CEUR-WS.org/Vol-3135/dataplat_paper2.pdf

RTGEN : A Relative Temporal Graph GENerator

Maria Massri[1], Zoltan Miklos[2], Philippe Raipin[1] and Pierre Meye[1]

[1] *Orange Labs, Cesson-Sévigné, France*
[2] *University of Rennes CNRS IRISA, Rennes, France*

Abstract
Graph management systems are emerging as an efficient solution to store and query graph-oriented data. To assess the performance and compare such systems, practitioners often design benchmarks in which they use large scale graphs. However, such graphs either do not fit the scale requirements or are not publicly available. This has been the incentive of a number of graph generators which produce synthetic graphs whose characteristics mimic those of real-world graphs (degree distribution, community structure, diameter, etc.). Applications, however, require to deal with temporal graphs whose topology is in constant change. Although generating static graphs has been extensively studied in the literature, generating temporal graphs has received much less attention. In this work, we propose RTGEN a relative temporal graph generator that allows the generation of temporal graphs by controlling the evolution of the degree distribution. In particular, we propose to generate new graphs with a desired degree distribution out of existing ones while minimizing the efforts to transform our source graph to target. Our proposed relative graph generation method relies on optimal transport methods. We extend our method to also deal with the community structure of the generated graphs that is prevalent in a number of applications. Our generation model extends the concepts proposed in the Chung-Lu model with a temporal and community-aware support. We validate our generation procedure through experiments that prove the reliability of the generated graphs with the ground-truth parameters.

Keywords
Temporal graphs, Graph generation, Optimal transport

1. Introduction

Graphs are the most natural model to describe real world interactions and are currently used in a myriad of application domains such as citation [1], transportation [?], and sensor networks [2] to cite just a few. These graphs are managed by a graph management system whose performance is usually evaluated through graph-centered benchmarks that address different performance metrics such as ingestion throughput, space usage and query execution time. In this context, practitioners refer to real-world and synthetic graphs to use in the benchmarks. Indeed, available graph generation techniques fill the gap between real and synthetically generated graphs by trying to mimic the characteristics of real graphs such as controlling the degree distribution [3, 4, 5, 6, 7, 8]. Besides, a number of existing graph generators are community-aware in the sense that they group vertices that are more densely connected between each other than they are with the rest of the graph, in separate or overlapping sub-graphs called communities [9, 10, 11].

Real graphs, however, are dynamic [12] such that their topology is subject to continuous changes. In this context, a new emphasis is being placed to support time as a first class citizen in graph management systems [13, 14, 15, 16]. Most of these systems rely on real-world temporal graphs to evaluate their proposed methods. Real-world graphs, however, do not often fit the scale requirements. Therefore, practitioners must rely on a temporal graph generator that is able to produce large scale graphs whose evolution correlates with that of real world temporal graphs. To tackle this challenge, we proposed RTGEN: a relative temporal graph generator that produces large scale temporal graphs by controlling a number of key features that characterises the evolution of real-world graphs. That is, our generation procedure, controls the evolution of the degree distribution by extending a very common generation technique [17] referred to as the Chung-Lu model with temporal and community-aware support.

We model a temporal graph by a sequence of snapshots $S = \{G_0, \dots, G_N\}$ where G_i is the graph snapshot at timestamp t_i and characterized by a degree distribution that is generated from sampling user-defined temporal parameters. Having this, our relative graph generation procedure consists of transforming G_{i-1} into G_i by applying a stream of atomic graph operations with respect to the desired degree distribution at time instants t_{i-1} and t_i. Based on the fact that a strong correlation exists between successive snapshots [18, 19, 20], we propose to minimize the number of graph operations that have to be applied in order to transform a graph snapshot into its successor. The main idea consists of minimizing the distance between degree distributions of successive

Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK
maria.massri@orange.com (M. Massri); zoltan.miklos@irisa.fr (Z. Miklos); philippe.raipin@orange.com (P. Raipin); pierre.meye@orange.com (P. Meye)
© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

snapshots. We achieve this goal by relying on an optimal transport solver which provides a transportation plan capable of transforming a "mass" from source distribution to target distribution with a minimum of work. In order to apply the obtained transportation plan, we proposed a straightforward generalization of the well-known Chung-Lu's model, also known as the CL model, that was first discussed in [21] and formalized in [17, 22]. We choose to extend this model for the reasons of simplicity and scalability. We also extended the CL model to partition the graph into ground-truth communities that coexist with the aforementioned time-dependent degree distribution. Our contributions are validated through experimental results showing the evolution of the degree distribution and community structure with respect to ground-truth input parameters.

The rest of the paper is organized as follows, Section 2 provides an overview of the generation procedure. Section 3 introduces the baseline generation procedure of the CL model. Section 4 describes the proposed community-aware extension of the CL model. Section 5 presents a detailed description of the proposed generation procedure. Section 6 provides an experimental evaluation of the synthetically generated temporal graphs. Section 7 describes the related work. Section 8 concludes the work.

2. Overview

In this section, we describe the overall generation procedure. Given the characteristics of a series of graph snapshots, our relative generation procedure produces the series of graph snapshots $\{G_1, \ldots, G_n\}$ whose characteristics approximate the given ones. These graph snapshots are relatively computed by applying a number of graph updates on each snapshot in order to produce its successor snapshot. To clarify, we apply a number of graph updates on a graph snapshot G_{i-1} to produce another graph snapshot G_i whose characteristics approximate the given parameters assigned for the ith graph snapshot.

Formally, we define a graph snapshot G_i valid at a time instant t_i as the tuple $\{V_{G_i}, E_{G_i}, \phi_{G_i}, M_{G_i}\}$ where V_{G_i} is the set of vertices, E_{G_i} is the set of edges, ϕ_{G_i} is a degree distribution and M_{G_i} is the density community matrix. For instance, we consider ϕ_{G_i} of the form $\{(x_1^{G_i}, \omega_1^{G_i}), \ldots, (x_n^{G_i}, \omega_n^{G_i})\}$ as a discrete distribution over $\mathbb{N}$ where $x_j^{G_i}$ refers to the degree of a node and $\omega_j^{G_i}$ refers to the total number of vertices in the graph whose total number of edges is equal to $x_j^{G_i}$. A density community matrix M_{G_i} defines the community structure of the generated graphs, each element m_{uv} of which is equal to the density of edges between the source community c_u and the target community c_v.

Given the number of vertices in each graph snapshot $k_i \in \{k_1, \ldots, k_n\}$, a stochastic community matrix M and a sequence of degree distributions $\{\phi_1, \ldots, \phi_n\}$, we generate a sequence of graph snapshots $\{G_1, \ldots, G_n\}$ such that each snapshot G_i is relatively generated by transforming G_{i-1}. This transformation is based on morphing the $\phi_{Gi-1} = \{(x_1^{G_{i-1}}, \omega_1^{G_{i-1}}), \ldots, (x_n^{G_{i-1}}, \omega_n^{G_{i-1}})\}$ into $\phi_i = \{(x_1^i, \omega_1^i), \ldots, (x_k^i, \omega_k^i)\}$ and preserving the community structure that is represented by the stochastic community matrix M such that $M_{G_{i-1}} = M_{G_i} = M$. Note that each element m_{uv} of M is equal to the probability of edge creation between the source and target communities c_u and c_v. Figure 1 illustrates the relative graph generation procedure. Each graph snapshot G_i is relatively generated by transforming its ancestor G_{i-1}. This transformation is based on computing a transportation matrix T that minimizes the cost of morphing $\phi_{G_{i-1}}$ into ϕ_i. The computation of the transportation matrix reduces to an optimal transport problem. Based on the computed transportation matrix, each vertex belonging to the graph G_{i-1} is assigned with a linkage or breakage probability to indicate the probability of adding or removing an edge. This phase is followed by creating or removing edges to or from the graph G_{i-1} to produce the graph G_i. These graph updates follows the linkage or breakage probabilities assigned for each of the vertices. Finally, the graph G_i is computed by applying the generated updates on G_{i-1}. Note that, the generation procedure depicted in this Figure shows a simplified scenario where the number of vertices does not change. However, if that number changes, a phase consisting of the addition or deletion of vertices should precede the computation of the transportation matrix to assure the following constraint:

$$\sum_{s=1}^{k} \omega_s^{G_i} = \sum_{t=1}^{m} \omega_t^i, \quad \forall 1 \leq i \leq n$$

This constraint implies that the sum of weights of distributions ϕ_{G_i} and ϕ_i should be equal.

3. Graph generation with given expected degree distribution

In this section, we describe the generation procedure of random static graphs with a given degree distribution.

Random graphs were introduced by Erdős and Rényi [23]. The popularity of this model, also known as the ER model, stems from its simple generation procedure that consists of generating a number of vertices and connecting them by an edge after picking each endpoint with a fixed probability p. However, this model produces graphs whose degree distribution follows a binomial distribution

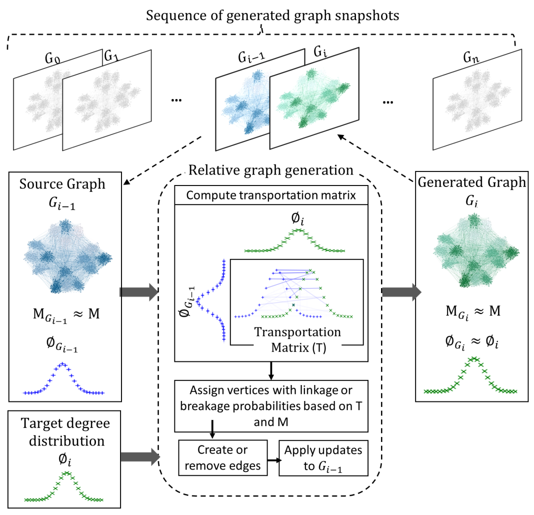
Sequence of generated graph snapshots
G_0
G_1
...
G_{i-1}
G_i
...
G_n
Source Graph
G_{i-1}
$M_{G_{i-1}} \approx M$
$\emptyset_{G_{i-1}}$
Target degree distribution
$\emptyset_i$
Relative graph generation
Compute transportation matrix
$\emptyset_i$
$\emptyset_{G_{i-1}}$
Transportation Matrix (T)
Assign vertices with linkage or breakage probabilities based on T and M
Create or remove edges
Apply updates to G_{i-1}
Generated Graph
G_i
$M_{G_i} \approx M$
$\emptyset_{G_i} \approx \emptyset_i$

Figure 1: Relative graph generation procedure.

with a mean degree equals to $(N-1)p$ where N is the total number of vertices. Hence, it fails to mimic real-world graphs that usually follow a power-law degree distribution. To tackle this limitation, the edge configuration model [24] consists of generating a random graph whose degree distribution matches, approximately, a given degree distribution. That is, each vertex is assigned with a number of stubs equal to its desired degree that is drawn independently from the given degree distribution. Having this, pairs of stubs are linked randomly forming edges between their endpoints. Although this technique approximately matches any given degree distribution, a relaxed version known as the Chung-Lu model was introduced in [21]. This model consists of generating a random graph that approximately matches a given degree distribution relying on a simple generation procedure that can be considered as a variant of the ER model. For simplicity, we will refer to this model as the CL model in the following description.

Consider the degree distribution ϕ as the input parameter to the CL model and the undirected, unweighted and unlabeled graph $G = \{V, E, \phi_G\}$ as the output where ϕ_G denotes the degree distribution of G, V and E denote the set of vertices and edges, respectively. Having this, the CL model produces a graph G such that ϕ_G is an approximation of ϕ. The main idea is to pick each endpoint of an edge with a certain probability such that, at the end of the generation procedure, the total number of incident edges to each vertex is close to its assigned degree. Hence, the starting phase consists of assigning each vertex $v_i \in V$ with a degree d_{v_i} and a linkage probability $p_{v_i} \propto d_{v_i}$. Considering that D is the sum of the degrees extracted from ϕ, we define the CL linkage probability p_{v_i} in the following Equation:

$$p_{v_i} = \frac{d_{v_i}}{D} \tag{1}$$

Subsequently, a linkage phase consists of picking $|E| = \frac{D}{2}$ pairs of vertices to connect such that for a sufficiently large $|E|$ the random variable denoting the degree of vertex v_i is Poisson distributed with a mean equals to d_{v_i}. Iterating the linkage phase $|E|$ times where an edge is equally likely to be chosen in both directions for undirected graphs, the insertion probability of an edge connecting vertex v_i and vertex v_j is $p_{v_i vj} = 2p_{v_i}p_{v_j}\frac{D}{2}$. The edge insertion probability can be rewritten in the more convenient form:

$$p_{v_i v_j} = \frac{d_{v_i} d_{v_j}}{D}$$

For optimisation sake, we gather all vertices sharing the same degree together in a pool $\gamma_d = \{v_i | v_i \in V \wedge d_{v_i} = d\}$ that we use as a subsidiary generation component. Each vertex in a pool is equally likely to be chosen assuring that the aforementioned linkage probability p_{v_i} is not affected for a sufficiently large number of vertices. After the degree assignment phase, vertices are distributed throughout the pools having each the following linkage probability:

$$p_{\gamma_d} = \frac{d|\gamma_d|}{D}$$

Now, instead of picking vertices a pool is first picked It should be highlighted that self-loops or multi-edges can be created since each endpoint of an edge is picked independently. The number of these edges, however, is independent of the number of vertices and thus can be neglected for large scale graphs.

4. Community-aware graph generation with given expected degree distribution

Although the CL model produces graphs with respect to a given degree distribution, it is not aware of the community structure existing in most real-world graphs. Hence, we propose a community-aware extension of the CL model based on the stochastic block model (SBM). Since a community is not quantitatively well defined, many definitions where provided in literature. Intuitively, one can consider a community as a subgraph which vertices are more densely connected between each other than they are with the rest of the graph. Let's consider the set of communities $C = \{c_i\}$ and suppose that a vertex should belong to one community and edges should be differentiated into within and between edges:

- Given a community c_i, an edge e is called a within edge if the source vertex $\in c_i$ and the target vertex $\in c_i$.
- Given two communities c_i and c_j, an edge e is called a between edge if the source vertex $\in c_i$ and target vertex $\in c_j$ or vice versa.

To insure that vertices belonging to a community are more densely connected to each other than they are with the rest of the graph, the within and between edge creation probabilities $p_{c_i}^{in}$ and $p_{c_i}^{out}$ of c_i must satisfy the condition $p_{c_i}^{in} > p_{c_i}^{out}, \forall c_i \in C$.

4.1. Stochastic block model

In this section, we formulate the SBM model [9] (also known as the planted partition model) which is commonly used for the generation of random graphs with a given community structure. Hence, this generation procedure only considers controlling the community structure of the graph and overlooks the resulting degree distribution. The input of the generation procedure is a stochastic community matrix M, each element m_{ij} of which defines the probability of edge creation between the source community c_i and the target community c_j. The output is a graph $G = \{V, E, M_G\}$ where M_G is the obtained density community matrix, each element m_{ij}^G of which defines the relative edge density between the source community c_i and the target community c_j. The generation procedure starts with the distribution of vertices between the planted communities such that each vertex belongs to a single community. Now, the linkage probability between a vertex belonging to community c_i and another vertex belonging to community c_j is equal to m_{ij}. However, the extracted community density matrix M_G from the resulting graph G is an approximation of M. That is, each element $m^G ij$ is binomially distributed with mean equals to m_{ij} and Poisson distributed with the same mean for a sufficiently large number of edges.

4.2. Stochastic block model with given degree distribution

In this section, we propose a static graph generation procedure which controls both the community structure and degree distribution. Given a degree distribution ϕ and a stochastic community matrix M, our proposed model generates a graph G which degree distribution ϕ_G is an approximation of ϕ and density community matrix M_g is an approximation of M. In the following, we provide a description of our generation mechanism that extends the stochastic block model depicted in Section 4.1.

Since the generated graph G is undirected, the matrix M is symmetric such that $m_{ij} = m_{ji}$. Having this, we define $\omega_{ij} = \omega_{ji} = 2m_{ij}$ and $\omega_{ii} = m_{ii}$. Furthermore, we assign each community c_i with a within edge creation probability $p_{c_i}^{in}$, a between edge creation equal to $p_{c_i}^{out}$ and a probability of edge creation p_{c_i} such that:

$$p_{c_i} = p_{c_i}^{in} + p_{c_i}^{out} = \omega_{ii} + 0.5 \sum_{j=1, j\neq i}^{|C|} \omega_{ij} \tag{2}$$

We define the linkage probability p_{v_i} of choosing a vertex v_i belonging to community c_m as follows:

$$p_{v_i} = \frac{d_{v_i}}{D_{c_m}} p_{c_m}, v_i \in c_m \tag{3}$$

where D_{c_m} is the sum of the degrees of vertices belonging to community c_m and p_{c_m} is the probability of choosing c_m. The linkage probability of a vertex is the product of the probability p_{c_m} of choosing the community to which the vertex belongs and the probability $\frac{d_{v_i}}{D_{c_m}}$ of choosing the vertex v_i in that community. Hence, Equation 3 assures the approximation of the community matrix. However, p_{v_i} should be equal to $\frac{d_{v_i}}{D}$ (Equation 1) to assure the approximation of the degree distribution. Therefore, we define the following condition in order to reduce Equation (3) to Equation (1):

$$D_{c_m} = D p_{c_m}$$

Now, replacing D by $\frac{D_{c_m}}{D p_{c_m}}$ in the original CL linkage probability (Equation 1) which assures the control of the degree distribution, we obtain Equation 3 which assures the control of the community structure. Having this, the duality of the linkage probability given in Equations (1) and (3) insures that both requirements are satisfied by our generation procedure.

For performance amelioration, we consider the selection of pools rather than vertices such that a pool is local to one community. That is vertices having the same degree variation and belonging to the same community c_m are grouped in a pool $\gamma_d^{c_m} = \{v_i | v_i \in c_m \wedge d_{vi} = d\}$ such that the probability of a pool selection for edge insertion is:

$$p_{\gamma_d^{c_m}} = \frac{d|\gamma_d^{c_m}|}{D}$$

4.3. Hierarchical community structure

The specification of the stochastic matrix is not straightforward and imposes an exhaustive number of user-defined parameters. Hence, we define an auto-generative procedure that fills the matrix with no exogenous effort. Considering a static graph, we construct a stochastic matrix that reflects a hierarchical community structure with only two given parameters. In a hierarchical community matrix, communities recursively embed subsequent communities in a self-similar fashion such that the community structure is represented by a hierarchical tree where each node represents a community. Each non-leaf node is expanded into b other nodes until reaching a desired tree height h (Figure 2). The ending recursion results in $n_c = b^h$ leaf-nodes referencing the finest scale communities having a linkage probability ω_{ij} proportional to the distance between c_i and c_j. The distance between two communities, $d(c_i, c_j)$, is equal to the number of hops traversed in order to reach the least common ancestor of these communities. In order to satisfy the condition stating that within edge linkage probability must be higher than between linkage probability ($p_{c_i}^{in} > p_{c_i}^{out}$), we define ω_{ii} as follows:

$$\omega_{ii} = 0.5 \sum_{j=0, j \neq i}^{n_c - 1} \omega_{ij} + k$$

where k is a tunable parameter which calibration steers the difference between within and between edge densities. The effect of varying k is further highlighted in the Section 6.

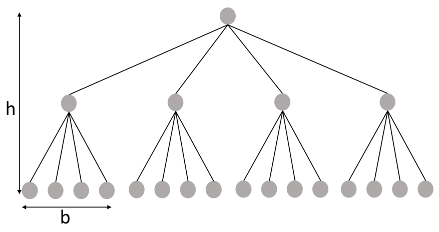

Figure 2: Hierarchical community tree with height h and branching factor b.

5. Relative graph generation

In order to control the evolution of the degree distribution of the generated temporal graphs, we propose in this section an extension of the CL model that is based on the optimal transport to compute the minimal distance between the degree distributions of each pair of successive graph snapshots.

5.1. Earth mover's distance

The Earth mover's distance can be defined as a measure of distance over a domain D between two distributions of the form $\{(x_1, \omega_1), ..., (x_n, \omega_n)\}$ where $x_i \in D$ and ω_i is the density of x_i. Having this, the problem reduces to the computation of an optimal flow (transportation matrix) $T = [t_{ij}]$ between two distributions $P = \{(x_1, p_1), ..., (x_n, p_n)\}$ and $Q = \{(y_1, q_1), ..., (y_n, q_n)\}$ such that t_{ij} is the mass transported between p_i and q_j which minimizes the overall cost:

$$\min_T \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij} d_{ij}$$

where $d_{ij} = d(x_i, y_j)$ is a measure of distance between x_i and y_j. The following constraints must hold for the optimal flow T:

$$t_{ij} \geq 0, \ 1 \geq i \geq n, \ 1 \geq j \geq m$$

$$\sum_{j=1}^{m} t_{ij} \leq p_i, \ 1 \geq i \geq n, \quad \sum_{i=1}^{n} t_{ij} \leq q_j, \ 1 \geq j \geq m$$

Once the optimal flow T is found, the EMD between P and Q is computed as follows:

$$EMD(P, Q) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij} d_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij}}$$

The EMD is fundamental in our generation procedure since it is used to compute the distance between two degree distributions as described in the following Section.

5.2. Baseline relative graph generation

In this section, we provide the baseline procedure of transforming a graph G with degree distribution ϕ into G' with degree distribution ϕ' which we refer to as the Baseline relative graph generation. Note that, we use this technique for generating temporal graphs such that G and G' corresponds to successive graph snapshots. For generalisation purposes, however, we remove the notion of time in this section. This transformation is enabled by a set of atomic graph operations including the addition and deletion of a vertex or an edge. Following the assumption that temporal graphs gradually evolve, this number of graph operations between successive snapshots should be minimized which is assured in our model by applying an optimal transport method.

Consider the input graph $G = \{V, E, \phi\}$ and degree distribution ϕ', the generated output graph $G' = \{V', E', \phi_{G'}\}$ such that $\phi_{G'}$ is an approximation of ϕ'. We define the distance between two degree distributions ϕ and ϕ' as the earth mover's distance $EMD(\phi, \phi')$.

Consider $\delta n = |V'| - |V|$ as the total number of vertices to be added to or removed from the graph based on whether δn is a positive or negative number, respectively. When adding a new vertex, this vertex is assigned with a degree equals to 0 and deleting a vertex consists of removing the vertex with its corresponding incident edges. This transformation phase assures that G and G' share the same number of vertices, hence, enables the transformation of ϕ into ϕ'. In order to morph ϕ into ϕ', a transportation matrix T is computed, where each row corresponds to a degree d in the set of degrees in the source distribution ϕ and each column corresponds to a

degree d' in the set of degrees in the target distribution ϕ'. Now, each cell consists of the portion of vertices having a degree d for which links are to be inserted or removed in order to be assigned a total number of edges equals to degree d'. That is, a vertex v_i, with a degree $d_{v_i} = d$, will be assigned a degree variation of $\delta d_{v_i} = d' - d$ resulting in a total number of edge insertions and deletions defined as D^+ and D^-, respectively.
We assign, for each vertex v_i, a linkage probability $p^+_{v_i}$ or a breakage probability $p^-_{v_i}$ defined as extensions of the CL linkage probability (1):

$$p^+_{v_i} = \frac{\delta d_{v_i}}{D^+}, \delta d_{v_i} > 0 \tag{4}$$

$$p^-_{v_i} = \frac{-\delta d_{v_i}}{D^-}, \delta d_{v_i} < 0 \tag{5}$$

We collect vertices sharing the same degree variation $\delta d = d' - d$ into a linkage pool if $\delta d > 0$ and in a breakage pool if $\delta d < 0$. Consider $\gamma_{d \to d'} = \{v_i | v_i \in V \wedge \delta_{v_i} = d' - d\}$ to be the pool containing vertices having a degree d that should be transformed into d'. We compute the probability of picking a linkage or breakage pool $p^+_{\gamma_{d\to d'}}$ and $p^-_{\gamma_{d\to d'}}$ as follows:

$$p^+_{\gamma_{d\to d'}} = \frac{\delta d |\gamma_{d\to d'}|}{D^+}, \delta d > 0$$

$$p^-_{\gamma_{d\to d'}} = \frac{-\delta d |\gamma_{d\to d'}|}{D^-}, \delta d < 0$$

However, breaking an edge might be impossible in situations where the source degree variation δd is negative and the sum of the negative degree variations of its neighbors is higher than δd. For the sake of illustration, we present in Figure 3 a graph in which the number of edges to remove from a node is higher than the sum of the number of edges to remove from its neighboring vertices. That is, the transformation of this graph implies removing 2 edges from vertex v_1 since $\delta v_1 = -2$. However, the number of the edges that have to be removed from the neighboring vertices of v_1 is equal to $\delta_{v_2} = -1$ since $\delta_{v_3} = 0$ and $\delta_{v_4} = 1$. To overcome this, we repeat the morphing procedure until EMD(ϕ, ϕ') reaches a desired threshold. Our simulations have proved that the value of EMD(ϕ, ϕ') converges rapidly towards the minimum threshold after a tolerable number of iterations. This statement will be further highlighted in Section 6.

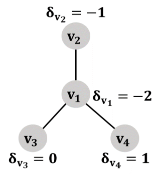

Figure 3: Graph representing the case of a non-possible edge breakage.

5.3. Relative community-aware graph generation

A more complex version of the previously described relative graph generation, consists of preserving the graph community structure in the transformation procedure. That is, the input of our community-aware relative graph generator is the graph $G = \{V, E, \phi_G, M_G\}$, the desired degree distribution ϕ and the stochastic block matrix M. However, the output consists of a graph $G' = \{V', E', \phi_{G'}, M_{G'}\}$ where $\phi_{G'}$ is an approximation of ϕ and $M_{G'}$ is an approximation of M. Recall that the generation procedure depicted in section 4.2 produces a graph with a given expected degree distribution and stochastic community matrix based on the proposed linkage probability duality presented in Equations (1) and (3). Indeed, a relative community-aware graph generation is based on an extension of the aforementioned duality by taking into consideration the degree variation of a vertex instead of the its degree. That is, the following linkage and breakage probabilities present a straightforward extension of Equations (4) and (5):

$$p^+_{v_i} = \frac{\delta d_{v_i}}{D^+_{c_m}} p_{c_m}, v_i \in c_m$$

$$p^-_{v_i} = \frac{\delta d_{v_i}}{D^-_{c_m}} p_{c_m}, v_i \in c_m$$

Where $D^+_{c_m}$ and $D^-_{c_m}$ are the total number of edge insertions and deletions in c_m, respectively. From the transportation matrix defined in section 5.2, we find n_{ij} as the portion of vertices with degree variation $\delta d = d_j - d_i$. However, finding the portion $n^{c_m}_{ij}$ of vertices in community c_m should satisfy three conditions detailed bellow. Each condition i results in a system of linear equations of the form $A_i X = B_i$ where X is a vector composed of $n^{c_m}_{ij}$ such that $X = \{n^{c_k}_{ij} | \forall 1 \leq i \leq |\phi_G| \wedge \forall 1 \leq j \leq |\phi| \wedge 0 \leq k \leq |C|\}$ where n_c is the total number of communities.

Condition 1: For each community $c_m \in C$, conditions stating that $D^+_{c_m} = D^+ p_{c_m}$ and $D^-_{c_m} = D^- p_{c_m}$ must hold, where D^+ and D^- are the total number of edge insertions and deletions in all communities of C, respectively. Incorporating $n^{c_m}_{ij}$ in the previous condition translates to the following equality:

$$\sum_{i=0}^{|\phi_G|} \sum_{j=0}^{|\phi'|} (d_j - d_i) n^{c_m}_{ij} = (\sum_{i=0}^{|\phi_G|} \sum_{j=0}^{|\phi'|} (d_j - d_i) n_{ij}) p_{c_m}$$

where ϕ_G and ϕ' are the source and target degree distributions.

Condition 2**:** This condition states that the sum of all portions of vertices with degree variation $d_j - d_i$ $\forall d_j \in \phi_t$ in c_m should be equal to the portion $n_i^{c_m}$ of vertices in c_m having a degree d_i resulting in the following equality:

$$\sum_{j=0}^{m} n_{ij}^{c_m} = n_i^{c_m}$$

Condition 3**:** This condition states that the portion n_{ij} of vertices with degree variation $d_j - d_i$ in the graph must be equal to the sum of all portions $n_{ij}^{c_m}$ $\forall c_m \in C$.

$$\sum_{c_m=0}^{n_c} n_{ij}^{c_m} = n_{ij}$$

By solving the concatenated system of equations obtained from the previous conditions $concat(A_1, A_2, A_3)X = concat(B_1, B_2, B_3)$, we find the vector X, hence the values of $n_{ij}^{c_m}$. Pools are created on a local basis in each community such that vertices with the same degree variation $\delta d = d' - d$ and belonging to the same community c_m are collected in a single pool $\gamma_{d \to d'}^{c_m}$. We compute the probability of picking a linkage or breakage pool $p^+_{\gamma_{d\to d'}, c_m}$ and $p^-_{\gamma_{d\to d'}, c_m}$ as follows:

$$p^+_{\gamma_{d\to d'}, c_m} = \frac{\delta d |\gamma_{d\to d'}^{c_m}|}{D^+}, \delta d > 0$$

$$p^-_{\gamma_{d\to d'}, c_m} = \frac{-\delta d |\gamma_{d\to d'}^{c_m}|}{D^-}, \delta d < 0$$

Algorithm `CRGG` depicts the relative community aware graph generation procedure. The input parameters are the graph snapshot G, desired degree distribution ϕ, density community matrix M, threshold of the EMD distance between ϕ_G and ϕ, maximum number of repetitions max_iter and the current number of repetitions cur_iter. Whereas, the output is a new graph snapshot G'. Note that, the value of cur_iter is equal to 0 in the first iteration. The transportation matrix T is computed using the function `getTransportMatrix` by taking the degree distributions ϕ_G and ϕ as input. The function `getVector`, computes A and B based on the Conditions 1, 2 and 3 and solves the system of equations defined by $AX = B$ to find the vector X. The total number of edges to add (D^+) and delete (D^-) are then computed based on the transporation matrix T. The function `getCDFComs` computes the cumulative distribution function $cdfCom$ based on the density community matrix M. Then, vectors $cdfPools^+$ and $cdfPools^-$ representing the cumulative density functions of the linkage and breakage pools and a list of logs (graph updates) L are initialized. The function `getCDFPools` is used to compute the cumulative distribution functions $cdfPools^+$ and $cdfPools^-$ based on the probabilities $p^+_{\gamma_{d\to d'}, c_m}$ and $p^-_{\gamma_{d\to d'}, c_m}$. The process of adding and removing edges is repeated D^+ and D^- times, respectively. In each iteration, communities c_n and c_m are picked based on $cdfComs$ and vertices n_i and n_j are picked using $cdfPools^+[n]$ and $cdfPools^-[m]$. Now, an addition or deletion graph update between the chosen vertices is added to the list of logs using functions `addEdge` and `removeEdge` whether the vertices where chosen from the linkage or breakage pools. However, breaking an edge might be impossible in some situations as shown in Figure 3. In such a use case, no graph update is added to the list of logs L. Finally, the EMD distance ϵ is computed between the obtained degree distribution ϕ'_G and the desired one ϕ. If ϵ' is higher than ϵ and the number of repetitions cur_iter has not yet reached max_iter, the same algorithm is repeated on the newly computed graph snapshot G'. The computation stops when ϵ' is lower than or equal to ϵ or the number of repetitions has already been reached.

Algorithm 1: CRGG

```
Input: $G = \{V, E, \phi_G, M_G\}$, $\phi$, $M$, $\epsilon$,
       $max\_iter$, $cur\_iter$
Output: $G' = \{V', E', \phi_{G'}, M_{G'}\}$
1  $T \leftarrow$ getTransportMatrix($\phi_G$, $\phi$) ;
2  X $\leftarrow$ getVector($\phi_G$, $\phi$, $T$, $M$) ;
3  $(D^+, D^-) \leftarrow$ getNumberOfEdges(T) ;
4  $cdfCom \leftarrow$ getCDFComs($M$) ;
5  $(cdfPools^+, cdfPools^-) \leftarrow$ initCDFPools ;
6  $L \leftarrow initLogs()$
7  for $c_m \in C$ do
8      $(cdfPools^+_{cm}, cdfPools^-_{c_m}) \leftarrow$
         getCDFPools($X$, $c_m$) ;
9      $cdfPools^+[m] \leftarrow cdfPools^+_{cm}$ ;
10     $cdfPools^-[m] \leftarrow cdfPools^-_{cm}$ ;
11 for $i \leftarrow 0$ to $D^+$ do
12     $(c_n, c_m) \leftarrow$ chooseComs($cdfCom$) ;
13     $(n_i, n_j) \leftarrow$ chooseVertices($cdfPools^+[n]$,
         $cdfPools^+[m]$) ;
14     $L$.addEdge $(n_i, n_j)$ ;
15 for $i \leftarrow 0$ to $D^-$ do
16     $(c_n, c_m) \leftarrow$ chooseComs($cdfCom$) ;
17     $(n_i, n_j) \leftarrow$ chooseVertices($cdfPools^-[n]$,
         $cdfPools^-[m]$) ;
18     $L$.removeEdge $(n_i, n_j)$ ;
19 $G' \leftarrow$ applyLogs($G$, $L$) ;
20 $\epsilon' \leftarrow$ getEMD($\phi$, $\phi'_G$) ;
21 if $\epsilon' \geq \epsilon \land cur\_iter < max\_iter$ then
22     $cur\_iter \leftarrow cur\_iter + 1$ ;
23     $G' \leftarrow$
         $CRGG(G', \phi, M, cur\_iter, max\_iter)$ ;
24 else
25     return $G'$ ;
```

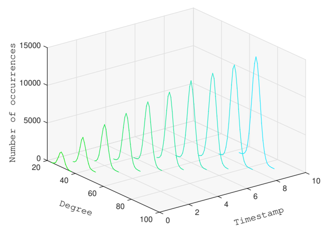

Figure 4: Gaussian degree distribution of a growth only graph

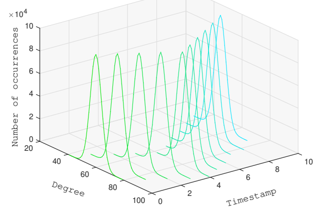

Figure 5: Gaussian degree distribution of a graph with edge deletions

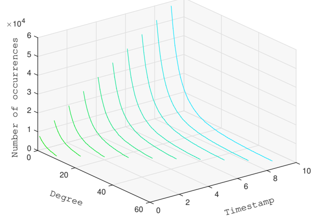

Figure 6: Zipfian degree distribution of a growth only graph

5.4. Accuracy of the generation procedure

In order to measure how far the characteristics of the generated graphs are from the ground truth parameters, we define two distance metrics ε_d and ε_c.

The first metric ε_d measures the inaccuracy of approximating the degree distributions of the generated graphs with the given sequence of degree distributions. That is, it measures the root mean square of the EMD distances between each degree distribution ϕ_i in the given sequence $\{\phi_1, \dots, \phi_n\}$ and its corresponding degree distribution ϕ_{G_i} in the sequence $\{\phi_{G_1}, \dots \phi_{G_n}\}$ extracted from the generated graphs. Having this, ε_d is computed as follows:

$$\varepsilon_d = \frac{\sqrt{\sum_{i=1}^{n}(EMD(\phi_i, \phi_{G_i}))^2}}{n}$$

Whereas, the second metric ε_c measures the inaccuracy of approximating the community density matrix of the generated graphs with a given stochastic matrix. That is, it measure the root mean square of the difference between the Frobenius norms of the given stochastic matrix M and the stochastic matrix M_{G_i} extracted from every generated graph snapshot. Having this, ε_c is computed as follows:

$$\varepsilon_c = \frac{\sqrt{\sum_{i=1}^{n}(F(M) - F(M_{G_i}))^2}}{n}$$

where $F(M)$ is the Frobenius norm of the stochastic community matrix M. We recall that the Frobenius norm of a matrix A of dimensions (n,m) is defined as follows:

$$F(A) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2}$$

6. Experimental evaluation

We conducted a number of experiments to validate the efficiency of our generator RTGEN. We also provide an insight on how changing the input parameters can steer the characteristics of the generated temporal graphs. Note that the source code of RTGEN is publicly available[1]. Besides the source code, we also provide the instructions describing how to use the tool to generate temporal graphs. For instance, users can pass the input parameters to describe the desired sequence of degree distributions or stochastic community matrix and the format of the generated output files to RTGEN using a terminal command. RTGEN proposes two output types: snapshot-based and event-based. The snapshot based type consists a sequence of graph snapshots represented each in a separate file. Whereas, the event-based type, consists of generating the sequence of graph updates (events) that we applied between successive snapshots to transform one snapshot into the next one.

6.0.1. Experimental setup

The experiments were conducted on a single machine equipped with Intel(R) Core(TM) i5-8350U CPU @ 1.70GHz 1.90 GHz, 16 GB memory and 500 GB SSD. We used Go 1.17.5 and Python 3.8.0. Besides, we referred to the optimal transport solver proposed in [25]. The graphs shown in this section are visualized using Gephi tool [26] which offers network visualization facilities and community detection algorithms [27].

6.0.2. Preliminaries

In the following experiments, we refer to two types of common degree distributions: Gaussian f_G and Zipfian f_Z that are defined as follows:

$$f_G(x) = \frac{1}{\sigma\sqrt{\pi}}e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$$f_Z(x) = \frac{1}{(x+v)^s} \quad x \in [0, d_{max}]$$

We consider a special case where the value of a parameter $x \in \mathbb{N}$ in iteration i depends on the its value in the previous iteration $i-1$ such as $x_i = x_{i-1} + \delta x$ such

[1]https://github.com/MariaMassri/RTGEN

that $\mu_i = \mu_{i-1} + \delta\mu$. This is applied on the parameters of the degree distributions $\mu, \sigma, d^{max}, s, v$ and n denoting the total number of vertices. That is, δn denotes the number of vertices to be added or removed from the graph in the relative generation process. Note that, RTGEN also generates the first snapshot which implies that the parameters of the degree distribution of the first snapshot should be given.

6.1. Controlling the evolution of the degree distribution

In this experiment, we show the evolution of the degree distribution of a sequence of graph snapshots generated with the relative generation procedure given a set of input parameters. Hence, we consider Gaussian and Zipfian degree distributions with different parameters and plotted the obtained degree distributions in Figures 4, 5 and 6. Figure 4 shows the evolution of the degree distribution of a generated sequence of 10 graph snapshots given the following parameters: $\{n^0 = 10K, \quad \mu^0 = 30, \quad \sigma^0 = 2, \quad \delta n = 10k, \quad \delta\mu = 5, \quad \delta\sigma = 0.1\}$. By setting $\delta\mu$ to 5, we increase the average degree by 5 between each pair of snapshots. This indeed, can model a growth-only graph where the average edge degree tend to regularly increase as the time elapses.
However, some real-world graphs are not growth-only in the sense that they are subject to edge deletions. This is indeed the case of human-proximity or transportation graphs where an important number of short-term connections is only valid during peak hours. To model this characteristic, RTGEN also supports edge deletions. The evolution of the degree distribution with edge deletions is presented in Figure 5. Let the following parameters define the evolution of degree distribution for $i \in [0, 4]$: $\{n^0 = 1M, \quad \mu^0 = 60, \quad \sigma^0 = 4, \quad \delta n = 0, \quad \delta\mu = 5, \quad \delta\sigma = 0\}$ Whereas the following parameters define its evolution for $i \in [5, 9]$: $\{n^0 = 10K, \quad \mu^0 = 80, \quad \sigma^0 = 2, \quad \delta n = 0, \quad \delta\mu = -5, \quad \delta\sigma = 0\}$. Indeed, setting $\delta\mu$ to -5 indicates that the average degree decreases by a value of 5 between each pair of successive graph snapshots.
Since real-world temporal graphs usually exhibit a power law degree distribution, we also generated graphs with an evolutionary Zipfian degree distribution composed of 10 graph snapshots as shown in Figure 6. For this generated temporal graph, we set the following parameters $\{n^0 = 50k, \quad s^0 = 2.5, \quad v^0 = 10, \quad d^0_{max} = 10, \quad \delta n = 50k, \quad s = 0, \quad \delta v = 0, \quad \delta d_{max} = 5\}$. By setting parameter δd_{max} to 5, we consider that the maximum degree of nodes increases by a value of 5 between each pair of successive snapshots. Whereas, the value of δn indicates that $50k$ new nodes join the graph between successive snapshots. These parameters reflect the growth of a large number of real-world temporal graphs where new nodes join the graph and new connections are created as the time elapses.

6.2. Controlling the community structure of the generated graphs

In this experiment, we show the generated community structure with different parameters of the stochastic community matrix and the effect of varying parameter k of the hierarchical tree. As described in Section 4.3, RTGEN is capable of auto-generating the stochastic community matrix representing a hierarchical community structure. Consider a stochastic community matrix generated by setting $b = 4$ and $h = 2$. As depicted in Equation 2, one can tune the parameter k in order to control the within and between edge densities. Hence, we select three different values of k in $\{2, 4, 8\}$. Furthermore consider, $n = 1000$ to be the total number of vertices and parameters $\mu = 30$ and $\sigma = 2$ to be the parameters of a Gaussian distribution. Note that, in this experiment, we generate a single graph snapshot relying on the generation procedure proposed in Section 4.2. The generated graphs are shown in Figures 7a, 7b and 7c using the Gephi tool. It can be noticed that the difference between the within and between edge densities is proportional to k since $k \propto p^{in}_{c_i} - p^{out}_{ci}$ where $p^{in}_{c_i}$ and p^{out}_{ci} are the within and between linkage probabilities of a community c_i. Furthermore, Figure 8 presents the modularity in function of parameter k which we vary from 0 to 32. The modularity is a measure to quantify the goodness of community structure. Its formula compares, for all the communities, the fraction of edges that falls within the given community with the expected fraction if edges were distributed at random. It is clear from the results that the modularity increases with the increase of k. This is justified by the fact that k is proportional to the difference between within and between edge linkage probabilities $p^{in}_{c_i} - p^{out}_{ci}$.

6.3. Generating graphs with deletions between snapshots

As mentioned in Section 5, the relative graph generation procedure may incur a number of edge deletions. This can be cumbersome when the number of edges to delete for a given vertex is higher than the total sum of edges to delete from its neighboring vertices. We solve this problem by repeating the generation process until reaching an acceptable error threshold that is defined by the EMD between the obtained and desired degree distributions. Figure 12 shows the variation of the number of iterations and the execution time of the generation process in function of the threshold error defined by the EMD. The obtained results show that our generation procedure converges rapidly to a tolerable threshold. That is, a threshold equals to 0.001 can be reached with only 7

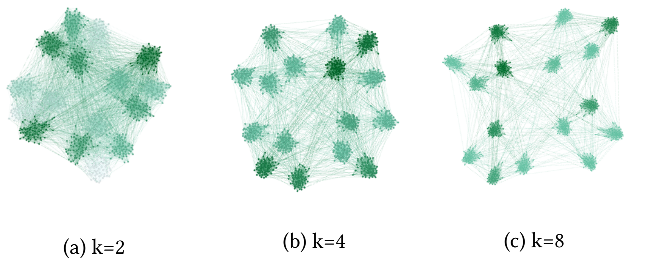

Figure 7: A visualization of the generated graphs with a hierarchical community structure with parameters: $b = 4$, $h = 2$, $c = 4$ and a varying k.

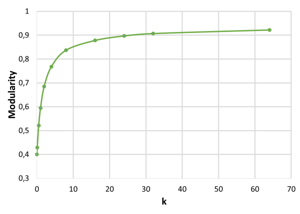

Figure 8: Modularity value in function of parameter k ranging from 0 to 32.

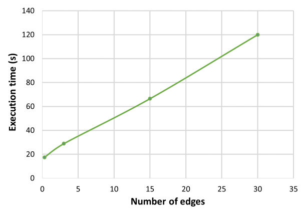

Figure 9: Execution time in function of the number of edges.

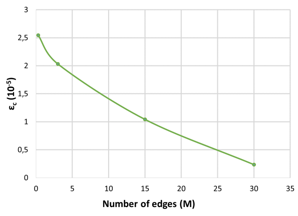

Figure 10: ε_d in function of the number of edges.

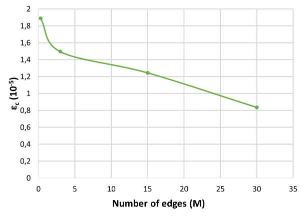

Figure 11: ε_c in function of the number of edges.

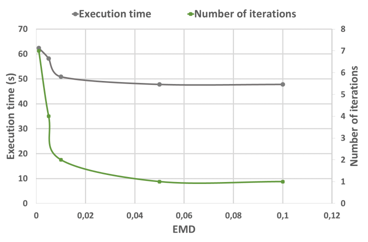

Figure 12: The variation of the number of iterations and execution time in function of the EMD.

iterations. By comparing the execution time of 1 iteration and 7 iterations, we can notice that the difference is lower than the execution time of a single iteration. Indeed, the execution time resulting from repeating the generation is lower than the first iteration since the majority of modifications are added in the first iteration and only the remaining vertices whose linkage probability does not satisfy the sum of the linkage probabilities of its neighboring vertices are considered in the next iteration. Note that these results are obtained from the generation of two successive snapshots with the following input parameters of a Gaussian degree distribution: $\{n_0 = 500k, \quad \mu_0 = 60, \quad \sigma_0 = 2, \quad \delta n = 0, \quad \delta mu = -30, \quad \delta\sigma = 0\}$.

6.4. Accuracy of the generation procedure

We quantify the accuracy of the generated graphs with the given parameters by computing the distance metrics ε_d and ε_c defined in Section 5.4. We generated a sequence of $n = 5$ snapshots with the following parameters of Gaussian degree distribution: $\{n_0 \in \{10k, \quad 100k, \quad 500k, \quad 1M\}, \quad \mu_0 = 30, \quad \sigma_0 = 2, \quad \delta_n = 0, \quad \delta\mu = 10, \quad \delta\sigma = 0\}$. Besides, we controlled the community structure by fixing the following parameters of a hierarchical tree: $h = 2, \quad b = 2, \quad c = 4, \quad k = 0$.

Figures 9, 10 and 11 plot the execution time, value of ε_d and ε_c in function of the total number of created edges from applying the Gaussian distribution whose parameters are given above. It is clear that the execution time increases with the number of the generated edges. The distance metric, however, decreases implying that RT-GEN approximates more accurately the given sequence of degree distribution and community structure as the total number of edges grows.

7. Related work

Synthetic graphs are important for developing benchmarks for assessing the performance of graph-oriented data platforms, when real graphs are not publicly available or expensive to obtain. This has been the incentive to design models and generators, which are very useful for evaluating the efficiency of graph management techniques as storage, query evaluation, indexing, partitioning, etc.

An extensive work has been posited for the generation of static graphs. For instance, a special emphasis has been placed to control the degree distribution of the generated graphs. In this context, many graph generators were designed such as RTG [3], RMAT [4] and its generalisation Kronecker [5] producing only Power-Law distributions. Since real-world graphs are not limited to power-law distributions, BTER [6] and its extension Darwini [7] and GMark [8] produce graphs with any user defined distribution.

Another graph generation model producing a given degree distribution is the CL model, forming the basis of the RTGEN tool. This model can be regarded as a successor of the Erdos-Rényi model [23] that is designed for the generation of random graphs and a variant of the edge configuration model of Newman et al. [24]. It was extensively discussed and reused [17, 28, 29, 30]. We choose to extend this model for its simplicity and scalability.

Besides, a number of existing graph generators are community-aware in the sense that they collect vertices that are more densely connected between each other than they are with the rest of the graph, in separate or overlapping subgraphs called communities [9, 10, 11]. Although these generators preserve a given community structure, they fail to produce a graph with respect to a given degree distribution. In this paper, we overcome this limitation by allowing not only the generation of a given community structure but also a given degree distribution.

Despite the extensive work posited on the generation of non-temporal graphs, the generation of temporal graphs has received much less attention. For instance, DANCer [31] is capable of generating temporal, community-aware property graphs. It separates operations performed on communities (macro operations) from operations performed on vertices and edges (micro operations). ComAwareNetGrowth [32] is a community-aware graph generator that is capable of creating growth only graphs. APA (Attribute-Aware Preferential Attachment) [33] is a graph generator capable of creating growth-only property graphs based on a non-conventional triangle closing. Instead of closing a triangle based on a uniform probability given as an input parameter, their proposed model consists of closing a triangle based on the similarity between the candidate edge's endpoints. While GMark [8] generates static graphs, EGG (Evolving Graph Generator) [34] proposes an extension including evolving properties attached to each vertex. EGG, however, disregard the topological changes to the network and narrow the temporal evolution of the graph to property updates. DSNG-M (dynamic social network generator based on modularity) [35] is a graph generator that is capable of generating temporal graphs by flipping the direction of edges of a given graph in order to satisfy a randomly chosen modularity value assigned to a single graph snapshot.

Some of the aforementioned graph generators produce temporal graphs with properties on nodes or vertices, which we do not address in this paper. None of them, however, allows the control of the evolution of the degree distribution given ground truth parameters that describe this evolution. This challenge lead to the elaboration of the RTGEN tool that allows the approximation of any given sequence of degree distributions that describes the evolution of the graph. We firmly believe, that the degree distribution is a key feature that characterizes graphs, hence, it should not be disregarded in graph generation tools.

8. Conclusion

In this paper, we addressed the generation of temporal graphs that represents a critical challenge in the design of benchmarks specific for evaluating temporal graph management systems. That is, we proposed RTGEN, a temporal graph generator that produces a sequence of graph snapshots whose community structure and evolution of the degree distribution results from approximating user defined parameters. This generation procedure consists of relatively generating a graph snapshot from a previous one by applying a number of atomic graph operations. Our generation technique relies on an Optimal transport solver to approximate a user-defined sequence of degree distributions while minimizing the number of operations needed to transform one snapshot into its successor. We conducted a number of experiments that validated the efficiency and accuracy of our generation procedure. In the future, we are planning to include a dynamic community structure to RTGEN. Indeed, the communities found in real-world graphs are subject to splits, merges, shrinks or expansions which should also be modelled in synthetic graphs.

References

[1] J. R. Clough, T. S. Evans, Time and citation networks, arXiv preprint arXiv:1507.01388 (2015).

[2] M. D. Mueller, D. Hasenfratz, O. Saukh, M. Fierz, C. Hueglin, Statistical modelling of particle number concentration in zurich at high spatio-temporal resolution utilizing data from a mobile sensor network, Atmospheric Environment 126 (2016) 171–181.

[3] L. Akoglu, C. Faloutsos, Rtg: a recursive realistic graph generator using random typing, in: Joint

European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2009, pp. 13–28.

[4] D. Chakrabarti, Y. Zhan, C. Faloutsos, R-mat: A recursive model for graph mining, in: Proceedings of the 2004 SIAM International Conference on Data Mining, SIAM, 2004, pp. 442–446.

[5] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani, Kronecker graphs: An approach to modeling networks, Journal of Machine Learning Research 11 (2010) 985–1042.

[6] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, A scalable generative graph model with community structure, SIAM Journal on Scientific Computing 36 (2014) C424–C452.

[7] S. Edunov, D. Logothetis, C. Wang, A. Ching, M. Kabiljo, Darwini: Generating realistic large-scale social graphs, arXiv preprint arXiv:1610.00664 (2016).

[8] G. Bagan, A. Bonifati, R. Ciucanu, G. H. Fletcher, A. Lemay, N. Advokaat, gmark: Schema-driven generation of graphs and queries, IEEE Transactions on Knowledge and Data Engineering 29 (2016) 856–869.

[9] P. W. Holland, K. B. Laskey, S. Leinhardt, Stochastic blockmodels: First steps, Social networks 5 (1983) 109–137.

[10] B. Karrer, M. E. Newman, Stochastic blockmodels and community structure in networks, Physical review E 83 (2011) 016107.

[11] B. Kamiński, P. Prałat, F. Théberge, Artificial benchmark for community detection (abcd): Fast random graph model with community structure, arXiv preprint arXiv:2002.00843 (2020).

[12] P. Holme, Modern temporal network theory: a colloquium, The European Physical Journal B 88 (2015) 234.

[13] E. Pitoura, Historical graphs: models, storage, processing, in: European Business Intelligence and Big Data Summer School, Springer, 2017, pp. 84–111.

[14] Y. Miao, W. Han, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, E. Chen, W. Chen, Immortalgraph: A system for storage and analysis of temporal graphs, ACM Transactions on Storage (TOS) 11 (2015) 1–34.

[15] U. Khurana, A. Deshpande, Storing and analyzing historical graph data at scale, arXiv preprint arXiv:1509.08960 (2015).

[16] M. Haeusler, T. Trojer, J. Kessler, M. Farwick, E. Nowakowski, R. Breu, Chronograph: A versioned tinkerpop graph database, in: International Conference on Data Management Technologies and Applications, Springer, 2017, pp. 237–260.

[17] F. Chung, L. Lu, The average distances in random graphs with given expected degrees, Proceedings of the National Academy of Sciences 99 (2002) 15879–15882.

[18] A. G. Labouseur, J. Birnbaum, P. W. Olsen, S. R. Spillane, J. Vijayan, J.-H. Hwang, W.-S. Han, The g* graph database: efficiently managing large distributed dynamic graphs, Distributed and Parallel Databases 33 (2015) 479–514.

[19] M. Then, T. Kersten, S. Günnemann, A. Kemper, T. Neumann, Automatic algorithm transformation for efficient multi-snapshot analytics on temporal graphs, Proceedings of the VLDB Endowment 10 (2017) 877–888.

[20] C. Ren, E. Lo, B. Kao, X. Zhu, R. Cheng, On querying historical evolving graph sequences, Proceedings of the VLDB Endowment 4 (2011) 726–737.

[21] W. Aiello, F. Chung, L. Lu, A random graph model for power law graphs, Experimental Mathematics 10 (2001) 53–66.

[22] F. Chung, L. Lu, Connected components in random graphs with given expected degree sequences, Annals of combinatorics 6 (2002) 125–145.

[23] P. Erdos, A. rényi on random graphs i, Publ. Math. Debrecen 6 (1959) 290–297.

[24] M. E. Newman, D. J. Watts, S. H. Strogatz, Random graph models of social networks, Proceedings of the national academy of sciences 99 (2002) 2566–2572.

[25] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, T. Vayer, Pot: Python optimal transport, Journal of Machine Learning Research 22 (2021) 1–8. URL: http://jmlr.org/papers/v22/20-451.html.

[26] M. Bastian, S. Heymann, M. Jacomy, Gephi: an open source software for exploring and manipulating networks, in: Third international AAAI conference on weblogs and social media, 2009.

[27] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of statistical mechanics: theory and experiment 2008 (2008) P10008.

[28] F. Chung, F. R. Chung, F. C. Graham, L. Lu, K. F. Chung, et al., Complex graphs and networks, 107, American Mathematical Soc., 2006.

[29] A. Pinar, C. Seshadhri, T. G. Kolda, The similarity between stochastic kronecker and chung-lu graph models, in: Proceedings of the 2012 SIAM International Conference on Data Mining, SIAM, 2012, pp. 1071–1082.

[30] M. Winlaw, H. DeSterck, G. Sanders, An in-depth analysis of the chung-lu model, Technical Report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2015.

[31] O. Benyahia, C. Largeron, B. Jeudy, O. R. Zaïane, Dancer: Dynamic attributed network with com-

munity structure generator, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2016, pp. 41–44.

[32] F. Gursoy, B. Badur, A community-aware network growth model for synthetic social network generation, arXiv preprint arXiv:1901.03629 (2019).

[33] A. Aghasadeghi, J. Stoyanovich, Generating evolving property graphs with attribute-aware preferential attachment, in: Proceedings of the Workshop on Testing Database Systems, 2018, pp. 1–6.

[34] K. Alami, R. Ciucanu, E. M. Nguifo, Synthetic graph generation from finely-tuned temporal constraints., in: TD-LSG@ PKDD/ECML, 2017, pp. 44–47.

[35] B. Duan, W. Luo, H. Jiang, L. Ni, Dynamic social networks generator based on modularity: Dsng-m, in: 2019 2nd International Conference on Data Intelligence and Security (ICDIS), IEEE, 2019, pp. 167–173.