TRIGGER: TempoRal Interaction Graph GenEratoR

M. Yusuf Özkaya
myozka@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia

Ali Pinar
apinar@sandia.gov
Sandia National Laboratories
Livermore, California

Ümit V. Çatalyürek
umit@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia

ABSTRACT

Efforts on temporal graph generation have focused on generating instances from the same steady state (e.g., keeping a fixed-size window over a sequence of edges generated from the same model). Unfortunately, such generators cannot capture the underlying information richness of the temporal aspects of activity graphs. Based on the underlying phenomena being represented by the graph, temporal properties of interest will vary. In addition to topological features, such as neighbor information, we are interested in frequencies of communication. Subsequently, our work can be split into two natural steps: building models that can represent the temporal characteristics of nodes of a graph and generating temporal activity graphs that display the features of our model.

We present TRIGGER (TempoRal Interaction Graph GenEratoR): A Markov Model-based activity generation approach that classifies the nodes into profiles and generates a series of repeating interactions in continuous time. Then, we show how to estimate an input model to represent a real world graph. We carried out extensive experiments to validate our approach using various real-world temporal datasets and metrics on the quality of generated graphs. We show that our approach can generate realistic temporal activity graphs and match temporal metrics such as burstiness, spread, and persistence and static metrics at both graph and the node scale.

KEYWORDS

graph generation, interaction network, activity network, activity graphs, temporal graphs, time-varying graphs, continuous time, network traffic, scalable graph algorithms, datasets

ACM Reference Format:

M. Yusuf Özkaya, Ali Pinar, and Ümit V. Çatalyürek. 2018. TRIGGER: TempoRal Interaction Graph GenEratoR. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Graph generation is one of the most important and increasingly popular research areas. The proprietarity and lack of availability of many datasets directed many efforts towards generation of realistic graph data. Benchmarking for scalable computation requires extremely large datasets; so large that regeneration of a graph is more efficient than storing and transferring. Even when a graph is available, researchers are interested in generating similar graphs to avoid over-tuning their algorithms for an instance and to observe how their techniques can perform for changing properties of the graphs (e.g., denser, more vertices, heavier tails in degree distributions). As such, many researchers have since focused their efforts to designing realistic graph generators [8, 10, 13, 14, 26]. Many also focus on scaling up such generators as well as designing novel scalable alternatives [15, 17, 23, 24, 27].

Until recently, only a few explored the features and qualities a realistic *temporal* graph generator should have. Many research areas such as epidemiology, sociology, and social network analysis have extensive use for temporal network representations [9, 19, 21, 22, 28, 30]. Temporal graph generation itself has a diverse set of problem definitions. We present our categorization for temporal graph generation. First, we divide the problems by what the temporal aspect represents: (i) changes in the graph structure (evolving/dynamic graphs) and (ii) interactions on a graph structure. Then, we further divide each category as: changes (interactions) represented as time window snapshots, as a series of interactions, and finally, *continuous-time* (interaction) graph generation where there are changes (interactions) with meaningful timestamps, where the point of interest is those timestamps when the events occur.

[9] reviews aspects of temporal networks including generation approaches. [4] studies random shuffling methods and [16, 29] propose latent (deep-learning based) information models. [3] present a generation algorithm built on power-law based inter-event times. As such, although there are many temporal graph generators, many models do not generate continuous-time interaction networks. The ones that do generate usually assume a single distribution to model the interaction intervals. The goal of this work is to create a scalable realistic temporal interaction graph generator that uses input data parsimoniously. So, the desiderata of our problem: (1) The algorithm should require minimal input. (2) The generation should be fast and scalable. (3) The generated graphs should be realistic. These three points are usually the corners of a tradeoff triangle: It is easier to create more realistic graphs with more useful input, it is harder to create a realistic graph fast and scalably, etc.

Depending on the phenomena represented by the graph, temporal features of interest vary widely. Therefore, it is imperative to be specific about the temporal features of interest. We focus on burstiness of nodes. Our intuition is that nodes of a network have different interaction burstiness characteristics at different points in time. We also consider *persistence* metric defined by Belth et al.[2] to explore the qualities of the temporal graphs. To the best of our knowledge, there is no scalable continuous-time temporal graph generator available with the focus on interaction times.

Our contributions in this work are: (1) a novel model building phase that parsimoniously represents the temporal characteristics

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

of a graph. (2) a novel continuous-time temporal interaction graph generator that uses the model. (3) an extensive empirical analysis of the quality of our algorithm in terms of generating a realistic graph using various real-world datasets and metrics.

2 PROBLEM DEFINITION

We believe, the *activities* of a node has an inherent structure that defines the relationship between interactions. For example, a person can be receiving a phone call, and that might trigger multiple follow-up calls right after. This could look like a burst of interactions with long *dead times* (time intervals of no activity) in between. A web server might be running a scheduled program with uniform dead times. Hence, many applications have inherent context-dependent inter-interaction dead time models. Following these examples, we believe many applications can be represented with a well-defined Markov model summarizing the correlation between dead times.

Definition 2.1 (Graph Generation). Given a number T of node types, number $|V|$ of total node count, Markov models with η states for T types, a time window size t, and a black-box static graph structure generator; generate a temporal graph consisting of interactions: 3-tuples of (source, destination, timestamp - ts). The generated stream of interactions should match node-scale and graph-scale temporal metrics such as burstiness, frequency, spread and persistence (see § 5.1).

3 THE TRIGGER MODEL

Let t be a 3-tuple of $(source, destination, timestamp)$ of a single interaction. TRIGGER generates a collective list of interactions $D = (t_1, t_2, \dots)$ for a given time limit.

For a single node, our algorithm consists of decoupled procedures of interaction destination generation and interaction timestamp generation. Capturing the inter-relation between time aspect and the destination is a significant context-dependent problem. In this work, the destinations are randomly selected from a set of possible neighbors provided by a black-box static graph generator. A simple extension with minimal effect on input data size would be to introduce a global tendency parameter for recurring interactions.

Our input model for graph generation consists of the following:

- a static graph model (degree-corrected stochastic block model),
- For each node $v \in V$:
  - number of interactions (C_v),
  - interaction profile ($type_v$),
  - start timestamp,
- Number of types (profiles) T
- For each type i a Markov data structure consisting of:
  - a Markov Matrix (MM_i) of size $\eta \times \eta$,
  - an array of Gaussian distribution mean (ctr_i) and standard deviations (std_i) of size η,
  - an overall state presence probability I_i (aka weights of individual Gaussian distributions over the dead times) of size η.

Most decisions are made towards requiring smaller input while maintaining important aspects of a random graph. We selected Markov modulated Gaussian process (MMGP) above others since central limit theorem is applicable as we combine independent nodes/dead time instances (as opposed to, e.g., BuSca [1]). MMGP is a model where each Markov state represents a separate Gaussian distribution. The inter-event (dead) times, dt, are sampled from the Gaussian process defined for the Markov state at hand. By using Gaussian processes and clustering the nodes into types, we only require Markov model information for T types instead of $|V|$ nodes. The memory footprint of the input for temporal aspect is $O(|V| + \eta^2 \times T)$. Taking the constants into account, 3 integers for each *node*, and $\eta \times (\eta + 3)$ doubles for each *type*.

In our work, we have implemented a Degree-Corrected Stochastic Block Model (DCSBM) based approach to generate a static graph. Then, during the temporal edge generation, the destinations for the interactions are selected from the neighbor set of the source node in this static graph. Static structure generation is an important aspect that may require extensive efforts and application-specific information. We treat DCSBM as a *black-box* generator. A DCSBM with B blocks uses $B \times B$ integer matrix for the block model, degree and group membership integers for each node ($3 \times |V|$). It is straightforward to replace with any other generic or context-specific generator. If having an evolving neighborhood is needed, one can easily use a custom approach such as a link prediction algorithm that allows it.

Better results relating to the destination node statistics can be achieved with a predefined graph generator. One way to do this is to collect all edges in a temporal graph and create a static fold of it, and then treat this static fold as the basis of destination generation. For the static metric evaluation, we use this static fold basis.

The temporal aspect, i.e., generation of timestamps for interactions, is defined completely independent of destination selection procedure. Here, MM_i Markov matrix consists of the dead time interval generators as states and the transition probabilities between the states as values in the matrix. Rarely, Gaussian distribution can even return negative numbers, we ignore and resample from the distribution in this case. One variation of our approach uses log-normal distribution instead of Gaussian, leading to no negative samples throughout. Each state represents a Gaussian distribution as below.

$$X \sim \mathcal{N}(\mu = ctr_v[s],\ \sigma^2 = std_v[s]^2),\ s \in [0, \eta),\ s \in \mathbb{Z}^+. \quad (1)$$

The generation of dead times are independent for each node. For each node, C_v interactions (C_v destinations and $C_v - 1$ dead times) are generated. At the beginning, a random state is selected with respect to their overall presence (probability of being in a state). Starting from this state, consecutive states are generated according to the Markov matrix. For each state s, dead times are generated from the Gaussian distribution defined for that state (eq. 1). Next, all dead times are converted to timestamps for interactions via prefix sum: $ts_i = start_v + \sum_{k=0}^{i-1} dt_k$. Then, destinations and timestamps are paired as the output data.

It is important to provide opportunities of variations in a generator. We introduced a $|noiseFactor| < 15\%$ that changes the total number of interactions of each node. Similarly, we provide the option to add a random deviation multiplier which affects the dead times of nodes.

The Markov model requires the dead times to be generated one by one. Thus, the runtime complexity of the interaction generation for a single node v is a linear function of C_v. Each node's interactions are generated independently, hence can be computed in parallel.

Burstiness is an important concept being used for anomaly detection. Our intention is to extend our work to allow further changes in the parameters (type of a node, neighbors of a node, states of a type, etc.) during generation, and create room for anomaly/change detection benchmarking.

4 TRIGGER: ESTIMATING PARAMETERS

Here, we explain how to create our input model using a temporal graph as input. For the static representation, the graph is fed into a DCSBM model (we use Peixoto's approach [20]). For each input instance, all edges are aggregated to a single directed weighted static graph. This graph is used as the input for DCSBM model.

Building the Markov models include several design choices decided with respect to ease of use, quality, and speed. They can be generated in many ways. Here we briefly discuss some of these decisions. We first create Markov Models for every node separately, and then, cluster them into T types (profiles).

4.1 Deciding the number and values of states

The number of states η for the Markov models represent the different interaction frequency/variation states.

The principal way to figure out how many components (states η) should be in the mixture and assign the parameters of those component distributions in the research community is through a Dirichlet Process. Another prominent approach is the use of Markov Chain Monte Carlo approximation. Both are algorithms that may take considerable amount of time and effort.

We generate the Markov matrices with the following algorithm. Let D_v be the set of interaction 3-tuples sourced from node v. Assuming there are C_v interactions, the list of dead times of length $C_v - 1$ is computed as the distances between adjacent interactions, i.e., $dt_i = ts_{i+1} - ts_i$ for $i \in [0, C_v - 1)$. Then, a user-specified number of states η is used as the number of components in a Gaussian Mixture Model (GMM) built with Expectation-Maximization (EM) algorithm. The states of our Markov model are then assigned as the components of this GMM model. This model assigns a label $l(dt_i)$ for each dead time representing the most likely state (Gaussian distribution) it is sampled from.

For each node v, we create a Markov matrix (MM_v) of size $\eta \times \eta$ and fill it. We use m_{sd} to represent the s^{th} row and d^{th} column of MM_v. Now, each row s of MM_v contains the number of times the node v jumped from state s to any of η states. Next, we normalize each row by the sum of the row (c_s) to convert them to probabilities. $m_{sd} = \sum_{i=0}^{C_v - 2} \mathbf{1}(s = l(dt_i) \wedge d = l(dt_{i+1}))$, $c_s = \sum_{k=0}^{\eta-1} m_{sk}$.

4.2 Deciding the number of types

Storing a separate Markov Model for each node is not memory efficient. We cluster the nodes with respect to their communication profiles represented by their Markov models. One principal way of doing this would be to represent the Markov matrices as a tensor, and apply Tensor Component Analysis (TCA) followed by a clustering algorithm using the output of TCA. Here, we require the user to provide a-priori knowledge on how many types are expected as an input (T) instead. Then, apply weighted K-Means (or EM with GMM) to assign types to nodes. Clustering algorithm uses the state center values and probabilities of being in those states as the feature vector. The weights are assigned with respect to the number of interactions of nodes. The resulting model for a type i is the average of models of all nodes v where $type_v = i$.

5 EVALUATION

5.1 Metrics

Many researchers evaluate their temporal generators by converting them into snapshots and using static graph metrics. For the sake of completeness, we briefly touch some static graph metrics including average degree, number of connected components, reciprocity, etc. Furthermore, we expect a generated temporal graph to match various temporal properties such as, burstiness, spread, and persistence at both node and graph scale. Exploring other temporal metrics such as variations of latency, reachability, connectedness, temporal motif counts, etc. are our ongoing future work.

5.1.1 Static Fold Graph Properties. Our first set of metrics are the static graph properties. We create a static fold of the graph by converting the interactions generated to weighted edges. Then, we measure: Maximum values for number of vertices, edges, average degree, edge weight, number of connected components, size of largest connected component, clique number, shell index, maximum in-degree, maximum out-degree, (average values for) density, local transitivity, assortativity, (weighted) diameter, reciprocity and weighted reciprocity.

5.1.2 Burstiness. Burstiness in a temporal graph is an empirical quantity that compares the sequence of dead times (dt) with one that is generated by a Poisson process. For a Poisson process, the ratio of standard deviation to the mean is 1 by definition. The burstiness measure compares σ/μ of dt to that of $Pois$ [5].

$$B = \frac{\sigma_{dt}/\mu_{dt} - \sigma_{Pois}/\mu_{Pois}}{\sigma_{dt}/\mu_{dt} + \sigma_{Pois}/\mu_{Pois}} = \frac{\sigma_{dt}/\mu_{dt} - 1}{\sigma_{dt}/\mu_{dt} + 1} \quad (1)$$

Burstiness has the range $[-1, 1]$. When $B = 1$, the sequence is maximally bursty. $B = 0$ means it is a Poisson process, and $B = -1$ points to a sequence with fixed intervals. Burstiness is undefined for a node v if $C_v < 2$ since that means there is no dead time.

We compare the burstiness of each node in the original graphs with the corresponding node in the generated graphs. We report mean (ME) and mean squared-errors (MSE). Finally, we compare the graph-scale burstiness of input and generated graphs.

5.1.3 Frequency, Spread, and Persistence. *Frequency* is defined as $log_{10}(\#interactions) + 1$ [2]. Shannon entropy is a measure for the average level of new information in a set. Belth et al. [2] define *Spread* as the normalized Shannon entropy (normalized by the $\log(\#interactions)$). The spread of a series of dead times dt is defined as: $S(dt) = \frac{H(dt)}{\log(|dt|)} + 1$ when $|dt| > 1$. And, $S(dt) = 1$ for $|dt| \in \{0, 1\}$. *Width* is the time difference between first and last interactions normalized by the whole time window size. A node may only have interactions between 3PM and 5PM in a 24 hour input span. Then,

$$Width = \frac{(5PM - 3PM) + 1}{24hours + 1} = \frac{7201sec}{86401sec} \approx \frac{1}{12}.$$

Persistence is the multiplication: $Width^{\alpha} \times Frequency^{\beta} \times Spread^{\gamma}$. We selected powers ($\alpha = 1; \beta = 0.2; \gamma = 5$) for all datasets. Since

Table 1: Comparison of static graph properties in single input and generated instances.

Graph	$\|V\|$	$\|E\|$	Avg. Deg	# CC	$\|CC\|$	In-	Out-	Density	Transitvty	Assort.	Recip.
bike	255	4635	18.176	19	231	73	66	0.0716	0.325	0.076	0.4363
gen-bike	255	3439	13.539	28	230	62	44	0.0535	0.261	0.085	0.3246
darpa-ip	25525	68910	2.700	18170	7356	8063	7295	0.00011	5.11E-06	-0.4144	0.2226
gen-darpa	23880	59417	2.490	16567	7321	8062	7260	0.00010	4.97E-06	-0.3843	0.2581
email-eu	986	24929	25.283	184	803	211	333	0.0257	0.267	-0.0137	0.7112
gen-email	968	20514	21.302	179	795	185	291	0.0221	0.254	0.0001	0.6583
mooc	3850	37080	9.631	3829	22	3658	19	0.0025	0.0125	-0.249	0.0031
gen-mooc	3850	28605	7.430	3833	22	3623	17	0.0019	0.0104	-0.235	0.0028
nyc-taxi	258	9339	36.198	31	228	89	231	0.141	0.563	-0.064	0.5585
gen-nyc	255	7971	31.259	36	227	83	225	0.123	0.56	-0.056	0.5576
reddit	9081	18382	2.024	7511	1470	442	329	0.00022	0.0655	-0.0702	0.1098
gen-reddit	8720	17612	2.020	7634	1446	432	323	0.00023	0.0443	-0.0451	0.0743

Spread and Persistence does not have upper bounds, we cannot convert that to error percentages directly. We show the MSE and Weighted MSE of these metrics averaged over all applicable nodes in the graphs. (for nodes v where $C_v > 1$).

Graph-scale versions of these metrics are computed similar to the node-scale versions. The difference is that instead of using the interactions of a single node, all interactions in the network are considered. This is an important distinction: Having small errors for individual nodes does not mean the whole graph as one entity has small errors, distinct nodes with maximal bursts can cause a monotonous overall rate if their start times are spread uniformly.

Following Belth et al.'s analysis approach, we show Burstiness, Spread, and Persistence plots against Frequency in Appendix B.

5.2 Datasets

The data sets we have used are summarized in Table A1.

In this work, we selected an arbitrary, intuitive chunk size for our datasets. By *chunk*s and *size*, we mean unique individual continuous segments from the dataset and their time windows we used as a single input. Selecting a meaningful size is an important research direction in itself [25]. We treat the chunks from the same source as distinct inputs. We collect them together for reporting the results, since ultimately, they are from the same context and may present similar trends. For all datasets we use $T = 16$ and $\eta = 8$.

5.3 Static Properties

Although it is not the main point of this work, we present a static property comparison for completeness. The comparison results of some selected static properties for a subset of input graphs are summarized in the Table 1. Here in this section, we only show results for single arbitrary inputs and corresponding outputs.

The table shows that our generator can match the given metrics closely. The variations, mostly under-generations, are caused by the random destination sampling with replacement: There is a chance some edges never occur for a node with a high degree. One can solve this by enforcing a pre- or post-processing step that enforce at least one interaction on all possible edges. This divergence from the number of neighbors reflect on other metrics as well: (inversely proportionally for) number of connected components, diameter, and (proportionally for) number of nodes, number of edges, average degree, connected component sizes, density, transitivity, and reciprocity. When the graphs do not have such skewed degree distributions, the fluctuations on those metrics are smaller, such as `reddit` and `boston-bike` datasets.

Table 2: (Weighted) Mean and (Weighted) Mean Squared-Error Comparison for Temporal Metrics.

	Burstiness		Burstiness		Spread		Persistence	
Graph	MSE	WMSE	ME	WME	MSE	WMSE	MSE	WMSE
boston-bike	0.029	0.024	-0.039	-0.055	0.025	0.010	17.48	16.38
collegemsg	0.020	0.018	-0.034	-0.068	0.029	0.021	6.11	16.27
darpa-ip	0.023	0.005	-0.088	-0.030	0.012	0.090	12.25	64.13
digg	0.012	0.022	-0.016	-0.043	0.021	0.032	6.18	11.49
email-eu	0.032	0.031	-0.038	-0.094	0.020	0.016	27.77	34.28
mooc	0.021	0.021	-0.050	-0.059	0.043	0.043	1.68	2.68
nyc-taxi	0.021	0.006	-0.025	-0.015	0.017	0.0003	39.94	16.60
reddit	0.018	0.025	0.002	-0.004	0.013	0.014	17.15	29.32
slashdot	0.008	0.014	-0.015	-0.028	0.013	0.021	4.85	10.57

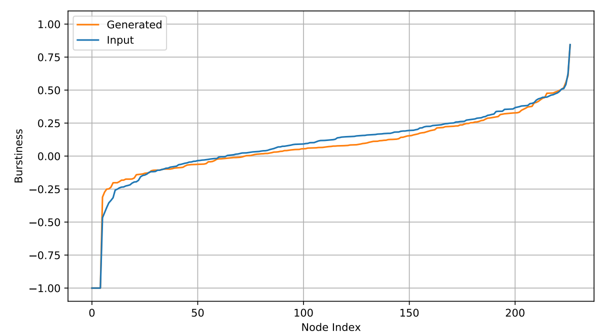


Figure 1: B_i values in input and generated graphs for all nodes in a `nyc-taxi` instance, in non-decreasing order.

5.4 Comparing Burstiness, Spread, and Persistence

Figure 1 shows the comparison of B_v's for each node of an input and the corresponding generated graph for `nyc-taxi`. The values in Table 2 and the Figure 1 show that our generator closely matches the burstiness of individual nodes. Table 2 also shows MSE and Weighted MSE for Burstiness, Spread, and Persistence averaged over the nodes in each graph instance. Our results show that our algorithm, on average, can closely match the temporal metrics for individual nodes of an input graph. The ME analysis on Burstiness suggests, on average, it tends to decrease the Burstiness by less than 0.1.

Since the Spread and Persistence are not normalized metrics, we cannot safely say how much of an error is a significant deviation. However, for all MSE and WMSE metrics, smaller is better. Table 2 also contains the highest deviations for `darpa` and `email` datasets.

Table 3 summarizes the graph-scale values for the same metrics. As shown, generated graphs, compared to the original graphs cannot always match the graph-scale metrics. Especially, `darpa` and `email` graphs have the highest variations between the generated and original graphs. This difference reflects on the other metrics as well. Since the burstiness is not well-matched, the dead time average also deviates, thus, the number of interactions in the given time limit is smaller. Our results show that our generator typically decreases the graph-scale burstiness. Other metrics are usually more closely matched.

Table 3: Temporal Metrics at Graph scale.

Graph	max(# Interactions)	$max(B)$	$max(S)$	$max(P)$
boston-bike	8312	0.699	1.886	30.055
gen-boston	7618	0.428	1.941	36.130
collegemsg	59835	0.739	1.804	26.098
gen-collegemsg	50053	0.431	1.929	36.398
darpa-ip	4554344	0.991	1.552	13.135
gen-darpa	3031864	0.685	1.787	26.412
digg	276831	0.680	1.944	38.956
gen-digg	238432	0.449	1.947	39.128
email-eu	332334	0.991	1.521	10.163
gen-email	278384	0.138	1.958	44.823
mooc	109200	0.878	1.880	32.413
gen-mooc	101095	0.849	1.943	38.211
nyc-taxi	286277	0.410	1.907	35.168
gen-nyc	280324	0.214	1.928	37.156
reddit	24091	0.153	1.936	36.538
gen-reddit	20008	0.191	1.937	36.439
slashdot	26131	0.687	1.840	28.291
gen-slashdot	25264	0.472	1.894	32.621

6 CONCLUSION

We propose a novel temporal interaction graph generator that specifically focuses on the continuous-time aspect of interactions generation. We present a number of temporal metrics and evaluate the quality of our generator at both the node and the graph scale. We demonstrate the generated replicas of our input datasets closely resemble the input graphs in terms of various static and temporal metrics.

It will be a valuable experience to test the effectiveness of this generator in well-known use case scenarios such as anomaly detection, streaming community detection, or temporal motif/path count problems as a future work.

REFERENCES

[1] Rodrigo Augusto da Silva Alves, Renato Martins Assuncao, and Pedro Olmo Stancioli Vaz de Melo. 2016. Burstiness scale: A parsimonious model for characterizing random series of events. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1405–1414.

[2] Caleb Belth, Xinyi Zheng, and Danai Koutra. 2020. Mining Persistent Activity in Continually Evolving Networks. *arXiv preprint arXiv:2006.15410* (2020).

[3] Ewan R Colman and Danica Vukadinović Greetham. 2015. Memory and burstiness in dynamic networks. *Physical Review E* 92, 1 (2015), 012817.

[4] Laetitia Gauvin, Mathieu Génois, Márton Karsai, Mikko Kivelä, Taro Takaguchi, Eugenio Valdano, and Christian L. Vestergaard. 2020. Randomized reference models for temporal networks. arXiv:physics.soc-ph/1806.04032

[5] K-I Goh and A-L Barabási. 2008. Burstiness and memory in complex systems. *EPL (Europhysics Letters)* 81, 4 (2008), 48002.

[6] Vicenç Gómez, Andreas Kaltenbrunner, and Vicente López. 2008. Statistical analysis of the social network and discussion threads in slashdot. In *Proceedings of the 17th international conference on World Wide Web*. 645–654.

[7] Tad Hogg and Kristina Lerman. 2012. Social dynamics of digg. *EPJ Data Science* 1, 1 (2012), 5.

[8] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. 1983. Stochastic blockmodels: First steps. *Social networks* 5, 2 (1983), 109–137.

[9] Petter Holme. 2015. Modern temporal network theory: a colloquium. *The European Physical Journal B* 88, 9 (2015), 1–30.

[10] Tamara G. Kolda, Ali. Pınar, Todd. Plantenga, and C. Seshadhri. 2014. A Scalable Generative Graph Model with Community Structure. *SIAM Journal on Scientific Computing* 36, 5 (2014), C424–C452. https://doi.org/10.1137/130914218 arXiv:https://doi.org/10.1137/130914218

[11] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1269–1278.

[12] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion)*. Association for Computing Machinery, New York, NY, USA, 1343–1350. https://doi.org/10.1145/2487788.2488173

[13] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical review E* 78, 4 (2008), 046110.

[14] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research* 11, Feb (2010), 985–1042.

[15] Ulrich Meyer and Manuel Penschuck. 2016. Generating massive scale-free networks under resource constraints. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 39–52.

[16] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. 969–976.

[17] M. Yusuf Özkaya, M. Fatih Balın, Ali Pınar, and Ümit V. Çatalyürek. 2020. A scalable graph generation algorithm to sample over a given shell distribution. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 227–236. https://doi.org/10.1109/IPDPSW50202.2020.00051

[18] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60, 5 (2009), 911–932.

[19] Leto Peel and Aaron Clauset. 2015. Detecting Change Points in the Large-Scale Structure of Evolving Networks. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 2914–2920.

[20] Tiago P. Peixoto. 2014. Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models. *Phys. Rev. E* 89 (Jan 2014), 012804. Issue 1. https://doi.org/10.1103/PhysRevE.89.012804

[21] Tiago P Peixoto and Martin Rosvall. 2017. Modelling sequences and temporal networks with dynamic community structures. *Nature communications* 8, 1 (2017), 1–12.

[22] Luis EC Rocha, Fredrik Liljeros, and Petter Holme. 2011. Simulated epidemics in an empirical spatiotemporal network of 50,185 sexual contacts. *PLoS Comput Biol* 7, 3 (2011), e1001109.

[23] Peter Sanders and Christian Schulz. 2016. Scalable generation of scale-free graphs. *Inform. Process. Lett.* 116, 7 (2016), 489–491.

[24] G. M. Slota, J. Berry, S. D. Hammond, S. Olivier, C. Phillips, and S. Rajamanickam. 2019. Scalable Generation of Graphs for Benchmarking HPC Community-Detection Algorithms. In *SC*. 1–14.

[25] Sucheta Soundarajan, Acar Tamersoy, Elias B Khalil, Tina Eliassi-Rad, Duen Horng Chau, Brian Gallagher, and Kevin Roundy. 2016. Generating graph snapshots from streaming edge data. In *Proceedings of the 25th International Conference Companion on World Wide Web*. 109–110.

[26] Isabelle Stanton and Ali Pınar. 2012. Constructing and sampling graphs with a prescribed joint degree distribution. *J. Exp. Algorithmics* 17 (2012), 3.1–3.25. https://doi.org/10.1145/2133803.2330086

[27] Christian L. Staudt, Michael Hamann, Alexander Gutfraind, Ilya Safro, and Henning Meyerhenke. 2017. Generating realistic scaled complex networks. *Applied Network Science* 2, 1 (13 Oct 2017), 36. https://doi.org/10.1007/s41109-017-0054-z

[28] Taro Takaguchi, Naoki Masuda, and Petter Holme. 2013. Bursty communication patterns facilitate spreading in a threshold-based epidemic dynamics. *PloS one* 8, 7 (2013), e68629.

[29] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*.

[30] James D Wilson, Nathaniel T Stevens, and William H Woodall. 2019. Modeling and detecting change in temporal networks via the degree corrected stochastic block model. *Quality and Reliability Engineering International* 35, 5 (2019), 1363–1378.

A PROPERTIES OF DATASETS

The `nyc-taxi`[1] data is the NYC Taxi trips of January 2019 and the `boston-bike`[2] dataset is the entries in the Bluebikes system of Boston for April 2019. `digg` is the trace of user/story rating interactions whereas `mooc` is the interactions between students and a set of actions they take in a MOOC platform. `reddit` dataset contains the timestamped references between subreddits. `darpa-ip` is an IP to IP interaction network, where there are normal and malicious traffic present. `slashdot` dataset consists of replies of users in Slashdot website. `collegemsg` is comprised of private messages sent on a social network at the University of California, Irvine. Finally, `email-eu` dataset contains the email transactions between employees of a European research institution.

The columns of Table A1 show the maximum numbers encountered among the chunks of a graph. For example, maximum of $|V|$ means the maximum number of nodes encountered in any of the chunks. One instance (chunk) from `nyc-taxi` graph may have 255 unique nodes in it while another has 260. We report 260 in this case.

Our analysis showed the datasets selected are diverse in terms of interaction frequency and distributions. Some datasets show a tendency towards power-law distribution of interactions among nodes. Some of them have very steep drops while some are very heavy-tailed. In addition, the ratio of $\#interactions$ to $|unique(E)|$ in Table A1 shows the selected datasets cover a wide range of interaction repetition patterns over the edges.

The minimum, average and maximum burstiness columns of Table A1 show that many of the graphs usually have a stable graph-scale burstiness. `boston-bike` is one exception with a range of $[0.393, 0.699]$. This means that even in a graph from the same context, two different time windows (e.g., two different days) may have significantly different interaction patterns.

The difference between the columns $|V|$ and $|Sources|$ shows that in some datasets, some nodes do not initiate any interactions and the subset of nodes that initiate an interaction can go as low as one third of the graph.

Another important takeaway from the datasets presented is that none of the graphs have negative burstiness. Inspecting the burstiness over a larger set of temporal interaction graphs may be a novel analysis approach in such datasets.

B COMPARISON OF A SAMPLE INPUT AND GENERATED GRAPH

We visually compare the log of number of interactions ($log(C_v)$, i.e., *frequency* in [2]) - Burstiness (B_v) relation of all nodes in a sample original graph and the generated copy. Following Belth et al.[2]'s analysis of streaming graphs, we present the Frequency vs Persistence and Frequency vs Spread scatter plots. Figure A1 and A2 show Burstiness (B), Spread (S), and Persistence (P) vs the number of interactions(C) (defined as Frequency (F) in [2]) for an arbitrary input instance from `nyc-taxi` data and one generated graph. The figures are color- and shape-coded according to the types derived during the MM clustering step for visual clarity.

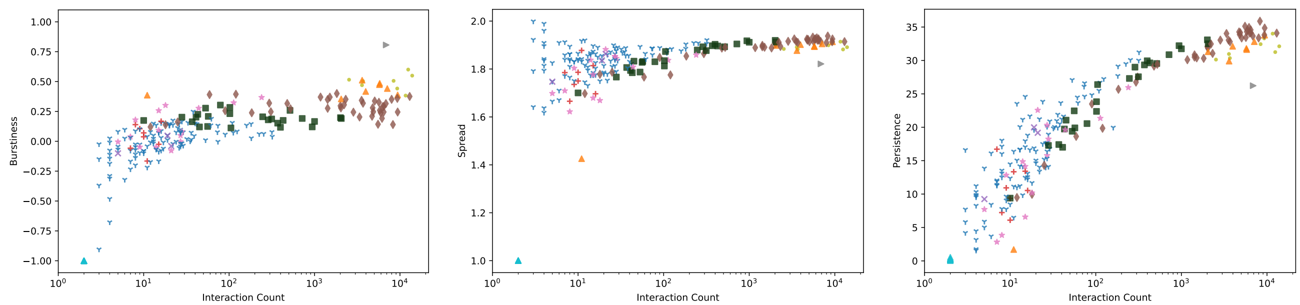
Figures show that the clustering step divided nodes into types of relatively balanced sizes. Our algorithm created a similar temporal graph to the input (with the noise factor of up to 15%). Within each type and overall, figures present similar trends and locations within the plots. It also shows that the clustering step does not put nodes which are vastly different from each other in terms of these metrics in the same type. Our experiments showed that the biggest burstiness variations usually happen for the nodes with less than 10 interactions. This is because there is much less data points and the fact that the law of large numbers do not apply for the random sampling of such small sizes.

We can also see that Persistence of the nodes increase as the number of interactions increase. This is expected since Frequency is a multiplicative factor in the definition of Persistence. And, the effect of noise factor, is also clearly visible in the Persistence plot of the generated graph as a wider spread in vertical axis.

[1](https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page)
[2](https://data.boston.gov/dataset/blue-bikes-system-data)

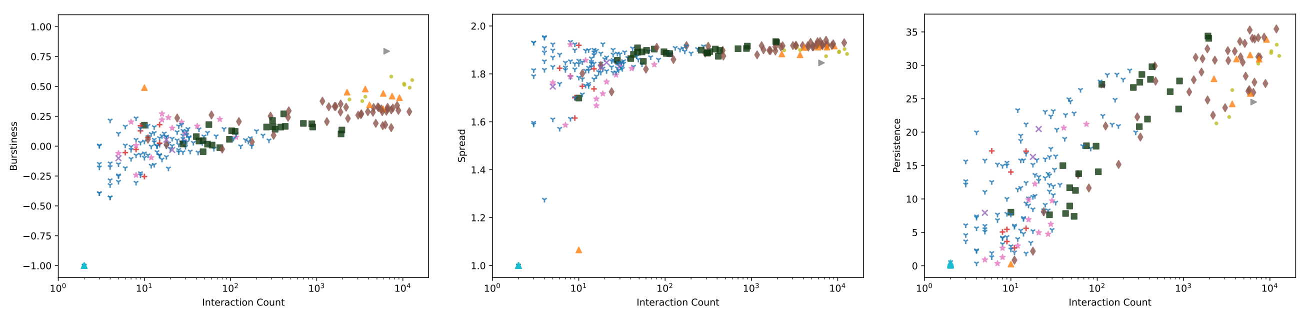
Table A1: Datasets and preliminary details of the instances in the dataset.

Graph	$\lvert V\rvert$	$\lvert Sources\rvert$	$\lvert unique(E)\rvert$	$\#interactions$	$min(B)$	$avg(B)$	$max(B)$	S	P	Chunks (Size)
`boston-bike`	263	263	5358	8312	0.393	0.582	0.699	1.854	27.575	30 (1 day)
`collegemsg [18]`	1899	1350	20296	59835	0.739	0.739	0.739	1.804	26.098	1 (whole data)
`darpa-ip [2]`	25525	9484	68910	4554344	0.991	0.991	0.991	1.552	13.135	1 (whole data)
`digg [7]`	49936	49139	276067	276831	0.111	0.316	0.680	1.944	38.956	2 (60 days)
`email-eu [2]`	986	824	24929	332334	0.991	0.991	0.991	1.521	10.163	1 (whole data)
`mooc [11]`	4263	4255	54048	109200	0.664	0.731	0.878	1.880	32.410	4 (1 week)
`nyc-taxi`	260	245	11114	286277	0.250	0.335	0.410	1.892	33.920	29 (1 day)
`reddit [2]`	9081	6813	18382	24091	0.128	0.142	0.153	1.936	36.538	3 (30 days)
`slashdot [6, 12]`	12623	12560	24431	26131	0.606	0.632	0.687	1.840	28.291	3 (60 days)



(a) Burstiness (b) Spread (c) Persistence

Figure A1: The log of number of interactions (C) vs Burstiness (B), Spread (S), and Persistence (P) scatter plots for nodes of input `nyc-taxi` graph, color- and shape-coded by the types (16 types).



(a) Burstiness (b) Spread (c) Persistence

Figure A2: The log of number of interactions (C) vs Burstiness (B), Spread (S), and Persistence (P) scatter plots for nodes of generated replica of `nyc-taxi` graph, color- and shape-coded by the types (16 types).