LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-670985

TimeCrunch: Interpretable Dynamic Graph Summarization

N. Shah, D. Koutra, T. Zou, B. Gallagher, C. Faloutsos

May 18, 2015

ACM SIGKDD Conference on Knowledge Discovery and Data Mining
Sydney, Australia
August 10, 2015 through August 13, 2015

Disclaimer

TimeCrunch: Interpretable Dynamic Graph Summarization

Neil Shah
Carnegie Mellon University
neilshah@cs.cmu.edu

Danai Koutra
Carnegie Mellon University
danai@cs.cmu.edu

Tianmin Zou
Carnegie Mellon University
tzou@andrew.cmu.edu

Brian Gallagher
Lawrence Livermore Lab
bgallagher@llnl.gov

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

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, botnet attackers forming a bipartite core with their victims over the duration of an attack, family members bonding in a clique-like fashion over a difficult period of time, or research collaborations forming and fading away over the years. Which patterns exist in real-world dynamic graphs, and how can we find and rank them in terms of importance? These are exactly the problems we focus on in this work. Our main contributions are (a) *formulation*: we show how to formalize this problem as minimizing the encoding cost in a data compression paradigm, (b) *algorithm*: we propose TIMECRUNCH, an effective, scalable and parameter-free 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 *6.3 million* nodes. We show that TIMECRUNCH is able to compress these graphs by summarizing important temporal structures and finds patterns that agree with intuition.

Keywords

dynamic graph; network; clustering; summarization; compression

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.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
 ISBN 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783321.

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.

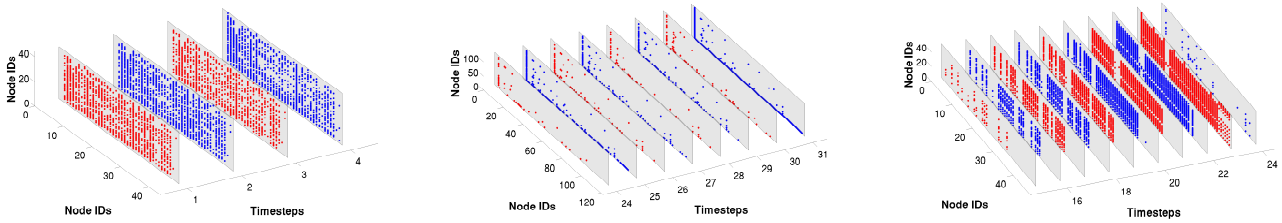
Though numerous graph algorithms suitable for static contexts such as modularity-based community detection, spectral clustering, and cut-based partitioning exist, they do not offer direct dynamic counterparts. Furthermore, the traditional goals of clustering and community detection tasks are not quite aligned with the endeavor we propose. These algorithms typically produce groupings of nodes which satisfy or approximate some optimization function. However, they do not offer characterization of the outputs – are the detected groupings stars or chains, or perhaps dense blocks? Furthermore, the lack of explicit ordering in the groupings leaves a practitioner with limited time and no insights on where to begin understanding his data.

In this work, we propose TIMECRUNCH, an effective approach to concisely summarizing large, dynamic graphs which extend beyond traditional dense and isolated "cavemen" communities. Our method works by leveraging MDL (Minimum Description Length) in order to identify and appropriately describe 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* with 55% density of 40 users in the Yahoo! messaging network over 4 weeks in April 2008. These users are likely bots messaging each other in an effort to appear normal and avoid suspension.
- Figure 1b depicts a *periodic star* of 111 callers in the phonecall network of a large, anonymous Asian city during the last week of December 2007. Notice that the star behavior oscillates over time – specifically, odd-numbered timesteps have stronger star structure than the even-numbered ones. Furthermore, the appearance of the star is strongest on Dec. 25th and 31st, corresponding to major holidays.
- Lastly, Fig. 1c shows a *ranged near clique* of 43 authors in the DBLP network who jointly published in biotechnology journals such as *Nature* and *Genome Research* from 2005-2012, agreeing with intuition as works in this field typically have many co-authors. The first and last timesteps serve only to demarcate the range of activity.

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.*



(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.

Table 1: Feature-based comparison of TIMECRUNCH with alternative approaches.

	Temporal	Time-consecutive	Time-agnostic	Dense blocks	Stars	Chains	Interpretable	Scalable	Parameter-free
GraphScope [25]	✔	✔	✘	✔	✘	✘	✘	✔	✔
Com2 [4]	✔	✔	✔	✔	✔	✘	✘	✔	✘
VoG [15]	✘	✘	✘	✔	✔	✔	✔	✔	✔
Graph partitioning [13, 17, 3]	✘	✘	✘	✔	✘	✘	✘	✔	✘
Community detection [23, 19, 5]	✘	✘	✘	✔	✘	✘	✘	?	?
TIMECRUNCH	✔	✔	✔	✔	✔	✔	✔	✔	✔

Our main contributions are as follows:

1. **Problem Formulation:** We show how to define the problem of dynamic graph understanding in in a compression context.
2. **Effective and Scalable Algorithm:** We develop TIMECRUNCH, a fast algorithm for dynamic graph summarization.
3. **Practical Discoveries:** We evaluate TIMECRUNCH on multiple real, dynamic graphs and show quantitative and qualitative results.

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. Table 1 gives a visual comparison of TIMECRUNCH with existing methods.

Static Graph Mining. Most works find specific, tightly-knit structures, such as (near-) cliques and bipartite cores: eigendecomposition [23], cross-associations [6], modularity-based optimization methods [19, 5]. Dhillon et al. [9] propose information theoretic co-clustering based on mutual information optimization. However, these approaches have limited vocabularies and are unable to find other types of interesting structures such as stars or chains. [13, 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 are not usually parameter-free. Subdue [7] and other fast frequent-subgraph mining algorithms [11] operate on labeled graphs. Our work involves unlabeled graphs and lossless compression.

Temporal Graph Mining. [2] aims at change detection in streaming graphs using projected clustering. This approach focuses on anomaly detection rather than finding recurrent temporal patterns. GraphScope [25] uses graph search for hard-partitioning of temporal graphs to find dense temporal cliques and bipartite cores. Com2 [4] uses CP/PARAFAC tensor decomposition with MDL for the same. [10] uses incremental cross-association for change detection in dense blocks over time, whereas [21] proposes an algorithm for mining cross-graph quasi-cliques (though not in a temporal context). These approaches have limited vocabularies and do not offer temporal interpretability. Dynamic clustering [27] aims to find 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. SlashBurn [12] is a recursive node-reordering approach to leverage run-length encoding for graph compression. [26] uses structural equivalence to collapse nodes/edges to simplify graph representation. These approaches do not compress the graph for pattern discovery, nor do they operate on dynamic graphs. VoG [15] 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.

3. PROBLEM FORMULATION

In this section, we give the first main contribution of our work: formulation of dynamic graph summarization as a compression problem, using MDL. For clarity, see Table 2 for a reference of the recurrent symbols used in this section.

The Minimum Description Length (MDL) principle aims to be a practical version of Kolmogorov Complexity [18], 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

Table 2: Frequently used symbols and definitions

Symbol	Definition
$G, \mathbf{A}$	dynamic graph and adjacency tensor resp.
$\mathcal{V}, n$	node-set, # of nodes of G resp.
$\mathcal{E}, m$	edge-set, # of edges of G resp.
$G_x, \mathbf{A}_x$	x^{th} timestep, adjacency matrix of G resp.
$\mathcal{E}_x, m_x$	edge-set and # of edges of G_x resp.
Δ	set of temporal signatures
Ω	set of static identifiers
Φ	lexicon, set of temporal phrases $\Phi = \Delta \times \Omega$
$\times$	Cartesian set product
M, s	model M, temporal structure $s \in M$ resp.
$\|S\|$	cardinality of set S
$\|s\|$	# of nodes in structure s
$u(s)$	timesteps in which structure s appears
$v(s)$	temporal phrase of structure s, $v(s) \in \Phi$
st, ch	star, chain resp.
fc, nc	full, near clique resp.
bc, nb	full, near bipartite core resp.
o, c	oneshot, constant resp.
r, p, f	ranged, periodic, flickering resp.
$\mathbf{M}$	approximation of $\mathbf{A}$ induced by M
$\mathbf{E}$	error matrix $\mathbf{E} = \mathbf{M} \oplus \mathbf{E}$
$\oplus$	exclusive OR
$L(G, M)$	# of bits used to encode M and G given M
$L(M)$	# of bits to encode M

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. However, our notation will reflect the treatment of the problem as one with a series of individual snapshots of graphs, rather than a tensor, for readability purposes. 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. The ideas proposed in this work, however, can easily be generalized to other types of dynamic graphs.

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 [4] : oneshot (*o*), ranged (*r*), periodic (*p*), flickering (*f*) and constant (*c*), and 6 very common structures found in real-world static graphs [14, 23] – stars (*st*), *full* and *near* cliques (*fc*, *nc*), *full* and *near* bipartite cores (*bc*, *nb*) and chains (*ch*) . Summarily, we have the signatures $\Delta = \{o, r, p, f, c\}$, static identifiers $\Omega = \{st, fc, nc, bc, nb, ch\}$ and temporal phrases $\Phi = \Delta \times \Omega$. We will further describe these signatures, identifiers and phrases after formalizing our objective.

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. We will use $area(s, M, \mathbf{A})$ to describe the edges $(i, j, x) \in \mathbf{A}$ which s induces, writing only $area(s)$ when context for M and $\mathbf{A}$ is clear.

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 the model $M \in \mathcal{M}$ which best mediates between the encoding length of the model M and the adjacency tensor $\mathbf{A}$ given M.

Our fundamental approach for transmitting the adjacency tensor $\mathcal{A}$ via the model M is described next. 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 in $area(s)$ 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.

In the following subsections, we further formalize the task of encoding the model M and the error matrix $\mathbf{E}$.

3.2 Encoding the Model

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

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

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 [22]. 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 [8], 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 next 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 in $area(s)$ 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.

3.2.1 Encoding Connectivity

In this section, we describe how to compute the encoding cost $L(c(s))$ for the connectivity for each type of static structure identifier in our identifier set Ω.

Stars: A star is characteristic of a single "hub" node connected to a set of 2 or more "spoke" nodes. We compute $L(st)$ of a star *st* as follows:

$$L(st) \quad = \quad L_{\mathbb{N}}(|st|-1) + log_2 n + log_2 \binom{n-1}{|st|-1}$$

First, we identify the number of spokes of the star. Next, we identify the hub out of n nodes using an index over the combinatorial number system. Lastly, we identify the spokes from the remainder.

Cliques: Cliques are comprised of densely connected sets of nodes. For a full clique *fc*, in which all nodes are directly connected to all other nodes in the clique, we give the cost $L(fc)$ as follows:

$$L(fc) \quad = \quad L_{\mathbb{N}}(|fc|) + log_2 \binom{n}{|fc|}$$

In this case, we encode the number of nodes in the clique followed by their ids. Note that as M is an approximation of G, fc need not *actually* be a full clique in G. If only a few edges of the full clique are not present in G, it may be worthwhile from a compression standpoint to describe it as such. In this case, each falsely represented edge will add to the error cost $\mathbf{E}$. Errors in connectivity encoding will be elaborated on in Sec. 3.3.1.

Less dense near-cliques are still interesting from a graph understanding perspective, provided they stand out from the background. For a near clique *nc*, we give $L(nc)$ as follows:

$$\begin{aligned} L(nc) \quad = \quad & L_{\mathbb{N}}(|nc|) + log_2 \binom{n}{|nc|} + log_2(|area(nc)|) \\ & + ||nc||\rho_1 + ||nc||'\rho_0 \end{aligned}$$

Here, we encode the number of nodes and their ids as in the full clique case. However, we additionally encode the edges in the near clique by encoding the number of total edges in $area(nc)$ by optimal prefix codes. We use $||nc||$ and $||nc||'$ to denote the counts for existing and non-existing edges in $area(nc)$. Then, $\rho_1 = -log(||nc||/(||nc|| + ||nc||'))$ and $\rho_0 = -log(||nc||'/(||nc|| + ||nc||'))$ represent the length of the optimal prefix codes for the existing and non-existing edges respectively. Intuitively, the more sparse or dense the near clique is, the cheaper its encoding becomes. As the encoding in this case is exact, we do not add any edges to $\mathbf{E}$.

Bipartite Cores: Bipartite cores consist of non-empty, non-intersecting node-sets L and R for which there only exist edges from L and R, but not within L or R. Note that stars can be construed as a fixed case of bipartite cores in which $|L| = 1$. The encoding cost $L(bc)$ for a full bipartite core *bc* is as follows:

$$L(fb) \quad = \quad L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2 \binom{n}{|L|} + log_2 \binom{n}{|R|}$$

In this case, we encode the number of nodes in L and R followed by the node ids in each set.

As with near cliques, near bipartite cores are also interesting if they stand out from the background. In this case, encoding is given analogously as follows:

$$\begin{aligned} L(nb) \quad = \quad & L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2 \binom{n}{|L|} + log_2 \binom{n}{|R|} \\ & + log_2(|area(nb)|) + ||nb||\rho_1 + ||nb||'\rho_0 \end{aligned}$$

Furthermore, as with near-cliques, encoding in this case is exact so we do not add any edges to $\mathbf{E}$.

Chains: A chain is characterized by series of nodes in which each node has an edge connecting it to the next node – for example, consider the node-set $\{1, 2, 3, 4\}$ in which 1 is connected to 2, 2 is connected to 3, and 3 is connected to 4. Given the right permutation, a perfect chain in an undirected graph will have edges only along two diagonals of the adjacency matrix. For a chain *ch*, we have the encoding cost $L(ch)$ as follows:

$$L(ch) \quad = \quad L_{\mathbb{N}}(|ch|-1) + \sum_{i=1}^{|ch|} log_2(n-i+1)$$

We first encode the number of nodes in the chain, followed by their node ids in order of connection.

3.2.2 Encoding Temporal Presence

For a given phrase $v \in \Phi$, it is not sufficient to only encode the connectivity of the underlying static structure. We must also encode the temporal presence $u(s)$, consisting of a set of ordered timesteps in which s appears, for each structure. In this section, we describe how to compute the encoding cost $L(u(s))$ for each of the temporal signatures in the signature set Δ.

We note that describing a set of timesteps $u(s)$ in terms of temporal signatures in Δ is yet another model selection problem for which we can leverage MDL. As with connectivity encoding, labeling $u(s)$ with a given temporal signature may not be *precisely* accurate – however, any mistakes will add to the cost of transmitting the error. Errors in temporal presence encoding will be further detailed in Sec. 3.3.2.

Oneshot: Oneshot structures appear at only one timestep in $G_1 \cdots G_t$ – that is, $|u(s)| = 1$. These structures represent graph anomalies, in the sense that they are non-recurrent interactions which are only observed once. The encoding cost $L(o)$ for the temporal presence of a oneshot structure o can be written as:

$$L(o) \quad = \quad log_2(t)$$

As the structure occurs only once, we only have to identify the timestep of occurrence from the t observed timesteps.

Ranged: Ranged structures are characterized by a short-lived existence. These structures appear for several timesteps in a row before disappearing again – they are defined by a single burst of activity. The encoding cost $L(r)$ for a ranged structure r is given by:

$$L(r) \quad = \quad L_{\mathbb{N}}(|u(s)|) + log_2 \binom{t}{2}$$

We first encode the number of timesteps in which the structure occurs, followed by the timestep ids of both the start and end timestep marking the span of activity.

Periodic: Periodic structures are an extension of ranged structures in that they appear at fixed intervals. However, these intervals are spaced greater than one timestep apart. As such, the same encoding cost function we use for ranged structures suffices here. That is, $L(p)$ for a periodic structure p is given by $L(p) = L(r)$.

For both ranged and periodic structures, periodicity can be inferred from the start and end markers along with the number of timesteps $|u(s)|$, allowing reconstruction of the original $u(s)$.

Flickering: A structure is flickering if it appears only in some of the $G_1 \cdots G_t$ timesteps, and does so without any discernible ranged/periodic pattern. The encoding cost $L(f)$ for a flickering structure f is as follows:

$$L(f) \quad = \quad L_{\mathbb{N}}(|u(s)|) + log_2 \binom{n}{|u(s)|}$$

We encode the number of timesteps in which the structure occurs in addition to the ids for the timesteps of occurrence.

Constant: Constant structures persist throughout all timesteps. That is, they occur at each timestep $G_1 \cdots G_t$ without exception. In this case, our encoding cost $L(c)$ for a constant structure c is defined as $L(c) = 0$. Intuitively, information regarding the timesteps in which the structure appears is "free," as it is already given by encoding the phrase descriptor $v(s)$.

3.3 Encoding the Errors

Given that M is a summary and the $\mathbf{M}$ induced by M is only an approximation of $\mathbf{A}$, it is necessary to encode errors made by M. In particular, there are two types of errors we must consider. The first is error in connectivity – that is, if $area(s)$ induced by structure s is not *exactly* the same as the associated patch in $\mathbf{A}$, we encode the relevant mistakes. The second is the error induced by encoding the set of timesteps $u(s)$ with a fixed temporal signature, given that $u(s)$ may not precisely follow the temporal pattern used to encode it.

3.3.1 Encoding Errors in Connectivity

We encode the error tensor $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ as two different pieces – specifically, we encode $\mathbf{E}^+$ and $\mathbf{E}^-$ where the former refers to the area of $\mathbf{A}$ which M models and $\mathbf{M}$ includes extraneous edges not present in the original graph, and the latter consists of the area of $\mathbf{A}$ which M does not model and therefore does not describe. Our reasoning for encoding these two separately is that they likely have different error distributions. Given that near cliques and near bipartite cores are encoded exactly per our model, we ignore the associated areas when encoding $\mathbf{E}^+$. The encoding for $\mathbf{E}^+$ and $\mathbf{E}^-$, denoted as $L(\mathbf{E}^+)$ and $L(\mathbf{E}^-)$ respectively is as follows:

$$\begin{aligned} L(\mathbf{E}^+) &= log_2(|\mathbf{E}^+|) + ||\mathbf{E}^+||\rho_1 + ||\mathbf{E}^+||'\rho_0 \\ L(\mathbf{E}^-) &= log_2(|\mathbf{E}^-|) + ||\mathbf{E}^-||\rho_1 + ||\mathbf{E}^-||'\rho_0 \end{aligned}$$

In both cases, we encode the number of 1s in $\mathbf{E}^+$ (or $\mathbf{E}^-$), followed by the actual 1s and 0s using optimal prefix codes.

3.3.2 Encoding Errors in Temporal Presence

For encoding errors induced by identifying $u(s)$ as one of the temporal signatures, we turn to optimal prefix codes applied over the error distribution for each structure s. Given the information encoded for each signature type in Δ, we can reconstruct an approximation $\tilde{u}(s)$ of the original timesteps $u(s)$ such that $|u(s)| = |\tilde{u}(s)|$. Using this approximation, the encoding cost $L(e_u(s))$ for the error $e_u(s) = u(s) - \tilde{u}(s)$ is defined as:

$$L(e_u(s)) = \sum_{k \in h(e_u(s))} \big(log_2(k) + log_2 c(k) + c(k)\rho_k\big)$$

where $h(e_u(s))$ denotes the set of elements with unique magnitude in $e_u(s)$, $c(k)$ denotes the count of element k in $e_u(s)$ and ρ_k denotes the length of the optimal prefix code for k. For each magnitude error, we encode the magnitude of the error, the number of times it occurs and the actual errors using optimal prefix codes. Using the model in conjunction with temporal presence and connectivity errors, a recipient can first recover the $u(s)$ for each $s \in M$, approximate $\mathbf{A}$ with $\mathbf{M}$ induced by M, produce $\mathbf{E}$ from $\mathbf{E}^+$ and $\mathbf{E}^-$, and finally recover $\mathbf{A}$ losslessly through $\mathbf{A} = \mathbf{M} \oplus \mathbf{E}$.

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.

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* the static structures from $G_1 \cdots G_t$ together to form temporal structures with coherent connectivity behavior and label them according to the 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. 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 detailed 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 $G_1 \cdots G_t$. Sec. 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.

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 leverage the encoding scheme discussed in Sec. 3.2.1: 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?

For stars, we identify the highest-degree node as the hub and all other nodes as spokes. For near and full bipartite cores, finding the right permutation can be reduced to finding the maximum bipartite subgraph, which is equivalent to finding the maximum cut and is NP-hard. As a result, we use a heuristic approach which formulates the problem as a two-class classification task. To this end, we initialize L to contain the highest-degree node in $\mathcal{L}$, and R to contain its neighbors. We then use Fast Belief Propagation [16] with heterophily (assuming connected nodes belong to different classes) to propagate the class labels and determine L and R. For near and full cliques, any permutation is equally good. Lastly, for chains, finding the right permutation is equivalent to finding the longest path, which is NP-hard. As a result, we again employ a heuristic approach in which we select a node in $\mathcal{L}$ at random, use BFS to find the furthest node away, and repeat with the resulting node while extending the chain through local search iteratively. For both near cliques and bipartite cores, we do not encode $\mathbf{E}_{nc}^+$ and $\mathbf{E}_{nb}^+$ as $L(nc)$ and $L(nb)$ encode the relevant edges exactly.

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, [20] 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 using the temporal encoding framework derived in Sec. 3.2.2. 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

Table 3: Dynamic graphs used for empirical analysis

Graph	Nodes	Edges	Timesteps
Enron [24]	151	20 thousand	163 weeks
Yahoo-IM [28]	100 thousand	2.1 million	4 weeks
Honeynet	372 thousand	7.1 million	32 days
DBLP [1]	1.3 million	15 million	25 years
Phonecall	6.3 million	36.3 million	31 days

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? Lastly, is TIMECRUNCH scalable?

5.1 Datasets and Experimental Setup

For our experiments, we use 5 real dynamic graph datasets – they are summarized in Table 3 and described below.

Enron: The Enron e-mail dataset is publicly available. It contains 20 thousand 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. It contains 2.1 million sender-receiver pairs between 100 thousand users over 5709 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 to attackers) It contains source IP, destination IP and attack timestamps of 372 thousand (attacker and honeypot) machines with 7.1 million 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, indicating joint publication. We used a subset of DBLP spanning 25 years, from 1990 to 2014, with 1.3 million authors and 15 million unique author-author collaborations over the years.

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

In our experiments, we use "SlashBurn" 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 only improve results given MDL. 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 SNMMs 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 from Table 3 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 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 4 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.

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

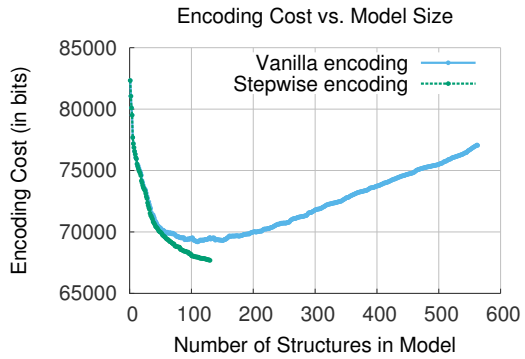


Figure 2: **TIMECRUNCH-STEPWISE summarizes** Enron **using just 78% of ORIGINAL's bits and 130 structures compared to 89% and 563 structures of TIMECRUNCH-VANILLA by pruning unhelpful structures from the candidate set.**

OBSERVATION 1. *Real-world dynamic graphs are not unstructured.* TIMECRUNCH *gives better encoding cost than* ORIGINAL*, indicating the presence of temporal graph structure.*

5.3 Qualitative Analysis

In this section, we discuss qualitative results from applying TIMECRUNCH to the graphs mentioned in Table 3.

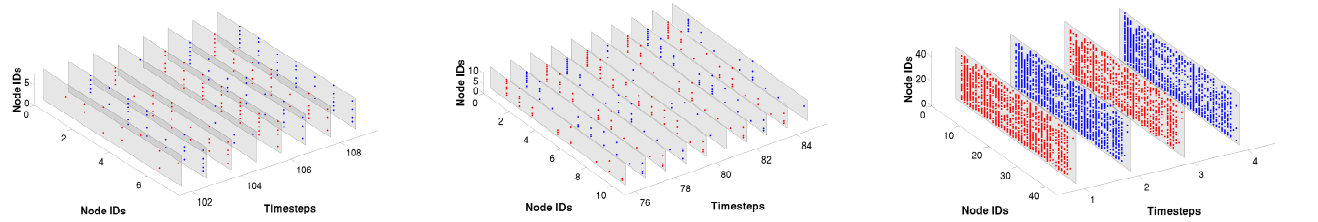
Enron: The Enron graph is characteristic of many periodic, ranged and oneshot stars and several periodic and flickering cliques. Periodicity is reflective of office e-mail communications (e.g. meetings, reminders). Figure 3a shows an excerpt from one flickering clique which corresponds to the several 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. 3d 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 3c 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 3e 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 3f 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 3g 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.

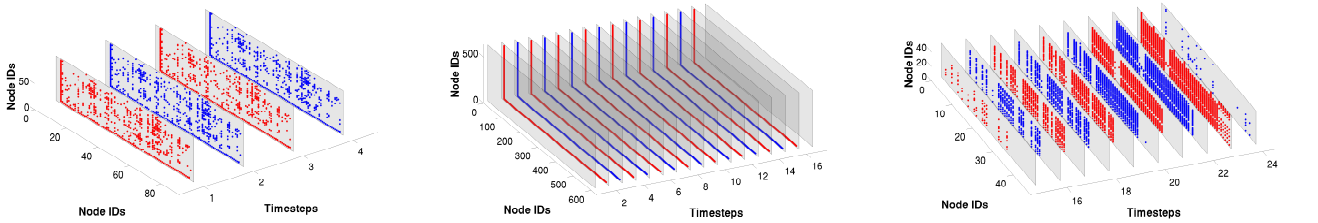
Phonecall: The Phonecall dataset is largely comprised of temporal stars and few dense clique and bipartite structures. Again, we have a large proportion of oneshot stars which occur only at single timesteps. Further analyzing these results, we find that 111 of the 187 oneshot stars (59%) are found on 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 3h 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 3i shows an oddly well-separated oneshot near-bipartite core which appears on Dec. 31st, consisting of two roughly equal-sized parts of 402



(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

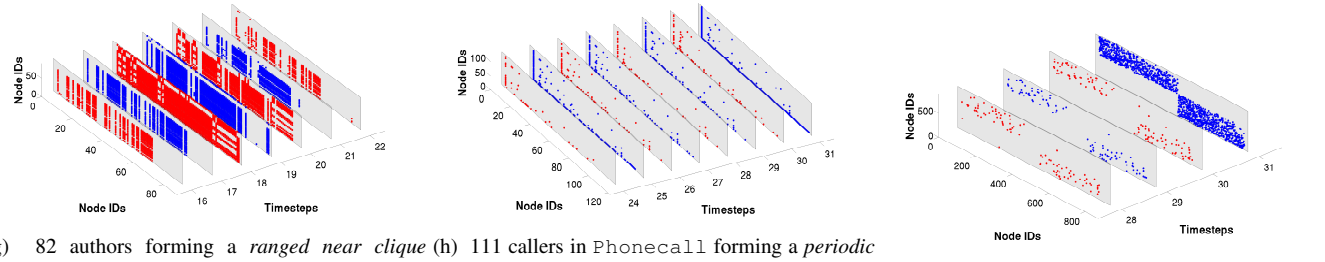
(c) 40 users in Yahoo-IM forming a *constant near clique* with 55% density over the observed 4 weeks



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

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

(f) 43 authors that publish together in biotechnology journals forming a *ranged near clique* on DBLP



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

(h) 111 callers in Phonecall forming a *periodic star* appearing strongly on odd numbered days, especially Dec. 25 and 31

(i) 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.

Table 5: Frequency of each temporal structure type discovered using TIMECRUNCH-STEPWISE for each dataset.

	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

and 390 callers. Though we do not have ground truth to interpret these structures, we note that a practitioner with the appropriate information could better interpret their meaning.

5.4 Scalability

All components of TIMECRUNCH are carefully designed to be linear or near-linear on the number of nonzero edges. Figure 4 shows the near-linear runtime of TIMECRUNCH on several induced temporal subgraphs (up to 14M edges) taken from the DBLP dataset at varying time-intervals. Our experiments were conducted on a machine with 80 Intel Xeon(R) 4850 2GHz cores and 256GB RAM. We use MATLAB for candidate subgraph generation and temporal stitching and Python for model selection heuristics.

Furthermore, much of the TIMECRUNCH pipeline (per-timestep summarization) is embarassingly parallelizable and can be easily split over nodes. Distributed eigensolver implementations also exist in practice for the stitching component.

6. 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 ap-

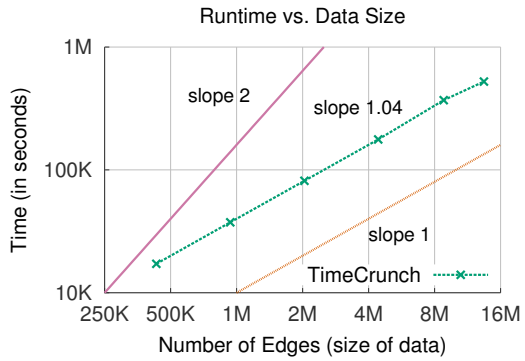


Figure 4: **TIMECRUNCH scales near-linearly on the number of edges in the graph.** Here, we use several induced temporal subgraphs from DBLP, up to 14M edges in size.

proaches. 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.

7. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. IIS-1217559, CNS-1314632 and DGE-1252522. Prepared by LLNL under Contract DE-AC52-07NA27344. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, DARPA, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

8. REFERENCES

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

[2] C. C. Aggarwal and S. Y. Philip. Online analysis of community evolution in data streams. SIAM.

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

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

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

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

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

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

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

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

[14] 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.

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

[16] D. Koutra, T.-Y. Ke, U. Kang, D. H. P. Chau, H.-K. K. Pao, and C. Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *ECML/PKDD*, pages 245–260. Springer, 2011.

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

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

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

[20] 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.

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

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

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

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

[25] 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.

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

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

[28] Yahoo! Webscope. webscope.sandbox.yahoo.com.