On Summarizing Large-Scale Dynamic Graphs

Neil Shah*, Danai Koutra†, Lisa Jin‡, Tianmin Zou§, Brian Gallagher¶, Christos Faloutsos*

* Carnegie Mellon University, † University of Michigan, ‡ University of Rochester
§ Google, ¶ Lawrence Livermore National Lab

Abstract

How can we describe a large, dynamic graph over time? Is it random? If not, what are the most apparent deviations from randomness – a dense block of actors that persists over time, or perhaps a star with many satellite nodes that appears with some fixed periodicity? In practice, these deviations indicate patterns – for example, research collaborations forming and fading away over the years. Which patterns exist in real-world dynamic graphs, and how can we find and rank their importance? These are exactly the problems we focus on. Our main contributions are (a) formulation*: we show how to formalize this problem as minimizing an information theoretic encoding cost, (b)* algorithm*: we propose* TIMECRUNCH*, an effective and scalable method for finding coherent, temporal patterns in dynamic graphs and (c)* practicality*: we apply our method to several large, diverse real-world datasets with up to* 36 million *edges and introduce our auxiliary* ECOVIZ *framework for visualizing and interacting with dynamic graphs which have been summarized by* TIMECRUNCH*. We show that* TIMECRUNCH *is able to compress these graphs by summarizing important temporal structures and finds patterns that agree with intuition.*

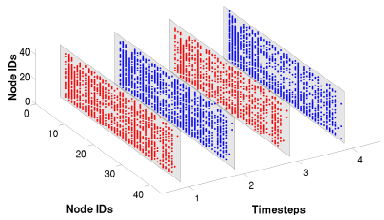
1 Introduction

Given a large phonecall network over time, how can we describe it to a practitioner with just a few phrases? Other than the traditional assumptions about real-world graphs involving degree skewness, what can we say about the connectivity? For example, is the dynamic graph characterized by many large cliques which appear at fixed intervals of time, or perhaps by several large stars with dominant hubs that persist throughout? Our work aims to answer these questions, and specifically, we focus on constructing concise summaries of large, real-world dynamic graphs in order to better understand their underlying behavior.

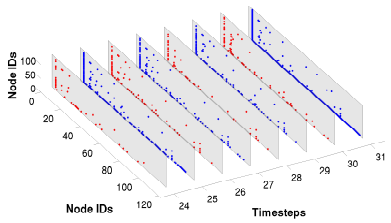
This problem has numerous practical applications. Dynamic graphs are ubiquitously used to model the relationships between various entities over *time*, which is a valuable feature in almost all applications in which nodes represent users or people. Examples include online social networks, phone-call networks, collaboration and coauthorship networks and other interaction networks.

Though numerous graph algorithms suitable for static contexts such as modularity, spectral and cut-based partitioning exist, they do not offer direct dynamic counterparts. Furthermore, the traditional goals of clustering and community detection tasks are not quite aligned with our goal. These algorithms typically produce groupings

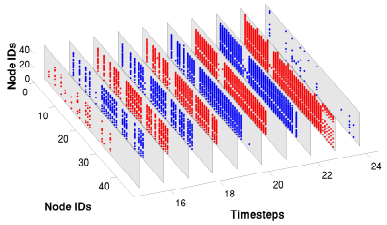
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering



(a) 40 users of Yahoo! Messenger forming a *constant near clique* with unusually high 55% density, over 4 weeks in April 2008.



(b) 111 callers in a large phonecall network, forming a *periodic star*, over the last week of December 2007 – note the heavy activity on holidays.



(c) 43 collaborating biotechnology authors forming a *ranged near clique* in the DBLP network, jointly publishing through 2005-2012.

Figure 1: **TIMECRUNCH finds coherent, interpretable temporal structures**. We show the reordered subgraph adjacency matrices, over the timesteps of interest, each outlined in gray; edges are plotted in alternating red and blue, for discernibility.

of nodes which satisfy or approximate some optimization function. However, they do not offer interpretation or characterization of the outputs.

In this work, we propose TIMECRUNCH, an approach for concisely summarizing large, dynamic graphs which extend beyond traditional dense, isolated "cavemen" communities. Our method works by leveraging MDL (Minimum Description Length) in order to represent graphs over time using a lexicon of *temporal phrases* which describe temporal connectivity behavior. Figure 1 shows several interesting results found from applying TIMECRUNCH to real-world dynamic graphs.

- Figure 1a shows a *constant near-clique* of likely bots on Yahoo! Messenger.
- Figure 1b depicts a *periodic star* of possible telemarketers on a phonecall network.
- Lastly, Figure 1c shows a *ranged near clique* of many authors jointly publishing in a biology journal.

In this work, we seek to answer the following informally posed problem:

Problem 1 (Informal): Given a dynamic graph, **find** a set of possibly overlapping temporal subgraphs to **concisely describe** the given dynamic graph in a **scalable** fashion.

Our main contributions are as follows:

1. **Formulation:** We define the problem of dynamic graph understanding in in a compression context.
2. **Algorithm:** We develop TIMECRUNCH, a fast algorithm for dynamic graph summarization.
3. **Practicality:** We show quantitative and qualitative results of TIMECRUNCH on several real graphs, and also discuss ECOVIZ for interactive dynamic graph visualization.

Reproducibility: Our code for TIMECRUNCH is open-sourced at `www.cs.cmu.edu/~neilshah/code/timecrunch.tar`.

2 Related Work

The related work falls into three main categories: static graph mining, temporal graph mining, and graph compression and summarization.

Static Graph Mining. Most works find specific, tightly-knit structures, such as (near-) cliques and bipartite cores: eigendecomposition [24], cross-associations [7], modularity-based optimization methods [20, 5]. Dhillon et al. [10] propose information theoretic co-clustering based on mutual information optimization. However, these approaches have limited structural vocabularies. [14, 17] propose cut-based partitioning, whereas [3] suggests spectral partitioning using multiple eigenvectors – these schemes seek hard clustering of all nodes as opposed

to identifying communities, and require parameters. Subdue [8] and other fast frequent-subgraph mining algorithms [12] operate on labeled graphs. Our work involves unlabeled graphs and lossless compression.

Temporal Graph Mining. Most work on temporal graphs focuses on the evolution of specific properties, change detection, or community detection. For example, [2] aims at change detection in streaming graphs using projected clustering. This approach focuses on anomaly detection rather than mining temporal patterns. GraphScope [27] uses graph search for hard-partitioning of temporal graphs to find dense temporal cliques and bipartite cores. Com2 [4] uses CP/PARAFAC decomposition with MDL for the same. [11] uses incremental cross-association for change detection in dense blocks, whereas [22] proposes an algorithm for mining atemporal cross-graph quasi-cliques. These approaches have limited vocabularies and no temporal interpretability. Dynamic clustering [29] finds stable clusters over time by penalizing deviations from incremental static clustering. Our work focuses on interpretable structures, which may not appear at every timestep.

Graph Compression and Summarization. Work on summarization and compression of time-evolving graphs is quite limited [30]. Some examples for compressing *static* graphs include SlashBurn [13], which is a recursive node-reordering approach to leverage run-length encoding, weighted graph compression [28] that uses structural equivalence to collapse nodes/edges to simplify graph representation, two-part MDL representation with error bounds [31], and domain-specific summarization using graph invariants [32]. VoG [16] uses MDL to label subgraphs in terms of a vocabulary on static graphs, consisting of stars, (near) cliques, (near) bipartite cores and chains. This approach only applies to static graphs and does not offer a clear extension to dynamic graphs. Our work proposes a suitable lexicon for dynamic graphs, uses MDL to label *temporally coherent* subgraphs and proposes an effective and scalable algorithm for finding them. More recent works on time-evolving networks include graph stream summarization [34] for query efficiency, and influence-based graph summarization [35, 33], which aim to summarize network propagation processes.

3 Problem Formulation

In this section, we give the first main contribution of our work: formulation of dynamic graph summarization as a compression problem.

The Minimum Description Length (MDL) principle aims to be a practical version of Kolmogorov Complexity [19], often associated with the motto *Induction by Compression*. MDL states that given a model family $\mathcal{M}$, the best model $M \in \mathcal{M}$ for some observed data $\mathcal{D}$ is that which minimizes $L(M) + L(\mathcal{D}|M)$, where $L(M)$ is the length in bits used to describe M and $L(\mathcal{D}|M)$ is the length in bits used to describe $\mathcal{D}$ encoded using M. MDL enforces lossless compression for fairness in the model selection process.

We focus on analysis of undirected dynamic graphs using fixed-length, discretized time intervals. We consider a dynamic graph $G(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, $m = |\mathcal{E}|$ edges and t timesteps, without self-loops. Here, $G = \cup_x G_x(\mathcal{V}, \mathcal{E}_x)$, where G_x and E_x correspond to the graph and edge-set for the x^{th} timestep.

For our summary, we consider the set of temporal phrases $\Phi = \Delta \times \Omega$, where Δ corresponds to the set of temporal signatures, Ω corresponds to the set of static structure identifiers and $\times$ denotes Cartesian set product. Though we can include arbitrary temporal signatures and static structure identifiers into these sets depending on the types of temporal subgraphs we expect to find in a given dynamic graph, we choose 5 temporal signatures which we anticipate to find in real-world dynamic graphs: oneshot (*o*), ranged (*r*), periodic (*p*), flickering (*f*) and constant (*c*):

- *Oneshot* structures appear at only one timestep
- *Ranged* structures appear for a series of consecutive timesteps
- *Periodic* structures appear at fixed intervals of time
- *Flickering* structures do not have discernible periodicity, but occur multiple times
- *Constant* structures appear at every timestep

and 6 very common structures found in real-world static graphs [15, 24, 16] – stars (*st*), *full* and *near* cliques

(*fc*, *nc*), *full* and *near* bipartite cores (*bc*, *nb*) and chains (*ch*):

- *Stars* are characteristic of a hub node connected to 2 or more "spokes"
- *(Near) Cliques* are sets of nodes with very dense interconnectivity
- *(Near) Bipartite Cores* consist of non-intersecting node sets L and R for which there exist only edges between L and R but not within
- *Chains* are a series of nodes in which each node is connected to the next

Summarily, we have the signatures $\Delta = \{o, r, p, f, c\}$, static identifiers $\Omega = \{st, fc, nc, bc, nb, ch\}$ and temporal phrases $\Phi = \Delta \times \Omega$.

In order to use MDL for dynamic graph summarization using these temporal phrases, we next define the model family $\mathcal{M}$, the means by which a model $M \in \mathcal{M}$ describes our dynamic graph and how to quantify the cost of encoding in terms of bits.

3.1 Using MDL for Dynamic Graph Summarization

We consider models $M \in \mathcal{M}$ to be composed of ordered lists of temporal graph structures with node, but not edge overlaps. Each $s \in M$ describes a certain region of the adjacency tensor $\mathbf{A}$ in terms of the interconnectivity of its nodes – note that nonzero $\mathbf{A}_{i,j,k}$ indicates edge (i, j) exists in timestep k .

Our model family $\mathcal{M}$ consists of all possible permutations of subsets of $\mathcal{C}$, where $\mathcal{C} = \cup_v \mathcal{C}_v$ and $\mathcal{C}_v$ denotes the set of all possible temporal structures of phrase $v \in \Phi$ over all possible combinations of timesteps. That is, $\mathcal{M}$ consists of all possible models M, which are ordered lists of temporal phrases $v \in \Phi$ such as flickering stars (*fst*), periodic full cliques (*pfc*), etc. over all possible subsets of $\mathcal{V}$ and $G_1 \cdots G_t$. Through MDL, we seek $M \in \mathcal{M}$ which best mediates between the length of M and the adjacency tensor $\mathbf{A}$ given M.

Our high-level approach for transmitting the adjacency tensor $\mathcal{A}$ via the model M is described as follows: First, we transmit M. Next, given M, we induce the approximation of the adjacency tensor $\mathbf{M}$ as described by each temporal structure $s \in M$ – for each structure s, we induce the edges described by s in $\mathbf{M}$ accordingly. Given that $\mathbf{M}$ is a summary approximation to $\mathbf{A}$, $\mathbf{M} \neq \mathbf{A}$ most likely. Since MDL requires lossless encoding, we must also transmit the error $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$, obtained by taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$. Given M and $\mathbf{E}$, a recipient can construct the full adjacency tensor $\mathbf{A}$ in a lossless fashion.

Thus, we formalize the problem we tackle as follows:

Problem 2 (Minimum Dynamic Graph Description): Given a dynamic graph G with adjacency tensor $\mathbf{A}$ and temporal phrase lexicon Φ, find the smallest model M which minimizes the total encoding length

$$L(G, M) = L(M) + L(\mathbf{E})$$

where $\mathbf{E}$ is the error matrix computed by $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ and $\mathbf{M}$ is the approximation of $\mathbf{A}$ induced by M.

3.2 Encoding the Model and Errors

To fully describe a model $M \in \mathcal{M}$, we have the following:

$$L(M) = L_{\mathbb{N}}(|M| + 1) + log_2 \binom{|M| + |\Phi| - 1}{|\Phi - 1|} + \sum_{s \in M} (-log_2 P(v(s)|M) + L(c(s)) + L(u(s)))$$

We begin by transmitting the total number of temporal structures in M using $L_{\mathbb{N}}$, Rissanen's optimal encoding for integers greater than or equal to 1 [23]. Next, we optimally encode the number of temporal structures for each phrase $v \in \Phi$ in M. Then, for each structure s, we encode the type $v(s)$ for each structure $s \in M$ using optimal prefix codes [9], the connectivity $c(s)$ and the temporal presence of the s, consisting of the ordered list of timesteps $u(s)$ in which s appears.

In order to have a coherent model encoding scheme, we must define the encoding for each phrase $v \in \Phi$ such that we can compute $L(c(s))$ and $L(u(s))$ for all structures in M. The connectivity $c(s)$ corresponds to the edges which are induced by s, whereas the temporal presence $u(s)$ corresponds to the timesteps in which s is present. We consider the connectivity and temporal presence separately, as the encoding for a temporal structure s described by a phrase v is the sum of encoding costs for the connectivity of the corresponding static structure identifier in Ω and its temporal presence as indicated by a temporal signature in Δ. Due to space constraints, we refer the interested reader to more detailed manuscripts [16, 25] for details regarding encoding processes and costs for the connectivity $L(c(s))$, temporal presence $L(u(s))$ and associated errors. In a nutshell, we have different encoding costs for encoding any subgraph and temporal recurrence pattern using a particular phrase in our lexicon Φ.

Remark: For a dynamic graph G of n nodes, the search space $\mathcal{M}$ for the best model $M \in \mathcal{M}$ is intractable, as it consists of all permutations of all possible temporal structures over the lexicon Φ, over all possible subsets over the node-set $\mathcal{V}$ and over all possible graph timesteps $G_1 \cdots G_t$. Furthermore, $\mathcal{M}$ is not easily exploitable for efficient search. As a result, we propose several practical approaches for the purpose of finding good and interpretable temporal models/summaries for G.

4 Proposed Method: TIMECRUNCH

Thus far, we have described our strategy of formulating dynamic graph summarization as a problem in a compression context for which we can leverage MDL. Specifically, we have described how to encode a model and the associated error which can be used to losslessly reconstruct the original dynamic graph G. Our models are characterized by ordered lists of temporal structures which are further classified as *phrases* from the lexicon Φ – that is, each $s \in M$ is identified by a phrase $p \in \Phi$ – over the node connectivity $c(s)$ (an induced set of edges depending on the static structure identifier st, fc, etc.) and the associated temporal presence $u(s)$ (ordered list of timesteps captured by a temporal signature o, r, etc. and deviations) in which the temporal structure is active, while the error consists of those edges which are not covered by $\mathbf{M}$, or the approximation of $\mathbf{A}$ induced by M.

Next, we discuss how we find good candidate temporal structures to populate the candidate set $\mathcal{C}$, as well as how we find the best model M with which to summarize our dynamic graph. The pseudocode for our algorithm is given in Alg. 1 and the next subsections detail each step of our approach.

4.1 Generating Candidate Static Structures

TIMECRUNCH takes an incremental approach to dynamic graph summarization. Our approach begins by considering potentially useful subgraphs over static graphs $G_1 \cdots G_t$. Section 2 mentions several such algorithms for community detection and clustering including EigenSpokes, METIS, SlashBurn, etc. Summarily, for each $G_1 \cdots G_t$, a set of subgraphs $\mathcal{F}$ is produced.

Algorithm 1 TIMECRUNCH

1: **Generating Candidate Static Structures**: Generate static subgraphs for each $G_1 \cdots G_t$ using traditional static graph decomposition approaches.
2: **Labeling Candidate Static Structures**: Label each static subgraph as a static structure corresponding to the identifier $x \in \Omega$ which minimizes the *local encoding cost*.
3: **Stitching Candidate Temporal Structures**: *Stitch* static structures from $G_1 \cdots G_t$ together to form temporal structures with coherent connectivity and label them according to the phrase $p \in \Phi$ which minimizes temporal presence encoding cost. Populate the candidate set $\mathcal{C}$.
4: **Composing the Summary**: Compose a model M of important, non-redundant temporal structures which summarize G using the VANILLA, TOP-10, TOP-100 and STEPWISE heuristics. Choose M associated with the heuristic that produces the smallest total encoding cost.

4.2 Labeling Candidate Static Structures

Once we have the set of static subgraphs from $G_1 \cdots G_t$, $\mathcal{F}$, we next seek to label each subgraph in $\mathcal{F}$ according to the static structure identifiers in Ω that best fit the connectivity for the given subgraph. That is, for each subgraph construed as a set of nodes $\mathcal{L} \in \mathcal{V}$ for a fixed timestep, does the adjacency matrix of $\mathcal{L}$ best resemble a star, near or full clique, near or full bipartite core or a chain? To answer this question, we try encoding the subgraph $\mathcal{L}$ using each of the static identifiers in Ω and label it with the identifier $x \in \Omega$ which minimizes the encoding cost.

Consider the model ω which consists of only the subgraph $\mathcal{L}$ and a yet to be determined static identifier. In practice, instead of computing the global encoding cost $L(G, \omega)$ when encoding $\mathcal{L}$ as each static identifier in Ω to find the best fit, we compute the *local* encoding cost defined as $L(\omega) + L(\mathbf{E}_\omega^+) + L(\mathbf{E}_\omega^-)$ where $L(\mathbf{E}_\omega^+)$ and $L(\mathbf{E}_\omega^-)$ indicate the encoding costs for the extraneous and unmodeled edges for the subgraph $\mathcal{L}$ respectively. This is done for purpose of efficiency – intuitively, however, the static identifier that best describes $\mathcal{L}$ is independent of the edges outside of $\mathcal{L}$.

The challenge in this labeling step is that before we can encode $\mathcal{L}$ as any type of identifier, we must identify a suitable permutation of nodes in the subgraph so that our model encodes the correct edges. For example, if $\mathcal{L}$ is a star, which is the hub? Or if $\mathcal{L}$ is a bipartite core, how can we distinguish the parts? We resort to heuristics, as some of these tasks are computationally difficult to perform exactly – for example, finding the best configuration of nodes to form a bipartite core is equivalent to finding the maximum cut which is NP-hard. Details of appropriate configurations for each static structure are given in [16] for space constraints.

4.3 Stitching Candidate Temporal Structures

Thus far, we have a set of static subgraphs $\mathcal{F}$ over $G_1 \cdots G_t$ labeled with the associated static identifiers which best represent subgraph connectivity (from now on, we refer to $\mathcal{F}$ as a set of static *structures* instead of *subgraphs* as they have been labeled with identifiers). From this set, our goal is to find meaningful temporal structures – namely, we seek to *find* static subgraphs which have the same patterns of connectivity over one or more timesteps and *stitch* them together. Thus, we formulate the problem of finding coherent temporal structures in G as a clustering problem over $\mathcal{F}$. Though there are several criteria we could use for clustering static structures together, we employ the following based on their intuitive meaning: two structures in the same cluster should have (a) substantial overlap in the node-sets composing their respective subgraphs, and (b) exactly the same, or similar (full and near clique, or full and near bipartite core) static structure identifiers. These criteria, if satisfied, allow us to find groups of nodes that share interesting connectivity patterns over time.

Conducting the clustering by naively comparing each static structure in $\mathcal{F}$ to the others will produce the desired result, but is *quadratic* on the number of static structures and is thus undesirable from a scalability point of view. Instead, we propose an incremental approach using repeated rank-1 Singular Value Decomposition (SVD) for clustering the static structures, which offers *linear* time complexity on the number of edges m in G.

We begin by defining $\mathbf{B}$ as the *structure-node membership matrix* (SNMM) of G. $\mathbf{B}$ is defined to be of dimensions $|\mathcal{F}| \times |\mathcal{V}|$, where $\mathbf{B}_{i,j}$ indicates whether the ith row (structure) in $\mathcal{F}$ ($\mathbf{B}$) contains node j in its node-set. Thus, $\mathbf{B}$ is a matrix indicating the membership of nodes in $\mathcal{V}$ to each of the static structures in $\mathcal{F}$. We note that any two equivalent rows in $\mathbf{B}$ are characterized by structures that share the same node-set (but possibly different static identifiers). As our clustering criteria mandate that we cluster only structures with the same or similar static identifiers, in our algorithm, we construct 4 SNMMs – $\mathbf{B}_{st}$, $\mathbf{B}_{cl}$, $\mathbf{B}_{bc}$ and $\mathbf{B}_{ch}$ corresponding to the associated matrices for stars, near and full cliques, near and full bipartite cores and chains respectively. Now, any two equivalent rows in $\mathbf{B}_{cl}$ are characterized by structures that share the same-node set and the same, or similar static identifiers, and analogue for the other matrices. Next, we utilize SVD to cluster the rows in each SNMM, effectively clustering the structures in $\mathcal{F}$.

Recall that the rank-k SVD of an $m \times n$ matrix $\mathbf{A}$ factorizes $\mathbf{A}$ into 3 matrices – the $m \times k$ matrix of

left-singular vectors $\mathbf{U}$, the $k \times k$ diagonal matrix of singular values $\mathbf{\Sigma}$ and the $n \times k$ matrix of right-singular vectors $\mathbf{V}$, such that $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$. A rank-$k$ SVD effectively reduces the input data into the best k-dimensional representation, each of which can be mined separately for clustering and community detection purposes. However, one major issue with using SVD in this fashion is that identifying the desired number of clusters k upfront is a non-trivial task. To this end, [21] evidences that in cases where the input matrix is sparse, repeatedly clustering using k rank-1 decompositions and adjusting the input matrix accordingly approximates the batch rank-k decomposition. This is a valuable result in our case – as we do not initially know the number of clusters needed to group the structures in $\mathcal{F}$, we eliminate the need to define k altogether by repeatedly applying rank-1 SVD using power iteration and removing the discovered clusters from each SNMM until all clusters have been found (when all SNMMs are fully sparse and thus *deflated*). However, in practice, full deflation is unneeded for summarization purposes, as most "important" clusters are found in early iterations due to the nature of SVD. For each of the SNMMs, the matrix $\mathbf{B}$ used in the $(i+1)^{th}$ iteration of this iterative process is computed as

$$\mathbf{B}^{i+1} = \mathbf{B}^{i} - I^{\mathcal{G}_i} \circ \mathbf{B}^{i}$$

where $\mathcal{G}_i$ denotes the set of row ids corresponding to the structures which were clustered together in iteration i, $I^{\mathcal{G}_i}$ denotes the indicator matrix with 1s in rows specified by $\mathcal{G}_i$ and $\circ$ denotes the Hadamard matrix product. This update to $\mathbf{B}$ is needed between iterations, as without subtracting out the previously-found cluster, repeated rank-1 decompositions would find the same cluster ad infinitum and the algorithm would not converge.

Although this algorithm works assuming we can remove a cluster in each iteration, the question of how we find this cluster given a singular vector has yet to be answered. First, we sort the singular vector, permuting the rows by magnitude of projection. The intuition is that the structure (rows) which projects most strongly to that cluster is the best representation of the cluster, and is considered a *base* structure which we attempt to find matches for. Starting from the base structure, we iterate down the sorted list and compute the Jaccard similarity, defined as $J(\mathcal{L}_1, \mathcal{L}_2) = |\mathcal{L}_1 \cap \mathcal{L}_2| / |\mathcal{L}_1 \cup \mathcal{L}_2|$ for node-sets $\mathcal{L}_1$ and $\mathcal{L}_2$, between each structure and the base. Other structures which are composed of the same, or similar node-sets will also project strongly to the cluster, and be stitched to the base. Once we encounter a series of structures which fail to match by a predefined similarity criterion, we adjust the SNMM and continue with the next iteration.

Having stitched together the relevant static structures, we label each temporal structure using the temporal signature in Δ and resulting phrase in Φ which minimizes its encoding cost. We use these temporal structures to populate the candidate set $\mathcal{C}$ for our model.

4.4 Composing the Summary

Given the candidate set of temporal structures $\mathcal{C}$, we next seek to find the model M which best summarizes G. However, actually finding the best model is combinatorial, as it involves considering all possible permutations of subsets of $\mathcal{C}$ and choosing the one which gives the smallest encoding cost. As a result, we propose several heuristics that give fast and approximate solutions without entertaining the entire search space. To reduce the search space, we associate with each temporal structure a metric by which we measure quality, called the *local encoding benefit*. The local encoding benefit is defined as the ratio between the cost of encoding the given temporal structure as error and the cost of encoding it using the best phrase (local encoding cost). Large local encoding benefits indicate high compressibility, and thus meaningful structure in the underlying data. Our proposed heuristics are as follows:

VANILLA: This is the baseline approach, in which our summary contains all the structures from the candidate set, or $M = \mathcal{C}$.

TOP-K: In this approach, M consists of the top k structures of $\mathcal{C}$, sorted by local encoding benefit.

STEPWISE: This approach involves considering each structure of $\mathcal{C}$, sorted by local encoding benefit, and adding it to M if the global encoding cost decreases. If adding the structure to M increases the global encoding cost,

the structure is discarded as redundant or not worthwhile for summarization purposes.

In practice, TIMECRUNCH uses each of the heuristics and identifies the best summary for G as the one that produces the minimum encoding cost.

5 Experiments

In this section, we evaluate TIMECRUNCH and seek to answer the following questions: Are real-world dynamic graphs well-structured, or noisy and indescribable? If they are structured, how so – what temporal structures do we see in these graphs and what do they mean?

5.1 Datasets and Experimental Setup

For our experiments, we use 5 real dynamic graph datasets – we briefly describe them below.

Enron: The `Enron` e-mail dataset is publicly available [26]. It contains 20K unique links between 151 users based on e-mail correspondence, over 163 weeks (May 1999 - June 2002).

Yahoo! IM: The `Yahoo-IM` dataset is publicly available [36]. It contains 2.1M sender-receiver pairs between 100K users over 5.7K zip-codes selected from the Yahoo! messenger network over 4 weeks starting from April 1st, 2008.

Honeynet: The `Honeynet` dataset is not publicly available. It contains information about network attacks on *honeypots* (i.e., computers which are left intentionally vulnerable). It contains source and destination IPs, and attack timestamps of 372K (attacker and honeypot) machines with 7.1M unique daily attacks over a span of 32 days starting from December 31st, 2013.

DBLP: The `DBLP` computer science bibliography is publicly available, and contains yearly co-authorship information [1]. We used a subset of DBLP spanning 25 years, from 1990 to 2014, with 1.3M authors and 15M unique author-author collaborations over the years.

Phonecall: The `Phonecall` dataset is not publicly available. It describes the who-calls-whom activity of 6.3M individuals from a large, anonymous Asian city and contains a total of 36.3M unique daily phonecalls. It spans 31 days, starting from December 1st, 2007.

In our experiments, we use SlashBurn [13] for generating candidate static structures, as it is scalable and designed to extract structure from real-world, non-"cavemen" graphs. We note that including other graph decomposition methods can be used for various applications instead of SlashBurn. Furthermore, when clustering each sorted singular vector during the stitching process, we move on with the next iteration of matrix deflation after 10 failed matches with a Jaccard similarity threshold of 0.5 – we choose 0.5 based on experimental results which show that it gives the best encoding cost and balances between excessively terse and overlong (error-prone) models. Lastly, we run TIMECRUNCH for a total of 5000 iterations for all graphs (each iteration uniformly selects one SNMM to mine, resulting in 5000 total temporal structures), except for the `Enron` graph which is fully deflated after 563 iterations and the `Phonecall` graph which we limit to 1000 iterations for efficiency.

5.2 Quantitative Analysis

In this section, we use TIMECRUNCH to summarize each of the real-world dynamic graphs discussed above and report the resulting encoding costs. Specifically, evaluation is done by comparing the compression ratio between encoding costs of the resulting models to the null encoding (ORIGINAL) cost, which is obtained by encoding the graph using an empty model.

We note that although we provide results in a compression context, compression is *not* our main goal for TIMECRUNCH, but rather the means to our end for identifying suitable structures with which to summarize

Graph	ORIGINAL (bits)	TIMECRUNCH			
		VANILLA	TOP-10	TOP-100	STEPWISE
Enron	86,102	89% (563)	88%	81%	**78%** (130)
Yahoo-IM	16,173,388	97% (5000)	99%	98%	**93%** (1523)
Honeynet	72,081,235	82% (5000)	96%	89%	**81%** (3740)
DBLP	167,831,004	97% (5000)	99%	99%	**96%** (1627)
Phonecall	478,377,701	100% (1000)	100%	99%	**98%** (370)

Table 1: **TIMECRUNCH finds temporal structures that compress real graphs.** ORIGINAL denotes cost in bits for encoding each graph with an empty model. Other columns show relative costs for encoding using the respective heuristic (size of model in parentheses). The lowest description cost is bolded.

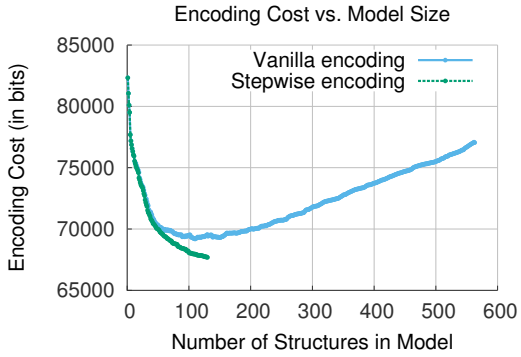


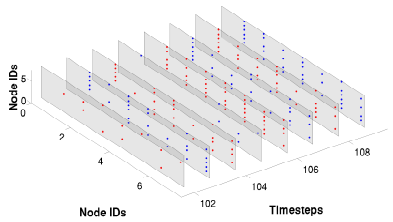
Figure 2: **TIMECRUNCH-STEPWISE summarizes** Enron **using just 78% of ORIGINAL's bits and 130 structures compared to 89% and 563 structures of TIMECRUNCH-VANILLA by pruning unhelpful structures from the candidate set.**

dynamic graphs and route the attention of practitioners. For this reason, we do not evaluate against other, compression-oriented methods which prioritize leveraging any correlation within the data to reduce cost and save bits. Other temporal clustering and community detection approaches which focus only on extracting dense blocks are also not compared to for similar reasons.

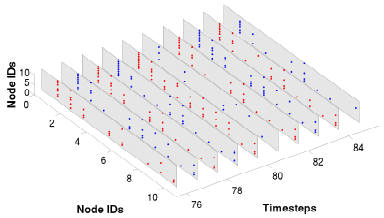
In our evaluation, we consider (a) ORIGINAL and (b) TIMECRUNCH summarization using the proposed heuristics. In the ORIGINAL approach, the entire adjacency tensor is encoded using the empty model $M = \emptyset$. As the empty model does not describe any part of the graph, all the edges are encoded using $L(\mathbf{E}^-)$. We use this as a baseline to evaluate the savings attainable using TIMECRUNCH. For summarization using TIMECRUNCH, we apply the VANILLA, TOP-10, TOP-100 and STEPWISE model selection heuristics. We note that we ignore small structures of < 5 nodes for Enron and < 8 nodes for the other, larger datasets.

Table 1 shows the results of our experiments in terms of encoding costs of various summarization techniques as compared to the ORIGINAL approach. Smaller compression ratios indicate better summaries, with more structure explained by the respective models. For example, STEPWISE was able to encode the Enron dataset using just 78% of the bits compared to 89% using VANILLA. In our experiments, we find that the STEPWISE heuristic produces models with considerably fewer structures than VANILLA, while giving even more concise graph summaries (Fig. 2). This is because it is highly effective in pruning redundant, overlapping or error-prone structures from the candidate set $\mathcal{C}$, by evaluating new structures in the context of previously seen ones.

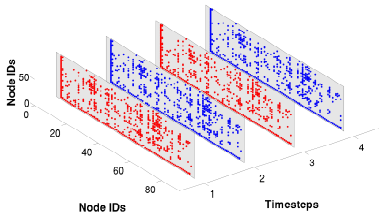
Our results indicate that real-world dynamic graphs are in fact structured, as TIMECRUNCH gives better encoding cost than ORIGINAL.



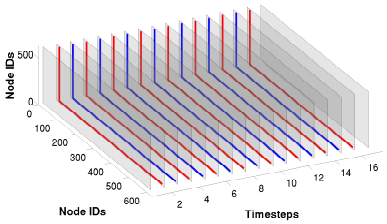
(a) 8 employees of the Enron legal team forming a *flickering near clique*



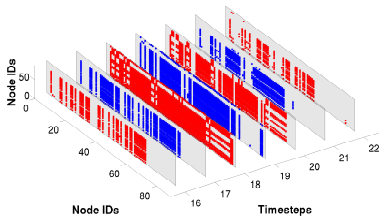
(b) 10 employees of the Enron legal team forming a *flickering star* with the boss as the hub



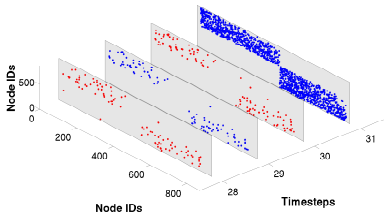
(c) 82 users in Yahoo-IM forming a *constant star* over the observed 4 weeks



(d) 589 honeypot machines were attacked on Honeynet over 2 weeks, forming a *ranged star*



(e) 82 authors forming a *ranged near clique* on DBLP, with burgeoning collaboration from timesteps 18-20 (2007-2009)



(f) 792 callers in Phonecall forming a *oneshot near bipartite core* appearing strongly on Dec. 31

Figure 3: **TIMECRUNCH finds meaningful temporal structures in real graphs.** We show the reordered subgraph adjacency matrices over multiple timesteps. Individual timesteps are outlined in gray, and edges are plotted with alternating red and blue color for discernibility.

	st	fc	ch
r	9	-	-
p	93	7	1
f	3	1	-
c	-	-	-
o	15	1	-

(a) Enron

	st	fc	nc	bc	nb	ch
r	147	43	-	1	45	6
p	59	25	-	-	42	3
f	179	55	-	1	62	3
c	185	118	-	-	66	-
o	295	129	1	2	56	-

(b) Yahoo-IM

	st	bc
r	56	-
p	125	1
f	39	-
c	-	-
o	3512	7

(c) Honeynet

	st	fc	nb	ch
r	43	80	-	5
p	19	26	-	-
f	1	-	-	-
c	-	-	-	-
o	516	840	97	-

(d) DBLP

	st	fc	nc	bc
r	15	-	-	-
p	68	-	-	1
f	88	-	-	-
c	5	-	-	-
o	187	4	1	1

(e) Phonecall

Table 2: Frequency of each temporal structure type discovered using TIMECRUNCH-STEPWISE for each dataset.

5.3 Qualitative Analysis

In this section, we discuss qualitative results from applying TIMECRUNCH to the real-world datasets.

Enron: The Enron graph is characteristic of many periodic, ranged and oneshot stars and several periodic and flickering cliques. Periodicity is reflective of office e-mail communications (e.g. meetings, reminders). Figure 3a shows an excerpt from one flickering clique which corresponds to the several members of Enron's legal team, including Tana Jones, Susan Bailey, Marie Heard and Carol Clair – all lawyers at Enron. Figure 3b shows an excerpt from a flickering star, corresponding to many of the same members as the flickering clique – the center of this star was identified as the boss, Tana Jones (Enron's Senior Legal Specialist). Note that the satellites of the star oscillate over time. Interestingly, the flickering star and clique extend over most of the observed duration. Furthermore, several of the oneshot stars corresponds to company-wide emails sent out by key players John Lavorato (Enron America CEO), Sally Beck (COO) and Kenneth Lay (CEO/Chairman).

Yahoo! IM: The Yahoo-IM graph is composed of many temporal stars and cliques of all types, and several smaller bipartite cores with just a few members on one side (indicative of friends who share mostly similar

friend-groups but are themselves unconnected). We observe several interesting patterns in this data – Fig. 3c corresponds to a constant star with a hub that communicates with 70 users consistently over 4 weeks. We suspect that these users are part of a small office network, where the boss uses group messaging to notify employees of important updates or events – we notice that very few edges of the star are missing each week and the average degree of the satellites is roughly 4, corresponding to possible communication between employees. Figure 1a depicts a constant clique between 40 users, with an average density over 55% – we suspect that these may be spam-bots messaging each other in an effort to appear normal.

Honeynet: `Honeynet` is a bipartite graph between attacker and honeypot (victim) machines. As such, it is characterized by temporal stars and bipartite cores. Many of the attacks only span a single day, as indicated by the presence of 3512 oneshot stars, and no attacks span the entire 32 day duration. Interestingly, 2502 of these oneshot star attacks (71%) occur on the first and second observed days (Dec. 31 and Jan. 1st) indicating intentional "new-year" attacks. Figure 3d shows a ranged star, lasting 15 consecutive days and targeting 589 machines for the entire duration of the attack.

DBLP: Agreeing with intuition, `DBLP` consists of a large number of oneshot temporal structures corresponding to many single instances of joint publication. However, we also find numerous ranged/periodic stars and cliques which indicate coauthors publishing in consecutive years or intermittently. Figure 1c shows a ranged clique spanning from 2007-2012 between 43 coauthors who jointly published each year. The authors are mostly members of the NIH NCBI (National Institute of Health National Center for Biotechnology Information) and have published their work in various biotechnology journals such as *Nature*, *Nucleic Acids Research* and *Genome Research*. Figure 3e shows another ranged clique from 2005 to 2011, consisting of 83 coauthors who jointly publish each year, with an especially collaborative 3 years (timesteps 18-20) corresponding to 2007-2009 before returning to status quo.

Phonecall: The `Phonecall` dataset is largely comprised of temporal stars and few dense clique and bipartite structures. Again, we have a large proportion of oneshot stars which occur only at single timesteps. Further analyzing these results, we find that 111 of the 187 oneshot stars (59%) are found on Dec. 24, 25 and 31st, corresponding to Christmas Eve/Day and New Year's Eve holiday greetings. Furthermore, we find many periodic and flickering stars typically consisting of 50-150 nodes, which may be associated with businesses regularly contacting their clientele, or public phones which are used consistently by the same individuals. Figure 1b shows one such periodic star of 111 users over the last week of December, with particularly clear star structure on Dec. 25th and 31st and other odd-numbered days, accompanied by substantially weaker star structure on the even-numbered days. Figure 3f shows an oddly well-separated oneshot near-bipartite core which appears on Dec. 31st, consisting of two roughly equal-sized parts of 402 and 390 callers.

6 Application: Leveraging TIMECRUNCH for Interactive Visualization

One promising application of TIMECRUNCH is for dynamic graph visualization. In this section, we overview ECOVIZ (for Evolving COmparative network visualization), an interactive web application which enables pairwise comparison and temporal analysis of TIMECRUNCH's dynamic graph summary output. ECOVIZ aims to (i) adapt TIMECRUNCH to domain-specific requirements and (ii) provide efficient querying and visualization of its summary structures.

Data: We illustrate ECOVIZ using a connectomics application, which we briefly introduce for context. Connectomics involves the inference of functional brain networks from fMRI data [6]. Regions of the brain are discretized into "voxels," between which edges are inferred based on the strength of inter-voxel time series correlations. To obtain sparse brain networks (instead of full cliques), a user-defined threshold is applied to keep only the strongest correlations. Dynamic graphs of these brain networks (obtained by dividing the time series into segments, and generating a brain network per segment) reflect how patients' brain behavior changes over

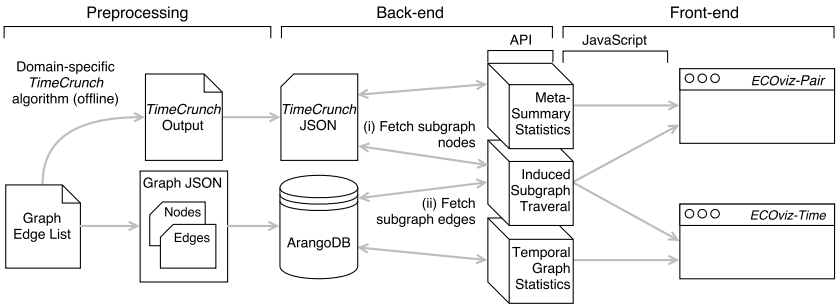


Figure 4: **End-to-end pipeline for our ECOVIZ visualization system.** Major components include offline preprocessing, ArangoDB & Flask API back-end, and web interface (JavaScript) front-end.

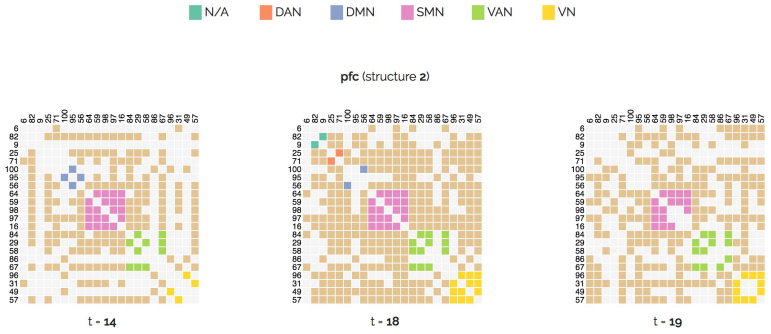


Figure 5: **ECOVIZ-TIME matrix sequence of a periodic full clique (pfc).** The colored resting-state modules show temporal patterns across time steps. Colors correspond to different voxel subnetworks: dorsal attention (DAN), default mode (DMN), sensorimotor (SMN), ventral attention (VAN), and primary visual (VN).

time. Furthermore, each voxel is associated with a subnetwork (e.g., Default Mode Network) reflecting a type of brain behavior (e.g., language, visual,sensorimotor).

ECOVIZ Overview: The first step in the ECOVIZ pipeline is to apply summarization to the input dynamic graph. In order to customize TIMECRUNCH for the connectomics application, we propose an application-specific candidate subgraph extraction routine: Instead of using the original SlashBurn clustering routine designed for real, large-scale graphs with power-law degree distribution, we use the set of all voxel nodes' egonets as candidate subgraphs. These egonets, or induced subgraphs of the central ego node and its neighbors, adjust the focus of TIMECRUNCH from high-degree hub nodes to labeled nodes from known brain regions. Note that ECOVIZ can just as well be used for visualization of large social graphs using the original candidate subgraph generation process. In addition to providing more connectomics-appropriate candidate subgraphs, these egonets also serve as natural communities suited for the small-worldness of brain networks [18].

Upon completing the TIMECRUNCH summarization process, ECOVIZ must interface the structure output format (node and time step participation per structure) with connectivity information in the underlying network. To do so, ECOVIZ receives a list of summary structures, and pairs each structural snapshot with connectivity data from the original network. This aggregation occurs in real time, and is the backbone of ECOVIZ's visualization component. For each temporal snapshot in each summary structure, the application (i) fetches participating node IDs from TIMECRUNCH results (stored in JSON), (ii) queries the graph database (ArangoDB) for an induced subgraph of the edges participating the structure, and (iii) makes asynchronous JavaScript requests to visualize each subgraph. This pipeline, as shown in Figure 4, is the source of two visualization views, ECOVIZ-PAIR and ECOVIZ-TIME, that support separate modes of data analysis.

The ECOVIZ-PAIR view allows end-users to compare pairs of summaries differing in data source (i.e., active

and rest state brain behaviors) or preprocessing method (i.e., threshold value and time interval granularity used for the dynamic network generation) for the same subject or underlying phenomenon. Meta-summary charts are also displayed to reveal summary diversity, which may be used to indirectly evaluate graph construction quality. ECOVIZ-TIME, as seen in Figure 5, shows temporal snapshots of a summary structures via panels displaying visualized subgraphs or adjacency matrices over time. Nodes can optionally be colored, reflecting group membership or targeted interest (Figure 5 shows a coloring which reflects voxels' involvement in various types of brain behavior), which aids in detection of inter- and intra-community patterns over time.

7 Conclusion

In this work, we tackle the problem of identifying significant and structurally interpretable temporal patterns in large, dynamic graphs. Specifically, we formalize the problem of finding important and coherent temporal structures in a graph as *minimizing the encoding cost* of the graph from a compression standpoint. To this end, we propose TIMECRUNCH, a fast and effective, incremental technique for building interpretable summaries for dynamic graphs which involves *generating* candidate subgraphs from each static graph, *labeling* them using static identifiers, *stitching* them over multiple timesteps and *composing* a model using practical approaches. Finally, we apply TIMECRUNCH on several large, dynamic graphs and find numerous patterns and anomalies which indicate that real-world graphs *do* in fact exhibit temporal structure. We additionally demo our ECOVIZ framework which enables interactive and domain-specific dynamic network visualization on top of TIMECRUNCH.

References

[1] DBLP network dataset. `konect.uni-koblenz.de/networks/dblp_coauthor`, July 2014.

[2] C. C. Aggarwal and P. S. Yu. Online analysis of community evolution in data streams. In *SDM*, 2005.

[3] C. J. Alpert, A. B. Kahng, and S.-Z. Yao. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics*, 90(1):3–26, 1999.

[4] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra. Com2: Fast automatic discovery of temporal ("comet") communities. In *PAKDD*, pages 271–283. Springer, 2014.

[5] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[6] E. Bullmore and O. Sporns. Complex brain networks: Graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

[7] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88. ACM, 2004.

[8] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *arXiv preprint cs/9402102*, 1994.

[9] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[10] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proc. 9th KDD*, pages 89–98, 2003.

[11] J. Ferlez, C. Faloutsos, J. Leskovec, D. Mladenic, and M. Grobelnik. Monitoring network evolution using MDL. *ICDE*, 2008.

[12] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In *KDD*, pages 606–611, 2005.

[13] U. Kang and C. Faloutsos. Beyond'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM*, pages 300–309. IEEE, 2011.

[14] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000.

[15] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The web as a graph: measurements, models, and methods. In *Computing and combinatorics*, pages 1–17. Springer, 1999.

[16] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. Vog: Summarizing and understanding large graphs.

[17] B. Kulis and Y. Guan. Graclus - efficient graph clustering software for normalized cut and ratio association on undirected graphs, 2008. 2010.

[18] C.-T. Li and S.-D. Lin. Egocentric information abstraction for heterogeneous social networks. In *International Conference on Advances in Social Network Analysis and Mining*. IEEE, 2009.

[19] M. Li and P. M. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2009.

[20] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[21] E. E. Papalexakis, N. D. Sidiropoulos, and R. Bro. From k-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE TSP*, 61(2):493–506, 2013.

[22] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

[23] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[24] N. Shah, A. Beutel, B. Gallagher, and C. Faloutsos. Spotting suspicious link behavior with fbox: An adversarial perspective. In *ICDM*. 2014.

[25] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD*, pages 1055–1064. ACM, 2015.

[26] J. Shetty and J. Adibi. The enron email dataset database schema and brief statistical report. *Inf. sciences inst. TR, USC*, 4, 2004.

[27] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696. ACM, 2007.

[28] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *KDD*, pages 965–973. ACM, 2011.

[29] K. S. Xu, M. Kliger, and A. O. Hero III. Tracking communities in dynamic social networks. In *SBP*, pages 219–226. Springer, 2011.

[30] Yike Liu and Tara Safavi and Abhilash Dighe and Danai Koutra Graph Summarization: A Survey In CoRR, abs/1612.04883, 2016.

[31] Saket Navlakha and Rajeev Rastogi and Nisheeth Shrivastava Graph Summarization with Bounded Error In SIGMOD, pages 419–432, 2008.

[32] Di Jin and Danai Koutra Exploratory Analysis of Graph Data by Leveraging Domain Knowledge In ICDM, 2017.

[33] Bijaya Adhikari and Yao Zhang and Aditya Bharadwaj and B. Aditya Prakash Condensing Temporal Networks using Propagation In SDM, pages 417–425, 2017.

[34] Nan Tang and Qing Chen and Prasenjit Mitra Graph Stream Summarization: From Big Bang to Big Crunch In SIGMOD, pages 1481–1496, 2016.

[35] Lei Shi and Hanghang Tong and Jie Tang and Chuang Lin VEGAS: Visual InfluEnce GrAph Summarization on Citation Networks In TKDE, pages 3417–3431, 2015.

[36] Yahoo! Webscope. `webscope.sandbox.yahoo.com`.