REPORT

Define a time-dependent social graph

- project deliverable 4.2

Authors: Barbara Guidi, Andrea Michienzi and Laura Ricci; Chrysanthi Iakovidou and Symeon Papadopoulos.

Confidentiality: *Public*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825585.

Define a time-dependent social graph	
Project name HELIOS	**Grant agreement #** 825585
Author(s) Barbara Guidi, Andrea Michienzi, Laura Ricci (UNIPI); Chrysanthi Iakovidou, Symeon Papadopoulos (CERTH).	**Pages** 2+34
Reviewers Kristina Kapanova (TCD), Javier Marín Morales (UPV)	
Keywords Online Social Networks, Decentralized Online Social Networks, Ego Networks, Multilayers Social Networks, Temporal Networks	**Deliverable identification** D4.2

Summary

This deliverable presents the modeling of the Heterogeneous Social Graph using a temporal network. The document describes what the Heterogeneous Social Network Graph is and the importance to model it using a graph formalism that includes the temporal complexity. The document furthermore introduces the Contextual Ego Network as the local view of each user of the whole Social Overlay. Thanks to the Contextual Ego Network, each user has to maintain information that is local instead of the whole graph, and the Social Overlay is maintained by the means of the union of each local view. The Contextual Ego Network is a multi-layer network where each layer represents a specific context of the user, and each layer is modelled by exploiting the ego network social model. The document moreover provides an overview of some possible studies that can be carried out on each layer, with a special focus on the usage of Graph Neural Networks.

Confidentiality	Public

30.9.2019
Written by

Andrea Michienzi

Project Coordinator's contact address
Ville Ollikainen, ville.ollikainen@vtt.fi, +358 400 841116

Distribution
HELIOS project partners, subcontractors, the Project Officer and HELIOS web site

Contents

List of Acronyms

Acronym	Description
API	Application Programming Interface
CEN	Contextual Ego Network
DOSN	Decentralized Online Social Network
DHT	Distributed Hash Table
GNN	Graph Neural Network
GPU	Graphics Processor Unit
HSG	Heterogeneous Social Graph
HTVG	Helios Time-varying graph
LSTM	Long Short-Term Memory
MANET	Mobile Ad Hoc Network
MNN	Multilayer Neural Network
NAT	Network Address Translator
OSN	Online Social Network
P2P	Peer-to-Peer
TVG	Time-varying Graph

1 Introduction

Online Social Networks (OSNs) have become the main channel to interact with people. The centralized nature of these platforms can be considered the primary problem as concerns the user privacy exposure.

Decentralized Online Social Networks (DOSNs) have been proposed as a valid alternative to a centralized solution. Indeed, a DOSN [1] is an Online Social Network implemented on a distributed information management platform, such as a network of trusted servers, Peer-to-Peer (P2P) systems or an opportunistic network.

During the last years, DOSNs have been the focus of several works and projects from both academic researchers and open source communities. By decentralizing OSNs, the concept of a service provider is changed, as there is no single provider but a set of peers that take on and share the tasks needed to run the system.

Usually, DOSNs are built by exploiting P2P networks. A P2P Network is defined in [2] as “A distributed network architecture may be called a Peer-to-Peer network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, etc…). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration); they are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept).” Considering the social nature of DOSNs, they are modelled by exploiting specific social network models, such as ego networks.

Furthermore, in a DOSN, the temporal dynamics are considered hard challenges, which require important design choices and specific measures and algorithms in order to better evaluate the real properties of the network. For this reason, they are usually represented and studied as temporal networks. Indeed, all real networks (online or offline) evolve over time, either by adding or by removing nodes or links. In a DOSN we can observe two types of dynamisms: social and infrastructure. The social dynamism concerns social relationships which can change over time, due to the variation of relations between users (edge dynamism), and of the number of users of the DOSN (node dynamism). Indeed, the social network of each user can change by adding or by removing nodes or links over time. This kind of dynamism is present also in centralized OSNs [3], but in DOSNs it has an impact on the structure of the underlying P2P overlay because communication links can be added or removed.

The infrastructure dynamism is related to the structure of the P2P social overlay network. Indeed, nodes may join/leave the underlying network (node churn), causing the number of available connections change in terms of the number of active links. Node churn is a real problem in distributed systems, especially in the scenarios where one cannot make assumptions about the availability and the reliability of the nodes composing the P2P networks. DOSNs, among others such as Mobile Ad Hoc Networks or Vehicular Ad Hoc Networks, is one of such scenarios. The fact that nodes may appear and disappear from the network, has side effects on the P2P topology, and can lead to situations in which the network may also be disconnected, thus making extremely difficult even the simplest problems. This kind of problem is even worse if we consider that nodes

may abruptly leave the network. This imposes the study of the problem of node churn, mainly in two directions: its effect on the P2P network, and possible ways to predict it. It is important to study the effect of node churn to be able to design a system which is able to recover from a situation in which it caused severe damage to the P2P network, and it is important to be able to predict it such that a more resilient network topology is created.

The study of churn in P2P networks is not an extremely recent research topic as, given its impact on the networks, was one of the first problems tackled since the early 2000s. Characterizing churn in P2P networks is also a very hard task because there are many different applications that can be built over these networks and each of them has a different usage pattern from the users. For instance, we expect few and long sessions in file sharing applications [4] but many and short sessions in DOSNs [5]. As such, the same application does not necessarily show uniform values. For instance, as summarized in [6], in Distributed Hash Tables (DHTs) we observe median session lengths ranging from a few minutes to up to one hour. A comprehensive study of churn is presented in [7], where the authors put their effort in defining some techniques to accurately measure churn mitigating some effects that may affect the results, such as: biased peer selection, handling brief events, Network Address Translation (NAT), and others. The study is carried out in terms of inter-arrival time distribution, session length, and several correlations. The problem of churn prediction is a well studied one in many fields, from economics [8] to telecommunications [9]. Also, a different number of techniques are used to perform the task, including decision trees [10], support vector machines and Random forest [11], but also social network analysis [12]. In the field of P2P networks, we find many works in the area of live video streaming [13–15].

DOSNs, as decentralized networks, are affected by the high infrastructure dynamism mentioned above and the social network is very shattered and not even close to the static view, as shown in [16]. Furthermore, the problem of churn can not be faced by exploiting the studies proposed for file sharing applications [7], because the time spent online is different. As studied in [5], in Facebook, users have less than 100 daily sessions while the average number of sessions is less than 4. Furthermore, almost 50% of users' sessions are usually shorter than 20 minutes, and only in few cases the session length exceeds the 2 hours.

For this reason, because of its ability to model an evolving network, a dynamic (or temporal) graph can be used as a model to represent the HELIOS P2P social network. Furthermore, specific design choices are necessary in order to be able to develop and analyse a DOSN.

HELIOS represents the new generation of decentralized social media platforms that addresses the dynamic nature of human communications in three dimensions: contextual, spatial and temporal. HELIOS will introduce novel concepts for social graph creation and management by exploiting trust and transparency, where the temporal aspect will be a property of the platform, and in particular of the social graph.

The rest of the document describes the feasibility to model the scenario of HELIOS using a temporal network formalism. We review several formalisms to identify the one that can be better adapted to our scenario, then we formalize the definitions of the Heterogeneous Social Network Graph and the Contextual Ego Network, given in D4.1, Heterogeneous Social Network Graph Topology and Lifecycle, in order to include the temporal dimension. After that, we show some studies performed in social networks in scenarios which are very close to ours. We conclude the document by presenting Graph Neural Networks as a model that leverages the Heterogeneous Social Network Graph's structural and temporal topology to perform machine learning, as well as a

promising decentralized formulation of Graph Neural Networks that propagates information through each user's Contextual Ego Network.

About this document

This document describes how a DOSN is affected by the time parameter and why. In detail, we present the main proposals in terms of temporal network models by describing their characteristics and how they are modelled. Among all the proposals, we decided to adopt the *Time-varying graphs* [17] because they are abstract enough to be implemented as needed, yet they are very expressive and extensible. In fact, starting from the original formalism, we decided to develop a new one called *HELIOS time-varying graph* which is heavily based on the *Time-varying graphs* with two major modifications. First of all, we drop all the expressive mechanisms which are not relevant or useful in our scenario, such that the formal model is left only with the necessary elements. This helped in making the formalism even lighter. Besides some minor modifications, we also needed to introduce a way to model the concept of context to our formalism. To this aim, we plug in the idea of layer of the Pillar multi-network [18] in the *HELIOS time-varying graph*.

With this formalism, we are able to redefine the Heterogeneous Social Network Graph (HSG), firstly introduced in D4.1 Heterogeneous Social Network Graph Topology and Lifecycle, where now we also add the temporal dimension to the model. The HSG can be seen as the social overlay of the whole platform HELIOS. The main goal of our dynamic overlay is to represent the social interactions of the HELIOS users by taking into account the dynamics of the decentralized systems (i.e. infrastructure dynamism).

We discuss that, both for practical reasons and for privacy reasons, we cannot expect that each user maintains a copy of the whole HSG. Therefore, we introduce a local view of the HSG, which is a concept much closer to the philosophy of the project. We call this local view the Contextual Ego Network (CEN). The CEN will be used as social overlay by each node independently, making it more than a mere stack of ego networks extracted from the HSG. Indeed, each user will build its own CEN, and each CEN will be updated by its own user independently. The Contextual Ego Network is managed by considering the dynamics of each layer, which represent a specific context of the user's daily life.

The document is organized as follows: in section 2 we present the state of the art concerning temporal networks. In section 3 we formalize the HSG, that is the social overlay of HELIOS, and the CEN, the local view of the nodes in the HSG. The section ends with some relevent studies that highlight the importance of studying the social overlay in a temporal fashion. Section 4 presents some requirements for the library that will be implemented for managing the CEN. In section 5 we discuss how to employ Graph Neural Networks considering the kind of tasks we have to perform and the distributed scenario in which the tool will be used. Section 6 concludes the document.

2 State of the art

In this section the reader is introduced to the concept of the temporal network as a graph model with the temporal dimension, and all the consequences. We start from a short survey overviewing some of the most important models proposed in the literature, and then we will choose the one that fits best our scenario. The section continues with a number of studies about how existing problems can be adapted to the dynamic case, and some new problems which are representative of temporal networks.

2.1 Temporal Networks

The most natural and used mathematical tool to represent social networks is the graph. In its basic form, a graph $G=(V,E)$ is defined by a set V of vertices and a set E of edges. An edge is defined as a pair of vertices $e=(u,v)$, $e \subseteq VxV$, for which $(u,v) \in E$ means that the nodes u and v are connected in G. The set V of vertices is used to model the entities, while the set E of edges is used to model the relations between these entities. In a typical social network, the set V is used to model the people and the set E can be used to model a number of different properties, such as interactions, degree of kinship, friendship and so on. Traditionally, social networks have been modelled using static graphs in which nodes represent people and edges between them represent a static relationship among pairs of people. But static graphs are not able to fully grasp the evolving nature of human interactions. Consider, for example, the scenario of OSNs, where people can freely add new friends or they can interact with people with whom they do not have a friendship. In a DOSN the effect is even more impactful because users may switch their state between online and offline. This is because in DOSNs the users themselves have to cooperate together to provide the service. Consider for example the problem of data availability, that is, making the data of each user available to all other users. One possible solution is to make replicas of the data, such that if a user is offline, the designated replica can provide the missing data. But then, what if also the designated replica becomes unavailable? Then even more advanced techniques need to be developed to cope with such situations. This kind of problem is absent in centralized OSNs, as the central server is in charge of delivering the service, and the fact that a user may go offline does not hinder the quality of the service. Moreover, a node going offline in a DOSN, also has side effects on the structure of the social graph made of online users. Things can get even more complicated in the world of mobile and opportunistic social networks, where people and their devices are able to communicate only with counterparts that are physically close. Temporal networks are suitable models for DOSNs because they do not consider a static, immutable graph, but instead are able to model the dynamism of these networks in all of their aspects.

The study of temporal networks is a relatively recent topic in the scientific community, however the presented models are already quite complex and some of them are able to model arbitrarily complicated scenarios.

A first model, called *Temporal graph*, is presented in [19] where the authors define the temporal network as a sequence of successive static networks. Given two nodes of the graph u and v, they define a *contact* between nodes u and v at time t, denoted with R_{uvt}, if an interaction happened at time t between the two nodes. They then define a time window w as the amount of time passed between two successive observations of the network. Thanks to these two concepts, they are able

to build a number of static (but time labeled) networks, each of which represents the state of the network once per *w* units of time. Each G_t, which contains information about the network from time *t* to time *t+w*, is made of the same set of nodes *V* as the static view of the network, and contains an edge *(u,v)* if and only if, there has been a contact R_{uvs} such that $t \leq s \leq t+w$. The temporal graph is now formally defined by the authors as the sequence $<G_t, G_{t+w}, \ldots G_{t+kw}>$ of each snapshot of the real network. A graphical representation of this formalism is shown in Figure 2.1. The main focus of the paper is then to use this formalism to study the temporal distance of the nodes of the network, that is how much time it takes to reach another node starting from an arbitrary node of the graph. The strength of this formalism is its simplicity and the fact that the problem they tackle is defined on a sequence of static graphs, thus enabling the possibility of using existing algorithms to solve the problem. A similar approach, called *evolving graph* [20], removes the constraint that each observation is made exactly once every *w*.

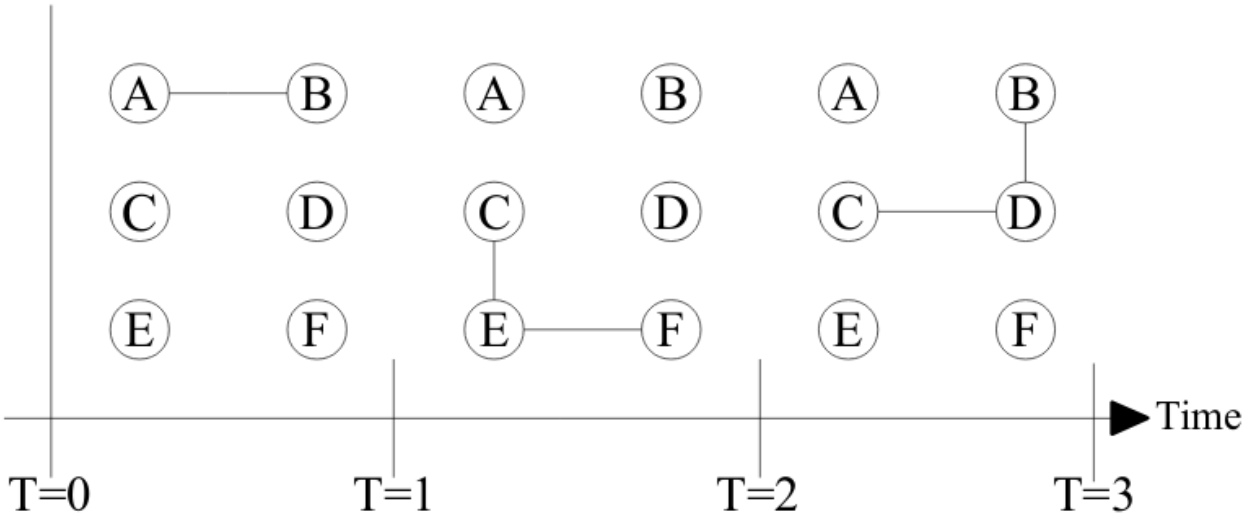

Figure 2.1 A graphical representation of *Temporal Graphs* taken from [19]. Time is sliced in time slots, and each slot contains a snapshot.

The model presented above is a very basic one and, in fact, it is not able to capture some more advanced dynamics. A slightly improved model, also called *Temporal graph* [21] abandons the concept of time window and snapshots to introduce a more granular model. The model was originally thought for email exchanges; however, it can be adapted to any kind of interactions, as long as it can be considered instantaneous. Interactions come in the form *(s,d,t)*, where *s* denotes the source of the interaction, *d* its destination and *t* the time at which the interaction took place. Even though the model was initially thought for directed interactions, the resulting graph can be symmetrized such that it becomes undirected. Starting from the list of interactions, a new graph is built in this way. For each interaction *(s,d,t)* two nodes are considered: s_t and d_t. These two nodes represent the two actors *s* and *d* respectively at time *t*. Between these two nodes, an edge respecting the direction of the interaction is created. Once all the interactions have been considered, all the nodes modelling the same entity are linked in a chain fashion using the time label they appear into. The structure created is once again a graph, but has two kinds of edges: the edges modelling the interactions and the edges modelling the flow of time (see Figure 2.2). This model is then used to introduce some topics, such as the notion of proximity. In addition to the

previous model, this one is able to provide all at once the whole history of the network, but besides that, it is not much more powerful.

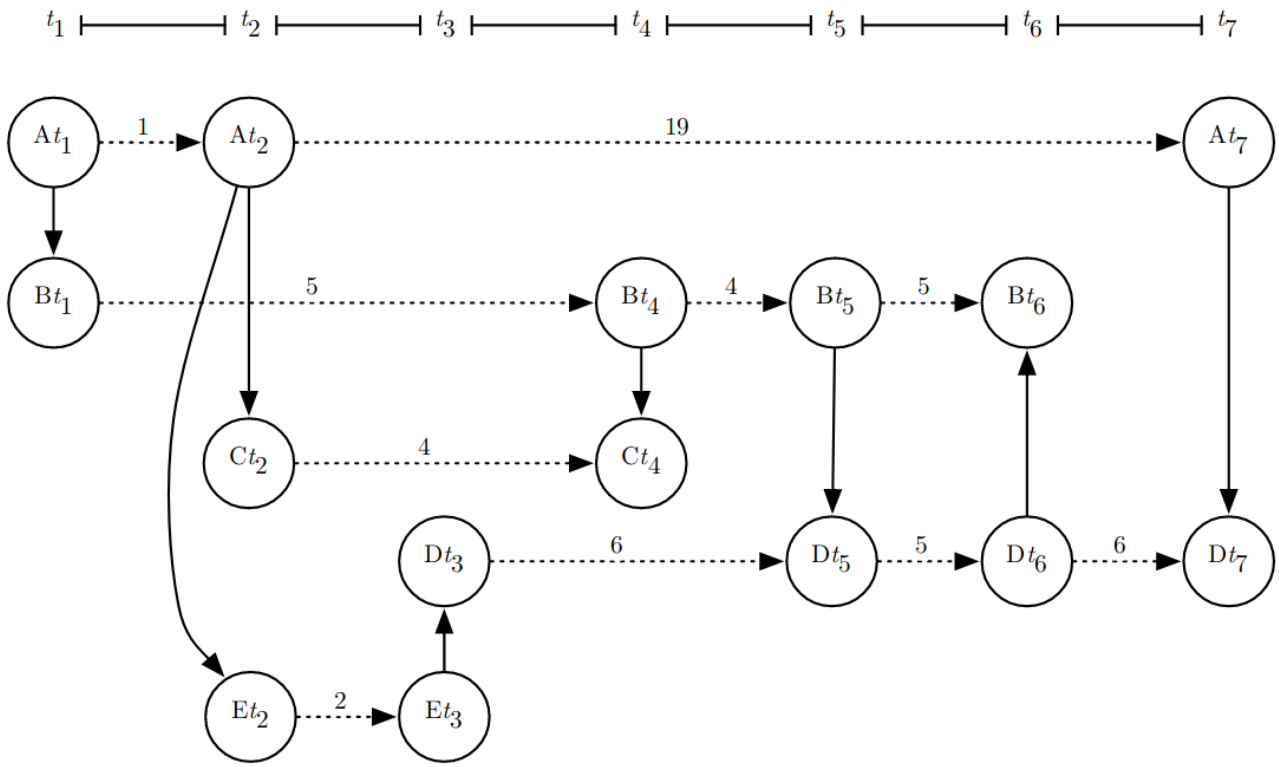

Figure 2.2. A graphical representation of the *Temporal Graphs* as presented in [21]. There are five actors in this toy example. Actor A appears only in three specific moments: t_1, t_2, and t_7, therefore node A appears three times. Horizontal arrows show how much time passes from two consecutive nodes modelling the same actor. Vertical arrows show that an interaction is happening at a specific time.

In the literature we also find some formalisms used for very specific use cases, such as the *temporal networks* introduced in [22] which are specifically thought for connectivity problems. In this model, edges are supposed to have two time labels: a departure time and a larger arrival time. Thanks to this double label, we can model the fact that interactions may not be instantaneous, but they have a starting time and a duration. This is most useful to model phenomena like phone calls or traversing a road network. In any case, if the arrival time is set as the same of the departure time for an edge, the authors assume that the interaction was instantaneous. To be able to infer some properties about time respecting paths in this scenario, the authors propose to add to the original network additional nodes. In detail, for each edge *(u,v,d,a)*, where *u* and *v* are the two interacting nodes, *d* is the departure time, and *a* is the arrival time, a new node *w* is added. Then, this new node is connected to *u* with a time label *d*, and to *v* with a time label *a*. The new node is therefore used to model the fact that we are waiting for that interaction to end before having the possibility to reach other portions of the network. Thanks to this artifice, the study of time respecting paths can be made simpler.

Another domain-specific proposal is the *periodically varying graph* introduced in [23] in which the authors focus on modelling trajectories which are repeated over time, such as public transportation routes, low earth orbiting satellites, security patrols or tourist tours. In this model we have a number

of sites, which can be visited, modelled as nodes, and a number of carriers, which are the entities that can visit the sites. Each carrier has a very specific route, defined as an ordered list of sites to be visited paired with an amount of time that the carrier will stay at each point. Each route defines a directed graph, and the sum of all the graphs induced by the routes is the so called periodically varying graph. In the rest of the paper, the authors investigate some different techniques to explore and infer knowledge over the graph.

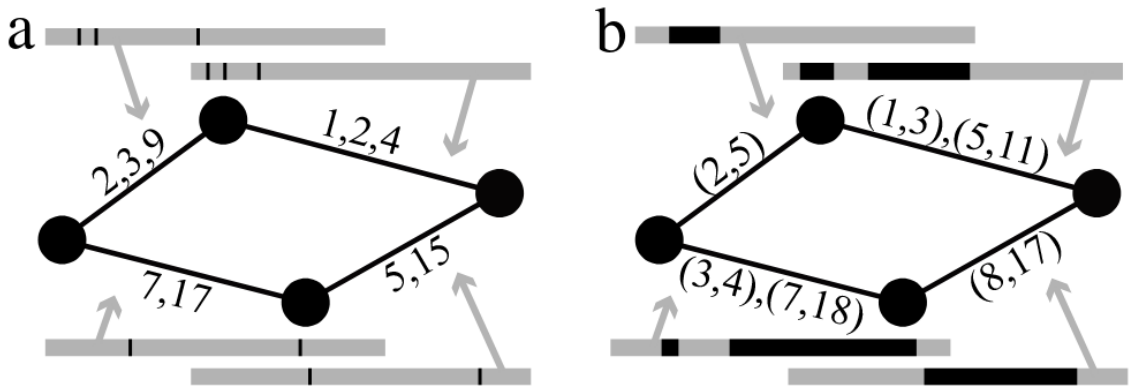

Figure 2.3. Figure a shows a graphical representation of an example of a *Contact sequence*, Figure b shows an example of graphical representation of an *Interval graph* [24]. In a contact sequence the edges are active only at specific time instants, while in an interval graph edges are active during time spans.

More abstract, general purpose and powerful models are the C*ontact sequences* and the I*nterval graphs* [24]. A contact sequence is made of a set of vertices which interact with each other at certain times, and the length of the interactions is negligible. The model is represented via a set of contacts in the form of *(s,d,t)*, where *s* and *d* are the source and the destination of the interaction respectively, and *t* is the time at which the interaction happens. The interval graph is just a generalization of the contact sequence, where the interactions have a duration. In this case a contact is represented with a quadruple *(s,d,t,δt)*, where the first three elements are the same as in a contact graph, and the last one is used to model the length of the interaction. A graphical representation of an example of the two models is presented in Figure 2.3, Figure a for the Contact sequence and Figure b for the Interval graph.

Stream graphs and *Link streams* [25] are similar to the interval graphs we presented above, but they also let us model the possible presence or absence of the nodes on the network. A stream graph is defined by a tuple $S=(T,V,W,E)$, where V is the set of nodes of the graph, T is the temporal domain, W is the set of temporal nodes, and E is the set of temporal edges. The temporal domain T is the set of valid temporal labels on which the temporal nodes and temporal edges are defined. The temporal domain can be either discrete or continuous and it is usually identified with $\mathbb{N}$ (the set of natural numbers), for the discrete case, or $\mathbb{R}^+$(the set of positive real numbers, including 0) for the continuous case. One can also consider intervals of these sets to model events that are bounded in duration. The set of temporal nodes $W \subseteq TxV$ is used to define the time spans for which nodes are involved in S. A node v is said to be involved in S at time $t \Leftrightarrow (t,v) \in W$. Finally, $E \subseteq TxVxV$ is the set of temporal edges which is used to define active edges in S. An edge between two nodes u and v is active at time $t \Leftrightarrow (t,u,v) \in E$. See Figure 2.4 for a graphical representation of a stream graph and a link stream.

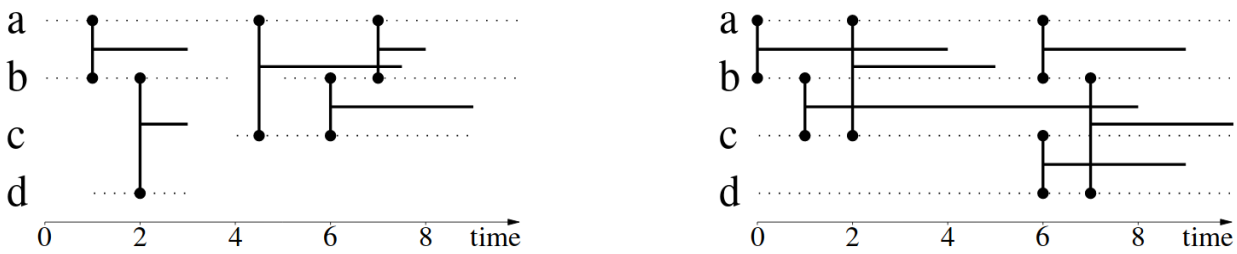

Figure 2.4. A graphical representation of examples of *Stream graphs* (left) and of *Link streams* (right) [25]. The dotted horizontal lines are used to denote when a node is available. In a Link stream (right) all nodes alre always avaiable, while in a Stream graph some nodes are available only during specific time spans. Vertical bold lines denote that an edge is becoming active, and the horizontal bold line is used to represent the time span it stays active. In a Stream graph, an edge may be active only if both ends of the edge are active.

Even more generic are the *Time-varying graphs (TVG)* [17], which are presented as a powerful and abstract framework, capable of modelling all the previous proposals. In its most generic form, a Time-varying graph $G=(V,E,L,T,\rho,\zeta,\psi,\varphi)$ is made of several elements. V is the set of vertices, which yet again are used to model the actors of the system. L is the set of domain-specific labels which can be assigned to the edges, and can be used to represent any possible information: from the strength of a tie using a real number to a string that describes the kind of relationship between the two actors. $E=VxVxL$ is the set of labeled edges between the nodes. Note that this kind of notation enables the presence of multiple edges between the same pair of nodes, as long as the label changes. T is called the lifetime of the system and can be understood as the span of time within which the relations happen. Most of the time, one can use the set of natural numbers $\mathbb{N}$ if the time is discretized, or the set of real numbers $\mathbb{R}^+$ if the time is continuous. The dynamics of the system are modelled using the four functions. The function ρ, called the presence function, is used to model the presence and absence of the edges on the network, while the function ζ, called latency function, is used to model the amount of time required to cross the edge. More formally, $\rho:ExT\rightarrow\{1,0\}$ and $\rho(e,t)=1$ means that the edge e is available at time t, while if $\rho(e,t)=0$ then the edge e is not available at time t. The latency function can be formally defined as $\zeta:ExT\rightarrow T$ and $\zeta(e,t_d)=t_c$ meaning that if we start crossing the edge e at time t_d, we need t_c amount of time to reach the destination node of the edge. The functions ψ and φ are the counterparts of the presence and latency functions respectively for the nodes. Formally, $\psi:VxT\rightarrow\{1,0\}$ and $\psi(v,t)=1$ represent that the node v is available at time t, but if $\psi(v,t)=0$ then the node v is not available at time t, while $\varphi:VxT\rightarrow T$ and $\varphi(v,t_d)=t_c$ means that if we reach the node v at time t_d, we need t_c amount of time to be able to move from that node. As we can see, both the presence function and the latency function (and their counterparts for the nodes) do not only take as input an edge, but also a time. This lets us model the fact that edges may be present or absent throughout time, but also the fact that their latency may vary over time.

Table 2.1 summarises the various formalisms, pointing out their main features: the possibility to model a discrete or continuous time, the possibility of having dynamic labels, and the possibility to model latencies on nodes and edges.

Formalism name	Time	Dynamics of labels	Latency of nodes	Latency of edges
Temporal graph [19]	Discrete	no	no	no
Evolving graph [20]	Discrete	no	no	no
Temporal graph [21]	Discrete	no	no	no
Temporal networks [22]	Discrete	no	no	yes
Periodically varying graph [23]	Discrete	no	no	no
Contact sequence / Interval graph [24]	Discrete	no	no	no/ yes
Stream graph / Link stream [25]	Discrete or Continuous	no	no	no
Time-varying graph [26]	Discrete or Continuous	yes	yes	yes

Table 2.1 Relevant properties of the various formalisms presented.

2.2 Storing techniques

Storing a complex structure such as a temporal network requires some expedient and, in some cases, very specific data structures. In the following, we briefly discuss some of the techniques already present in the literature. However, we immediately point out that there is no technique that works better than all the others in every case: each different scenario requires that the various possibilities are analysed, and a proper technique is selected afterwards.

A pretty straightforward technique is to discretize time and then build separate snapshots of the temporal graph, and to store the snapshots separately. In this way, each snapshot can be treated as a single graph, allowing storing techniques for static graphs to be used, such as an edge list, adjacency list or adjacency matrix. In the same scenario, more complex structures exist, such as

interval trees [26] and segment trees [27], which aim to store segments of the graph evolution in a tree structure for query efficiency reasons.

In [28] the authors identify two main techniques for storing temporal data, namely the “copy” approach and the “log” approach. The two approaches are opposite to each other. The copy approach is a generalization of the one presented above and consists of storing a copy of the whole graph each time it is needed, possibly once every time the graph changes. It is a generalization in the sense that it does not slice time in fixed length portions, but it rather decides certain points in time when to store the network. The copy approach has as a downside the fact that most of the data is possibly redundant, and therefore requires a lot of space. To overcome this issue, the log approach aims at minimizing the redundant information by storing only the changes that happen in the graph. This approach works very well if the changes on the network are extremely frequent, because only the part of the network that has changed will be stored. The technique also works much better if we are interested in limiting our studies in very small time spans, discarding the history of the network. However, this technique lacks an efficient way to rebuild the network at a given point in time, because it requires to read all the history up to that point. However, one can consider using a hybrid technique which uses the copy technique to build few “checkpoints” of the network that will be stored, and then use the log technique to store the evolution between these checkpoints.

2.3 Temporal network problems

In this section, we present several extensions of network problems in temporal scenarios that demonstrate the complexity and many variants that arise when we add a temporal dimension to graphs. To give a first insight, we will show the temporal equivalent of a well-known problem of graph theory: the shortest path. The problem of finding the shortest paths is a very basic and simple one and for this reason it is suitable for our aim. Moreover it can be used as a starting point to define more advanced problems, such as eccentricity and diameter, path based centralities (betweenness, closeness and harmonic), and even some community detection approaches. Additionally, we also show how to adapt more complex applications: community detection and motif detection.

2.3.1 Shortest paths and Journeys

Given a static graph, modelled as $G=(V,E)$, where V is the set of the entities modelled, and E is the set of relationships among them, we can define the so-called paths. A path P from $u \in V$ to $v \in V$ of length k is a sorted sequence of edges $P=\{(u_0, v_0), (u_1, v_1), \dots, (u_k, v_k)\}$, with $\forall\ 0 \leq i \leq k.\ (u_i, v_i) \in E$, such that $u_0=u$, $v_k=v$, and $\forall\ 0 < i \leq k.\ u_i = v_{i-1}$. In this static case, the length of a path can be defined as the number of edges composing the path. A path P of length k from u to v is said to be the shortest if there is no other path from u to v with length $l<k$. In an undirected and unweighted network, shortest paths can be found using a simple breadth-first [29] search from the source node u of the graph.

Things change dramatically when we have to include the temporal dimension in the problem. In this section we consider a simplified version of the problem, where edges may be traversed only during specific time spans. In particular, we will consider that nodes are always available and

visible, although they may be isolated or not connected to a portion of the graph at a given time, and that nodes and edges have no latencies. For this goal, we consider a temporal network using the Time-varying Graph (TVG) formalism, introduced earlier in this section, as $G=(V,E,T,\rho)$, where V is the set of nodes, T the temporal domain, E the set of edges, and ρ is the edge presence function. The notion of path on temporal networks is called journey, and the journey from node u to node v at time t can be defined as a sequence of pairs $J(u,v,t)=\{(e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)\}$, where $e_i \in E$ is a temporal edge, and $t \in T$ is the time at which the corresponding edge is crossed. Moreover it is also requested that $\forall\ e_i=(u_i, v_i)\ u_{i+1}=v_i,\ u_1=u,$ and $v_k=v$, such that the sequence of the edges actually makes a path from the source node to the destination node, $\forall\ 1<i\leq k\ t_i>t_{i-1}$ such that the sequence of edges is time-respecting, and finally $\forall\ 1\leq i\leq k\ \rho(e_i,t_i)=1$ such that the edges are available for crossing. A journey is characterised by at least three distances, one topological and two temporal. A journey $J(u,v,t)$ is said to have length k, because of the number of edges included in it, the duration is t_k-t_1, and time to reach equals to t_k-t. In this scenario, depending on what we want to minimize, we have three different versions of the problem: if we minimize the length we find the shortest journey, if we minimize the duration we find the fastest journey, and if we minimize the time to reach we find the foremost journey. Even more advanced versions of the problem may be defined as well. For instance, one may want to find the shortest journey with the constraint that the time to reach has to be lower than a given threshold. Even more than that, if we consider undirected edges, in the static case if there is a path from node u to node v, there is also a path from node v to node u. This does not necessarily hold for temporal networks, because it depends on how the edges appear and disappear.

As we saw in this section, even the simplest of the problems that can be defined on static graphs becomes very hard in a dynamic scenario, and usually more versions of the same problem can be defined. Nevertheless, this did not stop nor lessened the effort in defining measures and algorithms for studying temporal networks. Instead, it led to a proliferation of a very high amount of independent definitions of the various problems. In the following, some of them which attracted a lot of interest in the scientific community.

2.3.2 Community Detection

Community structure is a well investigated problem for complex networks in general [30], and the study of their structure could help to understand several properties of complex networks, including the ones of OSNs [31]. The problem of community detection is difficult per se, because it is ill-posed. In fact, there is no formal and widely accepted definition of a community, but at a high level it can be identified as the detection of sets of nodes that are closely related to each other, more than the other nodes of the graph. This problem was thoroughly studied in the literature [31–33].

Unfortunately, studying communities in a static way does not model well the evolving nature of the world we live in. Just consider, for instance, that human relationships continuously evolve over time (both in a smooth or in a disruptive way), interactions are limited in time, people switch continuously their context depending on their physical position and so on. Therefore, since OSNs are extremely dynamic environments, there is the need of studying communities as time evolves, which inevitably makes the problem more complex.

Recently, a lot of attention and effort has been placed on the dynamic counterpart of the same problem [34], [35]. In this case, it is also very hard to find a shared formal definition of the concept of a community. Intuitively, one can understand a dynamic community as an evolving set of nodes

which, at a given time, contains the set of nodes which are closely related at that time. Algorithms for dynamic community detection can be divided into three classes, as presented in [35]: Instant Optimal, Temporal Trade-off, and Cross-Time. These types of algorithms differ on the type of information used to determine the community structure at a given time.

In *Instant Optimal* algorithms, the network evolution is seen as a series of successive snapshots, each representing the state of the network at a particular instant of time. Communities are identified on each snapshot separately, using static communities detection approaches, and then matched together to build the complete history of each community. In this approach, communities existing at time t are discovered by considering only the state of the network at time t.

In *Temporal Trade-off* algorithms, the communities identified at time t are computed using the current state of the network, at time t, and previous information, possibly up to the initial known state. Typically, communities are discovered by a two steps process which consists of an initial bootstrap (with a static community detection algorithm), yielding the existing communities when the observation starts, followed by an iterative procedure that updates these initial communities as the network evolves.

Finally, *Cross-Time* algorithms use all available information, i.e. past, current and future with respect to time t, to identify communities at the instant t. Generally speaking, such methods find all the dynamic communities at once, instead of their state at a specific time. Moreover, the communities detected by the algorithms in this class are said to be temporally smoothed over time, meaning that they tend to produce more stable communities (nodes which are not unequivocally to be included in a community, are kept out).

2.3.3 Motif Detection

A network motif is a "pattern of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks", as defined in [36]. In other words, motifs are small subgraphs, usually made of few nodes, that can be found many times within a network. Motifs may reflect a specific behaviour of the actors modelled with the nodes of the graph and their interactions. For this reason, motifs can be understood as building blocks of complex networks and can uncover local properties of the networks as well as network-wide phenomena. The detection of motifs is a computationally hard problem because it involves graph isomorphism, and the computation times become extremely high as the size of the network and of the motif increases. In OSNs, the concept of motif can be used to find recurring patterns of interaction between users, but it can be also helpful to understand particular characteristics of the human behaviour, such as homophily.

Yet, as we already seen, static networks are not well suited for modeling OSNs and accordingly static motifs are not the correct tools to study human patterns of interactions. Therefore, the problem of motif detection also requires a temporal counterpart. A first, simplistic way to study motifs in a temporal network is to introduce a time discretization and divide the network in successive snapshots [37]. Then a static motif detection algorithm can be run on each snapshot of the network separately. In another relevant work [38] temporal motifs are defined as static motifs with the additional constraint that all the interactions happen within a fixed amount of time, thus removing the discretization of time. More advanced versions of the problem can also be defined, as in [39], where the interactions specified in the motif must also happen in a specific order.

3 Heterogeneous Social Network Graph and Contextual Ego Network formalizations

Many real-world phenomena we observe in everyday life can be modelled using graph theory. Traditionally, static graphs (i.e. graphs that do not change) have been used to model these phenomena because they are expressive enough but not too complex. However, in the last decades, new models were introduced to add a temporal dimension to the static graphs. This , in turn, produced more detailed models capable of describing more phenomena and more in detail. The flourishing of studies in this direction led to the development of a huge number of models, without having a specific formalism prevailing over the others. For the sake of clarity, we will refer to such models with the term “dynamic graphs” because of their ability to model a graph that shows some dynamics, opposed to the “static graphs” which do not change over time. The dynamic graph models can be focused on modeling the dynamics of the nodes, the dynamics of the edges, or they can have no specific focus and be expressive enough to model both dynamics. Dynamics of the nodes consist in considering that the nodes may change their state over time and that they may be present during a certain time span, but absent in other spans. Dynamics of the edges consist in the fact that the relation among the entities may evolve and change over time, but also the fact that some new relations can be established over time, while others disappear.

3.1 Helios Time-varying graph

Having stated the importance of modelling the social graph of a DOSN with a dynamic graph, we now establish how we can model the Heterogeneous Social Network Graph (HSG) with a temporal network formalism. Thanks to this overview about the main models for temporal networks found in the literature, we can now choose a suitable formalism for our scenario. We have decided to adopt the Time-varying graphs (TVG) [17] because they are abstract, general purpose and extensible. In particular, we need to adapt the formalism to our scenario, in which users are connected to each other based on their social ties, and see the relationships evolve over time either because they change or because users switch their state to online/offline. We will call this formalism HELIOS time-varying graph (HTVG).

The first change we need to make, concerns how we consider the labels within the edges. We recall, as defined in D4.1 Heterogeneous Social Network Graph Topology and Lifecycle, that in our scenario each edge is labeled with the tie strength existing between the two connected users. However, according to the original formalism of the Time-varying Graphs, an edge $e \in E \subseteq VxVxL$ is identified as comprising a source node, a destination node and a label. Now, if we use this formalism as it is, for each value of tie strength, we would have a different edge. To simplify the formalism, and make it a bit closer to our scenario, we simplify the formalism and decouple the edge from its label. So, in the HTVG an edge $e \in E \subseteq VxV$ is identified by its source node and its destination node. In addition to that, we still maintain the labels on the edges, although in a different form. Each edge will have an assigned label, but as long as the source and destination nodes of two edges match, we will consider the two edges as being the same edge. Since we will consider the labels as values attached to the edges, instead of having a set of labels L, we will consider having a labelling function. The labelling function λ of the HTVG is defined as $\lambda: ExT \rightarrow \mathbb{R}^+$ meaning that $\lambda(e,t)$ will return the tie strength of edge e at time t. Thanks to these adjustments, we

are able to model the fact that each pair of connected nodes has an associated tie strength that can change as the system evolves.

The other major change we have to make, is the one that lets us model the possibility for each user to have the so-called contexts. As we introduced in D4.1, Heterogeneous Social Network Graph Topology and Lifecycle, we use the concept multilayer network and, in particular, we will try to express the concept of the layer using a formalism similar to the one of Time-varying graphs. As each layer can be modelled using a graph, we will not make changes to the existing formalism. However, to model the various layers, we will consider the whole network as a collection of layers, and we will add some mechanism to identify nodes in different layers which model the same user. To cope with the need of this identification mechanism, we introduce a new function $\tau: V \times N \rightarrow V \cup \{\perp\}$ which we call translation function, which is a concept similar to the Node mapping [18]. In detail, $\tau_i(u,j)=v$ means that the node u in layer i and the node v in layer j represent the same user. The symbol $\perp$, also called "bottom", can be used to model the possibility that a node is present in a layer, but not in every layer. We say that $\tau_i(u,j)=\perp$ if and only if node u is not present in layer j, therefore it cannot be translated into an actual node. Each layer will have its own translation function, which, given a node, can find the corresponding node in each other layer.

3.2 Heterogeneous Social Network Graph formalization

At this point, using the HTVG, we take the definition of HSG given in D4.1, Heterogeneous Social Network Graph Topology and Lifecycle, and try to formalize it. The HSG $H=(V,E,L,T,\rho,\psi,\lambda)$, where V is the set of users in HELIOS, E is the set of relationships, L is the set of possible labels which in our scenario is the set of positive real numbers used to keep track of the tie strength between two users, T is the temporal domain, ρ and ψ are respectively the presence functions of the edges and nodes, and λ is the tie strength function. We dropped the edge and node latency functions ζ and φ, as we find no use for them in our scenario.

As we introduced in D4.1, Heterogeneous Social Network Graph Topology and Lifecycle, HELIOS considers a set of heterogeneous actors, which can be for instance people or smart objects, and the connections between such actors. The connections have a different nature and meaning that depends on the two actors involved. To model such a graph, we implement the Social Overlay with a *Heterogeneous Social Network Graph.* In the HSG, the nodes model the actors, such as people, smart objects and so on, and the edges model the (attributed) relationship among the actors.

3.3 Contextual Ego Network Formalization

Handling the HSG in a P2P environment such as the one of the DOSNs is a real challenge because there is no central server which stores the whole graph and we cannot expect each peer to store a copy as well. At the same time, we want to be able to have sufficient information in each peer such that the study of the graph can be helpful. Moreover, when talking about the data of the people, we also have to be sure that data cannot easily be propagated in the network to avoid privacy disclosures.

The HSG represents the Social Overlay Network of HELIOS. A Social Overlay is a logical overlay in which peers are connected to known peers and the meaning of an edge between two nodes is a social relation between them. In general, each node of a P2P network has a local view where only

a subset of nodes of the entire network are stored. In a DOSN, the local view of a node contains the friend nodes. However, each node cannot maintain a copy of the whole HSG, because it would be too big and it would disclose personal information of the users.

To overcome these problems, we need a distributed management of the social overlay, where each user stores just a small portion, possibly a portion that is relevant to itself. A possible approach is to a more local social overlay, such as the Ego Network (Figure 3.1). The *Ego Network* is a social network model widely used in literature to model the local view of the users in a DOSN. The Ego Network of a user is a structure built around the user itself, commonly called Ego, which contains her/his direct friends, called alters, and it also includes the connections ego-alters and alters-alters. Thanks to this social overlay, each user can only maintain information regarding its direct friends, making it viable for our scenario.

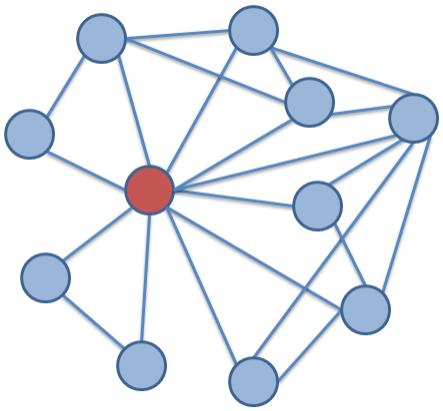

Figure 3.1: An example of ego network. The ego, represented with a red node, is connected to all the alters, represented with a blue node. In the ego network, besides the trivial relationship between ego and alter, there are also the relationship between the alters.

However, a plain and simple Ego Network cannot be used because it lacks two major concepts which are crucial to our scenario: the temporal aspect and the concept of context. To this aim, we take the concept of Contextual Ego Network (CEN) defined in D4.1, Heterogeneous Social Network Graph Topology and Lifecycle, as it captures all the aspects we want to include. We will now formalize the CEN using the HTVG. The CEN of a node u $CEN^u=\{(C^u_1, \tau^u_1) \dots (C^u_n, \tau^u_n)\}$ is made of n layers, which model the various life contexts of node u. The layers are made by a time evolving ego network and a translation function. A time evolving ego network can be seen as the ego network of a user that is capable of change and adapt as the environment around the user changes. A time evolving ego network of a layer can be formalized with $C^u_i=(V^u_i,E^u_i,L,T,\rho,\psi,\lambda)$, as a temporal network with the constraint of it to be the ego network of node u in its *i-th* context.

Modelling a DOSN with a dynamic network is not only a matter of having a more detailed model at our disposal, but thanks to it we can perform advanced and more accurate studies which take into account the time-dependent social overlay. One such study concerns the behaviour of users in the DOSN in order to address the infrastructure dynamism (node churn). More studies about time evolving ego networks will be presented in the next section.

3.4 Time evolving ego network: graph properties

The concept of ego network has been exploited in several fields for solving different kinds of problems.

For instance, a way to evaluate the betweenness centrality of the nodes of a graph using snowball sampling was proposed in [40]. In the work the authors try to show a correlation between the betweenness centrality of nodes and the ego network betweenness, namely the betweenness centrality of nodes in their ego network. While there is no theoretical link between the two rankings, the results show that there is actually a high correlation.

Another relevant work [41] employs the concept of ego network betweenness to the problem of routing in Mobile Ad hoc Networks (MANETs). Since connectivity and reachability is a real problem in MANETs, the authors propose an approach based on centrality and similarity to evaluate which is the better node to which a node must be forwarded to give it high probability to reach the destination node.

Concerning studies more related to the Online Social Networks scenario, in [42] the authors, by restricting their study to the scope of the ego network of each user, detected a not so trivial phenomenon. The results suggest that users of Online Social Network Facebook tend to have relationships that can be organized in a structure very similar to the one depicted by Dunbar for offline social networks [43]. An analogous result was obtained for the Twitter Online Social Network in [44], suggesting that this is a behaviour common to many of the most used Online Social Networks.

The literature shows some works that study the dynamic community structure in ego networks. In [16] a preliminary work about the community structure in ego networks of Online Social Networks user is presented, while in [45] a comparison between different methods of community detection in the same scenario is presented. Another relevant contribution [46] in the same field presents a protocol and an algorithm for dynamic community detection specifically thought for Decentralized Online Social Networks.

The concept of ego network was used also in the field of network visualization. In [47] new layouts for network visualization are developed which are specifically designed for an ego network. The major novelty introduced is the fact that these layouts take into account the temporal information and use it to build the visualization. A successive analysis on some real datasets uncovered the existence of some visual motifs that can be used to characterise recurring events in the ego networks.

4 Handling the Contextual Ego Network

Having defined how the Contextual Ego Network will look like, in a rather abstract way, there is still the need to define how this object will be handled and updated by each user. We will spend this entire section stating which are the functionalities and mechanisms that are needed, and debating some possible extensions. This is not meant to be an exhaustive or a definitive version of the APIs of the library which will be developed for managing the Contextual Ego Network, but rather an initial sketch, with some requirements emerged from this document.

4.1 Managing the Layers

We recall that we formalized the Contextual Ego Network of user *u* in section 3 as $CEN^u=\{(C^u_1, \tau_1) \dots (C^u_n, \tau_n)\}$, where C^u_i models the *i-th* context of user *u*, and τ_i is the translation function for the *i-th* context: a function that, given a node *u* in the *i-th* layer, is able to return the correspondent node in the other layers. Clearly, the layers of users are not static objects, and therefore they need to be manipulated accordingly. In particular, we foresee the need of mechanisms to create and modify the layers and its components.

For what concerns the creation, we need a function for creating a new context and a function for destroying a context:

Method name	**Short description**
createLayer(C^u_n, τ_n)	create a new layer for node *u* using the specified context and translation function
deleteLayer(i)	delete the *i-th* layer

Moreover, some specific functions to store and load layers from memory:

Method name	**Short description**
storeContextualEgoNetwork(location)	store the current state of the CEN in persistent memory at the specified location

loadContextualEgoNetwork(location)	load a CEN from the specified location in persistent memory
storeLayer(i, location)	store the *i-th* layer of the CEN to persistent memory at the specified location
loadLayer(i, location)	load the *i-th* layer of the CEN from the specified location in persistent memory

Finally, we also foresee that some methods to access the layers will be needed:

Method name	**Short description**
getContext(i)	return a reference to the *i-th* context such that it can be modified directly
getTranslationFunction(i)	return a reference to the *i-th* translation function such that it can be modified directly
getLayer(i)	return the *i-th* layer of the CEN, namely the *i-th* context and the *i-th* translation function
setLayerActive(i)	set the *i-th* layer as active
setLayerInactive(i)	set the *i-th* layer as not active
getActiveLayer()	returns the only active layer at the present time

4.2 Managing the Contexts

Also the contexts should be objects capable of changing and evolving as the users live their lives. We recall that we defined the *i-th* context of user *u* as $C^u_i=(V^u_i,E^u_i,L,T,\rho,\psi,\lambda)$, where V^u_i is the set

made of *u* and all its neighbours in the *i-th* context, and E^{u}_{i} is the set of existing relationships among them. The main requirements are to keep track of which ones are the nodes and edges in the context, and which ones are the times at which these nodes and edges available at each time.

Method name	**Short description**
addUser(v)	add the user *v* to the context
addEdge(v,z,l)	Add the edge *(v,z)* to the context with the specified label *l*
setUserOnline(v)	set the user *v* as online
setUserOffline(v)	set the user *v* as offline
setEdgeLabel(v,z,l)	set to *l* the label of the edge *(v,z)*
getUsers()	return the set of users in the context
getEdges()	return the set of edges in the context
hasUser(v)	check if the context contains the user *v*
hasEdge(u,v)	check if the edge *(u,v)* is in the context

5 Graph Neural Networks

In this section we will introduce the reader to the framework of Graph Neural Networks, giving some basic definitions, and then we will discuss how to use the tool in the scenario of the Contextual Ego Network. We present as example the link prediction task, and finally we discuss a possible architecture for using Graph Neural Networks in a decentralized scenario.

5.1 Definition

One of the goals of HELIOS is to enrich the user experience through machine learning. For example, suggesting new social interactions to users in which they may be interested in is a well-known link prediction task [48] that could help enrich the provided social experience.

To perform machine learning tasks for the users of the Decentralized Online Social Network, we propose leveraging the information encapsulated in their previous interactions. Our motivation comes from the premise that users interacting in a context sometimes pertains to common aspects of their personality that, in turn, relate to interacting or being interested in interacting with common third parties in the same context.

An emerging idea for extracting such information from graph structures, such as those of HELIOS, is to make use of the very popular concept of neural networks. Neural networks is one of the most prominent areas of research that is in the forefront of state-of-the-art developments in many fields. Their success can be attributed to the hardware and software scalability and performance improvements (e.g. GPU computing, NVIDIA's Tensorflow framework) that have been supporting them in recent years, and which are starting to be available for mobile platforms (e.g. NVIDIA's CodeWorks platform for Android[1]).

When applied on graphs, Graph Neural Networks (GNNs) are a type of neural network that uses available relational information to extract high quality low-dimensional representations of node features by iteratively propagating node information to their graph neighbors until it reaches neighbors sufficiently many hops away. These representations can then be input to other machine learning models or, more commonly, be integrated in a traditional neural network architectures that perform prediction tasks, such as the previously described link prediction.

One of the advantages of deriving representations through information propagation across the graph is that expressive representations are assigned to nodes with originally little or no available information, hence facilitating unsupervised and semi-supervised learning. For example, this process allows us to perform link prediction without any additional information of known user attributes (although additional information can certainly improve the predictive ability of GNNs) by using one-hot encoded user ids as attributes. This scenario is of particular interest to HELIOS, since social media users are often reluctant to share personal attributes with their platforms, which means that prediction tasks could target users whose relations are their only originally available information. Furthermore, flexible user modeling means that users may provide information for attributes that, even if known, do not help the prediction task too much. For users of either of these

[1] https://developer.nvidia.com/tools-overview

categories, the expressiveness of found representations allows GNNs to still perform high quality predictions.

For example, in the graph of Figure 6.1 no attributes are known for node E. However, a GNN could derive low-dimensional representations that are iteratively propagated to graph neighbors. This way, E's representation would be greatly influenced by the already similar (in that they both feature the attribute P) nodes B and C, with which it is linked through 3 (from B) + 4 (from C) = 7 paths in total. On the other hand, E's representation would be less similar to D's, as there exist only 3 paths to propagate the latter's representation. Hence, the link CE would be detected as a more likely link candidate than ED, even if no attributes were initially provided for E.

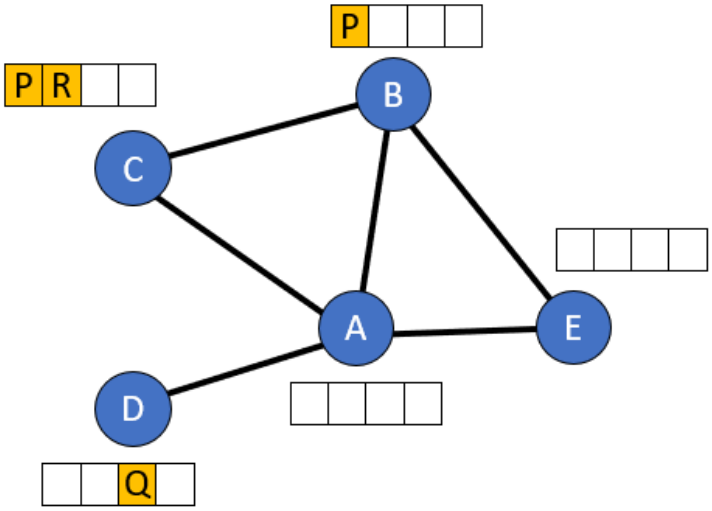

Figure 6.1. A contextual network where only a few attribute values are known. (Known and unknown attributes are depicted with filled and empty boxes respectively.)

5.2 Generic GNN Architectures

In Figure 6.2, we show a conceptually simple scheme that was first used to define GNNs [49]. In this scheme, the parameters of a neural network layer f are trained to produce high quality state representations *H=f(H,X)* that are helpful in predicting *O=g(H)*. Banach's fixed point theorem ensures that there always exist such state representations that can be found by iteratively applying *f*.

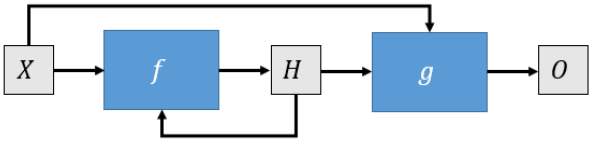

Figure 6.2. GNN using an iterative scheme. X,H,O correspond to node features, state representations of nodes and output predictions respectively

However, the iterative process makes the parameters of *f* difficult to train [50]. To avoid this problem, well-known approaches have proposed relaxing Banach's theorem to approximate f wth a Multilayer Neural Network (MNN) scheme [51], [52], as demonstrated in Figure 6.3. [52] and [53]

explain that the common approach of constructing f_N as an aggregation of neighborhood embeddings to pass on to the neural layer can become equivalent to graph isomorphism detection mechanisms, e.g. when using either average or max pooling of neighborhood features.

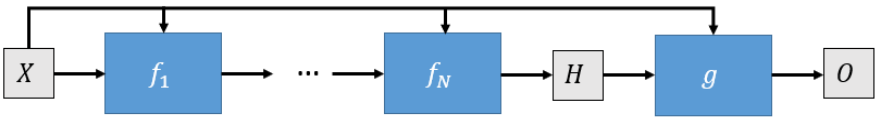

Figure 6.3. GNN using a MNN architecture.

The advantage of GNNs compared to simpler graph mining algorithms, such as random walks [54] and spectral analysis [55], is that they can inherently express multilayer (e.g. spatial and temporal as outlined by [56]) network representations and can express non-linear relations.

5.3 GNNs for Link Prediction

Given that HELIOS users may behave differently in different contexts, we can consider each context to be comprised of possibly different types of relations. We can hence perform link prediction using GNN architectures [57] that explicitly acknowledge possible differences between relations.

Architectures for link prediction usually find k-dimensional representations $H[u] \in \mathcal{R}^k$ of nodes *u* that help score network links *(u,v)* in a relation-context through a function *g(H[u],H[v],r_i)* that depends on context-related parameters $r_i \in \mathcal{R}^k$. In Table 6.1 we summarize some of the most well-known link scoring functions [57–60] that can be formulated in this manner.

Approach	**Link Scoring Function**
TransE [58]	$\|\|H[u]-H[v]+r_i\|\|_p$
DistMult [59]	$\langle H[u], r, H[v] \rangle$
R-GCN [57]	$\langle H[u], r, H[v] \rangle$
ConvE [60]	$\langle H[u], ReLU(1D(ReLU([2D(H[v]) ; 2D(r_i)]*\omega)W)) \rangle$

Table 6.1. Well-known link scoring functions, where $\langle \cdot,\cdot,\cdot \rangle$ is the triple inner product, $\langle \cdot,\cdot \rangle$ the inner product, and $*\omega$ convolution with a pattern ω.

To discover node representations, state-of-the-art approaches, such as those presented above, have been motivated by the success of convolutional neural networks [61] and have been adapted to graph structures. In particular, in each layer, they perform the graph equivalent of convolutional operations by aggregating the previous representations of each node's CEN. This way, representations $H_{n+1}=f_n(H_n)$ at layers $n+1$ become of increasingly higher quality compared to the ground truth features $X=H_0$, as they encapsulate more relational information from the graph compared to the previous layer. These types of approaches are a non-linear multilayer equivalent to neural network that convolves the representations of each layer with spectral graph filters [62].

As mentioned before, a promising architecture that is often adopted in state-of-the-art research[63] is to find node representations through average pooling of node neighborhoods and then applying a non-linear activation function $\sigma()$, which is usually the ReLU function $max(0,\cdot)$. This yields the following representation at each neural network layer:

$$H_{n+1}[u] = \sigma(W_0 H_n[u] + \sum_{v \in C_i^u} W_{i,n} H_n[v])$$

where C_i^u is the contextual ego network found on the i layer of node u's CEN.

5.4 Temporal GNNs

The above formulation of GNNs aims to improve prediction tasks through the structural properties of graphs. However, the structure of dynamic graphs, such as the Decentralized Online Social Network, can also change over time. In these cases, exploring the temporal aspect of relations may play a key role in further improving the quality of predictions. For example, a user's preferences may have drifted over time to ones that are better encapsulated by relations they have more recently formed [64].

To account for the temporal aspect of systems, traditional neural network architectures have proposed a recurrent type of layer called Long Short-Term Memory (LSTM) [65], [66]. Such layers use the representations they derived at previous time steps as additional inputs and have been found to work best with *tanh* as their activation function. Intuitively, the representations they create can be considered conditioned to those selected at previous timesteps. Thanks to the promising results of LSTM layers in modeling the dynamic nature of systems, they have also been adopted to GNNs [67]. In this domain, the LSTM layer can be either fully connected or defined through convolutions.

Contrary to the above, it has been argued that the conceptualization of dynamic graphs as consecutive graph snapshots allows their dynamic nature to be modelled through nodes that reside "close" to them from both a structural and a temporal perspective. This train of thought aligns with the previously discussed applications and defines node neighborhoods to comprise nodes that are linked within a temporal range [68], [69]. Applying this formulation in GNN architectures is simple, as it affects only which node neighbors contribute to extracting node representations.

For both types of approaches, modeling the temporal aspect of dynamic graphs has been shown to greatly enhance the predictive ability of GNNs.

5.5 Towards a Distributed Architecture for GNNs

Existing GNN implementations are not designed for use in systems similar to a Decentralized Online Social Network, where graph structure knowledge is distributed across multiple nodes-devices, of which only a few (e.g. those of a CEN) can contribute to the training and prediction tasks of each node. Instead, their training relies on global synchronization mechanisms across the graph. This holds true even for optimizations aimed at industrial applications that propagate local features and limit the number of parameters to be trained [70], [71]. In Figure 6.4 we present the existing centralized architecture that is required for training GNNs across user devices; devices need share their data (including private ones) with a central system that is trained towards discovering the optimal parameters of the GNN model. They also rely on this system to perform and communicate the outcome of prediction tasks, such as suggesting new user interactions.

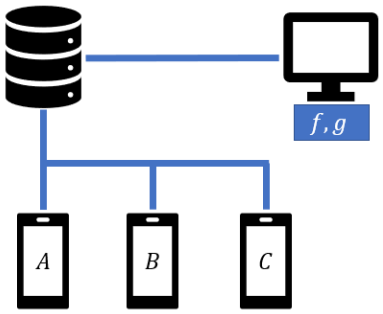

Figure 6.4. Centralized GNN architecture.

To address this shortcoming, we propose a GNN architecture based on federated learning principles [72], [73] to train local models on HELIOS devices without explicitly accessing the data of users outside the Contextual Ego Network. In particular, we propose learning models of different devices by sharing only their own *update parameters* (e.g. an updated model or model gradients) to their ego network's neighbors. This way, each device is responsible for creating and maintaining its own approximation of the GNN by utilizing only the information retrievable from its neighbors, maintaining the privacy of the latter's nodes.

In short, each user's approximation is improved using the approximations found in their contextual ego network, as shown in Figure 6.5. If the network of users is well-connected, local GNNs will eventually converge to the same global GNN. On the other hand, if the network is sparsely connected, this setting emphasises local structural characteristics of the network. The outlined model differs from federated learning in that there does not explicitly exist a central architecture to be updated; instead each device sees itself as the target of its ego network's learning task.

This setting is similar to training techniques for distributed agents [74], [75] whose behavior is driven by adapting neural network architectures [76]. The main difference of our approach is that, instead of training towards optimizing a domain objective, we focus on learning which context characteristics can best suggest the links of the ego network. To this end, we aim to adapt a scheme that uses the propagation mechanism of personalized PageRank for neural architectures [77] so that it can be applied on a distributed system.

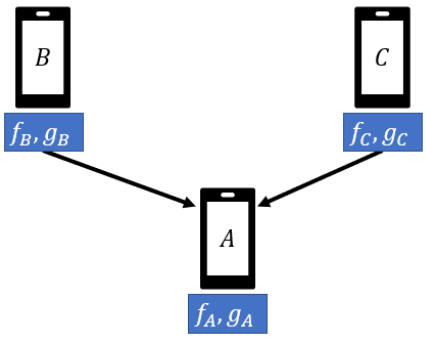

Figure 6.5. Distributed GNN architecture.

Potential Challenges

There are several challenges to overcome in order for the above distributed conceptualization of GNNs to work. These challenges arise from practical constraints that could arise during the practical deployment of our architecture and will be investigated in later deliverables.

- Firstly, the rate at which devices communicate may not yield fast enough convergence of the GNNs model into meaningful results.

- Furthermore, devices store and share only data pertaining to their contextual ego network. However, node representations are, by their nature, anonymized and additional steps could be needed to align them between GNNs of different devices, especially if they previously had little or no overlap between their ego networks.

- Finally, we mentioned that relational GNNs rely on context-related parameters to perform link prediction. However, contexts are unique to each user (i.e. different users may have assigned different contexts to the same type of interaction), which allows sharing of parameters only in the context in which the devices interact at each time. This may influence the predictive ability of GNNs, as only a small subset of relational information is propagated through the Decentralized Online Social Network.

6 Conclusions

This deliverable (named D4.2 “Define a time-dependent social graph”) discussed the need to model the P2P Social Overlay of HELIOS introducing a temporal dimension. We described several temporal graph formalism and adapted one to formalize the Heterogeneous Social Network Graph and the Contextual Ego Network with a temporal graph. We also discussed how to apply GNN for the study and prediction in the Contextual Ego Network.

In this deliverable we studied a number of different temporal graph formalisms trying to understand their expressiveness and abstractness. This was needed to choose the best formalism that adapts best in our scenario, but also to find a formalism easily extensible. After deciding that the Time-varying graph is the best one for our scenario, we decided to apply some minor modifications to the formalism to model our scenario in a more natural way, thus developing our own formalism which we called HELIOS time-varying graph. We also introduced an extension to the vanilla formalism, which is able to model graphs where nodes can be organized in separate layers.

Using this formalism, we propos a formalization of the Heterogeneous Social Network Graph, which includes the possibility that the Social Overlay evolves over time. However, it is unreasonable, but also against the idea to not have privacy disclosures, that all peers maintain the whole Heterogeneous Social Network Graph. The Contextual Ego Network, that was previously introduced as the local view of each node of the whole Heterogeneous Social Network Graph, is here formalized. A Contextual Ego Network is a multi-layer graph, where each layer models a context and is implemented with a time evolving ego network

We also discussed the importance of studying the Contextual Ego Network for different reasons: from understanding the human behaviour to develop a more intelligent system, to study the node churn to minimize the effect on the P2P overlay. We also studied the employment of Graph Neural Networks within the Contextual Ego Network as its application can help in several tasks, for instance link prediction. Finally, we also discuss a possible distributed architecture for Graph Neural Networks in which federated learning techniques are used to train local models, and a wisdom of the crowd is built within each ego network.

References

[1] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, and K. Rzadca, “Decentralized online social networks,” in *Handbook of Social Network Technologies and Applications*, Springer, 2010, pp. 349–378.

[2] R. Schollmeier, “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications,” in *Proceedings First International Conference on Peer-to-Peer Computing*, 2001, pp. 101–102.

[3] M. Cordeiro, R. P. Sarmento, P. Brazdil, and J. Gama, “Evolving networks and social network analysis methods and techniques,” *Social Media and Journalism: Trends, Connections, Implications*, p. 101, 2018.

[4] S. Saroiu, K. P. Gummadi, and S. D. Gribble, “Measuring and analyzing the characteristics of Napster and Gnutella hosts,” *Multimedia systems*, vol. 9, no. 2, pp. 170–184, 2003.

[5] A. De Salve, M. Dondio, B. Guidi, and L. Ricci, “The impact of user’s availability on on-line ego networks: a facebook analysis,” *Computer Communications*, vol. 73, pp. 211–218, 2016.

[6] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz, and others, “Handling churn in a DHT,” in *Proceedings of the USENIX Annual Technical Conference*, 2004, vol. 6, pp. 127–140.

[7] D. Stutzbach and R. Rejaie, “Understanding churn in peer-to-peer networks,” in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, pp. 189–202.

[8] N. Kamalraj and A. Malathi, “A survey on churn prediction techniques in communication sector,” *International Journal of Computer Applications*, vol. 64, no. 5, pp. 39–42, 2013.

[9] R. J. Jadhav and U. T. Pawar, “Churn prediction in telecommunication using data mining technology,” *International Journal of Advanced Computer Science and Applications*, vol. 2, no. 2, 2011.

[10] P. Datta, B. Masand, D. R. Mani, and B. Li, “Automated cellular modeling and prediction on a large scale,” *Artificial Intelligence Review*, vol. 14, no. 6, pp. 485–502, 2000.

[11] K. Coussement and D. Van den Poel, “Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques,” *Expert systems with applications*, vol. 34, no. 1, pp. 313–327, 2008.

[12] K. Dasgupta *et al.*, “Social ties and their relevance to churn in mobile telecom networks,” in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, 2008, pp. 668–677.

[13] C. Vassilakis and I. Stavrakakis, “Minimizing node churn in peer-to-peer streaming,” *Computer Communications*, vol. 33, no. 14, pp. 1598–1614, 2010.

[14] F. Wang, J. Liu, and Y. Xiong, “Stable peers: Existence, importance, and application in peer-to-peer live video streaming,” in *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, 2008, pp. 1364–1372.

[15] F. Wang, J. Liu, and Y. Xiong, “On node stability and organization in peer-to-peer video streaming systems,” *IEEE Systems Journal*, vol. 5, no. 4, pp. 440–450, 2011.

[16] B. Guidi, A. Michienzi, and G. Rossetti, “Dynamic community analysis in decentralized online social networks,” in *European Conference on Parallel Processing*, 2017, pp. 517–528.

[17] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, “Time-varying graphs and dynamic networks,” *International Journal of Parallel, Emergent and Distributed Systems*, vol. 27, no. 5, pp. 387–408, 2012.

[18] M. Magnani and L. Rossi, “The ml-model for multi-layer social networks,” in *2011 International Conference on Advances in Social Networks Analysis and Mining*, 2011, pp. 5–12.

[19] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, “Characterising temporal distance and reachability in mobile and online social networks,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 118–124, 2010.

[20] A. Ferreira, “Building a reference combinatorial model for MANETs,” *IEEE network*, vol. 18, no. 5, pp. 24–29, 2004.

[21] V. Kostakos, “Temporal graphs,” *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[22] D. Kempe, J. Kleinberg, and A. Kumar, “Connectivity and inference problems for temporal networks,” *Journal of Computer and System Sciences*, vol. 64, no. 4, pp. 820–842, 2002.

[23] P. Flocchini, B. Mans, and N. Santoro, “Exploration of periodically varying graphs,” in *International Symposium on Algorithms and Computation*, 2009, pp. 534–543.

[24] P. Holme and J. Saramäki, “Temporal networks,” *Physics reports*, vol. 519, no. 3, pp. 97–125, 2012.

[25] M. Latapy, T. Viard, and C. Magnien, “Stream graphs and link streams for the modeling of interactions over time,” *Social Network Analysis and Mining*, vol. 8, no. 1, p. 61, 2018.

[26] L. Arge and J. S. Vitter, “Optimal dynamic interval management in external memory,” in *Proceedings of 37th Conference on Foundations of Computer Science*, 1996, pp. 560–569.

[27] G. Blankenagel and R. H. Güting, “External segment trees,” *Algorithmica*, vol. 12, no. 6, pp. 498–532, 1994.

[28] B. Salzberg and V. J. Tsotras, “Comparison of access methods for time-evolving data,” *ACM Computing Surveys (CSUR)*, vol. 31, no. 2, pp. 158–221, 1999.

[29] E. F. Moore, “The shortest path through a maze,” in *Proc. Int. Symp. Switching Theory, 1959*, 1959, pp. 285–292.

[30] M. Girvan and M. E. Newman, “Community structure in social and biological networks,” *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[31] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, “Community detection in social media,” *Data Mining and Knowledge Discovery*, vol. 24, no. 3, pp. 515–554, 2012.

[32] S. Fortunato, “Community detection in graphs,” *Physics reports*, vol. 486, no. 3–5, pp. 75–174, 2010.

[33] A. Lancichinetti and S. Fortunato, “Community detection algorithms: a comparative analysis,” *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

[34] R. Cazabet and F. Amblard, “Dynamic community detection,” *Encyclopedia of Social Network Analysis and Mining*, pp. 404–414, 2014.

[35] G. Rossetti and R. Cazabet, “Community discovery in dynamic networks: a survey,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, p. 35, 2018.

[36] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, “Network motifs: simple building blocks of complex networks,” *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[37] D. Braha and Y. Bar-Yam, “Time-dependent complex networks: Dynamic centrality, dynamic motifs, and cycles of social interactions,” in *Adaptive Networks*, Springer, 2009, pp. 39–50.

[38] Q. Zhao, Y. Tian, Q. He, N. Oliver, R. Jin, and W.-C. Lee, “Communication motifs: a tool to characterize social communications,” in *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010, pp. 1645–1648.

[39] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, “Temporal motifs in time-dependent networks,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 11, p. P11005, 2011.

[40] M. Everett and S. P. Borgatti, “Ego network betweenness,” *Social networks*, vol. 27, no. 1, pp. 31–38, 2005.

[41] E. M. Daly and M. Haahr, “Social network analysis for routing in disconnected delay-tolerant manets,” in *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, 2007, pp. 32–40.

[42] V. Arnaboldi, M. Conti, A. Passarella, and F. Pezzoni, “Analysis of ego network structure in online social networks,” in *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, 2012, pp. 31–40.

[43] W.-X. Zhou, D. Sornette, R. A. Hill, and R. I. Dunbar, “Discrete hierarchical organization of social group sizes,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 272, no. 1561, pp. 439–444, 2005.

[44] V. Arnaboldi, M. Conti, A. Passarella, and F. Pezzoni, “Ego networks in twitter: an experimental analysis,” in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 3459–3464.

[45] B. Guidi, A. Michienzi, and G. Rossetti, “Towards the Dynamic Community Discovery in Decentralized Online Social Networks,” *Journal of Grid Computing*, vol. 17, no. 1, pp. 23–44, 2019.

[46] B. Guidi, A. Michienzi, and L. Ricci, “Sonic-man: a distributed protocol for dynamic community detection and management,” in *IFIP International Conference on Distributed Applications and Interoperable Systems*, 2018, pp. 93–109.

[47] M. Farrugia, N. Hurley, and A. Quigley, “Exploring temporal ego networks using small multiples and tree-ring layouts,” *Proc. ACHI*, vol. 2011, pp. 23–28, 2011.

[48] D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[49] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[50] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, “Graph neural networks: A review of methods and applications,” *arXiv preprint arXiv:1812.08434*, 2018.

[51] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[52] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[53] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?,” *arXiv preprint arXiv:1810.00826*, 2018.

[54] H. Tong, C. Faloutsos, and J.-Y. Pan, “Fast random walk with restart and its applications,” in *Sixth International Conference on Data Mining (ICDM’06)*, 2006, pp. 613–622.

[55] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. De Luca, and S. Albayrak, “Spectral analysis of signed graphs for clustering, prediction and visualization,” in *Proceedings of the 2010 SIAM International conference on Data Mining*, 2010, pp. 559–570.

[56] X. Wang and A. Gupta, “Videos as space-time region graphs,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 399–417.

[57] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, “Modeling relational data with graph convolutional networks,” in *European Semantic Web Conference*, 2018, pp. 593–607.

[58] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” in *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[59] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, “Embedding entities and relations for learning and inference in knowledge bases,” *arXiv preprint arXiv:1412.6575*, 2014.

[60] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, “Convolutional 2d knowledge graph embeddings,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[62] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[63] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *arXiv preprint arXiv:1901.00596*, 2019.

[64] H.-H. Chen, L. Gou, X. L. Zhang, and C. L. Giles, “Predicting recent links in FOAF networks,” in *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*, 2012, pp. 156–163.

[65] S. Hochreiter and J. Schmidhuber, “LSTM can solve hard long time lag problems,” in *Advances in neural information processing systems*, 1997, pp. 473–479.

[66] J. Donahue *et al.*, “Long-term recurrent convolutional networks for visual recognition and description,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[67] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, “Structured sequence modeling with graph convolutional recurrent networks,” in *International Conference on Neural Information Processing*, 2018, pp. 362–373.

[68] S. Yan, Y. Xiong, and D. Lin, “Spatial temporal graph convolutional networks for skeleton-based action recognition,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[69] M. J. Williams and M. Musolesi, “Spatio-temporal networks: reachability, centrality and robustness,” *Royal Society open science*, vol. 3, no. 6, p. 160196, 2016.

[70] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. Zemel, “Graph partition neural networks for semi-supervised classification,” *arXiv preprint arXiv:1803.06272*, 2018.

[71] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[72] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.

[73] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, “Federated multi-task learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[74] L. Xiao and S. Boyd, “Fast linear iterations for distributed averaging,” *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[75] L. Xiao, S. Boyd, and S.-J. Kim, “Distributed average consensus with least-mean-square deviation,” *Journal of parallel and distributed computing*, vol. 67, no. 1, pp. 33–46, 2007.

[76] H. Ma, Z. Wang, D. Wang, D. Liu, P. Yan, and Q. Wei, “Neural-network-based distributed adaptive robust control for a class of nonlinear multiagent systems with time delays and external noises,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 6, pp. 750–758, 2015.

[77] J. Klicpera, A. Bojchevski, and S. Günnemann, “Predict then propagate: Graph neural networks meet personalized pagerank,” 2018.