combination of static and temporal closeness can be used to provide insights on the importance and role of nodes in a network.

A temporal network analysis as performed here is especially pertinent for knowledge mobilization researches since that allows them to take time into account to understand the impact of academic researches beyond the narrow short-term context of academia. Measures of temporal closeness, as studied in this chapter, can provide researchers and funders with critical tools to more confidently investigate the role of specific mobilization actors for short and long-term impact within and beyond academia. However, it is important to mention that the knowledge mobilization network studied here was a test bed for the foremost closeness model and that such model is generalizable for a variety of other domains.

Chapter 7

Conclusions

In this thesis we presented some work done on temporal graphs. In particular, we focused on time-varying graphs, one of the many formal definitions of temporal graphs. We talked about journeys and distances in time-varying graphs. Then we considered the benefit of using time-varying graphs to analyze networks, in particular social networks.

For our temporal analysis, we implemented some temporal algorithms and added them to Gephi, an open source software for graph and network analysis which already contained some static algorithms. We then imported the network called Knowledge-Net into Gephi and compared the temporal results to the static ones.

We created our own variations of foremost closeness and used these variations on the network Knowledge-Net. The variations created are: our "basic closeness" which takes into account the disconnections happening in time-varying graphs for the computation and our "birth-adjusted closeness" which removes any advantages a node may have gained from its birthdate.

We also created different variations of Knowledge-Net, changing the time it takes to go through the edges, changing the amount of time the edges remain actives and removing the 3 most important nodes.

For the version "Basic Closeness with Zero Latency and Never-Disappearing Edges", we saw that some nodes had a low static rank, but a high temporal (foremost) rank. This was explained by the birthdates and neighborhood of these nodes. Since those nodes were born very early, they gained a temporal advantage. Furthermore, they were connected with nodes that had high temporal closeness values which also helped.

For the version "Basic Closeness with Zero Latency and Disappearing Edges", there wasn't a big difference when compared to the version "Basic Closeness with Zero Latency and Never-Disappearing Edges". The reason was the zero latency when traversing the edges. So, although the edges disappeared after one year, this was enough for a node to

reach a lot of the nodes born the same year, then it could wait for the next year's edges to reach other nodes, and so on. Therefore, nodes born early still had a temporal advantage since nodes born later weren't able to reach the ones born before them.

For the version "Basic Closeness with Zero Latency, Disappearing Edges and the 3 Most Important Nodes Removed", we saw major differences in both the static and temporal rankings when compared to the two previous versions. The reason of the static difference was because the 3 most important nodes were responsible of most of the connections in the graph, so when they were removed along with all the edges connected to them, the structure of the graph greatly changed, which impacted the static closeness of most nodes. The reason of the temporal difference was because most of the nodes born in the first year, which were in the top of the two previous versions, were connected with each other through the 3 most important nodes. So, when these 3 nodes were removed, most of the nodes born the first year became completely disconnected and therefore lost the temporal advantage they had gained in the two previous versions.

For the version "Basic Closeness with Non-Zero Latency and Never-Disappearing Edges", we had three variations. The first one had a 1-day latency, the second one had a 1-month latency and the third one had a 1-year latency. For all three variations, we got similar results. When compared to the previous versions, we had more ranks in the temporal ranking of the nodes and fewer nodes with the same rank. This was explained by the inclusion of the latency in the computation of the temporal closeness of the nodes. But the nodes born earlier still had a temporal advantage.

For the version "Birth-Adjusted Closeness with 1-year Latency and Never-Disappearing Edges", although the nodes born earlier still had high ranks in the temporal ranking, some nodes born later joined them as highly ranked nodes. The reason behind this was that these nodes born later were temporally connected to a lot of nodes and, at the same time, the nodes born earlier lost some ranks because we removed the temporal advantage gained from the birthdate.

For the analysis done in this thesis, we only considered foremost closeness for the temporal version of closeness. This analysis could be expanded to also include the temporal shortest closeness and fastest closeness. In fact, every static metric involving shortest paths could have a temporal equivalent with three variations (temporal shortest, foremost and fastest).

Temporal graphs have only been studied recently and one of the problems is the lack of a formal definition accepted by all. Different groups of researchers will have different definitions for temporal graphs. Since this is a computer science concept, it would be good to have some consistency.

There is still a lot that can be done on temporal graphs since we could have a temporal version of graph theory and study all the problems in graph theory in the temporal context. Some of these problems were already solved in the temporal context, but most have not been studied yet.

Bibliography

[1] F. Amblard, A. Casteigts, P. Flocchini, W. Quattrociocchi, N. Santoro. On the temporal analysis of scientific network evolution. In *Proceedings of Int. Conference on Computational Aspects of Social Networks* (CASoN), pages 169-174, 2011.

[2] A. Afrasiabi Rad, P. Flocchini, J. Gaudet. Tempus Fugit: The Impact of Time in Knowledge Mobilization Networks. In *Proceedings of 1st Int. Workshop on Dynamics in Networks (DyNo 2015), Workshop of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (ASONAM), 2015.

[3] S. Bhadra and A. Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Proceedings of 2nd International Conference on Ad Hoc, Mobile and Wireless Networks* (ADHOC-NOW), pages 259–270, 2003.

[4] B. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Comp. Science*, 14(2):267–285, 2003.

[5] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility. In *Proceedings of 5th IFIP Conference on Theoretical Computer Science*(TCS), pages 111–124, 2010.

[6] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Measuring temporal lags in delay-tolerant networks. *IEEE Trans. Computers* 63(2): 397-410, 2014.

[7] A. Casteigts, P. Flocchini, W. Quattrociocchi, N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387-408, 2012.

[8] Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3): 590-614, 2002.

[9] A. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proceedings of 27th ACM Symposium on Principles of Distributed Computing* (PODC), pages 213–222, 2008.

[10] P. Flocchini, B. Mans, and N. Santoro. Exploration of periodically varying graphs. *Theoretical Computer Science*, 469: 53-68, 2013.

[11] J. Gaudet. It takes two to tango: knowledge mobilization and ignorance mobilization in science research and innovation. *Prometheus*, 31(3): 169-187, 2013

[12] J. Gaudet. The "Mobilization-Network" Approach for the Social Network Analysis of Knowledge Mobilization in Science Research and Innovation. *uO Research, PrePrint*, 2014.

[13] P. Grindrod and M. Parsons. Social networks: Evolving graphs with memory dependent edges. Technical report, MPS_2010-02, University of Reading, 2010.

[14] F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7):79–88, 1997.

[15] P. Jacquet, B. Mans, and G. Rodolakis. Information propagation speed in mobile and delay tolerant networks. *IEEE Transactions on Information Theory*, 56(10):5001–5015, 2009.

[16] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (SIGCOMM), pages 145–158, 2004.

[17] E.P.C. Jones, L. Li, J.K. Schmidtke, and P.A.S. Ward. Practical routing in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 6(8):943–959, 2007.

[18] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of 32nd ACM Symposium on Theory of Computing* (STOC), page 513, 2000.

[19] A. Keränen and J. Ott. DTN over aerial carriers. In *Proceedings of 4th ACM Workshop on Challenged Networks*, pages 67–76, 2009.

[20] Kossinets, G., Kleinberg, J., Watts. The structure of information pathways in a social communication network. *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (SIGKDD), pages 435-443, 2008.

[21] V. Kostakos. Temporal graphs. *Physica A*, 388(6):1007–1023, 2009.

[22] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of 42nd ACM Symposium on Theory of Computing* (STOC), pages 513–522, 2010.

[23] F. Kuhn, and Y. Moses, and R. Oshman. Coordinated consensus in dynamic networks. In *30th ACM symposium on Principles of Distributed Computing* (PODC), pages 1–10, 2011.

[24] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *Mobile Computing and Communications Review*, 7(3):19–20, 2003.

[25] C. Liu and J. Wu. Scalable routing in cyclic mobile networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(9):1325–1338, 2009.

[26] M.E. Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1): 39-54, 2005.

[27] R. O'Dell, R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the 2005 joint Workshop on Foundations of mobile computing* (DIALM-POMC), pages 104-110, 2005.

[28] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, and F. Amblard. Time-varying graphs and social network analysis: Temporal indicators and metrics. *3rd AISB Social Networks and Multiagent Systems Symposium* (SNAMAS), pages 32–38, 2011.

[29] A. Scherrer, P. Borgnat, E. Fleury, J. L. Guillaume, and C. Robardet. Description and simulation of dynamic mobility networks. *Computer Networks*, 52(15):2842–2858, 2008.

[30] T. Spyropoulos, K. Psounis, and C.S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of ACM Workshop on Delay-Tolerant Networking*, page 259, 2005.

[31] J. Tang, M. Musolesi, C. Mascolo, V. Latora. Temporal Distance Metrics for Social Network Analysis. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Online Social Networks* (WOSN09), 2009.

[32] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora. Small-world behavior in time-varying graphs. *Physical Review E*, 81(5):55101, 2010.

[33] C. Tantipathananandh, T. Berger-Wolf, D. Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (SIGKDD), pages 717-726, 2007.