Project no. 732027

VIRT-EU

Values and ethics in Innovation for Responsible Technology in EUrope

Horizon 2020

ICT-35-2016
Enabling responsible ICT-related research and innovation
Start date: 1 January 2017 – Duration: 36 months

D3.1
Quantitative technical report

Due date: 30 April 2019

Actual submission date: 1 May 2019

Number of pages: 114

Lead beneficiary: UU

Authors: Matteo Magnani, Luca Rossi, Davide Vega

Project Consortium

Beneficiary no.	Beneficiary name	Short name
1 (Coordinator)	IT University of Copenhagen	ITU
2	London School of Economics	LSE
3	Uppsala Universitet	UU
4	Politecnico Di Torino	POLITO
5	Copenhagen Institute of Interaction Design	CIID
6	Open Rights Group	ORG

Dissemination Level

PU	Public	**X**
CO	Confidential, only for members of the consortium (including the Commission Services)	
EU-RES	Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)	
EU-CON	Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)	
EU-SEC	Classified Information: SECRET UE (Commission Decision 2005/444/EC)	

Dissemination Type

R	Document, report	**X**
DEM	Demonstrator, pilot, prototype	
DEC	Websites, patent filling, videos, etc.	
O	Other	
ETHICS	Ethics requirement	

Contents

Executive summary

The objective of this deliverable is to present *the definition of the adopted network analysis metrics, code and quantitative analysis*. These tasks were the responsibility of the *quantitative unit* of the VirtEU project, led by Matteo Magnani at Uppsala University and by Luca Rossi at the IT University, Copenhagen, and also including Davide Vega, Roberto Interdonato and Obaida Hanteer.

This report starts with a description of the theoretical framework on which the quantitative analysis is based, positioning the results presented in this document with respect to the objectives of other research units in the project (Section 1: Introduction). The remainder of the report is organized into three main parts.

Section 2 (Modelling online communities) and Section 3 (Analyzing online communities) describe the way in which we mathematically represent online social networks and the methods that we have developed to analyze them, including algorithms to automatically identify communities, similarity metrics to compare different aspects of the identified communities, as well as a visualization method to make the results more accessible to the qualitative unit of the project.

Sections 4 (MeetUp data analysis) and 5 (Twitter data analysis) present the results of our quantitative analysis, that is, the application of the aforementioned models and methods in addition to other traditional data analysis approaches to identify and study online IoT communities in the MeetUp and Twitter platforms. These sections also include some methodological reflections, and in particular a critical assessment of the limits and opportunities associated to the two data sources.

The last sections of the report give more details about the state of the art on online conversation monitoring and community detection (Appendix A: Related literature) and also include a description of the network analysis library where we implemented the developed methods. This software library was used to perform some of the analyses included in this document. The library, part of which had been developed before the beginning of the VirtEU project, has been extended with new algorithms and completely re-engineered, that is, most of the code has been rewritten to be more extensible and to deal more efficiently with the large datasets collected for this project. The full code is available at the address https://cran.r-project.org/package=multinet and Appendix B (The `multinet` library) describes the content of the library and how to use it, following training workshops we have given at major conferences in the social network analysis and computing areas (SunBelt and Social Informatics in 2018; SunBelt and IC2S2 in 2019 – planned). In In Appendix C (Meetup data) we list the MeetUp groups included in the analysis.

While this report mainly presents the results of the quantitative unit, the results concerning the analysis of MeetUp and Twitter data have been obtained through a close collaboration with members from the qualitative unit, as detailed in the corresponding parts of the deliverable. Discussions with other project members have also influenced the choice of which methods to focus on

and what features to support in the development of the methods. In addition, some of the methods described in this report have been developed as part of collaborations with researchers not affiliated to the project. These collaborations are acknowledged in the relevant parts of the report, where the collaborative articles from which some of the content has been taken are referenced.

Part of the results of the quantitative unit have been described in previous deliverables and have thus not been included in this report. In particular, Deliverable 2.2 (Research synthesis), Section 2, describes the software platform developed to collect data and to provide a set of data exploration tools to the qualitative unit. All the data analyzed in this report has been collected using the platform developed for the project. In addition, some preliminary data analyses done by the quantitative unit have been included in Deliverable 1.2 (First year progress report), and Deliverable 1.3 (Mid-way report), Section 3. These previous documents include the analysis of the Twitter data from specific IoT events, performed during the first year of the project to support the qualitative unit. These analyses are not included in this report. In addition, the aforementioned progress reports contain a preliminary analysis of what we call the *consolidated dataset*, explained in more detail in Section 5. These analyses have now been extended and completed, and are thus included in this report.

1 Introduction

This report summarizes the results of the quantitative unit of the VirtEU project concerning the analysis of online data. We present the mathematical models we defined to represent the data, the analysis methods we developed to extract knowledge from these data representations, the software we designed and implemented to do so, as well as the results obtained applying our models and analysis methods to two sources that had been previously selected during the first part of the project (MeetUp and Twitter). All these tasks were part of the original project description and have been performed as planned.

Before presenting these results, it is important to understand their role in the more general theoretical framework of the project. The next section summarizes this framework and positions the activities of the quantitative unit with respect to it. We then present the outline of the rest of the report and list the scientific publications whose content has been partly included in the report, or that are currently under preparation based on some of the content of the report.

1.1 Theoretical framework

In the upper part of Figure 1 we have indicated the main concern of the project, that is, IoT. Based on fieldwork we have carried out into IoT spaces in Europe and preliminary analysis of online discussions of IoT, we have identified that IoT emerges as a socio-technical assemblage of different things, issues and approaches. More specifically, we have come to identify IoT as an assemblage of 1) things (connected devices and technologies), 2) practices (i.e. practices of building connections between things and technologies) and 3) application contexts (e.g. smart cities, e-health, wearables and so on). Such a positioning of IoT builds on the assumption that technologies are not neutral, and external to the socio [e.g. human] and technical [e.g. things, technologies, network systems] they are part of. Hence, the significance of the results presented in this report is based on the assumption that **the IoT is constructed by the actors and communities of practice that define themselves as part of the IoT ecosystem and the relations that sustain this assemblage**.

As an example, whether sustainability or privacy will be values strongly associated with the IoT in the future might come to depend on whether these are "matters of concern" for the main IoT actors and communities, and to the relationship between these and other matters of concern of the actors and communities. To simplify, if no one in the community of IoT developers thinks that sustainability is an important value, then sustainability might not be embedded in future IoT products. At the same time, even if developers are concerned with sustainability, strong concerns on profitability and financial survival of the startup could also prevent the design of sustainable products if this implies additional costs that cannot be afforded.

Against this background it is then important to observe how different values (and not just the ones more clearly belonging to the ethical domain) are discussed inside these communities and how they are expressed as matters of

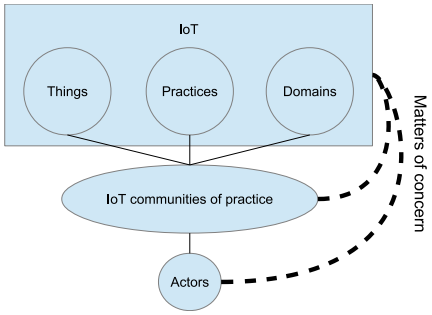


Figure 1: Theoretical framework used in this deliverable.

concern.

In the VirtEU project, a significant effort has been put into fieldwork and ethnographically studying the main IoT actors and communities. However, this requires a lot of resources and can only allow a limited study of these communities in time and space. Therefore, the objective of our study of the online information produced by IoT actors and communities is to complement the qualitative information about matters of concern obtained through fieldwork.

1.2 Outline

In Section 2 we present the mathematical models we used to represent online interactions between Twitter users. The basic model is the so called multiplex network, where actors can be connected through multiple types of edges. This is useful to represent the fact that Twitter users can reply to other users' tweets, or retweet them, or mention them. It is also useful to group online interactions depending on the topic of the tweet. The topics, and thus the exchanged text, are an important element given our objective of mapping matters of concern. Therefore, starting from the multiplex network model, we have further extended it including three fundamental elements to understand human information networks: the individuals (actors) in the network, the information they exchange, that is often observable online as text content, and the time when these exchanges happen. These different aspects of the data are also called *layers*. We present a simple, expressive and extensible model for temporal text networks, that we claim can be used as a common ground across different types of networks and analysis tasks. It is worth noticing that these models have been mainly applied to Twitter data, as for MeetUp we do not have detailed information about user-to-user interactions – in which case simpler tabular data representations have been used. However, the information collected from MeetUp is much cleaner and easier to associate to the matters of concern of IoT community users.

In Section 3 we present the analysis methods we developed for the aforementioned models. In particular, in Section 3.1 we extend the popular clique

percolation method to work on multiplex networks. Our extension requires to rethink the basic concepts on which the original clique percolation algorithm is based, including cliques and clique adjacency, to handle the presence of multiple types of ties. In Section 3.2 we show how simple data transformations allow the direct application of analysis methods already developed in other domains, from traditional data mining to multilayer network mining, to analyze temporal text networks and thus to identify communities of users characterized by common discussion topics. Computing layer similarities is another important way of characterizing multiplex networks because various static properties and dynamic processes depend on the relationships between layers. In Section 3.3 we provide a taxonomy of approaches to compare layers in multiplex networks that includes, systematizes and extends existing comparison functions, and is complemented by a set of practical guidelines on how to apply them. Finally, in Section 3.4 we introduce *multiforce*, a force-directed layout for multiplex networks where both intra-layer and inter-layer relationships among nodes are used to compute node coordinates. Despite its simplicity, this algorithm can reproduce the main existing approaches to draw multiplex sociograms, and also supports a new intermediate type of layout.

Sections 4 and 5 present the results of the application of the aforementioned methods, together with more basic approaches, to data extracted from MeetUp and Twitter. As we will see, the data from MeetUp provides a very good mapping with the theoretical framework presented at the beginning of this section, and also allows us to observe the spatio-temporal evolution of the matters of concern emerging inside this platform. Twitter data has been significantly more difficult to analyze. Therefore in the dedicated section we indicate the theoretical and practical difficulties we encountered on a large Twitter dataset and the consequent collection and analysis of smaller and manually curated data, called the *consolidated data*, that was also planned in the original description of the project.

We also include two appendices. In the first we present an overview of the literature related to the analysis models and methods discussed in the report, while in the second we provide a tutorial-like description of the software library used to performed some of the analyses. This library was re-engineered as part of the project, including the implementation or testing of the methods described in this report.

1.3 References

Some of the contents of this deliverable have been published in peer reviewed books of proceedings and journals: the clique-percolation-based community detection method [124], the *multiforce* visualization layout [41], the network comparison measures [18] and the model and methods for the representation and analysis of online conversations in temporal text networks [125]. All the published papers contain a significant amount of additional information, typically in the form of theoretical and experimental analyses of the proposed methods, that we have not included in this report to keep it compact. References to this

extended material are provided in the appropriate parts of the deliverable.

Parts of this deliverable have also been included in two papers currently under submission: a survey journal article on community detection in multiplex networks (for which some of the content has been taken from Appendix A) and a technical article describing the software library used to produce most of the results presented in this deliverable (roughly corresponding to Appendix B). The material included in Sections 4 and 5 will be used in an extended form as a basis for two journal articles presenting the results of our empirical analysis, planned to be submitted for publication before the end of the project.

2 Modelling online communities

In this project the quantitative analysis of online communities has been initially based on the models and methods from the discipline known as Social Network Analysis. This discipline is characterized by the use of graphs, that is, nodes and links connecting them, to represent social networks.

Simple social network models have been useful in the first part of the project, e.g., to identify central actors associated to IoT events to be then taken into consideration by the qualitative unit. For this kind of analysis, knowing who is interacting with whom and how often is typically sufficient, and the network analysis web-based platform we developed for the project provided such functionality to be used by the other project members, including those without a technical background in data analysis.

However, a simple network is not sufficient to reach the objectives of this report. The fact that two actors are interacting does not tell us anything about the matters of concern discussed by these actors, nor about the meaning of the interaction. Therefore, as originally planned, for these more advanced analysis we used a richer data model allowing us to represent different types of interactions. The *multiplex network* model, where each type of interaction is represented as a distinct *layer*, is described in the next section.

Building on this, we later introduce an additional extension where different layers represent either actors or the messages exchanged among these actors, including their text and the time of the interaction. This model, that we call *temporal text network*, has been specifically developed in the context of the VirtEU project.

2.1 Multiplex networks

Multiplex networks provide a simple yet expressive way to model a wide range of physical and social systems as sets of entities connected by multiple types of relationships, that we also call *layers* following the terminology in [68]. For example, a transport network can be modelled as a set of locations, such as cities or streets, connected by different types of public transport like airplanes, trains, and buses.

In this document we use the following definition of multiplex network:

Definition 1 (Multiplex network) *Given a set of nodes $\mathcal{N}$ and a set of layers $\mathcal{L}$, a multiplex network is defined as a quadruple $M = (\mathcal{N}, \mathcal{L}, V, E)$ where (V, E) is a graph, $V \subseteq \mathcal{N} \times \mathcal{L}$, and if $(n_1, l_1, n_2, l_2) \in E$ then $l_1 = l_2$.*

An example of multiplex network is shown in Figure 33, where $\mathcal{L} = \{l_1, l_2\}$, $\mathcal{N} = \{n_1, \ldots, n_6\}$, and (n_1, l_1, n_2, l_1) is an example of an edge in E. In the literature alternative terminologies are used, and here we adopt the one in [68], according to which we would say that node n_1 is present in both layer l_1 and layer l_2. In the literature some extended multiplex models have also been proposed, allowing multi-dimensional layers [68] and one-to-many relationships between nodes in different layers [83], but we do not consider these extensions here.

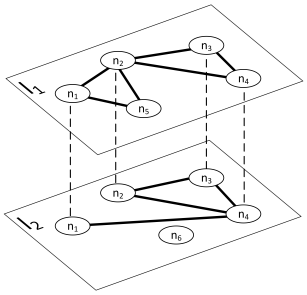


Figure 2: An example of a multiplex network consisting of two layers, six nodes, and ten edges.

Please note that the original definition of multiplex network introduced in the field of Social Network Analysis was more restrictive than the one adopted in this report. In particular, our definition allows some of the nodes not to be present in some layers. For example, $(n_5, l_2) \notin V$ in Figure 2. In some cases, when the term multiplex is used it is assumed that all nodes are present in all layers, and this assumption will often affect the result of layer comparisons. To avoid confusion, in this case we explicitly talk about a *node-aligned multiplex network* [68] and when it is not clear from the context we will call a multiplex network that is not node-aligned a generalized multiplex network.

Definition 2 (Node-aligned multiplex network) *A node-aligned multiplex network is a multiplex network* $(\mathcal{N}, \mathcal{L}, V, E)$ *where* $\forall n \in \mathcal{N}, l \in \mathcal{L} : (n, l) \in V$.

	n_1	n_2	n_3	n_4	n_5	n_6
n_1	0	1	0	0	1	0
n_2	1	0	1	1	1	0
n_3	0	1	0	1	0	0
n_4	0	1	1	0	0	0
n_5	1	1	0	0	0	0
n_6	0	0	0	0	0	0

(a) $\mathbf{A}_{l_1}$

	n_1	n_2	n_3	n_4	n_5	n_6
n_1	0	0	0	1	0	0
n_2	0	0	1	1	0	0
n_3	0	1	0	1	0	0
n_4	1	1	1	0	0	0
n_5	0	0	0	0	0	0
n_6	0	0	0	0	0	0

(b) $\mathbf{A}_{l_2}$

Figure 3: Adjacency matrices for both layers of the multiplex network in Figure 2

Multiplex networks have usually been represented as a set of adjacency matrices $\mathbf{A}_l$, one for each layer l, where $a_l(n_1, n_2) = 1$ if there is an edge between node n_1 and node n_2 in layer l, $a_l(n_1, n_2) = 0$ otherwise. The adjacency matrices for our the example in Figure 2 are shown in Figure 3.

The representation of multiplex networks using adjacency matrices, as in Figure 3, is not the most appropriate to define similarity measures, for two

main reasons. First, it is incomplete, because it only allows representing node-aligned multiplex networks. An example of why this is important is the case of online social media, where each layer represents a different service (Twitter, Facebook, etc.) and it makes a difference whether a user has no connections on Twitter or does not even have an account there. In the example in Figure 2, we would lose the information that nodes n_5 and n_6 are present in different layers.

Second, adjacency matrices present an edge-oriented view over the multiplex network, which might be the reason why most similarity measures in the literature have been limited to edge similarity. If we take a broader look at empirical networks, we can see how other structures can be relevant. As an example, if we look at Figure 33 we can see that the triangle $\{n_2, n_3, n_4\}$ is present in both layers. Unfortunately, this is not obvious from the adjacency matrices and would require checking several disparate entries making definitions more complicated than needed. Therefore, in the following, we use network representation targeted to the specific properties we want to consider when checking the similarity between layers. We call this representation a *property matrix*.

Definition 3 (Property matrix) *A* property matrix $\mathbf{P}$ *is a matrix where:*

1. *the columns correspond to a set S of network structures (nodes, edges, triangles, ...),*
2. *the rows correspond to a set C of contexts where these structures are observed (layers, groups, snapshots, ...), and*
3. *$p_{s,c}$ is the value of an observational function mapping each pair structure/context into a number (degree, distance, ...).*

In the following we will only use layers as contexts, that is, $C = \mathcal{L}$. In summary, each cell $p_{s,c}$ of a property matrix contains the value of the function describing the structure s (for example, a node) on layer c, and different observational functions can be used to define different types of similarity. Examples of property matrices for our working example are shown in Figure 4.

Given a structure s, we can further summarize its presence in the network by summing over all the values in $\mathbf{p}^s$, computing their standard deviation or performing any other kind of aggregation (sum, avg, median, min, max, etc.). As an example, from a node-degree property matrix (Figure 4b) we can obtain the total degree of a node in the whole multiplex network (sum) or its so-called *degree deviation* [12], which is 0 if a node has the same number of connections on all layers and higher when a node is present in different layers with different degrees, and so on. In summary, property matrices provide a more general and informative representation of multiplex networks than adjacency matrices – which are still useful when the objective is just to know about the edges in a node-aligned network. Property matrices also allow us to provide simple and general mathematical definitions of different ways to compare layers, which will instantiate into several existing and new measures when specific property matrices are used.

The notation used to represent multiplex networks is summarized in Table 1.

	n_1	n_2	n_3	n_4	n_5	n_6
l_1	1	1	1	1	1	0
l_2	1	1	1	1	0	1

(a) Nodes, existence

	n_1	n_2	n_3	n_4	n_5	n_6
l_1	2	4	2	2	2	NA
l_2	1	2	2	3	NA	0

(b) Nodes, degree

	n_1	n_2	n_3	n_4	n_5	n_6
l_1	2	1/3	1	1	1	NA
l_2	1	1	1	1/3	NA	0

(c) Nodes, CC

	(n_1, n_2)	...	(n_2, n_4)	...
l_1	1	...	1	...
l_2	0	...	1	...

(d) Dyads, edge existence (clique)

	n_1, n_2, n_3	...	n_1, n_2, n_5	...	n_1, n_2, n_6	...	n_2, n_3, n_4	...
l_1	0	...	1	...	0	...	1	...
l_2	0	...	0	...	0	...	1	...

(e) Triads, triangle existence (clique)

Figure 4: *Property matrices* for our working example in fig. 2. Each *property matrix* is defined by a type of structures (nodes, dyads, triads, etc.), the contexts (layers) and an observational function (existence, degree, forming a clique, distance, etc.)

2.2 Temporal text networks

Multiplex networks provide a richer data representation than the one offered by simple networks, but are still not sufficient to study online conversations. To this aim we define a multi-layer network model also including text and time, that we call temporal text network.

In our opinion, a good model for temporal text networks should be general enough to be able to represent a wide range of systems, but also contain a minimal number of modeling constructs, to make the model easier to use and study. In other terms, a good compromise should be found between expressiveness and simplicity. In addition, given the large number of existing models that have been used for a long time to describe specific aspects of temporal text networks,

Table 1: Terminology and notation

Symbol	Name
$\mathcal{N}$	set of nodes $\{n_1, n_2, ..., n_{\|N\|}\}$
$\mathcal{L}$	set of layers $\{l_1, l_2, ..., l_{\|L\|}\}$
$\mathbf{P}$	property matrix
C	set of contexts (e.g., network layers, snapshots, groups)
S	set of structures (e.g., nodes, edges, dyads, triangles)
$\mathbf{p}_c$	property vector for context $c \in C$
$\mathbf{p}^s$	property vector for structure $s \in S$
$p_{s,c}$	property of s in c (e.g., degree of node s on layer c)
$p_{C',S'}$	p restricted to contexts in $C' \subseteq C$ and structures in $S' \subseteq S$

we believe that both the modeling constructs and the terminology used in our model should be as aligned with previous work as possible. Following these design principles, we propose the following definition of temporal text networks:

Definition 4 (Temporal text network) *A temporal text network is a triple* (G, x, t) *where:*

1. $G = (A, M, E)$ *is a directed bipartite graph, where,* A *is a set of actors,* M *is a set of messages, and* $E \subseteq (A \times M) \cup (M \times A)$.
2. $x : M \to X$, *where* X *is a set of sequences of characters (texts).*
3. $t : E \to T$, *where* T *is an ordered set of time annotations.*

and where the following constraints are satisfied:

1. $\forall m \in M$, *in-degree*$(m) = 1$.
2. $(a_i, m), (m, a_j) \in E \Rightarrow t(a_i, m) \leq t(m, a_j)$.

In our model edge directionality indicates the flow of text in the network: $(a_i, m_j) \in E$ indicates that actor a_i has produced text m_j, while $(m_j, a_i) \in E$ indicates that actor a_i is the recipient of text m_j. Actors with out-degree larger than 0 are information producers, actors with in-degree greater than 0 are information consumers, and actors with both positive in- and out-degree are information prosumers.

Text is represented as a combination of a text container ($m \in M$), and a textual content ($x(m)$). As a consequence, actors in our model do not only generate text, but produce text messages. Two text messages (for example, two tweets, or two emails) may be different messages even if they contain the same text and have been exchanged between the same actors at the same timestamp.

The third key component of temporal text networks is the time attribute t. In our model, time is defined based on a generic set of ordered time annotations T. This enables the adoption of several ways of representing time: as an absolute date-time, as a relative date-time, as a timestamp with an arbitrary format or as a discrete time interval if time has been sliced into time windows as it often happens when temporal networks are analyzed.

When writing about the model's elements, we will sometimes use a concise notation. For example, we will sometimes write an edge and its time together, as in: (a_i, m_j, t_q), where $t_q = t(a_i, m_j)$, and we will sometimes write a message by also indicating its sender, its recipients and its text, as in: $(a_s, m_j, \{a_{r_1}, \ldots, a_{r_n}\}$, "text"), where "text" $= x(m_j)$. Finally, when all the timestamps on the edges adjacent to a message are equal, we can also add a time to the previous notation, as in: $(a_s, m_j, \{a_{r_1}, \ldots, a_{r_n}\}$, "text", $t_q)$.

While very simple, the model introduced above can be used to represent a range of different forms of communication and data from different sources. In particular, by explicitly dividing the network nodes into *actors* and *messages*, their relations implicitly carry more information. For example, whether the type

of communication implemented by a message is unicast, multicast or broadcast is indicated by the out-degree of the message.

With **unicast** a message such as a handwritten letter is sent from a single source to a specific target. This form of written communication has been preserved to the present day through instant messaging services such as those offered by Twitter, Facebook Messenger or Whatsapp and, more traditionally, using the electronic email. Unicast communication allows to keep some text private between two actors, but it can have a large overhead if the same text must be sent to multiple sources because it requires an individual message for every recipient. In order to reach a larger population it is sometimes preferable to use **broadcasting** or **multicasting**. In the former, the message is transmitted to all possible receivers[1], while when the information is addressed to a group of people but not to all possible receivers, such as a post on a Facebook wall, the communication is called multicast.

Figure 5 shows these different types of communication represented using our model.

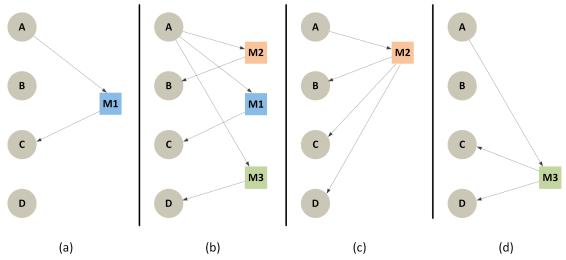


Figure 5: **Models for different types of communication**. *a)* unicast from A to C; *b)* unicast from A to B, C and D; *c)* broadcast from A — which can also be implemented as in the previous case if $x(M_1) = x(M_2) = x(M_3)$ and *c)* multicast from A to C and D.

Figure 6 shows an example of how a multicast communication through email can be modeled as part of a temporal text network. The resulting network includes the sender of the message (*User A*) and two other actors (*User B* and *User C*) who where explicit recipients of the message. The fourth vertex $M_1 \in M$ represents the email and $x(M_1)$ corresponds to its text content (the subject line and the body content). In this case, the time attribute associated to each one of the edges represents the time when the message was delivered or received by the SMTP and POP3 servers allowing us not only to represent the communication flow, but also the effect of the channel and/or medium. Representing multiple emails as in the example above would lead to a full temporal text network.

One of the design principles we used to define our model was simplicity, to

[1]For simplicity we use the expression "all possible receivers" to refer to the community in which the information is spread, independent of whether the community is the whole Internet, the whole world or a set of members registered to a private site.

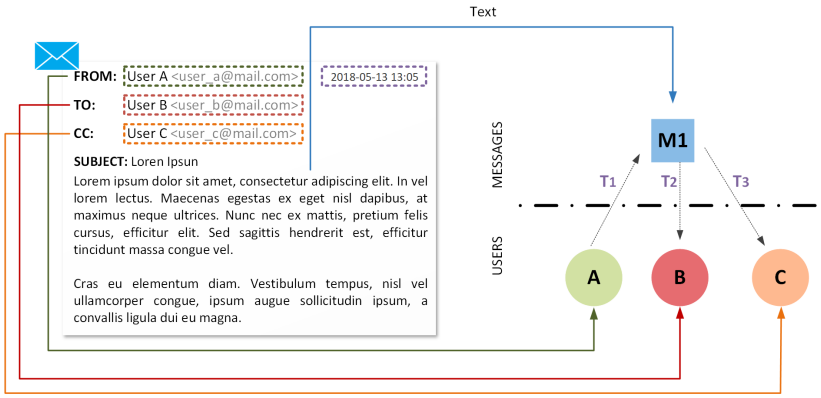


Figure 6: **Model of a multicast email as a temporal text network**. The entire text content of the email — including the subject line and the body — are encoded as a single message M_1. The sender of the email *(User A)* and the two friends — *User B*, *User C* — are modeled as individual actors. In this case, the ingoing and outgoing edges of the message contain a different time, indicating the delivery and receipt timestamps registered in the email servers.

make it tractable and general. On top of the basic model defined above, we can also easily add extensions to fit context-specific requirements.

With regard to the structure, we can straightforwardly add edges between messages to represent either information available from the data such as retweets on Twitter, or information deduced from the analysis of the data such as links indicating that one message is probably an answer to another, if we want to study information flows. Figure 7 shows, for example, the modeling process of a blog post M_1 and the associated comments from the readers $\{M_2, M_3, M_4\}$. In this particular case, we know the identity of each one of the authors, because they are authenticated in the web platform, but we do not know exactly who are the recipients of their comments. While we can assume by context that the blog post M_1 was read by follower B and that her message was then read by the blog owner A, it is uncertain what the third user (follower C) has read. We only know that the text produced by user C is a reply to the previous comment M_3, but we cannot infer if he has or has not read the previous messages M_1 and M_2. One possible way to model such scenario is to represent the relation between messages instead of the relation between messages *and* receivers. Similarly, in the example of Figure 8 the edges between messages are used to represent

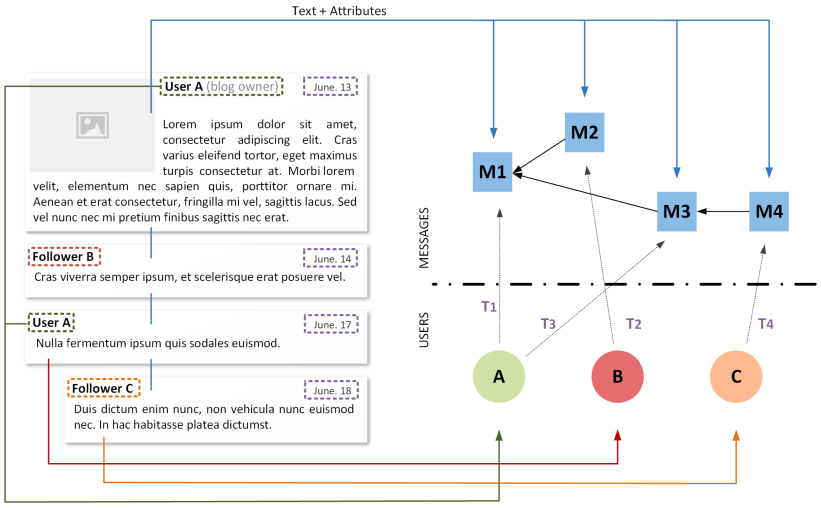


Figure 7: **Model of a blog post as a temporal text network**. The original data set contains a blog post M_1 and three comments (M_2, M_3, M_4); which are encoded as three individual messages. The three participants on the discussion — *User A*, *Follower B*, *Follower C* — are modeled as individual producers. In this case, the edges of the messages indicate the relation between their content.

retweets on a micro-blogging platform.

As we discuss in Section 3, this type of extension would nicely fit our analysis framework where one main class of operations transforms the data into a multilayer representation. Similarly, we may add edges between actors indicating other types of relations relevant for the analysis of the human information network such as indirect recipients. Figure 8 shows the modeling process of Twitter as a temporal text network. Unlike the previous communication channels we discussed, in Twitter the recipients of the information are encoded in the text of the messages rather than being explicit in the metadata (e.g., the edge (M_1, B, t_1) exists because actor A mentions B in the first message of the data set). In addition, Twitter users can also see messages from other users they are following, which in our model is represented by the actor-to-actor relations. This difference between intra- and inter-layer relations allows us to differentiate between direct and indirect communication in many social platforms.

In our basic model x represents a generic string of characters over some alphabet, whose interpretation will depend on the source of the data and the context of the analysis. For example, while the symbol # usually denotes the start of a filtering tag in online social networks such as Twitter or Instagram, in other media sites it is just an acronym for the word “number”. Therefore, for

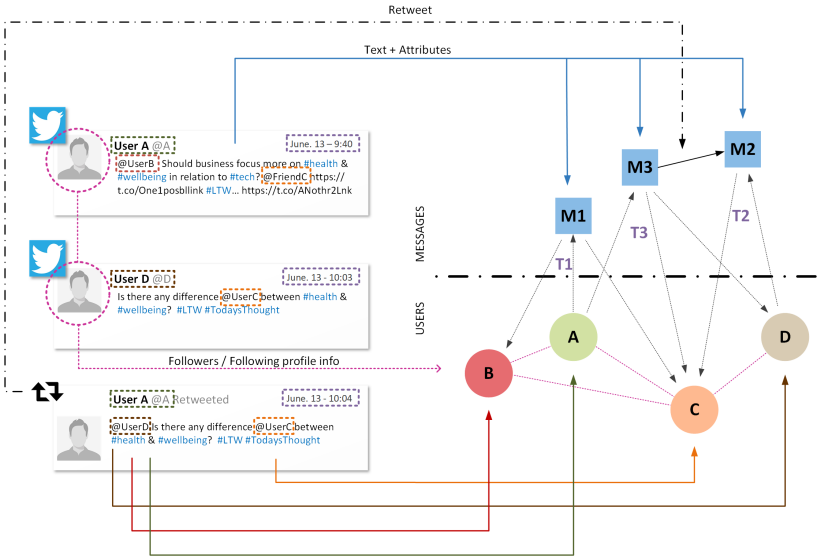


Figure 8: **Model of a Twitter network as a temporal text network**. The entire content of each tweet — including hashtags, urls and retweeted content — are encoded as messages. Senders — *@A*, *@D* — and mentioned users — *@B*, *@C*, *@D* — are modeled as individual actors. In this case, both the ingoing and outgoing edges of the message contain the same time, which indicates when the tweet has been sent. The edge between M_3 and M_2 indicates the retweet relation between both tweets.

specific application contexts additional attributes can be added for example to messages by providing special information, such as the hashtags included in the text in the case of Twitter (see Figure 8). In particular, we can think of having three types of information associated to each message:

1. The text, as in our basic model,
2. Metadata that is available in the specific data source used for the analysis, such as links to other resources (webpages, other tweets or multimedia content), like and retweet counts, or hashtags.
3. Additional information not directly available from the data source but obtained analyzing the text, for example through topic analysis.

Different types of temporal information have been used in existing works on temporal networks and temporal text analysis (see Appendix A.1). For example, time can represent actions from the users such as the time when a message is posted and/or the time when it is read as we did in the Twitter example.

Alternatively, times can be used to represent a physical property of the channel, as it happens in computer networks when there can be a transmission delay from the source to the destination of a message (See Figure 6). Finally, time can also be associated to the message, indicating for example the time interval when the message exists. Furthermore, this information can be complete or incomplete, so that if only the initial time of the interval exists we must assume the message is still valid at the time of analysis as we did when we described the blog posts; it can be private (accessible only to specific actors) or universally accessible by everyone.

3 Analyzing online communities

3.1 Community detection in multiplex networks

Community detection is one of the most popular social network analysis tasks, for which a large number of algorithms have been developed [43, 27]. The number of existing methods is not only justified by the importance of this task, but also by the absence of a unique definition of what a community is: different algorithms are often designed to identify different types of communities, and it is thus practically important for a social network analyst to have a toolbox with alternative algorithms.

The clique percolation method [96] is based on the intuition that the presence of a community can be observed in a social network through the presence of cliques, that is, sets of actors who are all adjacent to each other. This method has a set of features that make it well-suited to the discovery of communities in social networks: (1) it allows to specify how much connectivity is necessary to recognize the presence of a community (minimum clique size k), (2) it allows the same actor to be present in multiple communities (overlapping), and (3) it does not force all actors to be part of a community (partial).

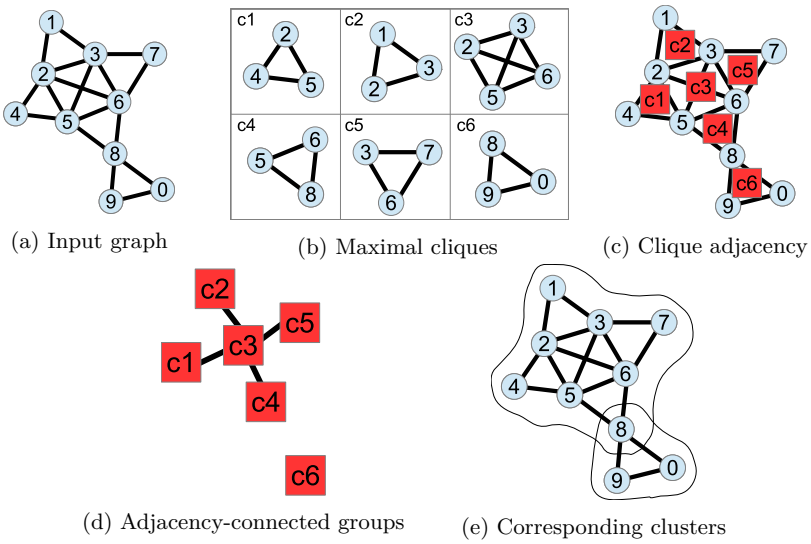
In this section we extend the clique percolation approach to deal with multiplex networks, to add clique percolation and the special features of its communities to the multiplex network analysis toolbox.

3.1.1 Clique percolation

The clique percolation method was introduced by Palla et al. in 2005 [96]. For a given k, CPM builds up communities from k-cliques, that is, complete subgraphs in the network with k vertices. Two k-cliques are said to be adjacent if they share $k-1$ vertices. A *k-clique community* is defined as a maximal union of k-cliques that can be reached from each other through a series of adjacent k-cliques. In general, if the number of links is increased above some critical point, a giant community would appear that covers a vast part of the system. Therefore, k is chosen as the smallest value where no giant community appears. CPM allows overlapping communities in a natural way as a vertex can belong to multiple cliques. Figure 9 shows an example of how CPM works. Given an input graph, first maximal cliques are identified, then adjacent cliques are grouped together to form communities.

3.1.2 Multiplex clique percolation

Our extended CPM algorithm for multiplex networks (CPM^M), of which we describe an implementation in the next section, follows the same main general steps of CPM. However, the concepts on which it is based must be extended to multiplex networks. In particular, we need to define what a clique on multiple edge types is, when two multiplex cliques can be considered adjacent, and how adjacent cliques should be grouped to build communities.



(a) Input graph (b) Maximal cliques (c) Clique adjacency

(d) Adjacency-connected groups (e) Corresponding clusters

Figure 9: A step-by-step view of the original clique percolation method

Cliques on multiple edge types While a clique on a simple graph is a well understood structure, defined as a set of vertices that are all adjacent to each other, the same concept can be extended in different ways for multiplex networks depending on how multiple edge types are allowed to contribute to the clique connectivity. Considering a specific set of edge types, we might require that a clique contains all the possible edges on all these edge types. In other words, a clique is formed by a combination of cliques on individual edge types. We refer to this type of cliques as AND-cliques.

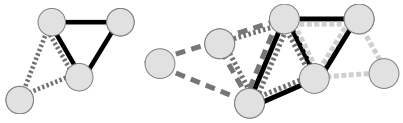
Definition 5 (k-m-AND-clique) *Let L_{ij} be the set of edge types between vertices i and j. We define a k-m-AND-clique as a subgraph in the multiplex network with k vertices that includes a combination of at least m different k-cliques from m different edge types. In other words, a k-m-AND-clique is a subgraph with k vertices C where*

$$|\bigcap_{i,j \in C} L_{ij}| \geq m \tag{1}$$

Similar to the case of cliques on simple graphs, we can define a concept of maximality for cliques in multiplex networks, where neither k nor m can be increased.

Adjacency and communities When cliques may exist on different edge types, the concept of adjacency should also consider this aspect.

To illustrate why, consider a definition of adjacency where k-m-AND-cliques only need to share $k-1$ vertices to be considered adjacent. As shown in Figure 10 (lhs) adjacent cliques do not necessarily share any edge types on all pairs and



(a) (b)

Figure 10: Adjacent cliques

they might share edge types only on their common pairs of vertices. It is worth noting that more diversity among the edge types in external connections of adjacent cliques results in denser internal connectivity. In addition, cliques at distance one still have to share some edge types on some of their pairs of vertices, as in the figure, but when the distance between cliques becomes greater than one, as in Figure 10 (rhs), they may end up having completely different edge labels. To enforce uniformity among edge types throughout the whole community, we thus need more constraints than what we can define at the level of clique adjacency.

Definition 6 (Multiplex clique-based community) *A multiplex clique-based $[(k-m)$-AND-clique$]_{(m',m'')}$ community is the maximal union of m'-adjacent k-m-AND-cliques where all cliques share at least m'' edge types on all of their pairs of vertices.*

Please notice that this is a very general definition, and in practice we can just use two parameters: k and $m(= m, m', m'')$.

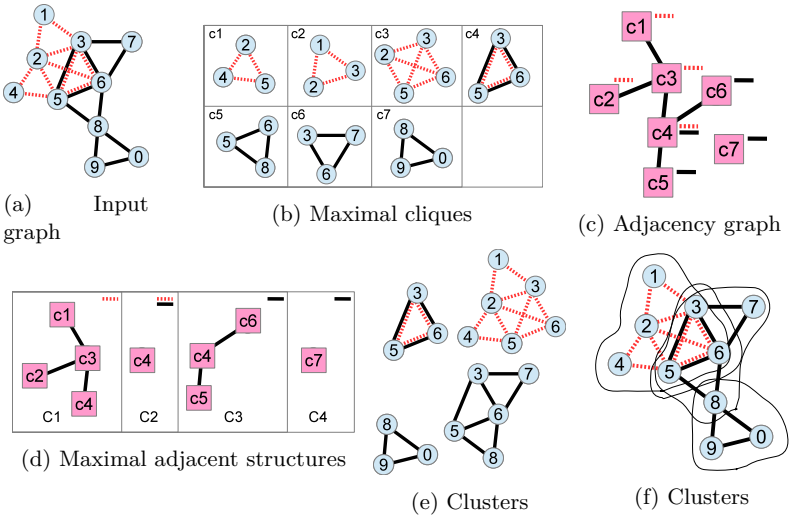
Algorithm In Figure 11 we have sketched an algorithm to detect communities according to our definitions, where without loss of generality we will assume that $m = m' = m''$. The details of the algorithm, including pseudo-code, are given in [124], and the algorithm is available in the library described in Appendix B.

The algorithm is divided into three parts, as in the original method: finding cliques, which can be done using an extension of Bron–Kerbosch's algorithm, building the adjacency graph, and extracting communities.

In a simple graph, each clique is included in exactly one community, therefore, communities can be identified from a clique-clique overlap matrix (see [**?**] for the details). However, this statement is not necessarily true for k-m-AND-cliques and the corresponding communities. Because of the more complicated relations between cliques, instead of the overlap matrix used in the original method we generate an adjacency graph as in Figure 11. In the graph we have indicated for each vertex the edge types where the corresponding clique is defined.

Two cliques can be included in at least one community if: (1) there exists a path between the corresponding vertices in the clique-adjacency graph, and (2) for all vertices in the path the corresponding cliques share at least m edge types on all of their pairs. Therefore, each community corresponds to a maximal tree in the clique-adjacency graph where condition (2) holds.

Figure 11 shows all the maximal trees from our clique-adjacency graph for $m \geq 1$. As we see, clique $c4$ can be included in three communities: C1, C2 and C3. No new clique can be added to these sets without reducing the value of m for which the cliques' constraint holds. As an example, community C4 satisfies the cliques' constraint for $m = 2$. Adding any adjacent clique to it, like c3, c5 and c6, the constraint would no longer hold for $m = 2$ because only one edge type would be common for both cliques. In Figure 11 for each maximal tree we have indicated the edge types where the constraint is satisfied, and we also show all communities in this example for $m \geq 1$.



(a) Input graph (b) Maximal cliques (c) Adjacency graph

(d) Maximal adjacent structures (e) Clusters (f) Clusters

Figure 11: A step-by-step view of our method

3.2 Community detection in temporal text networks

One reason to adopt a common model instead of defining ad hoc models for each application is to reuse existing analysis methods. While temporal text networks can be analyzed *directly*, for example studying dynamical processes such as text propagation in a similar way as in our motivating example, we can consider other strategies. Here we define two more approaches that can be used to analyze temporal text networks: we call them *continuous* and *discrete*.

The practical benefit of using these two approaches is that instead of developing new algorithms the analyst can focus on defining mapping functions encoding the model in a way that fits the data and analysis at hand. Then, these functions automatically generate model views on which existing algorithms can be computed.

3.2.1 Continuous analysis

The main idea behind this approach is to map the elements of the network (e.g., actors, messages, content, etc.) into an asymmetric metric space. This means that it is possible to compute distances between them.

Once distances are available, one can directly reuse existing data analysis methods for metric spaces, such as traditional distance-based and density-based algorithms (k-means, db-scan, etc.). Distances can also be used to retrieve relevant information from large temporal text networks, specifying an information query as an element of the metric space and retrieving those elements that are the closest. We present an example of this last type of analysis in the next section.

The first way of doing this is to use a network embedding method [48]. While network embedding was initially defined for simple graphs, more recent algorithms can be directly applied to attributed graphs [54]. Meanwhile, we foresee the definition of special versions of these algorithms that are specific for temporal text networks. Figure 12 shows an example of this first type of translation, where messages are the target of the analysis. The same approach can also be used to study other structures and elements in the temporal text network such as actors or combinations of actors and messages.

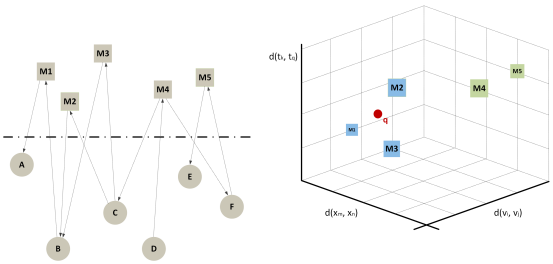
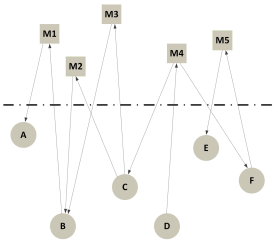


Figure 12: **Continuous approach: embedding**. (*left*) A temporal text network with 6 actors — circles — and 5 messages — squares; (*right*) the messages have been grouped into two clusters based on their topological, temporal and textual distance. The point marked with q represents a user's information requirements; in this example the left cluster (m_1, m_2, m_3) contains nodes that are more relevant for the user.

The second way to use the continuous approach is to directly define a distance function, without any explicit embedding into a coordinate system, so that the points form a metric space but have not an explicit position: only their relationships are defined. This approach is represented in Figure 13.

The two approaches may look similar: in both cases algorithms use distances, which can be computed after an embedding or are directly defined in the distance matrix. In practice, however, there can be relevant differences. For example, after embedding it is easier to index the data so that not all distances must be



	M1	M2	M3	M4	M5
M1		0.154	0.002	0.028	0.154
M2	0.634		0.641	0.505	0.448
M3	0.215	0.580		0.939	0.869
M4	0.073	0.154	0.939		0.711
M5	0.455	0.401	0.869	0.711	

Figure 13: **Continuous approach: distance-based**. (*left*) A temporal text network with 6 actors — circles — and 5 messages — squares; (*right*) a messages' distance matrix is obtained from the network topology and time attributes.

computed when algorithms are executed, leading to lower computation time. On the other hand, the direct usage of a distance function is more natural if distances are asymmetric, e.g., when $d(M1, M2) \neq d(M2, M1)$. Asymmetric distances often appear in temporal and directed networks, that are both features of our model.

3.2.2 Discrete analysis

The main idea behind this approach is to encode temporal and textual information into network structures, in particular layers in a multilayer network, so that methods from multilayer network analysis can be directly applied [68, 33]. This can be done by defining a mapping function from time and text into a discrete set of classes that are relevant for the analysis. Then, topic-and-time-based user centrality, topic-and-time-based relevance, as well as community detection algorithms can be used. An example of this last type of analysis on real data follows in the next section.

Textual discretization is typically performed using methods from Natural Language Processing such as topic, sentiment or semantic analysis. The main objective of the procedure is to group together messages whose contents have similar characteristics. *Time discretization* is apparently simpler, because only the cutting points between time slices must be indicated. However, also time discretization presents many options. First, there are often many ways of defining the cutting points, leading to different results. Second, after the cutting points have been defined there can still be different ways of distributing network structures into the slices. For example, if we want to discretize messages, we can place a message m_i in a specific interval (t_a, t_b) either if the incoming edge $e = (v_j, m_i, t)$ exists in the interval (t_a, t_b), if all the edges from/to m_i exist in the interval, if at least one of the out-going edges $e = (m_i, v_j, t)$ exist in the interval, etc. Finally, we use the term *multiple discretization* when both textual and time discretization are applied together to generate the different groups.

Under this procedure, our model would produce a k-partite network with one partition for each new cluster of messages and one partition for the actors. The

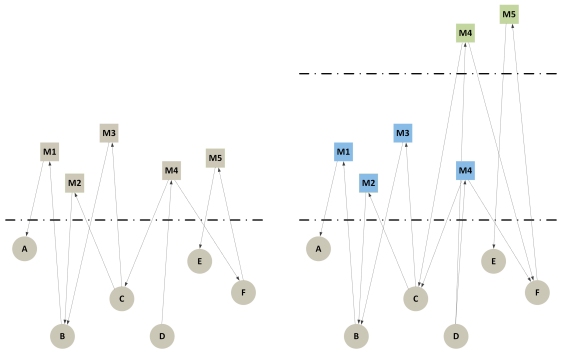


Figure 14: **Textual discretization**. (*left*) A temporal text network with 6 actors — circles — and 5 messages — squares; (*right*) the network has been discretized into two clusters — the top one with 2 messages, the bottom one with 4 — based on the topic of the messages.

procedure to generate such network is straightforward once the discretization function is defined. Figure 14 shows an example of textual discretization where the resulting 3-partite network contains the original layer of actors A, and two message layers with 2 and 4 messages each grouping together messages about the same topic. In this particular example, $x(M_4)$ was related to both topics, therefore the message M_4 appears in both layers. A similar network structure will emerge from time discretization.

An additional operation on multilayer networks that can be applied to the discretized data is projection, creating edges in one layer based on the information present in another layer. In the resulting multilayer network, a new edge $e_{ij}^{[l]} = (v_i, v_j)$ is created if there is a message m_k in the partition $l \in L$ of the original network with: a) an edge (v_i, m_k) from actor v_i to message m_k and b) an edge (m_k, v_j) from message m_k to actor v_j. Weights can also be added to the new edges, using various methods. Figure 15 shows one possible projection from the network in Figure 14. In this example the content of the messages (and more in general also the time) are now encoded into the relations between actors.

The main advantage of using a projected multilayer network to analyze temporal text networks is the vast available literature that has targeted this type of data. In Section 5.2 we use the approach described above together with a clustering algorithm for multilayer networks to find communities of actors discussing about the same topics during the same time spans.

3.3 Comparison of layers

Several studies have investigated the connection between layer similarity and other properties of the network. For example, we know from previous research

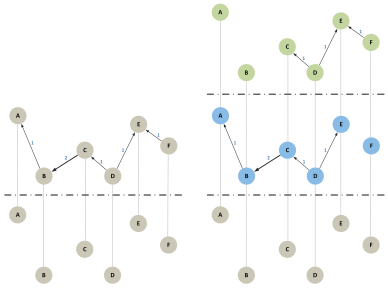


Figure 15: **Projection**. (*left*) A projection of the message layers into the actor layer in the original bipartite network in Figure 14-*left*. The projected multilayer network has 6 actors, 12 nodes and 5 weighted edges; (*right*) a similar projection using the 3-partite network described in Figure 14-*right* which generates a multilayer network with 6 actors, 18 nodes and 7 weighted edges.

that the relationships between layers have an impact on dynamic processes such as behaviour and information diffusion [110]. In the VirtEU project, layer similarity measures have been used to compare Twitter datasets from different IoT events (as described in the previous deliverables with status reports) and are used to compare online conversations on different topics, where each topic is represented as a layer in a temporal text network, as in Section 5 of this report.

Being able to measure relationships between layers is also essential to validate models aimed at explaining the formation of empirical multilayer networks [84, 90]. While the problem of comparing different networks has been thoroughly investigated in the literature, the problem of quantifying layer similarity where the same nodes can be present in multiple layers (which characterizes multiplex networks) has not been studied in a systematic and comprehensive way so far.

In the literature, we can find a large number of works using layer similarity measures, but most use them as a tool to study other phenomena such as multiplex network generation [64, 84, 90], link prediction [1, 105] and spreading processes [110]. As a result, different works use the same or very similar approaches presented with different names, the relationships between several of these similarity measures have not been explored, and there are no guidelines on how to quantify layer similarity in multiplex networks, e.g., how to choose the appropriate measure given a specific dataset. In addition, various potentially useful layer comparison measures have not been considered yet.

Therefore, in this section we define a set of layer comparison measures. This is part of the content published in [18], we also provide an empirical study of the relationships between different measures, compared on several real datasets.

3.3.1 Layer similarity functions

Given a property matrix $\mathbf{P}$ where each row represents a layer, we can compare two layers in three main ways. The first is to summarize each row using an

aggregation function f and compare $f(\mathbf{p}_{l_1})$ to $f(\mathbf{p}_{l_2})$. For example, if the property matrix contains node degrees we can compare the layers' average degrees mean($\mathbf{p}_{l_1}$) and mean($\mathbf{p}_{l_2}$). Comparing the distribution of values in $\mathbf{p}_{l_1}$ and $\mathbf{p}_{l_2}$ is the second way to compare layers. As an example, we can compare degree distributions on different layers and find that both fit well a power law distribution with the same exponent. The third way is to compare p_{s,l_1} with p_{s,l_2} for all s. As an example, we can compute degree correlation to check whether nodes with a high (resp., low) degree on one layer tend to have a high (resp., low) degree also on the other layer.

Comparing aggregations of layer property vectors This first class of comparison methods is based on comparing $f(\mathbf{p}_{l_1})$ to $f(\mathbf{p}_{l_2})$ using various functions (f) aggregating each layer into a single value. Typical choices are basic statistical summary functions such as mean, max, sum, skewness and kurtosis, combinations of the simple statistics, such as the coefficient of variation (the ratio between the standard deviation and the mean), the Jarque-Bera statistics (a combination of skewness and kurtosis), or the Shannon entropy of the distribution. These methods are summarized in Table 2.

Then, given $f(\mathbf{p}_{l_1})$ and $f(\mathbf{p}_{l_2})$ we can compare them, and in our experiments we have used their relative difference, i.e. $2 \cdot (|f(\mathbf{p}_{l_1}) - f(\mathbf{p}_{l_2})|)/(|f(\mathbf{p}_{l_1})| + |f(\mathbf{p}_{l_2})|)$.

Notice that depending on the property matrix these measures correspond to various existing network summaries. For example, the mean function may return the average degree (when applied to property matrices about node degrees, or the global clustering coefficient also known as transitivity index (for node clustering coefficients), or the average path length for property matrices about dyads and geodesic distances (which in the field of chemistry coincides with the Wiener index).

Whether the multiplex network is node-aligned or not, does not pose any problems regarding the computation of the functions in Table 2. These functions are computed for each layer, only for the nodes existing on the layer, so if some nodes are not present they are just not considered in the computation. Similarly, also the measures in Table 3 can be easily computed for node-aligned and for node-non-aligned as the the frequency distributions are computed layer by layer. However, the results of the function and of the comparison can be strongly affected by the alignment, as shown in [18].

Comparing distributions of layer property vectors

While using a single value to compare layers can provide some useful knowledge about the multiplex network, for example by highlighting the presence of denser or more clustered layers than others, looking at the whole distribution of values in the property matrix can reveal other types of relationships among layers. From a statistical point of view, some ways are open to pursuing this task. The first one consists in comparing the moments of two distributions. For example, it is possible to compare the first four moments, even if from a theoretical point of view this is not completely sufficient. Another possible approach consists in comparing the distributions directly. In this case, we have to apply to each property vector a function fr($\mathbf{p}_l$) that derives the relative frequency

Table 2: Summary of common aggregation functions for *property matrices*

Name	Function
$mean(\mathbf{p}_l)$	$\frac{\sum_s p_{s,l}}{card(\mathbf{p}_l)}$
$sd(\mathbf{p}_l)$	$\sqrt{\frac{\sum_s (p_{s,l} - mean(\mathbf{p}_l))^2}{card(\mathbf{p}_l)}}$
$skew(\mathbf{p}_l)$	$\frac{\sum_s (_{s,l} - mean(\mathbf{p}_l))^3}{card(\mathbf{p}_l) sd(\mathbf{p}_l)^3}$
$kurt(\mathbf{p}_l)$	$\frac{\sum_s (p_{s,l} - mean(\mathbf{p}_l))^4}{card(\mathbf{p}_l) sd(\mathbf{p}_l)^4}$
$entropy(\mathbf{p}_l)$	$\sum_{k=1}^{k} fr_{k,l} \log fr_{k,l}$
$CV(\mathbf{p}_l)$	$\frac{sd(\mathbf{p}_l)}{mean(\mathbf{p}_l)}$
$Jarque - Bera(\mathbf{p}_l)$	$\frac{card(\mathbf{p}_l)}{6}\left(skew(\mathbf{p}_l)^2 + \frac{(kurt(\mathbf{p}_l)-3)^2}{4}\right)$

$fr_{k,l}$ is the relative frequency of the k-th value of the property vector $\mathbf{p}_l$ in a generic layer l

distribution. In case of discrete distributions, such as the degree distribution, given a property vector $\mathbf{p}_l$ we derive the disjoint values $p_{k,l}$, $k = 1, \ldots, K$, and we associate to each value the relative frequency $fr_{k,l}$.

In case of continuous distribution, or in case of very large networks in which also the discrete distributions take a wide range of values, the function fr($\mathbf{p}_l$) derives histograms. We first divide the range of values of the property vector into K equal interval, or bins, $[b_{(k-1)}, b_k]$, with b_0 being the minimum value in the property matrix and $b_{K,l}$ being the maximum value in the property matrix[2]. Then we associate the relative frequency fr_k to each interval. Note that the bins of all histograms for all layers must be the same. Then we have to compare only the relative frequency distributions. This procedure is very fast and efficient also for very large networks.

Given the frequencies or histograms, in order to compare two layers we can use the distance between observed distributions based on distance between histograms, namely, the dissimilarity index (ID), the Kullback-Leibler divergence D_{KL}, the Jensen-Shannon divergence D_{JS} or the Jeffrey divergence D_J, as defined in Table 3. In the following, we do not consider the Jeffrey divergence, as the Jensen-Shannon divergence is its smoother version. Note that this kind of comparison can be made both for node-aligned and for not node-aligned multiplexes.

Comparing individual structures The main feature of multiplex networks is that the same structure can be present or not, and have different characteristics, on each layer. For example, a node can be present in one layer and not in the other, or the same node may have different degrees depending on the layer. Therefore, a peculiar set of measures to compare layers relies on the comparison of the structures of interest, one by one.

[2]If we only compare two rows, we can also choose the minimum and maximum values in those rows.

Table 3: Main methods to compare distributions across layers

Name	Notation	Function
Dissimilarity index	$ID(\mathbf{p}_{l_1}, \mathbf{p}_{l_2})$	$\frac{1}{2}\sum_{k=1}^{K} \lvert fr_{k,l_1} - fr_{k,l_2} \rvert$
Kullback-Leibler	$D_{KL}(\mathbf{p}_{l_1}, \mathbf{p}_{l_2})$	$\sum_{k=1}^{K} fr_{k,l_1} \log \frac{fr_{k,l_1}}{fr_{k,l_2}}$
Jensen-Shannon	$D_{JS}(\mathbf{p}_{l_1}, \mathbf{p}_{l_2})$	$\frac{1}{2}(\sum_{k=1}^{K} fr_{k,l_1} \log \frac{fr_{k,l_1}}{\hat{fr}_k} + fr_{k,l_2} \log \frac{fr_{k,l_2}}{\hat{fr}_k})$
Jeffrey	$D_J(\mathbf{p}_{l_1}, \mathbf{p}_{l_2})$	$\sum_{k=1}^{K} fr_{k,l_1} \log \frac{fr_{k,l_1}}{fr_{k,l_2}} + \sum_{k=1}^{K} fr_{k,l_2} \log \frac{fr_{k,l_2}}{fr_{k,l_1}}$

where: $\hat{fr}_k = \frac{fr_{k,l_1} + fr_{k,l_2}}{2}$

Two main cases are possible. In property matrices indicating the existence of different structures on the different layers, we only have two values, 0 and 1. While represented as numbers, these are in fact just nominal values indicating that the structure is present on the layer. For these binary matrices specific methods can be used, checking the overlapping or more in general, the common existence (or common absence) of structures across layers. For numerical matrices containing generic numbers, e.g., node degrees, other methods are more appropriate, as described in the following two sections.

Binary properties

When a structure can be present or not on different layers, a basic way to compute the similarity between layers is to quantify the overlapping of these structures, that is, how often the same structure appears or not on more than one layer. This is typically the case when the observation function defining the property matrix checks the existence of the structure.

Measures of overlapping have been defined and redefined many times during the last few years in different papers, but most definitions can be generalized using property matrices as:

$$C\mathbf{p}'_{l_1} \cdot \mathbf{p}_{l_2}, \tag{2}$$

where C is some normalization function. Most (but not all) measures in the literature compare edges across layers, this being the result of the traditional edge-based definitions of multiplex networks such as adjacency matrices. In our definition, the usage of property matrices allows us to apply similar comparisons to various other properties.

Consider two binary property vectors $\mathbf{p}_{l_1}$ and $\mathbf{p}_{l_2}$. Let us denote with:

- $a = \mathbf{p}'_{l_1} \cdot \mathbf{p}_{l_2}$ the number of properties that l_1 and l_2 share;
- $b = \mathbf{p}'_{l_1} \cdot (\mathbf{1} - \mathbf{p}_{l_2})$ the number of properties that l_1 has and l_2 lacks;
- $c = (\mathbf{1} - \mathbf{p}_{l_1})' \cdot \mathbf{p}_{l_2}$ the number of properties that l_1 lacks and l_2 has;
- $d = (\mathbf{1} - \mathbf{p}_{l_1})' \cdot (\mathbf{1} - \mathbf{p}_{l_2})$ the number of properties that both l_1 and l_2 lacks;
- $m = a + b + c + d = length(\mathbf{p}_{l_1}) = length(\mathbf{p}_{l_2})$

Table 4: Similarity functions for binary property matrices. Column C indicates the normalization function in Eq. 2. For the two functions also considering the non-existence of structures on both layers, we only provide the standard definition not based on the product of property vectors

Name	Normalization function C	Standard notation
Russel-Rao	$\frac{1}{length(\mathbf{p}_{l_1})}$	$\frac{a}{m}$
Jaccard	$\frac{1}{length(\mathbf{p}_{l_1})-(\mathbf{1}-\mathbf{p}_{l_1})'\cdot(\mathbf{1}-\mathbf{p}_{l_2})}$	$\frac{a}{m-d}$
Coverage	$\frac{1}{length(\mathbf{p}_{l_1})}$	
Kulczyński	$\frac{1}{2}(\frac{1}{\|\mathbf{p}_{l_1}\|_1}+\frac{1}{\|\mathbf{p}_{l_2}\|_1})$	$\frac{a}{2}(\frac{1}{a+b}+\frac{1}{a+c})$
Simple matching coeff. (SMC)	NA	$\frac{a+d}{m}$
Hamann	NA	$\frac{a+d-(b+c)}{m}$

Then, the binary similarity functions can be summarized as in Table 4.

Numerical properties Depending on the reason why we are computing the similarity between layers, we can use different approaches. As each layer is represented as a vector in a property matrix, one way is to compute vectorial distances such as Euclidean distance or cosine similarity. Another popular way to compare numerical layer property vectors is to compute correlations. An example of this is the so-called inter-layer correlation measure, which is just the Pearson coefficient computed on two node degree property vectors [10, 91]. It is interesting to notice that in the literature correlations across layers have been almost always computed on node degrees, and in [8] also on clustering coefficients. However, correlations can be in fact be computed on any property matrix.

In addition, we would like to stress that Pearson correlation here is used as measure of accordance of numerical vectors, and then it can be used also when usual statistical assumptions are not completely fulfilled. However, in case of highly skewed distributions, or in case of severe and numerous outliers, the Spearman rank correlation is a good solution. For this reason, we suggest to use them jointly.

Finally, when computing correlations in generalized multiplex networks a choice must be made on how to handle actors not present in all layers. The choice we adopted in our experiments was to discard pairs where at least one of the two values was missing, which is a typical option in statistical software packages.

3.3.2 Guidelines

From our literature study, theoretical framing and from the experiments we reported in [18] it appears how layer comparison measures can be very valuable and often succeed in practice to characterize the structure of multiplex networks, but they are not always straightforward to use. Therefore, in this section, we list a set of guidelines on how to use these measures.

Table 5: Similarity functions for numerical property matrices. The function $\rho(\cdot)$ provides the ranks of the values in the property vectors

Name	Function
Cosine Similarity	$\frac{\mathbf{p}'_{l_1} \cdot \mathbf{p}_{l_2}}{\|\mathbf{p}_{l_1}\| \cdot \|\mathbf{p}_{l_2}\|}$
Person Correlation Coefficient	$\frac{[\mathbf{p}_{l_1} - mean(\mathbf{p}_{l_1})]' \cdot [\mathbf{p}_{l_2} - mean(\mathbf{p}_{l_2})]}{\|[\mathbf{p}_{l_1} - mean(\mathbf{p}_{l_1})]\| \cdot \|[\mathbf{p}_{l_2} - mean(\mathbf{p}_{l_2})]\|}$
Spearman Correlation Coefficient	$\frac{[\rho(\mathbf{p}_{l_1}) - mean(\rho(\mathbf{p}_{l_1}))]' \cdot [\rho(\mathbf{p}_{l_2}) - mean(\rho(\mathbf{p}_{l_2}))]}{\|[\rho(\mathbf{p}_{l_1}) - mean(\rho(\mathbf{p}_{l_1}))]\| \cdot \|[\rho(\mathbf{p}_{l_2}) - mean(\rho(\mathbf{p}_{l_2}))]\|}$

One important aspect to consider when choosing which function to use is the distribution of values in the property matrix. Among the criteria that can be used to characterize layer property vectors and comparison functions, the following appear to be useful:

- Sparsity: A layer property vector is sparse if the number of 0s is much higher than the number of non-0 values.
- Degeneracy: A layer property vector degenerates if its values are (almost) constant. Sparsity is a special case of degeneracy.
- Linearity: A layer property vector is linear if the values in the vector and their rank are linearly correlated.
- Scale invariance: a similarity function is scale invariant if it does not (significantly) change when one or more layer property vectors are multiplied by a constant.

We now list our guidelines, divided into four main areas.

Number of measures The number of available measures is very large, considering that the fifty options used in our experiments are only some of the measures we can obtain using different combinations of property matrices and observation functions. While the choice of the measures to be used for a specific empirical network is of course influenced by what the analyst is interested in, e.g., degree-based similarity, betweenness-based, or specific motifs that are motivated by the application context, our experiments show that different measures highlight different types of similarities.

At the same time, even during exploratory analyses where it is often useful to compute several measures to get a good overview of the data, it can be practically preferable to identify a small number of measures. This can be due to time constraints, if the data is large, but also to the need of producing results that are easy to interpret and present. The choice of which measures to use can be simplified using the correlation plots in [18]. Groups of measures producing highly correlated values can be identified, and one measure for each group can be chosen. In particular, JS, KL and D divergences are similar, and JS divergence can be used from this group. Jaccard, coverage and Kulczyński are similar, and

Jaccard or coverage can be used — with the latter highlighting how the non-overlapping structures are distributed across the two layers, e.g., if one layer is containing the other.

When comparing layers by comparing a single value, particular attention should be paid to the so called discriminative power or uniqueness of the measure, i.e., the capability of a measure of taking different values on non-isomorphic networks. For example, while mean is not a representative measure for non-regular distributions, it can still be used to compare two distributions, such as degree distributions. But not alone, because the same degree does not imply the same topology.

While min can be useful in general to characterize a distribution if used together with other statistical summaries, it does not appear to be very useful to compare layers where there is typically at least one node having value 0. For example, min degree is 0 for all layers for most networks. On the contrary, max can be useful, e.g., to include the size of the layers in the comparison.

Node-alignment The choice of whether a node-aligned or generalized multiplex model should be used is often clear from the context. For example, we would typically not align nodes when layers represent different social network sites, to represent the fact that users may not have accounts on some sites, while we would typically align nodes in a multirelational network about people interacting in multiple ways, where not having edges on a layer does not imply that the person cannot interact in that specific way.

However, the choice may have a significant influence on the results of the analysis. Node-alignment may lead to some degeneracy. As expected, node-existence measures become useless, but also other cases are affected, such as measures 11-16 (degree) and 27-32 (clustering coefficient).

Measures based on node existence may also help us interpreting the results of other measures. So, before using link-based measures (such as edge Jaccard) it is important to check node overlapping to understand whether comparing higher order structures is meaningful, or whether the results will just be a consequence of the limited amount of node overlapping across layers.

Rank correlation can suffer from node alignment because of false tie resolution, and also Pearson correlation results may become less evident, with positive and/or negative correlations being lost or decreased depending on the type of networks.

Sparsity SMC and Hamann are only useful for non-sparse, non-degenerated cases, typically corresponding to node existence on generalized networks. Russel-Rao also suffers if property vectors are sparse. As an example, these measures do not work well for triangle-existence property matrices in general.

Linearity Having non-linear distributions of values in the property vectors, as it is the case for degree property matrices, is not problematic when computing linear correlation. Linear correlation (Pearson) is often preferable to rank correlation, which can be problematic in case of generalized networks (because of null values) and also for node-aligned networks (because of the many nodes with the same values).

3.3.3 Other practical considerations

Our framework captures several measures appeared in the literature: node activity overlapping [91], global overlapping of edges [13] and absolute binary multiplexity [46] are applications of the Russel-Rao function to node and edge existence property vectors, average edge overlap [31] and [8] are respectively the Jaccard and coverage functions applied to edge existence. A general recommendation is to use the original names, as we do in this report: all the measures used in this work and mentioned in this paragraph are applications of existing proximity measures, most of them well known to data analysts. Calling them by their name, such as edge Jaccard, makes it simpler to understand when it is reasonable to apply them if we already know the original measure.

Also, notice that our framework allows the definition of a large number of other functions not mentioned here, also considering directed/undirected networks, weights, and other meso-structures such as motifs. Other network summary functions that are not specific for multiplex networks can also be obtained as combinations of property matrices and observational functions. Examples are order (node existence + sum), size (edge existence + sum), density (edge existence + mean), average path length (dyad distance + mean), etc. We believe that splitting the problem of computing layer similarities into the two problems of (1) deciding what to observe and (2) deciding how to compare these observations using existing generic comparison functions gives the analyst the ability to easily generate custom layer comparisons that are appropriate for the problem at hand.

3.4 Data visualization

Sociograms, that is, visual representations of the relationships between individuals, have been used since the origins of social network analysis, a notable example being Moreno's seminal work introducing this concept [87]. In the same book we also find a sociogram representing a multiplex network, where multiple types of relationships between the same group of individuals coexist. However, while several layout algorithms for simple graphs have been developed since then, developments in visualization methods for multiplex networks have been limited.

One natural way to visualize a multiplex network is to treat it as an edge-typed multi-graph using different colors and line styles to distinguish between the different edge types, and as done by Moreno himself. However, this option can quickly lead to a very dense representation hiding relevant network structures even for very small networks [106]. Alternatively, different types of connections can be sliced into different layers, with the same node replicated on multiple layers. This approach has been used in the literature to represent social/historical [95, 83] networks, but also several other types of multiplex networks, from traffic [74, 30] to biological [30] and financial networks, sometimes visualized in a 2.5-dimensional space.

However, replicating the same node on multiple layers introduces a new

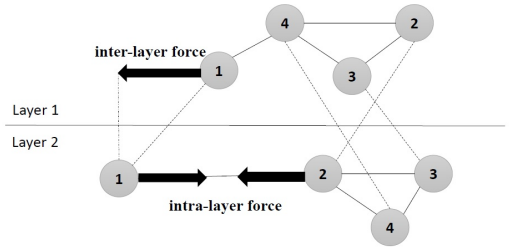


Figure 16: The effect of intra-layer and inter-layer forces on node positions

question: how should the positions of multiple occurrences of the same node relate to each other? Two main approaches for visualizing multiplex networks sliced into layers have been used: visualizing each layer independently of the others, or keeping the same layout in all layers, so that all occurrences of the same node will result aligned on a straight line if the layers are visualized one besides the other.

Using the first approach we can appreciate the structure of each layer. Using the second approach it is easier to find the same nodes across different layers, but this does not necessarily help in understanding relationships between whole layers, and can also be misleading.

In this section we present a simple algorithm that can replicate the approaches listed in the previous paragraph and can also produce new intermediate layouts between the ones mentioned above, following the principles explored in the context of dynamic graph drawing [17]. We call the general layout and algorithm described in this section *multiforce*.

Multiforce is based on a force-directed algorithm (in this section and in the software library used to produce the analyses included in this deliverable we extend the popular Fruchterman-Reingold method) and uses two main types of forces: *intra-layer* and *inter-layer*, that can be tuned to impact specific layers more or less than others. Intra-layer forces attract neighbors inside the same layer, making them closer, as in traditional layouts for monoplex networks. Inter-layer forces try to align instances of the same node on different layers[3]. Figure 16 gives an intuition of how these forces operate. In addition, we use repulsive forces as in the original algorithm, and also gravity for the cases where no inter-layer forces are active and the network contains more than one component.

3.4.1 The *multiforce* layout

Multiforce belongs to the slicing class of multiplex layouts, and is different from existing approaches, because it allows a balancing of the effects of intra-layer

[3]In theory inter-layer forces can also be used to visualize more general networks, where edges can cross layers, but here we focus on multiplex networks.

and inter-layer relationships. Multiforce extends the Fruchterman-Reingold algorithm [44]. This is one of the most popular options available in graph analysis software packages, although it is not a state-of-the-art method; several variations of this approach have been developed, but its simplicity allows us to focus on the simple variation of the way in which forces are defined, which can then be adapted to more complex algorithms. As mentioned in the introduction, multiforce is based on two types of attractive forces. The nodes are positioned on a set of planes, one for each layer or type of relationship – this setting is sometimes called 2.5-dimensional, because it looks 3-dimensional but the z-coordinates of the nodes are fixed and limited to the number of planes/layers. In this report we visualize all layers on the same plane, because a 3-dimensional graph is visually intriguing but not easy to understand without the option of interactively rotating the diagram. Intra-layer forces, that can be weighted differently in each layer, attract pairs of nodes connected on the same layer. Inter-layer forces influence the position of nodes in different layers connected by inter-layer edges, or corresponding to the same node in the case of multiplex networks.

The same idea behind multiforce was already proposed and tested in the context of dynamic graphs, where layers are not unordered as in the case of multiplex networks but are organized in a sequence.

The pseudo-code of multiforce is presented in Algorithm 1, and the algorithm is implemented in our software library. The algorithm takes a multiplex network $G = (N, L, V, E)$ as input, where N is a set of nodes, L is a set of layers, (V, E) is a graph and the elements of V are pairs $\langle$node, layer$\rangle$. We notate v.layer the layer of an element $v \in V$ and v.node the node corresponding to an element $v \in V$.

Lines 13-29 are the same as in the original algorithm, and compute the displacement of each node based on its neighbors (attractive forces) and other nodes (repulsive forces), with the addition of weights that can be used to specify on which layers the layout should be computed according to the original algorithm (27-28). Lines 30-37 extend the original algorithm and compute the displacement caused by the position of the node on other layers, to control node alignment. This is also weighted, to allow the user to turn this feature on and off for all or some layers (34-35). In our tests the function *cool* (45) reduces t linearly, so that it becomes 0 at the last iteration.

Some details of our algorithm can also be changed without affecting its underlying idea. First, we can modify lines 6-12 to assign the same initial random coordinates to the same node across different layers, anticipating line 8 before the for loop. A weighting factor INLA[v] can also be added at line 20, so that both attractive and repulsive forces are reduced or reinforced together. Finally, lines 41 and 42 have been retained from the original algorithm and ensure that the nodes do not exit the frame specified by the user, but are not necessary if the final coordinates are re-scaled to fit it or a gravity force is added to control the spreading of the nodes so that all slices retain similar extreme coordinates.

Algorithm 1 Multiforce

Require: $G = (N, L, V, E)$: a multiplex network
Require: W: width of the frame
Require: L: length of the frame
Require: #iterations
Require: INLA, INTERLA: intra- and inter-layer weights

f_r = function(z, k)\{ return k^2/z; \}
f_a = function(z, k)\{ return z^2/k; \}
area := $W \cdot L$
$k := \sqrt{\frac{\text{area}}{|N|}}$;
$t := \sqrt{|N|}$;
for $(n \in N)$ **do**
  for $(v \in V$ s.t. v.node $= n)$ **do**
    (x, y) = random coordinates;
    pos$[v] = (x, y)$
    $z[v]$:= index(v.layer);
  end for
end for
for ($i = 1$ to #iterations) **do**
  // calculate repulsive forces
  for $(v \in V)$ **do**
    disp$[v] := \vec{0}$;
    for $(u \in V)$ **do**
      if $(u \neq v$ and u.layer $= v$.layer$)$ **then**
        $\triangle$:= pos$[v]$ − pos$[u]$;
        disp$[v]$:= disp$[v]$ + $(\triangle/|\triangle|) * f_r(|\triangle|)$;
      end if
    end for
  end for
  // calculate attractive forces inside each layer
  for $((u, v) \in E)$ **do**
    $\triangle$:= pos$[v]$ − pos$[u]$;
    disp$[v]$:= disp$[v]$ − $(\triangle/|\triangle|) * f_a(|\triangle|, k) *$ INLA$[v]$;
    disp$[u]$:= disp$[u]$ + $(\triangle/|\triangle|) * f_a(|\triangle|, k) *$ INLA$[u]$;
  end for
  // calculate attractive forces across layers
  for $(n \in N)$ **do**
    for $(\{u, v\}$ with $u, v \in V$, u.node $= v$.node $= n)$ **do**
      $\triangle$:= pos$[v]$ − pos$[u]$;
      disp$[v]$:= disp$[v]$ − $(\triangle/|\triangle|) * f_a(|\triangle|, k) *$ INTERLA$[v, u]$;
      disp$[u]$:= disp$[u]$ + $(\triangle/|\triangle|) * f_a(|\triangle|, k) *$ INTERLA$[u, v]$;
    end for
  end for
  // assign new positions
  for $(v \in V)$ **do**
    pos$[v]$:= pos$[v]$ + (disp$[v]$/|disp$[v]$|) $*$ min(disp$[v]$, t);
    pos$[v].x$:= $\min(W/2, \max(-W/2, \text{pos}[v].x))$;
    pos$[v].y$:= $\min(L/2, \max(-L/2, \text{pos}[v].y))$;
  end for
  // reduce the temperature
  t := cool(t);
end for

3.4.2 Main algorithmic settings

The multiforce algorithm can produce both existing and new layouts using the following settings:

1. **Multi-graph:** this layout, where each node has a specific position that does not depend on the layer and all edge types are considered when computing the node coordinates, is obtained by setting the same positive value for intra-layer weight in each layer and infinite (or, in practice, very high) inter-layer weights. The intra-layer weights will then keep nodes aligned across layers, and intra-layer forces will produce the layout by moving these "node pillars" around.

2. **Sliced, independent**: this layout corresponds to the application of the force-based algorithm on each layer, and is obtained by using positive intra-layer weights and setting inter-layer weights to 0.

3. **Sliced, aligned on layer x:** this layout is computed based on one of the layers, and nodes are kept aligned on the other layers. It is obtained by specifying a positive intra-layer weight for layer x and setting the other intra-layer weights and the inter-layer weights to 0.

4. **Balanced:** this intermediate layout is obtained by setting a positive weight (for example, 1) for all inter- and intra-layer forces.

In [41] we provide a comparison between multiforce and a traditional non-multiplex force-based layout algorithm, in addition to a qualitative characterization of the diagrams produced by our algorithm.

4 MeetUp data analysis

MeetUp[4] is an online service used to organize groups that host in-person events for people with similar interests. Most of the events organized by these groups are periodical and they have a stable core set of participants that form a community of practice focused on one or several activities. Some examples of events organized by IoT MeetUp groups include contact meetings with investors to learn about new business models, hackathons for playing with the latest technologies or introductory workshops aimed for new members. In the social platform, the specific interests of the *groups*, their *organizers* and the *participants* attending the *events* are expressed using a set of *keywords*.

In this section we analyze the composition of the MeetUp groups to identify what elements (things, processes or domains) constitute, for each of these types of actors (groups, organizers and members) the IoT space (see Section 1 for a description of our theoretical framework). Using the public MeetUp search API we have identified 195 European groups that have included "IoT" or "Internet of Things" in their name. Then, for each of these groups we have collected detailed information about (1) the group and its purpose, (2) the events the group has organized since its creation, (3) the keyword preferences and geographic location of both, the organizer and all the participants who manifested an interest in attending at least one of the meetings. Figure 17 shows the relation between the different data entities stored for analysis.

Please notice that the data schema shows all the information available through the MeetUp API, but all personal information has been anonymized. In particular, all the unique identifiers (*id*) have been randomly reassigned and the *name* field in the *Members* table has been deleted.

Table 6: **Summary of the collected data**. The number of organizers and groups do not match because for some of the listed groups there is no official organizer according to the public API.

Entity	Observations
Groups	195
Events	2,386
Organizers	170
Participants	32,971
RSVPS	71,301
Keywords	8,685

Our data contains information about 195 IoT MeetUp groups that were active in October 2018, and the 2,386 events they have organized or have planned to organize until October 2020. Appendix C contains a detailed list of the collected groups and a summary of their activity over time. As we can observe, the activity of the groups, measured as the number of events per month, is very heterogeneous in our dataset. Larger groups have regular meetings, usually

[4]https://www.MeetUp.com/

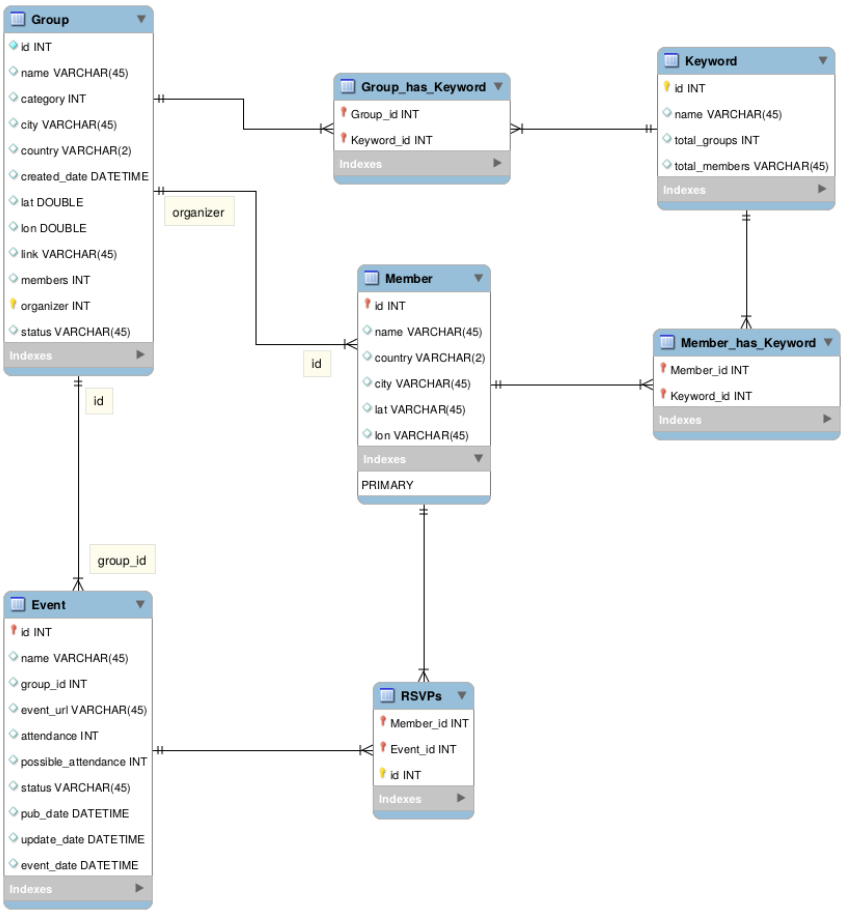


Figure 17: **MeetUp data schema**.

once per month, that are announced in time; while smaller groups meet more sporadically. Table 6 summarizes the number of instances collected for each entity.

The events' participants have been extracted from the positive *RSVPs* notified to the MeetUp service, and hence might not reflect the actual number of participants attending an event because members do not necessarily physically go a meeting after the RSVP and because most of the events accept members who have not previously registered as attendees. The organizers of the events can, afterwards, notify the platform about the total number of real participants but, in practice, organizers do not use this feature of the service. For the purposes of our analysis we will assume that all the data collected is static and

accurate.

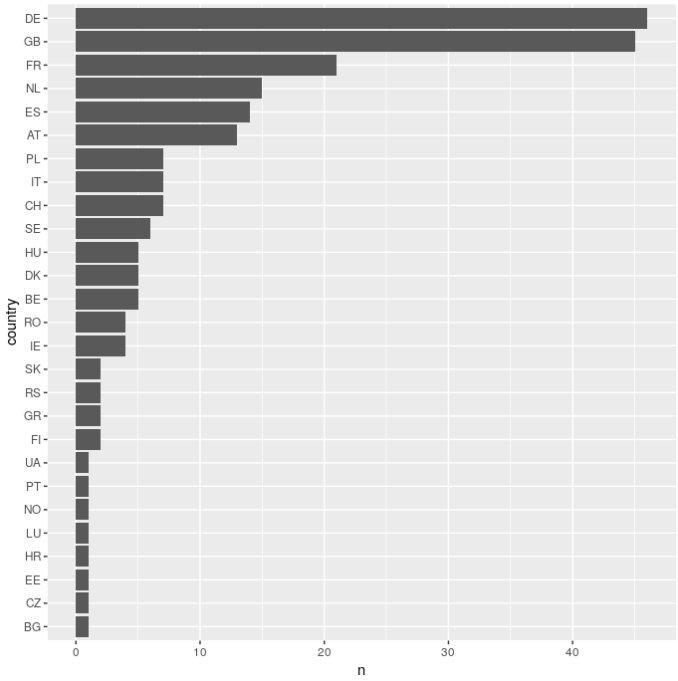
4.1 Basic statistics

As we know from the interviews performed during the field work, IoT activities in Europe have been unevenly distributed across countries, with Germany, UK and France being the most active, followed by Spain, Belgium, Netherlands and others. Figure 18 shows the amount of activity we have observed in our data by country and entity — groups, organizers and participants. It is not surprising that there are no significant differences between the three rankings. MeetUp communities are, unlike other participatory forums, local entities organized and supported off-line by members from the same country, even the same city in most of the cases. In 93% of the cases the nationality of the organizer matches the country where the events of his/her group take place.

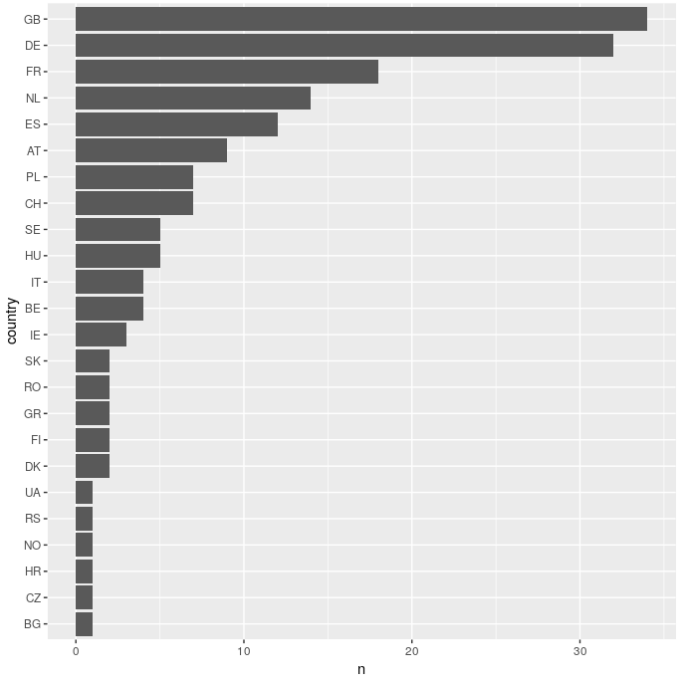
Overall, only 53% of the countries in Europe (27 / 51) have active groups with a name explicitly mentioning "IoT" or "Internet of Things", with a higher concentration of them in the central/western countries.

Figure 20 shows the overall number of IoT events the groups have organized over time, including some of the events they plan to have in the future. We can observe that the communities have a seasonal pattern, with periods of high activity concentrated just before summer and very few events in August concurring with the holiday season in most countries. According to the collected data, the amount of IoT-related events has increased during the past years, reaching a maximum of 78 events on May 2017 (indicated in Figure 20 with a vertical dashed line).

As we can observe in Figure 21, interest in IoT field has evolved differently across Europe. While in UK and Germany the number of events has grown every year since the creation of the social platform, in Spain the number of events has been small until recently. The first events in Netherlands and France happened later, in 2014 and 2015 respectively.

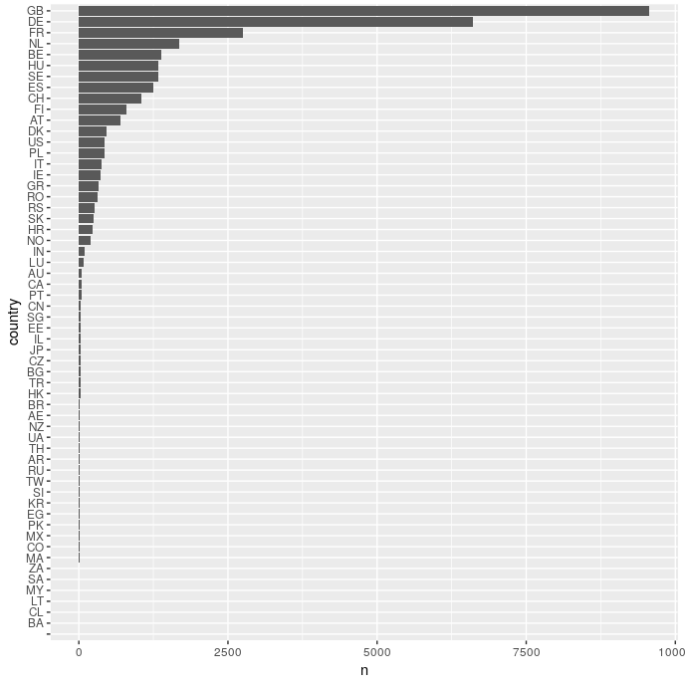


(a) Groups



(b) Organizers

Figure 18: **Data distribution by country**. Part I.



(a) Members

Figure 19: **Data distribution by country**. Part II. Countries with less than 2 members have been omitted.

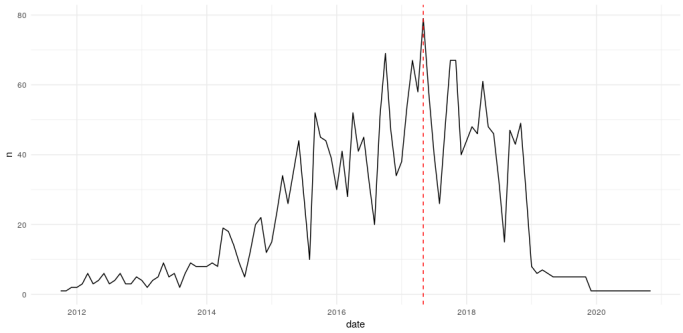


Figure 20: **Overall number of events over time**. Number of events organized by the IoT MeetUp groups over time. The dashed (red) vertical line indicates the maximum peak, corresponding to May 2017.

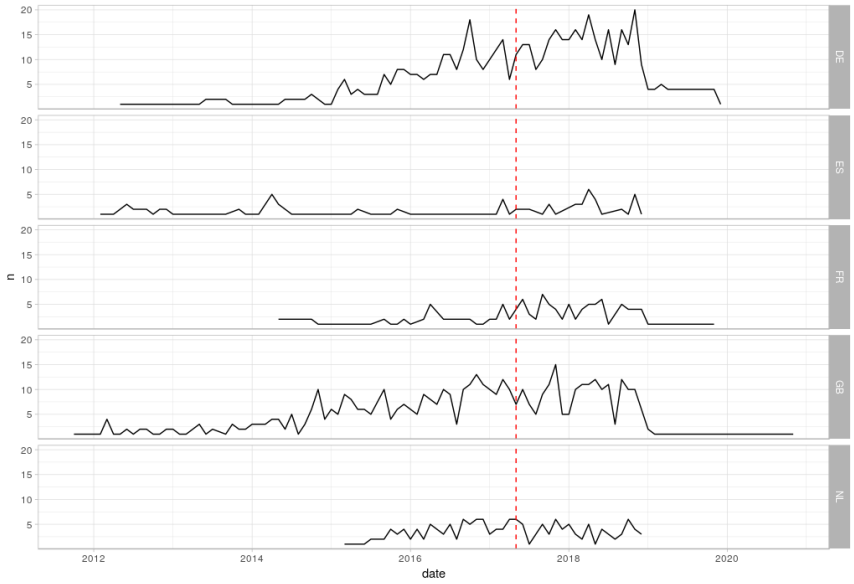
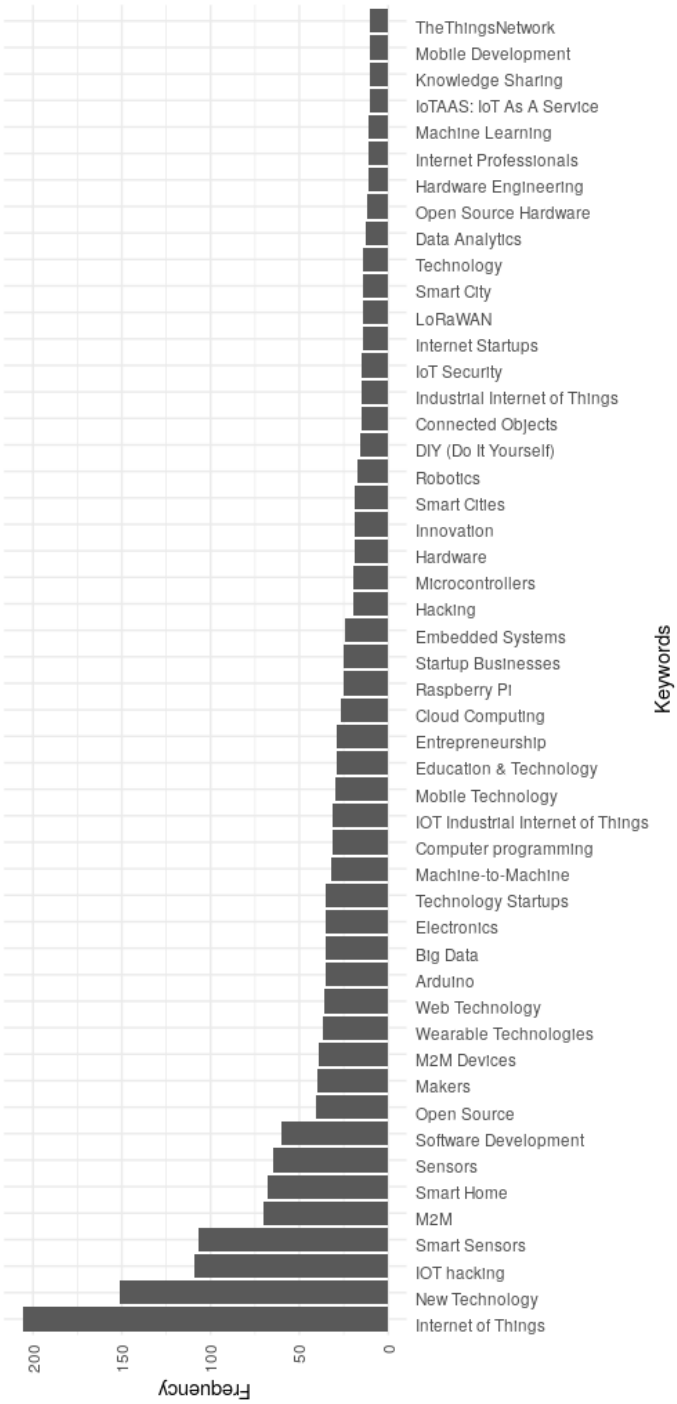


Figure 21: **Overall number of events over time**.

4.2 Topic analysis

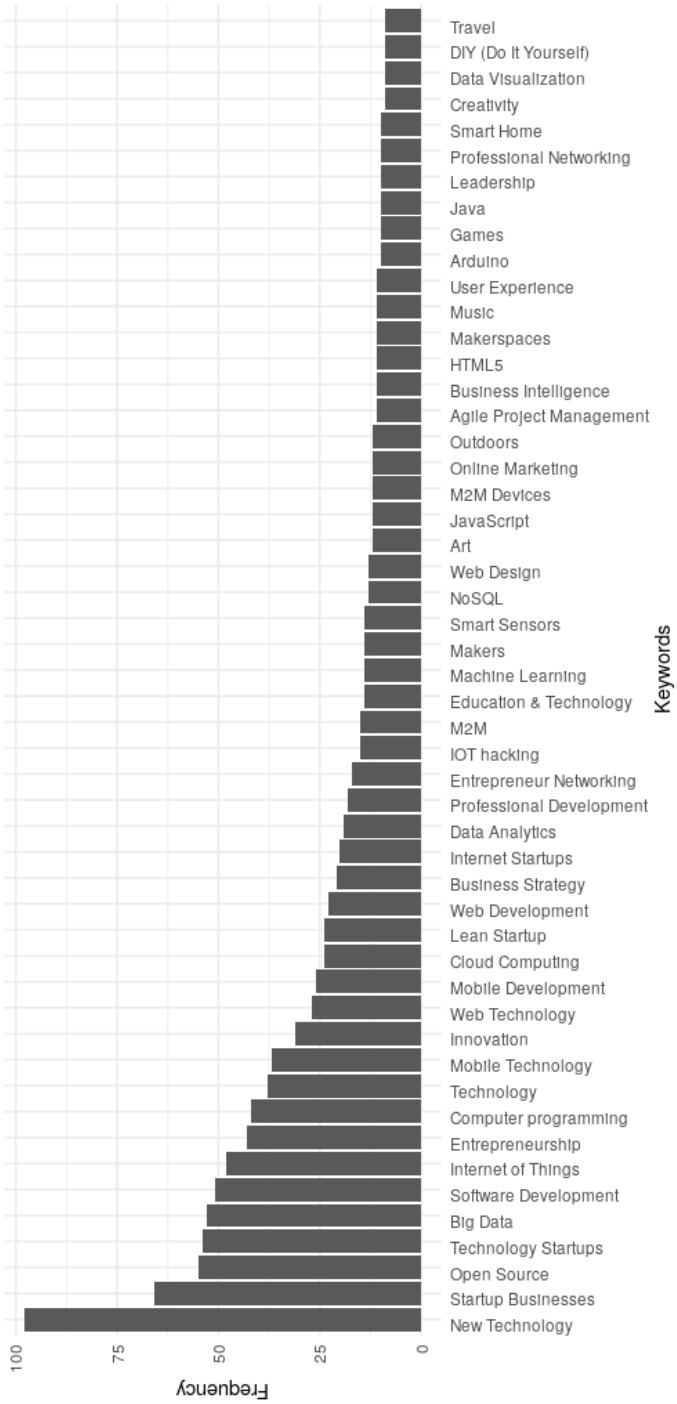
Apart from the distribution of popularity of IoT events in Europe, one of our main objectives in this project is to identify current and past matters of concern defining IoT. To use a uniform terminology across the data sources used in this project, here we call these matters of concern *topics*.

Figure 23 shows the number of times each of the 8,685 keywords we collected was used to define the purpose of a MeetUp group (Figure 23a) or mentioned in the profile of one of the organizers (Figure 22a). We can observe that some of the keywords the organizers used to define the purpose of their groups, were also used to define their own interests, but many of them differ. For example, there are only 4 keywords in common in the top 10 interests on each list: “Internet of Things”, “New Technology”, “Software Development”, and “Open Source”. The relative frequency of the use of these keywords is also different (e.g., Internet of Things is the top keyword used to define a group, while is the 7th to describe an organizer).



(a) Group

Figure 22: **Top-50 keywords**. Top-50 keywords used for organizers to define the interest of the MeetUp group they manage.



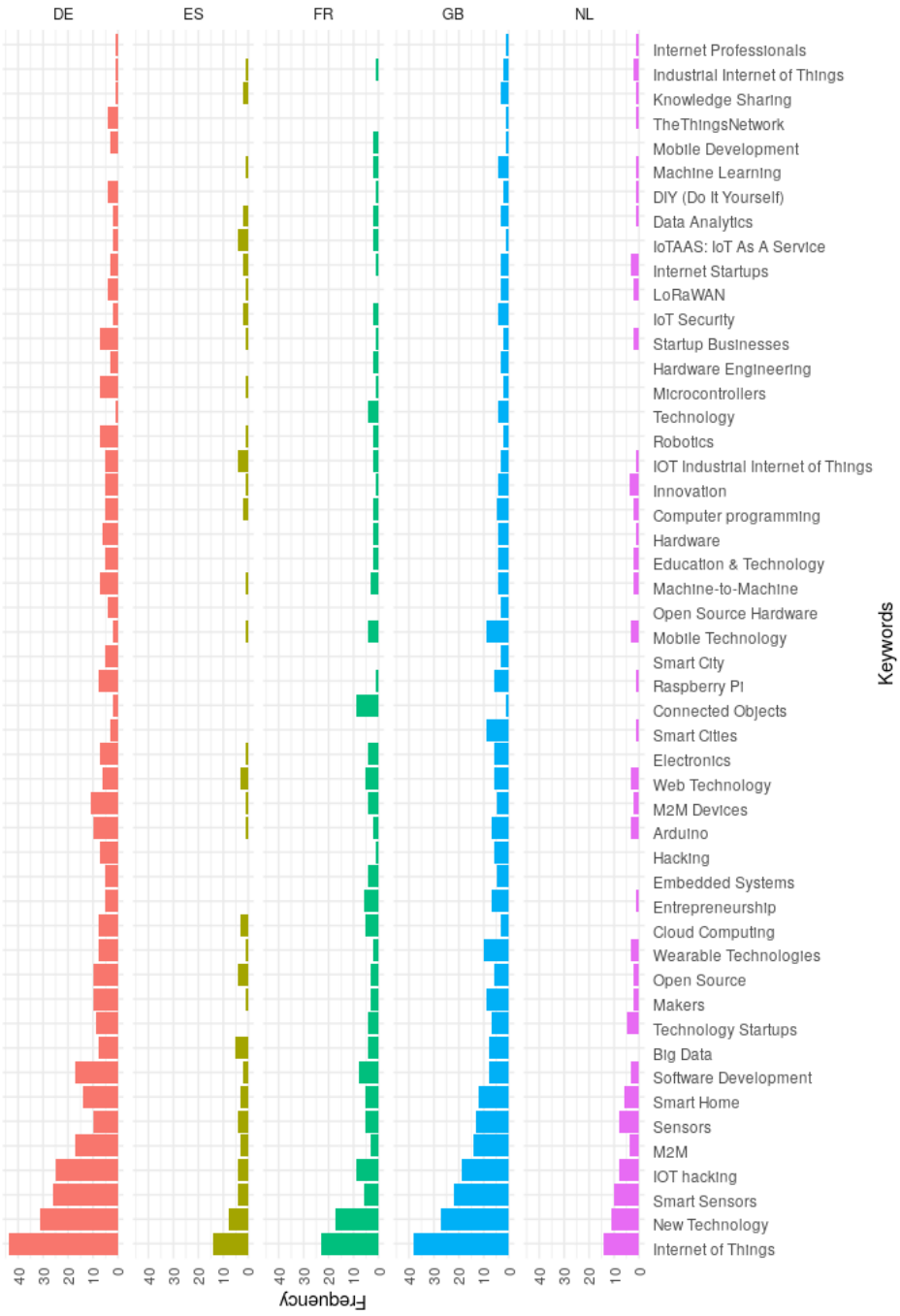
(a) Organizer

Figure 23: **Top-50 keywords**. Top-50 keywords used for organizers to define their own interests.

In particular, we observed that when organizers must define their own interests, they choose to use a large number of keywords directly related with business (e.g., "Startup Business", "Technology Startups" and "Entrepreneurship" appear in the top-10) in combination with keywords defining domains and practices. Instead, when organizers choose the keywords to define the IoT groups they manage, they use less business related-keywords (e.g., "Technology Startups" is the first one, and appears in 17th position), and a balanced combination of things, domains and practices.

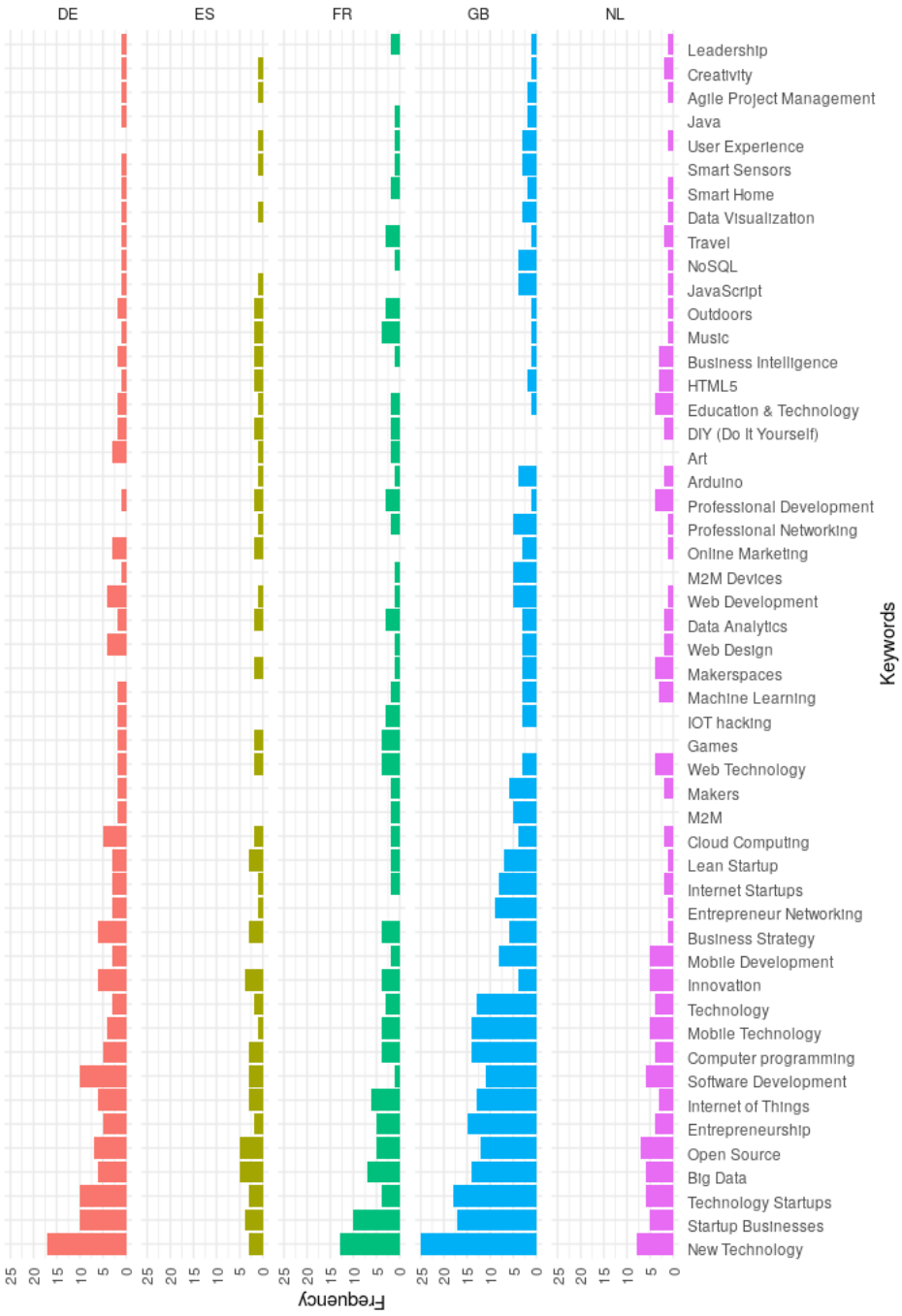
There can be many reasons for this difference of criteria, from a marketing strategy to an unconscious bias. The quantitative data we collected is not sufficient to find a reasonable answer, so we plan to further investigate this phenomenon during the following months.

A detailed observation of the keyword distribution among the 5 most active countries (see Figure 25) reveals that while communities from different geographic locations have a higher agreement on what elements represent the IoT field, the organizers of those groups have a large disagreement. In particular, many of the business keywords that we just identified, are very prominent in the two countries with larger number of events (UK and Germany) but they are relegated to a secondary role in the other countries we have examined.



(a) Group

Figure 24: **Top-50 keywords per country**. Top-50 keywords used for organizers to define the interest of the MeetUp group they manage filtered by the 5 countries with most groups.



(a) Organizer

Figure 25: **Top-50 keywords per country**. Top-50 keywords used for organizers to define their own interests filtered by the 5 countries with most groups.

Differences also arise between the most active countries. For example, compared with the organizers in Germany, their counterpart in UK and France have a larger vocabulary of keywords to define their MeetUp groups, which they use unevenly (e.g., some of the terms are only used by a very small fraction of the organizers).

5 Twitter data analysis

Twitter data is significantly different from the data that can be extracted from MeetUp. On the one hand, on Twitter we can observe finer-grained social interactions. MeetUp only allows us to know if two users have expressed an interest in the same event, which does not tell us whether they know each other or have interacted in any way. On Twitter, we can see if a user has mentioned another specific user, and we have also access to the text they exchanged. On the other hand, information on Twitter is more unstructured: we can think of hashtags as something playing a similar role as MeetUp events when we perform our analyses (that is, filtering users interested in the event or topic associated to the hashtag), but hashtags are not created through some centralized decision-making process as in the case of MeetUp organizers. This can lead to different hashtags referring to the same topic, single hashtags used to refer to different topics in different tweets, as well as many tweets not explicitly mentioning any hashtag. In addition, on Twitter there is no clear counterpart of the tagging happening on MeetUp, where both users and groups are self-annotated with precise keywords. Therefore, Twitter data can be more challenging to identify matters of concern, but can also highlight more informal conversations.

Ideally, we would like to use Twitter data to perform various types of analysis: what is discussed in IoT-related tweets, where are the different topics appearing, when are they discussed (for example, when they emerge for the first time and when they reach a peak of popularity), who is leading the discussion on specific topics, and finally a joint analysis putting all these aspects together to map online conversations. As we discuss in the next sections, some of these analyses are indeed possible given some assumptions and limitations, some are not, and to extract more knowledge from the data we required frequent interactions with the qualitative units – a methodology known as triangulation, and further discussed in the following.

5.1 The #IoT dataset

Using our Twitter data collection tool we have been retrieving tweets containing the #IoT hashtag since March 2017. The initial objective of this retrieval, that has happened in parallel with the collection of event-specific tweets (for example, in occasion of IoT conferences), was to perform the analyses mentioned above.

However, the analysis of this large dataset containing millions of tweets has highlighted some limitations.

The first problems we had to face concerned the spatial analysis of the tweets. First, we have no guarantee that the geographical distribution of the collected tweets is a good indicator of the number of tweets with the #IoT hashtag produced in different locations. This is due to uncertainty about the sampling algorithm used by Twitter and about the adoption of geo-localized tweets in different regions. Another problem concerns the decision of how to normalize statistics from the data: in one area there can be only a few very active users, in other areas a lot of less active ones. Whether one should normalize with

respect to the number of users or the number of tweets or the local population or the local population active on Twitter or some combination of these values is unclear. Finally, the Twitter API does not allow to filter both by hashtag and by location. As the large majority of collected tweets is not geo-localized, it becomes almost impossible to provide a zoomed-in analysis of Europe without including tweets from other locations.

A second type of problems regards the noise in the data, that is, the presence of irrelevant tweets. An example is the inclusion of tweets automatically produced by traffic cameras in Brazil. While these are clearly IoT devices, justifying their usage of the #IoT hashtag, on top of not being in Europe these tweets are not relevant in the identification of online discussions between important actors and inside IoT communities of practice. The removal of these tweets required intensive manual work, leading to a list of other hashtags identifying these cases and allowing us to automatically filter out the corresponding data. Examples of these filtering hashtags are #florianopolis, #joinville and #blumenau.

The previous issue also highlights an additional question: which users should be considered in the analysis? If the answer is clear about traffic cameras, it is less clear whether other types of bots[5] should be included or not. At this time the application of bot detection methods has not produced results that are accurate enough to be included in this report.

Another challenge to be able to perform the aforementioned analyses is the identification of topics. Our experiments with automated topic detection methods [14] confirmed that the special language and the short texts in Twitter do not allow an accurate topic extraction. Grouping tweets into larger texts did not help either. This, in addition to other challenges in the state of the art on automated topic detection (such as the selection of the number of topics to be retrieved and the presence of non-deterministic results) forced us to rely once more on manual pre-processing, where the qualitative team checked the list of most common hashtags and defined a set of rules to automate their normalization. For example, transforming #industrialiot into #iiot, and #virtualreality into #vr, so that the same hashtag would be consistently used to refer to the same topic. This led to the identification of the main discussion topics associated to the #IoT hashtag in the last two years, indicated in Table 7.

After this data pre-processing and considering these assumptions, we could finally try to apply our analysis methods for temporal text networks to the resulting data.

5.2 Conversational analysis

In this section, we apply some of the models and approaches introduced in Sections 2 and 3 to a subset of the the #IoT Twitter data, to test their effectiveness. In particular, we focus on using the discretization approach introduced in Section 3.2.2 to analyze the formation and evolution of communities of actors and messages.

[5] AI-based computer programs simulating users

#iiot	#fintech	#finance
#blockchain	#cryptocurrency	#gdpr
#security	#itsecurity	#cybersecurity
#datasecurity	#iotsecurity	#infosec
#vr	#ar	#analytics
#predictiveanalytics	#dataanalytics	#ai
#machinelearning	#deeplearning	#artificial
#hardware	#datamining	#arduino
#3dprinting	#raspberrypi	#smartdevices
#sensors	#wearables	#devices
#smartbuildings	#smartcity	#smart
#smartdevices	#smartcontract	#smartgrid
#smarthomes	#smartmobility	#smartphone
#smarttechnology		

Table 7: Main IoT topics in the large #IoT dataset identified through manual analysis

Dataset The dataset used for this experiment consists of the 247,399 tweets collected in June, 2017. The dataset contains mentions (tweets including *@username*), retweets (tweets starting with *RT @username*), other tweets that are neither mentions nor retweets, and the 51,369 users involved in the aforementioned communications. In the following experiments we focus on the network obtained starting from the tweets containing mentions (about 5% of the initial tweets), built by coding each tweet as in Figure 8.

Table 8: **Temporal text networks** used in the case study and its basic properties: number of actors ($|A|$), number of messages ($|M|$), number of edges ($|E|$) and number of layers ($|L|$).

Networks	Type	$\|A\|$	$\|M\|$	$\|E\|$	$\|L\|$
Original	bipartite	15,717	13,210	35,015	2
discretized	k-partite	15,717	17,273	44,943	182
Projected	multilayer	15,717	-	23,766	182

The resulting temporal text network contains about one third of the users in the initial dataset (15,717) and the 13,210 messages exchanged between them (See Table 8). We call this the *original network*, and use it as a the starting point for both the following experiments.

Discrete analysis Social interactions within a group of participants can form a community if they occur more frequently within the group than with other members of the network. In temporal text networks, those interactions are the result of the exchange of messages between actors. In this example we show how our model can be used to find communities of actors discussing about the same topics during the same weeks. Following the method described in Section 3.2.2 we first transform our network to a multilayer network preserving information

about interactions between users, topics and time, so that we can then apply an existing clustering algorithm.

The *discretized k-partite network* is built following the procedure explained in Section 3.2.2. In this particular example, we first split the original layer of messages using their hashtags as an indication of the topic, then we further discretize based on the week when messages are posted. The second discretization uses the posting time to create hashtag-week-specific layers.

Finally, we build the *multilayer network* by projecting each one of the layers containing messages into the actors' layer. Two actors in this network are connected in a given layer $L = (h, w)$ if at least one of them has sent a message to the other using the hashtag h during the week w. If multiple messages have been exchanged between two actors in the same layer, only a single edge is generated during the projection. At this step all edges are undirected and unweighted to fit the community detection algorithm we used. Table 8 describes the main properties of the original temporal text network, the projected k-partite and the final multilayer network used during the analysis.

Using the multilayer network and the clique percolation algorithm, we proceed to detect communities of actors across the whole network.

Figure 5 shows the communities with more than 3 actors formed in the multilayer network. Communities contain users and topics, and both users and topics can overlap across communities. The number of users is indicated by the size of the community, while the layers representing the topics of interest of the actors are annotated next to each community. The smallest community in the diagram has 4 actors in the same layer, while the largest community contains 27 different actors and 3 layers. The edges between communities in different weeks indicate that at least one third of the users in the second community were also present in its predecessor. The thicker the line, the more users are shared between them.

We can observe that some of the hashtags, in particular *artificial intelligence* (#ai), *augmented reality* (#ar) and *virtual reality* (#ai), are very popular in the IoT discussion space, with several groups of interest of different sizes forming around one or more of them. However, while the three topics are present across the whole month, the communities they form are very volatile. Only one of the smallest community with just 4 actors, for example, is preserved in time without changing its members or the topics they discuss. The largest communities formed during the first week, instead, disappear in week 2. Later on, some of the same users form new communities but with less members and a higher variance of topics. Less frequent hashtags such as #machinelearning, #security, #sensors, #smartcity and #blockchain also form groups of interest, usually smaller and with no or a few connections with the groups of users discussing the most common topics.

Overall these results highlighted some additional challenges in the study of online communities through large Twitter datasets. The identified conversational communities showed a very fragmented IoT space. None of the found communities was big enough to become the main arena to develop a long-standing conversation on a specific topic.

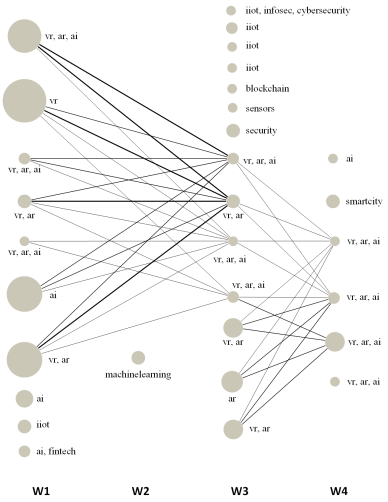


Figure 26: **Evolution of communities in the IoT space**. The size of the communities is indicated by the size of the nodes — representing the number of actors — and the annotated hashtags. The thickness of the edges between two communities indicates the number of common actors between them.

Our explanation of these results is the sparsity of Twitter conversations. While it is true, as mentioned at the beginning of this section, that on Twitter we have the possibility of observing dyadic interactions (a user mentioning a specific other user), the platform itself does not promote the stability of this type of interactions in time, and there are other ways of participating to the same conversation without explicitly marking this in a way that is recognized in the API and thus directly searchable. To address this additional challenge, we started developing new methods to identify user interactions beyond those returned by the Twitter API [51] and, as initially planned, we relied on yet another triangulation with the qualitative unit, leading to the establishment of the so-called *consolidate network database*.

5.3 Consolidated data

When the complexity of analyzing the whole Twitter space connected with IoT together with the discovery of the poor quality of the data that, on average, it contains became evident, the research team adopted an auxiliary strategy focused on a smaller set of qualitatively selected users. Thanks to the ongoing qualitative field-exploration that the teams from ITU and LSE were conducting, a large number of relevant European actors in the IoT field were already known as well as their use of social media platforms for professional communication. Therefore the research team defined a two-fold goal: a) on the one side it was

Network	Nodes	Edges	Density	Avg. Degree	Clust.coeff
Full network	319445	424781	0.000004	2.66	0.0001
Reduced network	51580	156916	0.00006	6.08	0.0014
Consolidated network	103	1360	0.130	26.4	0.385

Table 9: The basic statistics of the three Twitter networks generated from the consolidated dataset.

necessary to test the completeness of our qualitative selection of relevant IoT actors and b) on the other side it appeared important to understand the online communicative structure of this selected set of IoT users.

In order to achieve this, first ITU and LSE provided, for each selected IoT expert a set of background information such as: social media handlers, geographical location, personal background, type of activity within the IoT space, and if they have previously showed interest in ethics in IoT. From now on we will refer to this enriched set of data as "consolidated dataset". Starting from the consolidated dataset UU used the public Twitter API to retrieve the full list of followers and friends for each user. Using this in formation we built a directed network of the Twitter space *surrounding* our initial set of qualitatively selected IoT experts. Starting from this network we defined three derived networks that gave us the opportunity to better understand the digital conversation about IoT and its related community structure:

1. the *full network* containing all the followers and all the followees of the 103 initial Twitter users from the consolidated dataset.

2. the *reduced network* containing all the initial users from the consolidated dataset and the followers or followees connected with at least two of the initial users.

3. the *consolidated network* containing only the users from the consolidated dataset and the connections among them.

In the following paragraphs we will first briefly present these networks and their topology and then we will focus our attention on what can be learned using the consolidated network.

Table 9 shows how the networks are remarkably different both in terms of size as well as in terms of structure and basic topological characteristics. A first consideration that can be done is that each member of the consolidated dataset has a Twitter network that only partially overlaps with the ones of the other members. The *reduced network*, that includes only the users who are connected with at least two members of the consolidated dataset is one sixth of the full network. This suggests that each member of the consolidated dataset has her own Twitter audience that might or might not be interested in the conversation about IoT and ethics in IoT. Looking at the general evolution of the three networks it appears clear that beside the obvious reduction of the size of the

network the signs of community structure grow stronger: the *reduced network* and the *consolidated network* are denser and more clustered.

The conceptual dependency between the three networks can be used to test the completeness of the consolidated dataset as a selection of relevant expert in the IoT space. Knowing, from the insights obtained with the qualitative interviews, that Twitter is a valuable tool to be updated on the ongoing events and conversation within the professional space of IoT, it is reasonable to assume that any relevant name within that domain would be followed by a large number of the members of the consolidated dataset. Checking the data we can see that the top 100 most followed users of both the *full network* and *reduced network* belong to the initial 103 members of the consolidated dataset, showing how that qualitative selection is actually relevant for the large IoT community of Twitter.

5.3.1 Single-layer analysis

Once we have tested the completeness and the relevance of our selection of expert users in the consolidated dataset, we can study how this selected subset of the IoT community on Twitter is structured. We do this in two steps: first we analyze the following/follower relations between the members of the consolidated dataset to find out the social forces behind the observed (online) network structure. Then we adopt a multilayer approach as defined in Section 2 and using some of the methods described in Section 3 we explore the topical similarity between the members of the consolidated network.

Given the enriched information available for the original users forming the consolidated dataset, it has been possible to explore the social dynamics behind their network structure. This has been done studying the nominal assortativity [92] of the network for three specific attributes: geographic location, background and involvement in the online conversation about IoT. All these attributes were manually inserted and verified by the research team.

Nominal assortativity is a well known measure that quantifies the trend of a specific node to connect with other nodes with the same categorical attribute. In this case we tested nominal assortativity for geographic location (at a country level), users' professional background and involvement on the online conversation about ethics and IoT. The three hypotheses underlying these three tests were:

- Geographical proximity is an indicator of the presence of mutual interests between online IoT experts: users from the same geographical context will be more likely to be connected online than users from different geographical contexts. This hypothesis would assume a significant positive value of nominal assortativity.

- Complementary background is an indicator of the presence of mutual interests between online IoT experts: users with complementary background (e.g. one in Design and one in Software development) will be more likely to be connected due to the added value of their complementarity for per-

spective business opportunities. This hypothesis would assume a negative value of nominal assortativity.

- Ethical interest – the participation in the online discussion about ethics and IoT – is an indicator of the presence of mutual interests between online IoT experts: Users who participate in the ongoing online discussion about IoT and ethics will be more likely to follow other users equally vocal on the issue, This hypothesis would assume a positive value of nominal assortativity.

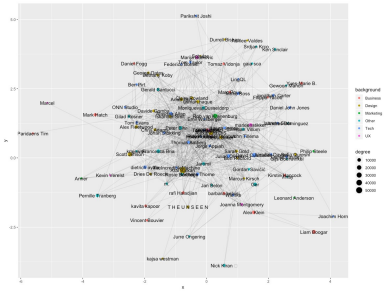


Figure 27: Consolidated Network, users' background

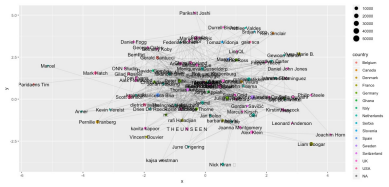


Figure 28: Consolidated Network, users' country

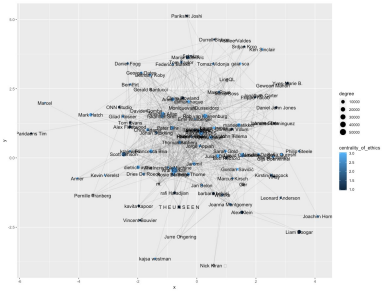


Figure 29: Consolidated Network, users' participation in ethical discussion

The analysis of nominal assortativity for these attributes shows very limited assortativity. Geographical assortativity is the strongest one with a value of 0.16, while both background complementarity and participation in ethical discussion show values close to zero (0.05 and 0.02). These data suggest that between the three hypotheses of possible social drivers behind online connectivity only geographical proximity is weakly supported. Professional background does not show an assortative (or dissortative) behaviour, suggesting that the reason to be connected on Twitter lies beyond the complementarity or simi-

larity of the professional profiles. Similarly, the level of activity on the online ethical discussion about IoT does not play any role in the connectn process, suggesting that ethics, even if valued on an individual level, does not act as a discriminant for online connections. The only attribute that is, weakly, positive is the geographical proximity suggesting that even if there is a European IoT scene, geography still matters with the local context acting as a force driving online connectivity.

5.3.2 Multilayer analysis

In addition to the topological structure of the following/follower relations discussed in the previous paragraph we also studied the actual interaction between the members of the consolidated dataset and their online audiences. The main goal of this part of research was to investigate the presence of topical communities within the European IoT space. Since, as we showed in the previous analysis, geographical proximity still plays a significant role into existing Twitter relations we based this analysis on the actual interactions rather than on the more stable following dynamic. Moreover we adopted the approach based on multilayer networks (described in Section 2) combined with a qualitative analysis of the most frequently used hashtags aimed at identifying how subgroups of the consolidated graph are actively engaged in thematic online conversation.

In order to do so, the hashtags included in the data have been first qualitative filtered and grouped in themes, then the interactions taking place within those *themes* have been mapped on a multilayer network structure where each layer represents a specific theme. This approach allows us to represent within a single multilayer network structure (the IoT Twitter space defined by the experts included in the consolidated dataset and the Twitter users they were interacting with) various types of interactions about various topics. It should be noticed that while the starting nodes for this network are always the actors in the consolidated datasets, the network also contains additional users that interacted with the members of the consolidated dataset.

Table 10 provides an overview of the topical multilayer network. As it should be visible from Table 10 the topical layers are different in size and topological characteristics. With few exceptions, the number of components is relatively small suggesting a common conversation involving multiple users rather than isolated discussions.

While Table 10 gives us an overview of the layers forming the multilayer structure it tells us nothing of how the users actually participated on the various layers. Figure 30 shows the coverage [18] for the actors on the various layers. It appears clear that the actual overlap between the users participating in the various topics/layer is rather limited with the exception of the #IoT that shows consistent overlap with almost all the other hashtags. This peculiar structure suggests the existence, within the IoT umbrella, of multiple sub-domains with relatively little overlap.

The existence of multiples thematic subgroups within the IoT Twitter space is confirmed by the application of community detection methods to the data.

Layer	Nodes	Edges	Density	Comp.	Avg. Degree	Clust.coeff
All	1396	2562	0.0001	11	3.67	0.024
IoT	819	1634	0.0024	7	3.99	0.032
Cybersecurity	213	422	0.009	6	3.96	0.072
Machine learning	75	92	0.016	5	2.453	0.015
Wearable tech	98	114	0.011	4	2.32	0.014
AR/VR	90	92	0.011	5	2.04	0.002
Women in Tech	90	119	0.014	2	2.644	0.028
eHealth	108	141	0.012	2	2.61	0.012
Events	68	90	0.019	1	2.64	0.027
Startups	81	102	0.015	6	2.51	0.006
Technology	118	116	0.008	12	1.966	0.005
AI	308	428	0.004	7	2.77	0.027
Privacy	88	90	0.011	9	2.04	0.008

Table 10: The basic statistics of the topical layers of the multilayer Twitter network.

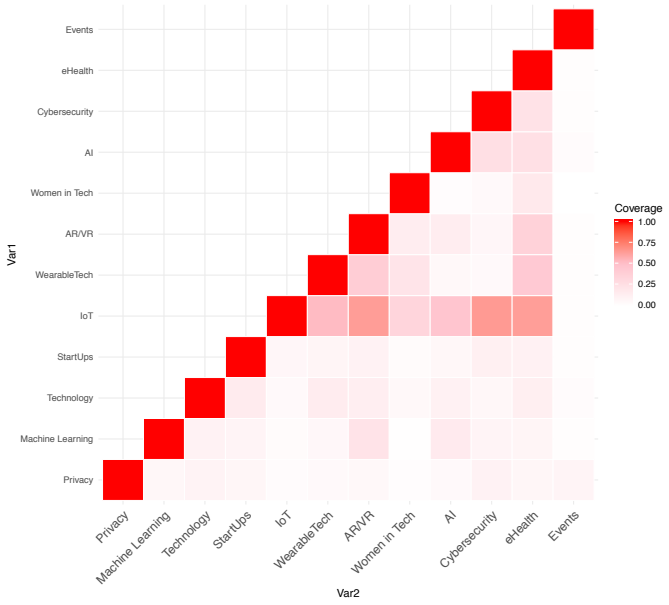


Figure 30: **Interlayer coverage**

Using the multiplex clique percolation method presented in Section 3.1.2 we can observe how in the data the number of communities detected when we require at least two common layers in a clique (m = 2) is relatively small counting for only 8 communities where the largest one counts 16 members.

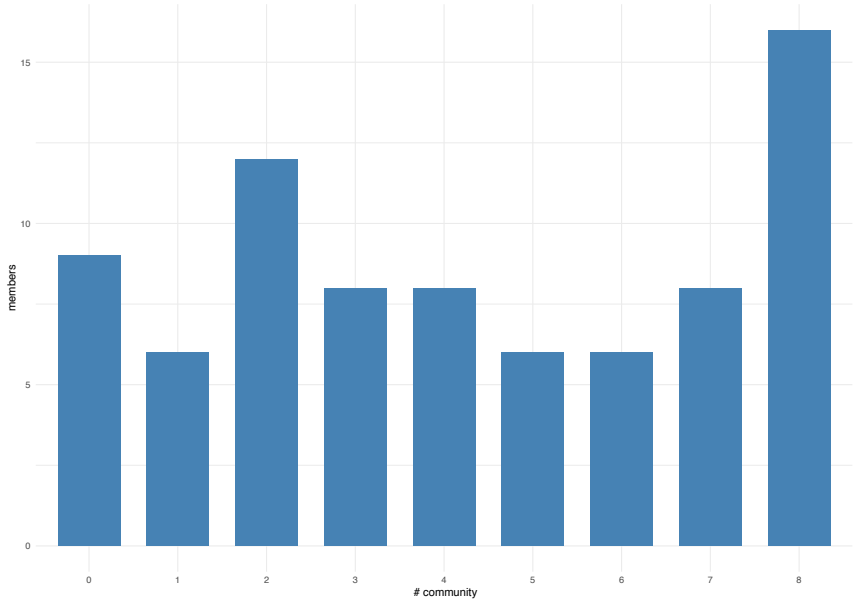


Figure 31: **Size of the communities detected with Clique Percolation (m=2)**

The overall picture that emerges from the study of the Twitter IoT consolidated network is complex. On the one side there is a topological common space properly defined and identified by our manual selection of the key actors. Nevertheless, beside this space of following/followers relations there is the actual space of interaction that, in itself, is subdivided into the many domains that characterize the IoT space.

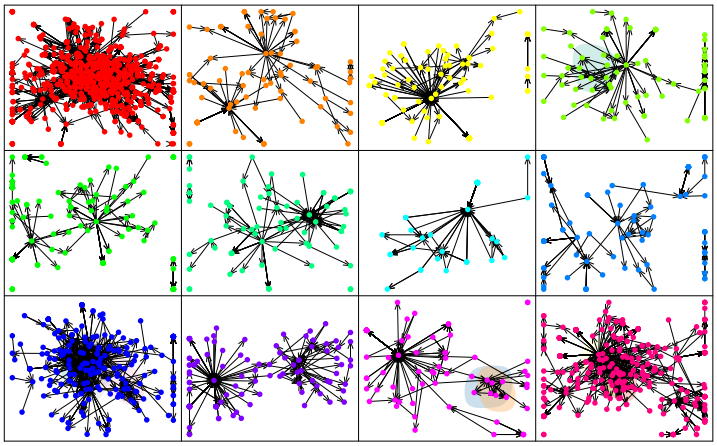


Figure 32: **Communities detected with Clique Percolation (m=2)**

A Related literature

In this section we present an overview of related work on models for temporal text networks, community detection in multiplex networks and network drawing, which are the main areas where our results are positioned.

A.1 Modelling networks, time and text

Our concept of temporal text network is a combination of text, network topology and time. In the literature there is a large number of models supporting one or more of these aspects, and the objective of this section is to characterize existing models from a common viewpoint.

Notice that there are entire well-established disciplines developed to address text, network topology and time in isolation, and we do not review these here as they are widely covered by text books [89, 3], described in numerous research papers (see for example [14] and subsequent extensions), and included in several software packages and systems. Instead we describe recent research efforts combining at least two of these aspects.

Table 11 presents a summary of the models selected for this review, also including our proposed model (core and extended temporal text network), organized according to four main criteria: (1) the type of graph used to represent the topological portion of the data, (2) the type(s) of nodes allowed in the graph,

(3) the way in which text is represented in the model and (4) the way in which time is represented in the model. As our aim is to comprehensively list models, not papers, and the number of works using some of the models is very large, we have sometimes arbitrarily and unavoidably chosen a key set of references based on our knowledge and personal selection. Therefore, please notice that in the table we only indicate selected references; additional references are included in the text. Figure 33 complements Table 11 providing a visual intuition of the reviewed models and of the new models introduced in this report.

Table 11: **Comparison of models** representing two or more of the main aspects of temporal text networks. The graph type is indicated as D: directed (undirected if D is not specified), O: ordered, G: Graph, MG: Multi graph, BG: Bipartite graph, ML: Multilayer graph. Node types indicate the domain of the nodes, and we distinguish between A (nodes used to represent actors) and X (nodes used to represent text-related objects). Given the variety of existing models, X is broadly used to represent full text documents, parts of it (phrases, words), other representations of documents such as bags of words (BoW), and also objects obtained by analyzing the text, such as concepts/topics. In this table we only indicate selected references.

Name	**Graph type**	**Node types**	**Text**	**Time**	**Refs.**
Contact sequence	DMG	A	—	mostly edges	[53, 45]
Time-slice	OML	A	—	layers	[88]
Longitudinal	OML	A	—	layers	[116, 117]
Memory	DG	A^n	—	edges (implicit)	[113, 107, 75, 98]
Memory (multilayer)	DML	$A^1 \cup \ldots \cup A^n$	—	edges (implicit)	[112]
Temporal text	—	X	document	vertices	[19]
Longitudinal text	—	X	document	layers	[93, 35]
Language networks	G/DG	X	word	—	[118]
Document networks	G/DG	X	document	—	[24, 86]
Document-phrase graph	BG	$X \cup X$	doc., phrase	—	[101]
HIN	BG	$A \cup X$	BoW, doc.	—	[25, 128, 71]
Socio-semantic network	BG	$A \cup X$	concept	—	[108, 52]
Temp. Socio-semantic network	BG	$A \cup X$	concept	edges	[109]
Citation network	DG	X	document	vertices	[42, 6]
Author-citation network	DG	$2^A \times X$	document	vertices	[129]
Spreading process	DG	$A \times X$	—	edge (delay)	[80]
Polyadic conversations	DG	$A \times 2^A \times X$	document	vertices	[82]
Core temporal text network	DBG	$A \cup X$	document	edges	
Ext. temporal text network	ML	$A \cup X$	document	edges	

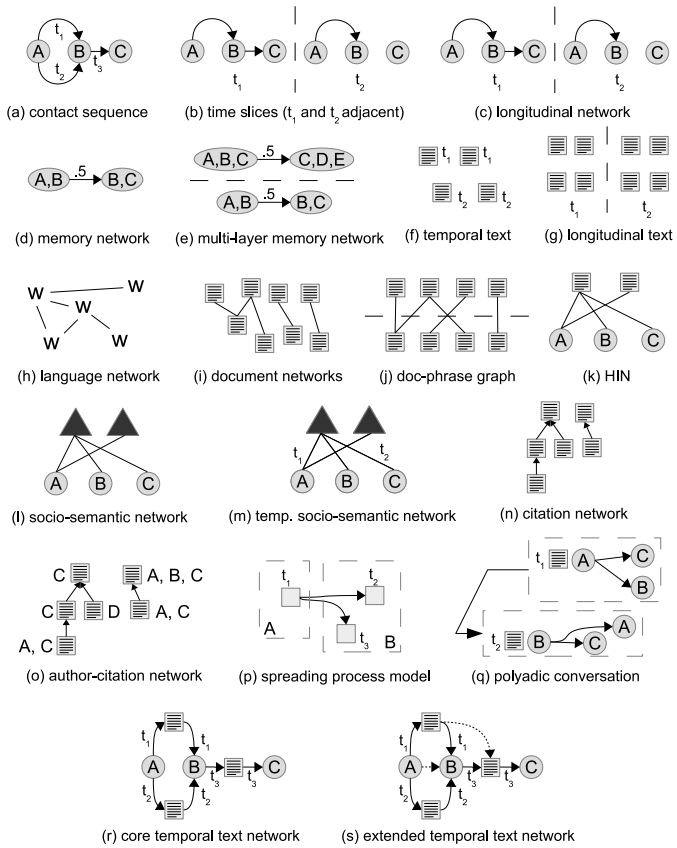


Figure 33: **A visual gallery of models for time text and networks.**

A.1.1 Time & Topology

The most basic family of models including both time and topology is the *contact sequence* [53, 45]. Typical contact sequence models include a set of actors interacting in the social graph, a set of directed edges, representing the direction of such interactions (e.g., who sends a message to whom), and edge time annotations represented as timestamps, absolute points in time or time intervals. This model has been typically used to study information spreading [76, 26], and existing concepts such as motifs and triadic closure have been re-defined to study the evolving structure of these networks [97, 126, 63].

Differently from contact sequences, where interactions are time-annotated one by one, other types of models use sequences of time-annotated graphs, where each graph is sometimes also called layer. In *time-sliced* models time is

expressed as an interval and an edge indicates that an interaction has happened at some point during the time interval associated to the graph [88]. These models are typically obtained starting from a contact sequence and aggregating edges by time. In *longitudinal networks* relationships about the same or similar actors are detected at different points in time [116, 117]. From a data modeling point of view, time slicing and longitudinal networks are very similar, and in practice the main difference lies in the nature of the time annotation associated to each slice, where in time slicing adjacent slices are typically associated with adjacent time intervals while in longitudinal network studies adjacent layers represent network snapshots obtained at specific points in time. Different types of time annotations are described for example in [7].

Memory models provide a different view over a temporal network, where ordered tuples of two or more actors are represented as single nodes [113, 107, 75, 98]. For example, if an actor A is receiving one message from B and one from C, and is later sending a message to B and one to C, a contact sequence loses information on whether A is replying to B and C (B $\rightarrow$ A $\rightarrow$ B, C $\rightarrow$ A $\rightarrow$ C) or forwarding the messages (B $\rightarrow$ A $\rightarrow$ C, C $\rightarrow$ A $\rightarrow$ B). A first-order memory model will contain nodes for each pair of users and have an edge between two nodes if the corresponding pairs appear on consecutive paths. In our example, if A is replying we will have two edges in the memory model: ($\overrightarrow{BA}$, $\overrightarrow{AB}$) and ($\overrightarrow{CA}$, $\overrightarrow{AC}$), while if A is forwarding the messages we will have the edges ($\overrightarrow{BA}$, $\overrightarrow{AC}$) and ($\overrightarrow{CA}$, $\overrightarrow{AB}$). Memory models typically have a fixed depth (also called order), but different path lengths can also co-exist inside the same model [112].

Time often plays an important role when networks are concerned, because networks often represent dynamical systems. However, in Table **??** we have only listed distinct data models explicitly providing time annotations. As an example, *growing network models* [89] such as preferential attachment [5] aim at explaining the observed topology of empirical networks based on how they evolve in time from an initial small network. However, even if nodes and edges join the network one after the other, there is no explicit representation of time in the final model. Similarly, we have not listed papers about methods not explicitly introducing new data models, such as [79].

A.1.2 Time & Text

Time is often present inside text, and commercial systems handling large human information networks from Google mail to common text messaging applications on smart phones can automatically identify the messages and annotate the text with temporal information.

In research, text and time are studied together in the field known as temporal information retrieval [2, 62]. This is an active area, also represented at the TREC conference where state-of-the-art information retrieval methods compete on various practical tasks. Time can be present in the text, as in the examples above or as metadata, expressed as absolute or relative time and it can also be specified in queries used to express information requirements [19].

Another set of studies has focused on how text evolves in time, and in particular sentiment, with case studies ranging from tweets [93] to songs, blogs and presidential speeches [35]. Text and time are also studied across data sources, for example to correlate texts from online news to trends emerging in time series such as financial data [77]. However, no specific data model is used for this type of tasks, but only time-annotated documents (understood in a broad sense, including words, etc.) and time series.

A.1.3 Text & Topology

Text and networks have been studied together in various areas, either without considering time or using networks to represent relationships between texts.

Models where nodes represent parts of a document have been used in structured information retrieval, which was a particularly active research area when hypertexts and markup languages became popular [70]. Text is often contained inside some structure (e.g., a title, sections, sub-sections, etc.) and queries can be tuned to return specific parts of a document instead of a full one. As an example, if the searched keyword is contained inside Subsections 3.1 and 3.3 of a document, a query may return either the two subsections, or the whole Section 3, depending on the method.

More relevant for this report are document networks, that are graphs whose nodes represent text documents [24, 86]. These network models can be classified in different groups depending on whether they include time or not; later in this section we refer to citation networks as a type of directed document network where time is also typically present. Text mining, and in particular clustering, can be applied to document networks to identify groups of documents that are similar not only because of their text but also because of their connections, as summarized in a recent article about clustering attributed graphs [15].

Several works have focused on networks extracted from text, and we can broadly classify them into models representing the text itself, aimed at characterizing language, and models representing actors and concepts mentioned in the text.

Networks where nodes represent words have been used to model both text documents and languages [118]. For example, a document can be modeled as a network where words are connected by an edge when they are contiguous, or appear in the same sentence, paragraph, etc. Similarly whole languages can be modeled focusing on the relationships between words, as in WordNet or BabelNet.

With regard to the second class of models for networks extracted from text, Named Entity Recognition methods are typically used to identify the nodes and co-occurrence (or other language analysis approaches) to create edges among them [34, 25]. In this case, the output network connects different portions of a text document, or concepts extracted from the text. A model that has been used to represent the relationships extracted from texts is known as heterogeneous information network (HIN) [115, 100], allowing multiple types of nodes. In [128], for example, the authors build a graph of relations between text documents,

phrases and salient phrases to study the similarities between different documents based on their structure and semantics. In [71] the authors use a text-enriched heterogeneous citation network to classify related authors and papers. Time can also be theoretically incorporated in the HIN model as a new type of edge attribute, but it remains an open research question how the model and/or the analysis would benefit from that [101].

One of the concerns recently raised against using methods from social network analysis to analyze social media is their intrinsic actor-centered approach (e.g., people, companies, stakeholders), focusing on social interactions without properly characterizing other aspects of the communication [108]. A similar argument can be used against the use of just Natural Language Processing or semantic networks [119].

Following this reasoning, a recent stream of research focused on combining structural and semantic data simultaneously, which led to the formalization of the socio-semantic network model [109, 108, 52]. Originally, socio-semantic networks were just bipartite graphs interconnecting *agents* (also known as actors in Social Network Analysis) with semantic objects called *concepts*, corresponding for example to terms, n-grams, or lexical tags.

During the last decade the socio-semantic network model has been extended to extract more valuable knowledge from social media. An illustrative example of such extension can be found in [52] where the authors proposed to combine the aforementioned social and socio-semantic networks into a single model. In short, they use a single matrix representation where the diagonal sub-matrices represent the relation between the same type of entities — agents and concepts — and the off-diagonal matrices represent the relation between different ones — agent/concept and concept/agent. From the point of view of data modeling, HINs are very related to socio-semantic network models, even though HINs have been introduced as more general modeling tools while socio-semantic networks have emerged and are used in a specific application context.

A final work worth mentioning in this class is [104], where topic modeling is performed using an extended model considering not only the association between topics, words and documents, but also the association between documents and their authors. However, this has not been included in our summary table because it introduces a generative model to summarize the data in the form of parameters indicating the probability that a given actor produces a given set of words, but not to represent the empirical data showing which actors have written what text.

A.1.4 Time & text & topology

Many works in the literature have dealt with time, text and topology using ad hoc models specifically designed to capture relevant aspects of specific platforms such as Twitter. For example, in [121] a communication network is built in three steps: (1) conversation trees are extracted from the dataset by inversely following the chain of Twitter user interactions (replies, mentions and retweets); (2) the trees are pruned based on the time elapsed between the root tweet and

the overlap of tweets and participants in the tree; (3) finally, all trees are merged to generate a simple weighted graph of interactions between authors. The tree of pruned conversations, also known as *polyadic conversation* has recurrently appeared in the literature of network modeling and information retrieval [82] to describe communication patterns, but there is no consensus on what is the best method to build such model (e.g., how to compute the length of a conversation in terms of time and/or tree's depth). In summary, the amount of arbitrary decisions made during network construction make it difficult to reuse these models and the resulting networks in different contexts.

In [109] a temporal model was used to compare the co-growth of two epistemic networks, a Twitter dataset and a set of related blogs, with the underlying social network of contacts. The temporal information attached to the edges of the network is, afterwards, used to compare the order of formation of epistemic and social communities.

Citation networks have received a lot of attention, and include text documents, directed edges between them and also time annotations [42, 6]. In addition, when author co-citation analysis is performed [129], the underlying data model must also contain information about who authored which documents.

Information diffusion processes are often modeled including the diffused information item (meme, blog post, etc.), the actors propagating it, and the times of propagation. This is for example the case of the model used in [80]. However, the majority of these models do not use the text to perform the data analysis, but (sometimes) to define the links between documents. Time can also be used to infer network structure based on the observation of propagation events. For example, the observation of a group of individuals repeatedly re-sharing common tweets in the same temporal order may suggest that these people are connected, and that information (tweets, in this case) passes through these hidden connections [47]. In [110] existing theoretical diffusion models for interconnected networks are reviewed, extending concepts in information diffusion to a multilayer model.

In order to preserve as much original information as possible, Šćepanović et al. use a more generic process to build the network, mixing techniques from social network and semantic analysis. In their work, the communication network is modeled as a simple, temporal graph using the Twitter "replies" to relate actors with each other. Then, they apply several semantic analysis procedures to generate supporting networks that describe the text-related features. A comparative analysis between the communication network and a subset of the semantic networks is used to study several aspects of the overall system such as semantic homophily and its evolution. However, from a modeling point of view text is not explicitly represented in this model, but coded inside the semantic layers. We will later use a related approach to exemplify how to use our model for data analysis.

Some attention has also been devoted to models describing co-evolutionary networks [49, 85]. Some of these models allow the representation of a status associated to each node. Statuses can be used for example to represent the political affiliation of the person represented by the node. In growing network

Figure 34: A taxonomy of multiplex community detection algorithms

models, the status can influence the evolution of the network for example by increasing the probability that people will create connections with other individuals sharing the same political affiliation [67, 78]. As for the case of simple network growing models, time is not typically kept at the end of the growing process, and in addition status has not been used to model text to the best of our knowledge. Therefore, we have not included these works in our summary table, even if we consider them potentially relevant for this field if extended in the future.

A.2 Community detection in multiplex networks

In this section we provide a taxonomy of multiplex community detection methods. We classify the main existing algorithms according to this taxonomy. Figure 34 and Table 12 show an overview of the related methods.

Table 12: Multiplex community detection algorithms covered in this survey

Algorithm	Notation	Reference
Non-Weighted Flattening	NWF	[9]
Weighted Flattening (Edge Count)	WF_EC	[9]
Weighted Flattening (Neighbourhood)	WF_N	[9]
Weighted Flattening (Differential)	WF_Diff	[65]
Cluster-Based Similarity Partitioning Algorithm	CSPA	[123]
Canonical Correlations Analysis	CCA	[123]
Frequent pattern mining-based community discovery	ABACUS	[12]
Subspace Analysis on Grassmann Manifolds	SC-ML	[36]
Ensemble-based Multi-layer Community Detection	EMCD	[120]
Principal Modularity Maximization	PMM	[122]
Generalized Louvain	GLouvain	[60]
Locally Adaptive Random Transitions	LART	[73]
Multi Layer Clique Percolation Method	ML-CPM	[124]
Modular Flows on Multilayer Networks	Infomap	[29, 39]
Multi Dimentional Label Propagation	MLP	[16]
Multilayer local community detection	ML-LCD	[57]
Andersen-Chung-Lang cut	ACLcut	[59]

Our taxonomy is constituted of three levels of comparison among multiplex community detection methods. The top-level distinction answers whether the method is *global* or *local*. Global methods are designed to discover all possible communities in a network, thus requiring knowledge on the whole network structure. Conversely, local methods (also known as *node-centric*) are *query-dependent*, i.e., they are designed to discover the (local) community of a set of query nodes as an input.

The second level of distinction concerns how the method solves the multiplexity problem. Three main approaches, illustrated in Figure 35, have been used to solve this problem. The first approach, *flattening*, consists in simplifying the multiplex network into a simple graph by merging its layers, then applying a traditional (i.e., designed for simple graph) community detection algorithm. The second approach, *layer-by-layer*, consists in processing each layer of the multiplex network separately, then aggregating the resulting solutions. The third class of algorithms, *multi-layer*, operates directly on the multiplex network model.

The third level in our taxonomy corresponds to a fine-grained distinction that defines the mathematical tools used to identify the multiplex communities. *Flattening* branch distinguishes between *non-weighted flattening* and *weighted flattening*, where the latter reflects some structural properties of the multiplex network in the form of weights assigned to the output simple-graph edges, then it uses a traditional community detection method for weighted graphs. *Layer-by-layer* has four branches: *pattern mining*, *matrix composition*, *consensus matrix* and *spectral clustering*. The former detects communities in each layer separately using a simple-graph community detection, then makes use of pattern mining algorithms to aggregate the resulting communities; the matrix-composition-based methods identify structural features from each layer of the network via modularity analysis, and then integrate them to identify community structure among actors; the consensus-matrix-based methods combine multiple solutions over the various layers to infer a single community structure that is representative of the set of layer-specific community structures; the spectral-clustering-based methods combine the characteristics of individual graph layers to form a low dimensional representation of the original data then makes use of spectral clustering to identify the multiplex communities. *Multi-layer* includes *clique*-based methods, which exploit the concept of multi-layer clique to identify multiplex communities, *random walk*-based methods, which introduce a multi-layer random walker that can traverse interlayer edges, *modularity*-based methods, which define a multi-layer modularity function and optimize it to produce the community structure solution, *label propagation* methods, which utilize a multi-layer affinity measure among actors given their connections on different layers and then use a labeling method for the actors controlled by these affinity scores, and *within-group connectivity* for local methods, which define a multi-layer within-group connectivity function for the multiplex community and try to maximize that function.

The output of a community detection algorithm for multiplex networks is a set of communities $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ such that each community contains a non-empty subset of V. We will also use *cluster* as a synonym of community, when it is clear from the context that we are referring to the set of nodes assigned to a community, not to the subgraph underlying a community; an analogous remark applies to *clustering* as a set of communities. Figure 36 illustrates different possible types of clusterings on a multiplex network.

A clustering $\mathcal{C}$ is **total** if every node in V belongs to at least one community, and it is **partial** otherwise. We also call a clustering **node-overlapping** if

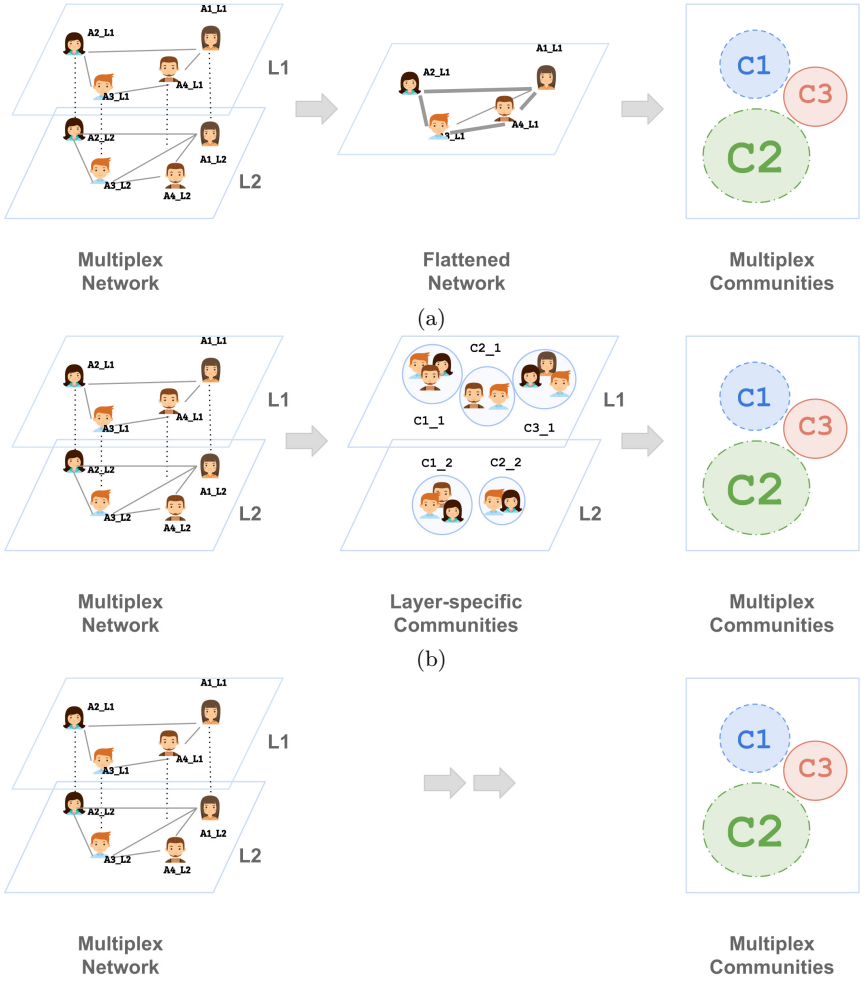
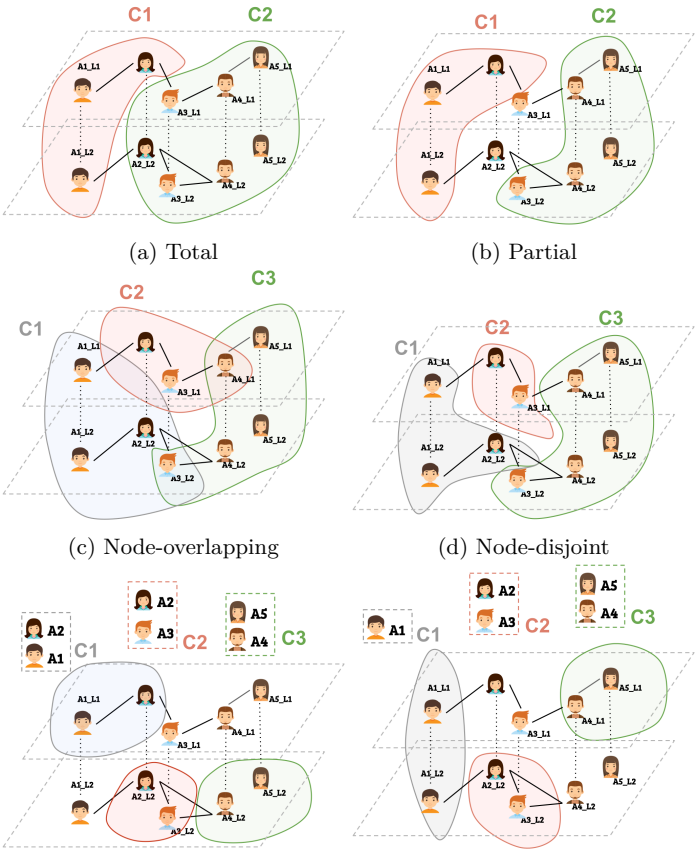


Figure 35: Three main global approaches to discover communities in multiplex networks. **a)** Flattening approach, **b)** layer by layer approach, and **c)** multi-layer approach

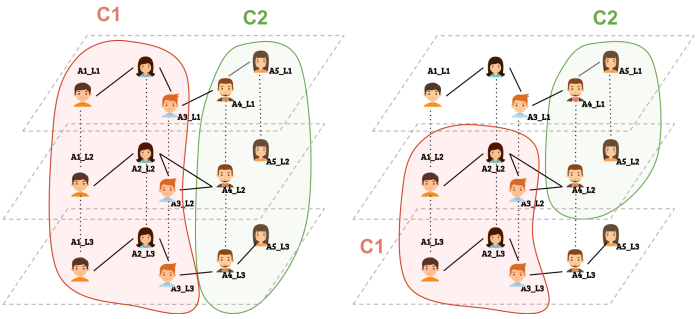


(a) Total (b) Partial

(c) Node-overlapping (d) Node-disjoint

(e) Actor-overlapping (f) Actor-disjoint

Figure 36: Different types of clustering on a multiplex network



(a) Pillar communities (b) Semi-pillar communities

Figure 37: Pillar and semi-pillar multiplex community structures

there is at least a node that belongs to more than one cluster, otherwise the clustering is called **node-disjoint**. Analogously, if there is at least an actor belonging to more than one cluster we call the clustering **actor-overlapping**, otherwise it is called **actor-disjoint**. Notice that a node-overlapping clustering is also actor-overlapping, while an actor-overlapping clustering may or may not be node-overlapping. Table 13 reports the type of clustering produced by each reviewed method. In addition, a multiplex community is called **pillar** if all the nodes of each actor included in the community are covered by that community and **semi-pillar** if the majority of (but not all) the nodes of each actor included in a community belong to that community (Figure 37).

Table 13: Types of clustering featured by the reviewed methods. (*) indicates that the answer depends on the single-layer clustering algorithm used by the method.

Algorithm	Actors	Nodes	Coverage
NWF	*	*	*
WF_N	*	*	*
WF_Diff	*	*	*
CSPA	Disjoint	Disjoint	Total
CCA	Disjoint	Disjoint	Total
ABACUS	Overlapping	Disjoint	Partial
SC-ML	Disjoint	Disjoint	Total
EMCD	Disjoint	Disjoint	Total
PMM	Disjoint	Disjoint	Total
GLouvain	Overlapping	Disjoint	Total
LART	Overlapping	Disjoint	Total
ML-CPM	Overlapping	Overlapping	Partial
Infomap	Overlapping	Overlapping	Total
MLP	Disjoint	Disjoint	Total

In their survey work, [66] discussed a classification framework based on a set of desired properties for multilayer community detection methods. These properties are: multiple layer applicability, consideration of each layer's importance, flexible layer participation (i.e., every community can have a different coverage of the layers' structure), no-layer-locality assumption (e.g., independence from initialization steps biased by a particular layer), independence from the order of layers, algorithm insensitivity, and overlapping layers (e.g., two or more communities can share substructures over different layers).

We observe that the first of the above listed properties (i.e., multiple layer applicability) should be true for all methods, therefore we do not elaborate on this further. By contrast, the second property (i.e., consideration of each layer's importance) deserves a clarification. Also, both properties about independence from node/layer order express a non-deterministic behavior of the community detection method, therefore we will treat them as a single property. The insensitivity property (i.e., independence or robustness against main tunable input

parameters) is instead specialized into two properties: one referring to whether the number of communities is automatically derived or is an input parameter, and one about whether an output multiplex community may contain a subset of the layer-relations (instead of just being an induced subgraph of the community node set).

In light of the above considerations, we next define four additional properties on top of the characteristics previously discussed. Table 14 organizes the reviewed methods according to these four properties.

- **P1: Layer relevance weight distribution.** Some methods may take into consideration each layer's importance, thus allowing the assignment of different weights to the layers in order to control their contribution to the computation of the multiplex community structure. Such layer weights could be learned based on the layer characteristics, or could be user-provided based on a-priori knowledge (e.g., user preferences).
- **P2: Determinism.** This refers to whether a method has a deterministic behavior, i.e., its output is independent from the order of examination of the nodes and/or layers.
- **P3: Auto-detection of the number of communities.** Some methods expect the number of communities to be decided ahead of time while other methods can automatically define the number of communities.
- **P4: Community structure inference.** The default is that the structure of the communities corresponds to the multiplex subgraph induced by the communities' node-sets, i.e., the internal and external links of the communities coincide with all links from the multiplex graph. Nevertheless, a method might detect communities such that the set of layers is only partially exploited to infer the community structure solution, e.g., in order to optimize the multilayer-modularity of the solution [120].

A.3 Network layouts

In this section we describe existing approaches to compute network layouts with a single type of edges, also known as monoplex networks, and for multiplex networks.

A.3.1 Monoplex Network visualization

Many layouts have been designed to visualize monoplex networks. Here we briefly review the ones that are more relevant for our approach. For an extensive review, the reader may consult [127].

Multi-scale layouts first create some core sub graphs, then they add other nodes until all nodes and edges have been processed [127, 69]. *Random layouts*

Table 14: Algorithmic properties featured by the reviewed methods. (*) indicates that the answer depends on the single-layer clustering algorithm used by the method. (-) indicates that this feature is not measurable

Algorithm	P1	P2	P3	P4
NWF	×	*	*	*
WF_N	×	*	*	*
WF_EC	×	*	*	*
WF_Diff	✓	*	*	*
CSPA	×	✓	×	✓
CCA	×	✓	✓	✓
ABACUS	×	*	✓	×
SC-ML	✓	✓	×	×
EMCD	✓	✓	×	✓
PMM	×	×	✓	×
GLouvain	✓	×	✓	×
LART	×	✓	✓	×
ML-CPM	×	✓	✓	×
InfoMap	×	✓	✓	×
MLP	×	×	✓	×
ML-LCD	✓	✓	-	×
ACLcut	×	✓	-	×

[32] and *circular layouts* [81] are two categories of layouts which are appropriate for small graphs with few nodes and edges, because they do not consider aesthetic criteria: many edge crossings and node overlappings can appear.

Among the most used visualization methods, *force-directed* algorithms consider a graph as a physical system where forces change the position of nodes. The two best known force-directed layouts are *Fruchterman-Reingold* [44] and *Kamada-Kawai* [61, 58]. In the Fruchterman-Reingold layout nodes have repulsive power and push other nodes away, while edges attract neighboring nodes. In this layout nodes are considered as steel rings having similar loads and edges are like springs attracting neighbouring rings. This algorithm consists of three main steps. First, all nodes are distributed randomly. Second, repulsive forces separate nodes. The value of repulsive force depends on the positions of the nodes. Third, for each edge and based on the position of nodes after repulsion, attractive forces are calculated [44]. In the Kamada-Kawai layout an energy function is defined for the whole graph based on shortest paths between nodes, and positions are iteratively updated until the graph's energy is minimized [61].

Bannister et al. [4] proposed a force-directed layout to change the position of nodes so that more graph-theoretically central nodes are pushed towards the centre of the diagram. In this algorithm, an additional force called gravity is used to change the position of more central nodes. For each node v in a graph

G the position of the node is influenced by the following force:

$$I[v] = \sum_{u,v \in V} f_r(u,v) + \sum_{(u,v) \in E} f_a(u,v) + \sum_{v \in V} f_g(v) \tag{3}$$

where f_r and f_a are respectively repulsive and attractive forces, and f_g is the gravity force, measured as:

$$f_g(v) = \gamma_t M[v](\xi - P[v]) \tag{4}$$

In this equation $M[v]$ is the mass of node v, which can be set according to node degree, $P[v]$ is the position of v, γ_t is the gravitational parameter and $\xi = \Sigma_v P[v]/|V|$ is the centroid of all nodes. Notice that forces in the equations above are vectors. Other extensions of force-directed layouts have considered the inclusion of additional domain-specific information in the definition of the forces.

Traditional force-directed methods are suitable only for small networks, but they are very popular because they often practically succeed in highlighting communities (that is, groups of well connected nodes) and increasing graph readability. To address their shortcomings, force-based layout algorithms are sometimes split into multiple phases, with an initial preprocessing of the data to generate good starting node dispositions or to reduce data complexity. As an example, Gajer et al.'s method first partitions the graph into subgraphs. The smallest subgraph is then processed independently and thus more efficiently. Afterwards, a force-directed refinement round changes the values of initial node positions and the next smallest subgraph is added to the previous one, with these steps being repeated untill all nodes have been processed.

Another family of layouts, that can also be combined with force-directed algorithms, are *constraint-based* layouts [37]. These layouts force nodes to appear at specific positions. For example, nodes are placed on a frame in a way that they do not overlap, or are horizontally and vertically aligned, as in the *orthogonal* layout [127, 37]. In these layouts it is more difficult to isolate special structures such as communities and time complexity is noticeably high.

One important assumption in graph drawing is a correspondence between some aesthetic features of the diagrams and their readability. Therefore, some visualization algorithms explicitly target these features. One such criterion is that too many edge crossings make a graph more difficult to interpret. The crossing number $cr(G)$ of a graph G is the smallest number of crossings appearing in any drawing of G [111]. Several algorithms have been proposed to reduce edge crossing in monoplex networks. For example, Shabbeer et al. [114] developed a stress majorization algorithm. In [111] and [20] the concept of edge crossing is elaborated and equations for measuring the number of edge crossings in different graphs are reviewed. Another aesthetic feature impacting graph readability is node overlapping. Two popular methods to reduce node overlapping were proposed in [55, 72].

A.3.2 Multiplex Network Visualization

Different methods have been discussed for visualizing multiplex networks. We categorize these methods into four main classes: slicing, flattening, simplification-based, and indirect.

One way of visualizing multiplex networks is to consider each layer or relationship type as a monoplex network and to connect these monoplex networks using inter-layer edges [30]. The layers can have aligned layouts or independent layouts, as shown in our initial example. Aligned layouts help users find similar nodes in different layers by forcing the same node to have the same coordinates on all layers, but structures existing only on one layer (for example communities) may not be clearly visualized. Independent layouts can show specific structures of each layer, but may hide inter-layer patterns [106].

In flattening-based methods all nodes and edges are placed on the same plane. In a *node-colored* network, nodes from different layers are shown with different colors [68], while for multiplex networks colors can be used to distinguish edges of different types. Apart from suffering from the same problems of aligned slicing, the disadvantage of this method is that for networks with high edge density relationships among nodes can be hidden by edges from non-relevant layers and readability quickly decreases due to the network's clutter.

Given the information overload associated to the presence of multiple types of edges, a third approach consists in adding a so called *simplification* step before using one of the two approaches above [106]. The simplification step removes edges or layers that are not considered relevant for the visualization task. In general, this can also be used as a pre-processing step before computing our multiforce layout.

The last approach tries to visualize information derived from the network instead of directly visualizing the original layers, which are again considered to be too complex to allow a simple visual representation. Renoust et al. [102] proposed a system for visual analysis of group cohesion in flattened multiplex networks. This system, called *Detangler*, creates a so-called substrate network from unique nodes of the multiplex network and a so-called catalyst network from edges of different types. Erten et al. proposed three modified force-directed approaches for creating slicing, flattened and split views of multilayer networks by considering edge weights and node weights. The weights of nodes and edges are based on the number of times they appear in different layers. In this approach interlayer relationships between nodes are ignored and node weights are the same for all nodes when multiplex networks are visualized, so this approach does not consider the specific features of the networks targeted in our work.

An extreme case of indirect methods, that we mention for completeness, consists in not visualizing nodes and edges at all but only indirect network properties, such as the degree of the nodes in the different layers or other summary measures [30, 106, 99]. These approaches are complementary to graph drawing, and can also be used in combination with our proposal.

B The `multinet` library

In this section we describe multinet, an R package to analyze multiplex social networks represented within the more general framework of multilayer networks.

Several packages for network analysis are available in R. Notable examples are statnet [50], containing libraries such as sna [23], network [22, 21] and ergm [56], igraph [28] and RSiena [103]. Multinet complements these libraries with several functions to analyze multiplex networks. In particular, the package provides functions focusing on the multilayer structure of the networks, for example to find how relevant some layers are for an actor or to discover communities spanning multiple layers. Individual layers of a multiplex network, each corresponding to a simple network, can instead be analyzed using the abovementioned packages, and in particular multinet contains functions to translate the layers into igraph objects. The methods provided by multinet are distinct from the ones provided by the multiplex library [94].

The multinet package also includes the following external code: eclat[6] (for association rule mining), Eigen[7] and spectra[8] (for matrix manipulation), Infomap[9] (for the Infomap community detection method) and Howard Hinnant's date and time library[10].

B.1 The Rcpp_RMLNetwork class

The multinet package defines the Rcpp_RMLNetwork class to represent multilayer networks. Objects of this type are used as input or returned as output by most functions provided by the package.

Internally, all the objects constituting the network are stored in sets with logarithmic lookup and random access time, implemented as skip lists. This solution is (linearly) less efficient than using a set in the C++ standard library, but supports quick random access to the objects in the set, which is important when synthetic networks are generated. For efficiency reasons, most of the functions in the package are written in native C++ and integrated with R using the Rcpp [38] library. Storage requirements for the network class are on the order of the number of vertices plus the total number of edges (inter-layer and intra-layer).

The ml.empty() function returns an empty multilayer network, not containing any actor, layer, vertex or edge[11]. The function accepts an optional character argument name, indicating the name of the network.

```
R> ml.empty()

Multilayer Network [0 actors, 0 layers, 0 vertices, 0 edges (0,0)]
```

[6] http://www.borgelt.net/eclat.html
[7] http://eigen.tuxfamily.org
[8] https://spectralib.org
[9] http://www.mapequation.org
[10] https://github.com/HowardHinnant/date
[11] Other ways to create networks, explained later, are the function read.ml() to load networks from files and the grow.ml() function to produce synthetic networks.

For convenience, the call to any of the network's constructors and readers returns an S4 object compatible with the R print function. Otherwise, all the other functions' return types are, by design, either (i) a named list of elements (if the data is not relational) or (ii) a data frame.

B.1.1 Adding, retrieving and deleting network objects

Objects in a Rcpp_RMLNetwork object can be queried using a set of utility functions. Built-in functions for retrieving and updating objects have the same signature name: op.objects.ml, where objects can be actors, layers, vertices or edges, and op is either blank, if we want to list the objects, or is the name of a specific operation: num, to compute the number of objects of the requested type, add or delete. If the number of actors is requested without specifying any layer, the total number of actors is returned, including those not present in any layer.

All the aforementioned functions require an Rcpp_RMLNetwork object as first argument. Listing functions operating on actors and vertices also require an array of layer names: only the actors/vertices in the input layers are returned. If the array is empty, all the actors/vertices in the network are returned. Listing functions operating on edges, instead, require two parameters: one indicating the layer(s) from where the edges to be extracted start, and a second one with the layer(s) where the edges to be extracted end. If an empty list of starting layers is passed (default), all the layers are considered, while if an empty list of ending layers is passed (default), the ending layers are set as equal to those in the first parameter.

Now we can show a small example of how these functions work together. We start by creating an empty network with two layers, named *UL* (upper layer) and *BL* (bottom layer), respectively.

```
R> net <- ml.empty()
R> add.layers.ml(net, c("UL", "BL"))
R> layers.ml(net)

[1] "BL" "UL"
```

New layers are by default undirected, that is, edges added to them are treated as undirected. Directed layers are created by setting the directed parameter to TRUE, or using the set.directed.ml() function, which is necessary if we want to set directed intralayer edges. This function takes an Rcpp_RMLNetwork object and a directionality data frame as input. The next fragment of code changes the directionality of the inter-layer edges between the bottom and upper layers.

```
R> dir <- data.frame(layer1="UL", layer2="BL", dir=1)
R> set.directed.ml(net, dir)
R> is.directed.ml(net)

  layer1 layer2 dir
1     BL     BL   0
```

```
2     BL     UL   1
3     UL     BL   1
4     UL     UL   0
```

Then, we create three actors $A = \{A1, A2, A3\}$.

```
R> add.actors.ml(net, "A")
R> add.actors.ml(net, c("B", "C"))
```

We can check that the actors have been added correctly:

```
R> num.actors.ml(net)

[1] 3

R> actors.ml(net)

[1] "C" "A" "B"
```

The next step to populate a network is to add actors to layers, where a pair actor-layer defines a vertex. Notice that if we try to create the vertices without having added the corresponding actors, the library will raise an error.

```
R> vertices <- data.frame(
+      actors = c("A",  "B",  "C",  "A",  "B",  "C"),
+      layers = c("UL", "UL", "UL", "BL", "BL", "BL"))
R> vertices

  actors layers
1      A     UL
2      B     UL
3      C     UL
4      A     BL
5      B     BL
6      C     BL

R> add.vertices.ml(net, vertices)
R> vertices.ml(net)

  actor layer
1     C    BL
2     A    BL
3     B    BL
4     C    UL
5     A    UL
6     B    UL
```

From the previous command you can see how the objects in a network are stored into (mathematical) sets, that is, they are unordered: we cannot assume that actor A will always be listed before actor B, and we have to sort the results if we want to keep a specific order.

We can now add some intra-layer edges, in this case between all the vertices in the upper layer and between vertices A and C in the bottom one. In addition, we create inter-layer edges between vertices $((A, UL), (B, BL))$ and $((A, UL), (C, BL))$. We begin by creating two data frames, one for each type of edges:

```
R> intra_layer_edges <- data.frame(
+      actors_from = c("A",  "A",  "B",  "A"),
+      layers_from = c("UL", "UL", "UL", "BL"),
+      actors_to =   c("B",  "C",  "C",  "C"),
+      layers_to =   c("UL", "UL", "UL", "BL")
+  )
R> intra_layer_edges

  actors_from layers_from actors_to layers_to
1           A          UL         B        UL
2           A          UL         C        UL
3           B          UL         C        UL
4           A          BL         C        BL

R> inter_layer_edges <- data.frame(
+      actors_from = c("A", "A"),
+      layers_from = c("UL", "UL"),
+      actors_to =   c("B", "C"),
+      layers_to =   c("BL", "BL")
+  )
R> inter_layer_edges

  actors_from layers_from actors_to layers_to
1           A          UL         B        BL
2           A          UL         C        BL
```

Now we can add these edges to the network, and observe the result.

```
R> add.edges.ml(net, intra_layer_edges)
R> add.edges.ml(net, inter_layer_edges)
R> edges.ml(net)

  from_actor from_layer to_actor to_layer dir
1          A         BL        C       BL   0
2          A         BL        B       UL   1
3          A         BL        C       UL   1
4          A         UL        B       UL   0
5          B         UL        C       UL   0
6          A         UL        C       UL   0
```

```
R> edges.ml(net, layers1 = "BL")

  from_actor from_layer to_actor to_layer dir
1          A         BL        C       BL   0
```

Notice that as we have only passed one argument (layers1 = "BL"), edges.ml() returns only the intra-layer edges in the *BL* layer.

B.1.2 Handling attributes

When we study a multilayer network, we can be interested in representing different types of actors, add some categorical attribute to vertices or use a numerical value to represent the strength of the ties. The multinet library provides a set of functions to create attributes and add and retrieve attribute values. attributes.ml() returns a data frame with two columns, the name of the attribute and its type. As most of the functions in the library, the function accepts a filtering parameter, target, to limit the query to specific types of objects: "actor" (attributes attached to actors), "vertex" (attributes attached to vertices) or "edge" (attributes attached to edges). All the functions handling attributes use target = "actor" by default.

```
R> attributes.ml(net)

[1] name type
<0 rows> (or 0-length row.names)
```

The list of attributes of a newly created network is empty. We can create attributes by calling the add.attributes.ml() function and passing a network, names of the attributes, types of the attributes ("string" or "numeric") and the target as parameters. For example, the following code creates two string attributes for actors (notice that "actors" is the default target, and "string" is the default attribute type):

```
R> add.attributes.ml(net, c("name", "surname"))
R> attributes.ml(net)

     name   type
1 surname string
2    name string
```

Using the add.attributes.ml() function we can also specify different attributes for nodes and edges on individual layers, for which we must supply the layer parameter. If we want, instead, to manage inter-layer edges two parameters are needed, layer1 and layer2, so that the attribute only applies to inter-layer edges from the first layer to the second and vice-versa. The example below shows how to use these parameters in practice to create a string attribute for the vertices in the bottom layer.

```
R> add.attributes.ml(net, "username", type = "string",
+                     target = "vertex", layer = "BL")
R> attributes.ml(net, target = "vertex")

  layer     name   type
1    BL username string
```

At this point the get.values.ml() and set.values.ml() functions can be used to set and retrieve attribute values.

```
R> set.values.ml(net, "name", c("A", "B"), values = c("Alice", "Scrondo"))
R> get.values.ml(net, "name", c("A", "C"))

  value
1 Alice
2
```

B.2 Input, output and generation of RMLNetwork data

In the previous section we have introduced the Rcpp_RMLNetwork class and various methods to modify Rcpp_RMLNetwork objects. However, users of the library would more often create Rcpp_RMLNetwork objects by reading them from a file, artificially generating them, or loading some of the datasets directly available in the library.

B.2.1 Importing and exporting data

The multinet package provides two input/output functions: read.ml() and write.ml(). Networks can be read from files using a library-specific text-based format, and written to file using the same format or the GraphML syntax[12]. The multinet format is not compatible with other libraries, but it allows us to specify various details, such as the directionality of intra-layer edges and attributes, as in the following example:

```
#VERSION
2.0

#TYPE
multiplex

#LAYERS
research, UNDIRECTED
twitter, DIRECTED

#ACTOR ATTRIBUTES
affiliation,STRING
```

[12] http://graphml.graphdrawing.org

```
#VERTEX ATTRIBUTES
twitter, num_tweets, NUMERIC

#EDGE ATTRIBUTES
research, num_publications, NUMERIC

#ACTORS
Luca,ITU
Matteo,UU
Davide,UU

#VERTICES
Luca,twitter,53
Matteo,twitter,13

#EDGES

Luca,Matteo,research,9
Luca,Matteo,twitter
```

When we read this multiplex network we can also specify that we want all the actors to be present in all the layers, using the align parameter. The difference between the two obtained networks can be seen by checking the basic network statistics:

```
R> net <- read.ml(file = "example_io.mpx")
R> net

Multilayer Network [3 actors, 2 layers, 4 vertices, 2 edges (2,0)]

R> aligned_net <- read.ml("example_io.mpx", align = TRUE)
R> aligned_net

Multilayer Network [3 actors, 2 layers, 6 vertices, 2 edges (2,0)]
```

Both Rcpp_RMLNetwork objects, net and aligned_net, have two layers and three actors; but the align = TRUE parameter in the second call to the read.ml() adds a new vertex to each layer for every actor in the input file.

When no special information is needed, e.g., there are no attributes, no isolated nodes and all edges are undirected, the format becomes as simple as a list of layer-annotated edges:

```
Luca,Matteo,research
Davide,Matteo,research
Luca,Matteo,friendship
```

A multiplex network can also be created starting from igraph objects, where each graph represents a layer. For this to be possible, the vertices of the graphs must have a name attribute indicating the name of the corresponding actor.

For example, consider the following graphs:

```
R> l1

IGRAPH de3bd83 UN-- 3 3 --
+ attr: name (v/c)
+ edges from de3bd83 (vertex names):
[1] A--B A--C B--C

R> l2

IGRAPH ae70b79 UN-- 2 1 --
+ attr: name (v/c)
+ edge from ae70b79 (vertex names):
[1] A--C
```

They can be added as layers of a multiplex network as follows:

```
R> n <- ml.empty()
R> add.igraph.layer.ml(n, l1, "layer1")
R> add.igraph.layer.ml(n, l2, "layer2")
R> n

Multilayer Network [3 actors, 2 layers, 5 vertices, 4 edges (4,0)]

R> edges.ml(n)

  from_actor from_layer to_actor to_layer dir
1          A     layer1        B   layer1   0
2          A     layer1        C   layer1   0
3          B     layer1        C   layer1   0
4          A     layer2        C   layer2   0
```

B.2.2 Generation

The library provides basic functionality to generate synthetic multiplex networks, following the approach proposed by [84]. This problem is approached by allowing layers to evolve at different rates, based on internal or external dynamics. Internal dynamics can be modelled using existing network models (for example, preferential attachment), assuming that how the layer grows can be explained only looking at the layer itself. External dynamics involve importing edges from other layers. Within this perspective the intuition is that relations existing on a layer might naturally expand over time into other layers (e.g. coworkers starting to add each other as friends on Facebook). The package also allows different growing rates for different layers.

In the following example we create a multiplex network with 3 layers based on the Preferential Attachment [5] and the Erdos-Renyi models [40]. The first and last layers will only evolve according to their internal models (pr.external = 0), while the second will have a probability of .8 of evolving according to external dynamics, that is, importing edges from other layers (pr.external = .8). Note that all the probability vectors must have the same number of fields, one for each layer. By defining pr.internal and pr.external, we are also implicitly defining pr.no.action (1 minus the other probabilities, for each field/layer). In the example, the third layer grows at a lower speed than the others, having an (implicitly defined) pr.no.action = .1.

```
R> models_mix <-
+    c(evolution.pa.ml(3, 1), evolution.er.ml(100), evolution.er.ml(100))
R> pr.internal <- c(1, .2, .9)
R> pr.external <- c(0, .8, 0)
```

The probability to import edges from the other layers in case external events happen is specified using a dependency matrix. The following matrix specifies that the second layer should import edges from the first layer with probability 1 if an external evolutionary event is triggered. It is expected that the values on each row of the matrix add to 1.

```
R> dependency <- matrix(c(1, 1, 0, 0, 0, 0, 0, 0, 1), 3, 3)
R> dependency

     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    1    0    0
[3,]    0    0    1
```

We can now generate the network, with 100 actors and 100 growing steps.

```
R> ml_generated_mix <-
+    grow.ml(100, 100, models_mix, pr.internal, pr.external, dependency)
R> num.edges.ml(ml_generated_mix, layers1 = "l0")

[1] 92

R> num.edges.ml(ml_generated_mix, layers1 = "l1")

[1] 100

R> num.edges.ml(ml_generated_mix, layers1 = "l2")

[1] 63
```

B.2.3 Predefined data

Another way to obtain network data without having to manually construct it is to load some well-known networks already available inside the package. These are loaded using functions beginning with "ml", followed by the name of the network, e.g., ml.florentine().

In the remainder of the article we will use the AUCS network, included in the current version of the multinet package as an example dataset and often used in the literature to test new methods. The data, described by [33], were collected at a university research department and include five types of online and offline relations. The population consists of 61 employees, incuding professors, postdocs, PhD students and administrative staff.

```
R> net <- ml.aucs()
R> net

Multilayer Network [61 actors, 5 layers, 224 vertices, 620 edges (620,0)]

R> layers.ml(net)

[1] "coauthor" "lunch"    "leisure"  "facebook" "work"
```

B.3 Data exploration

Multinet provides a basic visualisation function. We can produce a default visualization just by executing plot(net), but to make the plot more readable we shall add a few details. In particular: (1) we explicitly compute a layout that draws each layer independently of the others, as declared by setting interlayer weights (w_inter) to 0, (2) we plot the layers on two rows, to better use the space on the page (grid), (3) we remove the labels from the vertices, to increase readability (vertex.labels = ""), and (4) we add a legend with the names of the layers. The multiforce layout, used for all graph visualizations in this article, is described in Section 3.4.

```
R> l <- layout.multiforce.ml(net, w_inter = 0, gravity = 1)
R> plot(net,
+       layout = l,
+       grid = c(2, 3),
+       vertex.labels = "",
+       legend.x = "bottomright", legend.inset = c(.05, .05)
+       )
```

We can also use the attributes to inspect the relationship between the role of the actors and the topology of the network. We start by retrieving the role of each vertex (vertex_roles), and a list of all the distinct roles.

```
R> attr_values <-
+    get.values.ml(net, actors = vertices.ml(net)[[1]], attribute = "role")
```

```
R> vertex_roles <- as.factor(attr_values[[1]])
R> num_distinct_roles <- length(levels(vertex_roles))
R> levels(vertex_roles)

[1] "Admin"          "Assistant"      "Associate"
[4] "Emeritus"       "NA"             "PhD"
[7] "Phd (visiting)" "Postdoc"        "Professor"
```

Now we can map each vertex role into a color, representing the role of the corresponding actor. To do this, we use the RColorBrewer library, allowing us to produce an appropriate combination of colors.

```
R> color_map = brewer.pal(num_distinct_roles, "Paired")
R> vertex_colors <- color_map[vertex_roles]
```

Plotting works as usual, with an additional parameter to set the vertex colors (vertex.color) and two legends for the edge types and for the roles.

```
R> plot(net,
+       layout = l,
+       grid = c(2, 3),
+       vertex.labels = "",
+       vertex.color = vertex_colors
+       )
R> legend("bottomright",
+         legend=levels(vertex_roles),
+         col = color_map,
+         bty = "n", pch = 20, pt.cex = 1, cex = .5, inset = c(0.05, 0.05)
+         )
R> legend("bottomright",
+         legend=layers.ml(net),
+         bty = "n", pch = 20, pt.cex = 1, cex = .5,  inset = c(0.2, 0.05)
+         )
```

B.4 Measuring a network

A traditional way of measuring a multiplex network is to focus on each layer at a time, considering it as an independent graph. For example, the summary() function computes a selection of measures on all the layers, and also on the flattened network.

```
R> summary(net)

          n   m dir nc       dens        cc      apl dia
_flat_   61 620   0  1 0.33879781 0.4761508 2.062842   4
coauthor 25  21   0  8 0.07000000 0.4285714 1.500000   3
facebook 32 124   0  1 0.25000000 0.4805687 1.955645   4
leisure  47  88   0  2 0.08140611 0.3430657 3.115911   8
```

```
lunch    60 193   0  1 0.10903955 0.5689261 3.188701   7
work     60 194   0  1 0.10960452 0.3387863 2.390395   4
```

The columns indicate:

1. **n** order (number of nodes)
2. **m** size (number of edges)
3. **dir** directionality
4. **nc** number of connected components (strong components for directed networks)
5. **dens** density
6. **cc** clustering coefficient (directed networks are treated as undirected)
7. **apl** average path length
8. **dia** diameter

To compute other functions or perform another type of layer-by-layer analysis we can convert the layers into igraph objects, using the as.igraph() function, for a single (group of) layer(s), or the as.list() function to obtain a list with all the layers as igraph objects in addition to the flattened network. Once the igraph objects have been generated, all the network measures available in igraph can be computed.

```
R> layers <- as.list(net)
R> names(layers)

[1] "_flat_"   "coauthor" "facebook" "leisure"  "lunch"    "work"

R> # Degree centralization of a layer (with igraph)
R> centralization.degree(layers[[3]])$centralization

[1] 0.233871

R> # Actors with highest degree on a layer (with igraph)
R> head(sort(degree(layers[[3]]), decreasing = T))

 U79  U91 U124  U67   U4 U130
  15   14   13   13   12   12
```

B.4.1 Layer comparison

In addition to a layer-by-layer analysis, we can compare layers using several different approaches. All the methods mentioned in this section are explained in Section 3.3.

For example, to quantify the difference between the degree distributions in different layers we can use the layer.comparison.ml() function to produce a table with pair-wise comparisons. The following code computes the dissimilarity between degree distributions, computed using the Jeffrey dissimilarity function (the higher the values, the most dissimilar the two layers).

```
R> layer.comparison.ml(net, method = "jeffrey.degree")
```

```
          coauthor     lunch   leisure  facebook      work
coauthor 0.0000000 2.8966530 0.4521076 2.0214010 0.5917494
lunch    2.8966530 0.0000000 1.3288250 0.4207678 0.8372414
leisure  0.4521076 1.3288250 0.0000000 1.0177980 0.2118452
facebook 2.0214010 0.4207678 1.0177980 0.0000000 0.7106788
work     0.5917494 0.8372414 0.2118452 0.7106788 0.0000000
```

The layer.comparison.ml() function can also be used to compute multiplex-specific comparisons considering the fact that the same actors may be present on the different layers. In fact, one important comparison can be made to check to what extent this is true:

```
R> layer.comparison.ml(net, method = "jaccard.actors")
```

```
          coauthor     lunch   leisure  facebook      work
coauthor 1.0000000 0.4166667 0.4117647 0.2954545 0.4166667
lunch    0.4166667 1.0000000 0.7833333 0.5333333 0.9672131
leisure  0.4117647 0.7833333 1.0000000 0.5192308 0.7833333
facebook 0.2954545 0.5333333 0.5192308 1.0000000 0.5333333
work     0.4166667 0.9672131 0.7833333 0.5333333 1.0000000
```

The function returns 0 if there are no common actors between the pair of layers, and 1 if the same actors are present in the two layers. If there is a strong overlapping between the actors, then we can ask whether actors having a high (or low) degree on one layer behave similarly in other layers. To do this we can compute the correlation between the degrees:

```
R> layer.comparison.ml(net, method = "pearson.degree")
```

```
          coauthor     lunch    leisure  facebook       work
coauthor 1.0000000 0.1486368 0.48084471 0.5472774 0.42719422
lunch    0.1486368 1.0000000 0.28151667 0.3125598 0.24647515
leisure  0.4808447 0.2815167 1.00000000 0.3781743 0.06805041
facebook 0.5472774 0.3125598 0.37817432 1.0000000 0.54060113
work     0.4271942 0.2464752 0.06805041 0.5406011 1.00000000
```

The Pearson (or linear) correlation between the degree of actors in the two layers is in the interval $[-1, 1]$. The smallest value (-1) indicates that high-degree actors in one layer are low-degree in the other and vice versa, while the largest value (1) is returned if high-degree (resp., low-degree) actors in one layer are high-degree (resp., low-degree) actors in the other. It is important to note that the correlation only depends on the number of incident edges for each pair (actor, layer), and not on which actors are adjacent: they can be the same or different actors.

We can also check to what extent actors are adjacent to the same other actors in different layers, by checking the amount of overlapping between edges in the two layers, which will be 0 if no actors that are adjacent in one layer are also adjacent in the other and 1 if all pairs of actors are either adjacent in both layers or in none.

```
R> layer.comparison.ml(net, method = "jaccard.edges")
```

```
           coauthor      lunch   leisure   facebook       work
coauthor 1.00000000 0.06467662 0.1010101 0.05839416 0.09137056
lunch    0.06467662 1.00000000 0.2772727 0.17843866 0.33910035
leisure  0.10101010 0.27727273 1.0000000 0.15846995 0.20512821
facebook 0.05839416 0.17843866 0.1584699 1.00000000 0.18656716
work     0.09137056 0.33910035 0.2051282 0.18656716 1.00000000
```

The package provides additional similarity functions, listed in Table 15.

B.4.2 Degree and degree deviation

Various functions can be used to measure individual actors. As a starting point, the following is the list of highest-degree actors on the whole multiplex network:

```
R> deg <- head(sort(degree.ml(net), decreasing = T))
R> deg
```

```
  U4  U67  U91 U123  U79 U110
  49   47   46   44   44   41
```

However, in a multiplex context degree becomes a layer-specific measure. We can no longer just ask "who is the most central actor" but we should ask "who is the most central actor on this layer?" Let us see how the most central actors look like when we "unpack" their centrality on the different layers:

```
R> data.frame(
+    facebook = degree.ml(net, actors = names(deg), layers = "facebook"),
+    leisure =  degree.ml(net, actors = names(deg), layers = "leisure"),
+    lunch =    degree.ml(net, actors = names(deg), layers = "lunch"),
+    coauthor = degree.ml(net, actors = names(deg), layers = "coauthor"),
+    work =     degree.ml(net, actors = names(deg), layers = "work"),
+    flat =     deg
+    )
```

Overlapping	Distribution dissimilarity	Correlation
jaccard.actors	dissimilarity.degree	pearson.degree
jaccard.edges	KL.degree	rho.degree
jaccard.triangles	jeffrey.degree	
coverage.actors		
coverage.edges		
coverage.triangles		
sm.actors		
sm.edges		
sm.triangles		
rr.actors		
rr.edges		
rr.triangles		
kulczynski2.actors		
kulczynski2.edges		
kulczynski2.triangles		
hamann.actors		
hamann.edges		
hamann.triangles		

Table 15: Similarity functions implemented in the library.

```
     facebook leisure lunch coauthor work flat
U4         12       1    15       NA   21   49
U67        13       2    12       NA   20   47
U91        14      14     7        3    8   46
U123       11      NA     6       NA   27   44
U79        15       7    13       NA    9   44
U110        9       7     7        4   14   41
```

From the above result we can see how neighbors may not be equally distributed across the layers. Actor $U4$, for example, has the largest degree within the 6 actors analyzed in both the facebook layer and the flattened network. However, it has no presence in the coauthor layer and a very small degree in the leisure layer. If we want to quantify to what extent actors have similar or different degrees on the different (combinations of) layers, we can compute the standard deviation of the degree:

```
R> sort(degree.deviation.ml(net, actors = names(deg)))
```

```
    U110      U91      U79      U67       U4     U123
3.310589 4.261455 5.230679 7.418895 8.133880 9.987993
```

B.4.3 Neighborhood and exclusive neighboorhood

The neighbors of an actor a are those distinct actors that are adjacent to a on a specific input layer, or on a set of input layers. While on a single layer degree and

neighborhood have the same value, they can be different when multiple layers are taken into account, because the same actors can be adjacent on multiple layers leading to a higher degree but not a higher neighborhood.

```
R> degree.ml(net, actors = "U4", layers = c("work", "lunch"))

U4
36

R> neighborhood.ml(net, actors = "U4", layers = c("work", "lunch"))

U4
21
```

The xneighborhood.ml() function (exclusive neighborhood) counts the neighbors that are adjacent to a specific actor only on the input layer(s) [**?**]. A high exclusive neighborhood on a layer (or set of layers) means that the layer is important to preserve the connectivity of the actor: if the layer disappears, those neighbors would also disappear.

```
R> neighborhood.ml(net, actors = "U91", layers = c("facebook", "leisure"))

U91
 22

R> xneighborhood.ml(net, actors = "U91", layers = c("facebook", "leisure"))

U91
 13
```

B.4.4 Relevance

Based on the concept of neighborhood, we can define a measure of layer relavance for actors [11]. relevance.ml() computes the ratio between the neighbors of an actor on a specific layer (or set of) and the total number of her neighbors. Every actor could be described as having a specific "signature" represented by her presence on the different layers.

```
R> data.frame(
+    facebook = relevance.ml(net, actors = "U123", layers = "facebook"),
+    leisure =  relevance.ml(net, actors = "U123", layers = "leisure"),
+    lunch =    relevance.ml(net, actors = "U123", layers = "lunch"),
+    coauthor = relevance.ml(net, actors = "U123", layers = "coauthor"),
+    work =     relevance.ml(net, actors = "U123", layers = "work")
+    )

      facebook leisure     lunch coauthor      work
U123 0.3793103      NA 0.2068966       NA 0.9310345
```

Similarly to neighborhood also relevance can be defined using the concept of exclusive neighbor. The xrelevance.ml() function measures how much the connectivity of an actor (in terms of neighbors) would be affected by the removal of a specific layer (or set of layers):

```
R> data.frame(
+    facebook = xrelevance.ml(net, actors = "U123", layers = "facebook"),
+    leisure =  xrelevance.ml(net, actors = "U123", layers = "leisure"),
+    lunch =    xrelevance.ml(net, actors = "U123", layers = "lunch"),
+    coauthor = xrelevance.ml(net, actors = "U123", layers = "coauthor"),
+    work =     xrelevance.ml(net, actors = "U123", layers = "work")
+    )

       facebook leisure lunch coauthor      work
U123 0.06896552      NA     0       NA 0.5172414
```

B.4.5 Distances

In addition to single-actor measures, the library can also be used to compute multilayer distances between pairs of actors. Distances are defined by [85] as sets of lengths of Pareto-optimal multidimensional paths. As an example, if two actors are adjacent on two layers, both edges would qualify as Pareto-optimal paths from one actor to the other, as edges on different layers are considered incomparable (that is, it is assumed that it makes no sense in general to claim that two adjacent vertices on Facebook are closer or further than two adjacent vertices on the co-author layer). Pareto-optimal paths can also span multiple layers.

```
R> distance.ml(net, "U91", "U4")

   from to coauthor lunch leisure facebook work
1   U91 U4        2     1       0        0    0
2   U91 U4        0     2       0        0    0
3   U91 U4        0     1       1        0    0
4   U91 U4        1     0       2        0    0
5   U91 U4        0     0       3        0    0
6   U91 U4        0     0       0        1    0
7   U91 U4        2     0       0        0    1
8   U91 U4        0     0       1        0    1
9   U91 U4        0     1       0        0    1
10  U91 U4        0     0       0        0    2
```

B.5 Community detection

A common network mining task is the identification of communities. The library includes the multiplex clique percolation algorithm described in Section 3.1.

In addition, the function glouvain.ml() uses the algorithm described by [**?**] to find community structures across layers, where vertices in different layers can belong to the same or a different community despite corresponding to the same actor. This method belongs to the class of community detection methods based on modularity optimization, that is, it tries to find an assignment of the vertices to communities so that the corresponding value of modularity is as high as possible. Multilayer modularity is a quality function that is high if most of the edges are between vertices in the same community and if vertices corresponding to the same actors are also often in the same community. The function glouvain.ml() accepts three parameters to modify the resolution of the modularity (gamma), the inter-layer weight connectivity (omega) and the number of nodes after which the algorithm will make the computation on the fly without keeping the full data in memory (limit).

```
R> ml_clust <- glouvain.ml(net)
R> head(ml_clust)
```

```
  actor    layer cid
1   U99 coauthor   0
2   U99    lunch   0
3   U99  leisure   0
4   U99     work   0
5  U109    lunch   0
6  U109  leisure   0
```

The result of the function is a data frame with two columns identifying a vertex, as a pair (actor,layer), and a third column with a numeric value (cid) identifying the community to which the vertex belongs.

The library provides other community detection algorithms: ABACUS [11] (for overlapping and partial community detection) and Infomap (for partitioning/overlapping community detection on undirected or directed networks).

C Meetup data

In this section we present the detailed list of MeetUp groups analyzed in the project.

Name	**Country**	**Event date**		**Attendance**			**Events**
		first	*last*	*min.*	*max.*	*avg.*	
Internet of Things London	GB	2011-10-20	2020-11-27	1	349	104,71	91
IoT Zurich	CH	2011-12-05	2018-12-07	4	137	36,97	68
Internet of Things Madrid Meetup	ES	2012-02-11	2018-04-25	5	120	33,00	50
Internet of Things Munich	DE	2012-05-16	2018-11-29	1	318	79,38	39
Mozilla IOT	GB	2012-09-26	2018-05-10	5	88	47,41	17
Internet of Things Bilbao	ES	2013-07-13	2017-10-27	1	12	5,10	10
Internet of Things (IoT) Midlands UK	GB	2013-04-30	2017-11-07	32	63	49,875	16
Internet of Things Lisboa	PT	2013-05-08	2013-06-04	1	19	12,33	3
Internet of Things Stockholm	SE	2013-04-09	2017-09-27	30	134	83,83	30
IoT Berlin	DE	2013-05-21	2016-11-21	27	156	72,33	18
IoT Trento - The Internet of Things Group of Trento	IT	2013-05-25	2016-01-12	1	10	4,11	9
IoTBE - the Belgian Internet of Things community	BE	2013-07-04	2018-09-27	1	112	30,41	80
Hardware Pioneers #London	GB	2013-11-14	2018-12-06	1	252	68,78	41
Warsaw IOT Developers	PL	2013-10-24	2016-02-23	5	15	8,67	3
Internet of Things Novi Sad	RS	2014-02-27	2018-10-04	2	63	18,90	20
Internet of Things Santander	ES	2014-02-21	2018-03-15	5	23	16,11	9
Internet of Things Oxford	GB	2014-01-22	2017-04-27	25	89	46,60	10
Internet of Things Ghent	BE	2014-01-17	2019-04-09	2	65	23,56	37
Internet of Things Guildford	GB	2014-03-24	2018-10-10	8	55	22,77	31
Internet of Things Helsinki	FI	2014-04-09	2018-12-18	1	126	41,28	57
IoT Rhineland	DE	2014-11-27	2018-10-12	1	34	12,63	11
IoT Belfast	GB	2016-05-16	2019-02-20	9	87	46,89	19
IOT & Objets Connectés // Hardware en Auvergne - Rhône-Alpes	FR	2014-05-27	2018-09-19	25	94	53,60	10

Table 16: **MeetUp groups collected**. Part I.

Name	**Country**	**Event date**		**Attendance**			**Events**
		first	*last*	*min.*	*max.*	*avg.*	
IoT Toulouse	FR	2014-05-13	2016-06-09	19	55	37,00	6
Internet of Things Thames Valley	GB	2014-06-24	2018-11-21	1	163	88,72	33
Berlin Internet of Things (IoT) solutions Meetup	DE	2014-05-20	2014-06-05	3	4	3,50	2
IoT Austria - Local Group Vienna	AT	2014-05-14	2018-06-13	0	167	30,46	116
Grenoble Internet of Things and Embedded Systems	FR	2014-06-23	2018-09-25	3	47	22,31	16
Internet of Things Milton Keynes Meetup	GB	2015-11-12	2018-12-04	35	52	41,42	7
IoT Hessen (Frankfurt)	DE	2015-03-12	2019-03-13	2	90	40,09	31
Hamburg Internet of Things (IoT) User Group	DE	2014-10-08	2018-05-28	0	54	27,13	22
The Business of IoT	GB	2014-10-06	2016-03-09	15	91	61,14	7
IoT Bratislava	SK	2014-10-07	2018-10-02	10	45	25,27	37
IoT, IoE & WearableTech - London	GB	2014-11-13	2016-03-15	23	155	57,85	7
Evolution of IoT toward IoT Ecosystems - London Meetup	GB	2014-11-14	2015-05-05	14	21	17,33	3
Open IoT London	GB	2014-12-17	2015-02-19	20	61	42,00	3
London IoT for Art & Entertainment	GB	2014-12-03	2015-05-21	20	49	35,25	4
Automation Process Managment in Cloud like IoT [Gdansk]	PL	NA	NA	1	1	1,00	2
Internet of Things [IoT] Bologna	IT	2015-10-29	2016-10-25	1	25	11,00	3
IoT Austria - Local Group Salzburg	AT	2015-03-10	2018-10-02	1	101	19,44	18
IoT Austria - Local Group Linz	AT	2015-02-26	2017-11-23	1	100	16,57	14
IoT Austria - Local Group Graz	AT	2015-11-04	2017-05-09	1	101	16,20	10
openHAB Meetup Düsseldorf	DE	2015-03-06	2017-03-03	12	29	17,00	4
Internet of Things Malmö	SE	2015-02-18	2016-10-11	3	51	27,15	13
IoTNL - Internet of Things founders & employees	NL	2015-03-20	2015-11-11	46	110	73,50	4
Athens IoT Meetup	GR	2015-03-19	2018-05-10	22	195	70,00	7
IoT Meetup Franken	DE	2015-03-17	2019-11-19	2	26	8,82	37

Table 17: **MeetUp groups collected**. Part II.

Name	Country	Event date		Attendance			Events
		first	*last*	*min.*	*max.*	*avg.*	
Internet of Things Timisoara	RO	2015-03-19	2017-03-27	4	45	22,60	10
Amsterdam IoT Living Lab	NL	2015-04-23	2018-04-30	3	83	43,50	16
Internet of Things - Croatia	HR	2015-04-29	2017-11-21	27	93	69,50	6
IoT Workshop - Budapest	HU	2015-04-28	2018-04-26	47	187	119,94	19
IoT - Internet of Things Kraków	PL	2015-05-12	2017-12-06	1	53	21,40	3
Hungarian IoT Creativity Meetup	HU	2015-05-04	2019-03-05	1	38	16,23	47
IoT Austria - Local Group Klagenfurt	AT	2015-10-17	2017-05-09	1	100	12,00	14
Paris AI, Robotics & IoT Meetup	FR	2015-09-15	2016-07-13	49	82	67,75	4
IoT Dresden - The Internet of Things Group of Dresden	DE	NA	NA	15	15	15,00	1
Brighton 'Internet of Things' Meetup	GB	2015-06-30	2017-12-11	24	82	48,89	9
IOT in Brussels	BE	2015-06-02	2018-11-23	1	49	8,66	63
Internet of Things - Tallinn	EE	2015-06-04	2018-04-04	4	10	6,67	6
Internet of Things (IoT) Leuven	BE	2016-11-24	2016-11-24	22	22	22,00	1
IoT & Big Data Thinkers Meetup	FR	2015-09-10	2016-04-21	51	112	81,50	2
Torino IoT Meetup	IT	2015-07-20	2018-12-17	2	58	10,97	33
IoT & Data Science Innovators UK	GB	2015-09-22	2018-05-09	1	178	75,50	16
Eindhoven Internet of Things	NL	2015-09-24	2018-12-18	1	115	29,79	43
Stuttgart Industrie 4.0 und IoT Meetup	DE	2015-09-16	2019-11-19	1	52	22,57	37
Regensburg Big Data & IoT Enthusiasts	DE	2018-04-180	2018-04-18	38	38	38,00	1
Provence IoT Meetup	FR	2015-11-19	2016-04-21	5	99	40,00	3
LoRaWAN Berlin	DE	NA	NA	2	2	2,00	1
LoRaWAN Hamburg	DE	2015-09-16	2015-09-1	3	4	3,50	2
Internet of Things & Hardware / IOX LAB Düsseldorf	DE	2015-11-12	2019-12-03	1	29	9,85	39
IoT Denmark	DK	2015-11-20	2017-02-20	5	22	16,33	9

Table 18: **MeetUp groups collected**. Part III.

Name	Country	Event date		Attendance			Events
		first	*last*	*min.*	*max.*	*avg.*	
Internet of Things Utrecht	NL	2015-11-300	2018-11-13	1	55	18,37	27
IoT Innovation Lab	DE	2015-11-24	2017-03-20	10	117	40,22	31
IoT Lab - Internet of Things Lab	DE	2016-02-16	2018-11-21	1	45	14,27	11
Copenhagen Bluemix Meetup - Docker, IoT, Watson, Node.red	DK	2016-01-28	2017-06-15	8	32	23,50	4
IoT Praha	CZ	2016-04-28	2016-04-28	15	15	15,00	1
IoT Security - How to trust these Things ?	FR	2016-03-23	2018-06-27	1	56	11,13	15
Data Science for Internet of Things Meetup - London	GB	2016-03-03	2018-11-22	1	92	43,53	41
Meetup Voiture Connectée et Autonome	FR	2016-03-10	2018-12-13	26	171	115,84	19
IoT Austria - Local Group Innsbruck	AT	2016-04-08	2017-05-09	1	101	18,85	7
Bristol and Bath Internet of Things Meetup	GB	2016-04-11	2019-01-28	2	74	55,86	15
IoT Austria - Local Group Dornbirn	AT	2016-04-15	2018-02-20	1	100	8,65	23
Bucharest IoT Meetup	RO	2016-12-17	2018-06-27	9	53	31,00	2
Web of Things	GB	2016-07-08	2017-04-26	34	54	42,66	3
IoTDataViz London	GB	2016-04-12	2017-02-07	40	57	48,50	2
IoT Bucharest	RO	2016-04-04	2018-06-27	2	80	48,75	4
Glasgow Internet of Things	GB	2016-06-21	2018-09-13	5	86	57,18	11
IoT Austria - Local Group Leoben	AT	2016-10-03	2017-05-09	1	100	20,83	6
LoRa IoT network in Apeldoorn	NL	2016-04-21	2018-12-12	5	46	18,12	25
Infocenter Internet der Dinge Tübingen	DE	2016-04-18	2018-11-23	1	9	2,52	25
The Things Network Arnhem	NL	2016-12-15	2017-05-18	11	13	12,00	2
Smart contracts, blockchain et IOT	FR	2016-07-20	2017-11-30	3	17	10,60	5
IoT-Sofia	BG	2016-05-12	2016-06-09	15	18	16,50	2
Internet of Things - Sevilla	ES	2016-06-23	2017-11-09	31	36	33,50	2
Göteborg Drones/Web/IOT/Design Meetup	SE	2016-06-15	2016-06-15	18	18	18,00	1

Table 19: **MeetUp groups collected**. Part IV.

Name	Country	Event date		Attendance			Events
		first	*last*	*min.*	*max.*	*avg.*	
The Key Network (TransIP)(LoraWAN) - Meetup	NL	2017-01-26	2017-03-30	17	35	26,67	3
LPWAN - from prototypes to large-scale production for IoT	SE	2016-06-16	2018-02-08	5	27	13,33	3
IoT Makers Vienna	AT	2016-09-20	2018-03-23	0	13	5,13	14
Internet of Things Alliance	GB	2016-06-22	2018-06-28	30	69	51,17	12
The Things Network / IoT Groningen	NL	2016-07-06	2018-11-28	8	50	16,92	27
Schaffhausen IoT meetup	CH	2016-08-11	2016-08-11	5	5	5,00	1
Software Development for the Internet of Things (IoT)	DE	2016-10-06	2016-10-06	25	25	25,00	1
Internet of Things Amsterdam	NL	2016-09-23	2016-11-23	36	47	41,50	2
The Things Network Münster	DE	2016-08-13	2016-10-10	2	2	2,00	2
LPWAN London	GB	2016-09-08	2018-11-21	54	107	82,13	15
The Things Network - Bern	CH	2016-09-12	2018-03-03	5	29	10,80	5
Copenhagen IOTPEOPLE Meetup	DK	2016-10-10	2017-03-17	52	66	57,50	4
HackWorks by ThoughtWorks	GB	2016-10-12	2016-12-14	11	27	21,33	3
IoT/IIoT Göteborg	SE	2017-03-16	2017-12-12	22	63	43,00	3
LoRaWAN Bristol	GB	2016-10-17	2018-03-19	19	31	23,00	5
Brighton & Sussex Everynet - your local Things Network	GB	2016-11-01	2018-01-30	6	14	9,12	8
Nottingham IoT (Internet of Things)	GB	2018-03-15	2018-12-13	7	28	18,67	9
Internet of Things : Manchester	GB	2016-12-06	2018-10-02	34	80	61,80	10
The Things Network Region Stuttgart	DE	2016-11-23	2019-11-07	1	25	10,28	25
The Things Network Oxford Meetup	GB	2016-11-15	2016-11-15	17	17	17,00	1
hub:raum IoT Academy	PL	2016-11-30	2017-12-16	1	70	22,17	18
IoT Basel	CH	2016-11-03	2018-11-29	15	30	21,38	13
TheThingsNetwork Region München	DE	2016-11-10	2018-12-17	4	17	12,50	14
Makers and Coders (Maynooth)	IE	2017-05-06	2017-10-07	3	7	4,00	4

Table 20: **MeetUp groups collected**. Part V.

Name	Country	Event date		Attendance			Events
		first	*last*	*min.*	*max.*	*avg.*	
Future Cities, IoT & PropTech - London	GB	2017-06-07	2017-07-05	17	26	21,50	2
Dublin Cloud & IoT Meetup	IE	2017-01-25	2017-09-28	80	85	82,50	2
Solent Internet of Things Meetup	GB	2017-01-23	2018-07-18	10	55	35,75	4
IoT Budapest	HU	2017-02-07	2017-10-10	35	103	60,50	6
IoT 101 - Internet of Things & Applications	ES	2017-01-19	2017-01-19	58	58	58,00	1
IoT Austria - Local Group Eisenstadt	AT	2017-03-20	2017-05-09	1	2	1,33	6
IoT Austria - Local Group St.Pölten	AT	2017-02-21	2017-05-09	1	2	1,33	6
Internet of Things Tampere	FI	2017-01-11	2017-08-31	3	40	18,00	4
Agile Landscape	IT	2017-06-05	2018-11-16	1	45	21,50	4
Narrow Band IoT (NB-IoT), Berlin	DE	2017-02-16	2017-06-23	5	70	37,50	2
Microsoft IoT Meetup	CH	2017-03-15	2017-06-06	36	62	49,00	2
IoT Brunch	DE	2017-02-03	2018-11-09	24	108	59,14	27
Nantes IoT / Embedded Linux and Android Meetup	FR	2017-03-07	2018-12-10	12	43	22,11	9
Fagkvelder @ Itera	NO	2017-03-23	2018-10-25	34	50	42,14	7
IoT & BigData Sofia2 Lab	ES	2017-03-08	2018-02-28	42	76	59,40	5
Internet of Things Meetup	NL	2017-04-18	2017-04-18	1	1	1,00	1
LPWAN IoT Hackathon	DE	2017-03-11	2017-03-11	8	8	8,00	1
Women of Wearables	GB	2017-04-26	2018-10-30	4	34	14,73	15
Internet of Things and Industrial Design	NL	2017-04-20	2017-04-20	51	51	51,00	1
STHLMLPWAN	SE	2017-04-20	2018-03-13	10	48	29,00	2
The Things Network Madrid Community	ES	2017-04-06	2018-11-30	2	46	15,44	9
IoT & Blockchain Meetup	DE	2017-05-03	2017-12-01	58	177	98,25	8
Dublin Big data and IoT Meetup	IE	2017-06-20	2017-12-14	21	79	53,85	7
Hoogeveen & De Wolden IoT Meetup	NL	2017-05-24	2018-03-27	12	33	24,33	3

Table 21: **MeetUp groups collected**. Part VI.

Name	Country	Event date		Attendance			Events
		first	*last*	*min.*	*max.*	*avg.*	
IoT 2020	DE	2017-05-16	2018-04-26	1	7	4,00	2
Meetup IoT / LoRa & LoRaWAN Ile-de-France	FR	2017-06-29	2017-06-29	41	41	41,00	1
The Things Network Luzern Community	CH	2017-05-09	2017-11-07	4	11	6,67	3
ioBroker Meetup Karlsruhe Smart Home IoT	DE	2017-07-11	2017-07-11	26	26	26,00	1
Cambridge IIoT User Group	GB	2017-09-14	2017-09-14	32	32	32,00	1
IoT Leeds	GB	2017-06-19	2018-10-22	30	60	39,11	9
IoT Meetup Milano	IT	2017-06-07	2017-10-02	22	30	26,66	3
Internet of Things Workshop Series Paris	FR	2017-05-31	2018-05-31	8	84	37,47	21
Internet of Things Workshop Series Berlin	DE	2017-06-01	2017-06-01	4	9	6,50	2
The Things Network Wroclaw Community	PL	2017-06-20	2017-06-20	7	7	7,00	1
Microsoft IoT & AI Lab Munich	DE	2017-07-14	2018-11-09	7	123	34,28	38
Internet of Things Workshop Series Torino	IT	2017-07-13	2018-07-06	10	11	10,50	2
IoT Workshop Series Düsseldorf	DE	2017-08-08	2017-10-10	46	73	59,50	2
Romanian Smart City Meetup	RO	2017-09-22	2017-10-19	3	18	14,00	4
Internet of Things Workshops Series Copenhagen	DK	2017-08-22	2017-10-09	46	75	64,00	3
IoT Tech Meetup Dortmund	DE	2017-11-21	2018-04-25	48	52	50,00	2
IOTA Beyond Blockchain London	GB	2017-11-01	2017-11-01	12	12	12,00	1
Workshops Series - Internet des Objets BAB	FR	2017-10-17	2017-10-17	12	12	12,00	1
IoT // Beyond the Hype	HU	2017-10-03	2018-01-23	7	32	17,00	3
Katowice IoT Meet	PL	2017-10-17	2018-10-18	32	57	44,50	2
Tech Meetup - Reading	GB	2018-08-14	2018-08-14	1	1	1,00	1
IoT Roma	IT	2017-11-23	2017-11-23	36	36	36,00	1
IoT Engineers London	GB	2017-11-01	2017-11-01	28	28	28,00	1
München IoT Meetup	DE	2017-11-02	2018-12-20	14	128	88,62	8

Table 22: **MeetUp groups collected**. Part VII.

Name	Country	Event date		Attendance			Events
		first	*last*	*min.*	*max.*	*avg.*	
IoT North Poland HuB	PL	2017-11-30	2018-11-26	5	10	6,75	4
TTN-Network Freiburg (LoRaWAN)	DE	2017-11-22	2018-10-10	2	23	11,67	12
Barcelona IoT Core – Internet of Things	ES	2018-05-30	2018-05-30	8	8	8,00	1
Drenthe Lora Meetup	NL	2017-11-27	2018-01-17	17	54	35,50	2
IoT Meetup Bodensee	DE	2018-01-30	2018-11-27	3	11	6,12	8
Bits & Beer - Berlin's IoT Developer Evening	DE	2017-12-13	2018-04-23	21	109	59,50	4
IOTA Meetup Germany	DE	2017-12-20	2018-03-01	10	148	71,00	3
GDG Cloud and IoT Lyon	FR	2018-01-11	2019-11-14	0	50	12,56	9
IOTA Application Meetup	FR	2018-11-15	2018-12-13	16	43	29,50	2
IOTA Chapter Luxembourg	LU	2018-07-05	2018-12-03	13	102	57,50	2
IoT and Robotics Build Group	DE	2018-02-22	2018-06-28	2	6	4,00	3
The Things Network Paderborn	DE	2018-02-12	2018-12-10	3	7	5,07	13
NB-IoT / Smart City Meetup	AT	2018-03-01	2018-03-01	9	9	9,00	1
Hardware Pioneers #Munich	DE	2018-04-26	2018-11-22	9	33	20,60	5
Cardiff Internet of Things (IoT) Meetup	GB	2018-04-12	2018-11-22	9	22	14,00	6
ConnHack - Shape the future of Connected Life	DE	2018-03-09	2018-03-09	14	14	14,00	1
Athens IOT Industrial Internet of Things Meetup	GR	2018-03-29	2018-06-28	42	64	53,00	2
Big Data - IoT	ES	2018-03-15	2018-04-26	7	24	16,00	4
The Things Network Asturias // #IoT #Makers	ES	2018-04-13	2018-04-13	10	10	10,00	1
IoT meets Data Science	ES	2018-06-06	2018-12-11	71	74	72,50	2
The Things Network Catalunya	ES	2018-04-21	2018-11-17	9	30	16,50	8
IOTA Meetup München	DE	2018-04-25	2018-11-22	9	104	46,20	5
IoT Pioneers	HU	2018-04-24	2018-09-25	65	160	112,50	2
NB-IoT, Meet and Greet Hub	DE	2018-05-09	2018-08-29	1	33	13,75	4
Garage Lab	RS	2018-05-30	2018-06-28	28	44	36,67	3
IoT Hessen (Kassel)	DE	2018-08-21	2018-08-21	22	22	22,00	1
Copenhagen Iot & Data Science Tech Talk run by Cogniance	DK	2018-06-14	2018-06-14	55	55	55,00	1
Meetup IOTBOX, Paris	FR	2018-06-05	2018-12-06	1	5	2,00	5

Table 23: **MeetUp groups collected**. Part VIII. List of the meetUp groups identifying as IoT communities of practice. For each group, we have indicated the official name of the group, the country where it develops its main activity, the date of the first and last event registered, the maximum, minimum and average attendance to its event and the total number of events (n).

References

[1] Muhammad Aurangzeb Ahmad, Zoheb Borbora, Jaideep Srivastava, and Noshir Contractor. Link Prediction Across Multiple Social Networks. In *2010 IEEE International Conference on Data Mining Workshops*, pages 911–918. IEEE, dec 2010.

[2] Omar Alonso, Michael Gertz, and Ricardo Baeza-Yates. On the Value of Temporal Information in Information Retrieval. *SIGIR Forum*, 41(2):35–41, dec 2007.

[3] Ricardo A Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[4] Michael J. Bannister, David Eppstein, Michael T. Goodrich, and Lowell Trott. Force-directed graph drawing using social gravity and scaling. In *Proceedings of the 20th International Conference on Graph Drawing*, GD'12, pages 414–425. Springer-Verlag, Berlin, Heidelberg, 2013.

[5] Albert-Laszlo Barabasi, Reka Albert, and A. Barabási. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, oct 1999.

[6] Vladimir Batagelj. Efficient Algorithms for Citation Network Analysis. *CoRR*, cs.DL/0309023, 2003.

[7] Vladimir Batagelj and Selena Praprotnik. An algebraic approach to temporal network analysis based on temporal quantities. *Social Network Analysis and Mining*, 6(1):1–28, 2016.

[8] Federico Battiston, Vincenzo Nicosia, and Vito Latora. Structural measures for multiplex networks. *Physical Review E*, 89(3):032804, 2014.

[9] Michele Berlingerio, Michele Coscia, and Fosca Giannotti. Finding and Characterizing Communities in Multidimensional Networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 490–494. IEEE, jul 2011.

[10] Michele Berlingerio, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. Multidimensional networks: foundations of structural analysis. *World Wide Web*, 16(5-6):567–593, oct 2012.

[11] Michele Berlingerio, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. Multidimensional networks: Foundations of structural analysis. *World Wide Web*, 16(5-6):567–593, oct 2013.

[12] Michele Berlingerio, Fabio Pinelli, and Francesco Calabrese. ABACUS: frequent pAttern mining-BAsed Community discovery in mUltidimensional networkS. *Data Mining and Knowledge Discovery*, 27(3):294–320, 2013.

[13] Ginestra Bianconi. Statistical mechanics of multiplex networks: Entropy and overlap. *Physical Review E*, 87(6):062806, jun 2013.

[14] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, mar 2003.

[15] Cecile Bothorel, Juan David Cruz, Juan David, and Barbora Micenkova. Clustering attributed graphs: models, measures and methods. *Network Science*, 3(3):408–444, 2015.

[16] Oualid Boutemine and Mohamed Bouguessa. Mining Community Structures in Multidimensional Networks. *ACM Transactions on Knowledge Discovery from Data*, 11(4):1–36, jun 2017.

[17] Ulrik Brandes, Natalie Indlekofer, and Martin Mader. Visualization methods for longitudinal social networks and stochastic actor-oriented modeling. *Social Networks*, 34(3):291–308, 2012.

[18] P. Brodka, A. Chmiel, M. Magnani, and G. Ragozini. Quantifying layer similarity in multiplex networks: a systematic study. *Royal Society open science*, 5(8), 2018.

[19] Matteo Brucato and Danilo Montesi. Metric Spaces for Temporal Information Retrieval. In Maarten de Rijke, Tom Kenter, Arjen P de Vries, ChengXiang Zhai, Franciska de Jong, Kira Radinsky, and Katja Hofmann, editors, *Advances in Information Retrieval*, pages 385–397, Cham, 2014. Springer International Publishing.

[20] Christoph Buchheim, Markus Chimani, Carsten Gutwenger, Michael Jünger, and Petra Mutzel. *Crossings and Planarization*. CRC Press, 2013.

[21] Carter T Butts. \pkg{network}: A Package for Managing Relational Data in \proglang{R}. *Journal of Statistical Software*, 24(2), 2008.

[22] Carter T Butts. *\pkg{network}: Classes for Relational Data*. The Statnet Project (\url{http://statnet.org}), 2015.

[23] Carter T Butts. *\pkg{sna}: Tools for Social Network Analysis*, 2016.

[24] Jonathan Chang and David M Blei. Relational Topic Models for Document Networks. In David A Van Dyk and Max Welling, editors, *AISTATS*, volume 5 of *JMLR Proceedings*, pages 81–88. JMLR.org, 2009.

[25] Jonathan Chang, Jordan Boyd-Graber, and David M Blei. Connections Between the Lines: Augmenting Social Networks with Text. In *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 169–178, New York, NY, USA, 2009. ACM.

[26] Justin Cheng, Lada A Adamic, Jon M Kleinberg, and Jure Leskovec. Do cascades recur? In *Proc. of the 25th international conference on world wide web*, pages 671–681. International WWW Conferences Steering Committee, 2016.

[27] Michele Coscia, Fosca Giannotti, and Dino Pedreschi. A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining*, 4(5):512–546, oct 2011.

[28] Gabor Csardi and Tamas Nepusz. The \pkg{igraph} Software Package for Complex Network Research. *InterJournal*, Complex Systems:1695, 2006.

[29] Manlio De Domenico, Vincenzo Nicosia, Alexandre Arenas, and Vito Latora. Structural reducibility of multilayer networks. *Nature communications*, 6:6864, 2015.

[30] Manlio De Domenico, Mason A. Porter, and Alex Arenas. MuxViz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015.

[31] Marina Diakonova, Vincenzo Nicosia, Vito Latora, and Maxi San Miguel. Irreducibility of multilayer network dynamics: the case of the voter model. Technical report, arXiv:1507.08940, 2015.

[32] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, September 2002.

[33] Mark E. Dickison, Matteo Magnani, and Luca Rossi. *Multilayer Social Networks*. Cambridge University Press, 2016.

[34] Jana Diesner and Kathleen M Carley. Revealing Social Structure from Texts: Meta-Matrix Text Analysis as a novel method for Network Text Analysis. In *Causal Mapping for Research in Information Technology*, pages 81–108, 2004.

[35] Peter Sheridan Dodds and Christopher M Danforth. Measuring the Happiness of Large-Scale Written Expression: Songs, Blogs, and Presidents. *Journal of Happiness Studies*, 11(4):441–456, aug 2010.

[36] Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering on Multi-Layer Graphs via Subspace Analysis on Grassmann Manifolds. *IEEE Transactions on Signal Processing*, 62(4):905–918, feb 2014.

[37] Tim Dwyer, Kim Marriott, Falk Schreiber, Peter Stuckey, Michael Woodward, and Michael Wybrow. Exploration of networks using overview+detail with constraint-based cooperative layout. *IEEE transactions on visualization and computer graphics*, 14(6):1293–300, January 2008.

[38] Dirk Eddelbuettel and Romain François. \pkg{Rcpp}: Seamless \proglang{R} and \proglang{C++} Integration. *Journal of Statistical Software*, 40(8):1–18, 2011.

[39] Daniel Edler, Ludvig Bohlin, and Martin Rosvall. Mapping higher-order network flows in memory and multilayer networks with Infomap. *CoRR*, abs/1706.04792, 2017.

[40] Paul Erdos and Alfréd Rényi. On the Evolution of Random Graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[41] Z. Fatemi, M. Magnani, and M. Salehi. A generalized force-directed layout for multiplex sociograms. In *Social Informatics - 10th International Conference*, 2018.

[42] Institute for Scientific Information, E Garfield, I H Sher, and R J Torpie. *The Use of Citation Data in Writing the History of Science*. Institute for Scientific Information, 1964.

[43] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[44] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, November 1991.

[45] Laetitia Gauvin, André Panisson, Ciro Cattuto, and Alain Barrat. Activity clocks: spreading dynamics on temporal networks of human contact. *Scientific reports*, 3:3099, jan 2013.

[46] Valerio Gemmetto and Diego Garlaschelli. Multiplexity versus correlation: the role of local constraints in real multiplexes. *Scientific reports*, 5, 2015.

[47] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring Networks of Diffusion and Influence. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 1019–1028, New York, NY, USA, 2010. ACM.

[48] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: {A} Survey. *CoRR*, abs/1705.0, 2017.

[49] Thilo Gross and Bernd Blasius. Adaptive coevolutionary networks: a review. *Journal of The Royal Society Interface*, 5:259–271, 2008.

[50] Mark S Handcock, David R Hunter, Carter T Butts, Steven M Goodreau, and Martina Morris. *\pkg{statnet}: Software Tools for the Statistical Modeling of Network Data*. Seattle, WA, 2003.

[51] Obaida Hanteer, Luca Rossi, Davide Vega D'Aurelio, and Matteo Magnani. From Interaction to Participation: The Role of the Imagined Audience in Social Media Community Detection and an Application to Political Communication on Twitter. In *ASONAM*, pages 531–534. {IEEE} Computer Society, 2018.

[52] Iina Hellsten and Loet Leydesdorff. Automated Analysis of Topic-Actor Networks on Twitter: New approach to the analysis of socio-semantic networks. *CoRR*, abs/1711.0, 2017.

[53] Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, oct 2012.

[54] Xiao Huang, Jundong Li, and Xia Hu. Label Informed Attributed Network Embedding. In *Proc. of the 10th ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 731–739, New York, NY, USA, 2017. ACM.

[55] Xiaodi Huang, Wei Lai, A.S.M. Sajeev, and Junbin Gao. A new algorithm for removing node overlapping in graph visualization. *Information Sciences*, 177(14):2821 – 2844, 2007.

[56] David R Hunter, Mark S Handcock, Carter T Butts, Steven M Goodreau, and Martina Morris. \pkg{ergm}: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3):1–29, 2008.

[57] Roberto Interdonato, Andrea Tagarelli, Dino Ienco, Arnaud Sallaberry, and Pascal Poncelet. Node-centric community detection in multilayer networks with layer-coverage diversification bias. apr 2017.

[58] Skye BenderâdeMoll James Moody, Daniel McFarland. Dynamic network visualization. *American Journal of Sociology*, 110(4):1206–1241, 2005.

[59] LUCAS G. S. JEUB, MICHAEL W. MAHONEY, PETER J. MUCHA, MASON A. PORTER, Zhana Kuncheva, Giovanni Montana, Emmanuel Lazega, M. Mihail, Mark Newman, Stanley Wasserman, and Katherine Faust. A local perspective on community structure in multilayer networks. *Network Science*, pages 1–20, jan 2017.

[60] Inderjit S Jutla, Lucas G S Jeub, and Peter J Mucha. A generalized Louvain method for community detection implemented in Matlab. Technical report.

[61] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, April 1989.

[62] Nattiya Kanhabua, Roi Blanco, and Kjetil Nørvåg. Temporal Information Retrieval. *Found. Trends Inf. Retr.*, 9(2):91–208, 2015.

[63] Jinseok Kim and Jana Diesner. Over-time measurement of triadic closure in coauthorship networks. *Social Network Analysis and Mining*, 7(1):9, mar 2017.

[64] Jung Yeol Kim and K.-I. Goh. Coevolution and Correlated Multiplexity in Multiplex Networks. *Physical Review Letters*, 111(5):58702, 2013.

[65] Jungeun Kim, Jae-gil Lee, and Sungsu Lim. Differential Flattening: A Novel Framework for Community Detection in Multi-Layer Graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):27, 2016.

[66] Jungeun Kim and Jae-Gil J.-G. Lee. Community detection in multi-layer graphs: A survey. *SIGMOD Record*, 44(3):37–48, dec 2015.

[67] Daichi Kimura and Yoshinori Hayakawa. Coevolutionary networks with homophily and heterophily. *Phys. Rev. E*, 78(1):16103, 2008.

[68] Mikko Kivelä, Alexandre Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, Mason A. Porter, Mikko Kivela, Alexandre Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, Mason A. Porter, Mikko Kivelä, Alexandre Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer Networks. *Journal of Complex Networks*, 2(3):203–271, sep 2014.

[69] Yehuda Koren, L. Carmel, and D. Harel. Ace: a fast multiscale eigenvectors computation for drawing huge graphs. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 137–144, 2002.

[70] Evangelos Kotsakis. Structured Information Retrieval in XML Documents. In *Proc. of the 2002 ACM Symposium on Applied Computing*, SAC '02, pages 663–667, New York, NY, USA, 2002. ACM.

[71] Jan Kralj, Anita Valmarska, Miha Grčar, Marko Robnik-Šikonja, and Nada Lavrač. Analysis of Text-Enriched Heterogeneous Information Networks. In Nathalie Japkowicz and Jerzy Stefanowski, editors, *Big Data Analysis: New Algorithms for a New Society*, pages 115–139. Springer International Publishing, Cham, 2016.

[72] Pushpa Kumar and Kang Zhang. Visualization of clustered directed acyclic graphs with node interleaving. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1800–1805, New York, NY, USA, 2009. ACM.

[73] Zhana Kuncheva and Giovanni Montana. Community Detection in Multiplex Networks using Locally Adaptive Random Walks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015 - ASONAM '15*, pages 1308–1315, New York, New York, USA, 2015. ACM Press.

[74] Maciej Kurant and Patrick Thiran. Layered Complex Networks. *Physical Review Letters*, 96(13):138701, apr 2006.

[75] Renaud Lambiotte, Vsevolod Salnikov, and Martin Rosvall. Effect of memory on the dynamics of random walks on networks. *Journal of Complex Networks*, 3(2):177–188, 2015.

[76] Renaud Lambiotte, Lionel Tabourier, and Jean-Charles Delvenne. Burstiness and spreading on temporal networks. *The European Physical Journal B*, 86(7):320, jul 2013.

[77] Victor Lavrenko, Matt Schmill, Dawn Lawrie, Paul Ogilvie, David Jensen, and James Allan. Mining of Concurrent Text and Time Series. In *SIGKDD workshop on text mining*, pages 37–44, 2000.

[78] Jay Lee, Manzil Zaheer, Stephan Günnemann, and Alex Smola. Preferential Attachment in Graphs with Affinities. In Guy Lebanon and S V N Vishwanathan, editors, *Proc. of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proc. of Machine Learning Research*, pages 571–580, San Diego, California, USA, 2015.

[79] Hartmut H K Lentz, Thomas Selhorst, and Igor M Sokolov. Unfolding Accessibility Provides a Macroscopic Approach to Temporal Networks. *Phys. Rev.*, 110(11):118701–118706, 2013.

[80] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. *International conference on Knowledge Discovery and Data Mining (KDD)*, page 420, 2007.

[81] Ding Ma. Visualization of social media data: mapping changing social netwroks . Master's thesis, the Faculty of Geo-Information Science and Earth Observation of the University of Twent, 2012.

[82] Matteo Magnani, Danilo Montesi, and Luca Rossi. Conversation Retrieval from Microblogging Sites. *Information Retrieval Journal*, 15(3-4), 2012.

[83] Matteo Magnani and Luca Rossi. The ML-Model for Multi-layer Social Networks. In *ASONAM*, pages 5–12. IEEE Computer Society, 2011.

[84] Matteo Magnani and Luca Rossi. Formation of multiple networks. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 257–264. Springer Berlin Heidelberg, 2013.

[85] Matteo Magnani and Luca Rossi. Pareto Distance for Multi-layer Network Analysis. In Ariel M. Greenberg, William G. Kennedy, and Nathan D. Bos, editors, *Social Computing, Behavioral-Cultural Modeling and Prediction*, volume 7812 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[86] Filippo Menczer. Evolution of document networks. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5261–5265, 2004.

[87] J L Moreno and Helen Hall. Jennings. *Who shall survive? : a new approach to the problem of human interrelations / by J. L. Moreno*. Nervous and Mental Disease Publishing Co., Washington, D. C. :, 1934.

[88] Peter J Mucha and Mason A Porter. Communities in multislice voting networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(4), 2010.

[89] Mark E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[90] Vincenzo Nicosia, Ginestra Bianconi, Vito Latora, and Marc Barthelemy. Growing multiplex networks. *arXiv preprint arXiv:1302.7126*, 2013.

[91] Vincenzo Nicosia and Vito Latora. Measuring and modeling correlations in multiplex networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 92(3):1–20, 2015.

[92] Rogier Noldus and Piet Van Mieghem. Assortativity in complex networks. *Journal of Complex Networks*, 3(4):507–542, 2015.

[93] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge, and Noah A Smith. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In William W Cohen and Samuel Gosling, editors, *Proc. of the Eleventh International Conference on Web and Social Media*. The AAAI Press.

[94] Antonio Rivero Ostoic. *\pkg{multiplex}: Algebraic Tools for the Analysis of Multiple Social Networks*, 2018.

[95] John F Padgett and Paul D McLean. Organizational Invention and Elite Transformation: The Birth of Partnership Systems in Renaissance Florence. *American Journal of Sociology*, 111(5):pp. 1463–1568, 2006.

[96] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[97] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in Temporal Networks. In *Proc. of the 10th ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 601–610, New York, NY, USA, 2017. ACM.

[98] Tiago P Peixoto and Martin Rosvall. Modelling sequences and temporal networks with dynamic community structures. *Nature communications*, 8(1):582–594, 2017.

[99] Denis Redondo, Arnaud Sallaberry, Dino Ienco, Faraz Zaidi, and Pascal Poncelet. Layer-Centered Approach for Multigraphs Visualization. In *International Conference on Information Visualisation (iV)*, pages 50–55, 2015.

[100] Xiang Ren, Ahmed El-Kishky, Chi Wang, and Jiawei Han. Automatic Entity Recognition and Typing in Massive Text Corpora. In *Proc. of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 1025–1028, Geneva, Switzerland, 2016. International WWW Conferences Steering Committee.

[101] Xiang Ren, Yuanhua Lv, Kuansan Wang, and Jiawei Han. Comparative Document Analysis for Large Text Corpora. In *Proc. of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 325–334, New York, NY, USA, 2017. ACM.

[102] B. Renoust, G. Melançon, and T. Munzner. Detangler: Visual analytics for multiplex networks. *Computer Graphics Forum*, 34(3):321–330, 2015.

[103] Ruth M Ripley, Tom A B Snijders, Zsofia Bóda, András Vörös, and Paulina Preciado. Manual for \pkg{Siena} version 4.0. Technical report, Oxford: University of Oxford, Department of Statistics; Nuffield College, 2018.

[104] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The Author-topic Model for Authors and Documents. In *Proc. of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 487–494, Arlington, Virginia, United States, 2004. AUAI Press.

[105] Giulio Rossetti, Michele Berlingerio, and Fosca Giannotti. Scalable Link Prediction on Multidimensional Networks. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 979–986. IEEE, dec 2011.

[106] Luca Rossi and Matteo Magnani. Towards effective visual analytics on multiplex and multilayer networks. *Chaos, Solitons and Fractals*, 72:68–76, 2015.

[107] Martin Rosvall, Alcides V Esquivel, Andrea Lancichinetti, Jevin D West, and Renaud Lambiotte. Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications*, 5, 2014.

[108] Camille Roth. Knowledge Communities and Socio-Cognitive Taxonomies. In Rokia Missaoui, Sergei O Kuznetsov, and Sergei Obiedkov, editors, *Formal Concept Analysis of Social Networks*, pages 1–18. Springer International Publishing, Cham, 2017.

[109] Camille Roth and Jean-Philippe Cointet. Social and semantic coevolution in knowledge networks. *Social Networks*, 32(1):16–29, 2010.

[110] Mostafa Salehi, Rajesh Sharma, Moreno Marzolla, Matteo Magnani, Payam Siyari, and Danilo Montesi. Spreading processes in Multilayer Networks. *IEEE Transactions on Network Science and Engineering*, 2(2):65 – 83, 2015.

[111] Marcus Schaefer. The graph crossing number and its variants: a survey. *The electronic journal of combinatorics*, 1000:DS21–May, 2013.

[112] Ingo Scholtes. When is a Network a Network?: Multi-Order Graphical Model Selection in Pathways and Temporal Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 1 of *KDD 2017*, pages 1037–1046. ACM, 2011.

[113] Ingo Scholtes, Nicolas Wider, René Pfitzner, Antonios Garas, Claudio J Tessone, and Frank Schweitzer. Causality-driven slow-down and speed-up of diffusion in non-Markovian temporal networks. *Nature communications*, 5(1):5024–5033, 2014.

[114] A Shabbeer, C Ozcaglar, M Gonzalez, and KP Bennett. Optimal embedding of heterogeneous graph data with edge crossing constraints. In *Presented at NIPS Workshop on Challenges of Data Visualization*, page 1, 2010.

[115] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2017.

[116] Tom A B Snijders. Models for Longitudinal Network Data. In Peter J Carrington, John Scott, and StanleyEditors Wasserman, editors, *Models and Methods in Social Network Analysis*, Structural Analysis in the Social Sciences, pages 215–247. Cambridge University Press, 2005.

[117] Tom A B Snijders. Siena: Statistical Modeling of Longitudinal Network Data. In Reda Alhajj and Jon Rokne, editors, *Encyclopedia of Social Network Analysis and Mining*, pages 1718–1725. Springer New York, New York, NY, 2014.

[118] Ricard V Sole, Bernat Corominas Murtra, Sergi Valverde, and Luc Steels. Language networks: Their structure, function, and evolution. *Complexity*, 15(6):20–26, 2010.

[119] John F Sowa. *Principles of semantic networks: Explorations in the representation of knowledge*. Morgan Kaufmann, 2014.

[120] Andrea Tagarelli, Alessia Amelio, and Francesco Gullo. Ensemble-based community detection in multilayer networks. *Data Mining and Knowledge Discovery*, 31(5):1506–1543, sep 2017.

[121] L. Tamine, L. Soulier, L.B. Jabeur, F. Amblard, C. Hanachi, G. Hubert, and C. Roth. Social media-based collaborative information access: Analysis of online crisis-related twitter conversations. In *HT 2016 - Proceedings of the 27th ACM Conference on Hypertext and Social Media*, pages 159–168, 2016.

[122] Lei Tang, Xufei Wang, and Huan Liu. Uncoverning Groups via Heterogeneous Interaction Analysis. *2009 Ninth IEEE International Conference on Data Mining*, pages 503–512, 2009.

[123] Lei Tang, Xufei Wang, and Huan Liu. Community detection via heterogeneous interaction analysis. *Data Mining and Knowledge Discovery*, 25(1):1–33, jul 2011.

[124] Nazanin Afsarmanesh Tehrani, Matteo Magnani, N. Afsarmanesh, and Matteo Magnani. Partial and overlapping community detection in multiplex social networks. In *Social Informatics*, volume 11186 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2018.

[125] Davide Vega and Matteo Magnani. Foundations of Temporal Text Networks. *Applied Network Science*, 3(1):25:1—–25:26, 2018.

[126] Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609(1):245–252, 2016.

[127] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, and D.W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011.

[128] Chenguang Wang, Yangqiu Song, Haoran Li, Yizhou Sun, Ming Zhang, and Jiawei Han. Distant Meta-Path Similarities for Text-Based Heterogeneous Information Networks. In *Proc. of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1629–1638, New York, NY, USA, 2017. ACM.

[129] Howard D White and Belver C Griffith. Author cocitation: A literature measure of intellectual structure. *Journal of the American Society for Information Science*, 32(3):163–171, 1981.