Summarizing Dynamic Graphs using MDL

Divyam Saran[1] and Jilles Vreeken[2]

[1] Saarland University, Germany dsaran@mmci.uni-saarland.de
[2] CISPA Helmholtz Center for Information Security, Germany jv@cispa.saarland

Abstract. How can we succinctly describe a large, dynamic graph over time? Given a large dynamic graph, can we find "important" patterns that evolve over time, so that we can easily summarize and visualize it? In real life, these patterns signify interaction between nodes over time - for example, how the network traffic of a bank changes during the day, how calling patterns change season over season, or how people watch different genre of movies over different times of the year. Our work focuses on the problem of how we find and rank these patterns. To this end, we formalize this problem as minimizing the encoding cost in a data compression paradigm and propose Mango, an effective heuristic for finding evolving patterns in dynamic graphs. We then apply our method to synthetic datasets and *dblp* and show that Mango is able to summarize dynamic graphs by finding meaningful static and temporal patterns.

This work is currently in progress and we are evaluating performance on real world datasets. We are sharing preliminary evaluation on a few synthetic datasets and a popular real-world dataset, *dblp*.

Keywords: graph mining · dynamic graphs · minimum description length

1 Introduction

Given a social network, such as Twitter, over time, how can we best describe its structure and connectivity? Is it random, or are there a set of structures that exist through multiple timesteps, evolving through time? When summarizing dynamic graphs, it is important to not just generate succinct summaries for the present timestep, but also understand its corresponding interactions and relationships with the past and future instances. Our work aims to accomplish exactly this; specifically, we focus on generating concise summaries of large dynamic graphs to better understand their underlying characteristics.

Graph structures are among the most frequent and popular data structures given their flexibility and dynamism. The world wide web, social networks, databases, biological records, and knowledge graphs are all examples of large-scale interconnected graphs. Dynamic graphs are extensively used to model relationships over time: How does a LinkedIn user's connections evolve over time? Do certain authors always work together in certain conferences year over year? These queries that can easily be modeled using dynamic graphs.

Many approaches use community detection, clustering, or graph-cut algorithms to summarize the graph in terms of its communities, but lack explicit ordering [1–3]. These approaches also fail to characterize the subgraphs (e.g., clique, star) and do not align to our goal – we want to describe the structures in a graph using an enriched set of "vocabulary" terms: cliques, stars, and bi-partite cores. These vocabulary terms provide advantages over other characterizations: (a) "cavemen" graphs can be detected using cliques (b) stars [4] and bi-partite cores [3] are very common, and have semantic meaning (e.g., factions, bots) in real networks we have seen in practice (e.g., IMDB movie-actor graph, Netflix movie recommendations). Furthermore, these algorithms are only suitable for a static context, and do not offer direct dynamic counterparts. Other algorithms, like TimeCrunch [5], do work in a dynamic setting, but focus on finding static patterns that appear over multiple timesteps. Our work aims to model how these relationships between nodes evolves over time, i.e. we want to first find patterns, and then explain how these patterns change over time.

In this work, we propose MANGO, an efficient method to summarize and understand large dynamic graphs that extend beyond dense and isolated "cavemen" networks. We formalize our goal as a lossless compression problem, i.e. the best summary of a dynamic graph is a set of subgraphs and a set of dynamic nodes that best describes the graph over time. To accomplish this, we define a vocabulary to express the structures, its temporal presence, and its dynamic behavior. Our method, MANGO, then uses the Minimum Description Length (MDL) principle to select the model that results in the best compression of the graph. By finding subgraphs and corresponding dynamic nodes that compresses a dynamic graph best, our approach helps a human understand the main graph characteristics and its temporal connectivity behavior in a simple, non-redundant manner.

Our main contributions are as follows:

- We formulate the dynamic graph summarization problem in terms of Minimum Description Length (MDL) principle.
- To efficiently discover the static and temporal patterns in dynamic graphs, we propose MANGO.
- We evaluate our algorithm on several synthetic and real-world graphs with millions of edges, and show quantitative and qualitative results.

The remainder of this work is organized as follows. We cover related work in Section 2. In Sections 3 and 4, we respectively present the problem formulation and describe our method in detail. We evaluate MANGO using synthetic and real data in Section 5. In Section 6, we round up with conclusions and future work.

2 Related Work

We classify the related work into three categories: static graph mining, temporal graph mining, and graph compression and summarization. We discuss them below.

2.1 Static Graph Mining

Most works in static graph mining, like eigendecomposition [6], cross-associations [7], and modularity-based optimization methods [8, 9], find specific, tightly-knit structures, such as (near-) cliques and bipartite cores. Some information theory based approaches, like Dhillon et al. [10], exist, but they have limited structural vocabularies. Other works, like cut-based partitioning [11, 12], or spectral partitioning using multiple eigenvectors [13], require parameters and differ from our objective of identifying communities by instead seeking hard clustering of all nodes. Other approaches, like Subdue [14] and other fast frequent-subgraph mining algorithms [15], focus on labeled graphs. Our work focuses on unlabeled graphs and uses lossless compression.

2.2 Temporal Graph Mining

Most work on temporal graphs focuses on the evolution of specific properties, change detection, or community detection. Aggarwal et al. [16] use projection clustering to detect change in streaming graphs. GraphScope [17] finds dense temporal cliques and bipartite cores by using graph search for hard-partitioning of temporal graphs. Com2 [18] uses CP/PARAFAC decomposition with MDL for the same. Other approaches have limited vocabulary and provide no interpretability for dynamic graphs. Ferlez et al. [19] use incremental cross-association for change detection in dense blocks, whereas Pei et al. [20] mine atemporal cross-graph quasi-cliques. Dynamic clustering [21] finds stable clusters over time by penalizing deviations from incremental static clustering. Our work differs from these approaches as it focuses on interpretability and is based on a vocabulary, and we mine subgraphs that may be present in one, some or all instances of the dynamic graph. TimeCrunch [5] summarizes dynamic graphs and bears close resemblance to our work. TimeCrunch generates static subgraphs that are wholly present in one or more snapshots in a dynamic graph. Our approach, however, not only finds structures that are present in multiple snapshots, but also captures the evolution of these structures by giving it flexibility to change forms (for example, clique to star), grow/shrink, and split/merge.

2.3 Graph Compression and Summarization

There is limited work in Graph Compression and Summarization, and the approaches generally apply only to static graph without an obvious extension to dynamic graphs. SlashBurn [22] is a recursive node-reordering approach to leverage run-length encoding; weighted graph compression [23] simplifies graph representation by using structural equivalence to collapse nodes/edges. VoG [24] uses MDL to label subgraphs in terms of stars, (near) cliques, (near) bipartite cores and chains. MeGS [25] also uses MDL to label cliques, bipartite, tree, hub and sparse subgraphs. A direct extension of these algorithms to dynamic graphs would lead to multiple disconnected descriptions and not solve our purpose of interpretable, connected, and succinct summaries. These methods, however, are

still relevant when summarizing individual snapshots of graphs. Our work builds on the works of Koutra et al. [24], extending their vocabularies to model the dynamic structures over multiple timesteps.

3 Problem Formulation

In this section we formulate the dynamic graph summarization problem. We analyze undirected dynamic graphs using fixed length, discretised time intervals. We treat this problem as a series of individual snapshots of graphs over T timesteps. We consider a dynamic graph $G(\mathcal{V}, \mathcal{E})$ with adjacency tensor $\mathbf{A}$, $n = |\mathcal{V}|$ nodes, m = $|\mathcal{E}|$ edges, and T timesteps. Given G_t and $\mathcal{E}_t$, the respective graph and edge snapshots at t^{th} timestep, we can write our dynamic graph as:

$$G = \bigcup_t G_t(\mathcal{V}, \mathcal{E}_t) \tag{1}$$

We use a similar notation and vocabulary as described in Shah et al. [5] and Koutra et al. [24]. We consider a set of temporal phrases $\phi = \Delta \times \Omega \times \theta$, where Δ corresponds to the set of temporal signatures, Ω corresponds to the set of structure vocabulary, θ corresponds to the set of dynamic signatures, and $\times$ denotes the Cartesian set product. We choose four temporal signatures: oneshot (o), periodic (p), sequential (sq) and sporadic (sp), and five commonly occurring structures: stars (st), full and near cliques (fc, nc), and full and near bi-partite cores (bc, nb). The key difference between our work and TimeCrunch [5] is how flexible we let our structures be. Unlike TimeCrunch, structures in our work are afforded one or more of five dynamic properties: consistent (c), grow/shrink (gs), change pattern (cp), split (s), and merge (m). In particular, we define a set of dynamic signatures, θ, to capture the dynamic behavior of each structure, with $\theta = \{c,\ gs,\ cp,\ s,\ m\}$. We elaborate these in detail in Section 3.2. To summarize, we have temporal signatures $\Delta = \{o,\ p,\ sq,\ sp\}$, structure vocabulary $\Omega = \{st,\ fc,\ nc,\ bc,\ nb\}$, dynamic signatures $\theta = \{c,\ gs,\ cp,\ s,\ m\}$, and temporal phrases $\phi = \Delta \times \Omega \times \theta$.

Informally, our goal is to model parts of the adjacency tensor $\mathbf{A}$ by a set of static and temporal patterns that are part of our vocabulary ϕ, and encode the remaining unmodelled edges of $\mathbf{A}$ as noise/error. For example, a possible summary for a tensor $\mathbf{A}$ over 3 timesteps could be – a set of static structures (Oneshots) at t=1,2,3, a set of dynamic structures beginning at t=1,2, and an error matrix consisting of edges not described by models but present in $\mathbf{A}$.

We next formulate our problem in a compression setting, define our model family $\mathcal{M}$, and describe how a model $M \in \mathcal{M}$ is encoded in bits.

3.1 MDL for Dynamic Graph Summarization

In general, the MDL principle [26] is a practical version of Kolmogorov Complexity [27], and is often associated with the slogan *Induction by Compression*. In this work, we use two-part or *crude* MDL, and encode the model and data

separately. We choose two-part MDL as we are particularly interested in the model: the structures that best describe the dynamic graph. The two-part MDL can be described as follows. Given a model family $\mathcal{M}$, the best model $M \in \mathcal{M}$ minimizes:

$$L(M) + L(\mathcal{D}|M), \tag{2}$$

where

- $L(M)$ is the length in bits to describe model M, and
- $L(\mathcal{D}|M)$ is the length, in bits, to describe data $\mathcal{D}$ when encoded using M.

We consider ordered lists of temporal graph structures with possible node overlap as models $M \in \mathcal{M}$. Given the adjacency tensor $\mathbf{A}$, we describe edges on a first-come-first-serve basis. At timestamp t, a structure $s \in M$ describes a region on the adjacency tensor $\mathbf{A}$. We will use the *area(s, M, $\boldsymbol{A}$)* to describe the edges (i, j) induced by the structure s, and use *area(s)* whenever M and $\mathbf{A}$ are known.

We now discuss the approach to transmit the adjacency tensor, $\mathbf{A}$. We transmit the adjacency tensor per timestep. For each timestep t, we transmit the model at that timestep, M_t. Next, given model M, we iteratively consider each dynamic structure $s \in M_t$. A dynamic structure $s \in M_t$, is one that starts at timestep t, and can exist in subsequent timesteps. With each structure s, we induce the edges in area(s) in the approximation of adjacency tensor $\mathbf{M}$, and we want $\mathbf{M} \approx \mathbf{A}$. Given that $\mathbf{M}$ is a summary approximation of $\mathbf{A}$, it is most likely that $\mathbf{M} \neq \mathbf{A}$. To ensure fair comparison between models, MDL requires lossless encoding. Hence, besides $\mathbf{M}$, we also transmit the error $\mathbf{E}$, which encodes the error w.r.t. $\mathbf{A}$. We obtain $\mathbf{E}$ by taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$, i.e. $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$. Given $\mathbf{M}$ and $\mathbf{E}$, we can reconstruct the original adjacency tensor $\mathbf{A}$ without loss.

In the following subsections, we look at how to encode the model M. For encoding the error tensor $\mathbf{E}$, we use encodings provided by Koutra et al. [24].

3.2 Encoding the Model

We first describe how a model that first appears at timestep t, $M_t \in \mathcal{M}$, is encoded.

$$L(M_t) = \sum_{D \epsilon \Delta} \Bigg[\underbrace{\log |D|}_{\substack{\text{type of temporal} \\ \text{signature D}}} + \underbrace{L_N(|M_t| + 1) + \log \binom{|M_t| + |\Omega| - 1}{|\Omega| - 1}}_{\substack{\text{total \# of structures of type D beginning} \\ \text{at time } t\text{, per structure type}}} + \underbrace{\sum_{s \epsilon M_t} (-\log Pr(x(s)|M) + L(s) + L(t(s)) + L(d(s)))}_{\substack{\text{per structure, in order, type, its details, future} \\ \text{timesteps, and dynamic nodes for future timesteps}}} \Bigg] \tag{3}$$

For a given timestep t, we transmit all models of the same temporal structure together. For example, we first transmit the Oneshots, then the Sequentials, followed by the Periodics, and finally the Sporadics that begin at timestep t. So we first transmit the type of temporal structure D and the total number of structures of type D beginning at timestep t, using L_N, the MDL optimal encoding for integers ≥ 1 [26]. Next, we optimally encode the number of structures of each type $x \epsilon \Omega$ in model M_t. Then for each structure $s \epsilon M_t$, we encode its type using optimal prefix code [28], its ordered temporal presence, and the set of dynamic nodes over those timesteps. The overall cost for all T timesteps can then be calculated as:

$$L(M) = \underbrace{L_N(|T|)}_{\text{\# of timesteps}} + \sum_{t=1}^{T} \Big[\underbrace{\log T}_{\text{timestep id}} + \underbrace{L(M_t)}_{\text{cost at timestep t}} \Big] \tag{4}$$

We next define the different encoding costs to compute the encoded length of a model. For each graph structure type, $L(s)$ corresponds to the encoding cost of *area(s)* induced by s. Further, $L(t(s))$ and $L(d(s))$ correspond to the encoding cost of the future timestamps that the structure s occurs in, and changes to it over those timestamps, respectively. We next discuss these individual encoding costs.

Encoding the structure: In this section we discuss the encoding cost for a static structure belonging to our defined vocabulary Ω, $L(s)$. Unless stated specifically, we consider that we will pick nodes for structure s from all $n = |\mathcal{V}|$ nodes in the graph, i.e. we write $L(s)$ whenever we consider all n nodes in the graph. We write $L(s, m)$ whenever we pick structure s from m nodes, $m \neq n$.

Cliques : For a full clique, i.e. a set of fully-connected nodes, we first encode the number of nodes, and then their ids

$$L(fc) = \underbrace{L_N(|fc|)}_{\text{\# of nodes}} + \underbrace{\log \binom{n}{|fc|}}_{\text{node ids}} \tag{5}$$

Since M generalizes the graph, we do not require that fc is a full clique in G_t. If only a few edges are missing, it may still be convenient to describe it as a full clique. Every missing edge, however, adds to the cost of transmitting $\mathbf{E}$. Less dense or near-cliques are also interesting provided they stand out from the background. We can encode near-cliques as follows:

$$L(nc) = \underbrace{L_N(|nc|)}_{\text{\# of nodes}} + \underbrace{\log \binom{n}{|nc|}}_{\text{node ids}} + \underbrace{\log(|area(nc)|)}_{\text{\# of edges}} + \underbrace{||nc||l_1 + ||nc||'l_0}_{\text{edges}} \tag{6}$$

We transmit the number and ids of nodes as in Equation 5. We identify the number of present and missing edges by using optimal prefix codes. We write $||nc||$ and $||nc||'$ for the number of present and missing edges in *area(nc)*, respectively. Then, $l_1 = \log ||nc||/(||nc|| + ||nc||')$, and $l_0 = -\log ||nc||'/(||nc|| + ||nc||')$, are

the lengths of the optimal prefix codes for respectively present and missing edges. Intuitively, the denser/sparser a near-clique is, the cheaper it is to encode. This encoding is exact and no edges are added to **E**.

Bipartite Cores: Bipartite cores are non-empty, non-intersecting sets of nodes, A and B, for which there are edges only between the sets A and B, and not within. The encoded length of a full bipartite core bc is

$$L(bc) = \underbrace{L_N(|A|) + L_N(|B|)}_{\text{\# of nodes in A and B}} + \underbrace{\log \binom{n}{|A|, |B|}}_{\text{node ids in A and B}} \tag{7}$$

Here we first encode the size of A and B, and then their ids. Similar to cliques, we also encode the near bipartite cores as

$$\begin{aligned} L(nb) = \underbrace{L_N(|A|) + L_N(|B|)}_{\text{\# of nodes in A and B}} + \underbrace{\log \binom{n}{|A|, |B|}}_{\text{node ids in A and B}} \\ + \underbrace{\log(|area(nb)|)}_{\text{\# of edges}} + \underbrace{||nb||l_1 + ||nb||'l_0}_{\text{edges}} \end{aligned} \tag{8}$$

As with near-cliques, encoding of near-bipartite cores is exact and does not add to **E**.

Stars : A star is characterized by a single "hub" node connected to a set of 2 or more "spoke" nodes. We can encode stars as:

$$L(st) = \underbrace{L_N(|st| - 1)}_{\text{\# of spokes}} + \underbrace{\log n}_{\text{hub node id}} + \underbrace{\log \binom{n-1}{|st|-1}}_{\text{ids of spoke nodes}} \tag{9}$$

We first transmit the number of spokes of the star, followed by the id of the hub node (out of n nodes) and then the spokes from remaining n-1 nodes.

Encoding Temporal Signatures We now describe how to compute encoding cost $L(t(s))$ for each temporal signature $t(s) \epsilon \Delta$, where each temporal signature $t(s)$ consists of a set of ordered list of timesteps in which s occurs.

Oneshot: Oneshot structures appear in only one timestep. To encode them, we only need to encode the timestep they occur in, and given Equation 4 already incorporates this, we do not need any further encoding.

Sequential: A Sequential structure occurs in every timestep between t_{start} and t_{end}, and can be encoded as:

$$L(sq) = \underbrace{\log \binom{T}{2}}_{\text{choose } t_{start} \text{ and } t_{end}} \tag{10}$$

Periodic: Periodic structures occur in not all, but at fixed intervals between t_{start} and t_{end}. We can identify the timesteps the structure occurs in using the number of timesteps $|p|$, t_{start}, and t_{end}. We can encode periodics as:

$$L(p) = \underbrace{L_N(|p|)}_{\text{\# of timesteps}} + \underbrace{\log \binom{T}{2}}_{\text{choose } t_{start} \text{ and } t_{end}} \tag{11}$$

Sporadic: Sporadic structures occur in not all but in multiple timesteps between t_{start} and t_{end}, without following a periodic pattern. We can encode Sporadic structures as:

$$L(sp) = \underbrace{L_N(|sp|)}_{\text{\# of timesteps}} + \underbrace{\log \binom{T}{|sp|}}_{\text{timestep ids}} \tag{12}$$

Encoding Dynamic Structures We now discuss how we encode the dynamic properties, $L(d(s))$, for our structures. As we briefly mentioned earlier, each of the structures can occur in multiple timesteps, and is allowed to have these dynamic properties: consistent (c), grow/shrink (gs), change pattern (cp), split (s), and merge (m). These properties help capture how a structure can evolve over time. We now discuss each of these properties alongwith its encoding cost:

Consistent: A Consistent structure doesn't change over multiple timesteps, in type, or in number of nodes. Given no nodes are added/removed, we need not add anything to $L(d(s))$ for Consistent graphs. Given the simple encoding for such structures, it may be worthwhile to encode structures as Consistent even if such a labelling is not precisely accurate. However, the excess edges caused by Consistent structures adds to the cost of transmitting $\mathbf{E}$.

Grow/Shrink: A structure Grows/Shrinks if nodes are added/removed from it, and its type remains same. We first encode the base structure, and for each future timestep, we encode the nodes added/removed during that timestep. Given our base structure types, we have two different encoding costs, one for stars and cliques, and the other for bipartite cores.

– **Stars and Cliques:** For stars and cliques, we need to keep the list of nodes that are added or removed over different timesteps. The encoding cost is:

$$\begin{aligned} L(gs) = \sum_{t \epsilon t(s)} & \underbrace{L_N(|n_t^+|)}_{\substack{\text{\# nodes added} \\ \text{at time t}}} + \underbrace{L_N(|n_t^-|)}_{\substack{\text{\# nodes removed} \\ \text{at time t}}} \\ & + \underbrace{\log \binom{|n| - |S|}{|n_t^+|}}_{\substack{\text{node ids of nodes} \\ \text{added at time t}}} + \underbrace{\log \binom{|S|}{|n_t^-|}}_{\substack{\text{node ids of nodes} \\ \text{removed at time t}}} \end{aligned} \tag{13}$$

Let $|S|$ be the size of base structure, we first encode the number of nodes added/removed, and then respective ids of nodes that were added/removed. When picking ids that were added, we do not consider ids that were already part of the structure. Similarly, we only pick removed nodes from nodes that were already in $|S|$. To identify growth type, i.e. whether a structure grows or shrinks, we can check the sign of the difference $|n_t^+| - |n_t^-|$, a positive sign indicates a growing structure, while a negative sign indicates a shrinking structure. At every timestep, we update the error **E** based on the updated structure, i.e. considering the base structure, added nodes, and removed nodes.

– **Bipartite Cores:** For bipartite cores, in addition to maintaining the nodes that were added/removed, we also have to maintain in which core are these nodes added or removed. The encoding cost is then given by:

$$L(gs) = \sum_{t \epsilon t(s)} \underbrace{\sum_{j=1}^{2}}_{\text{for both cores}} \Bigg[\underbrace{\log 2}_{\text{core id}} + \underbrace{L_N(|n_t^+|)}_{\substack{\#\text{ nodes added} \\ \text{at time t}}} + \underbrace{L_N(|n_t^-|)}_{\substack{\#\text{ nodes removed} \\ \text{at time t}}} + \underbrace{\log \binom{|n| - |S|}{|n_t^+|}}_{\substack{\text{node ids of nodes} \\ \text{added at time t}}} + \underbrace{\log \binom{|S|}{|n_t^-|}}_{\substack{\text{node ids of nodes} \\ \text{removed at time t}}} \Bigg] \tag{14}$$

The encoding is similar to the previous case, however, we now consider both cores in the structure. To identify growth type, we now consider the sign of the difference $|n_t^+| - |n_t^-|$ for both cores, a positive sign on both cores indicates that both cores are growing, while a positive sign on one and negative on the other show that one core is growing while the other is shrinking. We update the error **E** at every timestep considering both cores in the structure, and the respective nodes added(removed) to(from) each.

Change Pattern: To capture changing patterns, we only need to specify the new structure at the next timestep. Let the initial structure be s_1 and it changes to a structure of different type s_2 at the next timestep, we can then just encode the new structure at the next timestep:

$$L(cp) = L(s_2) \tag{15}$$

It is important to note that while transmitting such structures, we need to specify to the receiver when exactly the pattern change is happening. As we will discuss later in this section, keeping track of when a change is happening enables us to use multiple different dynamic properties for the same structure.

Split: A split happens when a large structure splits into two or more smaller structures. Encoding splits is similar to encoding changing patterns, we only need to send bits of the new substructures. However, it is important to notify the receiver that a split is happening at the current timestep. Let the base structure s have $|S|$ nodes. Given the nodes for substructures can only be from

the base structure, we can then write a split of s into s_1 and s_2 as:

$$L(s) = \underbrace{\log \binom{|S|}{|s_1|}}_{\text{ids in } s_1} + \underbrace{L(s_1, |S|)}_{\text{encoding } s_1} + \underbrace{\log \binom{|S|}{|s_2|}}_{\text{ids in } s_2} + \underbrace{L(s_1, |S|)}_{\text{encoding } s_2} \tag{16}$$

Merge: Similar to split, we can encode a merge of two structures s_1 and s_2 into s by maintaining these nodes in the new structure. The encoding is:

$$L(m) = L(s, |s_1| + |s_2|) \tag{17}$$

Adding multiple dynamic properties to a structure: Structures are allowed to have multiple dynamic properties across timesteps, for eg., a structure may be consistent at one timestep, grow/shrink at the next timestep, and split later on. To enable the same, we add a cost $\log |\theta|$ for the structure at every timestep, to indicate what is the behavior of the structure at the current timestep (note $|\theta| = 5$, given we have 5 dynamic properties).

We now have all the necessary ingredients to formally define the dynamic graph summarization problem:

Problem 1. Minimum Dynamic Graph Description Problem
Given a dynamic graph G with adjacency tensor $\mathbf{A}$ and temporal phrases ϕ, we want the smallest model M that minimizes the total encoding length

$$L(G, M) = L(M) + L(\mathbf{E}) \tag{18}$$

where $\mathbf{E}$ is the error tensor computed as $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ and $\mathbf{M}$ is the approximation of $\mathbf{A}$ given M. It can be noted that $L(\mathbf{E})$ maps to $L(\mathcal{D}|M)$, introduced in Equation 2.

Complexity of search space $\mathcal{M}$: It is important to note that the search space of $\mathcal{M}$ is combinatorial, it involves choosing the smallest encoding cost from *all* possible subsets of temporal structures $\mathcal{C}$, from *all* possible structures over the vocabulary Ω, over *all* possible timesteps T. Additionally, it does not exhibit trivial structures like modularity or (weak) (anti)monotonicity that could be exploited for efficient search. Miettinen and Vreeken [29] showed that finding the MDL optimal model of only full-cliques is NP-hard. Considering the computationally challenging task at hand, we have to resort to heuristics. We now propose Mango, a greedy heuristic to summarize dynamic graphs.

4 Proposed Algorithm: Mango

Mango is a greedy heuristic that combines static candidates to generate temporal patterns in summaries. As a prerequisite, we generate static summaries on individual snapshots using VoG [24], MeGS [25], or any other graph summarization method that follows a similar vocabulary to ours. Mango then picks candidates from these static summaries and forms a priority queue to evaluate which candidates improve the MDL score for the dynamic graph using encodings detailed in Section 3.2. A pseudo code of the algorithm is given in Algorithm 1.

Broadly, the algorithm runs in four steps:

Step 1: Candidate Generation. First, we follow a one-vs-all approach to generate candidates for one snapshot w.r.t. all other snapshots. Given static summaries for graph snapshots $VG_1...VG_T$, for each snapshot G_i, we use edge intersection scores and MDL scores to find candidates from snapshots $G_1..G_{j\neq i}..G_T$ that are a good fit for G_i. This step also surfaces structures that were missed in G_i but found in other snapshots by our static summarizer.

Step 2: Form Similar Candidate Combinations. Next, given the candidate list from the previous step, we form combinations of candidates that would be a good fit for the current snapshot. This step is crucial for determining candidates for split and merge scenarios.

Step 3: Generate Priority Queue. Then, given a combination of candidates from other snapshots, we find if there are competing similar structures in the current snapshot. Lack of competing structures implies our summarizer failed to detect the structures in the current snapshot, and the structures should be added to the current snapshot's summary. We add the candidate combinations and their corresponding competing candidates to a priority queue based on the MDL scores of the candidates.

Step 4: Generate summary assembly from Priority Queue. Finally, for all candidate combinations and its corresponding competing structures, we check if adding candidates in a dynamic fashion (using scores defined in section 3.2) improves the MDL score for the dynamic graph. This comparison validates if we should keep individual static structures or replace them with a dynamic one. This also verifies if the competing model is a better fit or the candidate does a better job of explaining the snapshot.

We now evaluate the performance of MANGO on large dynamic graphs.

Algorithm 1: Greedy Dynamic Summarization

Input: Graph snapshots $G_1, ..., G_T$, respective static summaries $VG_1, ..., VG_T$
Step 1: Candidate Generation. For each snapshot G_i, generate list of candidates $C_1, C_2, ..., C_n$ from all other snapshots.
Step 2: Generate Candidate Combination. Given list of candidates $C_1, C_2, ..., C_n$, generate combinations $C_j...C_k, C_m...C_n, ...$ that are a good fit for the current snapshot.
Step 3: Form Priority Queue. Given combinations $C_j...C_k, C_m...C_n, ...$ and current snapshot summary VG_i, for each candidate combination generate competing models from VG_i. Add to priority queue based on MDL score fit.
Step 4: Summary Assembly. For each candidate combination and its corresponding competing model, determine if the combination forms a good dynamic summary w.r.t. baseline, if yes add to summary.
return *updated dynamic graph summaries* $VG_1, ..., VG_T$

5 Experiments

In this section, we aim to answer the following questions: Are dynamic graphs well structured, or random and noisy? What structures exist in the dynamic graphs, how do they change over time, and can MANGO detect them? We first run MANGO on synthetic data to evaluate if MANGO can find structures that exhibit dynamic behavior over multiple snapshots. We then run MANGO on *dblp*, a large real world dataset providing bibliographic information on major computer science journals and proceedings, to identify if temporal patterns exist in real world graphs and whether MANGO can find them.

For all our experiments, we use SlashBurn [22] for generating candidate static structures. It is scalable, and designed to extract structures from real-world non-cavemen graphs. We use VoG [24] for generating summary for individual snapshots, and pick `GREEDY'NFORGET` model selection heuristic which greedily selects upto 10000 models. We ignore small structures of <5 nodes.

5.1 Synthetic Experiments

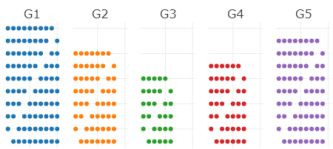
To understand if MANGO can detect the presence of structures over multiple snapshots and capture their evolution through time, we first consider data with known ground truth. We create two synthetic datasets. In the first setting, we create a dynamic graph with 10 timesteps, with each timestep having $n = 10000$ nodes. We randomly populate the snapshots so that they have an edge density of 0.1% ($\sim$50000 edges). We then create 25 structures (5 of each type in our vocabulary Ω), and randomly put each structure in 5 different timesteps. MANGO identifies all 25 structures, and also identifies the 5 timesteps each of them is present in.

When finding Consistent graphs over multiple timesteps, one important advantage our method has over TimeCrunch [5] is that MANGO can populate missing structures in all other timesteps provided it finds them in at least one timestep. As we use VoG [24] as our static summarizer, we have observed that VoG often misses very small patterns (between 5-7 nodes). If VoG, however, fails to report even one instance of the pattern, our algorithm cannot show it as well.

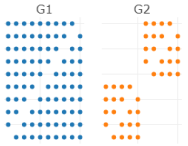
Our second setting captures the dynamic behavior of structures. We use a similar setting as above, but now specifically plant four structures to capture their dynamic behavior. We first plant a clique that shrinks initially and then grows. We then plant a clique that splits into two smaller cliques. We also do a reverse of the previous case and merge two smaller cliques into a larger clique. We use cliques for split/merge for a more understandable behavior. Finally, we plant a bipartite-core that changes into a clique in the next timestep. Our algorithm is able to detect all four scenarios, as shown in Fig. 2.

5.2 Experiments on *dblp*

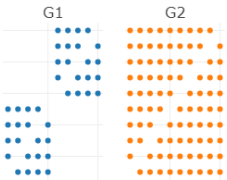
The synthetic datasets were ideally suited for our algorithm. We now explore if MANGO fares well on a large real-world dynamic graph. We run MANGO on the



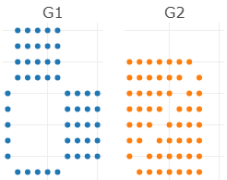
(a) A full clique shrinking and then growing



(b) A full clique splits into two smaller cliques



(c) Two cliques merge into a single large clique



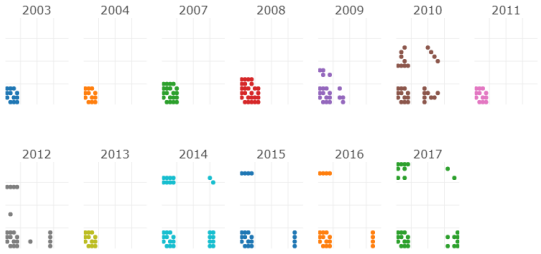
(d) A bipartite-core changes into a clique

Fig. 2: **Results of Mango on synthetic data**: Each dot signifies an edge in the adjacency tensor. In Fig. a) Mango detects a sequential growing/shrinking clique over 5 timesteps. In Fig. b) Mango detects a split of a clique into two cliques. In Fig. c) Mango detects a merge of two smaller cliques into a large clique. In Fig. d) Mango detects a change from a bipartite-core to a clique.

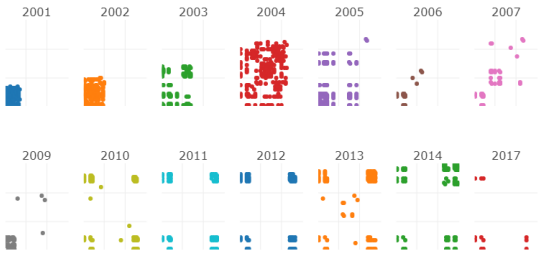
dblp dataset, a publicly available computer science bibliography showing which author has published with whom. We used *Inproceedings* data from 2001 to 2018, with 1.25 million authors and 7.7 million unique author-author collaborations. We now discuss a few of our findings.

Corresponding to many single instances of joint publications, Mango finds a lot of Oneshots in *dblp*. Mango also finds a large number of structures that stretch over multiple timesteps. Mango found over 450 structures that exist in 5 or more timesteps, and over 900 structures that exist in 4 or more timesteps. Mango found one structure, a sporadic growing/shrinking clique encompassing over 60 authors in biomedical engineering, including Wolfgang Birkfellner, Helmar Bergmann and Michael Figl, spanning over 14 timesteps. Mango also found a sporadic growing/shrinking clique centered around Vincenza Carchiolo, Michele Malgeri, Giuseppe Mangioni and Alessandro Longheu, authors in Embedded Systems, spanning over 13 timesteps. As you can see in Fig. 3a and Fig. 3b, not all edges are relevant in the structure, and our model grows/shrinks the structure to encode the non-relevant edges as errors.

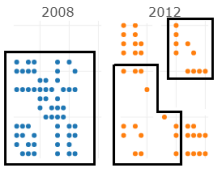
Fig. 3: **Results of Mango on *dblp***: We show some results of MANGO on *dblp* and its ability to capture dynamic behavior across timesteps. Each dot signifies presence of an edge in the adjacency matrix, which has been truncated to show only relevant nodes.



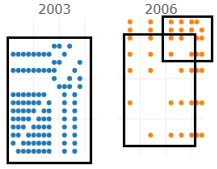
(a) A group of authors in Embedded Systems publishing together across 13 timesteps, the group grows from a small clique in 2003.



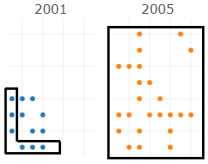
(b) A group of authors in Biomedical Engineering publishing together across 14 timesteps, the group grows from a small clique in 2001, and shrinks around 2006, before growing again.



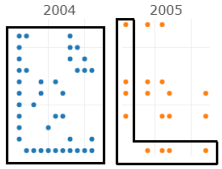
(c)A large clique splits into 2 smaller cliques.



(d) A large clique splits into 2 overlapping cliques.



(e) Structure changes type from star to clique



(f) Structure changes encoding from clique to star. Note that rest of the nodes are encoded as errors

We also observe many instances of dynamic changes to structures which confirms our intuition that not all structures are consistent, but they change over time. While the growing/shrinking of structures is fairly common, we often see structures changing types, see Fig. 3c and Fig. 3d, and some instances of structures splitting/merging, see Fig. 3e and Fig. 3f. It's important to note that given we are not altering any of the candidates found by our static summarizer VoG [24], and letting MDL pick a combination of these candidates that work best, not all edges fit well. In future, we can look into stitching candidates together so as to better fit the adjacency tensor.

6 Conclusion

We studied the problem of dynamic graph summarization. Specifically, we formalized the problem of how patterns in dynamic graphs evolve over time as *minimizing the encoding cost* of a dynamic graph from a compression standpoint. We proposed a heuristic, MANGO, that identifies candidates and generates a dynamic summary over time. Finally, we shared preliminary results on synthetic graphs and real world data.

It is important to note that this is an in-progress work and we are working on numerous facets of the problem. We have started formal evaluation and comparison with similar methods on other real-world datasets like *dblp*, *Enron* and *Honeynet*. While the method is designed to be parallelizable, we haven't performed a detailed performance study, and the algorithm might undergo further finetuning based on our analysis. We are also working on other aspects like determining similarity between graphs, and looking at other extensions like finding anomalous snapshots.

References

1. D. Blandford C. Faloutsos D. Chakrabarti, Y. Zhan and G. Blelloch. Netmine: New mining tools for large graphs. In *SDM Workshop on Link Anal., Count. Terror. and Priv.*, 2014.
2. G. Karypis and V. Kumar. Multilevel -way hypergraph partitioning. In *DAC, pages 343348*, 1999.
3. A. Sridharan S. Machiraju B. A. Prakash, M. Seshadri and C. Faloutsos. Eigenspokes: Surprising patterns and community structure in large graphs. In *PAKDD*, 2010.
4. U. Kang and C. Faloutsos. Beyond caveman communities: Hubs and spokes for graph compression and mining. In *ICDM*, 2011.
5. Christos Faloutsos Neil Shah; Danai Koutra; Tianmin Zou;Brian Gallagher. Timecrunch: Interpretable dynamic graph summarization. In *KDD*, 2015.
6. B. Gallagher N. Shah, A. Beutel and C. Faloutsos. Spotting suspicious link behavior with fbox: An adversarial perspective. In *ICDM*, 2014.
7. D. S. Modha D. Chakrabarti, S. Papadimitriou and C. Faloutsos. Fully automatic cross-associations. In *KDD, pages 7988*, 2004.

8. M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. In *Physical review E,69(2):026113*, 2004.
9. R. Lambiotte V. D. Blondel, J.-L. Guillaume and E. Lefebvre. Fast unfolding of communities in large networks. In *Journal of Statistical Mechanics: Theory and Experiment, 2008(10):P10008*, 2008.
10. S. Mallela I. S. Dhillon and D. S. Modha. Information-theoretic co-clustering. In *In Proc. 9th KDD, pages 8998*, 2003.
11. G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *VLSI design, 11(3):285300*, 2000.
12. B. Kulis and Y. Guan. Graclus - efficient graph clustering software for normalized cut and ratio association on undirected graphs. 2008.
13. A. B. Kahng C. J. Alpert and S.-Z. Yao. Spectral partitioning with multiple eigenvectors. In *Discrete Applied Mathematics, 90(1):326*, 1999.
14. D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. In *arXiv preprint cs/9402102*, 1994.
15. D. Polshakov S. Parthasarathy R. Jin, C. Wang and G. Agrawal. Discovering frequent topological structures from graph datasets. In *KDD, pages 606611*, 2005.
16. C. C. Aggarwal and P. S. Yu. Online analysis of community evolution in data streams. In *SDM*, 2005.
17. S. Papadimitriou J. Sun, C. Faloutsos and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD, pages 687696*, 2007.
18. S. Gunnemann C. Faloutsos P. Basu A. Swami E. E. Papalexakis M. Araujo, S. Papadimitriou and D. Koutra. Com2: Fast automatic discovery of temporal (comet) communities. In *PAKDD, pages 271283*, 2014.
19. J. Leskovec D. Mladenic J. Ferlez, C. Faloutsos and M. Grobelnik. Monitoring network evolution using mdl. In *ICDE*, 2008.
20. D. Jiang J. Pei and A. Zhang. On mining cross-graph quasi-cliques. In *KDD, pages 228238*, 2005.
21. M. Kliger K. S. Xu and A. O. Hero III. Tracking communities in dynamic social networks. In *SBP, pages 219226*, 2011.
22. U. Kang and C. Faloutsos. Beyondcaveman communities: Hubs and spokes for graph compression and mining. In *ICDM, pages 300309*, 2011.
23. A. Hartikainen H. Toivonen, F. Zhou and A. Hinkka. Compression of weighted graphs. In *KDD, pages 965973*, 2011.
24. J. Vreeken D. Koutra, U. Kang and C. Faloutsos. Vog: Summarizing and understanding large graphs.
25. Annika; Bhm Christian; Plant Claudia Goebl, Sebastian; Tonch. Megs: Partitioning meaningful subgraph structures using minimum description length. In *ICDM*, 2016.
26. Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.
27. Ming Li and Paul Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*. 01 1997.
28. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.
29. Pauli Miettinen and Jilles Vreeken. Mdl4bmf: Minimum description length for boolean matrix factorization. *ACM Trans. Knowl. Discov. Data*, 8(4):18:1–18:31, October 2014.