ENS DE LYON

LIRIS

Internship Report of Master 2 in Computer Sciences

Community detection in fine-grained dynamical networks

Author:
Fabrice LÉCUYER

Supervisor:
Rémy CAZABET

January to June 2020, Lyon

Contents

Introduction

This five months internship was part of the second year of master in Computer Sciences at ENS Lyon. It took place in Claude-Bernard University, in the LIRIS lab. My supervisor, Rémy Cazabet, was previously my teacher in Complex Networks.

This period was troubled by pandemic risks and confinement measures, which forced everyone to work remotely. While this is a frustrating and quite demotivating situation, it went relatively smoothly thanks to good communication within the team.

The first part of the internship consisted in assimilating state-of-the-art concepts in network science and community detection. The conclusions of this bibliography are presented in section 1.1 that defines the context of static and dynamic networks based on real-world data; section 1.2 which presents the how and the why of communities in networks; and section 1.3, a selected overview of static and dynamic community detection algorithms.

In order to understand the interest of link streams over other types of temporal graphs, some datasets have been closely observed. Section 2 presents these objects of study, with all the programming tools that were necessarily involved. It also presents bigger datasets that could be targeted with efficient and scalable algorithms.

Finally, section 3 presents a comparison framework between algorithms of the literature and attempts of new methods. Naive network aggregation is studied in 3.1; a possibility to transform link streams into static time-graphs is analysed in 3.2; random walks are used in 3.3 for local detection.

The last section references possible extensions of the work, as well as some methodological and practical mistakes that should be avoided in the future.

1 Context

1.1 Static and dynamic networks

1.1.1 Graphs and networks

Graph theory is a mathematical field that aims to study a set of vertices connected by edges. Various properties can be computed, such as colouring numbers, shortest paths, minimal cuts, etc. Graphs are a very abstract and general concept. When a graph is used to describe real-world situations, we usually call it a network, made of nodes and links.

Network science can therefore be seen as an applied graph theory that works on various data sources. Questions about international trade, traffic jam, scientific collaboration, online or physical social networks, or synaptic connections in the brain, can be translated to the vocabulary of networks.

Interestingly, real-world networks show properties that are not expected in the usual random graph definition: in the Erdős–Rényi model[4], a fraction of possible edges appear at random, leading to a bell curve distribution of degrees. On the contrary, networks from different fields show "fat tail" distributions that often follow a power law[3, 25]. This creates a scalle-free behaviour, with hubs of arbitrarily high degree[8].

Other studies have highlighted the small-world property in networks, meaning that the average length between any nodes is quite small despite the local structures[6]. This causes networks to be resilient to random failure, which is crucial in the web: though many servers shut down (because of disconnection or malfunction), there is always a way to reach anyone in the network.

Mathematical investigations can use graphs of any size, from paths in the "Seven Bridges of Königsberg" to asymptotic behaviour in random graphs. However, networks rely on real data collection. Our collective ability to gather and organise it makes increasingly massive networks available: online social networks can track millions of users and their billions of in-

teractions, web crawlers have indexed trillions of web pages, and the 10^{14} synaptic connections of our brain could be mapped entirely over the next years.

1.1.2 Dynamic representations

Moreover, it is often possible to timestamp data points, thus adding a temporal dimension. This information seems necessary to account for interactions between individuals or anything linked to the physical world. Studying this shift from networks to dynamic networks requires a modification of the mathematical setting, from graphs to temporal graphs.

Here the notation for a static graph is $G = (V, E)$, with V a set of n vertices and E a set of m edges, that may have a weight ω and a direction. Numerous mathematical objects have been proposed to deal with dynamic networks[39].

The most natural approach is to use N *snapshots* $G = (G_1, \dots G_N)$, each of them representing an aggregation of the interactions on a given time span. Alternatively, an *interval graph* defines its edges over time intervals. It is useful to describe long-term relationships that can appear or disappear, such as "friendships" in online social platforms. However, it does not allow for a high temporal resolution nor for instantaneous interactions such as sending an email.

Therefore, the concept of *links streams* has been introduced[43]. As shown in figure 1, it is a triplet $L = (T, V, E)$, where T is a (discrete or continuous) set of times, V is a set of vertices, and E is a set of edges. The more general notion of *stream graph* allows for nodes to exist at some specific dates, and for edges to have a duration, but here we simply consider instantaneous interactions: $E \subseteq T \times V \times V$.

As opposed to snapshots and interval graphs, where the network evolves slowly so that each time step presents a fully consistent network, a link stream may have only a few links per time step. Basic concepts such as degree, path or clustering coefficient had to be redefined[48].

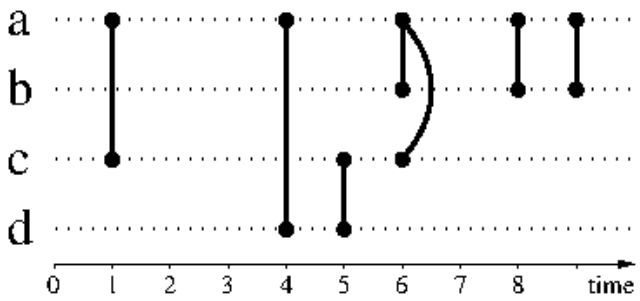

Figure 1: Example of link stream

Yet this fine-grained representation is not necessarily lossless as all link duration is lost. In discrete time scales, this problem can be tackled by repeating links when interactions last longer.

1.2 Community detection

1.2.1 Definitions of community

In complex systems involving multiple agents, the emergence of bigger scale phenomena relies on topological structures of the network. In real-world systems, individuals tend to form communities within which they interact preferably. It is clear intuitively what a community can be: a group of friends, countries involved in a free-trade agreement, proteins responsible for a biological function, web pages of the same site...

It is much harder however to give a formal definition of communities[26]. First, it is tempting to classify nodes in a single community each to obtain a partition of the network. But data shows that real communities overlap significantly[14], and they have an inner hierarchy, with some nodes belonging with more strength than others. Think of a group of friends in which some people are always keen on meeting while some barely answer to invitations.

Some definitions are local: they compare the inner topology of a community to its neighbourhood without looking into the whole graph. It is usually required from a community to be dense: strictest definitions will only select cliques (a set of completely interconnected nodes), but they can be relaxed. Various factors involving node internal degrees, clustering coefficients, diame-

ter, etc, can be used. As a dense subset is not a community if the whole network is dense, this cohesion is often benchmarked against external connectivity to compute the quality of a cluster.

Some definitions are global: they compare the density of a subgraph to an expected density, inferred from a reference model such as Erdős–Rényi or configuration. The problem is that modifications in one area of the network may affect other areas.

Finally, some definitions gather similar nodes into community. This homophily[11] can be based on a distance over node properties (such as age and geographical location), on the rate of shared neighbours, on common attractor of random walks[44], etc. It is also possible to detect communities with an intuitive procedure rather than a mathematical definition.

1.2.2 Quality functions

Mathematical definitions often go with a quantitative evaluation for the quality of communities. Their aim is to state whether a given subgraph is a "good" community. Unfortunately, there is no obvious formula and the literature keep producing new options[45]. Some metrics of importance are detailed hereafter.

In a graph of n vertices and m edges, let us consider a community C of c nodes and d edges, with d_{in} internal and d_{out} external edges.

Internal Density $=\frac{d_{in}}{\binom{c}{2}}$ is the number of inner edges out of the maximal possible number, Expansion$=\frac{d_{out}}{c}$ is the average number of outgoing edge per node, Out-Degree Fraction$=\frac{1}{c}\sum_{u\in C}\frac{d_{out}(u)}{d(u)}$, Significance=probability that C appears in a random graph. The clustering coefficient counts the rate of active triangles in C. Other metrics such as Permanence, Separability, Normalised Cut mix the same variables in different equations to obtain a fitness value.

All these definitions are intuitively relevant, but their number is overwhelming; some of them have been used in many papers and became standards even though they have some flaws. In this internship, the following metrics appear in algorithms comparison.

Conductance represents the proportion of outgoing edges. It describes the connectivity of a community by assessing the time required for a random walk process to reach its stability[10]. The name comes from an analogy with electrical networks[18].

$$\mathcal{C}(C) = \frac{d_{out}}{2d_{in} + d_{out}}$$

Surprise compares the internal edge rate $q = \frac{d_{in}}{d}$ to the maximal rate $\hat{q} = \frac{\binom{c}{2}}{\binom{n}{2}} \simeq \frac{c^2}{n^2}$. The asymptotic definition[40] is $\mathcal{S} = d \cdot D(q \mid\mid \hat{q})$, where D is the Kullback–Leibler divergence, used to compare probability distributions.

Modularity[17] compares the density of a subgraph to its expected density in the configuration model, a random process in which edges are rewired at random but nodes keep the same degree. It is an extensive property, which means that in a partition the network's modularity is the sum of communities' modularity.

$$\mathcal{Q}(C) = \frac{1}{2m}\sum_{u,v\in C}\left(1 - \frac{d_i d_j}{2m}\right)$$

This last metric is the most widely accepted, and the famous Louvain algorithm aims at optimising it. However, it has major drawbacks that are well documented as well. First, its maximum is **degenerate**[27]: a variety of completely different partitions give such close modularities that heuristics cannot distinguish them. This explains why many procedures output decent approximations with high structural variability. Second, modularity compares partitions of any size to the whole network. This establishes a **resolution limit**[19], ie a size depending on n and m under which no community can be found. The toy counter-example is a ring of cliques, in which modularity optimisation tends to merge cliques together instead of detecting them as separate objects.

Various patches were proposed such as modified/Erdős–Rényi/Z/adaptative modularity, modularity density/intensity. In particular,

it is common to use a resolution parameter to choose the size of smallest communities, but it does not solve degeneracy issues.

1.3 Overview of relevant algorithms

1.3.1 On static graphs

Community detection is a quite recent field of complex networks, but it has yielded a great variety of algorithms[26, 33]. Due to the several definitions of community and the multiple quality functions seen in 1.2, the methods can lead to very different results. More remarkably, they leverage various mathematical concepts.

Modularity optimisation consists in merging or refining communities while trying to improve the global score $\mathcal{Q}$. The number of distinct partitions in a discrete set is given by the Bell number, which grows faster than exponential: it is absolutely not possible to optimise it with an exhaustive search. Heuristics such as simulated annealing[15], spectral clustering[12], genetic algorithms, etc, are employed. The most widely used techniques rely on greedy approaches: each node starts in its own community, and neighbours are progressively regrouped. They create a dendrogram (a tree that represents the merging process) from which the partition with highest modularity can be retrieved.

The most famous algorithm belongs to this category: named after the **Louvain**[23] university, it moves each node to the neighbouring community that would benefit the most from its presence. After convergence, a new graph is created in which former communities become nodes. The process is repeated until all nodes are in a unique community, and the step with highest global modularity is returned.

On top of the drawbacks inherent in the modularity, this algorithm can lead to sparsely connected or even disconnected communities, which is really counter intuitive. It happens when a node bridging to areas of a same community switches to an other community of its neighbourhood; the two areas are disconnected and may never reconnect. The university of **Leiden**[53] proposed a slight modification that allows for community split. They also accelerate the process by queuing up possible switchers, instead of visiting all the nodes every time.

Other techniques use notions of topological closeness. Following the idea of Pagerank[7], they use random walks or diffusion to define communities. **Walktrap**[16] defines a probability $P_{u,v}$ to reach node v starting from u; nodes of similar outreach are merged progressively, forming a dendrogram; the step with best modularity (or else) is returned. It uses short random walks, a few jumps being enough to cover most of the network (since they usually have small diameters[9]), which leads to low time complexity and avoids reaching stationary distribution.

Random walks are also used in **Infomap**[24] as a preprocess, before applying a Minimum Description Length: nodes are grouped in communities so that walks can be described on two levels (inter- and intra-community jumps, like a separation between cities and street names). The partition allowing for shortest average descriptions (with a Huffman code[2]) is selected using greedy search or simulated annealing. An advantage of MDL procedures is that they require no parameter[13] and have a clear mathematical definition.

The **Label propagation**[20] paradigm consists in diffusing each node's label to its neighbours; nodes then switch to the community where most of their neighbours belong, until it converges. Finally, statistical inference is used in Stochastic Block Models[54], where nodes are partitioned into a given number of communities with predefined inner and outer density rates.

1.3.2 On temporal graphs

The previous sections show the abundance of definitions, algorithms, and evaluations for static communities. Since there are also several types of temporal graphs, we can imagine the plethora of possibilities for dynamic community

detection[52][1].

Considering snapshots, it is possible to extend previous algorithms in different ways. One solution is to run a static algorithm on each snapshot, then to try and match communities that have similar nodes at consecutive times. A simple way[28] is to match communities C_1 and C_2 when their Jaccard index exceeds a threshold Θ:

$$\mathcal{J}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} > \Theta$$

An issue is that the high variability of static methods will make dynamic partitions quite unstable: even if every snapshot contains the same network, these techniques will find varying communities. This can be avoided to some extent by core-node based methods[29]: a set of nodes is distinguished (with centrality or betweenness criteria for instance) and communities are built around them in each snapshot. Following core-nodes allows to define community events such as merge/split and grow/shrink[21].

An other approach is to turn the degeneracy problem into an asset: since many local optima have an almost-maximal modularity, authors[35, 41] had the idea to detect static communities in the first snapshot and use them as starting points in the next one. The complexity is very low and partitions are smooth in time.

Otherwise, a multi-objective optimisation can be lead, balancing instantaneous quality and temporal smoothness[30]. Extensions of Stochastic Block Models[46, 47] and Label propagation[36] have been proposed for snapshots and interval graphs, as well as modularity optimisation on a time-space graph (snapshots connected by varying edges)[31].

As for link streams, the literature is smaller because its definition is more recent, and it is rare to collect data with fine-grained temporal information. Some methods try to detect communities of different scales, either using a Poisson process alongside Stochastic Block Model[49], or a detection of core nodes[55], with time-space extension and pruning.

With a given time scale, it is also possible to set a Gaussian weight around each link so that time-nodes are bound relevantly[42]. Similarly, each link can become a node connected to other links with which it shares an extremity at some time[50]. In both cases, a conventional modularity optimisation is applied on the resulting graph.

2 Libraries and datasets

2.1 Libraries, modules, software

As I mostly use Python for its simplicity and clarity, it was possible to choose between two modules: `networkx`[2] or `igraph`[3]. I chose the first one, which provides simple ways to create and import networks. It has many built-in properties, such as weighted edges, node centrality measures, colouring, shortest paths, minimal cut, and even a couple of community detection algorithms.

However for this specific task it was more interesting to use `CDlib`[4], where CD stands for... community detection. It extends `networkx` by

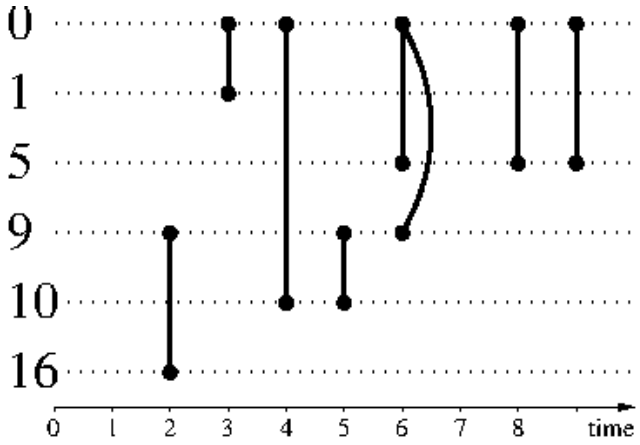

Figure 2: Link stream sample of the school dataset using `streamfig`. Rows are individuals and columns are 20s time frames.

[1] Live version of the survey: `http://cazabetremy.fr/rRessources/DCDsurvey.html`

[2] `https://networkx.github.io/documentation/`

[3] `https://igraph.org/python/doc`

[4] `https://cdlib.readthedocs.io`

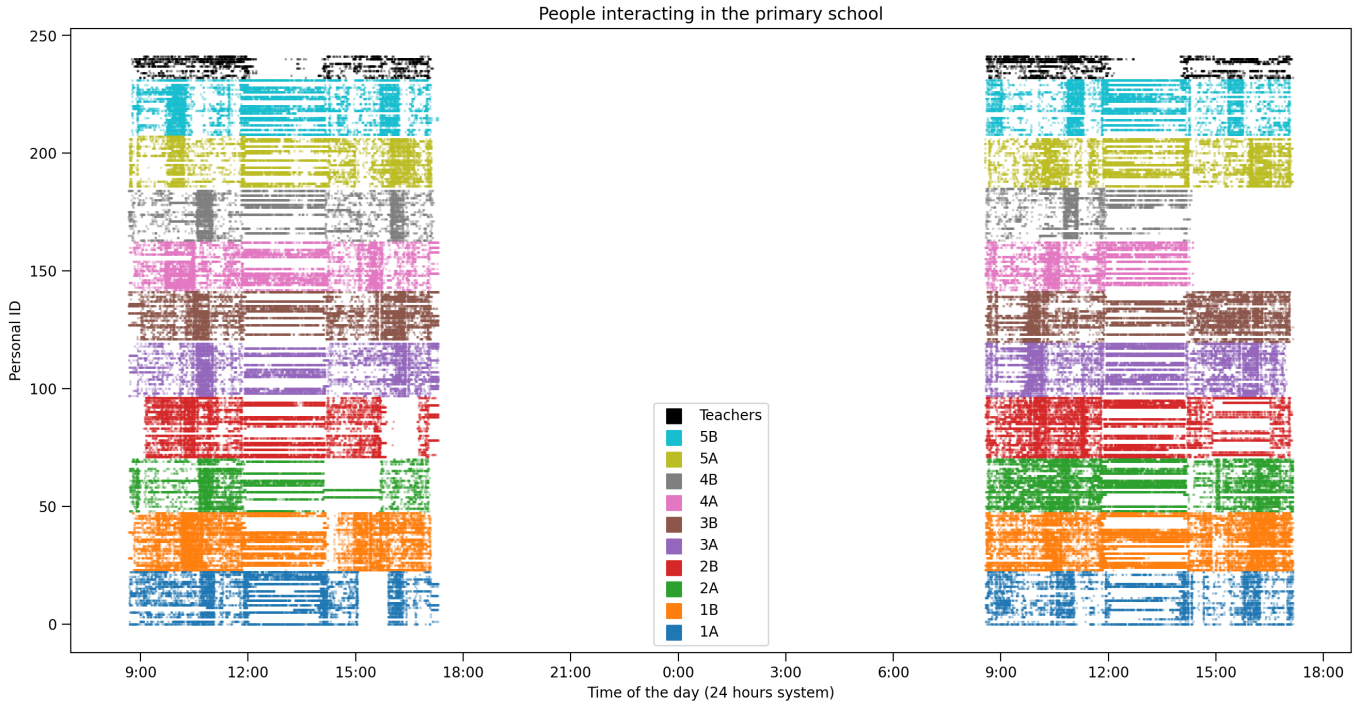

Figure 3: Two days in a primary school, showing individuals when they interact. Sensors are deactivated out of school and during sports sessions, which explains why some classes completely disappear in the afternoon; playground breaks during morning and afternoon spans display much denser interactions; lunch break has a specific pattern, with teachers and some students going home.

implementing various data-structures for node and edge clustering, as well as over 40 detection algorithms, among which those mentioned above. The module act as an interface between Python and external code provided by researchers. Dozens of fitness functions and partition comparisons are also available. Plus, it is designed to help researchers and developers running tests with varying parameters, and it can display scores and adjacency matrices.

As for nice plots, the software `gephi`[5] is very useful though not always convenient. Louvain algorithm is built in and can serve to colour nodes depending on their community. Edges and vertices can be styled in accordance to their properties, and different automatic layouts can be used. In the case of link streams, the `streamfig`[6] draws lovely plots that help describing toy examples, like in figure 2.

The library `tnetwork`[7], under construction, is dedicated to temporal networks. It also relies on `networkx` and introduces data-structures for snapshots and interval graphs, as well as state-of-the-art algorithms for community detection. It provides visualisation tools and not-yet-published evaluations of dynamic partitions. Moreover, it is able to read Sociopatterns link stream files directly (see below).

To better understand the different processes involved in my experiments, I designed a small additional module `tgraph` that could be plugged into `tnetwork` (in a quite redundant way). It turns a list of links into a structure of temporal adjacency lists. Nodes indices and timestamps are normalised on the intervals $[1, n]$ and $[1, T]$, the groundtruth (given by Sociopatterns for instance) is saved, and various tools make detection algorithms simpler to write: snapshot between given dates, neighbours of u in a given time window, time-space graph, uniform random sample of edges...

[5] `https://gephi.org`
[6] `https://github.com/TiphaineV/streamfig`
[7] `https://tnetwork.readthedocs.io`

2.2 Datasets

2.2.1 Sociopatterns

Sociopatterns[8] is a database of a particular kind, created by complex systems researchers. In various contexts, they equipped a group of people with RFID sensors in order to measure their proximity in real-time. A list of face-to-face interactions is obtained, with a time resolution of a few seconds. It is interesting to note that this was specifically designed to study networks of interactions, while other datasets are normally obtained and analysed in retrospect.

One example is a primary school[34], in which the 230 pupils and 10 teachers had sensors during two consecutive days. Figure 3 shows which individuals interact at which time, with colours given by the groundtruth. Funnily enough, this data was the basis for a study about school closures during epidemics[38].

This dataset was chosen here as the main benchmark to test algorithms because it is relatively intuitive: we can imagine children interacting by small groups inside their class during lessons, and then actively mingling during breaks. Some studies cut this dataset to analyse a single day, but it is also interesting to see how community detection algorithms behave when they face the long empty gap of night.

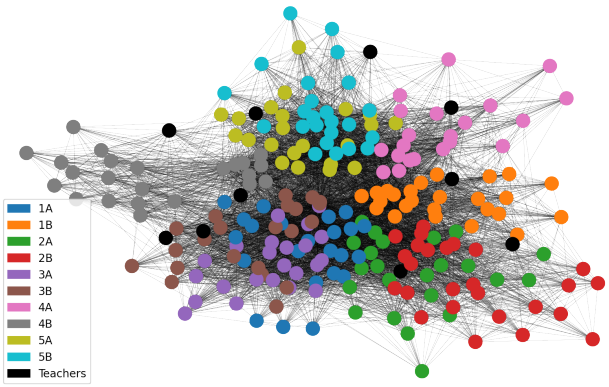

Figure 4: Aggregated school graph. Weighted edges are obtained by counting interactions over 2 days. Colours represent groundtruth categories.

[8] `http://www.sociopatterns.org/`

The data is given in tuples (t, u_1, u_2, c_1, c_2), each meaning that individuals u_1 of class c_1 interacted with u_2 of class c_2 in the time span $]t - 20s, t]$. The link stream representation is perfectly suited for that, and a sample is shown in figure 2. When all interactions between individuals are merged, we obtain a static graph figure 4. One question is whether classes could be detected from the data. It is likely, since the aggregate seems to cluster pupils of the same class together, with one teacher (black dot) who is probably theirs.

The size of the data is 242 nodes, 125'000 links, 3100 active time frames ranging over 32 hours. A **tnode** is a node at a given time, corresponding to an individual interacting for 20 seconds; there are about 175'000 tnodes, which means that interactions are not always one-to-one (think of a teacher facing the first raw of students).

An other interesting dataset was collected in a hospital[37]. There, people have different roles such as nurse, doctor or patient, so the groundtruth does not correspond to highly interactive groups. Yet a homophily-based approach may be able to recover the roles. While it is possible to imagine that hubs would be nurses who come and go between doctors and patients, it is unclear what communities should represent; maybe different wings of the hospital, or medical services.

Other datasets of the same type exist, collected in public exhibitions, rural households, conferences, workplaces, etc. Some of them include extra information such as demographics or friendship assertions.

2.2.2 Synthetic scenarios

On top of the descriptive tools it provides, the `tnetwork` library has functions to generate dynamic graphs following a given scenario. It takes as input a number of nodes and a number of communities; the latter can run for a given time, grow and shrink. They can be followed through splits and merges thanks to a label, or even vanish and reappear in a cyclic way.

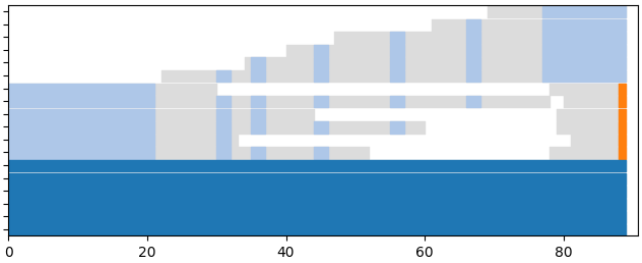

Figure 5: Ship of Theseus story represented in `tnetwork`: members of a community progressively quit and reconnect separately. Which final group corresponds to the original one?

Manual definition of scenarios is useful to create simple cases on which we want to test the behaviour of our algorithms. Moreover, it can confront them to tricky situations, like the ship of Theseus and its corresponding philosophical question: all parts of an object are progressively replaced by new ones and reassembled somewhere else, which of the two remaining objects is the original one? The library renders this situation as shown in figure 5.

Besides, it is possible to generate random scenarios, with parameters such as the number of events, the maximal size of clusters, or the inter-community edge density. One scenario is shown in figure 6. Generating several random instances allows to confront algorithms to various unpredictable situations.

2.3 Is this big data?

The datasets presented above range from tens of nodes to hundreds of thousands of links. If compared to what a human being can process, this is huge and very challenging. However, it is not much for a personal computer: the temporal graph for school interactions amounts for less than 3Mb. Machines are able to produce and to deal with much bigger sets, some of which can be interpreted as link streams.

Consider a list of emails metadata: sender, recipient, timestamp. All interactions are instantaneous, nodes represent email addresses that can belong to communities, and dates are important to see how they evolve. Such datasets can be collected for instance thanks to public mailing lists[22]. Their potential size is tremendous, as billions of emails are sent every day, by either people or machines.

Other interesting datasets come from cryptocurrencies: to guarantee the truthfulness of transactions, some of them store all the record in a public file that no one can change without collective agreement. The DM2L team (in which I worked during this internship) has 175Gb of Bitcoin data; parsing this gigantic file gives access to 150 millions of timestamped transactions. Users cannot be directly identified, but a wallet can be tracked to draw a graph of money flows. In this context, communities may be several address of one user, individuals paying each other regularly, currency-management firms, trading platforms, etc.

Online social networks offer countless instances of data that can fit in a stream graph: messages (similar to emails), reactions on one's shared content, co-identification in the same post... This data is scarcely released by its owners, yet companies may use it for internal research. With apps that track people's location, it is possible to imagine a big scale Sociopatterns-like database, where several billions of people would have their proximity interactions recorded. Privacy implications are a concern, though it is likely that such data already exists, even if not in a standardised format.

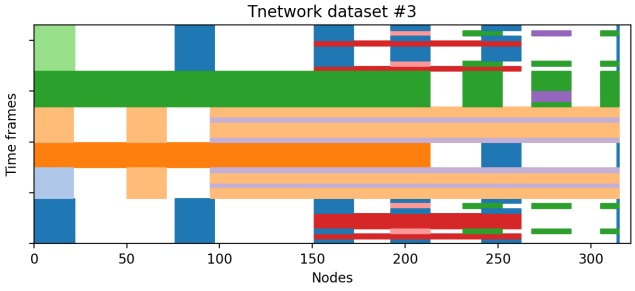

Figure 6: Example of random scenario generated by `tnetwork`.

Graph with 242 nodes and 8317 edges.

Method	#	aNMI%	Modul%	Conduc%	Surp	Stab%	Time (s)
groundtruth	11.0	100	62	70	874	100	1.005199999326578e-06
louvain	6.0	79	67	56	875	99	0.16850536679999947
leidenW	6.0	79	67	56	882	99	0.14660755630000039
leiden	5.3	75	63	52	856	94	0.13891863009999952
label	1.0	0	0	0	0	100	0.017861347699999898
walktrap	5.0	61	54	53	715	100	0.14857121959999803
infomap	1.0	0	0	0	0	100	0.03496186389999991

Table 1: Community detection on the aggregated school graph using standard static methods. Column # is the number of communities discovered; then partitions are compared to groundtruth using Adjusted Mutual Information. Modularity, Conductance and Surprise describe the fitness of the partition. Stability is the consistency of results over 10 random iterations. Time of execution is given for performance comparison.

Graph with 242 nodes and 2340 edges.

Method	#	aNMI%	Modul%	Conduc%	Surp	Stab%	Time (s)
groundtruth	11.0	100	68	33	2274	100	2.004000000965789e-06
louvain	7.2	83	72	18	2375	98	0.05116954990000124
leidenW	6.6	81	72	16	2432	96	0.04283884670000191
leiden	6.9	83	72	16	2535	99	0.040756389099999527
label	8.0	85	70	19	2468	100	0.015577493500001083
walktrap	8.0	86	72	18	2650	100	0.04235497800000019
infomap	8.0	87	72	18	2657	100	0.01987894130000143

Table 2: Community detection on the trimmed school graph. Thin edges have been removed because Infomap and Label propagation implementations were not able to take weights into account. The graph is sparser which makes all algorithms faster.

3 Experiments

3.1 Static analysis

In this section, we aim at testing famous static algorithms presented in section 1.3. Using different quality functions, we run them several times on static weighted graphs to measure their stability. To do so, we need to change the link stream into a static graph.

Definition. *The **cumulated graph** of a link stream $L = (T, V, E)$ is a static graph $G = (V, F)$, where every link in E adds a unit of weight on the corresponding edge in F:*

$$F = \big\{(u,v) \mid u,v \in V; \exists t \in T, (t,u,v) \in E\big\}$$

$$\forall (u,v) \in F,\ \omega_{u,v} = \Big|\big\{t \in T \mid (t,u,v) \in E\big\}\Big|$$

If we consider the Primary School dataset, the resulting cumulated graph G has 242 nodes and 8317 edges, while L had 125'773 links.

The `cdlib` community detection library offers tens of algorithms for static partitioning found in the literature. Here we only focus on the most widely spread ones, described in section 1.3.1: Louvain and Leiden algorithms, Infomap, Walktrap and Label propagation. The groundtruth is given by Sociopattern: eleven communities, consisting of ten school-classes and the group of teachers. They may not be related to edge density, since for instance teachers will mostly interact with their students. Yet it would be interesting if algorithms could infer this information from the graph.

To compare results and groundtruth, we use a mathematical tool called Adjusted Mutual Information (AMI). For two partitions A and B (sets of communities) in a set of n elements, the standard Mutual Information is $\mathcal{MI}(A,B) = \mathcal{H}(A) + \mathcal{H}(B) - \mathcal{H}(A \cap B)$, with the entropy $\mathcal{H}$ given by:

$$\mathcal{H}(A) = -\sum_{C \in A} \frac{|C|}{n} \log \frac{|C|}{n}$$

In the adjusted version, the expectation $\mathcal{E}$ of mutual information between random partitions is computed.

$$\mathcal{AMI} = \frac{\mathcal{MI}(A,B) - \mathcal{E}}{\max(\mathcal{H}(A), \mathcal{H}(B)) - \mathcal{E}}$$

Table 1 gives for each method the average number of communities detected. The partitions are compared to the groundtruth using AMI. For quality measurement, we compute the Modularity, Conductance and Surprise. Additionally, as some procedures are randomised, we are interested to see their sensitivity to randomness: a Stability index is computed by running 10 times the algorithm and comparing the results using AMI. Finally, the computation time is measured to compare performances.

Here the problem is that Infomap and Label propagation do not take weights into account in this implementation: they see an almost complete graph and are unable to detect any community structure, leading to a unique community. This is a big issue, and one way to go around it is to trim the graph: we remove edges when their weight is lower than 10, considering that rare interactions between children are not that relevant. Table 2 shows the results.

Note that this version is slightly faster, since 2000 edges are involved instead of 8000. All the algorithms give decent and quite similar results, with 85% of groundtruth information recovered. Interestingly, we see that they all output a partition with higher modularity than the groundtruth. This may be due to a resolution limit (with groundtruth having 11 communities while the optimal may be 7), yet it also shows that this quality function is not the "true" answer to community detection, rather one index among others, with pros and cons. Conductance as well is much higher in the groundtruth, meaning that the true groups are not that strongly separated. In general, groundtruth does not necessarily match any mathematical fitness function[51], so we need to be careful when using them.

3.2 Topochrone analysis

In this section, we transform link streams into a new type of static graph and discover dynamic communities using static algorithms. The idea is to represent time as just another topological dimension, like physicists sometimes consider it as a fourth spatial dimension. This concept is often used in the literature for snapshots (see section 1.3), with names such as time-space graph, transversal network or time-graph. In the case of link streams, we introduce the term *topochrone* for a graph containing both topological and chronological edges.

Definition. *Given a link stream $L = (T, V, E)$, the corresponding* ***topochrone*** *is a static weighted graph $G = (W, E_{topo} \cup E_{chrono}, \omega)$ with:*

$$\begin{aligned} W &= \big\{(t,u) \;\big|\; \exists v \in V, (t,u,v) \in E\big\} \\ &\subseteq T \times V \\ E_{topo} &= \big\{\big((t,u),(t,v)\big) \;\big|\; (t,u,v) \in E\big\} \\ E_{chrono} &= \big\{\big((t_1,u),(t_2,u)\big) \\ &\quad \big| \; \exists v \in V, (t_1,u,v) \in E; \\ &\quad t_2 = \min_{t>t_1} \{E \cap t \times u \times V \neq \emptyset\}\big\} \end{aligned}$$

In other words, there is now one so-called *tnode* for every moment when a node is interacting. Two tnodes are connected topologically if their corresponding nodes are interacting at that time. The weight for these edges is 1. Besides, two tnodes are connected chronologically if they represent the same node at consecutive

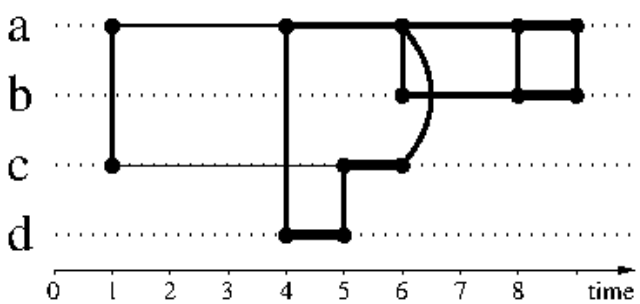

Figure 7: Conversion into a topochrone graph

Figure 8: Communities detected by Leiden algorithm in the topochrone of the primary school dataset.

moments. We propose 3 different designs for the weight of this edge: constant, linear decrease or exponential decrease. We use constants α to balance the importance of time over topology, and τ to adjust the time scale.

$$\omega^{\text{const}}_{(t,u),(t+\delta t,u)} = \alpha$$
$$\omega^{\text{lin}}_{(t,u),(t+\delta t,u)} = \frac{\alpha}{\frac{\delta t}{\tau}+1}$$
$$\omega^{\text{exp}}_{(t,u),(t+\delta t,u)} = \alpha \exp -\frac{\delta t}{\tau}$$

They all satisfy $\omega = \alpha$ when $\delta t = 0$, but have different rates of decrease.

Let us take the small link stream of figure 1 as a toy example. The corresponding topochrone is in figure 7. Longer chronological edges are thinner, to show the time decay.

For ℓ links and x tnodes, we obtain a graph of x nodes and about $\ell + x$ edges. While it is still possible to run all static algorithms in theory, we will have to take complexity into account. Indeed, quadratic procedures work just fine for 200 nodes, but may take ages to treat 200'000 of them. On the other hand, those operating in quasi-linear time over edges should work quite fast.

A first attempt with the usual 5 algorithms showed that Label propagation is not suitable: it computed communities in 31 minutes, while Leiden needed 9 seconds. Walktrap was also eliminated because it took 4 minutes, a bit too long for experiments.

Figure 8 shows the communities obtained with Leiden algorithm on the topochrone. We see consistently that classes are found during each class-session: 2 in the morning, separated by a small break, same in the afternoon, and a long lunch break where communities span over the whole network. We could not assess the stability of this algorithm because AMI (that we used for this purpose) is quadratic in the number of nodes; here it would require 30 billion comparisons, which is not worth considering. It found 122 communities, and does not seem to suffer too much of the resolution limit (which would cause classes to be merged into bigger communities).

3.3 Randow walks

In this section, we explore random walks as a way to discover communities in the link stream. No transformation into a static graph is needed. Instead, tnodes will be chosen at random as starting points, and walkers will be sent out. The walks will follow a path given by temporal and topological rules, and will heat up tnodes it encounters.

Our intention, is that the heat map will reveal communities: two tnodes are in the same community if they have many paths in-between. The random walks are here an approximation of a probability distribution that would be too costly to compute exactly.

In part 3.3.1, communities are found independently, by comparing the heatmap to the probability a tnode has to be encountered from a random starting point. In part 3.3.2, we launch competitive walkers then tnodes select their best option.

3.3.1 Independent communities

To detect one community around a tnode, a number of random walkers will first be launched from this tnode. They will explore the stream following some transition rules, and heat up every tnode they see (algorithm 1). Once the heatmap is obtained (algorithm 2), hottest tnodes will be tagged as poles (see definition of remarkability), and new walkers will be sent from the poles in the next run (algorithm 3). The successive refinement of the heatmap give what we call here a community, where each tnode has a specific rate of belonging. A threshold can then be applied to decide which ones are indeed part of the community.

In the first version, many parameters had to be set by hand:

- η: number of communities to find.
- ρ: number of runs. One run consists in launching walkers and selecting the poles.

Algorithm 1 RandomWalk

Input: tnode $a = (t, u)$
Output: Heatmap h
  Set heatmap h to zero
  for k times **do**
    Draw δt from a Geometric distribution of mean τ
    Heat up t', u for $t < t' \leq t + \delta t$
    Jump on new tnode $a = (t', v)$
  end for

Algorithm 2 HeatMap

Input: poles P, ω walkers
Output: Heat distribution h of length $\mathcal{D}(h)$
  Set heatmap h to zero
  for ω times **do**
    Pick tnode $b \in P$ at random
    $h = h$+RandomWalk(b)
  end for
  Normalise $h_b = \frac{h_b}{\sum_{c \in P} h_c}$ for all b

Algorithm 3 Partition using random walks

Input: link stream L, η communities,
  depth k, time constant τ, precision π
Output: Overlapping partition A
  A is empty
  $\omega = \pi \frac{|T| \cdot |V|}{k \cdot \tau}$ walkers
  $h =$HeatMap($\mathcal{D}$ all tnodes, ω)
  for η times **do**
    Select a random tnode a
    Create a set of poles $P = \{a\}$
    for k times **do**
      $h' =$HeatMap(P, ω)
      Remarkable poles $P = \{b \mid \mathcal{R}_{b|P} > 1\}$
    end for
    Add community P to partition A
  end for

- ω: factor to determine the number of walkers sent, for instance $\omega \propto \sqrt{p}$ walkers for p poles.
- λ, μ: describe the total length of the walk, and the length of stays before jumping.
- γ: decides the cutoff for heat.

Such a set of parameters makes an algorithm hard to tune, with results that can vary greatly depending on their choice. Therefore, we aimed at reducing the number of factors, and to make them more intuitive so that the tuning is straightforward.

First, λ and μ are not intuitive because we do not know what the length of a path should be. However, we may have an idea of a time constant τ for the data: interactions between people last a few minutes, while astronomy observe events over millennia, and biological processes happen over milliseconds. Besides, as real-world networks tend to have a small diameter, some methods like Walktrap stop the random walk after a few number of jumps k. We thus redefine our walk: a walker stays on a node during a time δt (sampled with a geometric distribution of mean τ) then jumps; after k topological jumps, the walk stops. Additionally, we fix the number of runs to k as well, because each run deepens the community topologically.

An other problem is γ. We would like the cutoff for heat to depend on the data instead of an external parameter. Starting from tnode a, we want to keep tnodes b that have been seen a lot (high heat $h_{b|a}$), but especially if they have been seen more often than with a random starting point (low background heat h_b). To measure the background heat, we add a pre-process in which many random walks are sent at random in the stream. Both types of heat can be normalised (over the explored domain) to approximate probabilities:

$$p_{b|a} = \frac{h_{b|a}}{\sum\limits_{c \in \mathcal{D}_a} h_{c|a}} \quad \text{and} \quad p_b = \frac{h_b}{\sum\limits_{c \in \mathcal{D}} h_c}$$

A criterion to select b as a good partner of a is to see if the probability to find b is higher if the walk starts from a than if it starts anywhere. We use local mutual information[1, 5]:

$$MI(a,b) = \log \frac{P(a,b)}{P(a)P(b)} = \log \frac{P(b|a)P(a)}{P(a)P(b)} \simeq \log \frac{p_{b|a}}{p_b}$$

As walks starting from a only explore a domain of tnodes $\mathcal{D}_a$ out of the full domain $\mathcal{D}$, it gives an advantage to all the nodes b visited by the walk. To avoid taking every tnode in the community, we confront mutual information to exploration rate. The cutoff criterion now states that b becomes a pole if it is *remarkable* from a:

Definition. *The* ***remarkability*** *of tnode b from tnode a is:*

$$\mathcal{R}_{b|a} = \frac{|\mathcal{D}_a| \cdot p_{b|a}}{|\mathcal{D}| \cdot p_b}$$

Tnode b is ***remarkable*** *from a when $\mathcal{R}_{b|a} > 1$.*

Note that the same definition works for a set C of tnodes (a community) instead of a. It represents the bonus of b if walkers start at random in C.

The next parameter to tackle is ω, the number of walkers. Random walks are a way to approximate a probability that would be too costly to compute exactly, so we would like to have an infinity of walkers. On the other hand, the program should run as fast as possible. A decent ω would allow to explore all the stream, regardless of the number of poles to start from:

$$\omega = \frac{\text{number of tnodes}}{\text{tnodes seen by one walker}} \simeq \frac{T \cdot |V|}{k \cdot \tau}$$

Note that T only counts active frames in practice, to avoid troubles when many frames are blank (at night in a school for instance).

The final set of parameters is η, τ, k, and a precision factor π which only aims at accelerating the computation when needed, sending $\omega_\pi = \pi \cdot \omega$ walkers.

We tested this algorithm on the school dataset, but it runs very slowly. The precision

had to be reduced to $\pi = 20\%$, and computation time was still 80 seconds on average. Plus, by design it finds exactly $\eta = 100$ communities, but many of them overlap greatly, or are basically the same.

3.3.2 Competitive communities

The main problem of the previous approach is that we need to define the number of communities η. In this section, we circumvent it by adding a competition between communities.

The idea comes from ant colony optimisation: random walkers move according to heat created by previous walks, or in the analogy, ants follow pheromone trails laid down by their kin.

Here, the process (described in algorithm 4) starts with plenty of distinct ant farms, each with one tnode. Each farm has a couple of ants (ω_π in total) who explore and lay down pheromones with the farm's identity. When exploration is done, each tnode b makes a decision: among farms F that make it remarkable ($\mathcal{R}_{b|F} > 1$), it joins the one with highest remarkability.

Note that if a tnode is not remarkable from any existing farm, it remains without a farm. Besides, we need to remove farms that are too small (in term of tnodes); a simple criterion is that a farm must allow at least one ant to realise a full walk, or in numbers: $|F| \geq \tau$, where constant τ has been modified a little and now represents the time span of a walk, regardless of the number of jumps k.

On top of tnodes selecting their farm, we tried to create a system in which ants are "afraid" of rival pheromones, and either step back or perish when they see too much of it. However, it was complex both in terms of understanding and computation time, and did not seem to yield better results.

3.3.3 Computation time

In the last version with competitive ant farms, the complexity mainly depends on the input and not on the results. Namely, the complexity is given by:

Algorithm 4 Partition using ant farms

Input: link stream L, depth k,
  walk duration τ, precision π
Output: Non-overlapping farms
  Create $f = \pi\frac{m}{\tau}$ farms of 1 random tnode
  for k times **do**
    for $\omega = \pi\frac{T \cdot n}{\tau}$ times **do**
      Choose F proportionally to $|F|$
      Ant starts at random in F
      Ant walks τ frames and k jumps, laying down F-pheromones
    end for
    Tnodes choose most remarkable farm
    Delete small farms when $|F| < \tau$
  end for

$$Complexity \simeq Rounds \times Ants \times Walk$$

One random walk visits an average number of tnodes given by $Walk = \frac{\tau \cdot m}{T \cdot n}$, where T, m and n are the number of active frames, links and nodes in the stream. This corresponds to the density of tnodes per frame multiplied by the number τ of frames crossed by a walk.

There is one walk for each of the ω ants at each of the k rounds. As $\omega = \pi\frac{T \cdot n}{\tau}$, the overall complexity reads:

$$Complexity \simeq \pi \cdot k \cdot m$$

We acknowledge that this is not a correct mathematical definition of the complexity, because it is not clear on which set it is averaged; therefore asymptotic notations are avoided.

However, it gives an indication about the size of datasets on which it can be used. We see that it is linear in m, the number of links in the stream. Depth k depends on the topology, but we suggest an integer between 4 and 8, after network diameter. Precision π indeed reduces the computation time, yet results may be much poorer. Finally, τ does not appear, because the extension of walks is balanced by the eviction of ants.

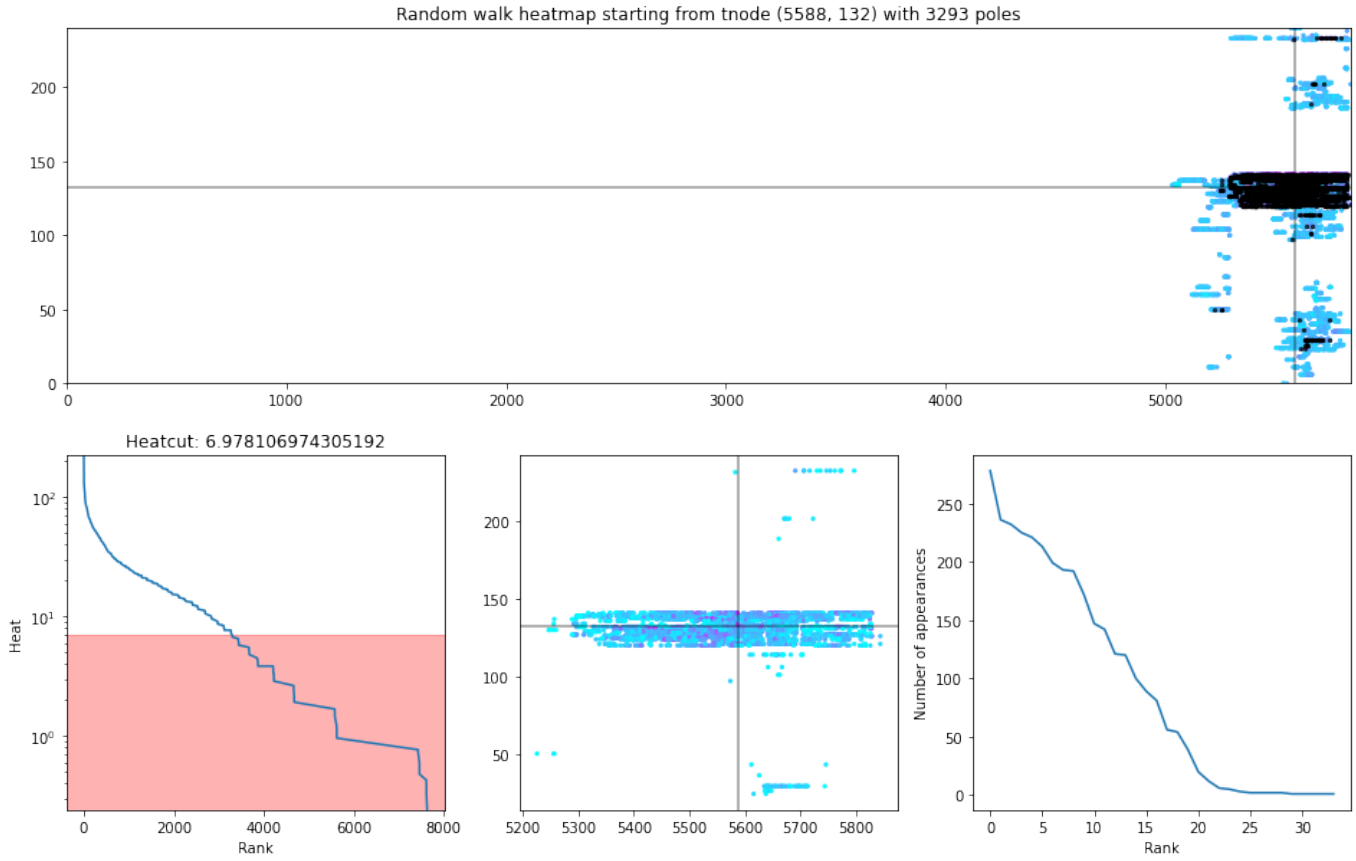

Figure 9: Test panel for communities detected by random walks. Top: heat map over the whole stream, where black points are the selected poles among visited blue points. Left: heat distribution, with the cut corresponding to the remarkability threshold. Center: detected community. Right: number of tnodes for each node.

For an order of magnitude, on the school dataset, it took on average 12 seconds to process the $m \simeq 125'000$ links, with depth $k = 4$ and full precision $\pi = 1$. This is comparable to Leiden algorithm applied on the topochrone (9 seconds). Both were run on my laptop, but Leiden is interfaced from C++ while ours is written in Python.

3.3.4 Tests

In order to see if the communities are coherent, a control panel was designed. For one instance, figure 9 shows heatmap, heat distribution and tnodes per node. It reveals that communities seem quite noisy: we see a clean rectangle, meaning that all pupils of one class have been selected together during a time interval, which is good; but there are also some individuals who only appear a couple of times. It is unclear why they should belong to the community, and why not during the whole interval. The distribution of tnodes per node measures that problem: we see that some nodes are present hundreds of times, while the last ones barely appear. This curve could be used to decide of a cut at the inflexion point: the 20 first nodes (which account for most tnodes) would be kept, while the remaining 10 noisy ones would be removed.

A few tests also reveal a high instability. With independent random walks, too much importance is given to the initial tnodes, which are drawn at random; they can lead to redundant highly-overlapping communities, while big areas of the network are completely forgotten.

In the case of random ants, it is less of an issue, as farms have to be good in order to survive: many communities are almost identical

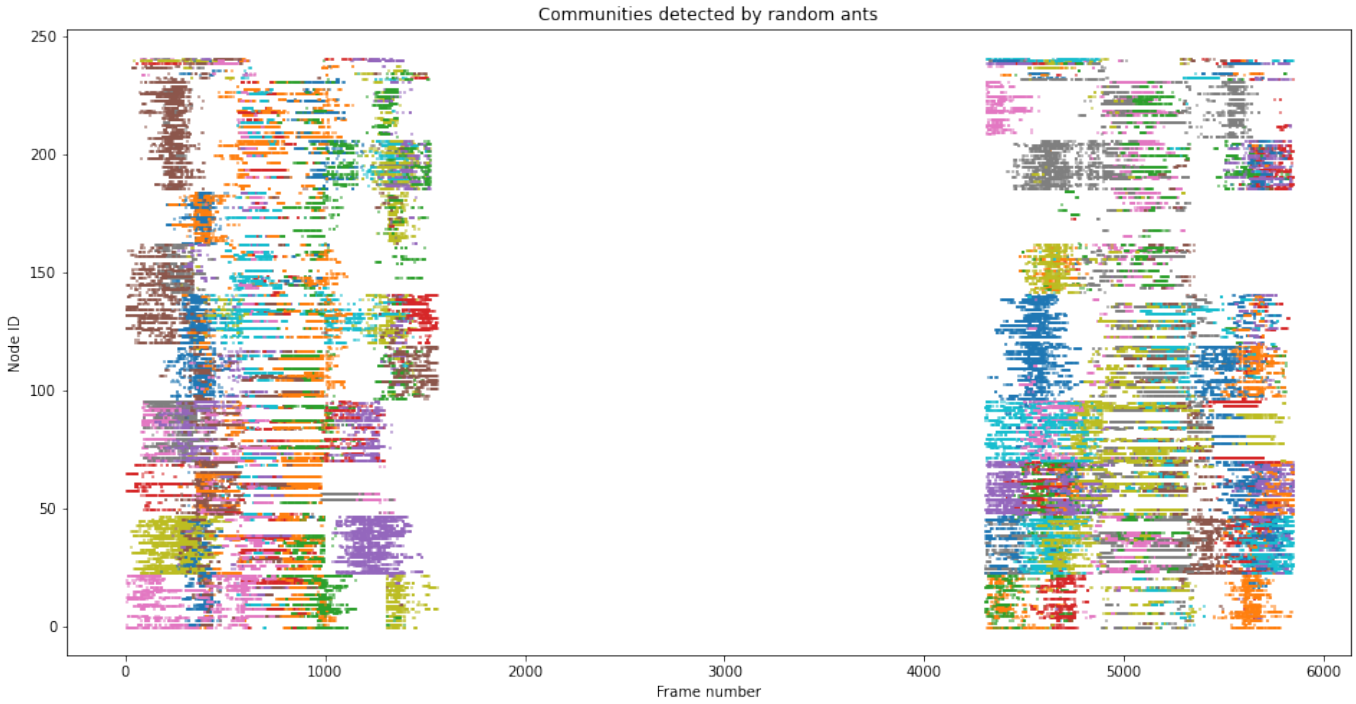

Figure 10: Communities detected by random ants. Parameters: depth $k = 4$, walk length $\tau = 90$ (one hour translated in 20s frames), precision $\pi = 1$. The execution took 12 seconds on average, and found between 80 and 100 communities.

throughout instances; but in some other zones, sometimes a community is found and sometimes it stays empty. This may mean that the zone is not significantly structured, in which case we need a mathematical concept to measure this. The visual result for one instance of the random ants is given in figure 10.

More generally, we need quantitative criteria for quality, smoothness and stability of the communities we discover. Longitudinal similarity is proposed by `tnetwork` for this purpose, as a value to balance with instantaneous fitness. It may be a useful tool for further research.

Another insight could be the comparison of all the algorithms on snapshots: static communities could be transferred on the stream and cut into snapshots; dynamic communities could be reduced by taking for each node the community in which it was most times. Then on these snapshot communities, all the quality functions such as modularity and conductance could be applied, as well as comparison to groundtruth. Unfortunately I had no time to conduct such extensive tests.

Conclusion

During this internship, I learnt about community detection in static networks and all the issues faced by quality functions optimisation. I discovered algorithms for dynamic communities on snapshots, and the recent concept of link stream, that brings fine-grained time information in the game of community detection.

After getting my hand in several programming tools specifically designed for networks and communities, I started observing various datasets, especially those of Sociopatterns. It was just a way to start designing algorithms, without trying to face really big data.

Next, I built a process to test static community detection algorithms on aggregated graphs, and compared some relevant methods of the literature. Then defining the topochrone of a link stream, I reused the same algorithms to obtain dynamic communities, ruling out those with high time complexity. Lastly, I ran experiments of random walks, with the underlying goal of rapidity and locality of the computation.

Results are not breathtaking, but they may be a basis for further improvement.

In the future, I would like to keep in mind the following mistakes that lead to big losses of time. Though I was told about the many existing programming modules soon enough, it took me a long time before finally using them all. In the meantime, I re-coded many functions that were already – and better – implemented in them.

It would also have been more efficient to build a testing infrastructure very soon in the project. Indeed, I waited to have fully functioning versions before thinking of how to test and compare them. While designing the algorithms, I mostly used intuition to make changes, instead of well-defined scores. On top of that, I did not save all the intermediary results, so a lot of redundant computation had to be done.

To go further with the project, I would try to improve the random ants algorithm. First, it needs a better mathematical grounding: the measure of remarkability has been introduced here, but there may be more relevant tools available in information theoretic literature[32].

In order to deal with bigger datasets such as crypto-currencies, the running time should be improved. While the complexity for random ants seems reasonable ($k \cdot m$), its factor could be reduced greatly. Indeed, everything was coded in Python with little optimisation: this language is very convenient to write and read code, but it is meant to be slower than lower-level languages like C++. Plus, the extensive use of dictionaries for heat maps is known to slow down the process significantly, and pheromones are erased at every step while some of their information could be kept.

I truly thank my advisor for ... his advice, but more particularly for his humble and honest approach to research: he was always listening to ideas and very careful in rejecting them, which lead to constructive discussions and a nice sense of belonging.

References

[1] Robert M Fano. The Transmission of Information. 1949.

[2] David A Huffman. A Method for the Construction of Minimum-Redundancy Codes. 1952.

[3] Herbert A. Simon. On a Class of Skew Distribution Functions. *Biometrika*, 1955.

[4] Erdős and Rényi. On the evolution of random graphs. 1960.

[5] David M Magerman and Mitchell P Marcus. Parsing a Natural Language Using Mutual Information Statistics. 1990.

[6] Duncan Watts and Steven Strogatz. Collective Dynamics of Small-World Networks. 1998.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 1998.

[8] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 1999.

[9] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. The diameter of the world wide web. *Nature*, 1999.

[10] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[11] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 2001.

[12] Ravi Kannan. On Clusterings: Good, Bad and Spectral. 2004.

[13] Deepayan Chakrabarti, Dharmendra S Modha, Spiros Papadimitriou, and Christos Faloutsos. Fully Automatic Cross-associations. 2004.

[14] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 2005.

[15] Roger Guimera and Luis A. Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 2005.

[16] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks (long version). 2005.

[17] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 2006.

[18] Nelson Alves. Unveiling community structures in weighted networks. *Physical Review E*, 2007.

[19] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 2007.

[20] Usha Nandini Raghavan, Reka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 2007.

[21] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 2007.

[22] Remi Dorat, Matthieu Latapy, Bernard Conein, and Nicolas Auray. Multi-level analysis of an interaction network between individuals in a mailing-list. 2007.

[23] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, 2008.

[24] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 2008.

[25] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 2009.

[26] Santo Fortunato. Community detection in graphs. *Physics Reports*, 2010.

[27] Benjamin H. Good, Yves-Alexandre de Montjoye, and Aaron Clauset. The performance of modularity maximization in practical contexts. *Physical Review E*, 2010.

[28] Derek Greene, Dónal Doyle, and Pádraig Cunningham. Tracking the Evolution of Communities in Dynamic Social Networks.. IEEE, 2010.

[29] Zhengzhang Chen, Kevin Wilson, Ye Jin, William Hendrix, and Nagiza Samatova. Detecting and Tracking Community Dynamics in Evolutionary Networks. 2010.

[30] Francesco Folino and Clara Pizzuti. Multiobjective evolutionary community detection for dynamic networks. 2010.

[31] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela. Community Structure in Time-Dependent, Multiscale, and Multiplex Networks. *Science*, 2010.

[32] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. 2010.

[33] Michele Coscia, Fosca Giannotti, and Dino Pedreschi. A Classification for Community Discovery Methods in Complex Networks. *Statistical Analysis and Data Mining*, 2011.

[34] Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Lorenzo Isella, Jean-François Pinton, Marco Quaggiotto, Wouter Van den Broeck, Corinne Régis, Bruno Lina, et al.. High-Resolution Measurements of Face-to-Face Contact Patterns in a Primary School. *Plos One*, 2011.

[35] Jiaxing Shang, Lianchen Liu, Feng Xie, Zhen Chen, Jiajia Miao, Xuelin Fang, and Cheng Wu. A Real-Time Detecting Algorithm for Tracking Community Structure of Dynamic Networks. 2012.

[36] Jierui Xie, Mingming Chen, and Boleslaw K. Szymanski. LabelRankT: incremental community detection in dynamic networks via label propagation.. ACM Press, 2013.

[37] Philippe Vanhems, Alain Barrat, Ciro Cattuto, Jean-François Pinton, Nagham Khanafer, Corinne Régis, Byeul-a Kim, Brigitte Comte, and Nicolas Voirin. Estimating Potential Infection Transmission Routes in Hospital Wards Using Wearable Proximity Sensors. *Plos One*, 2013.

[38] Valerio Gemmetto, Alain Barrat, and Ciro Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC Infectious Diseases*, 2014.

[39] Petter Holme. Modern temporal network theory: A colloquium. *The European Physical Journal B*, 2015.

[40] V. A. Traag, R. Aldecoa, and J.-C. Delvenne. Detecting communities using asymptotical surprise. *Physical Review E*, 2015.

[41] Jialin He and Duanbing Chen. A fast algorithm for community detection in temporal network. *Physica A*, 2015.

[42] Emmanuel Orsini. Détection de communautés dans les flots de liens par optimisation de la modularité. *Marami*, 2015.

[43] Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 2016.

[44] Honglei Zhang, Jenni Raitoharju, Serkan Kiranyaz, and Moncef Gabbouj. Limited Random Walk Algorithm for Big Graph Data Clustering. 2016.

[45] Tanmoy Chakraborty, Ayushi Dalmia, Animesh Mukherjee, and Niloy Ganguly. Metrics for Community Analysis: A Survey. 2016.

[46] Amir Ghasemian, Pan Zhang, Aaron Clauset, Cristopher Moore, and Leto Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *Physical Review X*, 2016.

[47] Catherine Matias and Vincent Miele. Statistical clustering of temporal networks through a dynamic stochastic block model. 2016.

[48] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream Graphs and Link Streams for the Modeling of Interactions over Time. *Social Networks Analysis and Mining*, 2017.

[49] Catherine Matias, Tabea Rebafka, and Fanny Villers. A semiparametric extension of the stochastic block model for longitudinal networks. 2017.

[50] Noé Gaumont. Groupes et Communautés dans les flots de liens: des données aux algorithmes. 2017.

[51] Leto Peel, Daniel B. Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science Advances*, 2017.

[52] Giulio Rossetti and Rémy Cazabet. Community Discovery in Dynamic Networks: a Survey. *ACM Computing Surveys*, 2018.

[53] Vincent Traag, Ludo Waltman, and Nees Jan van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 2019.

[54] Tiago P. Peixoto. Bayesian stochastic blockmodeling. 2019.

[55] Souaad Boudebza, Rémy Cazabet, Omar Nouali, and Faical Azouaou. Detecting Stable Communities in Link Streams at Multiple Temporal Scales. 2019.