UNIVERSITY OF COPENHAGEN

FACULTY OF SCIENCE

PhD thesis

Christopher Alan Ryther

Complex Temporal Networks

Metrics and embeddings for problems in real world complex networks

Advisors: Jakob Grue Simonsen, Andreas Koch, and Claus Parnes

Handed in: December 31, 2018

Thesis Summary

Temporal Graphs are collections of nodes and edges which change over time. These types of graphs can model many types of real-world scenarios such as email communications, online social networks, author citation networks and so on. With the growth of data globally, graph data has become more complex, e.g. introducing node attributes and labels, therefore new tools and methods are needed to efficiently analyze and solve complex graph problems. This thesis studies several challenges in static and temporal graphs and aims to explore how to best utilize complex network data.

The thesis consists of four parts. The first part of the thesis describes how to model both static and temporal graphs, as well as attributes and labels, in a streaming fashion, and important graph metrics and properties of nodes.

The second part investigates how node attributes in graphs can be used to uncover latent connections between nodes. Augmenting static graphs with additional edges, based on shared attributes between pairs of nodes, can connect similar but unconnected nodes. The experiments show that propagating labels along these edges based on attributes can reliably improve classification performance of relational classifiers without the need for complex inference or feature engineering.

The third part explains some shortcomings of current temporal node metrics, and how to improve them. Extensions based on two distinct concepts are proposed. The first focuses on encoding information about node labels directly into node metrics, making them "label-sensitive", for example by modifying how shortest paths are calculated in graphs. The second attempts to capture "recent" topological changes in temporal graphs. This is done by tracking the evolution, or change, of node metrics over time, and weighing the amount of change by how long ago they took place. Experiments on real-world datasets demonstrate that utilizing especially recent changes of label-sensitive node metrics can improve classification performance. The notions presented in these experiments are then generalized to weighted static and temporal graphs using functions of nodes and their weights. Through experiments, the proposed method is shown to be flexible and can be adapted to various domains and can capture a a wide range of information in a weighted graph, including label-sensitivity and recency mentioned earlier.

Finally, a novel embedding for temporal graphs is presented which consists of a combination of Gaussian mixture models and static graphs. The embedding is designed for streamed temporal graphs with a high degree of edge-insertions over time, and to maintain a compact size in memory. Experiments on real and synthetic datasets show that the embedding scales well and can reliably and accurately be used to query the state of the temporal graph at any point in time.

The conclusions of this thesis are that the use of node attributes and labels, combined with additional temporal information, can improve accuracy when solving classification tasks in temporal graphs. Furthermore, it shows that graphs can be modeled effectively with the use of mixtures of distributions when enough data is present. While examples of applications of the proposed methods have been given, there are still many interesting challenges ahead.

Contents

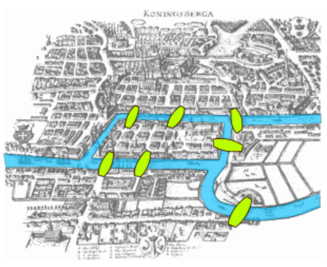
List of Figures

List of Tables

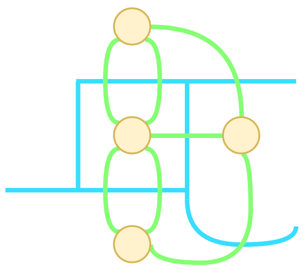
1

Introduction

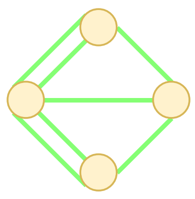
Whether realizing it or not, most people in the world use graphs as a tool throughout their lives for various tasks. A graph consists of a collection of *nodes* which can be connected to each other by *edges*, and can be used in a wide range of scenarios from brainstorm mind-mapping to online social network analysis. The strength lies in their ability to model relationships and processes in a both intuitive and efficient manner. Therefore it is no surprise they have been the subject of a large amount of research over many years.



(a) Map over central Königsberg river and bridges.



(b) Abstract visualization of Königsberg river and bridges.



(c) Graph model of Königsberg. Nodes correspond to boroughs and edges to bridges.

The earliest known paper in the field, the Seven Bridges of Königsberg, was written by Leonhard Euler in 1736 and laid out the foundation of the graph theory we use today [27]. In his work he split up central Königsberg (now Kalinigrad, Russia) into 4 parts, each physically separated from one another by the river Pregel which runs through the city, and represented each part as a node in a graph. He then placed edges between nodes if there was a bridge connecting the two areas, thereby capturing the structure of bridges over Pregel in the city. His task was to plan a walk through the city which traversed each bridge exactly once, and with the graph representation he proved it impossible. Although the Königsberg problem was small in todays scale, it was an important step in the advancement of network science.

For decades, the sizes of graphs used for analysis were limited by the state of technology, and the analysis could be performed by humans with relatively simple techniques. Factors such as the revolution of the internet and computer storage, changed how data was collected, shared and analyzed, and resulted in graphs being used to solve problems for many different kinds of data:

- Social networks made up of family and social ties. Especially online social networks have allowed for research into human behavior, such as how people can influence others opinions, on a massive scale [28].
- Communication networks, typically based on electronic devices, which can range from cell phone calls and emails to ip traffic between servers across the globe [29].
- Analysis of electrical grids, with networks of generators connected by power lines, is important to avoid blackouts and cascading failures [30].
- The brain itself is a large network of neurons and simplified models have been created to help understand its inner workings [31].
- Transport networks, such as public transit routes and road maps, can be used for urban and traffic planning in cities [32].
- Social contact networks, based on physical interaction or proximity of humans in the real world, have been used to predict the spread of epidemics such as H1N1 (swine flu). Furthermore the same models used for infections of humans have been successfully applied to other domains such as malware in computers [33].

However, while a network of bridges is mostly static, i.e. does not change over time, most real-world data and scenarios exhibit change over time. In online social networks friends are added/removed, in communications new emails are sent/received, etc. These factors necessitate adding a temporal dimension to graphs in order to accurately deal with these phenomena. This non-trivial task has spawned a large amount of research into extending the concepts of static graphs to temporal graphs. In the most general case, a temporal graph consists of the same components as a regular graph, a set of nodes and edges, but, for each node and edge, contains additional temporal information. Typically the temporal information takes the form of time-stamped events. For example, the time-stamp of an edge between two nodes in an email network can represent an email that was sent, or a node's timestamp, in an online social network, can indicate when that user first created their profile. The additional temporal information can take many forms, and depends on the domain which is being modeled.

Many surveys exist which cover the study of these types of graphs, especially static graphs, and the many methods which have been applied. In the following are three examples of real-world temporal data, which extend on the concepts of static graphs.

1.1 Movie Actor Network: IMDb

IMDb is a well-known online searchable database containing information on millions of movies and tv-programs, and the cast and crew of each. The information about each movie is vast and contains data such as country of origin, year of screening, actors in movie, box office earnings, length, user rating (from 0 to 10), and more. An example of how IMDb data is structured is shown in Figure 1.2.

A natural graph representation of the data is to model IMDb as a graph of actors, where two actors are connected if they have participated in a movie together. It can be said that the graph is constructed based on which movies these actors have in common. However, there are other attributes which actors share, which also can be used as a basis for the constructed graph, such as age, country of origin, gender, and so on. The graph and the following analysis or problem solution, can depend heavily on which data is selected and which relations are modeled. Some examples of different models are shown in Figure 1.3

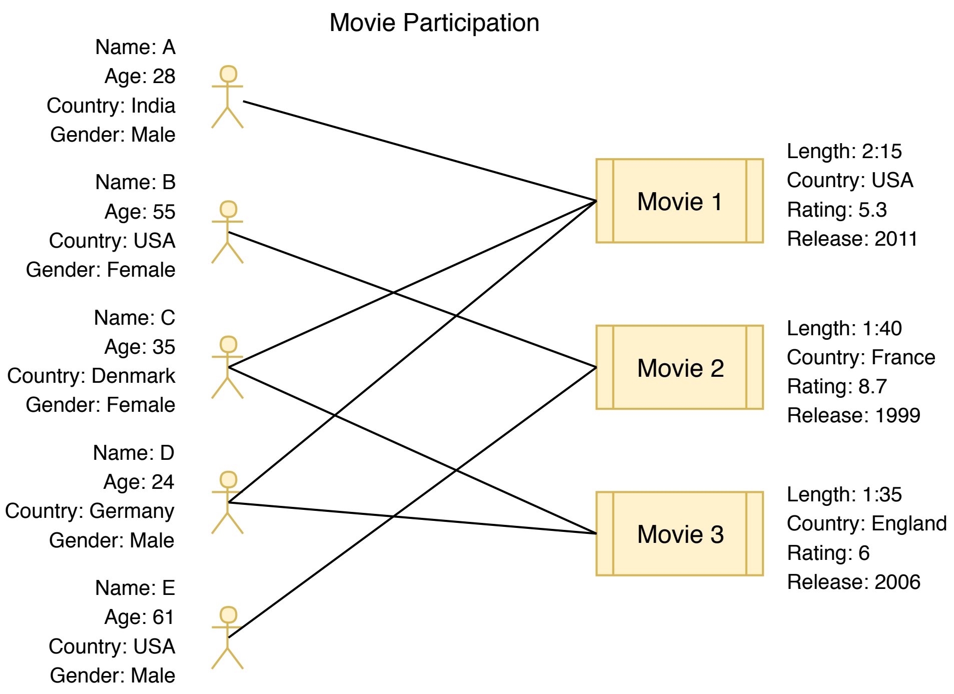


Figure 1.2: Visualization of how the online movie database IMDb is structured into actors, the movies they participate in, and associated metadata.

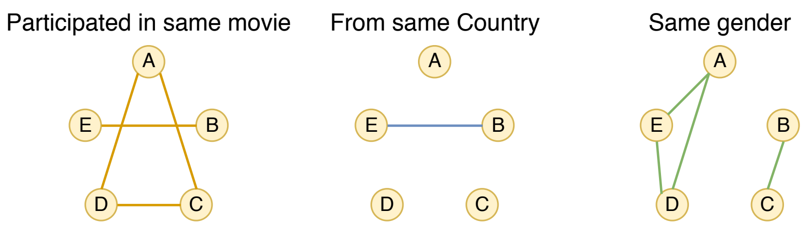


Figure 1.3: Visualization of three different approaches to representing the IMDb dataset as simple graphs. From left to right: 1) Edges are based on which movies actors have co-participated in 2) Edges between actors are based on shared country of origin 3) Edges are based on shared gender.

While each of the above graphs can be viewed and analysed independently, other approaches, which combine information from several of these graphs, can be used. An example of these types of methods is described in detail in Chapter 4.

1.2 Author Publication Network: DBLP

The website https://dblp.uni-trier.de/ is a public bibliography database in the field of computer science. Each entry in the database is the title of an article, and holds information such as authors, date of publication, and publication venue. For research purposes, data from this website was downloaded and held up against public records from ACM (and other sources), and the result was a public dataset called DBLP [34]. For each article, the DBLP dataset contains the aforementioned information, but also includes which articles cite one another. The structure of the DLBP data is visualized in Figure 1.4

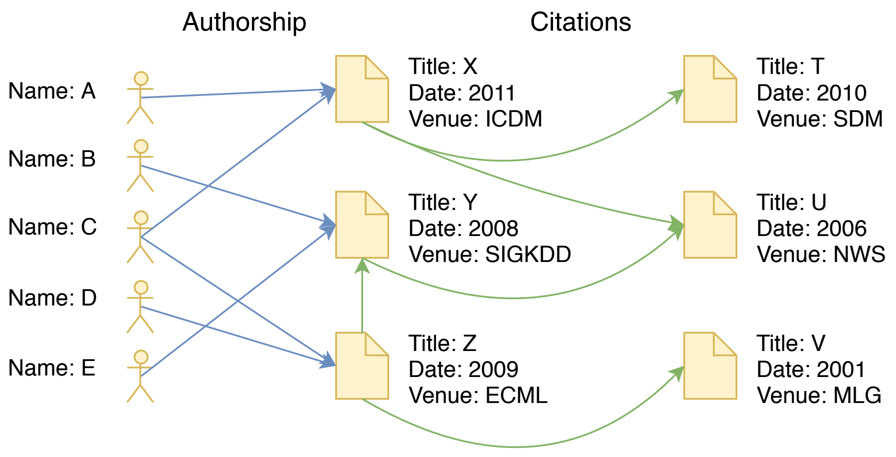


Figure 1.4: Visualization of how the DBLP dataset is structured. Authors on the left publish articles (blue connections), which in turn cite other previously published articles (green connections).

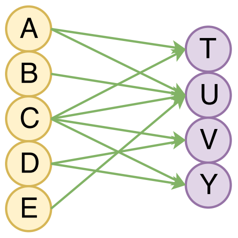


Figure 1.5: Author-citation network based on DBLP data.

This type of dataset can be used to solve problems such as characterizing emerging conferences or trends based upon the articles and authors attending, matching a new paper to appropriate reviewers from the same field, and predicting future collaborations between authors. It lends itself well to a graph representation, but since there are many types of relations in the data, there are several ways to model a temporal graph based on DBLP, depending on the problem at hand. An example of a problem is to predict whether an author, in the future, will publish a paper in a field, they haven't previously published in, based on which types of articles the author cites in their work. For this type of challenge, it can be advantageous to transform the DBLP dataset into a “author citation network”, shown in Figure 1.5. The author citation network consists of two types of nodes, authors and publications, and is a so-called bipartite graph because there exists two *types* of nodes in the graph. Using this graph and various methods designed for analysis it is then possible to predict what fields of science authors will publish in, in the future. The problem described above is treated in detail in Chapter 5.

1.3 Human Contact Network: RealityMining

The concept of social networks, especially online social networks, is widely known and has received significant attention in both media and academia. Often these types of graphs are used to better understand relationships and affiliations of people, based on their interactions such as communications. Human contact networks are a special case, and are the result of studies on close-range interactions between humans over time. These types of studies can be important in modeling how diseases spread in populations, or how certain social patterns emerge, by viewing humans as nodes, and edges as time-stamped contacts between pairs of nodes. Due to the difficulty of accurately monitoring and recording human interactions in the real world, such data is scarce. One project, the Reality Mining experiment, was based on tracking the contacts between 75 students or faculty members at MIT Media Laboratory, and 25 students of MIT Sloan business school [33]. The students were instructed to carry cell phones with bluetooth technology, and a monitoring application, over the course of 100 days. It was the first mobile data set of its kind and contains rich personal information, geospacial location, and interactions between participants. A visualization of the structure of data can be seen in Figure 1.6.

With fine-grained information on when each interaction took place, it is possible to model the human contacts as a temporal graph that grows over time, as each contact occurs. A temporal graph visualization of the network is shown in Figure 1.7. Each edge between two nodes corresponds to an interaction between two

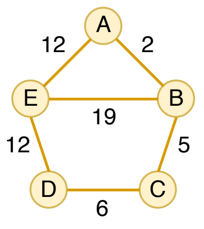


Figure 1.7: Temporal graph representation of the Reality Mining data.

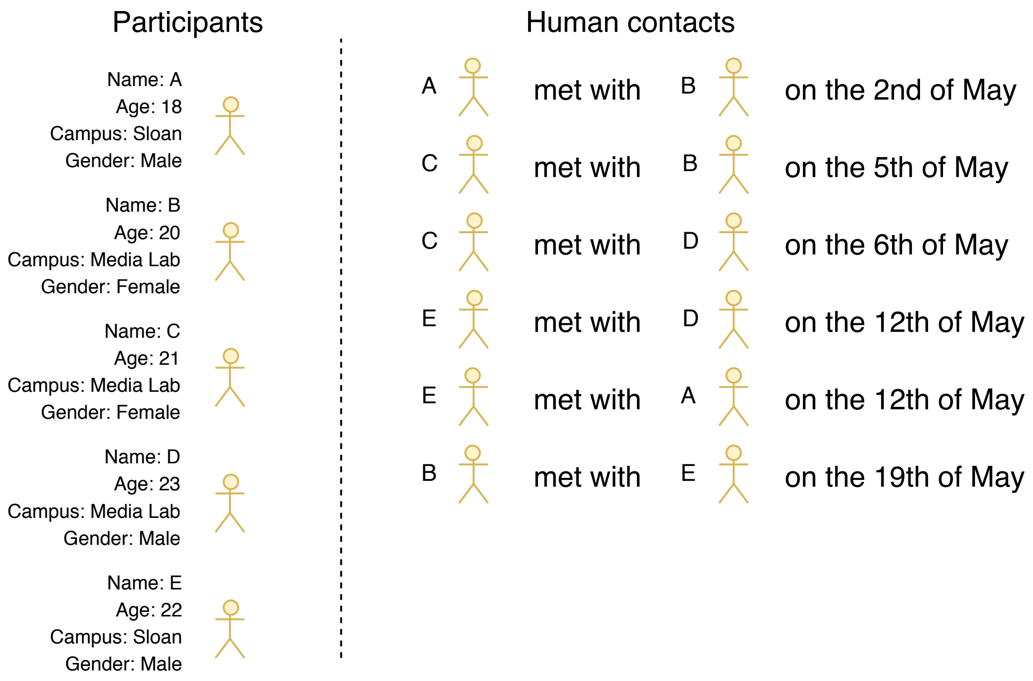


Figure 1.6: Visualization of how the Reality Mining research data is structured into participants, their physical contacts, and associated metadata.

people, and the edges are annotated with the day of the month on which the interaction took place.

The researchers behind the experiment found that they could predict the daily activities of students based on their behavior, waking up at 10 am in the morning, for example. The dataset in question consists of potentially many contacts between any two participants of the study, and can result in a very large temporal graph. Therefore a completely different type of challenge is how to represent the graph, so that it allows for efficient analysis. This is a problem that is explored further in Chapter 7.

1.4 List of Publications

Over the course of my PhD project I have written five papers listed below. The chapters of this thesis are based on the papers as follows: Chapter 4 is based on [35], Chapter 5 is based on [36, 37], Chapter 6 is based on [38], and Chapter 7 is based on [39].

- [35] Christopher Ryther, Jakob Simonsen, and Andreas Koch, ”Within-network classification with label-independent features and latent linkages”, in *Pro-*

ceedings of the 12th International Workshop on Mining and Learning with Graphs (MLG), 2016.

- [36] Christopher Ryther and Jakob Simonsen, ”Within-network classification in temporal graphs”, in *Proceedings of the Eighth IEEE ICDM Workshop on Data Mining in Networks (DaMNet)*, 2018.
- [37] Christopher Ryther and Jakob Simonsen, ”Recency and Label-Sensitivity for Classification in Temporal Graphs”, *awaiting submission*.
- [38] Christopher Ryther and Jakob Simonsen, ”Weighted and Time-Sensitive Metrics for Node Classification in Temporal Graphs”, submitted to *Network Science*, 2018.
- [39] Christopher Ryther and Jakob Simonsen, ”GraphGMM: History-Preserving Gaussian Embeddings for Temporal Graphs”, *awaiting submission*.

Each of the chapters begin with a brief introduction of the idea or motivation behind the work.

1.5 Contributions

The contributions of this thesis are summarized below:

1. In [35] we propose two methods to increase connectivity of static graphs for the purpose of within-network classification. One method creates an additional graph where edges are *only* based on shared attributes, and the other method augments the original graph with edges between nodes with shared attributes. Using a selection of well-known relational classifiers, and a baseline static graph representation, we show that our proposed methods can improve classification accuracy on sparsely labeled real-world datasets. The work is presented in Chapter 4.
2. In [36] and [37] we extend our focus on within-network classification to the temporal domain, using time-varying graphs to represent network evolution over time. We introduce variants of already established temporal centrality metrics and clustering which capture, what we posit are, desirable node properties in temporal attributed networks. The two variants are *recency-sensitive*, i.e. captures recent changes of a node's metrics, and

label-sensitive, which modifies the metrics to encode information about labels of nodes in relation to each other. Through classification experiments on nine real-world datasets we show improvement of weighted F1-score on all using a combination of both proposed concepts. The work of both is collectively presented in Chapter 5.

3. In [38] we propose theoretically-principled weighted node metrics for both static and temporal graphs. We present eight centrality based metrics and 2 clustering based metrics. The weighted metrics are easily adaptable to various domains using so-called *node functions* as metric parameters. We exemplify their use with four prototypical node functions which are able to capture the same or similar information as those used in [36, 37]. These weighted node metrics are then applied to three real-world datasets, and compared to unweighted metrics as features for within-network classification. Results show that careful tailoring of node functions can lead to improvement of classification accuracy. The work is presented in Chapter 6.

4. Finally in [39] we propose an embedding for temporal multigraphs called GraphGMM. The embedding consists of sequences of gaussian mixture models that for each node allow for efficient queries on the number of edge-insertions at any time-instant. We present two different version of GraphGMM, one node-centric and one edge-centric, and run experiments on large temporal datasets with up to 2.5 million edges. We show that our embeddings are efficient, compact and scalable when compared to baseline graph implementations. The work is presented in Chapter 7.

2 Temporal Graphs

In this section we introduce preliminaries and definitions of graphs that evolve over time. First we provide generic descriptions of static graphs, and then extend these to provide definitions of graphs with time-stamped edges (temporal graphs), graphs with weights (weighted graphs), and sequences of temporal graphs (time-varying graphs). Finally we discuss alternative graph models from related work.

2.1 Static graphs

Definition 1. *Let $G = (V, E)$ be a static graph, where V are the nodes of the graph, and E are the edges, where $E \subseteq (V \times V)$.*

2.2 Temporal Graphs

As the literature on time-windowed graphs contains many variations in nomenclature, we give full definitions appropriate for our domains of application, many of which originate from [8]. In the remainder, let $\mathbb{T}$ be a totally ordered set (representing *time*).

Definition 2. *A* ***temporal graph*** *G is a tuple (V, E), where V is the set of nodes of G and E is the set of edges. An edge $e \in E$ is a triple (x, y, t) where $x, y \in V$ and $t \in \mathbb{T}$. In the remainder of the paper $e = (x, y, t)$ should be understood as an undirected edge* *between x and y at time t. A **temporal multigraph** is a temporal graph which allows for multiple edges between pairs of nodes.*

Definition 3. *A **weighted temporal graph** G is a temporal graph (V, E, W), where V, E are the nodes and edges, respectively, and $W : V \mapsto \mathbb{W}$ are weights on nodes in V. For any node $x \in V$ there exists a weight $w_x \in W$.*

An example of the three temporal graph types is shown in Figure 2.1.

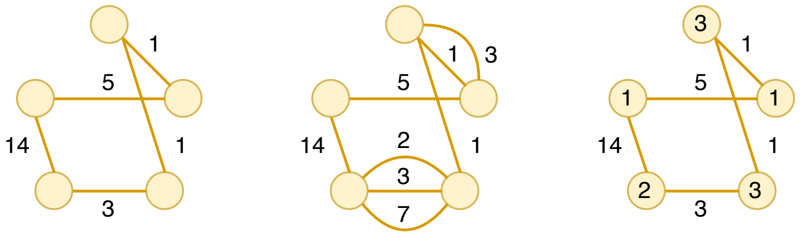


Figure 2.1: Three examples of temporal graphs: on the left a regular temporal graph, in the middle a temporal multigraph, and on the right a weighted temporal graph.

When graphs grow massive and evolve over time, one approach to handling them is to process them in a so-called data stream model, where changes to the graph are represented as continuous updates e.g. node-insertions or edge-insertions. In the following we define streams, and how temporal graphs can be induced from streams of edge-insertions.

Definition 4. *A **stream** S is any multiset $S \subseteq \mathbb{D}^2 \times \mathbb{T}$ consisting of triplets of the form (x, y, t). A **substream** of a stream S is a subset $s \subseteq S$. A **slice** of a stream S at time i, is a substream $s_i \subseteq S$ such that $s_i = \{(x, y, i) : (x, y, i) \in S\}$. The relation $\preceq$, defined by: $(x_i, y_i, t_i) \preceq (x_j, y_j, t_j) \quad if \quad t_i \leq t_j$, is a partial order on S.*

Intuitively, a stream $[(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots]$ is a set of "events" (x, y) each happening at some time t, and a slice s_i is the set of all events that took place at time i. Typically, (x, y) represents the creation of a directed edge between nodes x and y in a graph at time t.

Definition 5. *Let $\mathcal{S} = \{s_0, s_1, s_2 \cdots\}$ be a set of substreams of the stream S. $\mathcal{S}$ is then strictly partially ordered by $\prec$, defined by:*

$$s_i \prec s_j \quad \text{if} \min_{(x_a, y_a, t_a) \in s_i} t_a < \min_{(x_b, y_b, t_b) \in s_j} t_b \tag{1}$$

which results in ordering the substreams by which contains the earliest event.

Definition 6. *Let s be a substream. The **substream graph** G_s is a temporal graph (V, E) whose nodes and edges are defined as:*

$$V = \bigcup_{(x,y,t)\in s} \{x, y\} \quad \textit{and} \quad E = \bigcup_{(x,y,t)\in s} \{(x, y, t)\} \tag{2}$$

*A special case of temporal substream graph is the **bipartite substream graph** denoted G^b_s, which is a bipartite temporal graph (V, U, E) whose nodes and edges are defined as:*

$$V = \bigcup_{(x,y,t)\in s} \{x\} \quad \textit{and} \quad U = \bigcup_{(x,y,t)\in s} \{y\} \quad \textit{and} \quad E = \bigcup_{(x,y,t)\in s} \{(x, y, t)\} \tag{3}$$

where V is referred to as the set of *top nodes* and U as the set of *bottom nodes*. It follows from the definition of bipartite graphs that $V \cap U = \emptyset$.

Definition 7. *Let S be a stream. The **streamed multigraph** G_S is a temporal multigraph (V, E) whose nodes and edges are defined as:*

$$V = \biguplus_{(x,y,t)\in S} \{x, y\} \quad \textit{and} \quad E = \biguplus_{(x,y,t)\in S} \{(x, y, t)\} \tag{4}$$

where $\biguplus$ is the multiset union operator.

2.2.1 Time-Windowed Graphs

Some temporal metrics require modeling the stream as several temporal graphs, where each graph represents the "state" of the stream during a specific period of time. A straightforward approach is to split the stream into consecutive windowed substreams, such that each substream induces a temporal graph. To do this we define a type of graph container where each temporal graph in the container is the result of a batch of insertions/deletions. This type of graph is called a *time-windowed graph*. An illustration of this type of graph, and the stream that induces it, is shown in Figure 2.2.

Definition 8. *A **time-window** τ is a tuple $\tau = (t, t') \in \mathbb{T}^2$ where $t < t'$. Let $\mathcal{T} \subseteq \mathbb{T}^2$ be any non-empty set of time-windows. Then $\prec$, defined by:*

$$(t_i, t_j) \prec (t_a, t_b) \quad \textit{if} \quad t_i < t_a \textit{ or } (t_i = t_a \textit{ and } t_j < t_b) \tag{5}$$

is a strict partial order on $\mathcal{T}$.

Definition 9. *Let $\tau = (t_i, t_j)$ be a time-window and S be a stream. The* ***windowed substream*** *s_τ, induced by the time-window τ and stream S, is the substream s_τ of S defined by:*

$$s_\tau = \{(x, y, t) : (x, y, t) \in S \textit{ and } t_i \leq t < t_j\} \tag{6}$$

Thus, the windowed substream intuitively is the set of all "events" occurring between t_i and t_j.

Thus, the windowed substream intuitively is the set of all "events" occurring between t_i and t_j.

Definition 10. *A* ***time-varying graph*** *(TVG) is a sequence of temporal graphs $\mathcal{G} = (G_0, G_1, G_2 \cdots)$, where each graph G_{i+1} is constructed from its predecessor G_i by inserting additional nodes/edges or deleting existing nodes/edges. This definition differs from that of [8]: In our definition, $\mathcal{G}$ is based on temporal graphs instead of non-temporal graphs, and time-windows are not part of the time-varying graph.*

Definition 11. *Let $\mathcal{S} = (s_0, s_1, s_2 \cdots)$ be a sequence of substreams. We denote by $\mathcal{G}_\mathcal{S}$ the strictly partially ordered TVG $(G_{s_0}, G_{s_1}, G_{s_2} \cdots)$ induced by definition 6.*

Definition 12. *A* ***time-windowed graph*** *is a pair $(\mathcal{G}, \mathcal{T})$, consisting of a time-varying graph, $\mathcal{G}$, and a set of time-windows $\mathcal{T}$ where $|\mathcal{G}| = |\mathcal{T}|$. For each temporal graph $G_i \in \mathcal{G}$ there is a time-window $\tau_i = (t_m, t_n) \in \mathcal{T}$ so that for any edge $e = (x, y, t) \in G_i$, we have $t_m < t < t_n$. The* adjacency matrix *of a time-windowed graph is the map $A_{(\mathcal{G},\mathcal{T})}$ such that if $\tau_i \in \mathcal{T}$ is a time-window, then $A_{(\mathcal{G},\mathcal{T})}(\tau_i) = a_{xy}(\tau_i)$ is the adjacency matrix of the temporal graph for time-window τ_i.*

Definition 13. *Let $\mathcal{T} = \{\tau_0, \tau_1, \tau_2 \cdots\}$ be a set of time-windows and S be a stream. The time-windowed graph $(\mathcal{G}, \mathcal{T})$ induced from S and all time-windows in $\mathcal{T}$ consists of a sequence of (bipartite) substream graphs:*

$$\mathcal{G} = \{G_{s_{\tau_0}}, G_{s_{\tau_1}}, G_{s_{\tau_2}} \cdots\} \tag{7}$$

and satisfies:

$$\tau_0 \prec \tau_1 \prec \tau_2 \cdots \Rightarrow G_{s_{\tau_0}} \prec G_{s_{\tau_1}} \prec G_{s_{\tau_2}} \cdots \tag{8}$$

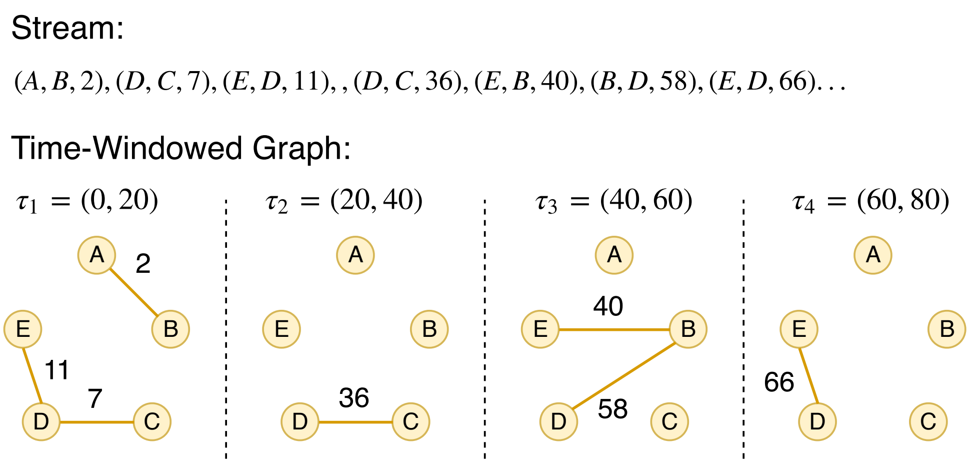


Figure 2.2: Example of stream and resulting time-windowed graph. At the top is a stream of edge-insertions over time, and on the bottom are the four resulting temporal graphs induced by four time-windows of equal size ranging from the time-instant 0 to time-instant 80.

Definition 14. *Let $\mathbb{T} = \mathbb{R}_{\geq 0}$ and $\Delta\tau$ be a value in $\mathbb{T}$. A* ***time-windowed graph with fixed time-window size*** *(TWGFW) is a time-windowed graph $(\mathcal{G}, \mathcal{T})$ where each time-window $(t, t') \in \mathcal{T}$ satisfies:*

$$t' = t + \Delta\tau \tag{9}$$

and that for any two adjacent time-windows $\tau_{ij} = (t_i, t_j)$, $\tau_{mn} = (t_m, t_n) \in \mathcal{T}$, where $\tau_{ij} \prec \tau_{mn}$, we have $t_j = t_m$.

2.2.2 Accumulated Temporal Graphs

Definition 15. *Let $\mathcal{G}$ be a time-varying graph. The* ***accumulated graph*** *$G^a(V^a, E^a)$ is the temporal graph whose nodes and edges are defined as:*

$$V^a = \bigcup_{\substack{G_i(V_i,E_i)\in\mathcal{G}:\\ x\in V_i}} \{x\} \qquad \text{and} \qquad E^a = \bigcup_{\substack{G_i(V_i,E_i)\in\mathcal{G}:\\ (x,y,t)\in E_i}} \{(x, y, t)\} \tag{10}$$

Similarly the ***accumulated bipartite graph*** *$G^a(V^a, U^a, E^a)$ is the bipartite temporal graph defined by:*

$$V^a = \bigcup_{\substack{G_i(V_i,U_i,E_i)\in\mathcal{G}:\\ x\in V_i}} \{x\} \quad \text{and} \quad U^a = \bigcup_{\substack{G_i(V_i,U_i,E_i)\in\mathcal{G}:\\ y\in U_i}} \{y\} \quad \text{and} \quad E^a = \bigcup_{\substack{G_i(V_i,U_i,E_i)\in\mathcal{G}:\\ (x,y,t)\in E_i}} \{(x, y, t)\} \tag{11}$$

2.2.3 Labels in Time-Windowed Graphs

As a main purpose of the paper is to predict future labels on *nodes*, temporal information on nodes, in particular their labels, must be treated; hence the following definitions.

Definition 16. *Let $\mathbb{L}$ be a set. An* ***observed label*** *L is a triplet (x, l, t) where $x \in \mathbb{D}$, $l \in \mathbb{L}$ and $t \in \mathbb{T}$. In the remainder of the paper $L = (x, l, t)$ should be understood as x being assigned label l at time t.*

Definition 17. *A* ***labelstream*** *$\mathcal{L}$ is a temporally ordered set of observed labels $(L_0, L_1, L_2 \cdots)$ strictly partially ordered by $\prec$, defined by*

$$(x_i, l_i, t_i) \prec (x_j, l_j, t_j) \textit{ if } t_i < t_j \tag{12}$$

2.2.4 Snapshots of Time-Windowed Graphs

For classification of nodes in a TVG based on streams, we introduce "snapshots" in order to clearly model the TVG at the "present" and "future".

Definition 18. *Let t_p be a value in $\mathbb{T}$, $\mathcal{T}$ be a set of time-windows, S be a stream, and $\mathcal{L}$ be a labelstream. A* ***snapshot*** *at time t_p is a quintuple $(S_{t_p}, \mathcal{G}_{t_p}, \mathcal{T}_{t_p}, G^a_{t_p}, \mathcal{L}_{t_p})$ where S_{t_p} is a substream of S, and $\mathcal{T}_{t_p}$ is a subset of $\mathcal{T}$, both consisting of elements timestamped before t_p. We denote $(\mathcal{G}_{t_p}, \mathcal{T}_{t_p})$ as the time-windowed graph induced by S_{t_p} and $\mathcal{T}_{t_p}$, and $G^a_{t_p}(V^a_{t_p}, E^a_{t_p})$ as the accumulated graph of $\mathcal{G}_{t_p}$. The labelstream subset $\mathcal{L}_{t_p} \subseteq \mathcal{L}$ consists of labels timestamped before t_p for nodes in $V^a_{t_p}$. A node $v_i \in V^a_{t_p}$ is said to be* labeled in the snapshot at time *t_p, if there exists a label $(v_i, l, t) \in \mathcal{L}_{t_p}$.*

Definition 19. *Let t_p be a value in $\mathbb{T}$, $\mathcal{T}$ be a set of time-windows, S be a stream, and $\mathcal{L}$ be a labelstream. A* ***snapshot*** *at time t_p is a quintuple $(S_{t_p}, \mathcal{G}_{t_p}, \mathcal{T}_{t_p}, G^a_{t_p}, \mathcal{L}_{t_p})$ where:*

$$S_{t_p} = \{(x, y, t) \in S : t \leq t_p\} \tag{13}$$

$$\mathcal{T}_{t_p} = \{(t_i, t_j) \in \mathcal{T} : t_i \leq t_p\} \tag{14}$$

S_{t_p} is a substream of S, and $\mathcal{T}_{t_p}$ is a subset of time-windows $\mathcal{T}$. We denote $(\mathcal{G}_{t_p}, \mathcal{T}_{t_p})$ as the time-windowed graph induced by S_{t_p} and $\mathcal{T}_{t_p}$, and $G^a_{t_p}(V^a_{t_p}, E^a_{t_p})$ or $G^a_{t_p}(U^a_{t_p}, V^a_{t_p}, E^a_{t_p})$ as the accumulated (possibly bipartite) graph of $\mathcal{G}_{t_p}$. We define the labelstream subset $\mathcal{L}_{t_p} \subseteq \mathcal{L}$ as:

$$\mathcal{L}_{t_p} = \{(x, l, t) \in \mathcal{L} : t \leq t_p \textit{ and } x \in V^a_{t_p}\} \tag{15}$$

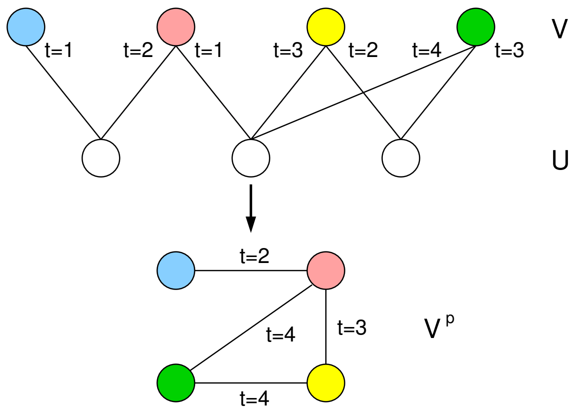


Figure 2.3: Example of a top-node bipartite temporal graph projection.

A node $v_i \in V^a_{t_p}$ is said to be labeled in the snapshot at time t_p*, if there exists a triplet $(x, l, t) \in \mathcal{L}_{t_p}$ where $x = v_i$.*

2.2.5 Projections of Time-Windowed Graphs

Some of the datasets, used for experiments in this paper, are naturally bipartite graphs, but the metrics used are designed for non-bipartite graphs. While any bipartite graph can be viewed as an non-bipartite graph, it is often better for modeling purposes to use projections from bipartite to non-bipartite graphs. We briefly give the necessary definitions below.

Definition 20. *Let $G(V, U, E)$ be a bipartite temporal graph, the **temporal projection** $G^p(V^p, E^p)$ of G, is the one-mode projection of G onto V. G^p is then a temporal graph containing only nodes from V, where two nodes are connected by an undirected edge when they have at least one common neighboring node in U. The edge is assigned the largest timestamp seen from either of the two nodes, to the common neighbor.*

A visualization of temporal projection is shown in Figure 2.3. Note: by the above definition, any node without 2-hop neighbors in G will be isolated in G^p. A definition for temporal projection of a time-windowed graph is not required by the graph metrics in this paper, and thus omitted.

2.2.6 Shortest Temporal Paths

Some of the temporal node metrics used in our work are based on paths in temporal graphs, as defined in [8]. Given a temporal graph $G(V, E)$, a **temporal path** from node x to y is defined as a sequence of edges

$$P = ((x, n_1, t_1), (n_1, n_2, t_2), \cdots, (n_{l-1}, y, t_l))$$

such that no nodes are visited more than once and $t_i \leq t_{i+1}$ for any two neighboring edges in the path. The length of a path $\text{len}(P)$ is the number of edges in the sequence and the duration is defined as $\text{dura}(P) = t_l - t_1$. A node y is said to be **temporally reachable** from node x if there exists a temporal path from x to y. Given the set of all temporal paths between node x and y, $\mathbf{P}_{xy}$, and a temporal graph $G(V, E)$, a temporal path $P \in \mathbf{P}_{xy}$ is a **shortest temporal path** if $\text{len}(P) = \min\{\text{len}(P') : P' \in \mathbf{P}_{xy}\}$, and is a **fastest temporal path** if $\text{len}(P) = \min\{\text{dura}(P') : P' \in \mathbf{P}_{xy}\}$. Both are viable for path based node metrics, however for the remainder of the paper we use shortest temporal paths as they are most compatible with existing efficient implementations.

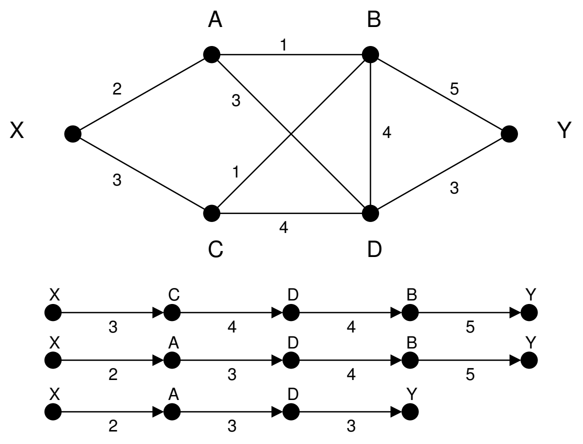


Figure 2.4: All temporal paths from X to Y.

2.3 Graph representations in related work

Several areas of research, such as analysis of communication networks, delay-tolerant networks or social networks, which seem unrelated at first, typically employ distinct domain-specific definitions to describe data. However, in reality they often share many of the same core concepts, such as time-stamped edges, but with different names [2]. There have been multiple attempts at creating models for temporal networks [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26] with a large amount of overlap.

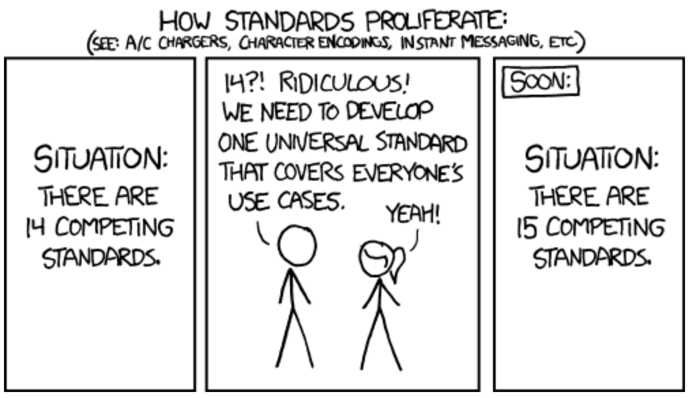

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Figure 2.5: Or a couple dozen [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]. From `www.xkcd.com/927/`.

In the following we review alternatives to the above definitions of time-varying graphs, as well as some of the more well-established unifying models, in order of increasing usefulness to our work.

A classic approach to representing graphs that change over time is to aggregate all edges presented into a single static graph, thereby discarding all dynamics of the data. The advantages of this are primarily simplicity; static graphs are much better understood, and one can therefore apply techniques from a rich set of static graph methods directly to the data [8, 21]. While still a severe oversimplification, there are methods which can encode temporal information into a static graph, for example by using weighted edges. In Figure 2.6 is an example of dynamic data, in this case phone calls between 5 people, and three examples of static representations of the phone calls. The phone calls are characterized by a caller, recipient, time of call, and a duration of call.

The first representation in Figure 2.6 is the aggregated graph described above, the second and third are weighted graphs, which encode temporal information in the edge weights. In the second graph, the weight of an edge between two nodes is equal to the total number of contacts between them, and in the third graph the weight is the total duration of phone calls between the pair of nodes.

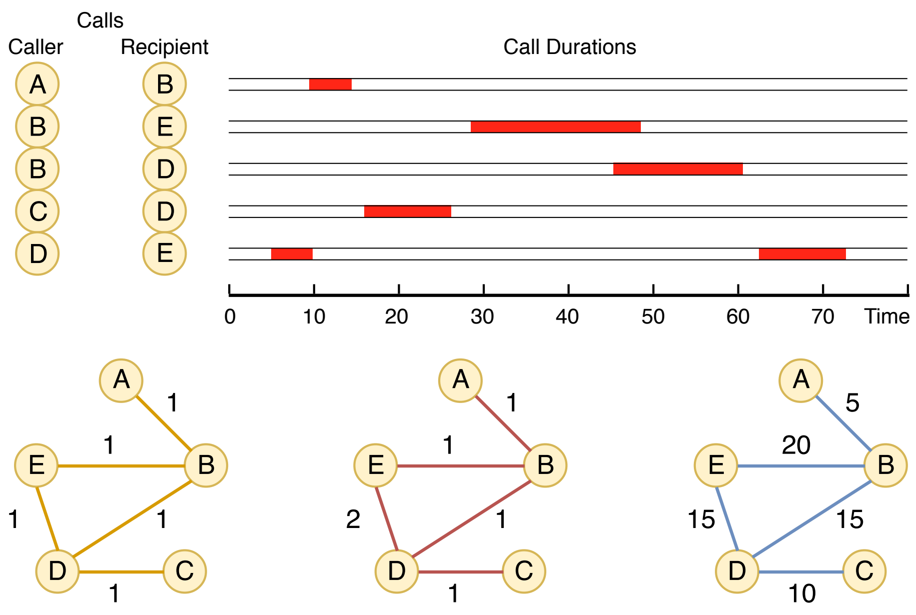


Figure 2.6: Static graph representations of dynamic call data. From left to right: a) Static aggregated graph, b) weighted by number of contacts, c) weighted by duration of contacts.

While the two last graphs have encoded some of the temporal data of the phone calls, they are not capable of representing more fine-grained temporal information, such as when each phone call took place, or the duration of each call. Furthermore these types of representations tend to over-represent the number of paths in the graph, which is undesirable when using path-based graph metrics [8]. Weighted graph representations, or similar, have been used in [18, 19, 40, 41, 42, 43].

A different graph representation that has been useful for especially very sparsely connected contacts graphs, is the *reachability graph* or *path graph*[12]. In sparsely connected graphs, nodes generally have very few connections to other nodes, or are completely isolated, in which case the typical classification methods can prove ineffective. Reachability graphs can in some cases remedy this, and are derived from temporal data by placing directed edges between a pair of nodes (A, B) if there exists a temporal path from A to B. This type of graph shows which nodes can affect others directly, and can be efficient for analyzing how epidemics spread in contact networks throughout time [11]. Reachability graphs have been used in [44, 45, 11, 46, 8, 47].

Time	$t_1 = 0$	$t_2 = 1$	$t_3 = 3$	$t_4 = 5$	$t_5 = 9$	$t_6 = 14$	$t_7 = 20$
Sender	A	A	E	B	B	D	A
Recipients	B	C,E	D	C	D	B	D

Table 2.1: Email communications.

Like static graphs can be represented by an adjacency matrix, temporal graphs can be represented by an *adjacency tensor* instead of traditional matrices [12, 48]. Adjacency tensors are higher order extensions of adjacency matrices, or data cubes, and essentially multidimensional arrays. The tensors used for adjacency in graphs resemble tensors used in mathematics and physics, but with fewer restrictions[48]. The simplest adjacency tensor describing a temporal graph is a third-order tensor where the first two dimensions represent the adjacency matrix and the last dimension represents time [49, 48]. The advantages and disadvantages of the adjacency tensor are similar to those of the adjacency matrix for static graphs: in relation to other representations, it takes up more space in memory compared to other representations, but with it follows an set of useful tensor algebra. Adjacency tensor representations have been used in [50, 51, 52, 53, 54, 55, 56].

A different option is to retain all temporal information within the graph. One solution is to use a so-called *static expansion*, which is a static graph representation of all the temporal data. To illustrate this method, we first present a small email dataset consisting of email communications from [6], shown in Table 2.1.

In contrast to the phone call dataset, the “contacts” in this dataset do not have durations, but are instead represented by a sender, one or more recipients, and the time the email was sent. The static expansion of the temporal data is a directed acyclic graph (DAG) $H = (S, E)$, and using the definitions adapted from work by Michail [21] is described as follows: Let $V = \{A, B, C, D, E\}$ be the set of people in the email dataset, $Q = \{(A, B, t_1), (A, C, t_2), \cdots\}$ be the set of sent emails, where a triplet (A, B, t_1) corresponds to an email sent from A to B at time t_1, and let $\lambda_{\min}$ and $\lambda_{\max}$ be the earliest and latest, respectively, time an email was sent. We then have

$$S = \bigcup_{v \in V} \bigcup_{i=\lambda_{\min}-1}^{\lambda_{\max}} v_i$$

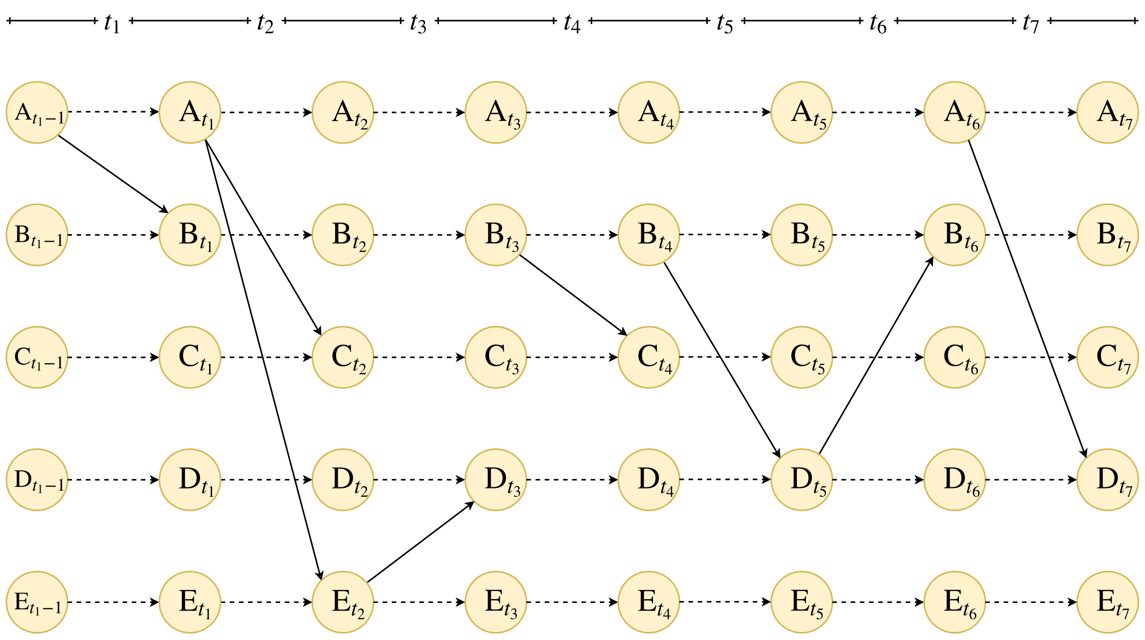


Figure 2.7: Static expansion of the temporal data in Table 2.1.

In words we create a so-called "time-node" for every node, for every point in time, with an additional "initial node" before the earliest time $\lambda_{\min}$, called A_{t_0-1}. Hence person A will be represented by eight nodes: $\{A_{t_0-1}, A_{t_1}, A_{t_2}, A_{t_3}, A_{t_4}, A_{t_5}, A_{t_6}, A_{t_7}, \}$. One can imagine a copy of all nodes for each time-instant, or "level". The edges E of the static expansion are then defined as:

$$E = \{(u_{(i-1)}, v_i) : \lambda_{\min} \leq i \leq \lambda_{\max} \quad \text{and} \quad (u = v \quad \text{or} \quad (u, v, i) \in Q)\}$$

The edges only connect time-nodes from one level to time-nodes one level later. Specifically a time-node, e.g. A_{t_1}, is only connected to A_{t_2} and every time-node which received an email from A at time t_2. The resulting graph is shown in Figure 2.7.

The static expansion has the advantages of being a static graph, but is sometimes counter-intuitive (in the above example, edges for emails *end* at nodes with the timestamp corresponding to when they were sent), and has a large size in memory compared to other representations since every node is represented multiple times. Similar representations have been used in [57, 58, 59, 16].

Kostakos proposed a type of temporal graph which improves on the static ex-

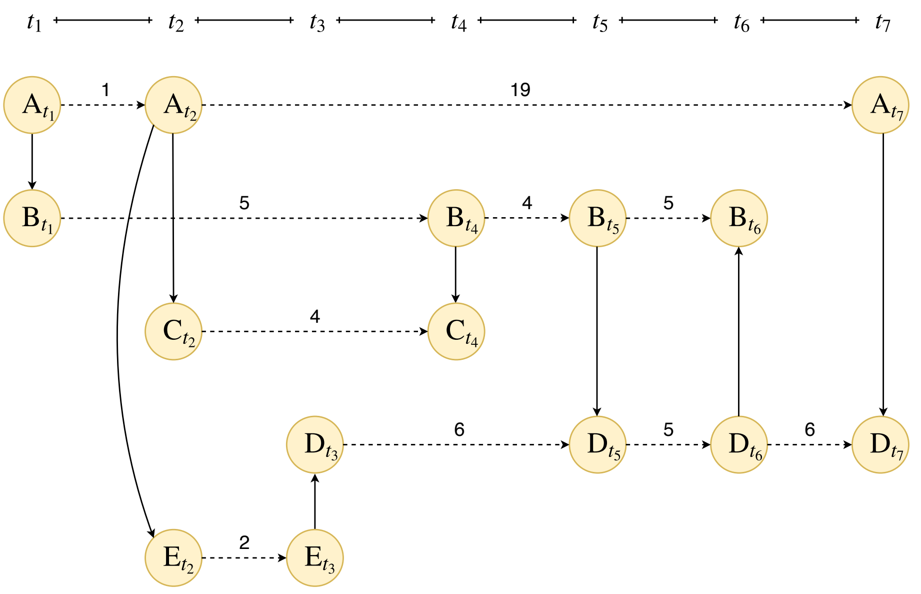


Figure 2.8: Kostakos' temporal graph representation of data in Table 2.1.

pansion above [6]. Kostakos creates the temporal graph in the following way: For each person, create a node for every point in time where that person sent or received an email. Hence person A will be represented by three node instances $\{A_{t_1}, A_{t_2}, A_{t_7}\}$. Then for each set of instances, they create edges between consecutive pairs $\{A_{t_i}, A_{t_{i+1}}\}$ with edge weight $t_{i+1} - t_i$. Finally, unweighted directed edges are created between the node instances of the corresponding senders and recipients of Table 2.1, e.g. an email from A to B at time t_1 results in an edge from A_{t_1} to B_{t_1}. Following these steps, the produced temporal graph is as shown in Figure 2.8

The temporal graph by Kostakos is more intuitive, and contains fewer nodes and edges, than the static expansion, while still allowing algorithms for static graphs, such as finding shortest paths, to be applied directly without extensive modifications [6]. However this representation is still relatively expensive, i.e. it unnecessarily takes up a lot of space in memory, and does not allow us to store information about durations of contacts, e.g. phone calls.

A different approach is instead using a sequence of graphs to represent time, for example the one used by Nicosia et al. [8]. Their definition of a time-varying graph consists of an ordered sequence of non-overlapping time-windows and an ordered sequence of graphs – similar to the ones we use in [36, 38, 37] – where each graph represents the activity/contacts between nodes that took place within its corresponding time-window. In our definitions, each "contact" had no duration and took place instantaneously, and they were therefore trivial to assign to time-windows. For a contact with a duration $(x, y, \tau, \delta\tau)$, where x was in contact with y at time τ for a duration of $\delta\tau$, and a time-window $[t, t + \Delta t]$, Nicosia et al. used the following criteria to determine if the contact belongs to the time-window:

$$t \leq \tau \leq t + \Delta t \tag{16}$$

$$t \leq \tau + \delta\tau < t + \Delta t \tag{17}$$

$$\tau < t \quad \wedge \quad t + \Delta t < \tau + \delta\tau \tag{18}$$

If the contact satisfies at least one of these criteria, it "overlaps" the time-window and therefore belongs to it. This also implies that a single contact can span multiple time-windows. An example of this type of time-windowed graph is shown in Figure 2.9.

Note that each of the graphs in the time-windowed graph of Nicosia et al. are static, whereas the ones we use are temporal and contain time-stamped edges, and thus don't allow for the use of temporal node metrics such as temporal betweenness centrality. Sequences of graphs, or similar, have been used in [22, 23, 24, 11, 25, 10, 26, 21, 60, 61]

Casteigts et al. have proposed a unifying graph representation for temporal network data which overcomes the above challenges [2]. They define a time-varying graph as a set of nodes N and a set of edges $E \subseteq V \times V \times L$ where L is a property or *label* that any relation between two nodes might have. They leave the definition of L open and allow L to contain multi-valued elements. Given a so-called "lifetime" $\mathcal{T} \subseteq \mathbb{T}$ of the data, within which all contacts or relations are assumed to take place, the time-varying graph has the form $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, where

- $\rho : E \times \mathcal{T} \mapsto \{0, 1\}$ is called the *presence* function and indicates whether an edge is available, or present, at a given time.

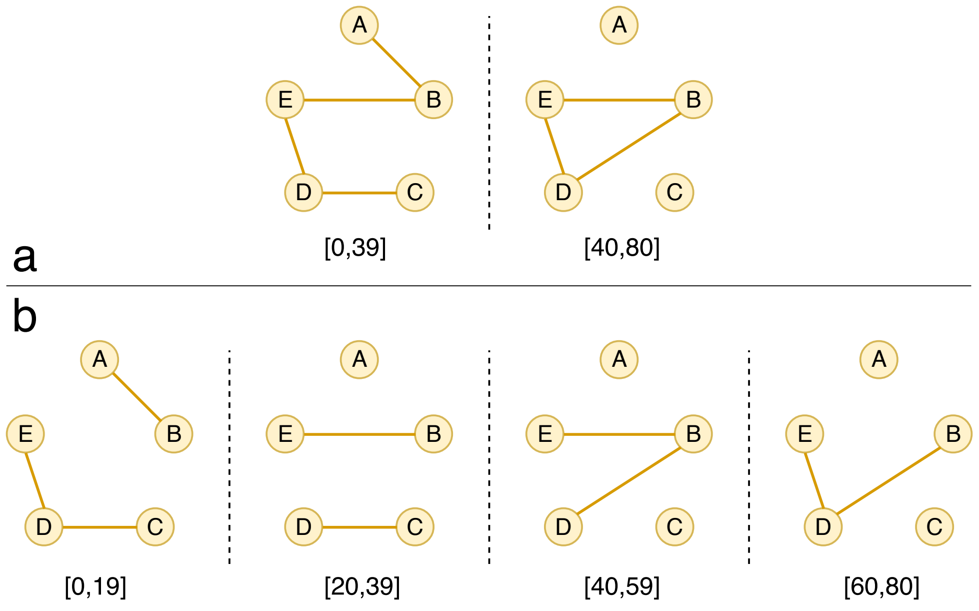


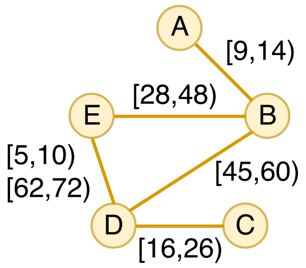
Figure 2.9: Example of a time-varying graph of Nicosia et al. using the phone calls in Figure 2.6. In **a)** the TVG is based on two time-windows of equal size and in **b)** there are four.

- $\zeta : E \times \mathcal{T} \mapsto \mathbb{T}$ is called the *latency* function and indicates the time it takes to "travel across" a given edge if traversed at a specific time.

The representation can also be extended to allow for penalties or presence functions for nodes, to account for e.g. disappearing nodes, or local processing times at each node. Examples of the time-varying graph of Casteigts et al. are shown in Figure 2.10.

Our definitions of temporal graphs can not represent latency as defined above, but if one assumes null latency, one can trivially see how our definition of temporal graphs are conceptually similar to those of Casteigts et al. TVGs, where an edge, i.e. a triplet (x, y, t), corresponds to the presence function $\rho(x, y, t) = 1$.

Casteigts et al. also explain how this representation can be aggregated into sequences of graphs, which they call *footprints*, using sub-intervals, or what we call time-windows. A footprint of a TVG from a time t_1 to t_2 is defined as the static graph $G^{[t_1,t_2)} = (V, E^{[t_1,t_2)})$ such that $\forall e \in E, e \in E^{[t_1,t_2)} \iff \exists t \in [t_1, t_2), \rho(t) = 1$. Therefore a footprint $G = (V, E)$ of a time-varying graph $\mathcal{G}$ is similar to our def-



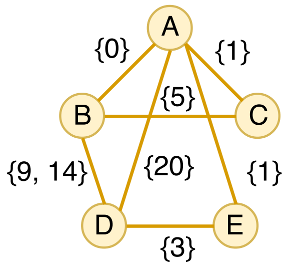


Figure 2.10: Two examples of graphs modeled using Casteigts et al. TVG definitions. The contacts from Figure 2.6 are modeled on the left, where the interval(s) on the edges indicate the times when each edge is available, i.e. $(t \in \mathcal{T} : \rho(e,t) = 1)$. The emails from Figure 2.8 are shown on the right, where the sets on the edges show when emails were sent between nodes.

inition of an accumulated graph (equation 10) except that footprints are static graphs instead of temporal. To represent what we have called a time-windowed graph, Cateigts et al. partition the lifetime $\mathcal{T}$ into consecutive sub-intervals $\tau = [t_0, t_1), [t_1, t_2) \cdots [t_i, t_{i+1})$, where each sub-interval $[t_k, t_{k+1})$ is also denoted τ_k. The *sequence of footprints* of the TVG $\mathcal{G}$ according to τ is then defined as the sequence: $SF(\tau) = G^{\tau_0}, G^{\tau_1}, \cdots$, which corresponds to how our time-windowed graphs are induced from a stream of data. While our definitions for temporal graphs overlap with those of time-varying graphs from Casteigts et al., theirs do not represent sequences of temporal graphs as needed in our work in [36, 37]. Furthermore, neither of the above works by Casteigts et al. nor Nicosia et al. discuss how to implement their representations efficiently, or how they scale with new edge- and/or node-insertions. Similar representations have been used in [3, 62, 63, 64, 65, 66, 67, 40, 68].

Lastly we discuss a unifying time-varying graph representation by Wehmuth et al. [1]. Their representation is based on the concept of MultiAspect Graphs (MAG), a graph generalization which is capable of representing multi-layer graphs, where time-varying graphs are a special case of MAGs [20, 69]. A MAG is an object $H = (A, E)$ where E is a set of edges and A is a finite list of sets, of which each set is called an “aspect”. In the time-varying graph, the list A contains two aspects, namely vertices and time instants, and is also referred to as $H = (V, E, T)$ where V is the set of nodes, T is the set of time instants, and $E \subseteq V \times T \times V \times T$ is the set of edges. They further denote $V(H)$ and $E(H)$ as the set of all nodes and edges in H, and $T(H)$ as all the time instants in H. An edge is defined as a

Time-instants (t_a, t_b)	(t_0, t_1)	(t_0, t_1)	(t_1, t_1)	(t_1, t_1)	(t_1, t_2)	(t_1, t_2)	(t_3, t_2)	(t_3, t_1)
Sender	D	C	D	C	B	C	B	A
Recipients	A	C	C	D	B	C	B	B

Table 2.2: Table of edges for Wehmuth time-varying graph example.

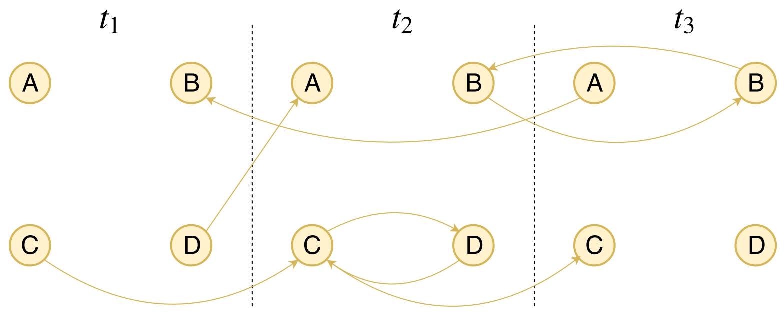


Figure 2.11: Wehmuth time-varying graph example with both progressive and regressive edges.

quadruplet $e = (u, t_a, v, t_b) \in E(H)$ where $u, v \in V$ are the origin and destination nodes, respectively, and $t_a, t_b \in T(H)$ are the origin and destination time-instants.

To illustrate the advantages of two time-instants in each edge, we present a small dataset consisting of 4 nodes shown in Table 2.2, and the corresponding Wehmuth et al. type of time-varying graphs based on Table 2.2 is shown in Figure 2.11.

As seen in Figure 2.12, the time-varying graphs of Wehmuth et al. are also capable of representing cyclic and periodic data by using so-called "regressive" edges where $t_b < t_a$, as opposed to progressive where $t_a < t_b$. Their time-varying graphs are shown to be computationally efficient and can be mapped to other more simple, or more useful for performing graph analysis, representations such as adjacency tensors, shown in Figure 2.11. The adjacency tensor of a TVG H is a 4th order tensor $\mathbb{A}(H)$ with dimension $|V(H)| \times |T(H)| \times |V(H)| \times |T(H)|$ that has a non-zero entry for every edge in H, and zero otherwise. As an example, the edge from D to A starting at t_1 and ending at t_2 is found on the 4th row and 5th column, with a value of 1. Note that the procedure used to generate the adjacency tensor matrix is a well-known matricization (or unfolding) of a tensor [70]. This is a reversible process and thus the original tensor can be ontained from its matrix form. Weimuth et al. furthermore show that the concepts of e.g. the snapshots of

						t_b							
		t_1				t_2				t_3			
		A	B	C	D	A	B	C	D	A	B	C	D
t_1	A	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	1	0	0	0	0	0
	D	0	0	0	0	1	0	0	0	0	0	0	0
t_a t_2	A	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	1	0	0
	C	0	0	0	0	0	0	0	1	0	0	1	0
	D	0	0	0	0	0	0	1	0	0	0	0	0
t_3	A	0	1	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	1	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.12: Matrix form of the adjacency tensor of the Wehmuth TVG in Table 2.2.

Nicosia et al., or the presence function of Casteigts et al., can be mapped to the time-varying graph of Wehmuth et al. [1]. The representations of Wehmuth et al. have been used in e.g. [71, 4, 72].

In our review we have omitted the following representations: Line graphs, transmission graphs, multi-layer networks, node-centric time lines, film clips, link-turnover graphs, influence graphs, concurrency graphs, difference graphs, memory networks [11, 12, 1, 2]. For additional literature on graph representations see [11, 12, 73, 8, 10].

2.4 Node Metrics

Every graph representation presents specific topological features which characterize the connectivity and dynamics of the underlying data. Efficient analysis of graphs, or nodes within the graph, often relies on measurements, or *metrics*, which are capable of capturing the most relevant features of the graph. A metric for a node can take on any form, but is typically a single numerical value which is the result of a more complex analysis of the graph structure around the node in question.

In this thesis we primarily use so-called *centrality* node metrics which can be useful for identifying important nodes in graphs in various ways. The concept of centrality in networks was first introduced by Alex Bavelas in the 1940s and was used for analysis of communication networks [74]. Bavelas' work was then later extended for connected and unconnected networks by Linton Freeman in 1978 [75]. These metrics allowed for identifying "important" or "central" nodes in social networks in an intuitive way, based on the shortest paths between nodes, and the distances between nodes, in a graph. Since these definitions of centrality metrics, many variants have been proposed (for an overview see [76]) and have successfully been used for problems such as within-network classification. In the following we review three examples of static node metrics: Betweenness centrality, closeness centrality, and clustering coefficient. Illustrations of the three metrics are shown in Figures 2.13, 2.14, and 2.15, where node colors range from blue (lowest value of metric) to red (highest value of metric).

Betweenness centrality is a node metric which describes the centrality of a node in a graph using shortest paths. It has been studied as a measure of how central a node is in regards to its "influence" on other nodes, for example in social networks [77]. Betweenness has been applied to a wide variety of network problems such as fault tolerance in electrical grids and finding influencers in social networks. The metric is defined as follows:

$$C_x^B = \begin{cases} \sum_{\substack{y \in V \\ y \neq x}} \sum_{\substack{z \in V \\ z \neq x \\ z \neq y}} \frac{\sigma_{yz}(x)}{\sigma_{yz}} & \text{if} \quad 0 < \sigma_{yz} \\ 0 & \text{otherwise} \end{cases}$$

where σ_{yz} is the number of shortest paths between node x and z, and $\sigma_{yz}(x)$ is the number of those paths that pass through x. This means, the more a node appears in shortest paths between pairs of nodes, the higher its centrality is. Figure 2.14 shows that only few nodes score high betweenness centrality since typically only few nodes act as "hubs" connecting large parts of the graph with each other.

Closeness centrality is also based on shortest paths in the graph, but in contrast to betweenness, is calculated using shortest paths from that node to every single other node in the graph. It is often regarded as a measure to quantify the node's "participation" in the graph, e.g. how much is a person communicating with others in an online social network [77]. Closeness centrality for a node x is calculated using the length of these shortest paths, and is defined as follows:

$$C_x^O = \begin{cases} \frac{|V|-1}{\sum_y d_{xy}} & \text{if} \quad 0 < \sum_y d_{xy} \\ 0 & \text{otherwise} \end{cases}$$

where d_{xy} is the length of the shortest path between x and y. As seen in Figure 2.14, the closeness centrality is lowest for nodes which are isolated and far away from most other nodes.

The **clustering coefficient**, or local clustering coefficient, measures how many edges are in the immediate neighborhood of a node, and can be used to determine how "tightly knit" a cluster of nodes are. Clustering for a node x is calculated by counting how many triangles are formed with x and is defined as:

$$C_x^U = \begin{cases} \frac{2T_x}{\deg_x(\deg_x - 1)} & \text{if} \quad 1 < \deg_x \\ 0 & \text{otherwise} \end{cases}$$

where $T_x(t)$ is the number of triangles through node x and $\deg_x(t)$ is the degree of x. The illustration in Figure 2.14 shows nodes with high clustering coefficient spread throughout the graph. In contrast to betweenness and closeness, clustering can be useful for identifying important members of smaller communities within the graph.

In the case of temporal or time-varying graphs there are many ways to extend these types of metrics. In this thesis we seek to explore some of these methods and propose new variants. Metrics in time-varying graphs are discussed more in depth in Chapters 5 and 6.

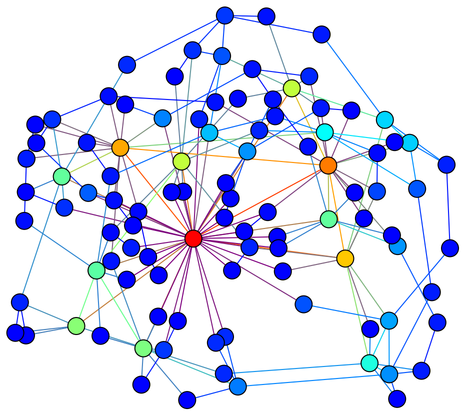


Figure 2.13: Betweenness centrality metric for nodes in graph. Blue nodes have the lowest betweenness centrality and red nodes have the highest.

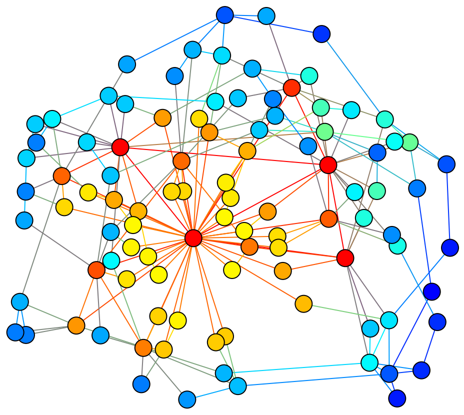


Figure 2.14: Closeness centrality metric for nodes in graph. Blue nodes have the lowest closeness centrality and red nodes have the highest.

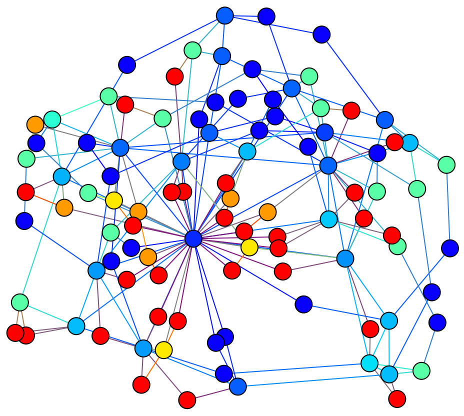


Figure 2.15: Clustering coefficient metric for nodes in graph. Blue nodes have the lowest clustering coefficient and red nodes have the highest.

3

Problems in Temporal Graphs

In this chapter we present a selection of six well-known problems in time-varying graphs as well as review related literature on the subjects. The six problems are within-network classification, node embedding, clustering, link prediction, anomaly detection, and node ranking. In this thesis we primarily attempt to solve the first two problems, while the remaining four are well-studied problems that are closely related to ours. Other problems which are not covered here include: Graph classification (of whole graph) [78, 79], subgraph mining [80], malware detection [81, 82], identifying influencers [83, 84], entity resolution [85], and network evolution [86].

3.1 Within-Network Classification

Classification of nodes, or within-network classification, is a problem in which the goal is to assign labels to nodes in the graph. Many definitions of the problem exist for static graphs, and it is a well-studied problem which has been applied to a plethora of domains [34, 87, 88, 89, 90, 91, 92, 28, 93, 79, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113].

A simple example of within-network classification is shown in Figure 3.1. Here four nodes have been assigned labels which take on values either 1 or 0, and the last node has no label, indicated by a question mark. The goal is then to assign either a 0 or 1 to the remaining node.

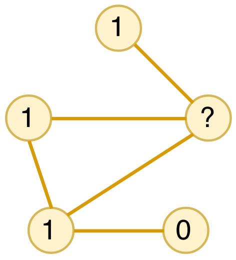


Figure 3.1: Example of within-network classification problem.

In a real-world scenario this could represent a criminal network, where each node is a person, and two people are connected if they have had contact by phone. Their label could then indicate whether or not the person has a criminal history, 1 if they do, and 0 if they do not.

Many approaches to this problem, for example in social sciences, exploit two important phenomena that can occur in graphs: homophily and co-citation regularity. Homophily is also informally known as "birds of a feather", and is when the existence of an edge between a pair of nodes (e.g. friendship or communication) is highly correlated with those two nodes being similar. For example, in online social networks, two people who are friends are more likely to share political beliefs. The second phenomenon is related, but applies to when similar nodes tend to connect to the same things. In these cases there are typically two sets of nodes in the graph (i.e. a bipartite graph) where, for example, one set represents humans and the other represents publications. Then, two authors who cite the same papers can be said to be similar, since their work closely relates to the same set of publications.

In some works, the classification is defined as a *statistical relational learning* problem, where methods have been shown to perform well because of their ability to utilize label dependencies of neighboring nodes [114, 115]. These methods assign labels to unlabeled nodes by propagating information (i.e. labels) from labeled nodes along the edges of the graph, to the unlabeled nodes. However, for graphs with sparse labels, this method can under-perform due to lack of information [90]. Within-network classification can also be viewed as a *semi-supervised learning* problem since both labeled and unlabeled data exist at training time [91]. Semi-supervised learning is not usually applied to relational data, but has been

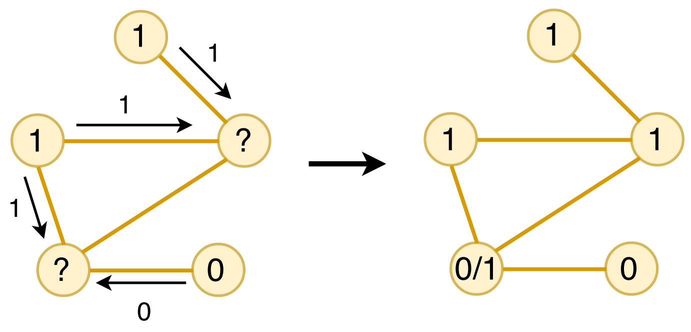


Figure 3.2: Example of within-network classification performed using wvRN. The arrows and numbers indicate the propagation of node labels to unlabeled nodes.

shown to have advantages such as less reliance on labeled training data, since they can utilize unlabeled nodes as well [92]. For more information on semi-supervised learning methods see work by Zhu et al. [105, 104].

Weighted-Vote Relational Neighbor (wvRN) is an example of a simple method which performs relational learning by label-propagation. Using wvRN, an unlabeled node is assigned a label-based on the distribution of labels of its neighbors, the label that is most common among its neighbors is also assigned to the node itself. An application of the procedure is illustrated in Figure 3.2. As seen in the graphs, the node labels travel along the edges to unlabeled nodes, resulting in the top-right node being assigned 1. However, the bottom-left node receives an equal number of 1's and 0's and therefore a consensus is not possible. In such cases the wvRN procedure can have trouble accurately classifying the node.

Unfortunately there is not much work on extending within-network classification to time-varying graphs [116, 117, 118, 86], and as with non-temporal graphs, there is no generally agreed upon formulation of the problem [119, 28, 120, 86, 121, 103]. Depending on the domain of application, and the choice of graph representation, the problem can take on many very different forms [122, 26, 123, 86, 124, 116, 125, 126, 117, 118, 127, 128, 129, 130, 131, 132, 133], and therefore we limit this section to work that is most relevant to the work done in Chapters 4, 5, and 6.

Aggarwal et al. propose a random-walk based method for within-network classification in time-varying graphs. A node's label is determined based on the distribution of labels of the nodes visited in a random walk starting from said node. Their approach combines both information about graph topology and node attributes by first creating a "pseudo-graph" where nodes are additionally connected based on shared attributes [127]. Kajdanowicz et al. implement an incremental algorithm for multiclass node classification in social networks which updates and adapts an already training classifier to new data in a mini-batched fashion [134]. They evaluate the performance on a problem where the goal is to predict the future emotional state of users of an online social network. Pei et al. use a dynamic factor graph model (dFGM) which attempts to capture three distinct correlations in dynamic graphs: a) correlation between node feature and node labels, correlation between neighbors' label and node label, and correlation of node label across two snapshots of the time-varying graph. Their method is evaluated on a subset of a co-citation network based on DBLP [135].

Günes et al. use a genetic algorithm, inspired by evolutionary processes in biology, called GA-TVRC-Het for evolving Heterogenous Networks. It uses a combination of iterative mutation/cross and relational classifiers to perform node classification in two real world networks [136]. Tagarelli et al. seek to enhance existing methods for time-varying graphs to better identify lurkers in online social networks. To do so they provide an in-depth analysis of lurker behavior and identify several temporal dynamics, such as information production and information consumption, that can be used for time-aware ranking to better classify nodes [137]. Yao et al. represent dynamic network data in a streaming fashion with continuous updates, for example edge-insertions, and propose a unified framework for classification tasks. Their framework uses methods from subgraph extraction, and online version of the Weisfeiler-Lehman graph kernel, and finally an application of kernel-based incremental learning [125]. Another streaming network approach by Yang et al. tackles the problem of active learning and classification in large networks. In addition to training a model on node labels in a graph, active learning allows querying the graph for additional labels of unlabeled nodes – at a cost. They propose a method which uses network sampling, and a active learning strategy which minimizes structural variability, to efficiently perform classification [124].

Metrics have been designed for time-varying graphs in order to better capture the evolution and structural changes. For example, Santoro et al. propose non-temporal and temporal metrics for time-windowed graphs [3] (originally from [2]), and Nicosia et al. extend centrality and clustering concepts from static graphs to time-varying graphs [61, 8]. Rossi and Neville propose a framework which uses temporal ensemble methods for discovering temporal representations of relational data. They are able to improve the accuracy of statistical relational learning algorithms, by a careful selection of graph components and temporal parameters (such as size of time-windows) [18]. Marginalized Gaussian Conditional Random Fields (m-GCRF) is a method for attributed weighted temporal graphs proposed by Stojanovic et al. which resembles the well-known method by Zhu et al., but extended to the temporal domain [104]. Their method can utilize both labeled and unlabeled parts of time-varying graphs and is shown to effectively predict future node labels on synthetic and real-world datasets [138]. In the work of Franzke et al. they attempt to find users of an online social network, who show tendencies towards affecting the performance of other users, in a positive or negative way. Their approach is to analyze how attributes describing an individual change over time, in combination with intervention analysis, to identify points in time when these change significantly, and then identify the candidates most likely to be responsible [83].

In the last decade deep neural networks have gained increasing attention, and have successfully been applied to a wide range of problems. However, only recently works have been published which deal with using (deep) neural networks directly with classification problems in time-varying graphs [118]. Most of these methods fall into these categories [120]: graph convolutional methods [139, 140, 141, 120, 142], graph neural methods [143], graph recurrent neural networks [144, 145], or graph reinforcement learning [146, 147]. Deep learning in graphs is a promising and fast-developing research field.

Additional surveys on methods and approaches to within-network classification can be found in [119, 28, 120, 86, 121, 103]. Finally, the work done in this thesis seeks to further knowledge on this topic for static and time-varying graphs.

3.2 Node Embedding

With the growth of available graph data, and the size of the graphs themselves, the space required to perform analysis often surpasses the available computation and storage resources. There are several ways to go about this challenge, and hence several problem definitions exist in literature with some overlap of nomenclature. A node embedding is a "succinct" and/or "efficient" representation of a node in a graph, that can more easily be processed, queried, and/or visualized [148]. For example, given a node in a time-varying graph, an embedding for this node could be a data structure which allows us to efficiently query the degree of the node at any time in that past.

In some works there is a sharp distinction between embeddings and representation learning, and graph summarization and compression. In these, embeddings or representations are generally described as low-dimensional vectors representing a node, part of the graph, or the whole graph, and are often tailored and evaluated as features for classification or prediction tasks in graphs [121]. The latter categories, summarization and compression, then primarily deal with the tasks of storing large graphs in concise forms that can more easily be visualized, processed, and managed, for example by designing efficient graph databases [148]. However because this distinction does not generally hold [149], and due to nomenclature used later in Chapter 7, we cover these topics broadly under the term "embeddings". Furthermore we review primarily with node-centric methods, also called node-embeddings.

In Figure 3.3 are two examples of node embeddings. These illustrate solutions to embedding with two distinct usecases. The top embedding can be used as a feature for within-network classification, for example identification of the most active nodes in a social network. The other embedding allows for efficient querying of which nodes are a distance of 2 away in the original graph, by simply viewing the current neighbors in the embedding.

Node embeddings for classification problems in static graphs have been widely studied [150, 96, 151, 152, 153, 154, 121, 155], with a recent increased interest in application to time-varying graphs as well [156, 129, 73, 157, 129, 152, 158, 149,

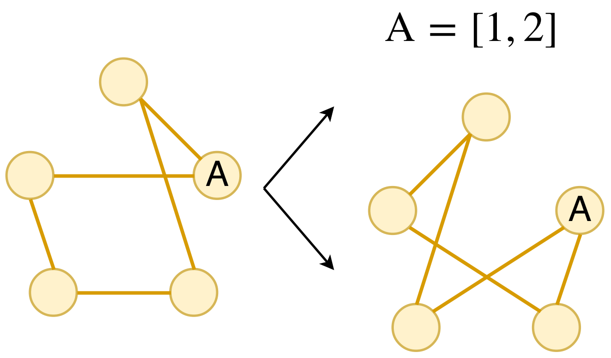


Figure 3.3: Two examples of embeddings based on the graph on the left. The top embedding is vector-based node embedding for node X, where the node's vector contains the degrees its neighbors. The bottom embedding is a copy of the original graph, where all edges have been removed, and then for each pair, an edge is added if they share a neighbor in the original graph on the left (similar to a 1-mode projection).

80, 159, 160, 154, 161]. For these types of node vector representations, a desirable quality is that two node embeddings are similar if the corresponding two nodes are "close" in the graph itself [121]. The difference in these types of embedding methods lies in how they define "closeness" (unrelated to closeness centrality) between the two nodes. Two commonly used metrics are first- and second-order proximity, which give an indication of the pairwise similarity between two nodes. For example, the method of Cao et al. creates embeddings for each node that captures structural information about neighbors up to k edges away [151], and the embeddings in both [158] and [162] consider two nodes to be "close" if they belong to the same community, and thus have similar embeddings.

Deep learning methods have also been shown to achieve high accuracy when used to create node embeddings for classification tasks, and are often based on random walks to sample parts of the graph [163, 164, 165, 96]. A well known example is DeepWalk by Perozzi et al. which uses a neural language model (SkipGram), originally designed to learn deep representations of words in text [166, 167, 168]. They then create sequences of nodes from repeated truncated random walks in the graph. These sequences make up the "sentences" for the SkipGram neural network to learn [96]. Many variants have since been proposed, modifying either the language model, how the random walks are performed, or both

[106, 155, 169, 165, 170]. Additional node embeddings for graph classification problems can be found in [164, 121, 171, 153].

Many types of *efficient* node embeddings for static graphs exist, and generally fall into three categories: aggregation-based, attribute-based, and compression [148]. The aggregation-based techniques create node-embeddings called "super-nodes" connected by "super-edges" and are useful in understanding complex data in an efficient manner [148, 172, 173, 174, 175]. Attribute-based methods must be aware of the fact that nodes in many real-world networks are assigned attributes, and take them into consideration [176, 177]. Compression embeddings are focused on minimizing the resulting size of the graph representation [178, 179, 180, 181]. This can allow for larger networks to remain in memory during analysis.

However, efficient node embeddings for time-varying graphs is a relatively new field [148]. Tsalouchidou et al. extend the idea of embeddings with reconstruction error from the static to the temporal domain using an approach they call tensor streaming [182]. As another example, Shah et al. have proposed a method called TimeCrunch for interpretable embeddings of large real-world datasets. They define a lexicon of temporal "phrases" which can be used to describe temporal connectivity patterns in time-varying graphs. They then employ the minimum description length principle to describe temporal graphs efficiently [56]. DiffNet by Seah et al. was designed specifically for biological networks to be able to create "differential" embeddings of graph snapshots [183].

Other methods have been designed specifically for graph streams, where edges and nodes arrive over time. The challenge here is that the embeddings need to be constructed in one pass (or very few) over the stream, and must be able to be updated incrementally for every incoming update to the graph. Zhao et al. proposed gSketch as a method to estimate edge frequencies. Their method is based on using a random projection, or hash-based technique, to compress the graph data into a smaller space, called sketches [184]. While allowing for fast and efficient estimation of edge frequencies, their method does not retain the underlying graph, and can therefore not be used for applications which utilize the graph structure [185].

Since their initial work, more advanced sketches, extending on the work of [185], have been proposed in [186] and [187]. Tang et al. present a method, TCM, which can embed information about streamed nodes and edges in constant time using a two-dimensional graphical sketch [186]. Khan et al. propose a query-friendly method called gMatrix which retains structural and frequency properties of the underlying graph data. They implement a three-dimensional sketch which can perform the embedding in real-time, and evaluate their method using large-scale real-world networks [187].

Additional surveys on methods and approaches to node embeddings can be found in [188, 187, 148, 73, 161]. Finally, the work done in this thesis, specifically Chapter 7, seeks to further knowledge on node embeddings for time-varying graphs.

3.3 Clustering

There does not exist a universally accepted definition of clustering in graphs. Generally, clustering is the task of grouping nodes into clusters of nodes which are similar, or well-connected, by some predefined metric. In other terms, the goal of clustering is to create groups, or clusters, of nodes within the graph in such a way that each node in a cluster is “similar” to each other node in the same cluster, and “dissimilar” to nodes belonging to other clusters. Clustering falls into the category of *unsupervised learning* problems because it deals with finding structure in data without labels (i.e. node labels).

A common way to cluster nodes is by taking into account the local node topology in such a way that there, informally, exist many edges between nodes within each cluster, but few edges between the clusters, also referred to as modularity optimization [189]. Other ways to measure cluster quality include measuring cut sizes or internal vs. inter-cluster densities [190]. A small example is shown below in Figure 3.4. Here a graph consisting of 14 nodes is clustered into three groups, each with their own color. In this case one can see that each node is “well connected’ to the other nodes in its own cluster, but few connections exist between clusters.

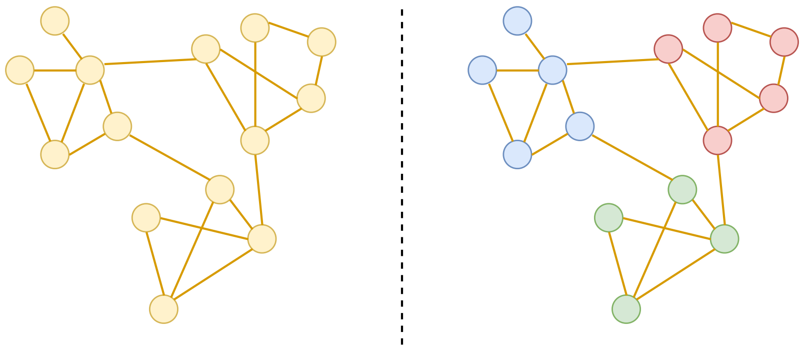


Figure 3.4: Example of clustering in graphs. On the left is the original graph, and on the right is the graph when clustered into three groups of nodes. The clustering performed here is an example of modularity clustering, hence there are many edges between nodes within the clusters, and few between clusters.

When the graph evolves over time, so must these clusters. Rossetti et al. describe clustering in time-varying graphs as a generic problem with the name "Dynamic Community Discovery" [191]. It is defined as follows: A "dynamic community" is a set of distinct pairs of nodes and lists of time-windows, $DC = \{(v_1, P_1), (v_2, P_2), \cdots, (v_n, P_n)\}$, where $P_n = ((t_0, t'_0), (t_1, t'_1), \cdots, (t_N, t'_N))$, with $t_i \leq t'_i$, is the set of time-windows indicating which periods of time node n is a member of the dynamic community. The challenge of dynamic community discovery is then identifying the set $\mathcal{C}$ of all dynamic communities in the time-varying graph. These communities may be non-overlapping as well as overlapping.

Several authors have made surveys categorizing methods for clustering in temporal networks [192, 193, 194]. They observe that methods generally fall into three categories depending on how communities are discovered in the time-varying graph. Using the nomenclature from Rossetti et al. [191], the three categories are: Instant-optimal, Temporal Trade-off, and Cross-Time. Due to the many types of clustering methods, we limit this section to a selection of methods which fall into the aforementioned categories.

The first method only considers the graph at its "current" state, and performs clustering without regard for the evolution of the graph or clusters that have been identified at a previous time-step. The advantages of instant-optimal clustering is that it builds directly on top of results from clustering in static graphs. This means that already proven efficient static methods can be applied at each time-step independently of each other, which furthermore allows for parallelization. As an example, Salleberry et al. create so-called “time-clusters” for each graph in sequences of graphs, which then are merged over time [195]. However, since the clusters of each time-step are independent of each other, they might change significantly from time-step to time-step, leading to instability. Recent approaches have attempted to compensate for this weakness e.g. by only considering the most “stable” parts of communities, called community cores, over time [196, 197].

A perhaps more natural approach to the clustering problem is the temporal trade-off, where nodes are clustered at each time-step with the arrival of new data, but where the clusters of the previous time-step(s) also are taken into account. An evolutionary clustering method by Chakrabarti et al. set forth two objectives for this type of algorithm: 1) the assigned clusters of any time-step should accurately reflect the graph at that time 2) clusters of any time-step should be similar to those of the previous time-step [198]. Another incremental clustering method, *DENGRAPH*, uses concepts from multidimensional density-based clustering methods such as DBSCAN [199]. This strategy mitigates some of the weaknesses of instant-optimal solutions, such as instability. However, with this approach there is naturally a trade-off of maintaining the results from previous time-steps versus adapting to newly formed edges and nodes, and in contrast to instant-optimal methods, these are typically not easy to parallelize [191]. Furthermore, these methods are susceptible to cluster drift, where clusters found and modified after many time-steps differ significantly from what a static clustering method would identify.

Cross-time methods do not consider the individual time-steps of a time-varying graph, but instead in a single process, consider all time-steps and states at the same time. Many of the methods that fall into this category rely on transforming the time-varying graph (i.e. a sequence of graphs) to a single graph, in such a way as to preserve temporal information. Then clustering is performed using methods for (potentially complex) graphs. As an example Matias et al. use a stochastic block models approach where the number of communities and community density is fixed for the entire process [200]. These methods are well-suited to avoid the cluster drift and instability of above methods since the time-varying graph is processed just once. However these methods typically require assumptions about the domain of application (e.g. number of clusters a priori), and cannot be used with streams of data [191].

Additional surveys can be found in [189, 201, 202, 203, 190] for static graphs, and in [193, 194, 191, 204, 205, 195, 192] for time-varying graphs.

3.4 Link Prediction

Link prediction is a fundamental problem in graphs that has received much attention [206, 207, 208, 209, 210, 211, 212, 213, 214]. The goal of link prediction is to estimate the likelihood of the existence of an edge between any pair of nodes, based on already existing edges, and e.g. the attributes or weights of nodes [215]. This can be useful in biological networks, such as protein-protein interaction networks, where distinguishing whether or not an edge between two proteins exists must be demonstrated by field work or real-life experiments, which are usually very costly and can take a long time [216]. Besides assisting in finding missing connections such as these, link prediction can also be used to predict future edges in time-varying graphs. This can be useful in e.g. online social networks for recommending friends who you know, but have not yet connected with [206]. The problem of link prediction is illustrated in Figure 3.5.

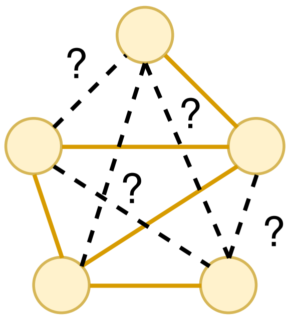


Figure 3.5: Example of link prediction where dotted lines indicate possible new edges. The goal of link prediction is then to assign probabilities each of these edges.

There are many methods to estimate the probabilities of edges, but many fall into one of two categories: similarity-based methods, and probabilistic methods [217]. Similarity-based methods assume that nodes tend to connect to other nodes which are similar (i.e. homophily) and define a function which assigns a similarity score for every pair of nodes in the graph. These functions can take many forms, some use only local structural information, such as counting the number of shared neighbors [206, 209, 218], while others use the whole graph structure like the negated length of shortest paths [219, 220, 221]. Meanwhile the probabilistic methods usually assume that the network has a known structure, and then build a model that estimate its parameters using statistical methods to fit the structure. After fitting, the model parameters can then be used to compute the probability of each non-observed edge [217]. Examples of models include the hierarchical structure model, which assumes that many real networks are hierarchically organized [222], the stochastic block model, in which nodes are distributed in communities or "blocks" [223], or the cycle formation model, in which the assumption is that networks have a tendency to close cycles in their formation processes [224].

Additional surveys can be found in [217, 215].

3.5 Anomaly Detection

Anomaly detection in graphs is a problem dealing with finding data, e.g. edges or nodes, that "stand out" compared to the rest of the graph. The problem has many real-world applications in domains such as finance, health care, law enforcement, and fraud detection. For example, in health care, anomaly detection can help reveal rare events such as disease outbreak or side effects which can be of vital importance for medical diagnosis [225]. An example of anomaly detection is shown in Figure 3.6.

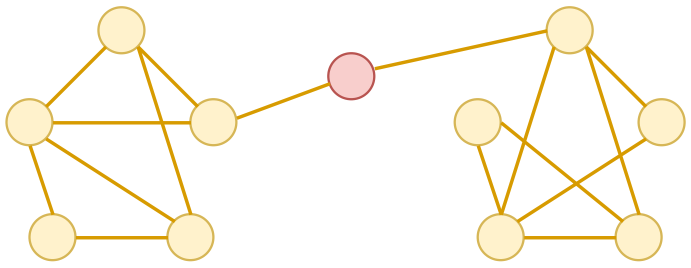


Figure 3.6: Example of anomaly detection. Two tightly-knit communities of nodes are formed on the left and right, however one node (red) is connected to both, and can be described as an anomaly.

There are several very comprehensive survey articles on anomaly detection in static and dynamic graphs, and large amount of work has been done in these subjects [226, 227, 228, 229, 230, 221, 231, 160, 232, 233, 234]. The existing methods for detecting anomalous nodes generally belong to one of three categories: Structure-based, community-based, or relational learning based [225]. The main idea behind structure-based methods is to capture what the structure around each node "normally" looks like, using graph metrics such as centralities and clustering coefficients, and then look for nodes which features deviate from the norm. An example is the technique called OddBall by Akoglu et al. which constructs features based on the local neighborhood, or sub-graphs, around each node, and then discovers which patterns are most common in the graph [235]. Several works have since extended and improved upon the techniques of OddBall [232, 233, 234]. Other approaches in this category can e.g. be based on random walks to create similarity measures [236, 206, 237].

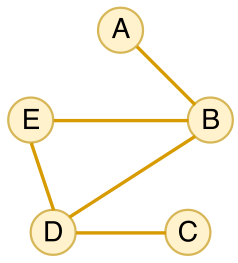
In community-based the technique for finding anomalies is closely related to clustering. These methods first divide the graph into closely connected (or "tightly knit") groups of nodes, and then attempt to find nodes and/or edges which have connections across communities. Example include *gskeletonclu* by Sun et al., which uses random-walk-with-restart based scores to find nodes which belong to more than one community [238]. Another approach by Chakrabarti et al. called AutoPart, is based on the assumption that nodes with similar neighborhoods also are clustered together in the graph. Therefore, any edges that do not belong to any of these clusters constitute an anomaly [239].

The works in the last category, relational learning based methods, exploit the relationships between nodes to assign classes to them: anomalous and normal. This is very similar to the node binary classification task described in Section 3.1. Generally relational learning methods which can exploit sparsely labeled networks, such as the "Ghost-Edges" approach by Gallagher et al. [91], can achieve high accuracy, since the number of labeled nodes in real world anomaly detection networks is typically low [225].

Additional surveys on anomaly detection can be found in [230, 225, 226, 240, 241].

3.6 Ranking

Ranking, also called link-based object ranking, is a well-known problem where the objective is to exploit the graph structure in order to rank, or prioritize, nodes within the graph. In the general case the nodes can be from any set of objects, but for most the graph and nodes represent a single object type and single edge type [95]. Ranking has found use in many applied domains such as web information retrieval, academic authorship, and sports analysis [242, 243, 244, 137, 245, 246, 242, 247, 248, 249, 214, 250, 251, 252]. An example node ranking is shown in Figure 3.7.



Rank	Node	Score
1	B	3
2	D	3
3	E	2
4	A	1
5	C	1

Figure 3.7: Example of node ranking. The nodes have been ranked in a list based on their individual score. The score in this case is simply the degree of each node.

For web information retrieval a well known method is the PageRank algorithm proposed by Page et al. which is currently in use by `www.google.com` to rank web pages online [243]. PageRank is designed to perform random walks in a graph where nodes are web pages connected by edges if there are links between them. The resulting rank of a node is the fraction of time that PageRank spends at that node in its random walks. A slightly more complex process called HITS also aims to rank web pages, but can also be used to classify web pages into two categories *Hubs* and *Authorities* [253]

Park and Newman prose a static ranking system for sports where players (represented as nodes) increase in rank or decrease in rank depending on their performance against each other [254]. Motegi et al. extend the concept of win-lose score to temporal graphs to account for the fact that nodes' ranking can fluctuate over time. They do this by implementing a temporal decay function which diminishes the contributions to node's rank over time as the contributions get older [242].

In contrast to ranking in static graphs, ranking in time-varying graphs seeks to track the changes in ranking over time as events (such as soccer matches) unfold [95]. O'Madadhain et al. have proposed a series of desired properties solutions to ranking in time-varying graphs, ad themselves introduce a framework for ranking based on "potential flow" that satisfies these specified requirements [255, 256].

Additional surveys on node ranking can be found in [257, 95, 242].

4 Classification using label-independent features and latent linkages

In this chapter we present work that has been published in [35] at the 12th International Workshop on Mining and Learning with Graphs (MLG). I originally wanted to solve within-network classification in pure static graphs without any specifics in mind. Through work and literature I saw that there was a growing availability of networks with rich node attributes such as age, profession, nationality etc. I was then inspired by the way in which "ghost-edges" were added to graphs in the work of Brian Gallagher [91] and thought to work with this idea of increasing connectivity between nodes to improve performance of simple classifiers.

4 Classification using label-independent features and latent linkages

Abstract

We study within-network classification in sparsely labeled networks, presenting two separate contributions: (A) a thorough reproduction of a label-independent method for classification from recent research using statistical relational learning (SRL) and semi-supervised learning (SSL), and (B) a novel approach that utilizes node attribute information to improve SRL and SSL classifier performance called Attribute Network Propagation (ANP). (B) uses a method of linearly combining predictions with a procedure of transforming node attributes into graph edges. For both contributions we use two well-known real-world datasets: the reality mining (RM) cell phone calls dataset and the Cora Portal publication citations dataset (CORA), both formulated as binary classification problems. We employ an existing classification framework to run 10 individual SRL- and SSL-based classifiers and evaluate performance using the area under the ROC curve (AUC). Results from (A) confirms that label-independent features can improve the performance of some relational classifiers using iterative methods, but in most cases, 26 out of 30, deteriorates performance on existing baselines. Results from (B) show that in 91 out of 100 cases it is possible to improve the performance of relational classifiers with ANP.

4.1 Introduction

Numerous problems concerning real-world phenomena involve classification of nodes in networks, for instance:

- Anomaly detection: detecting intrusions in networks based on network traffic. Identified intrusions make up for a small fraction of all traffic in and out.
- Social Network Analysis: Categorising and modelling user behaviour to be used in e.g. targeted advertising or anti-terrorism operations. Typically contains large amounts of unlabeled data and a small set of labeled nodes.
- Cell phone fraud: Cell phone fraud is an example where networks are often very sparsely labeled. We have a handful of known fraudsters and legitimate users, but the labels are unknown for the vast majority of users.

Other examples include classification of documents and webpages, protein interactions, and product recommendation systems.

More specifically, the problem of (univariate) within-network classification is the following: *given* a quadruple $G(V, E, W, L)$ where (V, E) is a (possibly directed) graph, a set of labels W, and a subset $L \subseteq V$ such that each node $v \in L$ has one or more known labels from W, *find* labels from W for each of the nodes in $V \setminus L$. Typically, the labels are sparse (i.e., L is small compared to V, for large real-world networks, often $|L|/|V| \leq 1\%$).

To tackle this problem, several modern approaches try to augment the traditional node-centric approach (using only information directly attributed to the individual nodes) with *relational* data. Relational data differs from traditional data in an important way: it violates the instance-independence assumption. The core concept of relational methods is to take advantage of these dependencies between instances. Statistical relational learning (SRL) algorithms have been shown to perform well on within-network classification problems [98], especially when two phenomena are present in the data: homophily and/or co-citation regularity. Homophily is the correlation between two connected nodes and their individual labels. Co-citation regularity is related and holds true when individuals have a tendency to connect to the same objects in networks. Under these circumstances, the label-propagation algorithms can assign labels throughout the network, using edges between nodes as ”pathways”, to successfully classify previously unlabeled nodes[97]. However because of the reliance on existing ground truths, relational classifiers can degrade significantly when labels are sparse.

Recent research has proposed various methods to tackle the label sparsity. Collective classification methods have been shown to improve performance when labels are sparse[100, 258]. Other methods synthetically construct additional relational data, e.g. directly adding new edges to the graphs [91], and other approaches employ more complex ways of combining various attributes and graph structural properties, called latent graphs, like in [258, 259]. The idea of latent graph methods is to use existing (node) attributes in graph datasets in order to construct new – so-called *latent* – features or graph augmentations in the underlying graph; with the new features, a supervised classifier can then predict class membership for remaining unlabeled nodes [87, 258, 89].

For instance, Tang uses spectral clustering and modularity maximization [87, 89] to generate a new set of latent features for each node. Another example, the Latent Network Propagation (LNP) algorithm by Shi et al. [258], transforms the original graph by adding weighted edges, where the weights are determined by factors such as attribute similarity or node proximity in the original graph. The LNP algorithm uses quadratic programming to add weight to edges between nodes that are likely to share the same class label. Such indirect edges are called ”latent linkages” and are already a part of the dataset, but need processing before becoming part of the graph structure.

A particularly simple version of this approach is simply to generate new edges in the underlying graph; using the augmented graph, with the new edges, a label propagation algorithm makes its final class membership predictions. The idea is that adding edges in this manner allows the ground truths to propagate more effectively throughout the graph[260]. By careful design and choice of latent edges, one can improve classification reliability without the need for relational features or more complex collective inference.

Our contribution: We extend the idea of using latent linkages by creating new graphs with edges based on attribute similarity, but running inference separately, on both the original graph and the newly constructed latent attribute graph, and finally combining predictions. We also experiment with a simpler method, merging latent attribute features directly into the original graph, to test whether the added complexity of the first approach provides any improvement of classification.

4.1.1 Related Work

Traditional statistical relational techniques have made use of label-dependent features. Lu and Getoor’s approach uses logistic regression to model class membership using neighboring nodes [261], Macskassy and Provost use weighted sums of neighboring nodes’ labels [98] for label propagation, and Neville and Jensen [262] use spectral clustering to group nodes based on their local edge structure, which in turn are used in learning classifiers.

Handling label-sparsity has been extensively studied: one approach is to use an iterative procedure, Iterative Classification Algorithm (ICA), that feeds predictions back into the network, which in turn are used to inform subsequent inferences as done by Neville in [99]. ICA is reported as a somewhat robust method which can be governed by a process called Gibbs sampling [261, 263]. Collective classification has also been proposed to overcome this problem [263, 100], where they key idea is to combine supervision knowledge about the graph with edge-structural information from the graph. However, previous work has shown that even collective classification can suffer when subjected to very sparsely labeled graphs [114].

Another approach aims to incorporate extra information, e.g. attribute information, already present in the data into the graph, to strengthen classification by providing more edges for information to travel through. Gallagher et al. propose a method where "ghost" edges are added to the original network to enable flow of information to hard-to-reach unlabeled nodes. A similar design by Macskassy [98] adds additional edges to the graph based on text-similarity between nodes in the data. Shi et al. [258] transform the dataset into a fully connected graph with latent edges, where edge weights are maximized between training data with the same labels. Tang and Liu [87] extract social latent dimensions, like affiliations, combined with discriminative learning to outperform relational collective classification methods. Finally work by Fleming [264] and McDowell et al. [260] compared latent edge methods with state-of-the-art non-latent methods and confirmed that these types of algorithms can perform competitively, but don't necessarily perform consistently. Our proposed method differs from previous approaches by keeping latent edges in separate graphs instead of adding them to the existing edges.

4.2 Methodology

To address the problem of label sparsity we propose two methods: **Attribute Network Propagation (ANP)** and **MixedEdges (ME)** that exploit latent linkages between nodes based on attribute information already present in data. For example, in a publication dataset like CORA, in addition to citations between articles we also have a list of keywords present in each of the articles' content. It can be expected that nearly all of these keywords appear multiple times across several papers. By adding edges between articles, that share keywords, it is possible to derive latent linkages in the dataset. The intuition behind this idea is that with the added edges, the ground truths may be able to propagate throughout the network more effectively. The problem lies in selecting the most useful shared attributes to transform into edges and in what way they are best utilized by relational classifiers.

The two methods require generation of two types attribute-augmented graphs, which we name after the dataset they are based on, with added suffixes: **-ANP** or **-ME**. The -ANP graphs are used by Attribute Network Propagation and the -ME graphs are used by MixedEdges.

The -ANP graphs are created by first making a copy of the original graph, without edges, from the original dataset. The -ME graphs are based completely in the original graph, nodes and edges. Then, for both -ANP and -ME graphs, unweighted, undirected edges are added between nodes depending on dataset-specific conditions, explained in detail in Section 4.4. These conditions are derived from available attribute information in the individual datasets.

For -ME sets all added edges are merged into the original graph, but for the ANP algorithm, the two sets of graphs (original and -ANP) remain separate. Informally, the nodes of the -ANP graph are only connected based on their shared attributes and the nodes of the -ME graph are connected using both shared attributes and the original edges. As described earlier in this section, the motivation behind choosing these attributes is the assumption that they connect unlabeled nodes to more labeled nodes than in the original graph (ME) or that attributes edges alone can uncover more meaningful paths for labels to propagate. The attribute graphs -ANP are each based on multiple attributes, which alternatively could be split into separate graphs, one for each attribute. This was decided against due to very low edge counts, $(\leq 100$ edges$)$ for most attributes, which we expect would lead to a decrease in performance of label-propagation methods.

For the **ME**, node class membership is determined simply by using relational classifiers directly on the -ME graphs. For **ANP** we first run the relational classifiers on both the original and the -ANP graphs individually. After, ANP combines

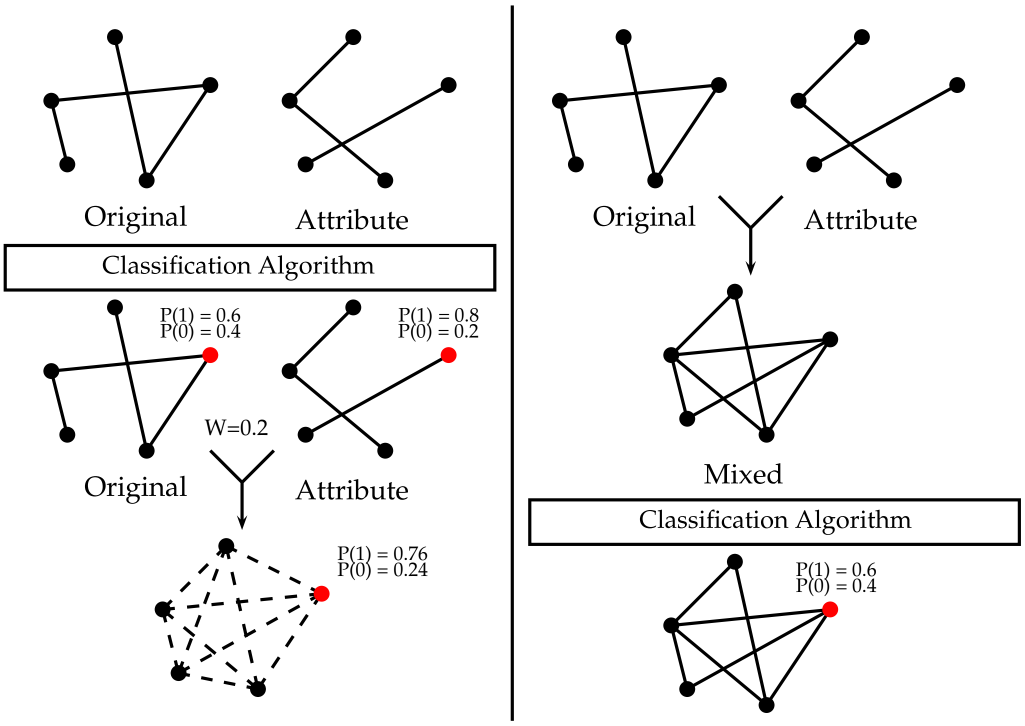


Figure 4.1: Methods of classification using attribute edges. On the left are is the ANP method using and on the right is the ME method. The red node is an arbitrary node in the graph, used to show the process of calculating its class membership $P(C)$. W is the pre-calculated weight using 10-fold cross-validation.

the results from -ANP sets with the original graphs in a fashion resembling how the individual classifiers are combined in Section 4.3.2. The final prediction of each node's class is linear combination of results on the original dataset and the shared attribute graph:

$$P(C) = w \cdot P_{baseline}^{original}(C) + (1 - w) \cdot P_{baseline}^{-ANP}(C) \tag{19}$$

Similar to equation 22 the weight w is calculated based on the individual performance of the baseline over 10-fold cross-validation on the original dataset and the -ANP dataset. The area under the receiver operating characteristic ROC curve (AUC) is calculated for each fold and then an average AUC score for each dataset, $AUC_{baseline}^{original}$ and $AUC_{baseline}^{-ANP}$, is obtained. The weighting parameter w is then defined as:

$$w = \frac{AUC_{baseline}^{original}}{AUC_{baseline}^{original} + AUC_{baseline}^{-ANP}} \tag{20}$$

The classification processes, ANP and ME on their respective graphs, -ANP and -ME, is shown in Figure 4.1.

4.3 Experimental Design

In the following we first present details about the classifiers chosen and how label-independent features are used. Following is a description of the real-world datasets, their characteristics and how they are sampled. The last part explains the experiment methodology in detail.

4.3.1 Label-Independent Features

Relational classifiers typically rely on the network edge-structure to make use of label information from neighbouring nodes. Another approach to creating relational features is to use the graph-structural properties of the surrounding nodes, such as node degree or other graph-metrics. The methods of creating relational features can be divided into two categories, label-dependent features and label-independent features.

An example using label-dependent features is the network-only edge-based classifier (nLB) from Lu et al. [261]. It models a node's class based on the classes of neighbouring nodes. A node's neighbourhood is summarized by edge-weighted counts of neighbouring nodes for each class. One disadvantage to this approach is that when labels are sparse, the relational features become useful since neighbouring nodes will tend not to be labeled.

Label-independent (LI) features are calculated using only the structural properties of the graph – labels and attributes are not included. The hypothesis is there is a correlation between the class label of a node and the network structure and that they can be used in cases where label-sparsity impedes traditional relational features. However, results have shown that label-independent features alone are not enough to model class membership in the graph, therefore additional information is needed. From the design of Gallagher & Eliassi-Rad [92] we use a logistic regression classifier (logLI) that is trained on the following four label-independent node features: number of neighbouring nodes (node degree), number of incident edges, the betweenness centrality coefficient, and the clustering coefficient. The results using label-independent features are then combined with predictions from baseline classifiers, described in detail in Section 4.3.2.

4.3.2 Classifiers

The work is based on 10 individual classifiers, some of which are hybrid methods combining results from multiple algorithms. In order to use the label-independent features of [92] we combine baseline classifiers with the logistic regression model ″logLI″, trained on the LI features, from Section 4.3.1. We use the implementations in NetKit, a modular toolkit for classification in networked data by Macskassy et al.[98]. The full list of methods used is as follows:

- wvRN: The weighted-vote relational neighbor classifier which is a non-learning classifier that uses label-propagation and an average weighted sum of neighboring nodes' labels to determine class membership[98]. The wvRN classifier calculates the class membership of a node i as:

$$P(C_i = c|N) = \frac{1}{L_i}\sum_{j\in N}\begin{cases} w_{i,j} \text{ if } C_i = c \\ 0 \text{ otherwise}\end{cases} \tag{21}$$

  where N is set of neighbors for node i, $w_{i,j}$ is the number of edges between node i and j and L_i is the number edges connecting i to labeled nodes.

- wvRNICA: Uses the wvRN classifier with collective classification as described in Section 4.3.3.

- wvRN+li: Is a combination of wvRN and the logLI logistic regression classifier, described in Section 4.3.1.

- wvRNICA+li: combination of wvRNICA and logLI.

- nLB: The network-only edge-based classifier [261]. Uses a logistic regression model to infer a node's class. A node's summary is the number of neighboring nodes. A node's features are neighboring nodes' summaries.

- nLBICA: Uses the nLB classifier with collective classification as described in Section 4.3.3.

- nLB+li: Is a combination of nLB and logLI, described in Section 4.3.1.

- nLBICA+li: Is a combination of nLBICA and logLI, described in Section 4.3.1.
- GRF: Is the semi-supervised Gaussian Random Field classifier of Zhu et al.[104].
- GRF+li: Is a combination of GRF and logLI, described in Section 4.3.1.

The methods which use logLI to improve predictions, calculate the probability of each class as:

$$P(C) = w \cdot P_{baseline}(C) + (1 - w) \cdot P_{logLI}(C) \tag{22}$$

where $P_{baseline}(C)$ is the class membership prediction of the baseline, i.e. wvRN, nLB, GRF, and $P_{logLI}(C)$ is the prediction from logLI. The weight w is calculated once per dataset and is used throughout all trials for that specific graph. The calculation is based on the individual performance of the baseline algorithm and logLI over 10-fold cross-validation on the labeled nodes of the dataset. The area under the receiver operating characteristic (ROC) curve (AUC) is calculated for each fold and an average AUC score for each classifier, $AUC_{baseline}$ and AUC_{logLI}, is obtained. The weighting parameter w is then defined as:

$$w = \frac{AUC_{baseline}}{AUC_{baseline} + AUC_{logLI}} \tag{23}$$

thereby enabling utilization of label-independent features. The intuition is that the label-independent features should be used to a degree based on their expected performance.

4.3.3 Collective Classification

To perform collective classification (CC) we use the Iterative Classification Algorithm (ICA) up to 1000 iterations with both the nLB (nLBICA) and wvRN (wvRNICA). We chose the Iterative Classification Algorithm over other methods, eg. Gibbs Sampling, to follow the methodology described in [92]. For partially labeled datasets, there are two approaches when using ICA: Perform collective classification on the entire graph or perform collective classification only on the core set of nodes. The latter is chosen because it is has been shown to perform better[92].

4.3.4 Datasets

We employ data from two sources, chosen for their appearance in the research papers that inspired this article: Reality Mining (RM) cell phone calls/texts[1] and CORA Research Paper Classification Dataset.[2]. CORA is a co-citation dataset originally created by crawling the internet for machine learning articles [265]. Shi et al. use the dataset to perform multi-class classification using latent graphs [258]. The publications themselves correspond to nodes in the graph and citations correspond to edges between publications. Each node is categorized into one of seven classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory. To pose a binary classification problem in the context of the CORA dataset, we formulate the classifiers' task as: identify papers with the topic ″Probabilistic Methods″.

The CORA data is already cleaned and pre-processed by [100]. Each publication in the dataset is represented by a label (machine learning topic) and a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words and there are 7 topics as the set of labels.

The RM data originates from an experiment at a college campus where 100 mobile phones were tracked over the course of 9 months [33]. Gallagher et al. used the collected data to create a graph where each user is considered a node and cell phone communications are considered edges between users [92]. They used the graph to evaluate a classification method using label-independent features. For the Reality Mining data our task is to identify whether or not a person is a student. In the RM graph there exists a subset of nodes, we call ″core″ nodes, for which we known the true class labels. For the rest of the RM nodes there exists no true class labels.

With RM we remove from the dataset any participants who do not communicate with other individuals. The resulting graph is then sampled using breadth first search, as done in [92], following edges based on communication. The pseudocode for the procedure is show in algorithm 1. Nodes are sampled using BFS

[1]http://realitycommons.media.mit.edu/realitymining.html
[2]http://linqs.umiacs.umd.edu/projects/projects/lbc/

Algorithm 1: BFS Dataset Sampling

```
input: Graph $G_i(E_i, V_i)$, SampleSize
output: Graph $G_o(E_o, V_o)$
begin
    nodequeue $\leftarrow$ getCoreNode($V_i$)
    $G_o(E_o, V_o) = \emptyset$
    while notEmpty(nodequeue) and $|V_o| <$ SampleSize
        $v_i \leftarrow$ pop(nodequeue)
        $V_o \leftarrow V_o \cup v_i$
        nodequeue $\leftarrow$ nodequeue $\cup$ (getNeighbors($v_i$) $\setminus V_o$)
        if $|V_o| ==$ SampleSize
            break
    end
    foreach $v_o$ in $V_o$
        $E_o \leftarrow E_o \cup$ allEdges($v_o, V_o$)
    return $G_o(E_o, V_o)$
end
```

Table 4.1: Dataset characteristics

Data Set	$\|V\|$	$\|E\|$	$\|L\|$	$P(+)$
RM Full	10.050	140.703	101	0.63
RM Samples	1K	16k-42k	18-92	0.04-1.00
RM-ANP Samples	1K	100-1800	18-92	0.04-1.00
RM-ME Samples	1K	16k-44k	18-92	0.04-1.00
CORA	2708	5429	2708	0.16
CORA-ANP	2708	220k	2708	0.16
CORA-ME	2708	225k	2708	0.16

starting from a core node. When 1000 nodes have been chosen, all edges present between them are added to the sample. The breadth-first sampling of the RM datasets in algorithm 1 is sensitive to which core node is chosen as the seed. To overcome this we create $|\text{core nodes}| = |L|$ number of datasets $G_o^i(E_o^i, V_o^i)$ for RM, one for each possible core node seed. The order of RM seed core nodes is unimportant since the experiments are run on the resulting samples independently of each other.

Table 4.1 contains the following information about the original and sampled datasets from RM as well as the CORA co-citation network: $|V|$ is the total number of nodes in the dataset, $|L|$ is the number of labeled nodes, $|E|$ is the number of edges and $P(+)$ is the fraction of labeled nodes which have a positive class label. Since the RM dataset is sampled multiple times, we give intervals in which the individual characteristics lie. The sets with suffix '-ANP' and '-ME' are datasets based on latent attribute graphs described in Section 4.2.

Algorithm 2: RM -ANP -ME Experiment Setup

```
input: Graphs $G_o^i(E_o^i, V_o^i)$
output: AUC
begin
for graph $G_i(E_i, V_i)$ in $G_o^i(E_o^i, V_o^i)$
for $c$ in classifiers
10-fold cross-validation on $G_i$ using $c$
Calculate $w$ from averaged AUC
for p in proportion {0.1,0.3,0.5,0.7,0.9}
for trial in 30 trials:
testset = getClassStratifiedSampling
trainingset = $G$\testset
Initialize uniform priors
Run classifiers using trainingset labels
Use weight $w$ to combine predictions
Calculate AUCs for trial
end
Average AUC per ratio for all trials
end
end
return avg. AUC per ratio for all graphs $G_o^i$
```

4.4 Experimental Methodology

Classifiers have access to the entire graph during both training and evaluation, but in the experiments we hide a proportion of the labeled nodes' labels, so they can be used for evaluation. The proportion of labeled nodes used for training is varied from 0.1 to 0.9 in 0.2 size increments. For each proportion we run 30 *trials* with each classifier. For each trial we perform a class-stratified sampling containing $100 \times (\text{proportion labeled})\%$ nodes as a training set and the remaining as a test set. The samples are chosen carefully so that each node appears the same amount of test sets over all trials. The train/test splits are the same for all classifiers and across the -ME -ANP types as well. The experiment setup described is repeated for each graph G_o^i sampled of RM. The process is shown in algorithm 2.

In this work we extract latent linkages from two datasets: a network of communications (RM) and an article citation network (CORA). The attributes extracted from RM are based on a questionnaire filled out by participants in the original paper and in CORA we use article keywords. As described in Section 4.2 we add edges depending on dataset-specific conditions. The conditions are the same for both -ANP and -ME graphs:

- RM: Two nodes are connected if, based on their answers in the dataset's questionnaire, they both consider each other a friend *or* they spend any time in the same physical vicinity. The intuition here is that students tend to be friends, and spend time on campus, with other students and less so with people who are not students.

- CORA: Two nodes are connected if they share any of the first 1200 word-vector features, sorted by ascending frequency, which are included in the dataset. Without using a cutoff – thereby including all word-vectors – resulted in a heavily connected graph (≈ 2.5M edges) where inference was computationally infeasible. The idea behind is that there exists a strong correlation between articles' use of globally infrequent words, and the topic of the article. By connecting articles that use the same words we hope to improve relational classifiers.

We use the area under the ROC curve (AUC) to compare the classifiers' performance since the class distribution, shown in table 4.1 under $P(+)$, is skewed and the accuracy measure alone therefore is not discriminative enough. For the RM samples we compute average AUCs for every sample and finally report the harmonic mean over all samples' AUCs.

4.5 Results

Results for baselines and addition of label-independent features are shown in Figures 4.2 and 4.3. Clearly, the baselines' performance suffers when the ratio of labeled nodes is around ≈ 0.1 on the original datasets. The lackluster performance of the baselines is most likely due to the sparse labeling: the algorithms employed rely on label propagation, but without enough labels the original network may not propagate correct labels effectively. In addition, the baselines cannot, in their original setup, make use of latent features or attributes inherent in the data.

Figures 4.2 and 4.3 also show the effects of adding label-independent features to classification methods. In general, the label-independent features do not seem affect the AUCs in an easily predictable way. For 26 out of the 30 different ratios of labeled-to-unlabeled data, the performance of nLB, wvRN and GRF *decreases* when using the LI method, and it is apparent that label-sparsity has a strong negative impact on both baselines and LI-methods. In addition, the logLI logistic regression classifier, run by itself, shows poor performance compared to all other baselines. This could explain why there are so few examples where LI-methods

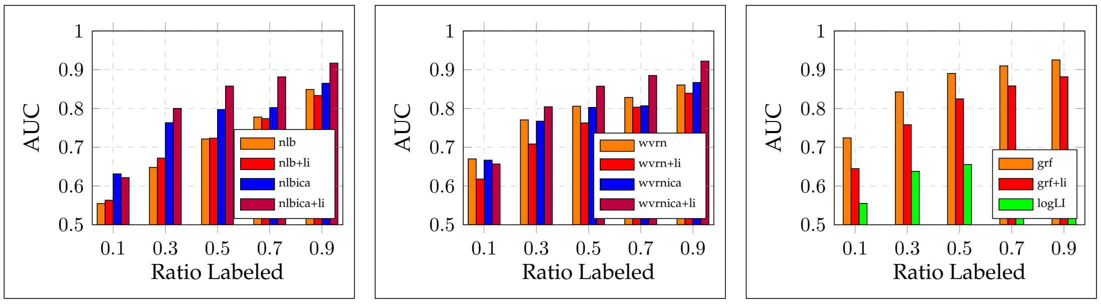


Figure 4.2: RM – Baseline vs. label-independent method.

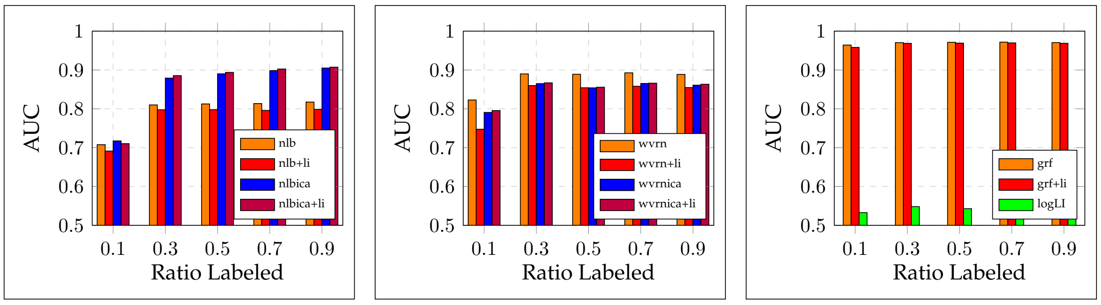


Figure 4.3: CORA – Baseline vs. label-independent method.

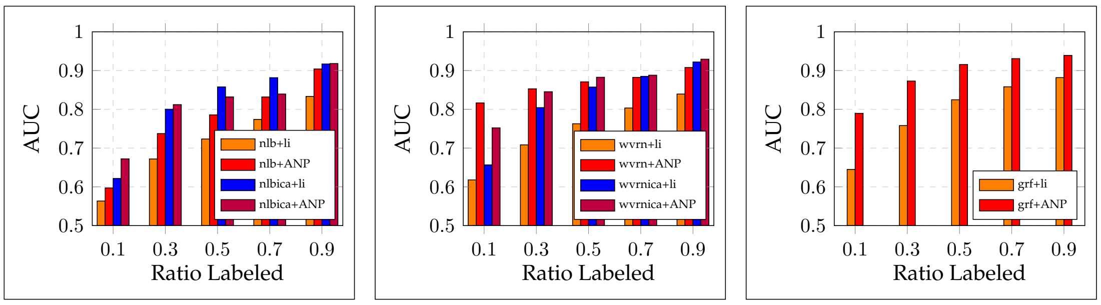


Figure 4.4: RM – Label-independent method vs. ANP method.

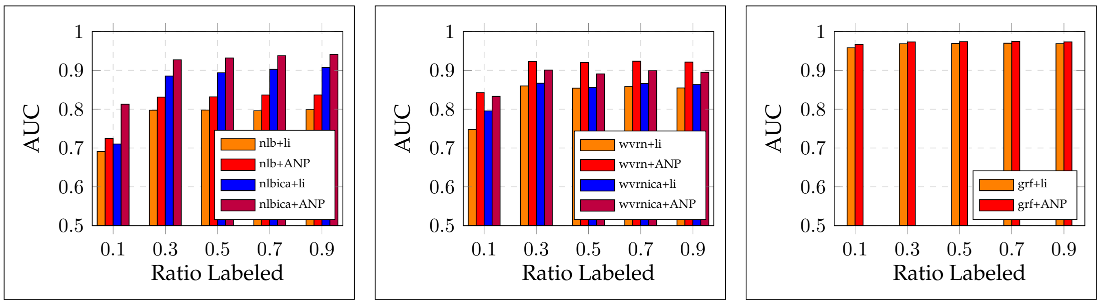


Figure 4.5: CORA – Label-independent method vs. ANP method.

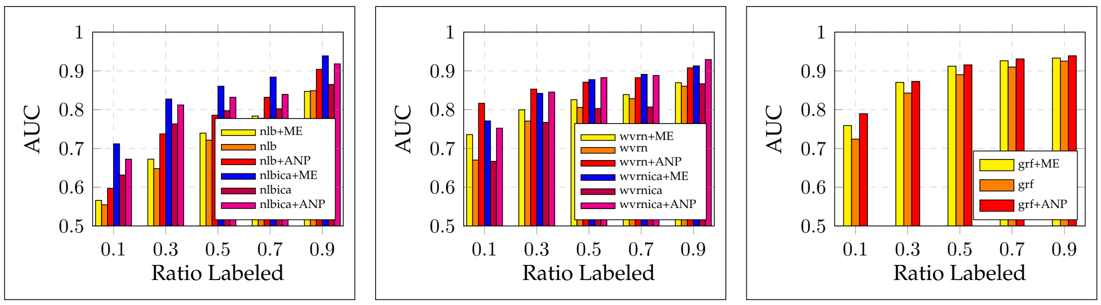


Figure 4.6: RM – Baseline vs. ME method vs. ANP method.

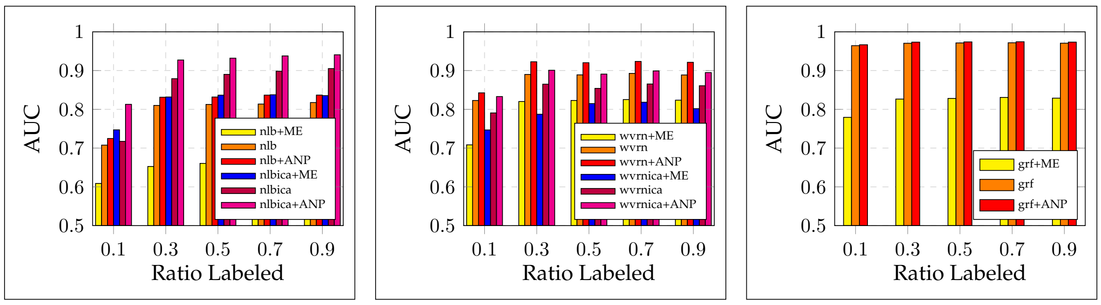


Figure 4.7: CORA – Baseline vs. ME method vs. ANP method.

outperform the baseline. The only cases where the LI approach improved AUC are when collective classification (ICA) is used for trials with ratio ≥ 0.3, but even then the improvement of AUC is negligible, $\leq 1\%$, for the CORA dataset.

Results for our proposed method of Attribute Network Propagation are shown in Figures 4.4 and 4.5. We compare the results from the label-independent methods against the ANP method described in Section 4.2. In 48 out of the 50 comparisons, our proposed method outperforms the label-indpendent method; the benefit is most pronounced for lower ratios of labeled-to-unlabed data (e.g., ratio ≤ 0.5 for the RM dataset), A prominent example of this effect can be seen for wvRN+ANP with ratio $= 0.1$ that outperforms wvRN+li, even when the latter is given access to a much higher proportion of labeled nodes ratio $= 0.7$.

Our results from Figure 4.4 and 4.5 show that using edges based on shared attributes may improve the AUC when relational classifiers are used on sparsely-labeled problems, but they are not enough to confirm that the complexity of our method of linearly combining predictions with ANP is justified.

In the last two figures 4.6 and 4.7 we have results from ANP compared with results of the less complex method ME. Its easily apparent from the CORA results in Figure 4.7 that simply adding edges based on shared attributes with ME can lead to a decrease in AUC scores. For all ratios of labeled-to-unlabeled data in the CORA dataset, the mixed approach +ME performs worse than baseline algorithms on the original dataset. In total, ANP performs better than ME in 43 out of 50 cases. Running inference on separate graphs, like with ANP, can in this case be an advantage. This phenomenon may be due to how relational classifiers like wvRN and nLB use neighboring nodes' labels. The added edges method of ME might cause more confusion when calculating the wvRN weighted sum in equation 21 by adding edges to nodes which have opposite labels of the one being classified, whereas with ANP the separation of edge types potentially avoids this problem. However we have not been able to confirm this yet.

4.6 Conclusions and Future Work

We have proposed a method for within-network classification where attributes generate new graph edges, thus taking advantage of latent linkages otherwise not present. Our method combines several ideas from recent research: (i) create a parallel attribute-graph for each dataset where edges only indicate shared node attributes, and (ii) run inference separately on both graphs using simple relational classifiers, and combine predictions after. We implement and reproduce results from a recent paper that proposed a label-independent approach. In experiments on two disparate real-world datasets, we have compared our methods to baselines and the label-independent method under varying label-sparsity conditions. Our experiments showed that: (1) The baseline methods perform poorly when very few labels are available; (2) Label-independent features do not consistently produce accurate predictions and can in some case worsen the performance of the constituent baseline methods; (3) Linearly combining predictions from multiple subgraphs can be superior to simply adding edges to the original graph.

In our current model we do not assign weight or value to individual attributes. However, we have observed *ad hoc* that certain attributes predict classification more than others others; thus we are currently investigating automatic ways of attribute selection and recombination of predictions. The attribute edges used by ANP and ME represent relations based only on homophily, but future work should incorporate co-citation regularity phenomena as well. Furthermore ME could be modified to handle classes of edges, instead of treating original and added edges the same. Finally, it remains to compare the performance of our method to recent complex methods like [258, 89, 87], as well as multi-relational network methods, on these types of binary classification problems.

5 Classification in temporal graphs

After working with graph structure transformations and classification in static graphs I wanted to move on the challenge of characterizing nodes in temporal graphs. Most of the work I found was either tailored to specific data or to problems which where not within-network classification, therefore I selected a set of well-known node metrics as features and began experimenting with how to extend them. A preliminary version of the work in this chapter has been presented at the Eighth IEEE ICDM Workshop on Data Mining in Networks 2018 (DaMNet) [36]. The chapter will form the basis of a paper to be submitted in early 2019.

5 Classification in temporal graphs

Abstract

Within-network classification, which involves correctly assigning labels to nodes in a graph, has many applications in several important domains such as anomaly detection and identifying criminal activity. Recent results indicate that static graph features, as well as feature-engineering, might not be adequate to solve these kinds of challenges in graphs that evolve over time. To investigate and solve this problem, we consider several classification problems using already established temporal metrics, and we propose new label-sensitive and recency-sensitive variants of these metrics that capture graph structural properties combined with labeling information and additional temporal patterns in the data. We test all new and old metrics using tuned off-the-shelf classifiers on 9 datasets of varying size and usage domain. Our results show that usage of label- and recency-sensitive metrics on real-world data provides more accurate results than static approaches and approaches based on temporal metrics alone.

5.1 Introduction

A common problem in network data is to ascertain or predict properties of individual nodes or edges, for instance to identify influential people in social networks. One standard way of doing so is to employ machine learning techniques to perform classification of single nodes, typically called *within-network classification*. This approach has traditionally used features based on graph-structural metrics on static (i.e., time-invariant) graph representations of the data, or using other features inherent in the data (e.g. social features)[261, 98, 91, 100, 266].

However, many real-world networks exhibit changes such as growth or labeling over time. For example, in social networks, friends are added or removed, and in citation networks, articles are cited by newer publications. While substantial work has been done on *static* graphs that do not evolve over time, classification in dynamic (aka. *time-varying* or *temporal*) graphs remains a relatively new field. The problem generally is to predict the *future* labels of existing nodes in a dynamic graph, using only the information available before a specified time-instant.

Generic methods [92, 102] for within-network classification employ feature sets that abstract away from the domain of application; thus, typical features are purely graph-theoretic, topological in nature, or local. However, such features do not a priori take into account the evolution over time, whence *temporal* variants of the features have been devised by a number of authors[8, 26, 267, 268]. Some temporal features require a "batched" representation of the dynamic graph [3, 269], hence models consisting of temporal subgraphs of the original dynamic graph, each covering successive time-windows[2], are frequently used. However, these methods do not experiment with combining temporal information with information (such as labels) in the data to improve accuracy.

For classification problems with information about the time of node or edge additions to a graph, we posit that (i) time-sensitive variants of standard graph-theoretic features will lead to better classification performance than using their static variants, and (ii) that incorporating *current* information about the known class of other nodes thereof will lead to better future classification of nodes whose class is unknown (iii) using information on how node metrics change over time, especially "recent" changes, can further improve classification accuracy.

Contributions: We define two new variants of temporal graph metrics (**label-sensitive metrics** and **recency-sensitive metrics**). One takes class information of other nodes in the graph into account, and the other captures recent topological changes in the graph. We use both these and a standard set of graph metrics devised elsewhere in the literature in the ensuing experiments.

We perform an experiment using 9 datasets for time-evolving graphs to compare a total of 6 configurations of feature sets, both static and temporal. The feature sets are compared to each other and to two baselines consisting of (i) purely static graph-theoretical features and (ii) a disease propagation model. Our key findings are: (1) temporal extensions of static metrics are insufficient for classification; (2) metrics that encode label information lead to better performance, both static and temporal; (3) Encoding recent activity into metrics improves accuracy, but only in combination with label information; (4) success of path-based graph metrics are highly dependent on graph characteristics.

The time-sensitive variants of graph metrics have appeared in previous work on metrics for time-windowed graphs [26, 8]; the label-sensitive and recency-sensitive variants are new to this paper, as is the experimental comparison.

5.1.1 Related Work

Within-network classification has been widely studied. Traditional machine learning approaches have in most cases made use of static representations of the network [98].

Desikan et al. [123] present a generic method to analyze temporal node behavior using node properties based on link structure and a weighting mechanism to take recency into a account, to classify web-pages, but their methods do not scale to large datasets and are not specific to within-network classification. Wang et al. [270] proposed a model to predict how a virus would propagate in a real dynamic network, using propagation; we use their SI-model as a non-machine learning-based baseline. Tagarelli et al. [137] give an in-depth analysis of lurkers in online social networks and identify several temporal dynamics that are used for time-aware ranking to better classify nodes, but limited to nodes with low activity. Instead of local topological features, Aggarwal and Li [127] use random walks and textual node attributes to perform within-network classification. Although comparable, they only perform experiments on 2 of the 9 datasets we use, and do not take recency into account.

Yao et al. [125] represent dynamic network data in a streaming fashion with continuous updates and use a kernel-based approach to classify nodes. Another streaming network approach uses active learning and network sampling to efficiently perform classification in large networks[124]. Our method differs from these approaches by additionally modeling node labels as a stream in time, thereby retaining temporal information about class membership over time.

Santoro et al. [3] experiment with temporal metrics which require a time-windowed (or batched) representation of time-windowed graphs similar to work by Casteigts etl al. [2]. Their work focuses on the evolution of node metrics, and do not demonstrate the performance when tasked with classification.

Other approaches use graph metrics, originally formulated for static graphs, and extend them to the temporal domain [8, 26, 11]. Our work builds on their work, by using and modifying their definitions of temporal node metrics, and seeks to further understand how to most efficiently encode the temporal information in the network through classification experiments.

5.2 Preliminaries and Definitions

As the literature on time-windowed graphs contains many variations in nomenclature, we give full definitions appropriate for our domain of application below, many of which originate from [8]. An extended set of definitions can be found in the supplementary material. In the remainder of the paper, let $\mathbb{T}$ be a totally ordered set (representing *time*).

Definition 21. *A* ***temporal graph*** *G is a tuple (V, E), where V is the set of nodes of G and E is the set of edges. An edge $e \in E$ is a triple (x, y, t) where $x, y \in V$ and $t \in \mathbb{T}$. In the remainder of the paper $e = (x, y, t)$ should be understood as an undirected edge between x and y at time t.*

Definition 22. *A* ***stream*** *S is a set of triplets of the form (x, y, t) where $t \in \mathbb{T}$. A* ***substream*** *of a stream S is a subset $s \subseteq S$.*

Intuitively, a stream is a succession of “events” (x, y) each happening at some time t. Typically, (x, y) represents the creation of a undirected edge between nodes x and y in a graph.

Definition 23. *A* ***time-window*** *τ is a tuple $\tau = (t, t') \in \mathbb{T}^2$ where $t < t'$. For $\tau = (t_i, t_j)$ a time-window and S a stream, the* ***windowed substream*** *s_τ induced by τ and S, is the substream $s_\tau = \{(x, y, t) : (x, y, t) \in S \text{ and } t_i \leq t < t_j\}$.*

Thus, the windowed substream intuitively is the set of all “events” occurring between t_i and t_j.

5.2.1 Time-Windowed Graphs

Some temporal metrics require modeling the stream as several temporal graphs, where each graph represents the “state” of the stream during a specific period of time. A straightforward approach is to split the stream into consecutive windowed substreams, such that each substream induces a temporal graph. To do this we define a type of graph container where each temporal graph in the container is the result of a batch of insertions/deletions. This type of graph is called a *time-windowed graph*.

Definition 24. *A* ***time-windowed graph*** *is a pair $(\mathcal{G}, \mathcal{T})$, consisting of a set of temporal graphs, $\mathcal{G}$, and a set of time-windows $\mathcal{T}$ where $|\mathcal{G}| = |\mathcal{T}|$. For each temporal graph $G_i \in \mathcal{G}$*

there is a time-window $\tau_i = (t_m, t_n) \in \mathcal{T}$ so that for any edge $e = (x, y, t) \in G_i$, we have $t_m < t < t_n$. The adjacency matrix *of a time-windowed graph is the map $A_{(\mathcal{G},\mathcal{T})}$ such that if $\tau_i \in \mathcal{T}$ is a time-window, then $A_{(\mathcal{G},\mathcal{T})}(\tau_i) = a_{xy}(\tau_i)$ is the adjacency matrix of the temporal graph for time-window τ_i.*

Definition 25. *Let $\mathcal{G}$ be a set of temporal graphs. The **accumulated graph** $G^a(V^a, E^a)$ is the temporal graph whose nodes and edges are the accumulation of all edges and nodes in $\mathcal{G}$.*

5.2.2 Labels in Time-Windowed Graphs

As a main purpose of the paper is to predict future labels on *nodes*, temporal information on nodes, in particular their labels, must be treated; hence the following definitions.

Definition 26. *A **labelstream** $\mathcal{L}$ is a temporally ordered set of observed labels $(L_0, L_1, L_2 \cdots)$ where each label is of the form $L = (x, l, t)$. In the remainder of the paper $L = (x, l, t)$ should be understood as node x being assigned label l at time t.*

5.2.3 Snapshots of Time-Windowed Graphs

For classification of nodes in a TVG based on streams, we introduce "snapshots" in order to clearly model the TVG at the "present" and "future".

Definition 27. *Let t_p be a value in $\mathbb{T}$, $\mathcal{T}$ be a set of time-windows, S be a stream, and $\mathcal{L}$ be a labelstream. A **snapshot** at time t_p is a quintuple $(S_{t_p}, \mathcal{G}_{t_p}, \mathcal{T}_{t_p}, G^a_{t_p}, \mathcal{L}_{t_p})$ where S_{t_p} is a substream of S, and $\mathcal{T}_{t_p}$ is a subset of $\mathcal{T}$, both consisting of elements timestamped before t_p. We denote $(\mathcal{G}_{t_p}, \mathcal{T}_{t_p})$ as the time-windowed graph induced by S_{t_p} and $\mathcal{T}_{t_p}$, and $G^a_{t_p}(V^a_{t_p}, E^a_{t_p})$ as the accumulated graph of $\mathcal{G}_{t_p}$. The labelstream subset $\mathcal{L}_{t_p} \subseteq \mathcal{L}$ consists of labels timestamped before t_p for nodes in $V^a_{t_p}$. A node $v_i \in V^a_{t_p}$ is said to be* labeled in the snapshot at time *t_p, if there exists a label $(v_i, l, t) \in \mathcal{L}_{t_p}$.*

5.2.4 Problem Definition

The classification problem tackled in this paper is described below.
Input: Let $(t_p, t_f) \in \mathbb{T}^2$ be a tuple where $t_p < t_f$, $\mathcal{T}$ be a set of time-windows, S be a stream, $\mathcal{L}$ be a labelstream with binary labels $l \in \{0, 1\}$, corresponding to negative and positive class labels, and $(S_{t_p}, \mathcal{G}_{t_p}, \mathcal{T}_{t_p}, G^a_{t_p}, \mathcal{L}_{t_p})$ be a snapshot at time

t_p, referred to as the *present* snapshot, where $\mathcal{G}_{t_p}$ and $G^a_{t_p}$ may be bipartite.
Output: Let $V^0_{t_p} \subseteq V^a_{t_p}$ be the set of nodes in the present snapshot where for every node $v_i \in V^0_{t_p}$ there does **not** exist an observed label $(v_i, l, t) \in \mathcal{L}_{t_p}$ where $l = 1$. For each node $v_i \in V^0_{t_p}$ predict if there exists an observed label (v_i, l, t) where $l = 1$ in the labelstream of the snapshot at time t_f, referred to as the *future* snapshot. For the remainder of the paper: Nodes in $V^a_{t_p}$ for which a positive label exists at t_p are referred to as *present positive nodes*, and nodes in $V^0_{t_p}$ for which a positive label exists at t_f are referred to as *future positive nodes*.

5.2.5 Projections of Time-Windowed Graphs

Some of the datasets, used for experiments in this paper, are naturally bipartite graphs, but the metrics used are designed for non-bipartite graphs. While any bipartite graph can be viewed as an non-bipartite graph, it is often better for modeling purposes to use projections from bipartite to non-bipartite graphs. We briefly give the necessary definitions below.

Definition 28. *Let $G(V, U, E)$ be a bipartite temporal graph, the **temporal projection** $G^p(V^p, E^p)$ of G, is the one-mode projection of G onto V. G^p is then a temporal graph containing only nodes from V, where two nodes are connected by an undirected edge when they have at least one common neighboring node in U. The edge is assigned the largest timestamp seen from either of the two nodes, to the common neighbor.*

A visualization of temporal projection is shown in figure 5.1. Note: by the above definition, any node without 2-hop neighbors in G will be isolated in G^p. A definition for temporal projection of a time-windowed graph is not required by the graph metrics in this paper, and thus omitted.

5.2.6 Shortest Temporal Paths

Some of the temporal node metrics used in our work are based on paths in temporal graphs, as defined in [8]. Given a temporal graph $G(V, E)$, a **temporal path** from node x to y is defined as a sequence of edges

$$P = ((x, n_1, t_1), (n_1, n_2, t_2), \cdots, (n_{l-1}, y, t_l))$$

such that no nodes are visited more than once and $t_i \leq t_{i+1}$ for any two neighboring edges in the path. The length of a path len(P) is the number edges in

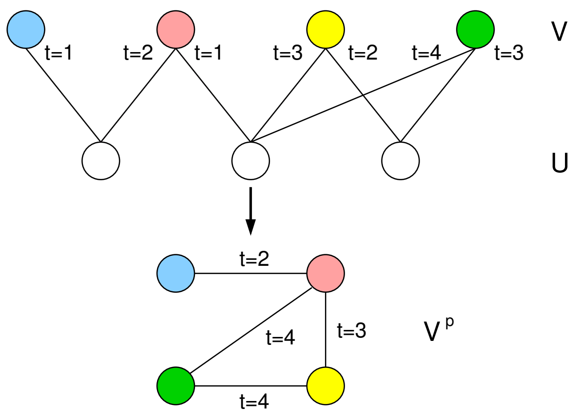


Figure 5.1: Example of a top-node bipartite temporal graph projection.

the sequence and duration is defined as $\text{dura}(P) = t_l - t_1$. A node y is said to be **temporally reachable** from node x if there exists a temporal path from x to y. Given the set of all temporal paths between node x and y, $\mathbf{P}_{xy}$, and a temporal graph $G(V, E)$, a temporal path $P \in \mathbf{P}_{xy}$ is a **shortest temporal path** if $\text{len}(P) = \min\{\text{len}(P') : P' \in \mathbf{P}_{xy}\}$, and is a **fastest temporal path** if $\text{len}(P) = \min\{\text{dura}(P') : P' \in \mathbf{P}_{xy}\}$. Both are viable for path based node metrics, however for the remainder of the paper we use shortest temporal paths as they are most compatible with existing efficient implementations.

5.3 Methodology

Our approach is to define a sequence of feature sets that take time-related and label-related information into account, and compare them experimentally to both each other and to a baseline that does not directly employ graph metrics.

Each of the six featuresets represents a specific way of extracting information from the data stream and are named (*and abbreviated*) as follows: Static Non-Label-Sensitive (*STop*), Temporal Non-Label-Sensitive (*TTop*), Temporal Recency-Sensitive Non-Label-Sensitive (*TRTop*), Static Label-Sensitive (*SLab*), Temporal Label-Sensitive (*TLab*), Temporal Recency-Sensitive Label-Sensitive (*TRLab*). All sets of features assume the existence of a snapshot of a time-windowed graph at a time instant t, $(S_t, \mathcal{G}_t, \mathcal{T}_t, G^a_t, \mathcal{L}_t)$. In the following, we describe how each feature in the feature sets is defined.

5.3.1 Static node features

The static approach discards all temporal information available and is based only on a single, undirected graph G_t. In the context of classification in time-windowed graphs, G_t is either the accumulated graph $G_t^a(V_t^a, E_t^a)$ taken at time t, or the temporal projection $G_t^p(V_t^p, E_t^p)$ (def. 28), if G_t^a is bipartite. Note than even though G_t^a and G_t^p are temporal graphs, the static methods described below do not take the temporal information into account. We use existing established definitions of metrics, but change notation slightly in order to be compatible with the nomenclature of time-windowed graphs and the problem definition in 5.2.4. We define the **betweenness centrality** of node x at time t:

$$C_x^B(t) = \sum_{\substack{y \in V_t^a \\ y \neq x}} \sum_{\substack{z \in V_t^a \\ z \neq x \\ z \neq y}} \frac{\sigma_{yz}(x)}{\sigma_{yz}}$$

where σ_{yz} is the number of shortest paths between node y and z in the temporal graph G_t, and $\sigma_{yz}(x)$ is the number of those paths that pass through x. **Closeness centrality** of node x at time t:

$$C_x^O(t) = \frac{N-1}{\sum_y d_{xy}}$$

where d_{xy} is the length of the shortest path between x and y in the temporal graph G_t and N is the number of nodes in G_t. **Local clustering coefficient:**

$$C_x^U(t) = \frac{2T_x(t)}{\deg_x(t)(\deg_x(t)-1)}$$

where $T_x(t)$ is the number of triangles through node x and $\deg_x(t)$ is the degree of x in G_t. The **static label-sensitive** node features are extensions of the above. For these metrics, only present positive nodes are considered. This means only paths to/between present positive nodes are considered, and a node's neighbors only consist of nodes which are also present positive nodes.

5.3.2 Temporal node features

We now list the temporal features, as well as label-sensitive variants of these, that will be used for classification. The features are based on the temporal node features from [8]. As with the static node features, we here assume the existence of a snapshot of at time instant t, and a single undirected temporal graph G_t (either accumulated or projected).

Temporal closeness centrality of node i, $C_x^{OT}(t)$, uses equation 5.3.1 as defined in [8], but for $C_x^{OT}(t)$, d_{xy} is defined as the length of the shortest temporal path between x and y. The **average temporal betweenness centrality** of node x at time t is:

$$C_x^{BT}(t) = \frac{1}{|\mathcal{T}|} \sum_{(t_i,t_j)\in\mathcal{T}} C_x^{tbc}(t_j)$$

where $C_x^{tbc}(t)$ is the temporal betweenness centrality of node x at time t, defined as:

$$C_x^{tbc}(t) = \frac{1}{(N-1)(N-2)} \sum_{\substack{y\in V_t \\ y\neq x}} \sum_{\substack{z\in V_t \\ z\neq x \\ z\neq y}} \frac{U(x,t,y,z)}{\sigma_{yz}}$$

where $U(x,t,y,z)$ is the number of shortest temporal paths from y to z which traverses node x at time $t' < t$. **Edge persistence** for node x is defined as follows: given two temporal graphs, $G_m(V_m, E_m)$ and $G_n(V_n, E_n)$ in $\mathcal{G}$, the edge persistence for x is the number of edges that persist across both graphs:

$$C_x^{ep}(G_m, G_n) = \frac{\sum_y a_{xy}(\tau_m) a_{xy}(\tau_n)}{\sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}}$$

The **average temporal edge persistence** of node x is then:

$$C_x^{UT}(t) = \frac{1}{|\mathcal{T}|-1} \sum_{m=0}^{|\mathcal{T}|-1} C_x^{ep}(G_{\tau_m}, G_{\tau_{m+1}})$$

where G_{τ_m} and $G_{\tau_{m+1}}$ are adjacent temporal graphs in $\mathcal{G}$, and $G_{\tau_m} \prec G_{\tau_{m+1}}$. As with the static metrics, the **temporal label-sensitive** node features are extensions of the temporal metrics. For these, only present positive nodes are considered. For temporal paths, this means only temporal paths to/between present positive nodes are considered, and a node's neighbors only consists of nodes which are also present positive nodes.

5.3.3 Recency-Sensitive Features

We propose an alternative to the temporal extensions in Section 5.3.2, called *recency-sensitive* features. Recency-sensitive features allow us to use definitions of common static graph metrics without change, but still capture temporal information in the context of recent changes. These are calculated using a meta-feature, which is a higher-level procedure that can use any node metric in combination with a set of time-windows, and produce a "temporal" node feature in $\mathbb{R}$. The recency-sensitive features capture temporal information in the network using changes in a node metric over a specified number of recent time-windows. Using the below definitions we create a *recency-sensitive* variant for each static node metric listed in section 5.3.1. Given a time-windowed graph containing a node x, the set of time-windows $\mathcal{T} = [(t_0, t_1), (t_1, t_2), \cdots, (t_{|\mathcal{T}|-1}, t_{|\mathcal{T}|})]$, the number of recent time-windows to consider n_τ, and a node feature function $f_x(t) \in \mathbb{R}$, the recency-weighted derivative (or recency-sensitive variant) of feature $f_x(t)$ of node x is:

$$f_x^{\Delta}(f_x, n_\tau) = \sum_{i=|\mathcal{T}|-n_\tau+1}^{|\mathcal{T}|} \frac{(f_x(t_i) - f_x(t_{i-1}))}{\log_2(2 + (t_{|\mathcal{T}|} - t_i))}$$

where the denominator is the freshness function[137].

5.4 Experimental Design

We now describe our experimental design and setup.

5.4.1 Classifier selection

Due to its prevalence in related literature [19], we use a decision tree classifier to classify based on the node features in subsection 5.3.2. To complement the classifier, we use a naïve disease propagation classifier as one baseline, the SI disease model based on [270] (details are in the supplementary material). The SI model infection rate is set to $\beta = 0.6$ and the decision tree is limited to a maximum depth of 6, both values based on highest F1 scores & precision/recall, over a 10-fold cross validation tuning experiment using 50% of labeled nodes. Tuning experiments were also run on SVM- and neural network-based classifiers, with comparable or worse results. The classifiers are trained on six configurations of node features for each dataset. The six sets of features consist of three non-

	deg	C^B	C^O	C^U	C^{BT}	C^{OT}	C^{UT}
STop	•	•	•	•			
TTop	•				•	•	•
TRTop	•/Δ	Δ	Δ	Δ	•	•	•
	deg^L	C^{BL}	C^{OL}	C^{UL}	C^{BTL}	C^{OTL}	C^{UTL}
SLab	•	•	•	•			
TLab	•				•	•	•
TRLab	•/Δ	Δ	Δ	Δ	•	•	•

Table 5.1: Feature Configurations. • indicates that the feature is included in configuration. Δ indicates that the recency-weighted derivative (Def. 5.3.3) of the feature is used.

label-sensitive and three label-sensitive configurations, see Table 5.1. The static featuresets STop and SLab use non-temporal features only, whereas TTop, TRTop, TLab and TRLab use the temporal node features from [8] instead. The two sets using recency-sensitive features have additional 4 recency-sensitive features each.

5.4.2 Datasets

We use 9 publicly-available datasets containing temporal information, from a variety of domains of application. Table 5.2 summarizes the resulting time-windowed graphs. Information about time-instants (i.e. t_p and t_f) used for each dataset as well as the size of time-windows can be found in the supplementary material. As seen in table 5.2, the fraction of positive labels is low ($P(+) < 0.1$ for most datasets). Although there exists strategies to compensate for label-imbalance in datasets, these can also have a negative influence on performance. With over-sampling, replicating positive labels, there is a high risk of over-fitting on many exact clones since the fraction of positive class labels is near zero. This would also result in almost a doubling in size of training data for each dataset. Conversely, under-sampling, removing non-positive labels, would waste 50%< of labels and might result in worse classification accuracy through lack of labels [271].

For 3 out of 5 of the bipartite datasets, the projection described in definition 28 results in a significant increase in number of edges (e.g. from $70k$ to $550k$ edges). Thus for simplicity and computability purposes, we impose an additional condition that for any temporal graph, there can exist at most one edge between two nodes. It is done in a way that ensures only the *latest* time-instant is used for

Dataset	$\lvert V^a_{t_p}\rvert$	$\lvert U^a_{t_p}\rvert$	$\lvert E^a_{t_p}\rvert$	$\lvert E^p_{t_p}\rvert$	Q_{t_p}	$P(+)$
IMDB	20k	66k	180k	460k	0.11	0.02
DBLP	4.2k	2.7k	15k	68k	0.39	0.02
ESCORT	1.6k	3.1k	7.3k	14k	0.07	0.04
DELIT	1.7k	-	7k	-	0.07	0.17
DELIB	1.7k	-	7k	-	0.07	0.13
CITEULIKE	0.9k	11k	12k	1.7k	0.03	0.06
DIGG	16k	-	29k	-	0.03	0.03
EPINIONS	17k	-	48k	-	0.004	0.02
WIKI	416	678	2.3k	4k	0.04	0.29

Table 5.2: Dataset Graph Characteristics: $|V^a_{t_p}|$ and $|U^a_{t_p}|$ are number of top and bottom nodes in the accumulated graph at time t_p, $|E^a_{t_p}|$ is the number edges in the accumulated graph, $|E^p_{t_p}|$ is the number of edges of the temporal projection of the accumulated graph, and Q_{t_p} is the modularity of the accumulated graph $G^a_{t_p}$ (temporal projection if bipartite). $P(+)$ is the fraction of nodes in $V^a_{t_p}$ which are future positive nodes.

the edge between x and y.

IMDb is a database of actors, movies, and ratings (between 0 and 10). **DBLP** is a database of metadata for papers [34], each consisting of a title, publishing year, authors, publication venue and references. The **Escort** dataset [272] consists of client ratings from an online forum detailing encounters with escorts. **Deli.cio.us** served as a platform for storing and sharing web bookmarks [273]. The dataset consists of social networking, bookmarking and tagging information from around 2k users. **CiteULike** is a network of users and publications [274]. There exists two types of timestamped events: a user u tags a document d at time t, and a user u uses a tag a at time t. **Digg** was a story aggregation website where stories were posted and voted upon by its users. The dataset consists user friendships and which stories users vote on [275]. The **Epinions** dataset [276] consists of data from a social product rating website. It contains two sets of timestamped events: trust (u_1 trusts/distrusts u_2 at time t) and ratings (u rates product p a value between 0 and 5, where 5 is the best rating). The **Wiki** dataset logs editing activity on wikipedia.com [277]. It consists of signed (positive/negative) interactions between users.

5.4.3 Experimental Methodology

We solve the problem of node classification described in subsection 5.2.4 for all datasets. For present and future timesteps, t_p and t_f, we pick two timestamps in the recent past to avoid lack of data, but also avoid using most recent data available due to computational restrictions. The rating threshold $r_{\text{threshold}}$ for labeling in the IMDb dataset is set as the mode of the distribution of ratings. For each dataset we choose fixed parameter values for t_p, t_f, $r_{\text{threshold}}$ and $\Delta\tau$ independently (details in supplementary material), and set the proportion of labeled nodes used during training to 50% and number of time-windows for recency metrics $n_\tau = 3$.

5.4.4 Classification

The classifier is trained on the features and labels of a subset of the nodes in the accumulated graph at time t_p. All features are standardized by removing the mean and scaling to unit variance. We run 50 trials and in each trial perform a class-stratified sampling containing $100 \times (\text{proportion labeled})\%$ nodes as a training set and the remaining as a test set. The samples are chosen so that each node appears the same amount of test sets over all trials. The baseline SI-model is run starting at the first time-window available in each dataset and is updated at the beginning of each new time-window using a fixed infection rate until reaching the final time-window at t_p.

5.4.4.1 Performance and scalability

The metrics were calculated on a standard laptop and took approx. 28 hours total for all datasets. As seen in Table 5.3, the time used for computation scales on the number of nodes and edges in the graph, except for IMDb. This trend is most likely because the majority of the chosen metrics are path-based, which scale with number of possible paths in the graph.

Although scalability was not a performance metric in this work, we briefly mention a few approaches to keeping node metrics updated, efficiently. A non-scalable way to deal with new data in the TWG is simply to re-calculate the node-metrics for each node. A scalable alternative is to use incremental graph metric algorithms, which do not need to iterate over all historical data to update the node metrics. By using incremental metrics, it could be possible to retain node

		SI				STop				TTop				TRTop				SLab				TLab				TRLab				t_c
Dataset	0/1	P	R	F1		P	R	F1		P	R	F1		P	R	F1		P	R	F1		P	R	F1		P	R	F1		
IMDB	0	.98	.18	.31	**.30**	.97	.65	.78	**.74**	.96	.74	.84	**.80**	.96	.71	.82	**.78**	.96	.74	.84	**.80**	.96	.81	.88	**.84**	.96	.75	.85	**.81**	11.25h
	1	.08	.95	.14		.12	.70	.21		.13	.56	.21		.13	.62	.22		.14	.60	.22		.15	.47	.23		.14	.58	.23		
DBLP	0	.96	.36	.52	**.50**	.95	.67	.77	**.73**	.94	.68	.78	**.74**	.95	.71	.80	**.76**	.95	.79	.86	**.82**	.95	.76	.84	**.80**	.95	.83	.88	**.84**	5.25h
	1	.07	.76	.13		.07	.41	.11		.06	.32	.10		.07	.36	.11		.09	.31	.14		.09	.32	.13		.09	.25	.13		
ESCORT	0	.99	.63	.77	**.75**	.98	.79	.88	**.85**	.97	.76	.85	**.82**	.98	.79	.87	**.85**	.99	.80	.89	**.86**	.98	.77	.86	**.83**	.98	.90	.94	**.91**	<10m
	1	.10	.90	.18		.13	.64	.22		.09	.51	.16		.12	.59	.19		.15	.74	.25		.10	.57	.17		.18	.48	.26		
DELIT	0	.85	.24	.38	**.37**	.84	.61	.70	**.63**	.81	.61	.69	**.61**	.82	.67	.74	**.65**	.81	.70	.75	**.65**	.81	.70	.75	**.65**	.82	.75	.78	**.68**	<5m
	1	.21	.82	.33		.24	.50	.32		.20	.40	.26		.22	.39	.28		.21	.34	.26		.22	.34	.26		.23	.31	.26		
DELIB	0	.88	.24	.38	**.36**	.88	.57	.68	**.62**	.87	.61	.72	**.65**	.87	.64	.74	**.66**	.88	.66	.75	**.68**	.86	.67	.74	**.66**	.87	.72	.78	**.70**	<5m
	1	.15	.80	.25		.17	.54	.26		.16	.44	.23		.16	.41	.23		.19	.45	.26		.16	.35	.19		.18	.33	.22		
CITEULIKE	0	.93	.78	.85	**.79**	.91	.80	.83	**.77**	.91	.76	.81	**.75**	.92	.87	.88	**.82**	.92	.80	.83	**.77**	.92	.92	.92	**.86**	.92	.95	.94	**.87**	<10s
	1	.11	.32	.16		.06	.15	.07		.07	.18	.09		.06	.11	.06		.10	.18	.10		.08	.07	.07		.13	.08	.10		
DIGG	0	.97	.56	.71	**.69**	.97	.77	.86	**.83**	.97	.76	.85	**.82**	.96	.82	.88	**.85**	.97	.81	.88	**.85**	.96	.85	.90	**.87**	.96	.89	.92	**.89**	12 h
	1	.05	.63	.10		.06	.33	.09		.05	.30	.08		.05	.24	.08		.05	.28	.09		.05	.19	.08		.05	.15	.07		
EPINIONS	0	.99	.54	.70	**.69**	.99	.70	.82	**.80**	.98	.79	.87	**.86**	.98	.74	.84	**.82**	.98	.74	.84	**.83**	.98	.72	.83	**.81**	.98	.77	.86	**.84**	9.5 h
	1	.02	.59	.05		.03	.48	.06		.04	.38	.07		.03	.40	.06		.03	.35	.05		.03	.39	.05		.03	.35	.05		
WIKI	0	.64	.23	.33	**.30**	.92	.68	.78	**.72**	.90	.71	.79	**.72**	.91	.63	.74	**.68**	1.00	.77	.87	**.82**	.92	.67	.78	**.72**	.97	.80	.88	**.83**	<10m
	1	.03	.16	.05		.23	.61	.33		.21	.50	.28		.20	.59	.29		.39	.98	.56		.23	.62	.33		.39	.82	.53		

Table 5.3: Results on datasets for feature set. Precision (P), recall (R), and F1-score is listed for both classes as well as a weighted F1-score (in bold). Green cells indicate best weighted F1-score for dataset. Column t_c contains running times for each dataset.

information, based on the first historical data in the stream, as long as there is space in memory for the TWG data structure [278]. To avoid infinite growth of a TWG, one can retain only a dataset-dependent number of the most recent time-windows, and discard older windows. Using this approach alone comes at the cost of re-calculating node metrics over time. We leave in-depth analysis and scalable implementations to future work.

5.5 Results

Results for the disease model and the classifier trained on node feature sets are shown in Table 5.3, and Figure 5.2. Table 5.3 consists of precision/recall and F1-scores for each featureset, where the leftmost two featuresets are the baselines, and an approximate running time needed to calculate node metrics for each dataset. We choose a weighted F1-Score for Table 5.3 and fig. 5.2 to account for label-imbalance.

For 8 of the 9 datasets, the baseline SI-model fails to achieve an average F1-score above 0.6. The relatively poor performance is likely due to the SI model relying on a large number of initial ground truths in order to accurately classify future nodes. However, in most datasets, the proportion of positive nodes is ≤ 0.06 (highest is 0.29) which is low compared to datasets in related work [19].

The purely temporal TTop features exhibit a small increase in performance over the static features for 4 of 9 of the problems. With the recency-sensitive TRTop features, this improves to 7 of 9, but leads to worse performance than TTop on two of the datasets. Thus, existing temporal metrics alone perform poorly, especially considering the number of nodes whose temporal features are zero-valued: as seen in Table 5.4, more than 70% of the values are zero for C^{BT} and C^{UT} for 7 of 9 datasets, and in the last two sets close to 50%. Hence, for most nodes there is little information available in the TTop features. The amount of zero-valued features in CiteULike (highest average of all datasets) might explain why the SI-model did comparatively best here as the features used by the classifier carried little information.

Using label-sensitive features increases the F1-score over non-label-sensitive counterparts. The static featureset SLab achieves performance similar to the static featureset STop, and slightly better results than TTop and TRTop, suggesting that the use of label-sensitive static features may perform better than simply extending static features to temporal graphs. Similarly, the set of temporal label-sensitive features, TLab, scores higher than pure temporal features TTop on 8 of 9 datasets and is the best performing featureset for IMDb. However for most of the remaining datasets, TLab performs worse than SLab and TRLab. The lackluster performance might again be attributed to many zero-valued features. As seen in Table 5.4, both temporal label-sensitive features, betweenness and edge persistence, are zero for over 78% of nodes and for 6 of 9 is nearly 100%. Therefore, with the TLab feature set, the classifier must rely mostly on degree and closeness metrics.

Finally, in 7 of 9 datasets, adding recency-sensitive features to temporal label-sensitive features outperforms all other featuresets, and achieves on average $\approx$ 10% higher weighted F1-score than STop for all datasets.

Adding recency-sensitive features alone has little effect. As seen for the TTop and TRTop featuresets, there is only a minor positive effect on classification performance, and only on 6 of 9 datasets. When comparing TLab vs. TRLab, accuracy depends on the dataset: For IMDb, TRLab increases positive class recall significantly, but degrades negative class recall by a factor of ≈ 0.15. Conversely, for 6 of 9 datasets, the positive class recall is reduced, but negative class recall is increased. The most significant improvement is observed for the Wiki dataset

Dataset	C^B	C^O	C^U	C^{BT}	C^{OT}	C^{UT}	C^{BTL}	C^{OTL}	C^{UTL}
IMDB	60%	4%	9%	73%	4%	86%	85%	15%	92%
DBLP	43%	0%	5%	87%	0%	91%	97%	12%	100%
ESCORT	53%	11%	20%	78%	11%	89%	100%	18%	97%
DELIT	23%	0%	20%	52%	0%	45%	78%	21%	87%
DELIB	22%	0%	21%	52%	0%	45%	81%	22%	89%
CITEULIKE	85%	59%	74%	93%	59%	98%	97%	65%	98%
DIGG	76%	0%	92%	88%	0%	97%	91%	31%	99%
EPINIONS	62%	0%	82%	81%	0%	95%	92%	30%	99%
WIKI	47%	0%	100%	80%	0%	93%	90%	12%	93%

Table 5.4: For each feature: Percentage of nodes with zero-valued feature (rounded to nearest whole number).

where precision & recall for both classes are improved by up to $\approx 30\%$.

The difference of adding recency to non-label-sensitive vs. label-sensitive metrics implies that changes in label-sensitivity is more important than recency. This is consistent with what recency-label-sensitive features capture: For IMDb, these enable easy identification of currently active actors that have not yet had many roles (i.e., having many temporal features near zero). However, recency-sensitive features only improve performance when labels are taken into account (TRLab vs. TLab) as it matters *which* movies an aspiring actor is in. If their co-stars of recent movies already are famous, it improves the probability of themselves becoming a star.

Table 5.3 shows that, for 7 of 9 datasets, the positive class recall is higher for all non-label-sensitive configurations, while precision is higher for their label-sensitive counterparts. The lower recall might be a result of a difference in number of paths: there are fewer label-sensitive paths (used in computing the centrality score) than non-label-sensitive, and therefore fewer future positive nodes are visited.

Low modularity for 7 of 9 datasets, as well as the percentage of positive nodes $P(+)$, provide added complexity, and might explain why Epinions, with almost zero modularity, does not show improvement with label-sensitive methods. Even the DeliT and DeliB datasets, with $\approx 15\%$ positive nodes, do not benefit as much from label-sensitive metrics, as other datasets. The low modularity combined with high percentages in Table 5.5 could indicate that homophily is not a good predictor for class membership in these datasets, and that future positive nodes

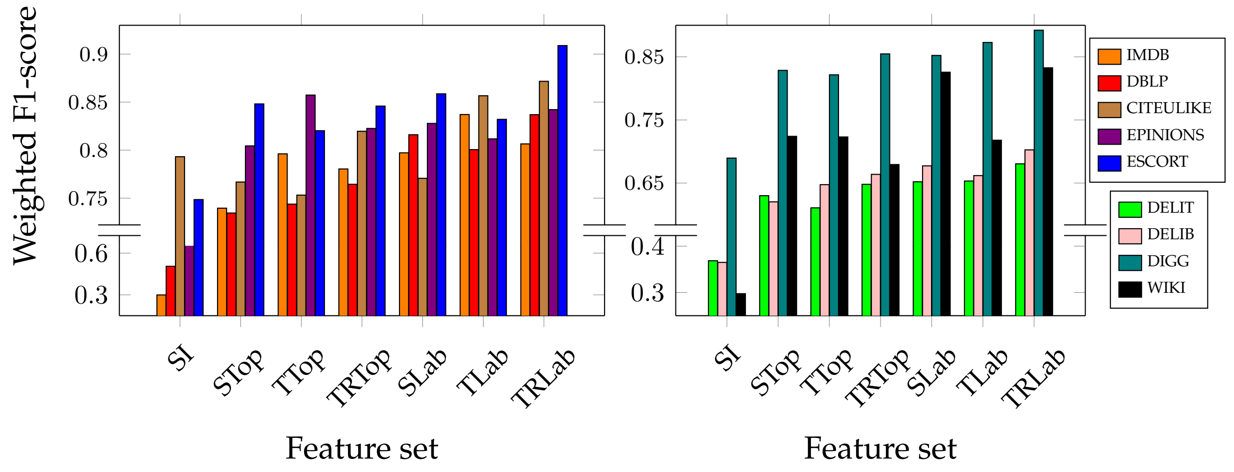


Figure 5.2: Weighted F1-scores for classification experiments on datasets.

Dataset	Static	Static LS	Temporal	Temporal LS
IMDB	7.79%	6.83%	7.91%	7.17%
DBLP	1.02%	0.12%	1.68%	0.39%
ESCORT	9.02%	2.33%	13.14%	14.56%
DELIT	16.02%	19.26%	18.60%	24.99%
DELIB	16.88%	22.61%	19.38%	27.29%
CITEULIKE	5.22%	2.92%	5.75%	5.79%
DIGG	5.23%	5.82%	5.04%	5.29%
EPINIONS	4.66%	5.40%	4.94%	5.60%
WIKI	1.51%	0.14%	2.03%	0.38%

Table 5.5: Percentage of nodes in shortest paths which are future positive nodes.

are not very well connected to existing positives. However, for all datasets presented, label-sensitive metrics unequivocally yielded the best performance overall.

5.6 Conclusions And Future Work

We have proposed and investigated two sets of new graph metrics for performing within-network classification on nodes in time-windowed graphs: label-sensitive and recency-sensitive graph metrics. The proposed metrics were compared to existing temporal graph metrics as well as a label-spreading algorithm through experiments on 9 real-world datasets. Results showed that incorporating both *label-* and *recency-sensitivity* to existing graph-structural metrics improves classification performance, and that the performance of popular temporal path based graph metrics are highly dependent on graph structure.

For further work, we firstly aim to experiment with tuning time window sizes depending on the activity in the labelstream, to potentially improve classification performance. Secondly, many topological features exist for time-windowed graphs, but either fail to take purely local features into account (e.g., connectivity in the immediate neighborhood of a node), or fail to accommodate time-dependence such as decay in the local neighborhood of nodes structure (e.g., for some applications "old friends" are more important than "new" friends", for others, vice versa). Devising metrics or local features that accommodate these phenomena could potentially improve classification performance.

5.7 Supplementary Material

Here we give supplementary information for the article ”Classification in temporal graphs” including additional information about datasets, sampling, and the label-spreading algorithm used.

5.7.1 Extended Datasets

In this section we describe for each dataset, how the corresponding stream of triplets S and the associated labelstream $\mathcal{L}$ are created.

Dataset	t_p	t_f	$\Delta\tau$
IMDB	2000	2005	1 year
DBLP	2006	2011	1 year
ESCORT	2005	2009	182 days
DELIT	5/2010	11/2010	91 days
DELIB	5/2010	11/2010	91 days
CITEULIKE	3/2005	7/2005	15 days
DIGG	8/6/2009	10/6/2009	1 days
EPINIONS	5/2001	9/2001	30 days
WIKI	5/2003	11/2004	91 days

Table 5.6: Time-windowed graph parameters for each dataset. Column t_p is the time of the present snapshot, t_f is the time of the future snapshot, and $\Delta\tau$ is the size of the time-windows

IMDB (Bipartite) is a database of actors, movies, and ratings (between 0 and 10). An IMDb triplet $(a, m, t) \in S$ corresponds to an actor a participating in a movie m at time t. An IMDb triplet $(a, l, t) \in \mathcal{L}$ means actor a has participated in a movie at time t where the label l is a binarization of that movie's rating r. Using the entire dataset was computationally infeasible, hence a sampled version of the IMDb dataset is used. Details about the sampling and binarization processes are in Section 5.7.2.

DBLP (Bipartite) is a database of metadata for papers [34], each consisting of a title, publishing year, authors, publication venue and references. We only use papers published in *database and data mining* (DBDM) or *computer vision and pattern recognition* (CVPR) venues. The set of DBDM venues is: PODS, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA, and the set of CVPR venues is: CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM. A DBLP triplet $(a, p, t) \in S$ corresponds to an author a publishing a paper at time t which cites paper p. A DBLP triplet $(a, 1, t) \in \mathcal{L}$ means an author a has published a paper at time t in any DBDM venue. A triplet $(a, 0, t)$ is given for every paper published at a CVPR venue.

Escort Service (Bipartite) [272] consists of client ratings from an online forum detailing encounters with escorts. The ratings are either -1, 0 or 1, corresponding to bad, neutral or good. Each triplet (e, c, t) in the stream represents an encounter between a client c and an escort e at time t. The labelstream is based on the five clients with most encounters: a triplet $(e, 1, t)$ means an escort e has had an encounter with any of the top five clients at time t. A label $(e, 0, t)$ is given for an encounter between e and any other client.

Deli.cio.us served as a platform for storing and sharing web bookmarks [273]. The dataset consists of social networking, bookmarking and tagging information from around 2k users. From these we define two separate datasets, DeliT and DeliB. A triplet (u_1, u_2, t) in the stream S corresponds to a contact between u_1 and u_2 and is used for both datasets. $(u, 1, t)$ in DeliB labelstream means that user u bookmarked a page on the domain wikipedia.org at time t. $(u, 0, t)$ is used for a bookmark of any other domain. Likewise for DeliT, a triplet $(u, 1, t)$ means a user tagged a website with "Art" at time t, and a triplet $(u, 0, t)$ is given when using any other tag.

Citeulike (Bipartite) is a network of users and publications [274]. There exists two types of timestamped events: a user u tags a document d at time t, and a user u uses a tag a at time t. Each triplet (u, d, t) in S corresponds to a user u tagging a document d at time t. The labelstream $\mathcal{L}$ is based on whether a user has used one or more of pre-selected set of popular tags. A triplet $(u, 1, t)$ means user u used one of the pre-selected tags at time t. A triplet $(u, 0, t)$ is given for use of any other tag.

Digg was a story aggregation website where stories were posted and voted upon by its users. The dataset consists user friendships and which stories users vote on [275]. A triplet (u_1, u_2, t) in the stream S means a friendship was established at time t. The labelstream is based on the top 10 most voted stories on Digg. A triplet $(u, 1, t)$ means a user u voted on one of the top 10 stories at time t. A triplet $(u, 0, t)$ is added for a vote on any other story.

Epinions dataset [276] consists of data from a social product rating website. It contains two sets of timestamped events: trust (u_1 trusts/distrusts u_2 at time t) and ratings (u rates product p a value between 0 and 5, where 5 is the best rating). The stream consists of triplets like (u_1, u_2, t) corresponding to trust relationships as described above. The labelstream is based on product ratings of the 1000 most rated products. A label $(u, 1, t)$ indicates that user u rated one of the 1000 most rated products with a rating of 5 at time t. A label $(u, 0, t)$ is given for any rating to any other product.

Wikipedia Conflicts (Bipartite) dataset logs editing activity on wikipedia.com [277]. It consists of signed (positive/negative) interactions between users. The sign is a decimal value between -10 and 10. A Wiki triplet $(u_1, u_2, t) \in S$ corresponds to a non-neutral interaction (i.e. the absolute value of the sign is greater than 1) between u_1 and u_2 at time t. A Wiki triplet $(u, 1, t) \in \mathcal{L}$ means a user u has had more than 1 strongly negative (i.e. sign is less than -4) interaction with another user, where the second interaction took place at time t. A label $(u, 0, t)$ is given for any other non-neutral interaction.

5.7.2 IMDb Sampling and Binarization

The binarization of the labels for the IMDb, using a fixed threshold, is too strict (i.e. two movies with ratings 6.999 and 7.001 would be given different labels, even though they are very close in rating), therefore the threshold must be softened. To soften the binarization process, we add Gaussian noise. Given a value $r_{\text{threshold}} \in [0, 10]$, and a random variable $X_r \sim \mathcal{N}(r_{\text{threshold}}, \frac{1}{2})$, we define the labelstream $\mathcal{L}$ as:

$$\mathcal{L} = \{(a, l, t) : (\exists (m, r) \in \mathcal{D})[(a, m, t) \in \mathcal{A}\}$$

where $(m, r) \in \mathcal{D}$ is a rating of r for movie m, $(a, m, t) \in \mathcal{A}$ is an actor a in a movie m which appeared in cinemas the year t, and $l \in \{0, 1\}$ is given by:

$$\Pr(l = 1 \mid r) = \Pr(X_r \leq r) \tag{24}$$

$$\Pr(l = 0 \mid r) = 1 - \Pr(l = 1 \mid r)$$

Thus, if the rating of a movie r is higher than $r_{\text{threshold}}$, the actors who participated are given a positive label (with some probability) as illustrated in fig. 5.3. Thus, some movies with $r < r_{\text{threshold}}$ will result in positive labels and vice versa -

however, the farther the rating is from $r_{\text{threshold}}$, the lower the probability is of this happening.

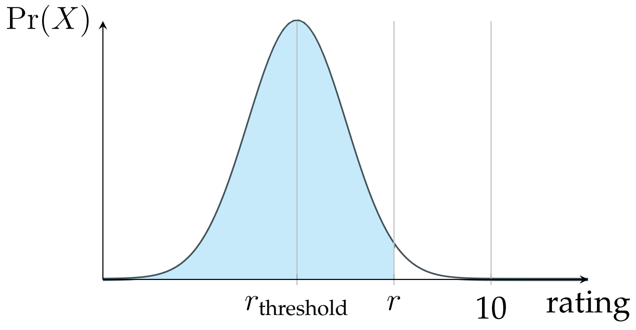


Figure 5.3: Distribution plot to illustrate added noise to IMDb labeling.

The original IMDb dataset is very large and for resource management purposes it was necessary to downsample. Furthermore, due to some low-quality entries in the ratings of movies, we set labels to zero of movies which were not rated by more than 200 members of IMDb to avoid movies which were rated very high/very low by only a handful of people. The strategy used to modify labeling and to downsample $\mathcal{A}$ and $\mathcal{D}$ is described below:

1. Define a set M_c which contains all movies originating from a selected subset of countries, in our case Denmark & Sweden.
2. Define a set A_c which contains the top 20 members of the cast, ordered by cast billing, for each movie in M_c.
3. For each member in A_c, add all movies they have appeared in, to M_c.
4. The downsampled $\mathcal{A}_d$ is then defined as:
$$A_d = \{(a, m, t) \in \mathcal{A} : a \in A_c \text{ and } m \in M_c\}$$
5. The downsampled $\mathcal{D}_d$ is then defined as:
$$\mathcal{D}_d = \{(m, r) \in \mathcal{D} : m \in M_c\}$$

6. Let n^r_m be the number of ratings for movie m. Equation 24 then becomes:

$$\Pr(l = 1 \mid r) = \begin{cases} \Pr(X \leq r) \text{ if } 200 \leq n^r_m \\ 0 \text{ otherwise} \end{cases}$$

5.7.3 SI Model Classification

The below model is an obvious adaptation of the seemingly most popular (and simple) graph-based disease-spreading model (from Wang et al., “Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint”)[270]. It is used as a baseline in our paper on within-network classification.

At regular time steps, an infected node can infect its neighbors with some time- and node-invariant probability β. In the SIS (Susceptible - Infected - Susceptible) model, each node may also be cured at each time step with probability δ. We assume that $\delta = 0$, resulting in the SI model.

Denote by $p_{(i,t)}$ the probability that node i is infected at time t (i.e., is already infected prior to time t, or gets infected at time t). Some nodes are ”born” infected (in our case, the original nodes for which there exists a positive label at time $t = 0$, plus possibly new nodes that ”immediately” are positive when they enter the network); we model this by assigning to each node i a time-invariant bit ξ_i such that $\xi_i = 1$ if the node is born infected and $\xi_i = 0$ otherwise.

Then:

$$p_{(i,t)} = \begin{cases} 1 \text{ if } \xi_i = 1 \\ p_{(i,t-1)} + (1 - p_{(i,t-1)}) * \left(1 - \prod_{j:\text{neighbor of } i} (1 - \beta * p_{(j,t-1)})\right) \text{ if } \xi_i = 0 \end{cases}$$

where we set $p_{(i,0)} = \xi_i$.

For a time-windowed graph, $(\mathcal{G}, \mathcal{T})$, the probability $p_{(i,t)}$ can be directly computed by the above formula for each time t, corresponding to the upper boundary of a window in $\mathcal{T}$. Observe that new nodes that are ”born” as time evolves affect the computation as well, not just the nodes present in the original graph.

Thus, one can classify each node at the final time $t = T$ as being infected or not by asking whether $p_{(i,T)}$ is above some threshold γ (e.g., $p_{(i,T)} > 0.5 = \gamma$). This threshold γ, and the infection parameter β are the only values that need to be tuned trough experiments. To keep things simple, we may keep $\gamma = 0.5$ and simply try different values of β until the fraction of predicted infected nodes is close to the fraction of the nodes we know to be infected in the data (e.g., the fraction of actors we know to be stars at $t = T$).

Note that as, e.g., stars in movies, are rare, we may end up with realistic values of β that are smaller than what is needed to maintain the disease (also called the "epidemic threshold"), estimated by Wang et al. to be the reciprocal of the largest eigenvalue of the adjacency matrix of the network. Accuracy, precision and recall, ROC, etc. can be directly calculated from the above classification of each node.

6 Weighted Temporal Graph Metrics

In this chapter we present work that has been submitted to the Network Science Journal in [38]. After experimenting with how to modify centrality metrics, allowing us to capture temporal or attribute information of nodes, I wanted to define metrics that could capture the same information, but also be tailored to other domains and problems. I began with adding functions of nodes to metrics to change their behavior, and then experimented with how to make the functions and metrics as general, or flexible, as possible.

6 Weighted Temporal Graph Metrics

Abstract

A temporal graph is a(multi-)graph where nodes and edges can be added over time, for example social or citation networks. A typical task involving temporal graphs is to predict which existing nodes will be of particular interest in the future (e.g., have high connectivity). Methods and metrics designed for static graphs can be used to treat temporal networks, but recent work has shown that established metrics for temporal graphs can be experimentally outclassed in tasks such as within-network classification by using metrics that take temporal or attribute information into account.

In this work we (**a**) review the shortcomings of existing metrics for static and temporal graphs; (**b**) present a theoretically principled extension of existing metrics incorporating *weights* in static and temporal graphs; (**c**) exemplify the use of weights for classification tasks in temporal graphs; and (**d**) perform an experimental validation of the new metrics by applying them to two distinct classification tasks in 3 datasets and comparing them to baselines using unweighted time-sensitive metrics.

6.1 Introduction

Graphs are used to model many types of real-world phenomena. Extensive research has been done aiming to engineer metrics for graphs, or nodes in a graph, for example with the goal of measuring how "central" or "well-connected" a node is. Such metrics have been successfully applied as descriptors, or as features in machine-learning systems, when attacking problems such as link prediction, anomaly detection, and within-network classification. In this work we focus solely on within-network classification: the task of assigning labels to nodes in the graph when incomplete information is available.

Real-world data is often complex: nodes have many types of additional information (e.g. age, or affiliation in a social network), and real-world graphs often change over time (e.g., friendships are created/removed in social networks, authors cite new articles in citation networks, and new emails are sent/received in communication networks). For many types of problems the additional information present in the data is critical to solving the problem. For example, when determining whether a node in an internet traffic network is infected with malware, it is important to know both the activity of the node over time (especially *recent* activity) and attributes such as hardware, country of origin, operating system, and so forth. Therefore, *static* graphs, and their associated metrics may fail to capture the temporal phenomena typical of real-world problems.

Extensive research has sought to extend concepts from static graphs to temporal graphs. Most existing metrics for temporal graphs, such as temporal betweenness, or closeness centrality, are similar to their static counterparts, but are subject to additional criteria such as all paths must be "time-respecting" [26, 8, 16, 279]. Prior work has shown that, for within-network classification, using temporal node metrics may in some cases outclass static metrics by effectively capturing temporal information as well as structural information [228, 83, 280, 26], however there exists only few works dealing solely with node metrics in complex temporal graphs [267, 8, 59].

A different approach is to solve within-network classification problems by relying on the attributes of neighboring nodes, *instead* of classical graph metrics [259]. These types of methods often rely on diffusing information from certain nodes to other nodes for which little is known. For example, in malware networks the diffusion approach has proven useful because of "guilt by association", that is, new threats are often identified based on connections to existing malware [81, 82, 281]. The strength of these types of approaches lies in the assumption of *homophily* of the underlying graph: that nodes are more likely to be connected to other nodes of the same type, or with the same labels. This approach allows, for example, collective classification methods to determine labels of a node based on those of the nodes in its immediate neighborhood [97]. However, these methods can fail on graphs with few links between nodes with similar labels, large distances between them in the graph, or low modularity overall. By modifying node metrics, like centrality or coverage, to take node weights into account, it is possible to label nodes based on information of nodes which are not in the immediate neighborhood.

Consider the graph in Figure 6.1, consisting of node labels (green/red) and numerical weights associated to said labels: weight 1 is assigned to green nodes, weight 0 to red nodes, and nodes x and y are unlabeled. The task is to label the currently unlabeled nodes x and y as either green or red.

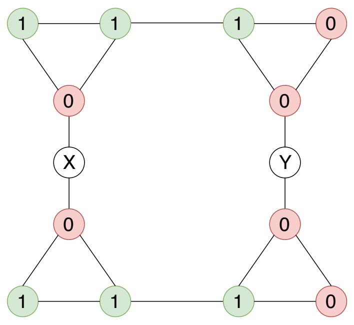


Figure 6.1: Within-network classification: Labels (and weights) of red- and green-labeled nodes are known. What colours should be assigned to nodes x and y?

In this example, traditional, static node metrics such as betweenness centrality and closeness centrality are identical for both x and y, and relational classifiers might mislabel x since none of the neighbors are green. However, by incorporating node weights in the definition of metrics e.g. closeness centrality, it possible to differentiate between x and y and correctly label them. Furthermore, recent work in time-varying graphs has shown that combining this type of node weights with information on "recent" topological changes in the network, can further improve the performance of classifiers [36].

We propose: a flexible way to parameterize existing graph metrics with a node weighting function (or just, "node function" for short) that can easily be adapted to several problem domains. A typical node function will be a simple, additional multiplicative factor in each metric, and can be based on topological or time-sensitive information in the graph (e.g. node weights, time-stamps of edges, etc.). This allows for straightforward customization of general graph metrics to specific problems or domains. Examples of customization include basing the metric calculations only on subsets of the most important nodes or weighing the most recently inserted edges higher in temporal paths.

Hence, for within-network classification on complex real-world temporal data, we posit that (i) using *weighted* temporal node metrics will lead to improved performance over purely unweighted static/temporal metrics, and that (ii) combining several types of weight functions may provide further valuable insights. We thus investigate the problem of within-network classification using novel weighted static and temporal node metrics.

We contribute: A new set of principled, weighted static and temporal node metrics, as well as several node weight functions, all designed for problems in temporal graphs, with within-network classification being one of the prime examples. The weighted metrics are designed such that: (i) They allow us to capture both graph-topological properties and node weights; (ii) the time and space complexity is the same as their unweighted counter-parts, *mutatis mutandis*; (iii) they allow for time-dependent weighting, for example based on the "recency" of edge or node additions; (iv) their flexibility allows for application to a variety of real-world domains with ease.

To show the usefulness of our new metrics, we perform within-network classification experiments on 3 real-world datasets, and we compare the accuracy of classification methods using the proposed metrics to baseline unweighted node metrics. Our key findings in the experiments are: (1) unweighted node metrics are in some cases insufficient for within-network classification (2) using weighted metrics with tailored node functions can lead to better performance, both static and temporal (3) the choice of node function for weighted metrics can present a trade-off between positive class precision and recall.

6.1.1 Related Work

There is a substantial volume of related work in which unweighted/weighted and static/temporal graph metrics have been used to solve a very eclectic set of problems. Below, we have categorized related work into four rough categories, based on whether the approach applies to static/temporal graphs, and unweighted/weighted metrics.

6.1.1.1 Static Graphs

Methods have been developed and used for solving a plethora of problems in static graphs for decades [282, 283, 89, 95]. The concept of centrality in networks was first introduced by Bavelas in the 1940s for communication networks [74], and extended for connected and unconnected networks by [75]. These metrics allowed for identifying "important" or "central" nodes in social networks in an intuitive way, based on the shortest paths between nodes, and the distances between nodes, in a graph. Since these definitions of centrality metrics, many variants have been proposed (for an overview see [76]) and have successfully been used for problems such as within-network classification [102, 92, 19, 26], link-prediction [209], and anomaly detection [225]. Extensions of centrality metrics, such as relative centrality that describes the relative importance of a node in respect to other nodes in the graph, have been designed to overcome centrality metrics' shortcomings for specific domains [284]. Other examples include routing betweenness centrality [285], centrality based on random walks [286], and process-driven betweenness centrality [287]. However, these metrics and methods typically do not perform well on weighted graphs [288]. Surveys on within-network classification and metrics in static graphs can be found in [19, 122, 95].

6.1.1.2 Static Attributed/Weighted Graphs

There have been numerous attempts at generalizing metrics, such as centralities from [75], to weighted graphs and attributed graphs [289, 290]. These are in most cases modeled as simple weighted graphs where edge/node weights are scalar numerical values. [288] have proposed a Dempster-Shafer based centrality which provides a trade-off between node strength and degree, and [291] have created the Laplacian centrality which is analogous to eigenvalue centrality for unweighted graphs. Separate types of metrics have been designed for graphs with other types of attributes, such as categorical attributes on nodes/edges [292, 293]. Real world datasets are often complex and sometimes lend themselves well to so-called multiplex, or multilayer, graphs, which has spawned another field of weighted graph metrics [294, 246, 295]. A survey on metrics for static weighted graphs can be found in [296, 297]. Various methods have been proposed to improve accuracy in classification tasks using attributes or weights. [298, 111] propose learning on sub-graphs which consist only of nodes which have target labels, but this can in label-sparse cases result in many isolated subgraphs, and also does not produce results for non-target nodes since they are omitted from subgraph. Other methods for attributed/weighted graphs include clustering using graph/attributes [299, 300, 205], identifying untrustworthy individuals in multi-agent communities from a combination of observable features and network connections [301, 266], pattern mining in attributed graphs [112], and classification based on topological and label attributes [79].

6.1.1.3 Temporal Graphs

The early approaches at modeling temporal graphs, used sequences of static graphs, sometimes called time-varying graphs (TVG) or time-windowed graphs (TWG), where each graph in the sequence is a static representation of the data as time passes [2]. While static metrics can be applied to each graph in the sequence, they do not capture sufficient temporal information of the underlying data [8]. Extensions of static metrics have been proposed in several works [25, 61, 8, 279, 302, 303]. Several metrics such as cover time, and time-constrained coverage, have been designed specifically to take advantage of the additional temporal information, and do not have "static" counter-parts [267, 59]. Applications of metrics on temporal graphs include using social temporal motifs (i.e. stars, ordered chains, ping-pong patterns) in social networks [132], using a custom ranking score to track the evolution of node features over time and identify rising stars [93, 304], and using temporal centrality metrics to characterize the evolution of graphs [26, 268, 16]. Surveys on applications of metrics on real-world networks can be found in [257, 18].

6.1.1.4 Temporal Attributed/Weighted Graphs

There are relatively few published works which regard both temporal and attribute data for node metrics [228, 225, 19, 122]. Most related work defines measures which typically are appropriate only for a few domains. [83] define a node metric which is weighted by the change of node attributes over time to identify influencers in social networks. [130] proposed to use the strength of social ties to weight centrality metrics in temporal weighted social networks, and [280] propose an entropy degree centrality metric to find influential nodes in mobile social networks. However both these methods model the temporal graph as a sequence of static graphs. Metrics can also be weighted by the temporal information in the graph, e.g. time-weighted node degree, where edges are weighted based on their respective ages, to predict future trends in data [245]. Applications in temporal weighted graphs include: [138] Classification in temporal weighted graphs; [305] Anomaly detection in graphs using attributes and temporal information.

Our proposed metrics are based on the abstract concept of a node function that can take many different forms depending on the problem at hand. Furthermore we show that the weighted metrics can benefit from combinations of several independent node functions. In comparison to the above metrics for static/temporal weighted graphs, our proposed metrics are flexible, can be used with both static and temporal graphs, and can be applied to a large range of domains and problems by modifying the node functions, or creating new types and tailoring them to fit. Compared to the static graph metrics in Section 6.1.1.1 and 6.1.1.2, our proposed metrics can be used on both static and temporal graphs. And in contrast to the metrics in 6.1.1.3 and 6.1.1.4, our metrics are not limited to just functions of node weights, but can be e.g. combinations of shortest path lengths and functions describing how recent edges around each node are.

6.2 Preliminaries and Definitions

The literature on time-varying graphs contains many variations in nomenclature, therefore we give full definitions appropriate for our domain of application below, many of which originate from [26, 8, 2] and have appeared in earlier yet unpublished work [37].

In the remainder of the paper, let $\mathbb{T}$ and $\mathbb{D}$ be totally ordered sets (representing, *time* and *data*, respectively).

Definition 29. *A **time-window** τ is a tuple $\tau = (t, t') \in \mathbb{T}^2$ where $t < t'$. Let $\mathcal{T} \subseteq \mathbb{T}^2$ be any non-empty set of time-windows. Then $\prec$, defined by:*

$$(t_i, t_j) \prec (t_a, t_b) \quad \text{if} \quad t_i < t_a \text{ or } (t_i = t_a \text{ and } t_j < t_b) \tag{25}$$

is a strict partial order on $\mathcal{T}$.

Definition 30. *A **temporal graph** G is a tuple (V, E), where V is the set of nodes of G and E is the set of edges. An edge $e \in E$ is a triplet (x, y, t) where $x, y \in V$ and $t \in \mathbb{T}$. In the remainder of the paper $e = (x, y, t)$ should be understood as a directed edge between x and y at time t. We usually represent a temporal graph by its adjacency matrix A, where $a_{xy} = 1$ iff there exists at least one edge from x and y in G*

Definition 31. *A **weighted temporal graph** G is a temporal graph (V, E, W), where V, E are the nodes and edges, respectively, and $W : V \mapsto \mathbb{W}$ are weights on nodes in V. For any node $x \in V$ there exists a weight $w_x \in W$.*

Definition 32. *A **stream** S is any subset $S \subseteq \mathbb{D}^2 \times \mathbb{T}$. A **substream** of a stream S is a subset $s \subseteq S$. The relation $\prec$, defined by:*

$$(x_i, y_i, t_i) \prec (x_j, y_j, t_j) \quad \textit{if} \quad t_i < t_j \tag{26}$$

is a strict partial order on the set of triplets. Intuitively, a stream is a succession of "events" (x, y) each happening at some time t. Typically, (x, y) represents the creation of a directed edge between nodes x and y in a graph.

Definition 33. *Let $\tau = (t, t')$ be a time-window and S be a stream. The **windowed substream** s_τ, induced by the time-window τ and stream S, is the substream s_τ of S defined by:*

$$s_\tau = \{(x, y, t) : (x, y, t) \in S \textit{ and } t \leq t < t'\} \tag{27}$$

Thus, the windowed substream intuitively is the set of all "events" occurring between t and t'.

Definition 34. *Let $\mathcal{S} = \{s_0, s_1, s_2 \cdots\}$ be a set of substreams of the stream S. $\mathcal{S}$ is then strictly partially ordered by $\prec$, defined by:*

$$s_i \prec s_j \quad \textit{if} \min_{(x_a, y_a, t_a) \in s_i} t_a < \min_{(x_b, y_b, t_b) \in s_j} t_b \tag{28}$$

which results in ordering the substreams by which contains the earliest event.

Definition 35. *Let s be a substream. The **substream graph** G_s is a temporal graph (V, E) whose nodes and edges are defined as:*

$$V = \bigcup_{(x,y,t) \in s} \{x, y\} \quad \textit{and} \quad E = \bigcup_{(x,y,t) \in s} \{(x, y, t)\} \tag{29}$$

*A special case of temporal substream graph is the **bipartite substream graph** denoted G_s^b, which is a bipartite temporal graph (V, U, E) whose nodes and edges are defined as:*

$$V = \bigcup_{(x,y,t) \in s} \{x\} \quad \textit{and} \quad U = \bigcup_{(x,y,t) \in s} \{y\} \quad \textit{and} \quad E = \bigcup_{(x,y,t) \in s} \{(x, y, t)\} \tag{30}$$

where we refer to V as the set of *top nodes* and U as the set of *bottom nodes*. It follows from the definition of bipartite graphs that $V \cap U = \emptyset$.

.

6.2.1 Time-Varying Graphs

Some metrics, such as average edge persistence, require modeling the stream as sequences of temporal graphs, where each graph represents a "state" of the stream during a specific period of time. A direct approach is to split the stream into consecutive windowed substreams, such that each substream induces a temporal graph. To do so we first define a type of graph sequence structure that allows for "batched" graph updates. Intuitively each temporal graph in the container is the result of a batch of changes applied to its predecessor. This type of graph is called a *time-varying graph.*

Definition 36. *A **time-varying graph** (TVG) $\mathcal{G}$ is a sequence of temporal graphs $(G_0, G_1, G_2 \cdots)$, where each graph G_{i+1} is constructed from its predecessor G_i by inserting additional nodes/edges or deleting existing nodes/edges. This definition differs from that of [8]: In our definition, $\mathcal{G}$ is based on temporal graphs instead of non-temporal graphs, and time-windows are not part of the time-varying graph.*

Definition 37. *Let $\mathcal{S} = (s_0, s_1, s_2 \cdots)$ be a sequence of substreams. We denote by $\mathcal{G}_\mathcal{S}$ the strictly partially ordered TVG $(G_{s_0}, G_{s_1}, G_{s_2} \cdots)$ induced by definition 35.*

Definition 38. *A **time-windowed graph** (TWG) is a pair $(\mathcal{G}, \mathcal{T})$, consisting of a time-varying graph $\mathcal{G}$ and a set of time-windows $\mathcal{T}$ where $|\mathcal{G}| = |\mathcal{T}|$. For each graph $G_i \in \mathcal{G}$ there exists a paired time-window $\tau_i = (t_m, t_n) \in \mathcal{T}$ so that for any edge $e = (x, y, t) \in G_i$, we have $t_m < t < t_n$. The* adjacency matrix *of a time-windowed graph is the map $A_{(\mathcal{G},\mathcal{T})}$ such that if $\tau_i \in \mathcal{T}$ is a time-window, then $A_{(\mathcal{G},\mathcal{T})}(\tau_i) = a_{xy}(\tau_i)$ is the adjacency matrix of the temporal graph for time-window τ_i.*

Definition 39. *Let $\mathcal{T} = \{\tau_0, \tau_1, \tau_2 \cdots\}$ be a set of time-windows and S be a stream. The time-windowed graph $(\mathcal{G}, \mathcal{T})$ induced from S and all time-windows in $\mathcal{T}$ consists of a sequence of (bipartite) substream graphs:*

$$\mathcal{G} = \{G_{s_{\tau_0}}, G_{s_{\tau_1}}, G_{s_{\tau_2}} \cdots\} \tag{31}$$

and satisfies:

$$\tau_0 \prec \tau_1 \prec \tau_2 \cdots \Rightarrow G_{s_{\tau_0}} \prec G_{s_{\tau_1}} \prec G_{s_{\tau_2}} \cdots \tag{32}$$

Definition 40. *Let $\mathbb{T} = \mathbb{R}_{\geq 0}$ and $\Delta\tau$ be a value in $\mathbb{T}$. A **time-windowed graph with fixed time-window size** (TWGFW) is a time-windowed graph $(\mathcal{G}, \mathcal{T})$ where each time-window $(t, t') \in \mathcal{T}$ satisfies:*

$$t' = t + \Delta\tau \tag{33}$$

and that for any two adjacent time-windows $\tau_{ij} = (t_i, t_j)$, $\tau_{mn} = (t_m, t_n) \in \mathcal{T}$, where $\tau_{ij} \prec \tau_{mn}$, we have $t_j = t_m$.

6.2.2 Labels in Time-Varying Graphs

As a main purpose of the paper is to predict future labels on *nodes*, time-varying information on nodes, in particular their labels, must be treated; hence the following definitions.

Definition 41. *Let $\mathbb{L}$ be a set. An* ***observed label*** *L is a triplet (x, l, t) where $x \in \mathbb{D}$, $l \in \mathbb{L}$ and $t \in \mathbb{T}$. In the remainder of the paper $L = (x, l, t)$ should be understood as x being assigned label l at time t.*

Definition 42. *A* ***labelstream*** *$\mathcal{L}$ is a set of observed labels $(L_0, L_1, L_2 \cdots)$ strictly partially ordered by $\prec$, defined by*

$$(x_i, l_i, t_i) \prec (x_j, l_j, t_j) \text{ if } t_i < t_j \tag{34}$$

6.2.3 Snapshots of Time-Windowed Graphs

For classification of nodes in a TVG based on streams, we introduce "snapshots" in order to clearly model the TVG at the "present" and "future".

Definition 43. *Let t_p be a value in $\mathbb{T}$, $\mathcal{T}$ be a set of time-windows, S be a stream, and $\mathcal{L}$ be a labelstream. A* ***snapshot*** *at time t_p is a quintuple $(S_{t_p}, \mathcal{G}_{t_p}, \mathcal{T}_{t_p}, G^a_{t_p}, \mathcal{L}_{t_p})$ where S_{t_p} is a substream of S, and $\mathcal{T}_{t_p}$ is a subset of $\mathcal{T}$, both consisting of elements timestamped before t_p. We denote $(\mathcal{G}_{t_p}, \mathcal{T}_{t_p})$ as the time-windowed graph induced by S_{t_p} and $\mathcal{T}_{t_p}$, and $G^a_{t_p}(V^a_{t_p}, E^a_{t_p})$ as the accumulated graph of $\mathcal{G}_{t_p}$. The labelstream subset $\mathcal{L}_{t_p} \subseteq \mathcal{L}$ consists of labels timestamped before t_p for nodes in $V^a_{t_p}$. A node $v_i \in V^a_{t_p}$ is said to be* labeled in the snapshot at time *t_p, if there exists a label $(v_i, l, t) \in \mathcal{L}_{t_p}$.*

6.2.4 Problem Definition

The classification problem tackled in this paper is described below.
Input: Let $(t_p, t_f) \in \mathbb{T}^2$ be a tuple where $t_p < t_f$, $\mathcal{T}$ be a set of time-windows, S be a stream, $\mathcal{L}$ be a labelstream with binary labels $l \in \{0, 1\}$, corresponding to negative and positive class labels, and $(S_{t_p}, \mathcal{G}_{t_p}, \mathcal{T}_{t_p}, G^a_{t_p}, \mathcal{L}_{t_p})$ be a snapshot at time t_p, referred to as the *present* snapshot, where $\mathcal{G}_{t_p}$ and $G^a_{t_p}$ may be bipartite.
Output: Let $V^0_{t_p} \subseteq V^a_{t_p}$ be the set of nodes in the present snapshot where for every

node $v_i \in V^0_{t_p}$ there does **not** exist an observed label $(v_i, l, t) \in \mathcal{L}_{t_p}$ where $l = 1$. For each node $v_i \in V^0_{t_p}$ predict if there exists an observed label (v_i, l, t) where $l = 1$ in the labelstream of the snapshot at time t_f, referred to as the *future* snapshot. For the remainder of the paper: Nodes in $V^a_{t_p}$ for which a positive label exists at t_p are referred to as *present positive nodes*, and nodes in $V^0_{t_p}$ for which a positive label exists at t_f are referred to as *future positive nodes*.

6.2.5 Projections of Time-Windowed Graphs

All three of the datasets, used for experiments later in this paper, are naturally bipartite graphs, but the metrics used are designed for non-bipartite graphs. While any bipartite graph can be viewed as a non-bipartite graph, by simply ignored top and bottom node associations, it is often better for modeling purposes to use projections from bipartite to non-bipartite graphs. We briefly give the necessary definitions below.

Definition 44. *Let $G(V, U, E)$ be a bipartite temporal graph, the **temporal projection** $G^p(V^p, E^p)$ of G, is the one-mode projection of G onto V. G^p is then a temporal graph containing only nodes from V, where two nodes are connected by an undirected edge when they have at least one common neighboring node in U. The edge is assigned the largest timestamp seen from either of the two nodes, to the common neighbor.*

A visualization of temporal projection is shown in figure 6.2. Note: by the above definition, any node without 2-hop neighbors in G will be isolated in G^p. A definition for temporal projection of a time-windowed graph is not required by the graph metrics in this paper, and thus omitted.

6.2.6 Shortest Temporal Paths

Some of the temporal node metrics used in our work are based on paths in temporal graphs. Using definitions from Nicosia et al.[8] we describe several types of paths in temporal graphs, collectively called shortest temporal paths, because each minimizes a distinct graph measure.

Definition 45. *Given a temporal graph $G(V, E)$, a **temporal path** from node x to y is defined as a sequence of edges*

$$P = ((n_0, n_1, t_1), (n_1, n_2, t_2), \cdots, (n_{l-1}, n_l, t_l)) \tag{35}$$

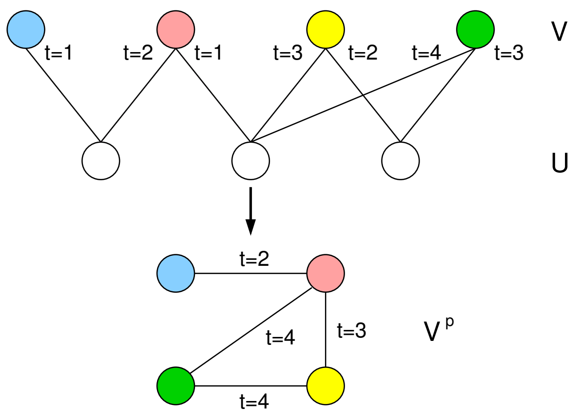


Figure 6.2: Example of a top-node bipartite temporal graph projection.

where $n_0 \equiv x$, $n_l \equiv y$, such that no nodes are visited more than once and $t_i \leq t_{i+1}$ for any two neighboring edges in the path. The length of a path $len(P)$ is the number of edges in the sequence and the duration of a temporal path is defined as $dura(P) = t_l - t_1$. A node y is said to be ***temporally reachable*** *from node x if there exists a temporal path from x to y.*

Definition 46. *Given the set of all temporal paths between node x and y, $\mathbf{P}_{xy}$, and a temporal graph $G(V, E)$, a temporal path $P \in \mathbf{P}_{xy}$ is a* ***shortest temporal path*** *if $len(P) = \min\{len(P') : P' \in \mathbf{P}_{xy}\}$. Similarly, P is a* ***fastest temporal path*** *if $dura(P) = \min\{dura(P') : P' \in \mathbf{P}_{xy}\}$. The* ***geodesic distance*** *between two nodes in a static graph is defined as the number of edges in a shortest path connecting them. For a temporal graph we define the* ***temporal geodesic distance*** *as the number of edges in a* temporal *shortest path connecting the pair of nodes.*

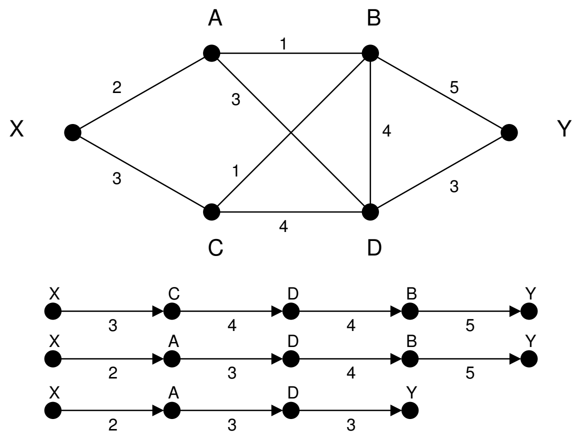


Figure 6.3: All temporal paths from X to Y.

6.3 Methodology

We now present the baseline node metrics as well as our proposed weighted node metrics and the node functions used for our experiments. In Section 6.3.1 we introduce node functions which will later be used to weight node metrics proposed in 6.3.2.

6.3.1 Weights

On a graph with weights on nodes, we handle both weighted edges and generalization of metrics by introducing a single extra multiplicative factor to existing metrics. As the local neighborhood of a node is dictated by the edges incident on the node (or short paths from the node), and as most metrics already perform aggregation in their definition, weighting information about the neighborhood can be expressed using aggregation over very simple functions: we can make do with the multiplicative factor being a function of a pair of nodes $I(x, y)$, and this can be adapted and inserted into the metrics on a case-by-case basis. We now define four prototypical "node functions" taking the place of $I(x, y)$, leading to four different types of weighted metrics, exemplifying various approaches to defining $I(x, y)$. Many other node functions are possible, but we believe that these four are intuitive and prove their effectiveness well.

For simplicity, we assume that weights are real numbers; let $I(x, y) \mapsto \mathbb{R}^+$ be a function of two nodes x, y that returns a value in $\mathbb{R}^+$.

1. (Type I): This node function is based on the node weights on the nodes passed to it. The node weights are multiplied, allowing for operations such as calculating metrics on only a subset of nodes of the graph by assigning binary weights (0/1) to nodes. The node function is defined as:

$$I(x, y) = w_x w_y$$

2. (Type II): Like the first node function, type II can also allow us to calculate node metrics on just a subset of nodes. However, in contrast to type I, the contribution from nodes with low weights, e.g. node weight 0, is not completely disregarded, but instead averaged. This can be useful to avoid node metrics evaluating to zero, by performing a "softer" node subset selection.

The function is defined as:

$$I(x,y) = \frac{w_x + w_y}{2}$$

3. (Type III): The function takes on values in the range $[0,1]$ and is based on the concept of recency from [137, 36] which weighs edges higher the more "recent" they are. The closer $I(x,y)$ is to 1, the more recent the path between x and y is. The function is defined as:

$$I(x,y) = \frac{1}{|P(x,y)|} \sum_{p \in P(x,y)} \sum_{(u,v,t) \in p} \frac{|p|}{\log_2(2 + (t_p - t))}$$

where $P(x,y)$ is the set of shortest temporal paths from x to y. Note that this node function is only applicable to temporal graphs.

4. (Type IV) We also base type IV on paths between nodes, but in contrast to type III, it is based on the weights on node in the paths, not the edges, and can be used for both temporal and static graphs. The function returns the average of node weights encountered on all shortest paths between x and y, and is therefore larger the more nodes with large weights are visited. It is defined as follows:

$$I(x,y) = \frac{1}{|P(x,y)|} \sum_{p \in P(x,y)} \frac{w_x + \sum_{(u,v,t) \in p} w_v}{|p|}$$

where $P(x,y)$ is the set of shortest paths from x to y.

Next we define both unweighted and weighted graph metrics for static and temporal graphs which utilize $I(x,y)$.

6.3.2 Metrics

We define four variants of metrics: Static unweighted, static weighted, temporal unweighted, and temporal weighted. In Section 6.3.2.1 are the static metrics, and in Section 6.3.2.2 are the temporal metrics. The following metrics are from Nicosia et al.: temporal betweenness, temporal closeness, and edge-persistence [8], while cover time and constrained coverage are inspired by articles from Costa et al. and Takaguchi et al. [267, 59]. Cover time and constrained coverage were originally proposed as "time-centrality" metrics, analogous to betweenness and closeness centrality, and were thus included in our work. For each of the metrics, we first describe their "original" unweighted definition, then how they each are modified into weighted node metrics. For all metrics we assume a non-empty graph, i.e. for a temporal graph $G = (V, E)$:

$$|V| \neq \emptyset \qquad \text{and} \qquad |E| \neq \emptyset$$

and for temporal metrics we assume time-windowed graphs contain more than 1 time-window, i.e. for a TWG $(\mathcal{G}, \mathcal{T})$: $1 < |\mathcal{T}|$. An overview of the metrics is presented in Table 6.1.

Metric	Unweighted	Weighted
Static		
Betweenness Centrality	$C_x^B = \begin{cases} \sum_{\substack{y\in V\\ y\neq x}} \sum_{\substack{z\in V\\ z\neq x\\ z\neq y}} \frac{\sigma_{yz}(x)}{\sigma_{yz}} & \text{if } \sigma_{yz}\neq 0 \\ 0 & \text{otherwise}\end{cases}$	$C_x^{WB} = \begin{cases} \sum_{\substack{y\in V\\ y\neq x}} \sum_{\substack{z\in V\\ z\neq x\\ z\neq y}} \frac{\sigma_{yz}(x)I(y,z)}{\sigma_{yz}} & \text{if } \sigma_{yz}\neq 0 \\ 0 & \text{otherwise}\end{cases}$
Closeness Centrality	$C_x^O = \begin{cases} \frac{\lvert V\rvert-1}{\sum_y d_{xy}} & \text{if } \sum_y d_{xy}\neq 0 \\ 0 & \text{otherwise}\end{cases}$	$C_x^{WO} = \begin{cases} \frac{\lvert V\rvert-1}{\sum_y I(x,y)d_{xy}} & \text{if } \sum_y I(x,y)d_{xy}\neq 0 \\ 0 & \text{otherwise}\end{cases}$
Cover Time	$\underset{\alpha\in[0,1]}{C_x^T(\alpha) = \begin{cases} d^t(x,\alpha) & \text{if } \alpha \text{ is reached} \\ \infty & \text{otherwise}\end{cases}}$	$\underset{\alpha^w\in\mathbb{R}^+}{C_x^{WT}(\alpha^w) = \begin{cases} d^{wt}(x,\alpha^w) & \text{if } \alpha^w \text{ is reached} \\ \infty & \text{otherwise}\end{cases}}$
Constrained Coverage	$\underset{\phi\in[0,\lvert V\rvert-1]}{C_x^C(\phi) = d^c(x,\phi)}$	$\underset{\phi^w\in[0,\lvert V\rvert-1]}{C_x^{WC}(\phi^w) = d^{wc}(x,\phi^w)}$
Clustering	$C_x^U = \begin{cases} \frac{2T_x}{\deg_x(\deg_x-1)} & \text{if } 1<\deg_x \\ 0 & \text{otherwise}\end{cases}$	$C_x^{WU} = \begin{cases} \frac{\sum_{(x,y,z)\in T_x} I(x,z)+I(x,y)}{\deg_x(\deg_x-1)} & \text{if } 1<\deg_x \\ 0 & \text{otherwise}\end{cases}$
Temporal		
Betweenness Centrality	$C_x^{TB} = \begin{cases} \sum_{\substack{y\in V\\ y\neq x}} \sum_{\substack{z\in V\\ z\neq x\\ z\neq y}} \frac{\sigma^t_{yz}(x)}{\sigma^t_{yz}} & \text{if } \sigma^t_{yz}\neq 0 \\ 0 & \text{otherwise}\end{cases}$	$C_x^{TWB} = \begin{cases} \sum_{\substack{y\in V\\ y\neq x}} \sum_{\substack{z\in V\\ z\neq x\\ z\neq y}} \frac{\sigma^t_{yz}(x)I(y,z)}{\sigma^t_{yz}} & \text{if } \sigma^t_{yz}\neq 0 \\ 0 & \text{otherwise}\end{cases}$
Closeness Centrality	$C_x^{TO} = \begin{cases} \frac{\lvert V\rvert-1}{\sum_y d^t_{xy}} & \text{if } \sum_y d^t_{xy}\neq 0 \\ 0 & \text{otherwise}\end{cases}$	$C_x^{TWO} = \begin{cases} \frac{\lvert V\rvert-1}{\sum_y I(x,y)d^t_{xy}} & \text{if } \sum_y d^t_{xy}\neq 0 \\ 0 & \text{otherwise}\end{cases}$
Cover Time	$\underset{\alpha\in[0,1]}{C_x^{TT}(\alpha) = \begin{cases} d^{tt}(x,\alpha) & \text{if } \alpha \text{ is reached} \\ \infty & \text{otherwise}\end{cases}}$	$\underset{\alpha^w\in\mathbb{R}^+}{C_x^{TTW}(\alpha^w) = \begin{cases} d^{wtt}(x,\alpha^w) & \text{if } \alpha^w \text{ is reached} \\ \infty & \text{otherwise}\end{cases}}$
Constrained Coverage	$\underset{\phi\in\mathbb{T}^+}{C_x^{TC}(\phi) = d^{tc}(x,\phi)}$	$\underset{\phi^w\in\mathbb{T}^+}{C_x^{TWC}(\phi^w) = d^{twc}(x,\phi^w)}$
Edge-Persistence	$C_x^{TU} = \frac{1}{\lvert\mathcal{T}\rvert-1}\sum_{m=0}^{\lvert\mathcal{T}\rvert-1} C_x^{ep}(G_{\tau_m},G_{\tau_{m+1}})$ $C_x^{ep}(G_m,G_n) =$ $\begin{cases} \frac{\sum_y a_{xy}(\tau_m)a_{xy}(\tau_n)}{\sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}} & \text{if } \sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}\neq 0 \\ 0 & \text{otherwise}\end{cases}$	$C_x^{TWU} = \frac{1}{\lvert\mathcal{T}\rvert-1}\sum_{m=0}^{\lvert\mathcal{T}\rvert-1} C_x^{ep}(G_{\tau_m},G_{\tau_{m+1}})$ $C_x^{wep}(G_m,G_n) =$ $\begin{cases} \frac{\sum_y a_{xy}(\tau_m)a_{xy}(\tau_n)I(x,y)}{\sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}} & \text{if } \sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}\neq 0 \\ 0 & \text{otherwise}\end{cases}$

Table 6.1: Overview of static/temporal unweighted/weighted metrics for a node x. The static are all node metrics for static graphs, while the temporal are node metrics for temporal graphs, *except* edge-persistence which is only applicable to time-windowed graphs.

6.3.2.1 Static

Static betweenness centrality is a node metric which describes the centrality of a node in a graph using shortest paths. It has been applied to a wide variety of network problems such as fault tolerance in electrical grids and finding influencers in social networks, and is defined as follows:

$$C_x^B = \begin{cases} \sum_{\substack{y \in V \\ y \neq x}} \sum_{\substack{z \in V \\ z \neq x \\ z \neq y}} \frac{\sigma_{yz}(x)}{\sigma_{yz}} & \text{if } \sigma_{yz} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

where σ_{yz} is the number of shortest paths between node x and z, and $\sigma_{yz}(x)$ is the number of those paths that pass through x. This means, the more a node appears in shortest paths between pairs of nodes, the higher its centrality is. In the weighted version, **static weighted betweenness centrality**, we weigh the number of paths from y to z with the node function of them:

$$C_x^{WB} = \begin{cases} \sum_{\substack{y \in V \\ y \neq x}} \sum_{\substack{z \in V \\ z \neq x \\ z \neq y}} \frac{\sigma_{yz}(x) I(y,z)}{\sigma_{yz}} & \text{if } \sigma_{yz} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

where as before σ_{yz} is the number of shortest paths between node x and z, and $\sigma_{yz}(x)$ is the number of those paths that pass through x.

Static closeness centrality for a node is also based on shortest paths in the graph, but in contrast to betweenness, is calculated using shortest paths from that node to every single other node in the graph. Closeness centrality for a node x is calculated using the length of these shortest paths, and is defined as follows:

$$C_x^O = \begin{cases} \frac{|V|-1}{\sum_y d_{xy}} & \text{if } \sum_y d_{xy} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

where d_{xy} is the length of the shortest path between x and y. The **static weighted**

closeness centrality of node x is defined as:

$$C_x^{WO} = \begin{cases} \frac{|V|-1}{\sum_y I(x,y)d_{xy}} & \text{if } \sum_y I(x,y)d_{xy} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{39}$$

where the inverse length of the shortest paths from x to y are weighted by the node function between x and y.

There is no common definition of ***static*** **cover time**, however we argue that a static cover metric for a node x, analogous to cover time, can be defined as the minimum number of edges distance from x needed in order to reach $\alpha \in [0,1]$ fraction of nodes:

$$C_x^T(\alpha) = \begin{cases} d^t(x,\alpha) & \text{if } \alpha \text{ is reached} \\ \infty & \text{otherwise} \end{cases} \tag{40}$$

where $d^t(x,\alpha)$ is the minimum geodesic distance from x needed in order to reach a fraction α of nodes $V_\alpha \subseteq V \setminus \{x\}$. The corresponding **static weighted cover time** for a node x is given by:

$$C_x^{WT}(\alpha^w) = \begin{cases} d^{wt}(x,\alpha^w) & \text{if } \alpha^w \text{ is reached} \\ \infty & \text{otherwise} \end{cases} \tag{41}$$

where $d^{wt}(x,\alpha^w)$ is the minimum geodesic distance from x needed in order to reach a set of nodes V_{α^w}, by breadth-first-search, so that the sum of node weight functions $\sum_{y \in V_{\alpha^w}} I(x,y)$ is greater than, or equal, to $\alpha^w \in \mathbb{R}$.

Similarly we define a **static constrained coverage** based on a diffusion process which only can travel a limited number of edges away. Given a node x, the constrained coverage $C_x^{TC}(\phi)$ returns the fraction of nodes which can be reached within a geodesic distance of $\phi \in [0, |V|-1]$, i.e.

$$C_x^C(\phi) = d^c(x,\phi) \tag{42}$$

where $d^c(x,\phi)$ is the number of nodes which have geodesic distance $\leq \phi$ to x. The weighted counterpart, **static weighted constrained coverage**, for a node x is then defined as:

$$C_x^{WC}(\phi^w) = d^{wc}(x,\phi^w) \tag{43}$$

where $d^{wc}(x, \phi^w)$ is the sum of node weight functions $I(x, y)$ of the nodes $y \in V$ which have geodesic distance $\leq \phi^w \in [0, |V| - 1]$ to x.

The **Static clustering coefficient**, or local clustering coefficient, measures how many edges are in the immediate neighborhood of a node, and can be used to determine how "tightly knit" a cluster of nodes are. Clustering for a node x is calculated by counting how many triangles are formed with x and is defined as:

$$C_x^U = \begin{cases} \frac{2T_x}{\deg_x(\deg_x - 1)} & \text{if } \deg_x(\deg_x - 1) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

where $T_x(t)$ is the number of triangles through node x and $\deg_x(t)$ is the degree of x. The **static weighted clustering coefficient** is weighted by applying the node function to each pair of nodes which, in combination with x, form a triangle:

$$\begin{cases} C_x^{WU} = \frac{\sum_{(x,y,z) \in T_x} I(x,z) + I(x,y)}{\deg_x(\deg_x - 1)} & \text{if } \deg_x(\deg_x - 1) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

6.3.2.2 Temporal

Temporal betweenness centrality is an extension of the static betweenness centrality to temporal graphs. The temporal version uses temporal shortest paths in place of non-temporal shortest paths, and is defined as:

$$C_x^{TB} = \begin{cases} \sum_{\substack{y \in V \\ y \neq x}} \sum_{\substack{z \in V \\ z \neq x \\ z \neq y}} \frac{\sigma_{yz}^t(x)}{\sigma_{yz}^t} & \text{if } \sigma_{yz}^t \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{46}$$

where σ_{yz}^t is the number of shortest temporal paths between node x and z, and $\sigma_{yz}^t(x)$ is the number of those paths that pass through x. In the weighted version, **temporal weighted betweenness centrality**, we weigh the number of paths from y to z with the node function of them:

$$C_x^{TWB} = \begin{cases} \sum_{\substack{y \in V \\ y \neq x}} \sum_{\substack{z \in V \\ z \neq x \\ z \neq y}} \frac{\sigma_{yz}^t(x) I(y,z)}{\sigma_{yz}^t} & \text{if } \sigma_{yz}^t \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

where as before σ^t_{yz} is the number of shortest paths between node x and z, and $\sigma^t_{yz}(x)$ is the number of those paths that pass through x.

The **temporal closeness centrality** is defined as in equation 38, but with temporal shortest paths, and, for a node x at time t, defined as:

$$C^{TO}_x = \begin{cases} \frac{|V|-1}{\sum_y d^t_{xy}} & \text{if } \sum_y d^t_{xy} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{48}$$

where d^t_{xy} is the length of the shortest temporal path between x and y. **Temporal weighted closeness centrality** of node x at time t is weighted similarly to the static counterpart:

$$C^{TWO}_x = \begin{cases} \frac{|V|-1}{\sum_y I(x,y) d^t_{xy}} & \text{if } \sum_y d^t_{xy} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{49}$$

Temporal cover time is originally a graph metric defined as the *average* number of time-steps a diffusion takes to reach a specified fraction $\alpha \in [0,1]$ of nodes in the temporal graph following a breadth-first search starting at a specific time [267, 59]. We define cover time as a node-metric for a node x as:

$$C^{TT}_x(\alpha) = \begin{cases} d^{tt}(x,\alpha) & \text{if } \alpha \text{ is reached} \\ \infty & \text{otherwise} \end{cases} \tag{50}$$

where $d^{tt}(x,\alpha)$ is the amount of time (number of time-steps) for a diffusion starting on node x at time t, where t is the earliest time-stamp of x's edges, to reach a fraction $\alpha \in [0,1]$ of nodes. **Temporal weighted cover time** for a node x is defined as:

$$C^{TTW}_x(\alpha^w) = \begin{cases} d^{wtt}(x,\alpha^w) & \text{if } \alpha^w \text{ is reached} \\ \infty & \text{otherwise} \end{cases} \tag{51}$$

where $d^{wtt}(x,\alpha^w)$ is the minimum number of time-steps for a diffusion starting on node x at time t, where t is the earliest time-stamp of x's edges, to reach a set of nodes V_{α^w}, so that the sum of node weight functions $\sum_{y \in V_{\alpha^w}} I(x,y)$ is greater than, or equal, to $\alpha^w \in \mathbb{R}^+$.

Similarly, the **temporal constrained coverage** is originally the average fraction of nodes which can be reached by a diffusion process in specified amount of time [267, 59]. We define temporal constrained coverage as a node-metric as follows: $C_x^{TC}(\phi)$ returns the fraction of nodes which can be reached from x starting at time t, where t is the earliest time-stamp of x's edges, in $\phi \in \mathbb{T}^+$ number of time steps, i.e.

$$C_x^{TC}(\phi) = d^{tc}(x, \phi) \tag{52}$$

where $d^c(t, x, \phi)$ is the number of nodes reached from node x after ϕ time steps. The **weighted temporal constrained coverage** for a node x at time t is:

$$C_x^{TWC}(\phi^w) = d^{twc}(x, \phi^w) \tag{53}$$

where $d^{wc}(x, \phi^w)$ is the sum of node weight functions $I(x, y)$ of the nodes $y \in V$ reached from x after ϕ^w time steps.

Edge persistence [61] for node x in a time-windowed graph G is defined as follows: given two temporal graphs, $G_m(V_m, E_m)$ and $G_n(V_n, E_n)$ in G, the edge persistence for x is the number of edges that persist across both graphs:

$$C_x^{ep}(G_m, G_n) = \begin{cases} \frac{\sum_y a_{xy}(\tau_m) a_{xy}(\tau_n)}{\sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}} & \text{if } \sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{54}$$

In a time-windowed graph $(\mathcal{G}, \mathcal{T})$, the **average edge persistence** of node x is defined as the average over all pairs of adjacent temporal graphs in $\mathcal{G}$:

$$C_x^{TU} = \frac{1}{|\mathcal{T}| - 1} \sum_{m=0}^{|\mathcal{T}|-1} C_x^{ep}(G_{\tau_m}, G_{\tau_{m+1}}) \tag{55}$$

where G_{τ_m} and $G_{\tau_{m+1}}$ are adjacent temporal graphs in $\mathcal{G}$, and $G_{\tau_m} \prec G_{\tau_{m+1}}$. The

weighted edge persistence for node x is given by:

$$C_x^{wep}(G_m, G_n) = \begin{cases} \frac{\sum_y a_{xy}(\tau_m) a_{xy}(\tau_n) I(x,y)}{\sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]}} & \text{if } \sqrt{\left[\sum_y a_{xy}(\tau_m)\right]\left[\sum_y a_{xy}(\tau_n)\right]} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{56}$$

where each persisting edge's contribution has been weighted by the node function of its end-points. The **average weighted edge persistence** C_x^{TWU} is defined as in equation 55.

6.3.3 Examples

In this part we present two small example networks in Figures 6.4 and 6.5, and then apply all metrics from Section 6.3.2 to two selected nodes, X and Y, one from each network. Each of the networks consists of a set of nodes, time-stamped edges, binary node weights, and two time-windows each. The accumulated graph is used for calculating all metrics except edge-persistence and weighted edge-persistence where the time-windowed graph is used instead. The resulting node metric values are shown in Table 6.2. One significant difference between the two examples is that since node X has weight 0, node function I results in zero-valued metrics because the first term in the node function is always zero. We also see that node Y is part of zero shortest paths in the graph, and therefore has zero-valued betweenness centrality. For node Y, the temporal cover time is zero for all node functions except type III. This is as expected as there is only one "recent" edge to a node with weight 1, therefore the cover time using a recency node function is higher than the other types.

6.3.4 Case Studies

We now present three examples of weighted metrics, each with distinct node functions, applied to three different real-world datasets in the context of within-network classification. The node functions applied to each dataset have each been selected in order to capture some phenomena in the data which might improve the performance of a classifier. These datasets are the same that are used for experiments in Section 6.4.

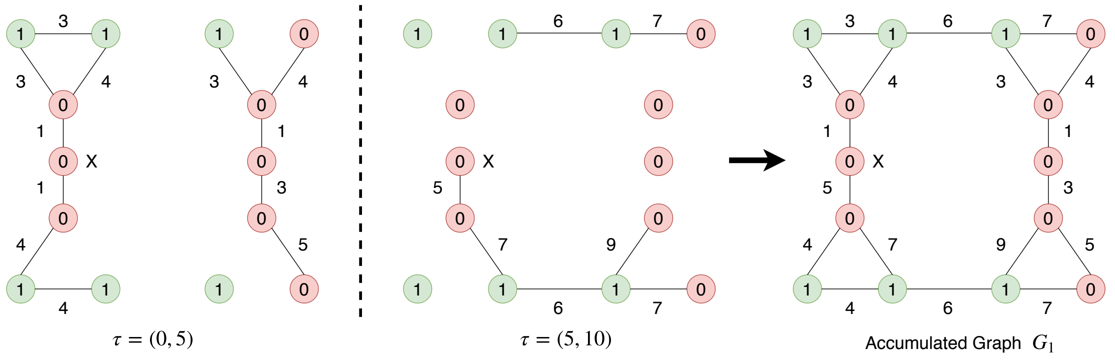


Figure 6.4: Time-windowed graph with two windows on the left, and the accumulated graph G_1 at time $t = 10$ on the right.

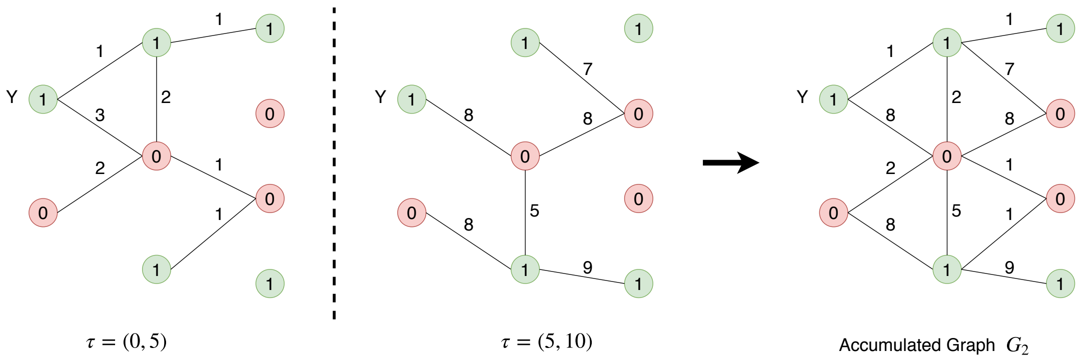


Figure 6.5: Time-windowed graph with two windows on the left, and the accumulated graph G_2 at time $t = 10$ on the right.

	Static				Temporal				
	U	I	II	IV	U	I	II	III	IV
	Figure 6.4 node X								
Betweenness	36	14	21	12.66	2	0	0.50	0.67	0.25
Closeness	0.35	0	1.86	0.95	0.87	0	2.89	2.57	2.55
Clustering	0	0	0	0	0.35	0	0	0.13	0
Cover Time	1	inf	3	4	2	inf	6	6	6
Constrained Coverage	2	0	0	0	3	0	1	0.90	0.83
	Figure 6.5 node Y								
Betweenness	0	0	0	0	0	0	0	0	0
Closeness	0.53	1	0.70	0.85	0.57	1	0.73	1.63	0.83
Clustering	1	0.50	0.75	0.25	0.35	0	0.18	0.18	0.18
Cover Time	1	2	2	2	0	0	0	8	0
Constrained Coverage	2	1	1.50	1.50	4	2	3	1.17	3.17

Table 6.2: Node metrics for one node from each of the graphs in Figures 6.4 and 6.5. The rows indicate the node metric, and columns indicate whether it is the unweighted (U) version, or a node function (I, II, III, or IV) is used. Parameters used for metrics: $\alpha = 0.2, \alpha^w = 3, \phi = 1, \phi^w = 1$ for static metrics, and $\alpha = 0.2, \alpha^w = 2, \phi = 1, \phi^w = 1$ for temporal metrics.

6.3.4.1 IMDb: Identifying future movie stars

The first dataset is an online actor-movie database, IMDb, consisting of movies and which actors participated in them. Furthermore, for each movie there exists a rating between 0 and 10 based on user reviews on the website. The graph representation of the dataset is defined as a set of nodes, where each node represents an actor, and edges that connect actors if they have participated in a movie together. Details can be found in Section 6.3.4.1. The challenge in this example is to predict which less "successful" actors will achieve success in the future. We define an actor as successful when they act in a movie with a high rating. If we assume future successful actors more often associate with currently successful actors (i.e. homophily), then using node function type I, II or IV might assist in identifying these future stars if the node weights are chosen appropriately. To illustrate this, we pick node function I, and assign a weight of 1 to all nodes which have participated in a movie with rating 7 or higher, and a weight of 0 otherwise. The weight is therefore an indication of whether or not the actor is successful. Let V_1 designate all nodes with weight 1. When applying the first node function $I(x, y) = w_x w_y$ the metrics for a node x capture the following:

- Betweenness centrality: Only considers paths between pairs of nodes in V_1. Centrality is higher the more the node "connects" otherwise disconnected subgraphs of successful actors - possibly indicating participation in many different movies with already successful actors.
- Closeness centrality: Represents normalized distance to nodes in V_1 i.e. how "close" the actor is to other successful actors.
- Cover time: Calculates the number of edges needed to reach a fraction α of nodes in V_1 and is similar to closeness centrality. In combination with closeness centrality, the temporal cover time captures the information about the age of the actor e.g. how long time they have had close ties to existing nodes in V_1.
- Constrained coverage: Returns the fraction of all nodes, that can be reached within a certain distance, which are in V_1. Temporal coverage captures information about how many highly rated movies actor x has participated in within a specified window of time.
- Clustering: Only counts triangles through x where both the other nodes are

in V_1 and can be used for information about the success of the immediate neighborhood around x. Temporal clustering (edge persistence) is indicative of repeated participation in movies with the same successful actors.

6.3.4.2 CiteULike: Identifying active users in community

The second dataset, CiteULike, is a network of users which assign tags, such as category, genre, etc., to publications [274]. The nodes in the network represent the users, and two nodes are connected by an edge if they have tagged (co-tagged) the same publication. Details on the dataset can be found in Section 6.3.4.2. Given a set of tags, the goal in this example is to identify users who have not yet used any of the tags in the set, but in the near future will use at least one tag from the set. CiteULike, like many online portals, hosts many users which over time become inactive for any number of reasons. Using node function type III it is possible to weigh nodes which have recently been active higher, and thereby "filter out" inactive users. In contrast to the previous example, whether or not the nodes have weights, is not of importance since node function III does not utilize them. Furthermore, node function III assumes a temporal graph, therefore only temporal metrics are discussed below. When applying this node function, the metrics for a node x capture the following:

- Temporal Betweenness centrality: Centrality is higher the more recent the edges in the shortest paths through node x are - possibly indicating recent increase in "important" connections.
- Temporal Closeness centrality: Represents average distance to all nodes in graph, where each distance is weighted by the age of the edges. Informally a node with high centrality will have had a recent connections to most other nodes in the graph.
- Cover time: Captures how many recent temporal paths exist from node x to its neighborhood. Lower values could indicate "inactive" users while midrange values could indicate user only maintains contact with few of the total number of acquaintances.
- Constrained coverage: Returns the number of nodes that can be reached within a certain "temporal distance", weighted by how recent the temporal paths to these nodes are.

- Clustering: As with the previous example, edge persistence indicates repeated co-tagging with the same users, weighted by how recent the repetition took place. Could be used to identify groups of bots (fake user accounts) which have been tasked with tagging specific documents with desirable tags.

6.3.4.3 DBLP: Identifying data mining authors

The final dataset is from an online database of articles abbreviated DBLP. It consists of authors in computer science, their published articles, and which other papers are referenced by each article. Each article published also contains information about which venue it was accepted to. The graph representation of the dataset is defined as a set of nodes, where each node represents an author, and edges connecting pairs of authors if they have referenced each other. Details can be found in Section 6.3.4.3. Given a set of venues belonging to the category ″Database and Data Mining″, the goal is to identify authors which have not yet published in those venues, but will do so in the near future. In this case it is not enough just to track which authors are active recently, because there will be many which publish to completely unrelated fields. However, combining trends on which authors have been active recently, with information on which scientific communities, or which types of authors, they have collaborated with, can help identify authors who will soon publish in the desired venues. We are able to capture this by using a combination of node function type I and III. To illustrate, we assign a weight of 1 to all authors which have published a paper in a set of venues collectively named ″Database and Data Mining″, or DBDM, and 0 to all other authors. As with the first example, V_1 denotes all nodes with weight 1. The combination of node function I and III results in the node function:

$$I(x,y) = \frac{w_x w_y}{|P(x,y)|} \sum_{p \in P(x,y)} \sum_{(u,v,t) \in p} \frac{|p|}{\log_2(2 + (t_p - t))}$$

When applying this combination, again assuming a temporal graph, the metrics for a node x capture the following:

- Betweenness centrality: Only considers paths between pairs of nodes in V_1. Centrality is higher the more the node recently has “connected” otherwise disconnected groups of researchers - possibly indicating that node x is an important middleman between many different DBDM research groups.

- Closeness centrality: Represents normalized distance to nodes in V_1 i.e. how close the author is to other DBDM authors, where centrality is higher the more recent the connections to DBDM authors are.
- Cover time: Calculates the number of edges needed reach a fraction α of nodes in V_1 and is similar to closeness centrality. In combination with closeness centrality, the temporal cover time captures the information about the age of an author e.g. how long time they have had close ties to existing nodes in V_1.
- Constrained coverage: Returns the fraction of all nodes, that can be reached within a certain distance, which are in V_1, and captures information about how many DBDM publications author x has referenced recently.
- Clustering: Only counts triangles through x where both the other nodes are in V_1 and can be used for information about how involved the immediate neighborhood around x is in the DBDM field. Temporal clustering (edge persistence) is indicative of repeated references to significant DBDM articles.

6.4 Experimental Design

We now describe our experimental design and setup[3]. Our approach is to define several feature sets that take time-related and weight-related information into account, and compare them experimentally to each other.

We define seven feature sets containing node metrics from Section 6.3.2, where each feature set represents a specific way of extracting information from the graph. The first two feature sets, named **S** and **T**, are calculated for each dataset, and together form the baselines which will be compared to feature sets of metrics using node functions. **S** contains all unweighted static node metrics, and **T** contains all unweighted temporal node metrics.

[3]which is similar to that of work done in [36], but with different node metrics / features.

	C_x^B	C_x^O	C_x^T	C_x^C	C_x^U	C_x^{WB}	C_x^{WO}	C_x^{WT}	C_x^{WC}	C_x^{WU}	C_x^{TB}	C_x^{TO}	C_x^{TT}	C_x^{TC}	C_x^{TU}	C_x^{TWB}	C_x^{TWO}	C_x^{TTW}	C_x^{TWC}	C_x^{TWU}
S	●	●	●	●	●															
T											●	●	●	●	●					
I1						●	●	●	●	●										
I2						●	●	●	●	●										
I4						●	●	●	●	●										
C																●	●	●	●	●
D																●	●	●	●	●

Table 6.3: Feature Configurations. Feature sets are in the first column, and node metrics are in the first row. • indicates that the feature is included in corresponding feature set.

The four remaining feature sets are named **I1**, **I2**, **I4**, **D**, and **C**. The first three feature sets **I1**, **I2**, and **I4** are for the IMDb dataset, **D** is for DBLP, and finally **C** is for CiteULike. The purpose of each feature set is to capture the phenomenon described in Section 6.3.4. Therefore **I1**, **I2**, and **I4** all consist of all static weighted node metrics, where **I1** uses node function type I, **I2** uses node function type II, and **I4** uses node function type IV, set **C** contains all temporal weighted node metrics using node function III, and finally **D** contains all temporal weighted node metrics using a combination of node function I and III. We could use all node functions for all metrics over all datasets, but the purpose of our experiments is to showcase node-functions tailored to varied domains. An overview of the resulting feature sets is shown in Table 6.3.

6.4.1 Classifier selection

To compare the feature sets above, we use a decision tree classifier from the well-known python library: Scikit-Learn. We use this method because of its prevalence in related literature [19]. One might expect that neural networks using advanced deep learning might produce better results, however due to aspects such as label-imbalance and parameter tuning, which complicate deep learning experiment setup, and that the focus of this paper is comparison of node features, we leave deep learning to future work. The decision tree is limited to a maximum depth of 6, based on the highest F1 scores obtained during a 10-fold cross-validation tuning experiment using 50% of the labeled nodes. The tuning experiments were also run on SVM- and neural network-based methods yielding comparable or worse results.

6.4.2 Datasets

We use 3 publicly-available datasets, containing both temporal information on edges and labels, from a variety of domains: IMDb, DBLP, and CiteULike. Table 6.4 contains a summary of the resulting time-windowed graphs for each dataset.

The **IMDb** data originate from a snapshot of the online portal called the Internet Movie Database[4] taken on the 3rd of February 2017. The dataset consists of a list of actors, which movies they have particpated in, what year the movies were released, and for most movies a user rating from 0 to 10, where 10 is the best rating possible. In the time-windowed graph representation, the actors are labeled based on the ratings of the movies they have participated in. The weighting is performed as in Section 6.3.4.1 where for any node, if there exists a positive label, then it is assigned a weight of 1, and 0 otherwise.

DBLP is an online database of metadata for published papers in computer science [34]. For each article in the database there is information on the title, publishing year, authors, publication venue, and references. In the time-windowed graph representation the authors are labeled based on which venues they have published articles to. The weighting is performed as in Section 6.3.4.3 where for any node, if there exists a positive label, then it assigned a weight of 1, and 0 otherwise.

CiteULike is a network of users, publications, and tagging events where a user tags a publication [274]. In the time-windowed graph representation, the users are labeled based on whether they have used a tag from a predefined set of tags. The weighting is performed as in Section 6.3.4.2 where for any node, if there exists a positive label, then it assigned a weight of 1, and 0 otherwise.

Further details on the provenance of the datasets and their processing into time-windowed graphs can be found in Appendix 6.7.

As can be seen in Table 6.4, the fraction of future positive nodes $P(+)$ is close to 0 for all three datasets, which can be a challenge for classification algorithms

[4]ftp://ftp.fu-berlin.de/pub/misc/movies/database/

Dataset	$\lvert V_{t_p}^a \rvert$	$\lvert U_{t_p}^a \rvert$	$\lvert E_{t_p}^a \rvert$	$\lvert E_{t_p}^p \rvert$	t_p	t_f	$\Delta\tau$	Q_{t_p}	$P(+)$
IMDb	20k	66k	180k	460k	2000	2005	1 year	0.11	0.02
DBLP	4.2k	2.7k	15k	68k	2006	2011	1 year	0.39	0.02
CiteULike	0.9k	11k	12k	1.7k	3/2005	7/2005	15 days	0.03	0.06

Table 6.4: Dataset TWG Characteristics: $|V_{t_p}^a|$ and $|U_{t_p}^a|$ are number of top and bottom nodes in the accumulated graph at time t_p, $|E_{t_p}^a|$ is the number edges of the accumulated graph, $|E_{t_p}^p|$ is the number of edges of the temporal projection of the accumulated graph, t_p is the time of the present snapshot, t_f is the time of the future snapshot, $\Delta\tau$ is the size of the time-windows, Q_{t_p} is the modularity of the accumulated graph $G_{t_p}^a$ (temporal projection if bipartite), and $P(+)$ is the fraction of nodes in $V_{t_p}^a$ which are future positive nodes.

due to label-imbalance. Although there exist strategies to compensate for these imbalances, using them can also have unwanted effects on the classification performance [288]. All three datasets form bipartite graphs, and the projection described in definition 44 results in a significant increase in number of edges (e.g. from $70k$ to $550k$ edges). For simplicity and the sake of computability, we restrict any temporal graph from having more than one edge between any two nodes. It is done in such a way that only the *latest* time-instant is used for an edge between any pair of nodes.

6.4.3 Experimental Methodology

We solve the problem of node classification described in Section 6.2.4 for all three datasets. The time instants representing "present" and "future", t_p and t_f, are selected from the recent past in such a way to avoid lack of data, but also not the most recent time-stamps available due to computational restrictions. The rating threshold for the IMDb dataset's labeling, $r_{\text{threshold}}$, is set as the mode of the distribution of ratings. For each of the three datasets we choose fixed parameter values for t_p, t_f, $r_{\text{threshold}}$ and $\Delta\tau$ independently (details can be found in Table 6.4), and set the proportion of labeled nodes to be used during training to 50%.

6.4.3.1 Classification

The decision tree classifier is trained on the node metrics and labels of a subset of the nodes in the accumulated graph at time t_p. All features are standardized by removing the mean and scaling the values to unit variance. For each dataset and feature set we run 50 trials, and in each trial perform a class-stratified sampling containing $100 \times (\text{proportion labeled})\%$ nodes as a training set and the remaining as a test set. The sampled nodes are chosen to that each node appears in the same amount of test sets over all trials.

6.4.3.2 Performance and scalability

The node metrics were calculated on a standard laptop and took approx. 50 hours total for all three datasets. As seen in Table 6.5, the time used for computation approximately scales on the number of nodes and edges in the graph, except for DBLP. The long running time for DLBP is due to non-optimized implementations of cover time and constrained coverage. This scaling is most likely explained by the fact that the majority of the chosen metrics are path-based, and therefore scale with number of possible paths in the graph. For strategies on scaling we refer to discussions in earlier work [36, 37]

6.5 Results

Results for the decision tree classifier trained on sets of node metrics are shown in Table 6.5, and Figure 6.6. Table 6.5 consists of precision/recall values and F1-scores for each feature set, where the leftmost two featuresets are the baselines (**S** and **T**), and an approximate running time needed to calculate node metrics for each dataset t_c. We choose a weighted F1-Score for Table 6.5 and Figure 6.6 to account for label-imbalance.

For all three datasets the baseline unweighted temporal node features **T** exhibit a small increase in performance over the static unweighted features **S**. However, while temporal unweighted features result in higher F1-scores, the overall recall of the positive class is reduced slightly (< 0.01) for all three datasets. This is most likely due to how temporal metrics are restricted to using temporal paths, and thus for e.g. betweenness centrality, fewer nodes are visited (or not visited at all), which in turn can result in nodes having zero-valued features.

For all of the datasets, there is at least one feature set of weighted metrics that out-classes the pure unweighted static and temporal node metrics. For IMDb, all three weighted feature sets **I1**, **I2**, and **I4** result in higher F1 scores than the baselines **S** and **T**. However, **I4** provides an improvement of approximately 15% increase of F1, while **I1** and **I2** only improve F1 by $\leq 2\%$. This indicates that the use of weighted static metrics can improve classification performance over using pure static metrics, especially when using path-based node functions. This is possibly due to the fact that node function type IV can encode information about more of the topology around a node. For example, in closeness centrality, node function type I and II only encode weights from the nodes and the beginning and end of the shortest paths used for calculation. Type IV, using weights from all nodes traversed, is able to more accurately describe the neighborhood of the node in question.

For CiteULike we see an F1-score increase of approximately 15% when using the weighted node metrics of **C**. The difference of using the recency node function (type III) on weighted metrics, versus temporal node metrics **T**, implies that recent changes in graph topology are more important than just using temporal paths in node metrics. This is consistent with what node function type III metrics capture: For CiteULike, these enable easy identification of currently active authors that have not yet had many publications (i.e., having many temporal features near zero).

Lastly we see that combining node function I and III, on DBLP, results in even larger improvement of F1-score. Using feature set **C** increases the F1-score approximately 20% over static and temporal node metrics. This could indicate that for some datasets recency node metrics can further improve performance when node weights are taken into account, as it matters *which* articles an aspiring author has recently referenced. If the articles they recently cited already are published in the target venues (DBDM), it improves the probability of themselves publishing in these in the near future.

The significant increase in F1-score using weighted node metrics (**I4**, **D**, and **C**) come at a cost of lower positive class recall, and the choice of node function represents a trade-off of positive class recall and positive class precision. Table 6.5 shows that for all datasets the positive class recall is higher for all non-weighted configurations, while precision is higher for their weighted counterparts. The

Dataset	0/1	S				T				Weighted Metrics												t_c
		P	R	F1		P	R	F1		P	R	F1		P	R	F1		P	R	F1		
										I1				**I2**				**I4**				
IMDB	0	.97	.66	.78	**.75**	.96	.68	.80	**.76**	.96	.71	.81	**.78**	.96	.74	.84	**.79**	.95	.87	.91	**.87**	6h each
	1	.12	.70	.21		.12	.62	.20		.13	.61	.21		.14	.58	.22		.18	.41	.25		
										D												
DBLP	0	.95	.63	.75	**.71**	.94	.63	.74	**.70**	.95	.79	.85	**.81**									43 hours
	1	.07	.44	.12		.07	.41	.11		.08	.27	.12										
										C												
CITEULIKE	0	.91	.75	.79	**.73**	.91	.79	.82	**.76**	.92	.95	.94	**.87**									30 min
	1	.07	.19	.07		.08	.18	.09		.11	.08	.09										

Table 6.5: Results on datasets for feature set. Precision (P), recall (R), and F1-score is listed for both classes as well as a weighted F1-score (in bold). Green cells indicate best weighted F1-score for dataset. Column t_c contains running times for each dataset.

lower recall might be a result of a difference in number of paths: there are fewer paths in weighted metrics (used in computing the centrality score) than in purely static or temporal metrics, and therefore fewer future positive nodes are visited.

Finally, for all datasets presented, our proposed weighted metrics unequivocally yielded the best performance overall.

6.6 Conclusion and Future Work

We have proposed and investigated two sets of new theoretically principled node metrics for weighted temporal graphs and evaluated them as features for performing within-network classification: static and temporal weighted node metrics. The proposed metrics were compared to existing unweighted node metrics through experiments on 3 real-world datasets. The new node metrics were tailored to each dataset using four node functions, which each captured some phenomena in the data. Results showed that using weighted node metrics with tailored node functions, or combinations of these, improves classification performance, and that the weighted node metrics can be adapted to a range of domains.

For further work, there are three general directions: (I) to employ the weighted node metrics, including possible tuning of node functions, to attack problems in temporal weighted graphs beyond the ones we have done here, for example clustering and link prediction; (II) to extend ours, and related work on metrics, to for temporal *multi*graphs where multiple edges can exist between pairs of nodes; and

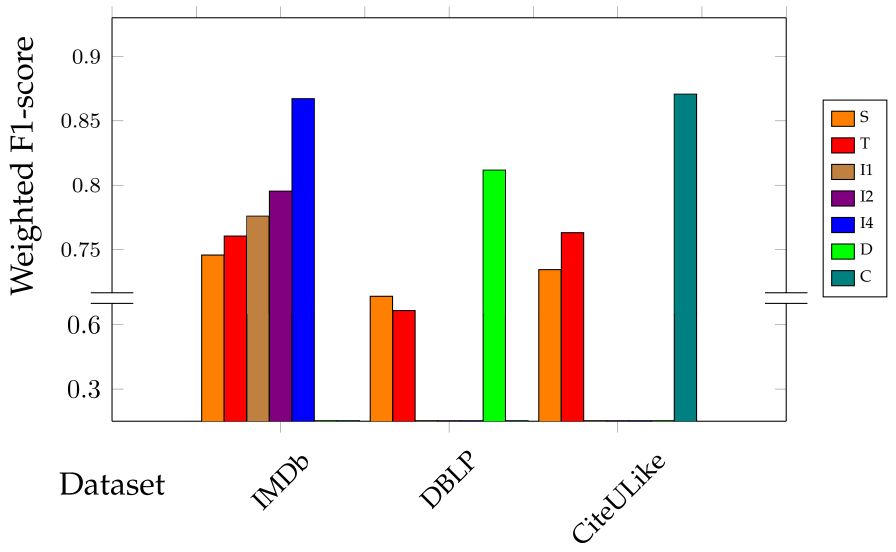


Figure 6.6: Weighted F1-scores for classification experiments on datasets. Results for feature sets are grouped and presented as bars for each dataset.

(III) to develop appropriate metrics using edge weights (as opposed to solely using node weights) weights.

Christopher Ryther & Jakob Simonsen have nothing to disclose.

6.7 Dataset Details

In this section we first describe for each dataset, how the corresponding stream of triplets S and the associated labelstream $\mathcal{L}$ are created. Then we describe in detail how the IMDb and DBLP were sampled and processed. This supplementary material has also appeared in earlier yet unpublished work by us [37].

6.7.1 Dataset Streams

6.7.1.1 IMDB (Bipartite)

is a database of actors, movies, and ratings (between 0 and 10). An IMDb triplet $(a, m, t) \in S$ corresponds to an actor a participating in a movie m at time t. An IMDb triplet $(a, l, t) \in \mathcal{L}$ means actor a has participated in a movie at time t where the label l is a binarization of that movie's rating r. Using the entire dataset was computationally infeasible, hence a sampled version of the IMDb dataset is used (see Section 6.7.2).

6.7.1.2 DBLP (Bipartite)

is a database of metadata for papers [34], each consisting of a title, publishing year, authors, publication venue and references. We only use papers published in *database and data mining* (DBDM) or *computer vision and pattern recognition* (CVPR) venues. The set of DBDM venues is: PODS, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA, and the set of CVPR venues is: CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM. A DBLP triplet $(a, p, t) \in S$ corresponds to an author a publishing a paper at time t which cites paper p. A DBLP triplet $(a, 1, t) \in \mathcal{L}$ means an author a has published a paper at time t in any DBDM venue. A triplet $(a, 0, t)$ is given for every paper published at a CVPR venue.

6.7.1.3 Citeulike (Bipartite)

is a network of users and publications [274]. There exists two types of timestamped events: a user u tags a document d at time t, and a user u uses a tag a at time t. Each triplet (u, d, t) in S corresponds to a user u tagging a document d at time t. The labelstream $\mathcal{L}$ is based on whether a user has used one or more of pre-selected set of popular tags. A triplet $(u, 1, t)$ means user u used one of the pre-selected tags at time t. A triplet $(u, 0, t)$ is given for use of any other tag.

6.7.2 IMDb

The IMDb dataset is a snapshot of the Internet Movie Database[5] downloaded on the 3rd of February 2017. IMDb is an online database of movies, actors and related information about them, as well as a platform for users to rate movies. The dataset consists of a list of actors, which movies they appeared in, which year the movie was released, and for most movies: a user rating, which is a decimal value between 0 and 10.

Let $\mathcal{A}$ be the set of triples $(a, m, t) \in \mathbb{D}^2 \times \mathbb{T}$ where a is an actor that appeared in movie m released in cinemas the year t, and let $\mathcal{D}$ be the set of tuples $(m, r) \in$

[5]ftp://ftp.fu-berlin.de/pub/misc/movies/database/

$\mathbb{D} \times [0, 10]$ where m is a movie that has a rating of r from users of the IMDb website. The IMDb time-windowed graph consists of a sequence of bipartite graphs, and is induced by the stream S defined by:

$$S = \{(a, m, t) : (a, m, t) \in \mathcal{A} \text{ and } \exists r \in \mathbb{D}[(m, r) \in \mathcal{D}]\} \tag{57}$$

The IMDb labelstream is based on the ratings of movies in the following way: a triplet $(a, l, t) \in \mathcal{L}$ means an actor a has participated in a movie m at time t where the label l is a binarization of the movie's rating r. Binarization using a fixed threshold is too strict (i.e. two movies with ratings 6.999 and 7.001 would be given different labels, even though they are very close in rating), therefore the threshold must be softened. To soften the binarization process, we add Gaussian noise. Given a value $r_{\text{threshold}} \in [0, 10]$, and a random variable $X_r \sim \mathcal{N}(r_{\text{threshold}}, \frac{1}{2})$, we define the labelstream $\mathcal{L}$ as:

$$\mathcal{L} = \{(a, l, t) : (\exists (m, r) \in \mathcal{D})[(a, m, t) \in \mathcal{A}\} \tag{58}$$

where $l \in \{0, 1\}$ given by:

$$\Pr(l = 1 \mid r) = \Pr(X_r \leq r) \tag{59}$$

$$\Pr(l = 0 \mid r) = 1 - \Pr(l = 1 \mid r) \tag{60}$$

Thus, if the rating of a movie r is higher than $r_{\text{threshold}}$, the actors who participated are given a positive label (with some probability) as illustrated in Figure 6.7. The means some movies with $r < r_{\text{threshold}}$ will result in positive labels and vice versa - however, the farther the rating is from $r_{\text{threshold}}$, the lower the probability is of this happening.

The original IMDb dataset is very large and for resource management purposes it was necessary to downsample. Furthermore, due to some low-quality entries in the ratings of movies, we set labels to zero of movies which were not rated by more than 200 members of IMDb to avoid movies which were rated very high/very low by only a handful of people. The strategy used to modify labeling and to downsample $\mathcal{A}$ and $\mathcal{D}$ is described below:

1. Define a set M_c which contains all movies originating from a selected subset

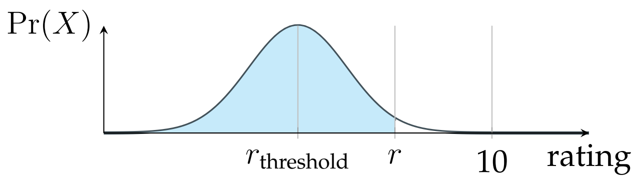


Figure 6.7: Distribution plot to illustrate added noise to IMDb labelling.

of countries, in our case Denmark & Sweden.

2. Define a set A_c which contains the top 20 members of the cast, ordered by cast billing, for each movie in M_c.
3. For each member in A_c, add all movies they have appeared in, to M_c.
4. The downsampled $\mathcal{A}_d$ is then defined as:

$$A_d = \{(a, m, t) \in \mathcal{A} : a \in A_c \text{ and } m \in M_c\}$$

5. The downsampled $\mathcal{D}_d$ is then defined as:

$$\mathcal{D}_d = \{(m, r) \in \mathcal{D} : m \in M_c\}$$

6. Let n^r_m be the number of ratings for movie m. Equation 59 then becomes:

$$\Pr(l = 1 \mid r) = \begin{cases} \Pr(X \leq r) \text{ if } 200 \leq n^r_m \\ 0 \text{ otherwise} \end{cases}$$

6.7.3 DBLP

DBLP is an online database[6] of published computer science articles [34] which we downloaded a snapshot of from 2016-07-14. Each article in DBLP is described by a title, publishing year, authors, publication venue and references. Similar to the work in [125] we are interested in predicting topic field labels in the binary case. The positive class is given to papers which have been published in *database and data mining* (DBDM) venues, and the negative class is given to papers published in *computer vision and pattern recognition* (CVPR) venues. We define $DBDM$ as

[6]https://aminer.org/citation

the set of DBDM venues which contains the following: PODS, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA. Finally we define $CVPR$ as the set of CVPR venues, containing: CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM. Note: Any article that is not published into at least one of these categories is excluded from the dataset.

Let $\mathcal{A}$ be the set of triples $(a, p, c, t) \in \mathbb{D}^3 \times \mathbb{T}$, where p is an article published by author a in venue c at time t. Furthermore let $\mathcal{C}$ be the set of tuples $(p_i, p_j) \in \mathbb{D}^2$ which means a reference to article p_j exists in article p_i. The DBLP time-windowed graph consists of a sequence of bipartite graphs, and is induced by the stream S defined by:

$$\begin{aligned} S = \{(a, p_j, t) : (\exists(p_i, c) \in \mathbb{D}^2) & \\ & [(p_i, p_j) \in \mathcal{C} \text{ and } (a, p_i, c, t) \in \mathcal{A}]\} \end{aligned} \tag{61}$$

Likewise, given the sets $\mathcal{A}$, $DBDM$ and $CVPR$, we define the DBLP labelstream $\mathcal{L}$ as:

$$\mathcal{L} = \{(a, l, t) : (\exists(p, c) \in \mathbb{D}^2)[(a, p, c, t) \in \mathcal{A}]\} \tag{62}$$

where $l \in \{0, 1\}$ given by:

$$l = \begin{cases} 1 \text{ if } c \in DBDM \\ 0 \text{ if } c \in CVPR \end{cases} \tag{63}$$

Thus, a triplet $(a, l, t) \in \mathcal{L}$ means an author a has published an article at time t, where l indicates which type of venue the articles was published in. . Note: Any article that is not published into at least one of these categories is excluded from the dataset.

Let $\mathcal{A}$ be the set of triples $(a, p, c, t) \in \mathbb{D}^3 \times \mathbb{T}$, where p is an article published by author a in venue c at time t. Furthermore let $\mathcal{C}$ be the set of tuples $(p_i, p_j) \in \mathbb{D}^2$ which means a reference to article p_j exists in article p_i. The DBLP time-windowed graph consists of a sequence of bipartite graphs, and is induced

by the stream S defined by:

$$S = \{(a, p_j, t) : (\exists (p_i, c) \in \mathbb{D}^2) \\ [(p_i, p_j) \in \mathcal{C} \text{ and } (a, p_i, c, t) \in \mathcal{A}]\} \quad (64)$$

Likewise, given the sets $\mathcal{A}$, $DBDM$ and $CVPR$, we define the DBLP labelstream $\mathcal{L}$ as:

$$\mathcal{L} = \{(a, l, t) : (\exists (p, c) \in \mathbb{D}^2)[(a, p, c, t) \in \mathcal{A}]\} \quad (65)$$

where $l \in \{0, 1\}$ given by:

$$l = \begin{cases} 1 \text{ if } c \in DBDM \\ 0 \text{ if } c \in CVPR \end{cases} \quad (66)$$

Thus, a triplet $(a, l, t) \in \mathcal{L}$ means an author a has published an article at time t, where l indicates which type of venue the articles was published in.

7 Temporal graph embedding using Gaussian mixture models

The final chapter is on how to represent large temporal multigraphs. I was working on extending some of the metrics in the above works to multigraphs when I read about incremental Gaussian mixture models. Soon thereafter I was experimenting with representing various evolving features of graphs as Gaussian mixture models. The work in this chapter will form the basis of a paper to be submitted in early 2019.

Abstract

A *temporal graph* is a (multi-)graph that evolves over time as dictated by a graph *stream*: a sequence of node and edge insertions. A basic question is how to query information about node connectivity at prior points in time without storing all edges from the stream. For large streams of graph data (e.g., the evolution of social networks over time) queries require substantial computational resources, and efficient temporal graph representations are needed. We propose a type of embeddings, *GraphGMM*, that is tailor-made for efficient queries on a growing number of nodes and edges. GraphGMM accurately stores edge insertions to a graph using static graph representations combined with incremental Gaussian mixture models. We run experiments on large temporal datasets (80K–2.5M edges), and compare our embeddings to baseline temporal multigraph implementations. Our results show that GraphGMM embeddings are (i) *accurate*: average errors for aggregated and unaggreated number of edges are typically single-digit for all time steps, (ii) *efficient:* updates performed approximately in constant time, (iii) *compact:* embeddings are a factor 10-100 smaller than baseline multigraph implementations, and (iv) *suitable for large streams:* queries performed on embeddings are faster than on baselines for large numbers of edge insertions, and scale well.

7.1 Introduction

A temporal graph is a graph that can grow over time as more edges and nodes are added. Temporal graphs can model a wide variety of real-world scenarios, e.g. social networks, p2p traffic, or citation networks, and can be analyzed through various graph-based methods such as anomaly detection, link prediction, and node classification [206, 86, 306]. As the size of temporal graphs grows over time, storing and processing all historical data can become computationally cumbersome; this is especially true for multigraphs where two nodes may have more than one edge between them, each annotated with distinct information. It is thus a pertinent problem how to store, or represent, relational information in a way that allows for efficient usage and precise analysis while maintaining a low memory footprint. Such representations of graphs are usually called *embeddings* [171, 121] or *summaries* [73, 148]. Embeddings are typically node, edge embeddings, or so-called whole-graph embeddings [307, 106, 159, 121], while summaries fall into a more eclectic range of types, including edge/node-grouping, wavelets, histograms, and sketches [187, 188]. In this work, we focus solely on *node* and *edge* embeddings.

Node and edge embeddings for static (i.e., ordinary non-temporal) graphs have been widely studied [150, 96, 151, 152, 153, 154, 121], but the study of embeddings in temporal graphs is a fairly recent field [156, 129, 73], despite the many real-world networks exhibit changes over time. For temporal graphs, the embedding must not only be able to respond to queries concerning, e.g., the number of edges between two nodes, but also do so *for any prior point in time*, as well as answer queries about the *changes* to the graph at any prior point in time; thus, the temporal evolution of the graphs has to be represented somehow in the embedding. Informally, the problem to be solved is: Given an initially empty temporal (multi)graph $G(V, E)$ and a stream S of edge insertions applied in G, construct a representation of G_r of G that uses less space than G, and where for each node $x \in V$, or pair of nodes $x, y \in V$, and timestamp t, a query (degree, number of new edges, etc.) $\deg(x, t)$ or $\deg(x, y, t)$ performed on G_r deviates little from the same query performed on G and S. For sparse temporal graphs the above typically involves a trivial task of simply counting edges, but for multigraphs with high average multiplicity (i.e., many edges between two nodes), the task is very time consuming [73]. Other approaches include representing a temporal graph as a sequence of static snapshots of the graph at different time steps, but this has been shown to be practically unviable in many cases [157], because (a) embeddings can vary greatly between two consecutive snapshots and (b) the running time – already expensive for a single static graph – scales poorly with the number of snapshots. Other recent methods have proposed scalable sketch-based summaries which can perform queries efficiently, but under assumptions about the underlying size of the graph being known in advance, or without retaining information about graph structure [73].

Contributions: We propose Graph-GMM which uses *Incremental Gaussian Mixture Models* to create embeddings designed specifically for temporal multigraphs with high average multiplicity and over many time-steps, such that (i) the size of an embedding scales better than a corresponding temporal multigraph and can accommodate both edge- and node-insertions, (ii) queries about the number of *new* edges at any prior point in time, and aggregated number of edges, are answered with only small error, (iii) the embeddings can grow dynamically with introduction of new nodes and are agnostic towards node identifiers, (iv) and the embeddings retain information about graph structure, and are easily interpretable by humans.

We perform experiments on three datasets, two real-world, and one synthetic, and compare GraphGMM to two standard implementations of temporal graphs. Our key findings are: (1) The average error for a lookup of edge-insertions at specific time is $\approx 10\%$ and $\leq 3\%$ for aggregated number of edges in our experiments, and grows slowly $\leq 1\%$ for 10-fold increase in data. (2) The efficiency of GraphGMM lookup queries is dependent on size of stream - the more edge-insertions the better it compares to baselines. (3) GraphGMM shows robust performance by all measured metrics across the three datasets in our work, but is subject to seasonal patterns.

7.1.1 Related Work

Embeddings for static graphs have been heavily investigated [150, 155, 171, 153, 154, 121] using a variety of methods, e.g. guided random-walks [96, 151, 106], edge-sampling [307], matrix-factorization [158], and role-based embeddings [154, 129]. Static embeddings have been applied to problems such as within-network classification and link prediction [307, 96, 106]. The recent focus on deep learning for graphs [163, 164, 165] has been used to create proximity-preserving embeddings for temporal graphs[171, 152, 156], and other recent work taking time-dependent growth into account have been applied to link prediction in temporal graphs [308], and to clustering [309], and classification tasks [310, 164] in temporal graphs by embedding nodes as Gaussians. While Gaussian distributions have been used to augment embeddings for various types of graphs [311], and for time-series analysis in other fields [312, 313, 314, 315], to our knowledge, no previous work has attempted to use scalable Gaussian mixture models for node embeddings in temporal multigraphs. Finally, although the above embeddings capture temporal information in the data, they themselves are "static" in the sense that they are based on a single state of a temporal network, limiting their ability to "roll-back" or query the graph at a previous state, unlike the method put forth in the present paper [121].

Graph compression and summarization methods are similar to embeddings, but focus on presenting concise summaries of graphs [56, 148, 188, 73], and have been applied to graph streams, and hence temporal graphs [185, 161, 187, 186]. However, these methods assume the set of nodes in the underlying graph to be static (or, at least, known in advance) and consisting of well-behaved node identifiers (for hashing purposes), whereas the models in the present paper can grow and shrink arbitrarily and without penalty to existing embeddings, and can be used on heterogeneous node and edge sets. While several summarization methods are designed to perform the same queries as we consider, they typically only allow for edge-insertions between existing nodes, rather than both node- and edge-insertions [187, 186, 73].

7.1.2 Preliminaries and notation

As the literature on temporal graphs contains many variations in nomenclature [2], we clarify notation and nomenclature below. In the remainder of the paper, let $\mathbb{T}$ and $\mathbb{D}$ be totally ordered sets (representing, *time* and *data*, respectively).

Definition 47. *A **temporal (multi)graph** G is a tuple (V, E), where V is the set of nodes of G and E is the set of edges. An edge $e \in E$ is a triplet (x, y, t) where $x, y \in V$ and $t \in \mathbb{T}$. In the remainder of the paper $e = (x, y, t)$ should be understood as an edge between x and y at time t.*

Definition 48. *A **stream** S is any multiset $S \subseteq \mathbb{D}^2 \times \mathbb{T}$. A **slice** of a stream S at time i, is a subset $s_i \subseteq S$ such that $s_i = \{(x, y, i) : (x, y, i) \in S\}$. The relation $\preceq$, defined by: $(x_i, y_i, t_i) \preceq (x_j, y_j, t_j) \quad \text{if} \quad t_i \leq t_j$, is a partial order on S.*

Intuitively, a stream $[(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots]$ is a set of "events" (x, y) each happening at some time t, and a slice s_i is the set of all events that took place at time i. Typically, (x, y) represents the creation of a directed edge between nodes x and y in a graph at time t.

Definition 49. *Let S be a stream. The **streamed multigraph** G_S is a temporal multigraph (V, E) whose nodes and edges are defined as:*

$$V = \biguplus_{(x,y,t)\in S} \{x, y\} \quad \textit{and} \quad E = \biguplus_{(x,y,t)\in S} \{(x, y, t)\} \tag{67}$$

where $\biguplus$ is the multiset union operator.

7.2 Methodology

In this section we present GMM-Nodes and GMM-Edges as well as explain how these embeddings can be queried to obtain approximations of the original graph.

We assume the underlying graph data is streamed, and arrives in the form of ordered "slices" $\{s_1, s_2, \cdots\}$ (defined in 48) of a stream S, where S consists of edge insertions to a streamed multigraph over time. These slices are in turn used as input, one-by-one, to our embeddings where the time-stamps of the edge-insertions are used to calculate or update the embeddings. Intuitively, this procedure corresponds to batches of data arriving, with each batch containing all edge-insertions, or "events" in some time-window (e.g., all new friends added by a user of a social network in a single day).

Each model consists of a pair $(G(V,E), \mathcal{P})$ where $G(V,E)$ is a static graph and $\mathcal{P}$ is a set of incremental Gaussian mixture models (IGMM).

The models are as follows:

1. **GMM-Nodes**: Each node in V has an associated Gaussian mixture in $\mathcal{P}$, fitted to the timestamps of its' edges in the stream. For each pair of nodes, for which there has appeared at least one edge in the stream, place a single edge in the graph $G(V,E)$.

2. **GMM-Edges**: Each pair of nodes in V has an associated Gaussian mixture in $\mathcal{P}$, fitted to the timestamps of edges between the two in the stream. For each pair of nodes, for which there has appeared at least one edge in the stream, place a single edge in the graph.

The remainder of the section is organized as follows: In subsection 7.2.1 we describe the method used to incrementally fit a Gaussian mixture model to edge insertions in slices of a stream. In subsection 7.2.2 we describe how to use a fitted mixture model to query (or lookup) the number of edge insertions (or new edges) and/or the cumulative number of edges inserted up until a time t.

7.2.1 Incremental Gaussian Mixture Model

In this section we describe how we fit the Gaussian mixture models. Our task is to continually fit and update Gaussian mixture models with the edge-insertion times from slices s_t of a stream S. The edge-insertion times from a slice s_t is a multiset of $\mathbf{t}_t = \{t\}$ with multiplicity m_t.

The expectation maximization (EM) algorithm is commonly used to find the optimal parameters fitting a Gaussian Mixture Model (GMM) to a set of values such as $\mathbf{t}_t$. However, the number of components k of the GMM is itself an input to the EM algorithm, and for most real world problems it is very difficult to know the optimal number of components a priori. Solutions do exist, such as iterating through values for k, by performing repeated runs of EM, and then evaluating which number of components produces the best model. However these tend not to scale well and have high update costs.

Instead, we use the incremental Gaussian mixture model (IGMM) of [314, 315] with a few modifications. The incremental Gaussian mixture model for the univariate case has the form:

$$p(t) = \sum_{k=1}^{K} \pi_k \mathcal{N}(t|\mu_k, \sigma_k) \tag{68}$$

$$c(t) = \sum_{k=1}^{K} \frac{\pi_k}{2} \left[1 + \text{erf}\left(\frac{t-\mu_k}{\sigma_k\sqrt{2}}\right)\right] \tag{69}$$

$$\mathcal{N}(t|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{70}$$

where $p(t)$ and $c(t)$ are the likelihood and cumulative distribution functions of a Gaussian mixture model, and $\mathcal{N}$ is the Gaussian distribution with mean μ and deviation σ. The IGMM is continually fitted, using an online algorithm, which performs an update iteration with every value $t \in \mathbf{t}_t$. For every update the IGMM: adds a new component to the model, recalculates the model covariances, and compresses the model if needed. The compression aims to keep the number of components as small as possible [314]. An example of GMM fitting is shown in Figure 7.1.

7.2.2 Lookup

After fitting an IGMM to a slice of edge-insertions, equation 68 can be used to compute the likelihood of an edge-insertion at any previous time t. However, one difference from [314, 315] is that we need to use the IGMM to lookup the number of edge-insertions at time t, not just the likelihood. In practice, we need to scale the distribution to fit the size of the slices (since $\int_x \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma_k) = 1$). We define two separate scale-factors, one for the lookup of edge-insertions at a specific time, using equation 68, and one for the lookup of total number of edges using equation 69. Using constant scale-factors, a lookup of the number of new edges at any time t, $\mathcal{L}_p(t)$, and a lookup of the number of edges inserted up until

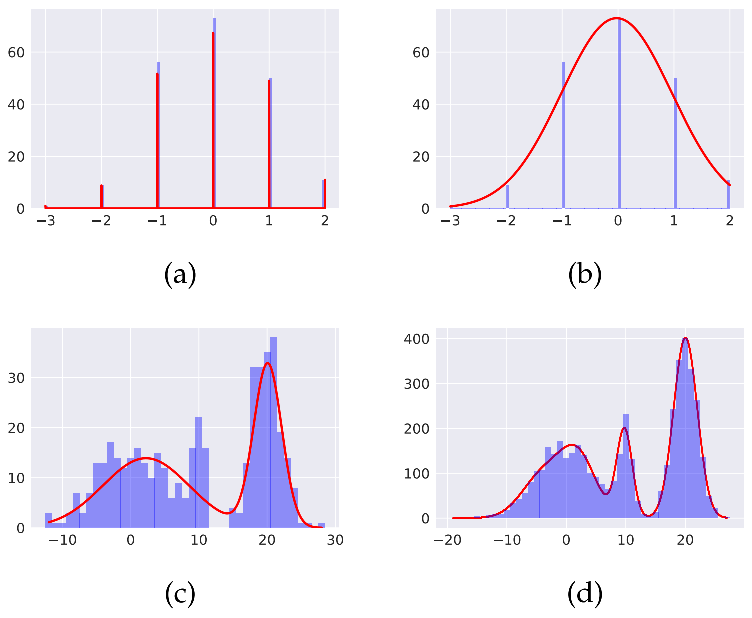


Figure 7.1: Distributions of fitted Gaussian mixture models (red) vs underlying real distribution (blue histogram). From left to right: (7.1a) Too many components in model resulting in overfitting. (7.1b) Correctly fitted GMM. (7.1c) Too few components in model resulting in underfitting. (7.1d) Correctly fitted GMM.

any time t, $\mathcal{L}_c(t)$, then becomes:

$$\mathcal{L}_p(t) = \mathcal{A}_p p(t) \quad \text{and} \quad \mathcal{L}_c(t) = \mathcal{A}_c c(t)$$

where $\mathcal{A}_p, \mathcal{A}_c \in \mathbb{R}^+$ are the *amplifications* or scaling factors of the IGMM. As the model grows, the scaling factors will need to be updated as well. For the cumulative lookup $\mathcal{L}_c(t)$ the amplification $\mathcal{A}_c$ is equal to the total number of edges used to fit the model, and thus trivial to update using a counter. For $\mathcal{L}_p$ we opt to scale based on the ratio of the distribution of the edge-insertion times, to that of the models likelihood function. To do this efficiently, we use values from a fixed window of recent slices. Given the size of the window $m \in \mathbb{N}^+$, and the time of the last slice t_p, we define the memory as $\mathcal{M} = \left[|\mathbf{t}_{t_p-m}|, \cdots, |\mathbf{t}_{t_p-2}|, |\mathbf{t}_{t_p-1}|, |\mathbf{t}_{t_p}|\right]$ which functions as an first-in-first-out queue, ordered by descending age. The window is updated every time a new slice is used to fit the IGMM.

The amplification can therefore be calculated as the mean of the set of ratios between the values of $\mathcal{M}$ and the corresponding likelihoods from the IGMM. To avoid inaccuracies from components currently undergoing significant changes (typically the most recently added components), we only use the *oldest* $q < m$ values of $\mathcal{M}$. Furthermore, to avoid inaccuracies from insufficient, or "small", sets of edge-insertions, we define a lower limit $l \in \mathbb{N}^+$ and require $l \leq |\mathbf{t}_t|$ in order for $|\mathbf{t}_t|$ to be committed to $\mathcal{M}$. Finally, we define the amplification as:

$$\mathcal{A}_p = \frac{1}{q} \sum_{t=t_p-m}^{t_p-q} \frac{|\mathbf{t}_t|}{p(t)} \tag{71}$$

This requires minimal information about the "true" distribution to be stored in the IGMM, which in turn allows for efficient computation and updates.

7.2.3 GraphGMM

In the following we describe our two proposed model variants.

7.2.3.1 GMM-Nodes

The GMM-Nodes embedding is a tuple $(G(V, E), \mathcal{P})$ where (V, E) is a, possibly directed, static graph, and $\mathcal{P} = \{p_{x_1}, p_{x_2}, \cdots, p_{x_{|V|}}\}$ is a set of incremental Gaussian mixture models. For each node in $x_i \in V$ there is an associated mixture model $p_{x_i} \in \mathcal{P}$.

Given a slice of edge insertions s_i at time i, and a (possibly empty $V = E = \mathcal{P} = \emptyset$) embedding, the update is performed as follows. For every triplet $(x, y, i) \in s_i$ with multiplicity m_{xy}, if they do not already exist, create nodes x and y in V, an edge (x, y) in E (directed if triplet is directed), and create associated IGMMs p_x and p_y in $\mathcal{P}$.

Next, update the IGMM for x using a multiset $\{i\}$ with multiplicity $m_i = m_{xy}$. If the edges in the stream are undirected, update p_y as well, using the same set. Repeat these two steps for all pairs of nodes in s_i.

The size of the GMM-Nodes+ embedding for a stream of directed edges has an exact upper bound of

$$|V| + |V|^2 + \sum_{x_i \in V} 2 + 3k_{x_i} + m - q$$

where $m - q$ is the size of the short-term memory, and k_{x_i} is the number of components of x_i's associated IGMM p_{x_i}.

7.2.3.2 GMM-Edges

The GMM-Edges embedding is a tuple $(G(V,E),\mathcal{P})$ where (V,E) is a, possibly directed, static graph, and $\mathcal{P}=\{p_{e_1},p_{e_2},\cdots,p_{e_{|E|}}\}$ is a set of incremental Gaussian mixture models. For each edge in $e_i \in E$ there exists an associated mixture model $p_{e_i} \in \mathcal{P}$.

Given a slice of edge insertions s_i at time i, and a (possibly empty $V = E = \mathcal{P} = \emptyset$) embedding, the update is performed as follows. For every triplet $(x,y,i) \in s_i$ with multiplicity m_{xy}, if they do not already exist, create nodes x and y in V, an edge (x,y) in E (directed if triplet is directed), and create an associated IGMM p_{xy} in $\mathcal{P}$ (and p_{yx} if undirected and $p_{yx} \notin \mathcal{P}$.

Next, update the IGMM p_{xy} using a multiset $\{i\}$ with multiplicity $m_i = m_{xy}$. If the edges in the stream are undirected, update p_{yx} as well, using the same set. Repeat these two steps for all pairs of nodes in s_i. The size of the GMM-Edges embedding for a stream of directed edges has an exact upper bound of

$$|V| + |V|^2 + \sum_{(x,y)\in E} 2 + 3k_{xy} + m - q$$

where $m-q$ is the size of the short-term memory, and k_{xy} is the number of components of (x,y)'s associated IGMM p_{xy}.

7.3 Experimental Design

We now describe our experimental design and setup.

7.3.1 Datasets

We use 3 disparate datasets, 2 of which are publicly available datasets containing temporal information, namely (i) an email communication dataset [316] (called *Manufacturing* or **M**), and (ii) a human contact dataset [33] (called *RealityMining* or **R**). In addition, we use a synthetically created temporal graph based on Gaussian mixture models (called *Synthetic* or **S**). Sets **M** and **R** were chosen due to their large number of temporal edges distributed over a large time-period, and **S** was created to have a higher number of edge-insertions, but also out of necessity since public multigraph datasets with sizes of **R** and **M** are rare. Table 7.1 summarizes the resulting temporal graphs.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\overline{m}$	$\tilde{m}$	T
Manufact. (**M**)	167	82,927	5,784	14	271
RealityMin. (**R**)	96	1,086,404	2,539	428	232
Synthetic (**S**)	400	2,474,268	2,378	1,040	230

Table 7.1: Dataset Graph Characteristics: $|V|$ and $|E|$ are the number of nodes and edges, $\overline{m}$ is the number of unique edges, $\tilde{m}$ is the average edge multiplicity (average number of edges between any pair of nodes), and T is the number of slices in the stream.

M contains internal email communication between employees of a manufacturing company. A triplet (x, y, t) in the stream for **M** is a *directed* edge corresponding to an email sent from person x to person y at time t. The set **R** contains proximity data collected over 9 months using 100 mobile phones. A triplet (x, y, t) in the stream for **R** is an *undirected* edge corresponding to person x having physical contact with person y at time t. To smooth the real-world datasets, we group all triplets into 24 hour blocks, or time-windows. Hence, every slice of data s_i corresponds to all edges in a 24 hour time period.

The set **S** is created in a two-step process. First, we generate a graph with power-law degree distribution and approximate average clustering [317] to populate a directed static graph. Then, for each pair of nodes (x, y), we create a random number of randomized Gaussian distributions, and sample a random amount of values from each distribution. For every value sampled $t \in \mathbb{N}^+$, we create a directed temporal edge with (x, y, t). This type of temporal graph could, for instance, model a real-world online social network of communications between people over time [317]. For the power-law graph algorithm we use parameters $n = 400$, $m = 3$ and $p = 0.1$. n is the number of nodes, m is the number of random edges added, and p is the probability of adding a triangle [317]. For the Gaussians we use the following ranges for the parameters $\mu = [0, 200]$, $\sigma = [1, 10]$, $k = [2, 10]$, and $|x| = [50, 300]$. μ and σ are the means and deviations, k is the number of distributions, and $|x|$ is the number of values to sample from each distribution.

7.3.2 Experimental Methodology

We construct embeddings for temporal multigraphs as described in Section 7.1 for all datasets.

The IGMM algorithm's parameter c_{scale}, which is a fixed value used to avoid over- and undersmoothing, is tuned through a gridsearch experiment using values in the range $[0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1.1]$. Adjusting the c_{scale} parameter allows us, in an intuitive way (i.e. larger c_{scale} results in higher accuracy and more components), to fine-tune the models[314]. The resulting values are 0.01 for the Manufacturing dataset, 0.1 for RealityMining, and 0.1 for the synthetic. The parameters for the short-term memory described in 7.2.2 are set as follows: $m = 20$, $q = 10$, $l = 5$. We employ a Matlab implementation of [315], wrapped and augmented by our own untuned Python implementation. All experiments were performed on a standard off-the-shelf 3Ghz CPU laptop with 64Gb RAM without GPU use.

7.3.3 Evaluation

To evaluate the performance of our proposed models we measure the differences between two temporal multigraph representations of each dataset, which saves all edge-insertions as edges in the graph, and the two embeddings GMM-Nodes and GMM-Edges.

7.3.3.1 Lookup Error

For each node in the temporal multigraph, we calculate the difference between its distribution of new edges, and the values provided by the lookup functions of the corresponding GMM-Nodes embedding. Similarly, for each pair of nodes in the multigraph, we calculate the difference between its distribution of new edges, and the values returned by the GMM-Edges lookup functions. We report these errors using three common metrics. The first is Symmetric Mean Absolute Percentage Error (sMAPE), which in contrast to the traditional Mean Absolute Percentage Error (MAPE) can be used in the presence of zero-values, $|\mathbf{t}_i| = 0$, in the data. The second metric is the Mean Absolute Scaled Error (MASE) for non-seasonal time-series [318]. Since the one-step forecast of MASE always will underpredict the real value of a monotonous growing function, we instead use the aforementioned MAPE, since there are no zero-values to avoid, for the $\mathcal{L}_c$ lookup function.

Given a set of sets of edge-insertion times $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_T\}$ used to fit a GMM (as in subsection 7.2.1), sMAPE and MASE for the lookup function $\mathcal{L}_p(t)$ are defined as:

$$\text{sMAPE}_p = \frac{1}{T} \sum_{\mathbf{t}_i \in \mathcal{T}} \frac{|\mathcal{L}_p(i) - |\mathbf{t}_i||}{|\mathbf{t}_i| + |\mathcal{L}_p(i)|}$$

$$\text{MASE}_p = \frac{\sum_{i=1}^{T} |\mathcal{L}_p(i) - |\mathbf{t}_i||}{\frac{T}{T-1}\sum_{i=2}^{T} ||\mathbf{t}_i| - |\mathbf{t}_{i-1}||}$$

The sMAPE and MAPE for $\mathcal{L}_c(t)$ are defined in a similar way for the accumulated sizes of $\mathbf{t}_i \in \mathcal{T}$:

$$\text{sMAPE}_c = \frac{1}{T}\sum_{i=1}^{T} \frac{|\mathcal{L}_c(i) - \sum_{j=1}^{i} |\mathbf{t}_j||}{\sum_{j=1}^{i} |\mathbf{t}_j| + |\mathcal{L}_c(i)|}$$

$$\text{MAPE}_c = \frac{1}{T}\sum_{i=1}^{T} \left| \frac{\mathcal{L}_c(i) - \sum_{j=1}^{i} |\mathbf{t}_j|}{\sum_{j=1}^{i} |\mathbf{t}_j|} \right|$$

7.3.3.2 Time and Space Complexity

To illustrate the growth of the embeddings as the number of edges in the stream increases, we perform measurements of the IGMMs while fitting them to the stream of slices. We take 50 samples for each IGMM during fitting. The samples are evenly distributed in the interval $[0, T]$ in such a way that between any two consecutive samples, the IGMM has been fitted to approximately the same number of non-empty slices. This way we can track how the embeddings evolve, while avoiding sampling multiple times when no new data has been fitted. Each sample contains the number of components in the IGMM and the total accumulated time taken to fit the IGMM so far. Due to how the sampling process is performed, any embedding with less than 50 slices of data is omitted from the experiment.

7.3.3.3 Lookup Timing

To measure the speed of the embedding's lookup functions $\mathcal{L}_p$ and $\mathcal{L}_c$ we conduct a separate experiment for each dataset defined as follows: For each embedding, run a lookup at all times in the range $t \in [0, T]$ and return the average time taken to perform each lookup. This is done for both lookup functions.

7.3.3.4 Baseline Comparison

As noted in Section 7.1.1, recent methods solving similar problems to ours have limitations in their application domain that make direct comparison difficult: some do not handle multi-graphs [56], others assume the number of nodes in the underlying graph to be known *a priori* [187, 186], or do not retain the graph structure after embedding [185]. Hence, we compare our proposed method to two baseline temporal multigraph representations (i.e., non-embedded and non-summarized) of the datasets. The first baseline is "node-centric" and uses an unordered associative array to store information for each node; the second baseline is "edge-centric" and uses a simple list for each *pair* of nodes. The node-centric baseline is best suited for lookups of a node's edge-insertions, where the edge-centric baseline is best for lookups of edge-insertions between a pair of nodes - analogous to our two proposed models GMM-Nodes and GMM-Edges.

In the node-centric baseline, each key-value pair $(t, \mathbf{y})$, in any node x's dictionary, consists of a time t and a list of nodes $\mathbf{y} = [y_1, y_2, \cdots]$, corresponding to triplets in the dataset stream $[(x, y_1, t), (x, y_2, t), \cdots]$. The lookups $\mathcal{L}_p(t)$ and $\mathcal{L}_c(t)$ correspond to simply counting nodes in the lists for the appropriate key(s) t. In the edge-centric baseline, each list $\mathbf{t}$), for any pair (x, y), consists of times $[t_1, t_2, \cdots]$ corresponding to triplets in the dataset stream $[(x, y, t_1), (x, y, t_2), \cdots]$. The lookups $\mathcal{L}_p(t)$ and $\mathcal{L}_c(t)$ correspond to counting the occurrences of t.

7.4 Results

Results for the experiments run on GMM-Nodes and GMM-Edges for all datasets are shown in Figures 7.2 and 7.3. Figure 7.2 shows the relative errors of the lookup functions and Figure 7.3 shows how the embeddings scale.

Figure 7.2 and Tables 7.2 and 7.3 show that the average sMAPE of $\mathcal{L}_p$ is below 15% across all datasets, and the $\mathcal{L}_c$ sMAPE is below 2%. sMAPE$_c$ for both lookup functions increases $\leq 1\%$ for a 10-fold increase in edge-insertions. As explained in Section 7.2.1, there is a tradeoff between the number of components used for each IGMM and the size of the relative lookup errors. This effect can be seen in Figures 7.2e and 7.2g and Tables 7.2 and 7.3 where c_{scale} is the same value for both the **R** and **S** sets, but because the number of edges used to fit IGMMs are higher for the **R** set, the sMAPE error is larger. Therefore the largest multiplicities of a temporal multigraph are a better indicator of the expected sMAPE of the GraphGMM embedding than the average multiplicity (Table 7.1) where the average multiplicity of **S** is more than double than that of **R**.

The MASE in Figures 7.2b and 7.2d is for most IGMMs around 1, with a few outliers. This suggests that most fitted IGMMs have similar error to the naïve

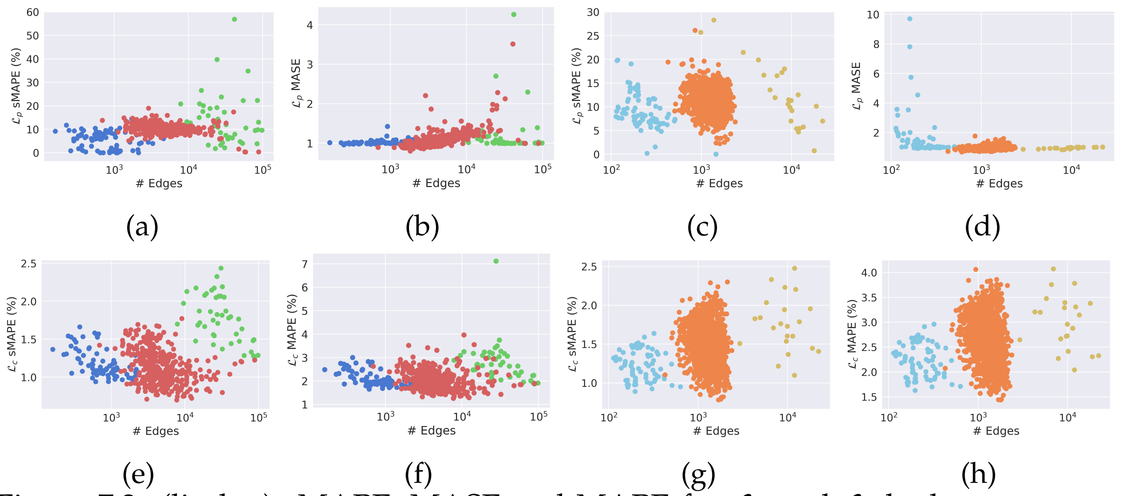
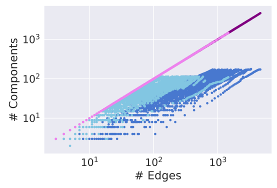


Figure 7.2: (lin-log) sMAPE, MASE and MAPE for $\mathcal{L}_p$ and $\mathcal{L}_c$ lookup errors vs. edge-insertions used to fit the IGMM. The two leftmost columns show errors for GMM-Nodes embeddings, and the two rightmost columns show errors for GMM-Edges.

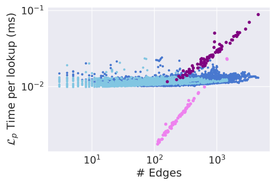
🟦 Manufacturing GMM-Nodes
🟩 RealityMining GMM-Nodes
🟥 Synthetic GMM-Nodes
🟦 Manufacturing GMM-Edges
🟨 RealityMining GMM-Edges
🟧 Synthetic GMM-Edges
🟪 Baseline Graph Node-centric
🟪 Baseline Graph Edge-centric

forecast method which uses the value of the previous time-step as prediction. The remaining outliers can be an effect of noisy data or seasonal effects which are not compensated for. As an example, the **M** dataset is based on a real-life white-collar company, and therefore few emails are sent/received in the weekends. When removing the outliers in Figure 7.2d, the embeddings for the **M** set have a stable upper bound on the relative error of $\mathcal{L}_c$ of 0.5 as seen in Figures 7.2f and 7.2h and tables 7.2 and 7.3. The MAPE$_c$ is below 3% on average for all embeddings on all datasets, and scales well with $\leq 1\%$ increase in MAPE$_c$ for a 10-fold increase in edge-insertions, as seen in Figures 7.2f and 7.2h.

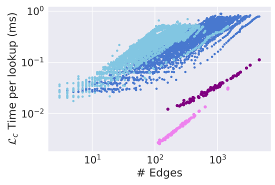
Figures 7.3a, 7.3e and 7.3i show that the number of IGMM components in all embeddings grow at a similar rate and that the size of the baseline temporal multigraph representations grows linearly as expected. The experiments show that the GraphGMM models scale better than both the baselines in number of components. Furthermore both GMM-Nodes and GMM-Edges appear robust, and show almost identical growth, in relation to number of edge-insertions, across all metrics measured in Figure 7.3. However, the models have comparatively more components, and follow the baselines more closely, for the **M** set. This is possibly due to seasonal patterns in **M**, where the number of emails, and thus



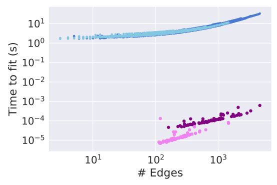
(a)



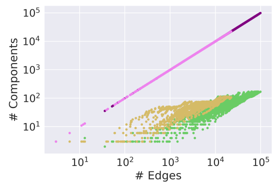
(b)



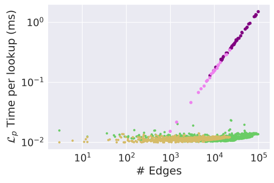
(c)



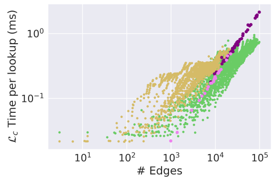
(d)



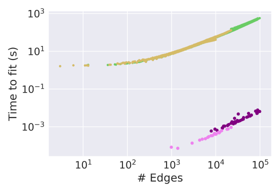
(e)



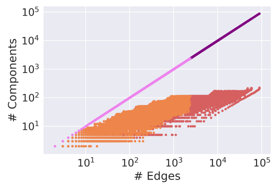
(f)



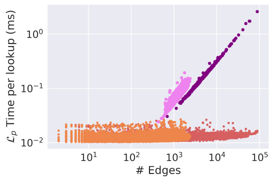
(g)



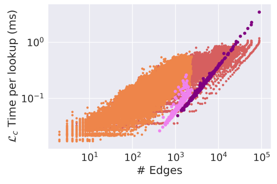
(h)



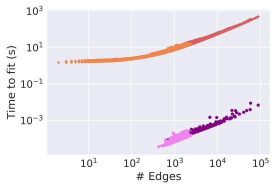
(i)



(j)



(k)



(l)

Figure 7.3: (log-log) Scalability results for GraphGMM. Each row corresponds to a distinct dataset, and the following is plotted, in relation to the number of edge-insertions, in the columns from left to right: (1) Number of components in each embedding (2) Average time per lookup $\mathcal{L}_p$ (3) Average time per lookup $\mathcal{L}_c$ (4) Total time to fit IGMM to data.

Dataset	sMAPE$_p$	MASE$_p$	sMAPE$_c$	MAPE$_c$
M	5.68%	1.03	1.19%	2.11%
R	13.99%	1.21	1.77%	2.80%
S	10.62%	1.10	1.12%	2.04%

Table 7.2: Average GMM-Nodes errors over all samples.

Dataset	sMAPE$_p$	MASE$_p$	sMAPE$_c$	MAPE$_c$
M	9.14%	1.60	1.23%	2.23%
R	12.51 %	0.94	1.76%	3.00%
S	11.32%	0.95	1.48%	2.66%

Table 7.3: Average GMM-Edges errors over all samples.

edge-insertions, drops to zero on weekends, and the IGMM algorithm is thus unable to generalize a single component to more than 5 days without large inaccuracies.

The two centre columns of Figure 7.3 show that the time complexity of lookup functions $\mathcal{L}_p$ and $\mathcal{L}_c$ are dependent on the number of edge-insertions, with $\mathcal{L}_p$ scaling better than $\mathcal{L}_c$. This is most likely due to how the cumulative distribution function is implemented and executed as these are not optimized (as mentioned in Section 7.3). Compared to the baselines, the GraphGMM models perform better than the node-centric baseline for $\mathcal{L}_p$ type lookups and when the number of edge-insertions is non-trivial (1000<). For small numbers of edge-insertions (<1000, see Figure 7.3c), the naïve baselines may still outperform GraphGMM, as expected. The same observation holds $\mathcal{L}_c$ lookups where the embeddings require more time to compute $\mathcal{L}_c$ than the corresponding baselines, but with enough edge-insertions – for **R** in Figure 7.3g and **S** in Figure 7.3k (10^5 edge-insertions) – the embeddings outperform the baseline. The final column in Figure 7.3 shows the accumulated time taken to fit each IGMM. Both GMM-edges, GMM-nodes, and both baselines have constant time updates, but the baselines have much better constants, again as expected. While the baseline implementations are faster, both GraphGMM models fulfill our update-efficiency requirements, but are targets for further optimization.

In summary, our proposed methods present a trade-off for modeling *large* real-world temporal multigraphs: a higher computational complexity per update, but significantly lower memory footprint, faster edge-insertion queries ability to add/remove embeddings without penalty.

7.5 Conclusion and Future Work

We have proposed GraphGMM, a type of embedding consisting of a static graph and incremental Gaussian mixture models for approximate representation of temporal multigraphs. GraphGMM exploits that large amounts of edge-insertions over time can be approximated by probability distributions in a streaming fashion while providing efficient retrieval, or queries, of the previous states of the original temporal multigraph. GraphGMM does not require a priori information about the number of nodes in the graph, and adapts dynamically to both edge-insertions and node-insertions. Experiments demonstrate the advantage of our model over two traditional graph representations. Future work includes (a) using GraphGMM to solve other types of problems, including link-prediction and classification, (b) investigate the performance of GraphGMM on other types of graph-structural queries, and (c) perform further optimization of (implementations of) GraphGMM. Furthermore, extending or augmenting GraphGMM to better adapt to certain temporal patterns (e.g., sudden drops in the number of edge-insertions) should be explored.

References

[1] K. Wehmuth, A. Ziviani, and E. Fleury, "A unifying model for representing time-varying graphs," in *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015*, 2015, pp. 1–10.

[2] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," *IJPEDS*, vol. 27, no. 5, pp. 387–408, oct 2012.

[3] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, and F. Amblard, "Time-Varying Graphs and Social Network Analysis: Temporal Indicators and Metrics," *SNAMAS*, pp. 32–38, 2011.

[4] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *J. Complex Networks*, vol. 2, no. 3, pp. 203–271, 2014.

[5] M. Latapy, T. Viard, and C. Magnien, "Stream graphs and link streams for the modeling of interactions over time," *Social Netw. Analys. Mining*, vol. 8, no. 1, pp. 61:1–61:29, 2018.

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

[7] C. Xin, B. Xie, and C.-C. Shen, "A novel layered graph model for topology formation and routing in dynamic spectrum access networks," *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, pp. 308–317, 2005.

[8] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, *Graph Metrics for Temporal Networks*, 2013.

[9] F. Harary and G. Gupta, "Dynamic graph models," *Mathematical and Computer Modelling*, vol. 25, no. 7, pp. 79 – 87, 1997.

[10] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora, “Small-world behavior in time-varying graphs,” *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 81, p. 055101, 05 2010.

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

[12] P. Holme, “Modern temporal network theory: a colloquium,” *The European Physical Journal B*, 2015.

[13] M. Mesbahi, “On a dynamic extension of the theory of graphs,” in *American Control Conference, 2002. Proceedings of the 2002*, vol. 2. IEEE, 2002, pp. 1234–1239.

[14] C. C. Bilgin and B. Yener, “Dynamic network evolution: Models, clustering, anomaly detection,” *IEEE Networks*, 2006.

[15] C. Tan, J. Tang, J. Sun, Q. Lin, and F. Wang, “Social action tracking via noise tolerant time-varying factor graphs,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 1049–1058.

[16] H. Kim and R. Anderson, “Temporal node centrality in complex networks,” *Physical Review E*, 2012.

[17] S. A. Hill and D. Braha, “Dynamic model of time-dependent complex networks,” *Phys. Rev. E*, vol. 82, p. 046105, Oct 2010.

[18] R. A. Rossi and J. Neville, “Representations and ensemble methods for dynamic relational classification,” *CoRR*, vol. abs/1111.5312, 2011.

[19] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville, “Transforming Graph Data for Statistical Relational Learning,” *Journal of Artificial Intelligence Research*, vol. 45, pp. 363–441, 2012.

[20] K. Wehmuth, E. Fleury, and A. Ziviani, “On multiaspect graphs,” *Theor. Comput. Sci.*, vol. 651, pp. 50–61, 2016.

[21] O. Michail, “An introduction to temporal graphs: An algorithmic perspective,” *Internet Mathematics*, vol. 12, no. 4, pp. 239–280, 2016.

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

[23] B.-M. Bui-Xuan, A. Ferreira, and A. Jarry, “Evolving graphs and least cost journeys in dynamic networks,” in *WiOpt’03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Mar. 2003, p. 10 pages.

[24] P. Holme, “Network reachability of real-world contact sequences,” *Physical Review E*, 2005.

[25] J. K. Tang, M. Musolesi, C. Mascolo, and V. Latora, “Temporal distance metrics for social network analysis,” in *Proceedings of the 2nd ACM Workshop on Online Social Networks, WOSN 2009, Barcelona, Spain, August 17, 2009*, 2009, pp. 31–36.

[26] J. Tang, M. Musolesi, C. Mascolo, V. Latora, and V. Nicosia, “Analysing information flows and key mediators through temporal centrality metrics,” in *Proceedings of the 3rd Workshop on Social Network Systems - SNS ’10*, 2010, pp. 1–6.

[27] R. Shields, “Cultural topology: The seven bridges of königsburg, 1736,” *Theory, Culture & Society*, vol. 29, no. 4-5, pp. 43–57, 2012.

[28] C. Aggarwal, *An introduction to social network data analytics*, 2011.

[29] J.-P. Onnela, J. Saramaki, J. Hyvonen, G. Szabo, M. A. de Menezes, K. Kaski, A.-L. Barabasi, and J. Kertesz, “Analysis of a large-scale weighted network of one-to-one human communication,” pp. 1–25, 2007.

[30] R. Albert and A.-L. Barabasi, “Statistical mechanics of complex networks,” 2001.

[31] E. Bullmore and O. Sporns, “Complex brain networks: graph theoretical analysis of structural and functional systems,” *Nature Reviews Neuroscience*, vol. 10, no. 3, p. 186, 2009.

[32] V. Guihaire and J.-K. Hao, “Transit network design and scheduling: A global review,” *Transportation Research Part A: Policy and Practice*, vol. 42, no. 10, pp. 1251–1273, 2008.

[33] N. Eagle and A. Pentland, "Reality mining: Sensing complex social systems," *Personal and ubiquitous computing*, vol. 10, no. 4, pp. 255–268, 2006.

[34] J. Tang, J. Zhang, L. Yao, and J. Li, "Extraction and mining of an academic social network," in *Proceeding of the 17th international conference on World Wide Web - WWW '08*, 2008, p. 1193.

[35] C. Ryther, J. G. Simonsen, and A. Koch, "Within-network classification with label-independent features and latent linkages," in *Proceedings of the 12th International Workshop on Mining and Learning with Graphs (MLG)*, 2016.

[36] C. Ryther and J. G. Simonsen, "Within-network classification in temporal graphs," in *Proceedings of ehe Eighth IEEE ICDM Workshop on Data Mining in Networks (DaMNet)*, 2018.

[37] ——, "Recency and Label-Sensitivity for Classification in Temporal Graphs," *Awaiting submission*, 2019.

[38] ——, "Weighted and Time-Sensitive Metrics for Node Classification in Temporal Graphs," *Submitted to: Network Science*, no. November, 2018.

[39] ——, "GraphGMM : History-Preserving Gaussian Embeddings for Temporal Graphs," *Awaiting submission*, 2019.

[40] P. Holme, "Epidemiologically optimal static networks from temporal network data," *PLoS Computational Biology*, vol. 9, no. 7, 2013.

[41] A. Barrat, B. Fernandez, K. K. Lin, and L. Young, "Modeling temporal networks using random itineraries," *CoRR*, vol. abs/1303.4411, 2013.

[42] E. Ser-Giacomi, R. Vasile, E. Hernández-García, and C. López, "Most probable paths in temporal weighted networks: An application to ocean transport," *Phys. Rev. E*, vol. 92, p. 012818, Jul 2015.

[43] R. Lambiotte, L. Tabourier, and J. Delvenne, "Burstiness and spreading on temporal networks," *CoRR*, vol. abs/1305.0543, 2013.

[44] E. Cheng, J. W. Grossman, and M. J. Lipman, "Time-stamped graphs and their associated influence digraphs," *Discrete Applied Mathematics*, vol. 128, no. 2-3, pp. 317–335, 2003.

[45] J. Moody, “The importance of relationship timing for diffusion*,” *Social Forces*, vol. 81, no. 1, pp. 25–56, 2002.

[46] V. Batagelj and S. Praprotnik, “An algebraic approach to temporal network analysis based on temporal quantities,” *Social Netw. Analys. Mining*, vol. 6, no. 1, pp. 28:1–28:22, 2016.

[47] J. Whitbeck, M. D. de Amorim, V. Conan, and J. Guillaume, “Temporal reachability graphs,” in *The 18th Annual International Conference on Mobile Computing and Networking, Mobicom'12, Istanbul, Turkey, August 22-26, 2012*, 2012, pp. 377–388.

[48] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[49] E. Acar, D. M. Dunlavy, and T. G. Kolda, “Link prediction on evolving data using matrix and tensor factorizations,” in *ICDM Workshops 2009, IEEE International Conference on Data Mining Workshops, Miami, Florida, USA, 6 December 2009*, 2009, pp. 262–269.

[50] L. Gauvin, A. Panisson, A. Barrat, and C. Cattuto, “Revealing latent factors of temporal networks for mesoscale intervention in epidemic spread,” *CoRR*, vol. abs/1501.02758, 2015.

[51] L. Gauvin, A. Panisson, and C. Cattuto, “Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach,” *CoRR*, vol. abs/1308.0723, 2013.

[52] D. M. Dunlavy, T. G. Kolda, and E. Acar, “Temporal link prediction using matrix and tensor factorizations,” *TKDD*, vol. 5, no. 2, pp. 10:1–10:27, 2011.

[53] B. Bach, E. Pietriga, and J.-D. Fekete, “Visualizing dynamic networks with matrix cubes,” in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '14. ACM, 2014, pp. 877–886.

[54] R. Hamon, P. Borgnat, P. Flandrin, and C. Robardet, “Duality between temporal networks and signals: Extraction of the temporal network structures,” *CoRR*, vol. abs/1505.03044, 2015.

[55] E. Valdano, L. Ferreri, C. Poletto, and V. Colizza, “Analytical computation of the epidemic threshold on temporal networks,” *Phys. Rev. X*, vol. 5, p. 021005, Apr 2015.

[56] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, “TimeCrunch: Interpretable Dynamic Graph Summarization,” *Proceedings of SIGKDD*, pp. 1055–1064, 2015.

[57] R. Pfitzner, I. Scholtes, A. Garas, C. J. Tessone, and F. Schweitzer, “Betweenness preference: Quantifying correlations in the topological dynamics of temporal networks,” *CoRR*, vol. abs/1208.0588, 2012.

[58] L. Speidel, T. Takaguchi, and N. Masuda, “Community detection in directed acyclic graphs,” *CoRR*, vol. abs/1503.05641, 2015.

[59] T. Takaguchi, Y. Yano, and Y. Yoshida, “Coverage centralities for temporal networks,” *European Physical Journal B*, vol. 89, no. 2, pp. 1–11, 2016.

[60] N. Perra, B. Gonçalves, R. Pastor-Satorras, and A. Vespignani, “Activity driven modeling of time varying networks,” *Scientific reports*, vol. 2, 03 2012.

[61] V. Nicosia, J. K. Tang, M. Musolesi, G. Russo, C. Mascolo, and V. Latora, “Components in time-varying graphs,” *CoRR*, vol. abs/1106.2134, 2011.

[62] G. Krings, M. Karsai, S. Bernhardsson, V. D. Blondel, and J. Saramäki, “Effects of time window size and placement on the structure of an aggregated communication network,” *EPJ Data Sci.*, vol. 1, no. 1, p. 4, 2012.

[63] I. Psorakis, S. Roberts, I. Rezek, and B. Sheldon, “Inferring social network structure in ecological systems from spatio-temporal data streams,” *Journal of the Royal Society, Interface / the Royal Society*, vol. 9, pp. 3055–66, 06 2012.

[64] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[65] V. Sekara, A. Stopczynski, and S. Lehmann, “The fundamental structures of dynamic social networks,” *CoRR*, vol. abs/1506.04704, 2015.

[66] A. Walker, D. Eyre, D. Wyllie, K. Dingle, R. Harding, L. O'Connor, D. Griffiths, A. Vaughan, J. Finney, M. Wilcox, D. Crook, and T. Peto, "Characterisation of clostridium difficile hospital ward-based transmission using extensive epidemiological data and molecular typing." *PLoS Med*, vol. 9, no. 2, p. e1001172, 2012.

[67] F. Liljeros, J. Giesecke, and P. Holme, "The contact network of inpatients in a regional healthcare system. a longitudinal case study," *Mathematical Population Studies*, vol. 14, no. 4, pp. 269–284, 2007.

[68] M. Génois, C. L. Vestergaard, J. Fournet, A. Panisson, I. Bonmarin, and A. Barrat, "Data on face-to-face contacts in an office building suggests a low-cost vaccination strategy based on community linkers," *CoRR*, vol. abs/1409.7017, 2014.

[69] K. Wehmuth, E. Fleury, and A. Ziviani, "Multiaspect graphs: Algebraic representation and algorithms," *Algorithms*, vol. 10, no. 1, p. 1, 2017.

[70] T. G. Kolda, *Multilinear operators for higher-order decompositions*. United States Department of Energy, 2006, vol. 2.

[71] S. Boccaletti, G. Bianconi, R. Criado, C. I. D. Genio, J. Gómez-Gardeñes, M. Romance, I. Sendiña-Nadal, Z. Wang, and M. Zanin, "The structure and dynamics of multilayer networks," *CoRR*, vol. abs/1407.0742, 2014.

[72] K. Lee, B. Min, and K. Goh, "Towards real-world complexity: an introduction to multiplex networks," *CoRR*, vol. abs/1502.03909, 2015.

[73] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Comput. Surv.*, vol. 51, pp. 62:1–62:34, 2018.

[74] A. Bavelas, "A mathematical model for group structures," *Human Organization*, vol. 7, no. 3, pp. 16–30, 1948.

[75] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215 – 239, 1978.

[76] S. P. Borgatti and M. G. Everett, "A graph-theoretic perspective on centrality," *Social Networks*, vol. 28, no. 4, pp. 466–484, 2006.

[77] J. M. Hernández and P. Van Mieghem, "Classification of graph metrics," *Delft University of Technology Technical Report*, 2011.

[78] S. Bonner, J. Brennan, G. Theodoropoulos, I. Kureshi, and A. S. McGough, "Deep Topology Classification: A New Approach for Massive Graph Classification Stephen," *IEEE ICBD*, 2017.

[79] G. Li, M. Semerci, B. Yener, and M. J. Zaki, "Effective graph classification based on topological and label attributes," *Statistical Analysis and Data Mining*, vol. 5, no. 4, pp. 265–283, 2012.

[80] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs," in *14th International Workshop on Mining and Learning with Graphs (MLG '17)*, 2017.

[81] I. Santos, C. Laorden, and P. G. Bringas, "Collective classification for unknown malware detection," in *SECRYPT 2011 - Proceedings of the International Conference on Security and Cryptography, Seville, Spain, 18 - 21 July, 2011, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, 2011, pp. 251–256.

[82] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 02, p. 56, 2014.

[83] M. Franzke, J. Bleicher, and A. Züfle, "Finding influencers in temporal social networks using intervention analysis," in *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016, Sydney, NSW, Australia, September 28-29, 2016, Proceedings*, 2016, pp. 3–16.

[84] K. Xue and J. Wang, "Identifying influential spreaders by temporal efficiency centrality in temporal network," in *Cloud Computing and Security - 4th International Conference, ICCCS 2018, Haikou, China, June 8-10, 2018, Revised Selected Papers, Part V*, 2018, pp. 369–383.

[85] L. Getoor and A. Machanavajjhala, "Entity resolution: theory, practice & open challenges," *Proceedings of the VLDB Endowment*, 2012.

[86] C. Aggarwal and K. Subbian, "Evolutionary Network Analysis," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–36, 2014.

[87] L. Tang and H. Liu, "Relational learning via latent social dimensions," *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining*, vol. 14, no. 16, pp. 817–825, 2009.

[88] ——, "Scalable learning of collective behavior based on sparse social dimensions," *Proceeding of the 18th ACM conference on Information and knowledge management CIKM 09*, p. 1107, 2009.

[89] ——, "Leveraging social media networks for classification," *Data Mining and Knowledge Discovery*, vol. 23, no. 3, pp. 447–478, 2011.

[90] B. Gallagher and T. Eliassi-Rad, "An examination of experimental methodology for classifiers of relational data," in *Workshops Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, 2007, pp. 411–416.

[91] B. Gallagher, H. Tong, T. Eliassi-Rad, and C. Faloutsos, "Using ghost edges for classification in sparsely labeled networks," *Proceedings of the 14th ACM SIGKDD*, 2008.

[92] B. Gallagher and T. Eliassi-Rad, "Leveraging label-independent features for classification in sparsely labeled networks: An empirical study," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5498 LNAI, pp. 1–19, 2009.

[93] X.-l. X. Li, C. C. S. Foo, K. K. L. K. Tew, and S. S.-k. Ng, "Searching for Rising Stars in Bibliography Networks," *International Conference on Database*, pp. 288–292, 2009.

[94] F. Lin and W. W. Cohen, "Semi-supervised classification of network data using very few labels," *Proceedings - 2010 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2010*, pp. 192–199, 2010.

[95] L. Getoor and C. P. Diehl, "Link mining: a survey," *SIGKDD Explorations*, vol. 7, no. 2, pp. 3–12, 2005.

[96] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," *CoRR*, vol. abs/1403.6652, 2014.

[97] S. Macskassy and F. Provost, "A simple relational classifier," 2003.

[98] S. a. Macskassy and F. Provost, “Classication in Networked Data: A toolkit and a univariate case study,” *Journal of Machine Learning Research*, vol. 8, no. December 2004, pp. 1–41, 2005.

[99] J. Neville and D. Jensen, “Iterative classification in relational data,” *AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.

[100] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, “Collective Classification in Network Data,” *Association for the Advancement of Artificial Intelligence*, 2008.

[101] G. Seni and J. F. E. IV, *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*, ser. Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers, 2010.

[102] C. Desrosiers and G. Karypis, “Within-network classification using local structure similarity,” *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part I*, pp. 260–275, 2009.

[103] S. Bhagat, G. Cormode, and S. Muthukrishnan, “Node Classification in Social Networks,” *Social Network Data Analytics*, pp. 115–148, 2011.

[104] X. Zhu, Z. Ghahramani, and J. Lafferty, “Semi-supervised learning using gaussian fields and harmonic functions,” *ICML*, 2003.

[105] X. Zhu, “Semi-Supervised Learning Literature Survey,” *Sciences-New York*, pp. 1–59, 2007.

[106] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *KDD*, 2016, pp. 855–864.

[107] T. Kajdanowicz, P. Kazienko, and P. Doskocz, “Label-dependent feature extraction in social networks for node classification,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6430 LNCS, pp. 89–102, 2010.

[108] T. Kajdanowicz, P. Kazienko, P. Doskocz, and K. Litwin, “An assessment of node classification accuracy in social networks using label-dependent feature extraction,” in *Knowledge Management, Information Systems, E-Learning,*

and Sustainability Research - Third World Summit on the Knowledge Society, WSKS 2010, Corfu, Greece, September 22-24, 2010. Proceedings, Part I, 2010, pp. 125–130.

[109] T. Kajdanowicz, P. Kazienko, and P. Doskocz, "A method of label-dependent feature extraction in social networks," in *Computational Collective Intelligence. Technologies and Applications - Second International Conference, ICCCI 2010, Kaohsiung, Taiwan, November 10-12, 2010, Proceedings, Part II*, 2010, pp. 11–21.

[110] T. Kajdanowicz, R. Michalski, K. Musial, and P. Kazienko, "Active learning and inference method for within network classification," in *Advances in Social Networks Analysis and Mining 2013, ASONAM '13, Niagara, ON, Canada - August 25 - 29, 2013*, 2013, pp. 1299–1306.

[111] T. Kajdanowicz, P. Kazienko, and M. Janczak, "Collective classification techniques: An experimental study," in *New Trends in Databases and Information Systems, Workshop Proceedings of the 16th East European Conference, ADBIS 2012, Poznań, Poland, September 17-21, 2012*, 2012, pp. 99–108.

[112] M. Atzmueller, H. Soldano, G. Santini, and D. Bouthinon, "MinerLSD: Efficient Local Pattern Mining on Attributed Graphs," in *Proc. 2018 IEEE International Conference on Data Mining Workshops (ICDM)*, 2018.

[113] H. Ahmad, A. L. I. Daud, L. Wang, H. Hong, H. Dawood, and Y. Yang, "Prediction of Rising Stars in the Game of Cricket," vol. 5, 2017.

[114] J. Neville and D. Jensen, "Relational Dependency Networks," *Journal of Machine Learning Research*, vol. 8, pp. 653–692, 2007.

[115] B. Taskar, P. Abbeel, and D. Koller, "Discriminative probabilistic models for relational data," in *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, 2002, pp. 485–492.

[116] Y. Yao and L. B. Holder, "Scalable svm-based classification in dynamic graphs," in *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, 2014, pp. 650–659.

[117] K. Li, S. Guo, N. Du, J. Gao, and A. Zhang, "Learning, analyzing and predicting object roles on dynamic networks," in *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, 2013, pp. 428–437.

[118] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *CoRR*, vol. abs/1704.06199, 2017.

[119] M. Woźniak, M. Graña, and E. Corchado, "A survey of multiple classifier systems as hybrid systems," *Information Fusion*, 2014.

[120] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *Unpublished*, 2018.

[121] H. Y. Cai, V. W. Zheng, and K. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications," *IEEE TKDE*, 2018.

[122] L. Lü, D. Chen, X. Ren, Q. Zhang, Y. Zhang, and T. Zhou, "Vital nodes identification in complex networks," *CoRR*, vol. abs/1607.01134, 2016.

[123] P. Desikan and J. Srivastava, "Mining Temporally Evolving Graphs," in *WEBKDD Workshop*, vol. 22, no. March, 2004, pp. 13–22.

[124] Z. Yang, J. Tang, and Y. Zhang, "Active Learning for Streaming Networked Data," *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management - CIKM '14*, pp. 1129–1138, 2014.

[125] Y. Yao and L. Holder, "Classification in dynamic streaming networks," *Advances in Social Networks Analysis and Mining*, 2016.

[126] T. Guo, X. Zhu, J. Pei, and C. Zhang, "SNOC: streaming network node classification," in *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, 2014, pp. 150–159.

[127] C. C. Aggarwal and N. Li, "On supervised mining of dynamic content-based networks," *Statistical Analysis and Data Mining*, vol. 5, no. 1, pp. 16–34, apr 2012.

[128] Y. Yao and L. B. Holder, "Scalable classification for large dynamic networks," in *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, 2015, pp. 609–618.

[129] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proceedings of CIKM*, 2017, pp. 387–396.

[130] Meng, Yanhong, Liu, Xianxian, Zhao, Peirong, and Yi, Yunhui, "A study on the node centrality based multisocial attributes weighted in mobile social networks," *ITM Web Conf.*, vol. 17, p. 01001, 2018.

[131] J. Zhang, B. Liu, J. Tang, T. Chen, and J. Li, "Social influence locality for modeling retweeting behaviors," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 2761–2767, 2013.

[132] Y. Q. Zhang, X. Li, J. Xu, and A. Vasilakos, "Human interactive patterns in temporal networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 2, pp. 214–222, 2015.

[133] S. Nandanwar and M. N. Murty, "Structural neighborhood based classification of nodes in a network," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1085–1094.

[134] T. Kajdanowicz, K. Tagowski, M. Falkiewicz, and P. Kazienko, "Incremental learning in dynamic networks for node classification," in *Network Intelligence Meets User Centered Social Media Networks [4th European Network Intelligence Conference, ENIC 2017, Duisburg, Germany, September 11-12, 2017]*, 2017, pp. 133–142.

[135] Y. Pei, J. Zhang, G. Fletcher, and M. Pechenizkiy, "Node classification in dynamic social networks," in *Proceedings of AALTD 2016: 2nd ECMLPKDD International Workshop on Advanced Analytics and Learning on Temporal Data*, 2016, pp. 1–8.

[136] I. Günes, Z. Çataltepe, and S. G. Ögüdücü, "Ga-tvrc-het: genetic algorithm enhanced time varying relational classifier for evolving heterogeneous networks," *Data Min. Knowl. Discov.*, vol. 28, no. 3, pp. 670–701, 2014.

[137] A. Tagarelli and R. Interdonato, "Time-aware analysis and ranking of lurkers in social networks," *Social Network Analysis and Mining*, 2015.

[138] J. Stojanovic, M. Jovanovic, D. Gligorijevic, and Z. Obradovic, "Semi-supervised learning for structured regression on partially observed attributed graphs," *CoRR*, vol. abs/1803.10705, 2018.

[139] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 3837–3845.

[140] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[141] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *CoRR*, vol. abs/1506.05163, 2015.

[142] P. Yanardag and S. Vishwanathan, "Deep Graph Kernels," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, pp. 1365–1374, 2015.

[143] Y. Ma, Z. Guo, Z. Ren, Y. E. Zhao, J. Tang, and D. Yin, "Dynamic graph neural networks," *CoRR*, vol. abs/1810.10627, 2018.

[144] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 5694–5703.

[145] F. Monti, M. M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 3700–3710.

[146] J. You, B. Liu, Z. Ying, V. S. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, 2018, pp. 6412–6422.

[147] N. D. Cao and T. Kipf, “Molgan: An implicit generative model for small molecular graphs,” *CoRR*, vol. abs/1805.11973, 2018.

[148] A. Khan, S. S. Bhowmick, and F. Bonchi, “Summarizing static and dynamic big graphs,” *Proceedings of the VLDB Endowment*, vol. 10, pp. 1981–1984, 2017.

[149] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, “Continuous-time dynamic network embeddings,” in *Companion of the The Web Conference 2018 on The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*, 2018, pp. 969–976.

[150] M. Belkin and P. Niyogi, “Laplacian Eigenmaps for Dimensionality Reduction and Data Representation,” *Neural Computation*, vol. 15, pp. 1373–1396, 2003.

[151] S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph representations with global structural information,” in *Proceedings of CIKM*, 2015, pp. 891–900.

[152] D. Wang, P. Cui, and W. Zhu, “Structural Deep Network Embedding,” *Proceedings of SIGKDD*, pp. 1225–1234, 2016.

[153] W. L. Hamilton, R. Ying, and J. Leskovec, “Representation learning on graphs: Methods and applications,” *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[154] N. K. Ahmed, R. Rossi, J. B. Lee, X. Kong, T. L. Willke, R. Zhou, and H. Eldardiry, “Learning Role-based Graph Embeddings,” in *Proceedings of the Statistical Relational AI Workshop (at IJCAI)*, 2018.

[155] Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Revisiting semi-supervised learning with graph embeddings,” in *Proceedings of ICML*, 2016, pp. 40–48.

[156] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, “Representation learning over dynamic graphs,” *CoRR*, vol. abs/1803.04051, 2018.

[157] P. Goyal, N. Kamra, X. He, and Y. Liu, “DynGEM: Deep Embedding Method for Dynamic Graphs,” in *IJCAI*, 2018.

[158] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, “Community Preserving Network Embedding,” *AAAI*, pp. 203–209, 2017.

[159] A. Narayanan, M. Chandramohan, R. Venkatesan *et al.*, "graph2vec: Learning Distributed Representations of Graphs," *CoRR*, vol. 1707.05005, p. 53, 2017.

[160] J. Liang, P. Jacobs, and S. Parthasarathy, "SEANO: Semi-supervised Embedding in Attributed Networks with Outliers," 2017.

[161] A. McGregor, "Graph stream algorithms," *ACM SIGMOD Record*, vol. 43, no. 1, pp. 9–20, 2014.

[162] S. Cavallari, V. W. Zheng, H. Cai, K. C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, 2017, pp. 377–386.

[163] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *CoRR*, vol. abs/1611.07308, 2016.

[164] A. Bojchevski and S. Günnemann, "Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking," in *ICLR*, 2018.

[165] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-Party Deep Network Representation," in *Proceedings of IJCAI*, 2016, pp. 1895–1901.

[166] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[167] ——, "Distributed Representations of Words and Phrases and their Compositionality," *Nips*, pp. 1–9, 2013.

[168] T. Mikolov, W. Yih, and G. Zweig, "Linguistic Regularities in Continuous Space Word Representations." *HLT-NAACL*, 2013.

[169] Z. Li and J. Liu, "A multi-agent genetic algorithm for community detection in complex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 449, pp. 336 – 347, 2016.

[170] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017, 2017, pp. 135–144.

[171] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, 2018.

[172] G. Buehrer and K. Chellapilla, "A scalable pattern mining approach to web graph compression with communities," in *Proceedings of the International Conference on Web Search and Web Data Mining, WSDM 2008, Palo Alto, California, USA, February 11-12, 2008*, 2008, pp. 95–106.

[173] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, "Graph OLAP: towards online analytical processing on graphs," in *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, 2008, pp. 103–112.

[174] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, 2008, pp. 567–580.

[175] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, 2008, pp. 419–432.

[176] B. Seah, S. S. Bhowmick, C. F. D. Jr., and H. Yu, "FUSE: a profit maximization approach for functional summarization of biological networks," *BMC Bioinformatics*, vol. 13, no. S-3, p. S10, 2012.

[177] J. Zhang, S. S. Bhowmick, H. H. Nguyen, B. Choi, and F. Zhu, "Davinci: Data-driven visual interface construction for subgraph search in graph databases," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015, pp. 1500–1503.

[178] P. Boldi and S. Vigna, "The webgraph framework I: compression techniques," in *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, 2004, pp. 595–602.

[179] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, “On compressing social networks,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, 2009, pp. 219–228.

[180] N. R. Brisaboa, S. Ladra, and G. Navarro, “k2-trees for compact web graph representation,” in *String Processing and Information Retrieval, 16th International Symposium, SPIRE 2009, Saariselkä, Finland, August 25-27, 2009, Proceedings*, 2009, pp. 18–30.

[181] U. Kang and C. Faloutsos, “Beyond ’caveman communities’: Hubs and spokes for graph compression and mining,” in *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, 2011, pp. 300–309.

[182] I. Tsalouchidou, G. D. F. Morales, F. Bonchi, and R. A. Baeza-Yates, “Scalable dynamic graph summarization,” in *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, 2016, pp. 1032–1039.

[183] B.-S. Seah, S. S. Bhowmick, and C. F. Dewey, “Diffnet: Automatic differential functional summarization of de-map networks,” *Methods*, vol. 69, no. 3, pp. 247 – 256, 2014.

[184] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[185] P. Zhao, C. C. Aggarwal, and M. Wang, “gsketch: On query estimation in graph streams,” *PVLDB*, vol. 5, no. 3, pp. 193–204, 2011.

[186] N. Tang, Q. Chen, and P. Mitra, “Graph stream summarization: From big bang to big crunch,” in *Proceedings of the SIGMOD*, 2016, pp. 1481–1496.

[187] A. Khan and C. Aggarwal, “Toward query-friendly compression of rapid graph streams,” *Proceedings of ASONAM*, vol. 7, no. 1, pp. 130–137, 2016.

[188] M. Besta and T. Hoefler, “Survey and taxonomy of lossless graph compression and space-efficient graph representations,” *CoRR*, vol. abs/1806.01799, 2018.

[189] B. S. Khan and M. A. Niazi, "Network community detection: A review and visual survey," *CoRR*, vol. abs/1708.00977, 2017.

[190] S. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[191] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 35:1–35:37, 2018.

[192] T. Hartmann, A. Kappes, and D. Wagner, "Clustering evolving networks," in *Algorithm Engineering - Selected Results and Surveys*, 2016, pp. 280–329.

[193] T. Aynaud, E. Fleury, J.-L. Guillaume, and Q. Wang, *Communities in Evolving Networks: Definitions, Detection, and Analysis Techniques*, 04 2013, vol. 2, pp. 159–200.

[194] N. Masuda and R. Lambiotte, *A Guide to Temporal Networks*. World Scientific (Europe), 2016.

[195] A. Sallaberry, C. Muelder, and K. Ma, "Clustering, visualizing, and navigating for large dynamic graphs," in *Graph Drawing - 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, 2012, pp. 487–498.

[196] M. K. Goldberg, M. Magdon-Ismail, S. Nambirajan, and J. Thompson, "Tracking and predicting evolution of social communities," in *PASSAT/-SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*, 2011, pp. 780–783.

[197] M. Rosvall and C. T. Bergstrom, "Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems," *CoRR*, vol. abs/1010.0431, 2010.

[198] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, 2006, pp. 554–560.

[199] T. Falkowski, A. Barth, and M. Spiliopoulou, “DENGRAPH: A density-based community detection algorithm,” in *2007 IEEE / WIC / ACM International Conference on Web Intelligence, WI 2007, 2-5 November 2007, Silicon Valley, CA, USA, Main Conference Proceedings*, 2007, pp. 112–115.

[200] C. Matias, T. Rebafka, and F. Villers, “Estimation and clustering in a semiparametric poisson process stochastic block model for longitudinal networks,” 12 2015.

[201] T. Chakraborty, A. Dalmia, A. Mukherjee, and N. Ganguly, “Metrics for community analysis: A survey,” *ACM Comput. Surv.*, vol. 50, no. 4, pp. 54:1–54:37, 2017.

[202] A. Tanay, R. Sharan, and R. Shamir, “Biclustering algorithms: A survey,” in *In Handbook of Computational Molecular Biology Edited by: Aluru S. Chapman & Hall/CRC Computer and Information Science Series*, 2005.

[203] S. C. Madeira and A. L. Oliveira, “Biclustering algorithms for biological data analysis: a survey.” *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, vol. 1, no. 1, pp. 24–45, 2004.

[204] Q. Cai, L. Ma, M. Gong, and D. Tian, “A survey on network community detection based on evolutionary computation,” *IJBIC*, vol. 8, no. 2, pp. 84–98, 2016.

[205] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenková, “Clustering attributed graphs: Models, measures and methods,” *Network Science*, vol. 3, no. 3, pp. 408–444, 2015.

[206] D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” *JASIST*, vol. 58, pp. 1019–1031, 2007.

[207] J. Kunegis and A. Lommatzsch, “Learning spectral graph transformations for link prediction,” *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 561–568, 2009.

[208] M. Fire, L. Tenenboim, O. Lesser, R. Puzis, L. Rokach, and Y. Elovici, “Link prediction in social networks using computationally efficient topological

features," in *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*, 2011, pp. 73–80.

[209] H. Liu, Z. Hu, H. Haddadi, and H. Tian, "Hidden link prediction based on node centrality and weak ties," *EPL (Europhysics Letters)*, vol. 101, no. 1, p. 18004, 2013.

[210] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006.

[211] A. Popescul, R. Popescul, and L. H. Ungar, "Statistical relational learning for link prediction," in *Workshop on Learning Statistical Models from Relational Data at the International Joint Conference on Articial Intelligence*, 2003.

[212] M. Gao, L. Chen, B. Li, Y. Li, W. Liu, and Y.-c. Xu, "Projection-based link prediction in a bipartite network," *Information Sciences*, vol. 376, pp. 158–171, 2017.

[213] N. Benchettara, R. Kanawati, and C. Rouveirol, "Supervised machine learning applied to link prediction in bipartite social networks," *Proceedings ASONAM 2010*, pp. 326–330, 2010.

[214] T. Wu, S. H. Yu, W. Liao, and C. S. Chang, "Temporal bipartite projection and link prediction for online social networks," *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*, pp. 52–59, 2015.

[215] L. Lu and T. Zhou, "Link prediction in complex networks: A survey," *CoRR*, vol. abs/1010.0725, 2010.

[216] H. Yu, P. Braun, M. A. Yıldırım, I. Lemmens, K. Venkatesan, J. Sahalie, T. Hirozane-Kishikawa, F. Gebreab, N. Li, N. Simonis, T. Hao, J.-F. Rual, A. Dricot, A. Vazquez, R. R. Murray, C. Simon, L. Tardivo, S. Tam, N. Svrzikapa, C. Fan, A.-S. de Smet, A. Motyl, M. E. Hudson, J. Park, X. Xin, M. E. Cusick, T. Moore, C. Boone, M. Snyder, F. P. Roth, A.-L. Barabási,

J. Tavernier, D. E. Hill, and M. Vidal, “High-quality binary protein interaction map of the yeast interactome network,” *Science*, vol. 322, no. 5898, pp. 104–110, 2008.

[217] V. Martínez, F. Berzal, and J. C. C. Talavera, “A survey of link prediction in complex networks,” *ACM Comput. Surv.*, vol. 49, no. 4, pp. 69:1–69:33, 2017.

[218] F. Tan, Y. Xia, and B. Zhu, “Link prediction in complex networks: A mutual information perspective,” *CoRR*, vol. abs/1405.4341, 2014.

[219] E. Leicht, P. Holme, and M. Newman, “Vertex similarity in networks,” *Physical Review E*, 2006.

[220] R. Li, J. X. Yu, and J. Liu, “Link prediction: the power of maximal entropy random walk,” in *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, 2011, pp. 1147–1156.

[221] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. V. Dooren, “A measure of similarity between graph vertices: Applications to synonym extraction and web searching,” *SIAM Review*, vol. 46, no. 4, pp. 647–666, 2004.

[222] A. Clauset, C. Moore, and M. Newman, “Hierarchical structure and the prediction of missing links in networks,” *Nature*, vol. 453, pp. 98–101, 2008.

[223] R. Guimerà and M. Sales-Pardo, “Missing and spurious interactions and the reconstruction of complex networks,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, pp. 22 073–8, 12 2009.

[224] Z. Huang, “Link prediction based on graph topology : The predictive value of the generalized clustering coefficient,” in *Proceedings ofthe Workshop on Link Analysis (KDD’06)*, 2006.

[225] L. Akoglu, H. Tong, and D. Koutra, *Graph based anomaly detection and description: A survey*, 2015, vol. 29, no. 3.

[226] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection: A Survey,” *ACM computing surveys (CSUR)*, 2009.

[227] E. E. Papalexakis, A. Beutel, and P. Steenkiste, "Network anomaly detection using co-clustering," in *Encyclopedia of Social Network Analysis and Mining*, 2014, pp. 1054–1068.

[228] P. V. Bindu and P. S. Thilagam, "Mining social networks for anomalies: Methods and challenges," *J. Network and Computer Applications*, vol. 68, pp. 213–229, 2016.

[229] Q. Ding, N. Katenka, P. Barford, E. Kolaczyk, and M. Crovella, "Intrusion As (Anti)Social Communication: Characterization and Detection," *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 886–894, 2012.

[230] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, 2004.

[231] V. Blondel and A. Decuyper, "A survey of results on mobile phone datasets analysis," *EPJ Data*, 2015.

[232] C. C. Aggarwal and S. Sathe, "Theoretical foundations and algorithms for outlier ensembles," *SIGKDD Explorations*, vol. 17, no. 1, pp. 24–47, 2015.

[233] A. Zimek, R. J. G. B. Campello, and J. Sander, "Ensembles for unsupervised outlier detection: challenges and research questions a position paper," *SIGKDD Explorations*, vol. 15, no. 1, pp. 11–22, 2013.

[234] K. Henderson, B. Gallagher, T. Eliassi-rad, and C. Faloutsos, "It 's Who You Know: Graph Mining Using Recursive Structural Features," *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 663–671, 2011.

[235] L. Akoglu, M. McGlohon, and C. Faloutsos, "oddball: Spotting anomalies in weighted graphs," in *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II*, 2010, pp. 410–421.

[236] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, 2002, pp. 538–543.

[237] H. Chen and C. L. Giles, "ASCOS: an asymmetric network structure context similarity measure," in *Advances in Social Networks Analysis and Mining 2013, ASONAM '13, Niagara, ON, Canada - August 25 - 29, 2013*, 2013, pp. 442–449.

[238] H. Sun, J. Huang, J. Han, H. Deng, P. Zhao, and B. Feng, "gskeletonclu: Density-based network clustering via structure-connected tree division or agglomeration," in *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, 2010, pp. 481–490.

[239] D. Chakrabarti, "Autopart: Parameter-free graph partitioning and outlier detection," in *Knowledge Discovery in Databases: PKDD 2004, 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy, September 20-24, 2004, Proceedings*, 2004, pp. 112–124.

[240] E. Schubert, A. Zimek, and H. Kriegel, "Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection," *Data Min. Knowl. Discov.*, vol. 28, no. 1, pp. 190–237, 2014.

[241] A. Zimek, E. Schubert, and H. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining*, vol. 5, no. 5, pp. 363–387, 2012.

[242] S. Motegi and N. Masuda, "A network-based dynamical ranking system," *CoRR*, vol. abs/1203.2228, 2012.

[243] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120.

[244] B. Mitra, E. T. Nalisnick, N. Craswell, and R. Caruana, "A dual embedding space model for document ranking," *CoRR*, vol. abs/1602.01137, 2016.

[245] H. Liao, M. S. Mariani, M. Medo, Y. Zhang, and M. Zhou, "Ranking in evolving complex networks," *CoRR*, vol. abs/1704.08027, 2017.

[246] A. Solé-Ribalta, M. D. Domenico, S. Gómez, and A. Arenas, "Centrality rankings in multiplex networks," in *ACM Web Science Conference, WebSci '14, Bloomington, IN, USA, June 23-26, 2014*, 2014, pp. 149–155.

[247] H. Sayyadi and L. Getoor, "Futurerank: Ranking scientific articles by predicting their future pagerank," *Proceedings of the 2009 SIAM International*, 2009.

[248] A. Daud, M. Ahmad, M. S. Malik, and D. Che, "Using machine learning techniques for rising star prediction in co-author network," *Scientometrics*, vol. 102, no. 2, pp. 1687–1711, 2014.

[249] A. Daud, N. N. R. N. N. R. Aljohani, R. A. Abbasi, Z. Rafique, T. Amjad, H. Dawood, and K. H. Alyoubi, "Finding Rising Stars in Co-Author Networks via Weighted Mutual Influence." *WWW (Companion Volume)*, no. April, pp. 33–41, 2017.

[250] S. Bozóki, L. Csató, and J. Temesi, "An application of incomplete pairwise comparison matrices for ranking top tennis players," *European Journal of Operational Research*, vol. 248, no. 1, pp. 211–218, 2016.

[251] K. P. Chitrapura and S. R. Kashyap, "Node ranking in labeled directed graphs," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, 2004, pp. 597–606.

[252] M. Franceschet and G. Colavizza, "Timerank: A dynamic approach to rate scholars using citations," *Journal of Informetrics*, vol. 11, no. 4, pp. 1128–1141, 2017.

[253] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[254] J. Park and M. E. Newman, "A network-based ranking system for us college football," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 10, p. P10014, 2005.

[255] J. O'Madadhain, J. Hutchins, and P. Smyth, "Prediction and ranking algorithms for event-based network data," *SIGKDD Explorations*, vol. 7, no. 2, pp. 23–30, 2005.

[256] J. O'Madadhain and P. Smyth, "Eventrank: a framework for ranking time-varying networks," in *Proceedings of the 3rd international workshop on Link discovery, LinkKDD 2005, Chicago, Illinois, USA, August 21-25, 2005*, 2005, pp. 9–16.

[257] M. A. Brandão and M. M. Moro, "Social professional networks: A survey and taxonomy," *Computer Communications*, vol. 100, pp. 20–31, 2017.

[258] X. Shi, Y. Li, and P. Yu, "Collective prediction with latent graphs," *Proceedings of the 20th ACM CIKM*, 2011.

[259] L. McDowell and D. Aha, "Labels or attributes?: rethinking the neighbors for collective classification in sparsely-labeled networks," *Proceedings of the 22nd ACM CIKM*, 2013.

[260] L. K. McDowell, A. Fleming, and Z. Markel, "Evaluating and extending latent methods for link-based classification," *Advances in Intelligent Systems and Computing*, vol. 346, pp. 227–256, 2015.

[261] Q. Lu and L. Getoor, "Link-based classification," *ICML*, pp. 496–503, 2003.

[262] J. Neville and D. Jensen, "Leveraging relational autocorrelation with latent group models," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 322–329, 2005.

[263] D. Jensen, J. Neville, and B. Gallagher, "Why collective inference improves relational classification," *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04*, p. 593, 2004.

[264] A. Fleming, L. K. McDowell, and Z. Markel, "A hidden treasure? Evaluating and extending latent methods for link-based classification," *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration*, 2014.

[265] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, 2000.

[266] X. Wang and G. Sukthankar, "Multi-label relational neighbor classification using social context features," *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, p. 464, 2013.

[267] E. C. Costa, A. B. Vieira, K. Wehmuth, A. Ziviani, and A. P. C. da Silva, "Time Centrality in Dynamic Complex Networks," *Advances in Complex Systems*, vol. 18, 2015.

[268] J. Tang, I. Leontiadis, S. Scellato, V. Nicosia, C. Mascolo, M. Musolesi, and V. Latora, "Applications of temporal graph metrics to real-world networks," *Understanding Complex Systems*, pp. 135–159, 2013.

[269] A. Clauset and N. Eagle, "Persistence and periodicity in a dynamic proximity network," in *DIMACS Workshop on Computational Methods for Dynamic Interaction Networks*, nov 2007.

[270] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, "Epidemic spreading in real networks: An eigenvalue viewpoint," *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pp. 25–34, 2003.

[271] G. Weiss, K. McCarthy, and B. Zabar, "Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs?" *DMIN*, pp. 1–7, 2007.

[272] L. E. C. Rocha, F. Liljeros, and P. Holme, "Information dynamics shape the sexual networks of Internet-mediated prostitution," *Proceedings of the National Academy of Sciences*, vol. 107, no. 13, pp. 5706–5711, 2010.

[273] I. Cantador, P. Brusilovsky, and T. Kuflik, "HetRec," in *Proceedings of the 5th ACM conference on Recommender systems*, 2011.

[274] K. Emamy and R. Cameron, "Citeulike: A Researcher's Social Bookmarking Service," *Ariadne*, vol. 51, no. 51, p. 0, 2007.

[275] T. Hogg and K. Lerman, "Social dynamics of digg," *EPJ Data Science*, vol. 1, no. 1, pp. 1–26, 2012.

[276] P. Massa and K. Souren, "Trustlet, open research on trust metrics," in *CEUR Workshop Proceedings*, vol. 333, 2008, pp. 31–44.

[277] U. Brandes and J. Lerner, "Structural similarity: Spectral methods for relaxed blockmodeling," *Journal of Classification*, vol. 27, no. 3, pp. 279–306, 2010.

[278] K. Miray, M. Wachs, K. M. Carley, and L. R. Carley, "Incremental algorithm for updating betweenness centrality in dynamically growing networks," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2013, pp. 33–40.

[279] H. Habiba, C. Tantipathananandh, H. Habiba, C. Tantipathananandh, and T. Berger-Wolf, "Betweenness Centrality Measure in Dynamic Networks," *Work*, no. November, pp. 1–16, 2007.

[280] Z. Gao, Y. Shi, and S. Chen, "Measures of node centrality in mobile social networks," *International Journal of Modern Physics C*, vol. 26, no. 09, p. 1550107, 2015.

[281] S. a. Macskassy and F. Provost, "Classification in Networked Data : A Toolkit and a Univariate Case Study," *Journal of Machine Learning Research*, pp. 935–983, 2007.

[282] W. W. Zachary, "An Information Flow Model for Conflict and Fission in Small Groups," *Journal of Anthropological Research*, vol. 33, pp. 452–473, 1977.

[283] S. Chakrabarti, B. Dom, and P. Indyk, "Enhanced hypertext categorization using hyperlinks," *ACM SIGMOD Record*, 1998.

[284] C. Chang, C. Chang, W. Hsieh, and D. Lee, "Relative centrality and local community detection," *Network Science*, vol. 3, no. 4, pp. 445–479, 2015.

[285] S. Dolev, Y. Elovici, and R. Puzis, "Routing betweenness centrality," *J. ACM*, vol. 57, no. 4, pp. 25:1–25:27, 2010.

[286] M. E. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, no. 1, pp. 39–54, 2005.

[287] M. Bockholt and K. A. Zweig, "Process-driven betweenness centrality measures," in *Network Intelligence Meets User Centered Social Media Networks [4th European Network Intelligence Conference, ENIC 2017, Duisburg, Germany, September 11-12, 2017]*, 2017, pp. 17–33.

[288] D. Wei, X. Deng, X. Zhang, Y. Deng, and S. Mahadevan, "Identifying influential nodes in weighted networks based on evidence theory," *Physica*

A: Statistical Mechanics and its Applications, vol. 392, no. 10, pp. 2564 – 2575, 2013.

[289] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social Networks*, vol. 32, no. 3, pp. 245–251, 2010.

[290] M. E. J. Newman, "Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality," *Physical Review E*, vol. 64, no. 1, pp. 016 132+, 2001.

[291] X. Qi, E. Fuller, Q. Wu, Y. Wu, and C. Zhang, "Laplacian centrality: A new centrality measure for weighted networks," *Inf. Sci.*, vol. 194, pp. 240–253, 2012.

[292] O. Benyahia and C. Largeron, "Centrality for graphs with numerical attributes," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*, 2015, pp. 1348–1353.

[293] M. G. Everett and S. P. Borgatti, "Categorical attribute based centrality: E-I and G-F centrality," *Social Networks*, vol. 34, no. 4, pp. 562–569, 2012.

[294] F. Battiston, V. Nicosia, and V. Latora, "Metrics for the analysis of multiplex networks," *CoRR*, vol. abs/1308.3182, 2013.

[295] N. Z. Gong, W. Xu, L. Huang, P. Mittal, E. Stefanov, V. Sekar, and D. Song, "Evolution of social-attribute networks: measurements, modeling, and implications using google+," in *Proceedings of the 12th ACM SIGCOMM Internet Measurement Conference, IMC '12, Boston, MA, USA, November 14-16, 2012*, 2012, pp. 131–144.

[296] F. Battiston, V. Nicosia, and V. Latora, "The new challenges of multiplex networks: measures and models," *CoRR*, vol. abs/1606.09221, 2016.

[297] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu, "A survey of heterogeneous information network analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 17–37, 2017.

[298] P. Kazienko and T. Kajdanowicz, "Label-dependent node classification in the network," *Neurocomputing*, 2012.

[299] Y. Zhou, H. Cheng, and J. Yu, "Graph clustering based on structural/attribute similarities," *Proceedings of the VLDB Endowment*, 2009.

[300] J. Neville, M. Adler, and D. Jensen, "Clustering relational data using attribute and link information," in *In Proceedings of the Text Mining and Link Analysis Workshop, 18th International Joint Conference on Artificial Intelligence*, 2003, pp. 9–15.

[301] X. Wang, M. Maghami, and G. Sukthankar, "Leveraging network properties for trust evaluation in multi-agent systems," in *Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2011, Campus Scientifique de la Doua, Lyon, France, August 22-27, 2011*, 2011, pp. 288–295.

[302] D. Taylor, S. A. Myers, A. Clauset, M. A. Porter, and P. J. Mucha, "Eigenvector-based centrality measures for temporal networks," *Multiscale Modeling & Simulation*, vol. 15, no. 1, pp. 537–574, 2017.

[303] L. E. Rocha and N. Masuda, "Random walk centrality for temporal networks," *New Journal of Physics*, vol. 16, pp. 1–27, 2014.

[304] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, "Temporal motifs in time-dependent networks," *CoRR*, vol. abs/1107.5646, 2011.

[305] B. A. Miller, N. Arcolano, and N. T. Bliss, "Efficient anomaly detection in dynamic, attributed graphs: Emerging phenomena and big data," in *2013 IEEE International Conference on Intelligence and Security Informatics, Seattle, WA, USA, June 4-7, 2013*, 2013, pp. 179–184.

[306] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson, "Modeling dynamic behavior in large evolving graphs," *Proceedings of WSDM*, p. 667, 2013.

[307] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *Proceedings of WWW*, 2015, p. 1067.

[308] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks (extended abstract)," in *Proceedings of ICDE*, 2017, pp. 57–58.

[309] Y. Pei, X. Du, J. Zhang, G. Fletcher, and M. Pechenizkiy, "struc2gauss: Structure Preserving Network Embedding via Gaussian Embedding," *CoRR*, vol. abs/1805.1, 2018.

[310] L. Dos Santos, D. Ludovic, B. Piwowarski, and P. Gallinari, "Modelling Relational Time Series using Gaussian Embeddings," in *ICLR*, 2017, pp. 2–43.

[311] S. He, K. Liu, G. Ji, and J. Zhao, "Learning to Represent Knowledge Graphs with Gaussian Embedding," in *Proceedings of CIKM*, 2015, pp. 623–632.

[312] E. Eirola and A. Lendasse, "Gaussian mixture models for time series modelling, forecasting, and interpolation," in *Proceedings of IDA*, 2013, pp. 162–173.

[313] R. C. Pinto and P. M. Engel, "A fast incremental Gaussian mixture model," *PLoS ONE*, vol. 10, no. 10, p. e0139931, oct 2015.

[314] M. Kristan, D. Skocaj, and A. Leonardis, "Incremental learning with Gaussian mixture models," *Proc. of Computer Vision Winter Workshop*, no. 1, p. 25, 2008.

[315] M. Kristan, A. Leonardis, and D. Skočaj, "Multivariate online kernel density estimation with Gaussian kernels," *Pattern Recognition*, vol. 44, p. 2630, 2011.

[316] R. Michalski, S. Palus, and P. Kazienko, "Matching organizational structure and social network extracted from email communication," in *Proceedings of BIS*, 2011, pp. 197–206.

[317] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Physical Review E*, 2002.

[318] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, 2006.