Noname manuscript No.
(will be inserted by the editor)

Online Summarization of Dynamic Graphs using Subjective Interestingness for Sequential Data

Sarang Kapoor[1] · **Dhish Kumar Saxena**[2] · **Matthijs van Leeuwen**[3]

Received: date / Accepted: date

Abstract Many real-world phenomena can be represented as dynamic graphs, i.e., networks that change over time. The problem of dynamic graph summarization, i.e., to succinctly describe the evolution of a dynamic graph, has been widely studied. Existing methods typically use objective measures to find fixed structures such as cliques, stars, and cores. Most of the methods, however, do not consider the problem of *online* summarization, where the summary is incrementally conveyed to the analyst as the graph evolves, and (thus) do not take into account the knowledge of the analyst at a specific moment in time.

We address this gap in the literature through a novel, generic framework for subjective interestingness for sequential data. Specifically, we iteratively identify atomic changes, called 'actions', that provide most information relative to the current knowledge of the analyst. For this, we introduce a novel *information gain* measure, which is motivated by the minimum description length (MDL) principle. With this measure, our approach discovers compact summaries without having to decide on the number of patterns. As such, we are the first to combine approaches for data mining based on subjective interestingness (using the maximum entropy principle) with pattern-based summarization (using the MDL principle).

We instantiate this framework for dynamic graphs and dense subgraph patterns, and present DSSG, a heuristic algorithm for the online summarization of dynamic graphs by means of informative actions, each of which represents an interpretable change to the connectivity structure of the graph. The experiments on real-world data demonstrate that our approach effectively discovers informative summaries. We conclude with a case study on data from an airline network to show its potential for real-world applications.

Keywords Graph Summarization · Maximum Entropy Principle · Subjective Interestingness · Dynamic Graphs

[1] Department of Computer Science and Engineering, Indian Institute of Technology, Roorkee, India; skapoor@cs.iitr.ac.in
[2] Department of Mechanical and Industrial Engineering, Indian Institute of Technology, Roorkee, India; dhishfme@iitr.ac.in
[3] Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands; m.van.leeuwen@liacs.leidenuniv.nl

1 Introduction

Many real-world phenomena, including interactions between people (e.g., social media, e-mail), web browsing, transport and logistics operations, and asset management, can be modelled in terms of the relationships between entities. That is, the corresponding data can be naturally represented as a network or *graph*, where vertices represent the entities and edges represent their relationships. When these relationships change over time, the graphs are called *dynamic graphs*.

The problem of *static graph summarization* has been widely studied, e.g., to efficiently store large volumes of data (Navlakha et al., 2008); improve query efficiency (LeFevre and Terzi, 2010); visualize large graphs (Koutra et al., 2014); and provide high-level descriptions (Goebl et al., 2016). Some of the popular methods rely on compression (Koutra et al., 2014), aggregation of vertices/edges (LeFevre and Terzi, 2010), or finding meaningful patterns (Goebl et al., 2016).

The need to incorporate the temporal dimension has led to the introduction of the problem of *dynamic graph summarization*. Lately, this problem has gained much attention. Here, the focus is on finding a minimal set of temporal structures that describe a dynamic network or graph. A typical way to achieve this is by considering a dynamic network as a sequence of static graph states/snapshots (Sun et al., 2007; Shah et al., 2015; Adhikari et al., 2017) and subjecting those to static graph summarization methods. Such sequences of static graphs, constructed by segmenting a dynamic graph into different states, can be referred to as *sequential data*. For instance, the method proposed by Shah et al. (2015), namely TimeCrunch, extends VoG, a method for static graph summarization by Koutra et al. (2014). It creates a summary by stitching together the graph structures found in different snapshots while minimizing the global description length of the dynamic network. TimeCrunch uses a predefined vocabulary of graph structures, including cliques, stars, cores, and bipartite cores.

Most existing methods, however, do not consider the problem of *subjective online summarization*, where the summary is iteratively and incrementally conveyed to the analyst as the graph evolves. In that, the analyst is progressively updated on all changes up to the current state of the network, relative to his/her prior knowledge. This problem has two key characteristics that differentiate it from posthoc summarization and therefore require a different approach. First, at any state, *it is only possible to use data that has been observed until this very moment*; it is impossible to use parts of the dynamic graph that lie in the future. Second, each change that is observed and communicated to the analyst should be *relative to what that analyst already knows about the graph.*

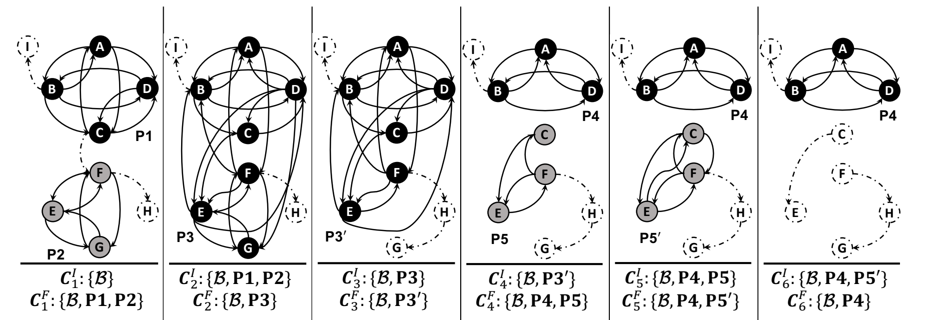
One motivation for such an approach comes from airline network analysis, where vertices represent airports and (directed) edges represent operating flights or routes between two airports. As the edges in an airline network change with time, it can be considered as a dynamic network. Here, an analyst may be interested in learning the *informative changes*, for example, as to how the traffic load is changing in real-time between different airports. An airline schedule is generated based on comprehensive knowledge on air traffic load management (Bazargan, 2016). Hence, a domain analyst may well have prior knowledge/expectation at the block-hour level, of the total number of routes operated by an airline, total number of flights, number of unique routes from each airport, or even the densely connected set of airports. However, delays are a reality, as the schedules are not necessarily robust enough to perfectly factor and accommodate them. Hence, a compact and *subjective online summarization* bears real-time utility for airliners. It is *critical to note* that the application and utility of this approach is not limited to airline domain but spans across many other real-world scenarios, including evolving co-authorship network, co-actor network, and interaction network.

Our first significant contribution is the introduction of a novel, generic framework for subjective interestingness for sequential data. For this, we build on previous work by De Bie (2011), who first introduced a formalization of subjective interestingness for exploratory data mining, in which the analyst's prior beliefs are modelled as constraints and a background distribution—representing the current knowledge of the analyst—is derived using the maximum entropy principle. The novelty of our framework for sequential data is two-fold. First, the patterns that we define, called 'actions', represent atomic changes to the data that provide information relative to the current knowledge of the analyst. Second, we introduce a novel *information gain* measure that is motivated by the minimum description length (MDL) principle (Grünwald, 2007). With this measure, our approach can automatically discover compact summaries without having to decide on the number of patterns. As such, we are the first to combine approaches for data mining based on subjective interestingness (using the maximum entropy principle) with pattern-based summarization (using the MDL principle).

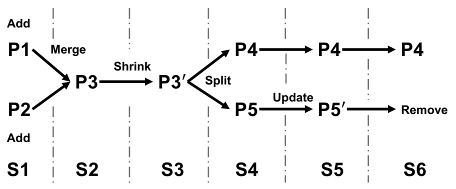
Our second significant contribution is the instantiation of this generic framework for dynamic graphs. As van Leeuwen et al. (2016) instantiated subjective interestingness for dense subgraph discovery from (static) graphs, indeed we here build on their results. The concrete actions that we define, include `add`, `remove`, `update`, `shrink`, `split`, and `merge`. An instance of each of the action types is presented in Figures 1a-1f, for a toy example depicting an evolving airline network. Each of these actions adds, updates, and/or removes one or more dense subgraphs to/in/from the current summary, represented by set $\mathbf{C}_s$ for each state s. The set $\mathbf{C}_s$ comprises of the analyst's prior beliefs (represented by $\mathcal{B}$) and the dense subgraphs as patterns (represented by $\mathbf{P_i}$). In Figures 1a-1f, we indicate the initial summary $\mathbf{C}_s^I$ and final summary $\mathbf{C}_s^F$ after performing the actions in each state. By iteratively communicating these actions to the analyst, the analyst learns about the relevant changes in the graph (as shown in Figure 1g) *relative to what they already know.* The use of our information measure ensures that we always communicate actions that provide more information about the data than that is required to describe the patterns and corresponding actions, effectively making sure that the analyst always gains information.

Our third and final significant contribution is DSSG, a heuristic algorithm for the online summarization of dynamic graphs by means of iteratively discovering actions. Guided by the *information gain* criterion, it always considers all possible types of actions but only returns that action that provides the largest gain.

The remainder of the paper is organized as follows. The relevant literature is summarized in Section 2, followed by notation and preliminaries in Section 3. Our framework for subjective interestingness for sequential data and its online summarization is presented in Section 4.1 and Section 4.2, respectively, leading to the introduction of the problem of online summarization of dynamic graphs in Section 4.3. In this context, the DSSG algorithm is presented in Section 5. The experimental results on publicly available real-world datasets are discussed in Section 6, followed by a case study in the airline domain in Section 7. Important


P1
P2
P3
P3′
P4
P5
P5′
C_1^I: {$\mathcal{B}$}
C_1^F: {$\mathcal{B}$, P1, P2}
C_2^I: {$\mathcal{B}$, P1, P2}
C_2^F: {$\mathcal{B}$, P3}
C_3^I: {$\mathcal{B}$, P3}
C_3^F: {$\mathcal{B}$, P3′}
C_4^I: {$\mathcal{B}$, P3′}
C_4^F: {$\mathcal{B}$, P4, P5}
C_5^I: {$\mathcal{B}$, P4, P5}
C_5^F: {$\mathcal{B}$, P4, P5′}
C_6^I: {$\mathcal{B}$, P4, P5′}
C_6^F: {$\mathcal{B}$, P4}

(a) S1: Add (b) S2: Merge (c) S3: Shrink (d) S4: Split (e) S5: Update (f) S6: Remove



(g) Patterns and their changes over time.

Fig. 1 Toy example showcasing an evolving graph over six states (S1-S6), as summarized by background information $\mathcal{B}$ and patterns P1-P5′. (a-f) In each state s the initial and final summary are represented by $\mathbf{C}_s^I$ and $\mathbf{C}_s^F$, respectively; (g) Patterns P1-P5′ and corresponding add/merge/shrink/split/update/remove actions can be used to summarize the six consecutive states of the dynamic graph as depicted in a-f.

features of the proposed framework, key observations, limitations and future scope are discussed in Section 8, after which we conclude in Section 9.

2 Related Work

We divide the relevant literature into the following categories: static graph mining; static graph summarization; dynamic graph mining; and dynamic graph summarization. The dynamic graph summarization category is most closely related to our work; we discuss the other categories for completeness.

Static Graph Mining. Dense subgraph mining is a well-researched problem. The terms cliques, quasi-cliques (Abello et al., 2002; Matsuda et al., 1999), k-cores (Seidman, 1983), k-plex (Seidman and Foster, 1978), kD-cliques (Luce, 1950) and k-club (Mokken, 1979) in static graphs have been systematically defined and explored in the literature. Recent work on identifying quasi-cliques includes Tsourakakis et al. (2013); Veremyev et al. (2016), while Wu and Hao (2015) summarize all methods for solving the maximum clique problem. Although these measures to identify graph structures are objective, van Leeuwen et al. (2016) argued that the interestingness of each graph structure or pattern is subject to prior information in most applications. On similar lines, Bendimerad et al. (2020) defined subjectively interesting attributed subgraphs. In line with ideas given by van

Leeuwen et al. (2016) and Bendimerad et al. (2020), we also consider the analyst's prior beliefs.

Another popular sub-category of static graph mining is clustering or partitioning of the graph. Most of those methods focus on discovering splits, cuts, or partitions in a graph to identify different regions or communities of interest using spectral partitioning (Alpert et al., 1999), min-max cut (Ding et al., 2001), minimum cut trees (Flake et al., 2004), betweenness measures (Newman and Girvan, 2004), or modularity maximization (Newman, 2006). These methods cover the graph as a whole, while pattern mining in graph data restricts the knowledge discovery to some areas of interest.

Static Graph Summarization. The idea of static graph summarization is to compress a graph (Navlakha et al., 2008; Koutra et al., 2014) or aggregate nodes/edges in a graph (LeFevre and Terzi, 2010; Toivonen et al., 2011; Goebl et al., 2016). It is found to improve query efficiency (LeFevre and Terzi, 2010), speed up clustering algorithms (Toivonen et al., 2011), effectively compress a graph dataset (Navlakha et al., 2008), and provide better visualization (Koutra et al., 2014) of a graph dataset. Koutra et al. (2014) describe a graph by identifying structures using a predefined vocabulary of graph structures such as stars, full & near cliques, full & near bipartite cores, and chains, which minimizes the total encoded length of the graph along with the model (based on the minimum description length principle). Another popular objective of static graph summarization is to find influential dynamics in a network through patterns (Goebl et al., 2016). These patterns provide a high-level description of a graph and are considered relevant and informative in the case of real datasets such as social networks, where information propagation is an essential characteristic of the data. Cook and Holder (1994) *subjectively* summarize a graph by providing a hierarchical description of structural regularities guided by the background knowledge in terms of rules, including compactness, connectivity, coverage and other types of domain-dependent rules. Similar to our proposed approach, the authors also combine the concept of minimum description length with background knowledge. However, we model background knowledge using constraints and the maximum entropy principle.

Dynamic Graph Mining. This category covers methods that identify temporal graph patterns in a dynamic network. Rozenshtein et al. (2017) study interaction networks to find dense and temporally compact patterns. The authors introduce the k-Densest episode identification problem on temporal graphs (Rozenshtein et al., 2018), where an episode is defined as a pair of a time interval and a subgraph. Galimberti et al. (2018) propose the idea of maximal span-cores and span-cores decomposition of temporal networks.

Dynamic Graph Summarization. This category is different from dynamic graph mining: graph summarization methods identify structures and evolution that provide a *succinct description* of a network, while graph mining methods identify *all possible patterns* in the network. As our proposed method fits this category, Table 1 shows an overview of both existing methods and ours; we will elaborate on this comparison in the last paragraph of this section.

GraphScope (Sun et al., 2007) was one of the very first methods that focused on summarizing temporal graphs. It partitions the graph into bipartite cores and cliques. Simultaneously, by detecting the change in encoding cost of graph segment upon presentation of a new graph with the evolution in the state, segments are identified.

Com^2 (Araujo et al., 2014) identifies temporal edge-labelled communities in a graph and uses the minimum description length (MDL) principle with Canonical Polyadic (CP) or PARAFAC decomposition. TimeCrunch (Shah et al., 2015) also uses the MDL principle to summarize a temporal graph. The authors identify graph structures, using the vocabulary of graph structures given by Koutra et al. (2014), along with their corresponding temporal presence in terms of one-shot, periodic, flickering, and ranged. Adhikari et al. (2017) summarize a dynamic network by aggregating nodes into supernodes and time pairs into 'super time'. This method creates a flattened graph (static) after aggregation. Each of these methods concerns an instance of MDL-based dynamic graph compression (either lossy or lossless), but none of them directly summarizes how a dynamic graph changes and evolves.

Various methods in the literature have directly or indirectly addressed the problem of summarizing the evolution of a dynamic graph. You et al. (2009) captures repeated addition and removal of subgraphs between two consecutive graph snapshots in a dynamic graph. Scharwächter et al. (2016) proposed to find frequent structural changes, such as triadic closure and homophilic rewiring, in the form of evolution rules. Ahmed and Karypis (2015) summarize graph evolution by capturing co-evolving relational motifs, which occur when all or a majority of the occurrences of a relational pattern—or motif—evolve similarly over time. Robardet (2009) proposed to capture the evolution of isolated pseudo-cliques over time by means of a sequence of five temporal events, including formation, dissolution, growth, diminution and stability.

Similarly, Ahmed and Karypis (2012) proposed to epitomize an evolving graph by identifying Evolving Induced Relational States (EIRS). The authors defined EIRS as a sequence of Induced Relational States (IRS), which are a set of vertices that remain connected by similar edges having the same direction and label for several consecutive snapshots (based on a threshold). In EIRS, the time interval of each IRS cannot overlap with other IRS and has several or at least a certain number of common vertices. Lin et al. (2011) focus on discovering evolving communities by analyzing the dynamic interactions between vertices by representing the multi-dimensional and multi-relational characteristics as a relational hypergraph called a 'metagraph'. Another recent method based on TimeCrunch (Shah et al., 2015) that aims to capture the evolution of graph structures is given in the preliminary work by Saran and Vreeken (2019). They capture evolving graph patterns by capturing dynamic events such as growth, split, merge, and change in structure type (e.g., from clique to star) of a pattern. Based on their characteristics, these methods can be referred to as methods for discovering evolving graph patterns.

All methods mentioned in this category thus far are defined for a 'fixed' dynamic graph, i.e., over a fixed time interval, and not for a 'streaming' dynamic graph that is generated on-the-fly and should also be analysed on-the-fly, where the summary should change upon the presentation of a new snapshot of a graph. In other words, these methods do not support *online summarization*. Recent methods for online dynamic graph summarization, discussed next, include Tang et al. (2016); Khan and Aggarwal (2016); Qu et al. (2016); Tsalouchidou et al. (2020).

Tang et al. (2016) and Khan and Aggarwal (2016) generate a graphical sketch of a dynamic graph, aggregating vertices and edge weights, which is updated after each snapshot of a graph sequence. These graphical sketches are useful to improve the efficiency of graph-based queries. Qu et al. (2016) summarize a diffusion network, i.e., a dynamic graph where information propagates with time,

Table 1 Comparison of dynamic graph summarization methods. The four rightmost columns show whether a method supports: **OS**—Online Summarization; **IS**—Incremental Summarization; **EP**—Evolving Patterns; **AS**—Automatic selection of summary Size.

Paper	**Algorithm**	**Summary type**	**Selection criterion**	**Pattern type/structure**	**OS**	**IS**	**EP**	**AS**
Sun et al. (2007)	GraphScope	Temporal graph segments	MDL	Bipartites, cliques	✓	✗	✗	✓
Araujo et al. (2014)	COM^2	Edge labelled communities	(Lossy) MDL with tensor decomposition	Stars, bipartites, tiny groups	✗	✗	✗	✓
Shah et al. (2015)	TimeCrunch	Patterns with temporal presence	MDL	Stars, bipartites, cliques, chains	✗	✗	✗	✗*
Adhikari et al. (2017)	NetCondense	Condensed flattened network	Node and time pair aggregation measure	Aggregated node and time pairs	✗	✗	✗	✗
You et al. (2009)	—	Graph rewriting rules	Structural changes	Compressed subgraphs	✗	✗	✗	✗
Scharwächter et al. (2016)	EVOMINE	Evolution rules	Embedding based and event base support	Triadic closure, homophilic rewiring	✗	✗	✓	✗
Ahmed and Karypis (2012)	EIRS	Evolution paths of stable relational states	Maximal evolving induced relational state	Induced relational states	✗	✗	✓	✗
Ahmed and Karypis (2015)	CRMminer	Frequent co-evolutions	User defined minimum support	Co-evolving relational motifs	✗	✗	✓	✗
Lin et al. (2011)	MetaFac	Emergent communities	Metagraph factorization	Communities	✓	✗	✓	✗
Saran and Vreeken (2019)	Mango	Evolving patterns with temporal presence	MDL	Stars, bipartites, cliques, chains	✗	✗	✓	✗*
Tang et al. (2016)	TCM	Condensed graph stream	Aggregation of nodes/ edges	Graph sketch	✓	✗	✗	—
Khan and Aggarwal (2016)	gMatrix	Condensed graph stream	Hash-mapping of nodes	3D sketch	✓	✗	✗	—
Qu et al. (2016)	OSNet	Set of interesting subgraphs of cascades	proScope & proRadius based interestingness	Spreading tree	✓	✗	✓	✗
Tsalouchidou et al. (2020)	SDGM	Condensed network	k-partitions of nodes	Dense micro-clusters	✓	✗	✗	✗
This paper	DSSG	Informative evolving patterns	Maximum entropy and MDL based	Changes in dense subgraphs of any shape	✓	✓	✓	✓

* The size of the summary is dependent on the size of candidate structures generated in each static snapshot of the given dynamic graph, which is not necessarily automatic.

by discovering spreading trees (n-ary) as cascades, which grows with a change in state. Recently, Tsalouchidou et al. (2020) proposed the Scalable Dynamic Graph summarization Method (SDGM) to generate an online summary by extending the static graph summarization approach of LeFevre and Terzi (2010). Although these methods provide online summarization, they do not summarize informative state-to-state relative changes in a dynamic graph. That is, they do not provide *incremental summaries*, where each relative change in the structure of the graph is summarized and communicated to the analyst step by step.

To bridge this gap in the literature, we consider the problem of discovering informative changes in a streaming dynamic graph in an incremental manner. As we are interested in finding all informative changes, we require our method to automatically determine the number of returned patterns. To this end we propose to identify subgraphs that maximally deviate from the current knowledge of the analyst. For this we build on the notion of subjective interestingness proposed by De Bie (2011). To the best of our knowledge, we are the first to consider the problem of subjective, incremental, online graph summarization. This is corroborated by the qualitative comparison in Table 1, which shows the relevant characteristics for all dynamic graph summarization methods discussed in this section.

Since we propose to summarize a dynamic graph by means of dense patterns, we will adapt TimeCrunch (Shah et al., 2015) and SDGM (Tsalouchidou et al., 2020) to establish two baseline methods for empirical comparison in Section 6.

3 Preliminaries

This section defines the notation adopted in this paper, and briefly describes the two most closely related works on which we build in this paper. These are 1) the framework for FORmalizing Subjective Interestingness in Exploratory Data mining (FORSIED) introduced by De Bie (2011), and instantiated for different types of data and patterns; and 2) the work on Subjective interestingness of SubGraph patterns (SSG) in static graphs by van Leeuwen et al. (2016).

3.1 Data and Notation

A *rectangular dataset* is a matrix $\mathbf{D} \in \mathbb{D}^{M \times N}$, where the dimension of the dataset is given by $M \times N$ and $\mathbb{D}$ is the domain of an individual cell. A (simple) *graph* is denoted as $G = (V, E)$, where V is a set of vertices and E is a set of edges such that $u, v \in V$ for each edge $(u, v) \in E$. Its adjacency matrix is a rectangular dataset and hence, represented by $\mathbf{D} \in \mathbb{D}^{|V| \times |V|}$, where $\mathbb{D} = \{0, 1\}$.

A *dynamic* (rectangular) *dataset* $\mathbf{D}_T$ changes with time, where T is the timespan of the dataset. This time interval can be segmented into several consecutive intervals, where each interval $t = (t^b, t^f) \subset T$ represent a state s, such that t^b is the begin time and t^f is the finish time. For any two consecutive states, s and $s + 1$, time t^f_s is equal to time t^b_{s+1}. Thus, a sequence of snapshots $\mathbf{D}_1, \ldots, \mathbf{D}_S$ is observed, indexed by state $s \in \{1, \ldots, S\}$, where S is the total number of states. Note that, in a sequence of snapshots, each D_s is a static rectangular dataset, such that $\mathbf{D}_s \in \mathbb{D}^{M \times N}$. We refer to such a sequence of snapshots as *sequential data*.

A *dynamic graph*, denoted $G_T = (V, E_T)$, is a graph in which each edge is present for a given period within time interval T, i.e., E_T is the set of edges that occur in time interval T. More specifically, each $e = (u, v, t^b, t^f) \in E_T$ defines an edge $u, v \in V$ that appears at start time t^b and continues to exist until it disappears at finish time t^f. Again, the time interval T can be segmented into several intervals, as seen earlier for dynamic datasets. This assumption implies that each $t \subset T$ defines a static state s of the dynamic graph, that is essentially a (simple) graph: each edge either exists or not. We denote the dynamic graph projected to its graph corresponding to a fixed time t by G_s, and its corresponding adjacency matrix by $\mathbf{D}_s \in \mathbb{D}^{|V|\times|V|}$, such that $\mathbb{D} = \{0, 1\}$. Hence, a dynamic graph, G_T can be represented as a sequential dataset $\mathbf{D}_T$, with a sequence of static graph snapshots $G_1, \ldots, G_S$ and a corresponding sequence of adjacency matrices $\mathbf{D}_1, \ldots, \mathbf{D}_S$.

Notably, even when time is not discrete, one can easily discretize it by segmenting it into equal-length intervals (e.g., seconds, minutes, ...). As we will see, the length of these intervals determines the granularity at which the approach will identify changes in the data. For instance, in the airline case, it is implausible that (relevant) changes will occur within seconds or even minutes, hence, it may be reasonable to segment time in hours.

3.2 Subjectively Interesting Patterns in Static Graphs

Informally, the FORSIED framework (De Bie, 2011) defines subjective interestingness of a pattern as the information it provides with regard to the analyst's expectations (or prior knowledge), normalized by its complexity. Given a dataset $\mathbf{D}$, the analyst's *background distribution* P^*, is the distribution that maximizes entropy, is given by

$$P^* = \underset{P(\mathbf{D})}{\operatorname{argmax}} - \sum_{\mathbf{D}\in\mathcal{D}} P(\mathbf{D}) \log(P(\mathbf{D})), \tag{1}$$

$$\text{s.t.} \underset{\mathbf{D}\in\mathbb{D}}{\mathbb{E}}[f_i(\mathbf{D})] = \sum_{\mathbf{D}\in\mathbb{D}} P(\mathbf{D}) f_i(\mathbf{D}) = c_i, \forall i, \tag{2}$$

$$\sum_{\mathbf{D}\in\mathbb{D}} P(\mathbf{D}) = 1. \tag{3}$$

The set of constraints enforced in Equation 2 is presented in a generalized form, where each constraint $B_i \in \mathcal{B}$ is a pair consisting of a function f_i over $\mathbf{D}$—as properties of the data—and a corresponding constant c_i, i.e., $B_i = (f_i, c_i)$. The set of constraints $\mathcal{B}$ represents the analyst's prior knowledge or expectations on the data. The exact type(s) of constraints and their interpretation depends on the type and nature of the dataset $\mathbf{D}$.

Next, the interestingness of a pattern θ is defined as the ratio of the pattern's self-information (denoted $\mathcal{SI}$) to its description length (denoted $\mathcal{DL}$). Self-information is the negative log-probability that the pattern is present in the data, i.e., $-\log(P(\theta \in \mathbf{D}))$, while description length is the number of bits required to describe or communicate the pattern to the analyst.

Instantiating these generic concepts for dense subgraph patterns in static graphs, van Leeuwen et al. (2016) defined interestingness $\mathbf{I}$ of a static graph pattern, $\theta = (W, k_W)$, denoting a vertex set W having k_W edges, as[1]

$$\mathbf{I}\left[(W, k_W)\right] = \frac{\mathcal{SI}\left[(W, k_W)\right]}{\mathcal{DL}\left[(W, k_W)\right]} = \frac{n_W \cdot \mathbf{KL}\left(\frac{k_W}{n_W} || p_W\right)}{|W| \cdot \log\left(\frac{1-q}{q}\right) + |V| \cdot \log\left(\frac{1}{1-q}\right)}, \tag{4}$$

where n_W is the number of possible edges in subgraph W, q is a hyperparameter representing the 'expected' probability of a random node to be present in W, and p_W is the probability of the subgraph occurring given background distribution P^*. The latter probability is computed as $p_W = \frac{1}{n_W} \sum_{u,v \in W} p_{u,v}$, where $p_{u,v}$ is the probability that an edge between vertices u and v exists as given by P^*.

Iterative learning. The framework above can be motivated by the observation that *compression equates learning* (Grünwald, 2007): in order to learn as much as possible about the data, the implicit goal of the analyst is to (internally) represent the data using as few bits as possible. This observation implies minimizing $-\log P^*(\mathbf{D})$, i.e., the length of the data encoded by the background distribution. This can be accomplished by changing the analyst's knowledge on $\mathbf{D}$. Here, change in the analyst's knowledge on $\mathbf{D}$ implies that a new set of constraints $\mathbf{C}$ corresponding to each discovered pattern must be constructed, which is used to update the background distribution P^*. Specifically, when a graph pattern is discovered, a constraint is added to ensure that the updated expectations of the analyst conform with the actual number of edges. For instance, when a graph pattern (W, k_W) is presented to the analyst, a new constraint $C_W = (f_W, k_W)$ is added to $\mathbf{C}$, where f_W is a function over W vertices which counts the number of edges, i.e., $f_W(\mathbf{D}) = \sum_{u,v \in W, u<v} \mathbf{D}[u, v]$, and k_W is the actual number of edges in the vertex-induced subgraph of W vertices. Notably, the solution to the following problem provides the updated background distribution (van Leeuwen et al., 2016):

$$P^{*\prime} = \underset{P}{\operatorname{argmin}} \sum_{\mathbf{D}} P(\mathbf{D}) \log\left(\frac{P(\mathbf{D})}{P^*(\mathbf{D})}\right), \tag{5}$$

$$\text{s.t.} \sum_{\mathbf{D}} P(\mathbf{D}) f_W(\mathbf{D}) \geq k_W, \tag{6}$$

$$\sum_{\mathbf{D}} P(\mathbf{D}) = 1. \tag{7}$$

Hence, the analyst can learn everything about the data by iteratively discovering the most interesting pattern and updating the background distribution after each iteration.

4 Proposed Approach

In this section, we introduce our novel framework for subjective interestingness for sequential data, which extends the FORSIED framework but also incorporates crucial changes. We introduce the problem of subjective summarization of sequential

[1] All logarithms in this paper are to the base 2.

data, and to solve this problem we propose the method of online summarization of sequential data. Finally, we instantiate this generic problem for dynamic graphs.

4.1 Subjective Interestingness for Sequential Data

Given a sequential dataset $\mathbf{D}_T$, we consider the setting where an analyst is interested in learning informative patterns about the data as the snapshots unfold in an online fashion. As with static data, the analyst may have prior beliefs about the data already before the first snapshot—these are represented by a set of constraints $\mathcal{B}$.

When the snapshot corresponding to the first state is analyzed, we aim to find a compact set of constraints, i.e., patterns, that—together with the prior beliefs—minimize the negative log-probability of the data, given the implied background distribution. To avoid finding either too many or too complex patterns, we draw inspiration from the minimum description length principle (Grünwald, 2007) and use a two-part code to balance the goodness of fit of the data with the complexity of the constraint set. More precisely, we aim to find a new set of constraints $\mathbf{C}_1$ with corresponding background distribution P_1^* that minimizes $-\log P_1^*(\mathbf{D}_1) + L(\mathbf{C}_1)$, where L is a function that computes the encoded length for any given set of constraints. It is of note that we require an additional set of constraints $\mathbf{C}_1$ other than the existing set of constraints $\mathcal{B}$ to achieve the optimal (feasible) solution of the above problem. The set of constraints $\mathbf{C}_1$ is used to ensure that the knowledge mined by the discovered patterns is reflected in the background distribution P_1^*.

For any consecutive snapshot, we now want to *adapt what the analyst has learned before*; by only providing the analyst with information about changes that have occurred in the data since the previous state, he requires minimal effort, and we obtain a minimal summary. Given the previous, this implies that—for each snapshot s after the first—we need to find a set of constraints $\mathbf{C}_s$ with corresponding background distribution P_s^* that minimizes $-\log P_s^*(\mathbf{D}_s) + L(\mathbf{C}_s|\mathbf{C}_{s-1})$, where L is a function that computes the encoded length for any given set of constraints *given* another set of constraints; i.e., smaller changes require fewer bits.

With the given discussion, we formally introduce the following problem statement.

Problem 1 (Subjective Summarization of Sequential Data) *Given a sequential dataset $\mathbf{D}_T$, i.e., sequence of snapshots $\mathbf{D}_1, \ldots, \mathbf{D}_S$, and prior beliefs $\mathcal{B}$, find:*

- *for $\mathbf{D}_1$: a set of constraints $\mathbf{C}_1$ that minimizes $-\log P_1^*(\mathbf{D}_1) + L(\mathbf{C}_1)$, where P_1^* is computed using constraints $\mathcal{B} \cup \mathbf{C}_1$;*
- *for $\mathbf{D}_s$, with $s \in \{2, \ldots, S\}$: a set of constraints $\mathbf{C}_s$ that minimizes $-\log P_s^*(\mathbf{D}_s) + L(\mathbf{C}_s|\mathbf{C}_{s-1})$, where P_s^* is computed using constraints $\mathcal{B} \cup \mathbf{C}_s$.*

4.2 Online Summarization of Sequential Data

Apart from the fact that optimally solving each iteration of Problem 1 would require to consider a very large search space, i.e., that of all possible constraints sets, we do not want to present unordered sets of constraints to the analyst: this would very likely overwhelm the analyst and therefore cause confusion. Instead, we prefer to present atomic changes to $\mathbf{C}$ to the analyst one by one, as is also done in the framework for static data. We will therefore now derive an approach that heuristically approximates Problem 1 by iteratively looking for the largest changes and communicating those to the analyst immediately.

After each atomic change α, also called *action*, the set of constraints $\mathbf{C}$ is updated to a new set $\mathbf{C}'$, and hence the background distribution P^* is updated accordingly. α reduces the negative log-probability of the data by updating the background distribution, and we define this reduction as Information Content, $\mathcal{IC}$.

Definition 1 (Information Content) Given an action α, and constraint sets $\mathbf{C}$ (original) and $\mathbf{C}'$ (updated), we define the *information content* of α, denoted by $\mathcal{IC}$, as the difference between the length of the data encoded by the background distributions specified by constraint sets $\mathbf{C}$ and $\mathbf{C}'$:

$$\begin{aligned}\mathcal{IC}(\alpha) = \mathcal{IC}(\mathbf{C}'|\mathbf{C}) &= -\log P^*_{\mathbf{C}}(\mathbf{D}) - \left(-\log P^*_{\mathbf{C}'}(\mathbf{D})\right) \\ &= \log P^*_{\mathbf{C}'}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D}),\end{aligned} \tag{8}$$

where P^*_X is the MaxEnt probability distribution given a set of constraints X (i.e., using Equations 1-3).

An *action* on $\mathbf{C}$ can be categorized as one of the following:

1. Addition of a new constraint C, i.e., $\mathbf{C}' = \mathbf{C} \cup \{C\}$,
2. Deletion of a constraint C , i.e., $\mathbf{C}' = \mathbf{C} \setminus \{C\}$,
3. Update of an already present constraint $C \in \mathbf{C}$, i.e., replacing C with a constraint C', and hence $\mathbf{C}' = \mathbf{C} \setminus \{C\} \cup \{C'\}$.

Definition 2 (Description Length) The *description length* of an action α, denoted $\mathcal{DL}(\alpha)$, is defined as the (minimum) number of bits required to encode the changes in the set $\mathbf{C}$ when communicated to the analyst.

Remark 1 Given a set of constraints $\mathbf{C}_{s-1}$, let $\mathcal{A}$ be an ordered set of actions performed on $\mathbf{C}_{s-1}$ to get an updated set $\mathbf{C}_s$, then the encoded length L of $\mathbf{C}_s$ is computed as:

$$L(\mathbf{C}_s|\mathbf{C}_{s-1}) = \sum_{\alpha \in \mathcal{A}} \mathcal{DL}(\alpha). \tag{9}$$

We now have two different quantities associated with each atomic change α, i.e., $\mathcal{IC}$ and $\mathcal{DL}$. Maximizing $\mathcal{IC}$ and minimizing $\mathcal{DL}$ leads to our overall goal of minimizing $-\log P^*_s(\mathbf{D}_s) + L(\mathbf{C}_s|\mathbf{C}_{s-1})$. Thus, we discount $\mathcal{IC}$ with $\mathcal{DL}$ and perform the action with maximal difference at each step. We call this difference *information gain* and denote it by $\mathcal{IG}$.

Definition 3 (Information Gain) Let α be an action that transforms a given set of constraints $\mathbf{C}$ into an updated set $\mathbf{C}'$. Then, the *information gain* $\mathcal{IG}$ on performing α on $\mathbf{C}$ is given by

$$\mathcal{IG}(\alpha) = \mathcal{IC}(\alpha) - \mathcal{DL}(\alpha). \tag{10}$$

The process of online summarization begins with the initialization of background distribution $P^*_{\mathcal{B}}$ using the prior belief(s) $\mathcal{B}$ that an analyst may have. At the start of state 1, no patterns have been discovered yet, i.e., $\mathbf{C}_1 = \emptyset$, which implies $P^*_{\mathcal{B} \cup \mathbf{C}_1} = P^*_{\mathcal{B}}$. Then patterns with maximum $\mathcal{IG}$ (Equation 10) are discovered iteratively and for each such pattern a corresponding constraint C is added to $\mathbf{C}_1$ and hence the background distribution $P^*_{\mathcal{B} \cup \mathbf{C}_1}$ is updated (using Equations 5-7). Note that $\mathbf{C}_1$ is initially an empty set, thus the only action that can be performed on $\mathbf{C}_1$ is the addition of a new pattern. The process continues until no feasible action can be performed on set $\mathbf{C}_1$. Here, a feasible action is any action which satisfies a user-provided criteria, for example, to be in agreement with the MDL principle an action α it is recommended default that α is feasible if $\mathcal{IG}(\alpha) > 0$. The process then moves to the following state. For any state s (except state 1), $\mathbf{C}_s$ is initialized to the final $\mathbf{C}_{s-1}$ and $P^*_{\mathbf{C}_s}$ to the final $P^*_{\mathbf{C}_{s-1}}$. This is followed by iterative actions on $\mathbf{C}_s$ with maximal, $\mathcal{IG}$ until no feasible action can be performed.

4.3 Online Summarization of Dynamic Graphs

The concept of subjective summarization of sequential data can be directly adapted to dynamic graphs by segmenting such a graph into a sequence of static graph snapshots (see Section 3.1). By making the data type more specific, however, we can also instantiate the other components of the generic framework—e.g., actions, prior beliefs, constraints, and description length—with more precise definitions. As discussed earlier, a graph pattern, $\theta = (W, k_W)$ is a subgraph of $W \subset V$ vertices that is connected by k_W edges. Thus, by definition a graph pattern is *connected*, i.e., there exists a path from every vertex to every other vertex. Note that, since we consider graph patterns, the definition of constraints follows the discussion in Section 3.2. Following, we introduce the following problem statement as an instance of Problem 1.

Problem 2 (Subjective Summarization of Dynamic Graphs) *Given a dynamic graph G_T consisting of a sequence of snapshots $G_1, \ldots, G_S$, with $\mathbf{D}_s$ the corresponding adjacency matrix for a state s and prior beliefs $\mathcal{B}$, solve Problem 1 such that each pattern in every set $\mathbf{C}_s$ is a connected subgraph pattern.*

As discussed previously, optimally solving Problem 2 requires to consider a very large number of possible constraint sets. Similarly, we heuristically address Problem 2 by iteratively communicating atomic changes, or *actions*, having maximal $\mathcal{IG}$ to the analyst. Based on the properties of a graph pattern and possible structural changes, we now formalize six specific types of actions which we use to communicate changes on graph data, as initially depicted in Figure 1.

The `add` action communicates a newly discovered subjectively dense subgraph pattern. In Figure 1a, two patterns, **P1** and **P2**, are identified and `add`ed in state S1. A `remove` action deletes a pattern that no longer holds in the current state, i.e., when the pattern is no longer connected and/or its density decreases substantially. An example is shown in Figure 1f, where a sparse pattern **P5′** is `remove`d in state S6—removing a constraint is informative when it has a positive $\mathcal{IC}$.

The other actions are `update`, `merge`, `shrink`, and `split`, which all represent modifications of constraint(s) already present in **C**. When the density of a pattern corresponding to an existing constraint increases, this is communicated via

Table 2 Conditions that must be met to perform an action α on a constraint C present in constraint set $\mathbf{C}$, with initial pattern θ_i, resultant pattern θ_f and density function ρ (defined as the ratio of the number of edges to the maximum possible number of edges in a graph).

Type of α	**is $C \in \mathbf{C}$?**	**$\rho(\theta_i)$ increases?**	**$\rho(\theta_i)$ decreases?**	**is θ_i connected?**	**is θ_f connected?**
`Add`	✗	—	—	—	✓
`Remove`	✓	✗	✓	✗	—
`Update`	✓	✓	✗	✓	✓
`Shrink`	✓	✗	✓	?	✓
`Split`	✓	✗	✓	✗	✓
`Merge`	✓	?	✗	✓	✓

✓: true, ✗: false, ?: may or may not be true, —: not applicable

`update`. Thus, a constraint $C = (f_W, k_W) \in \mathbf{C}$ is replaced by a similar but updated constraint $C' = (f_W, k'_W)$. In Figure 1e, pattern **P5** is updated to pattern **P5′** in state S5, when its density increases compared its density in state S4 (Figure 1d). By applying a `merge` action, two previous patterns are merged to form one new pattern. That is, two constraints $C_i = (f_{W_i}, k_{W_i}), C_j = (f_{W_j}, k_{W_j}) \in \mathbf{C}$ are replaced by a single new constraint $C' = (f_{W_i \cup W_j}, k_{W_i \cup W_j})$, such that the resulting pattern of vertices $W_i \cup W_j$ is connected. An instance is presented in Figure 1b, where two patterns, **P1** and **P2**, are merged to create a new pattern **P3** in S2.

Actions `shrink` and `split` either reduce an existing constraint or decompose one into multiple constraints. A constraint is shrunk when the density of a pattern decreases with the evolution of the graph (see Figure 1c, where pattern **P3** shrinks to form pattern **P3′** in state S3). Similarly, a constraint can be decomposed into multiple new constraints if the pattern corresponding to an original constraint consists of two or more connected components (see Figure 1d, where pattern **P3′** splits into two new patterns **P4** and **P5** in state S4). In `shrink`, the original constraint $C = (f_W, k_W) \in \mathbf{C}$ is replaced by a new reduced constraint $C = (f_{W'}, k_{W'})$ such that $W' \subset W$. In `split`, on the other hand, a constraint $C = (f_W, k_W) \in \mathbf{C}$ is replaced by M new constraints, $C_1 = (f_{W_1}, k_{W_1}), \ldots, C_M = (f_{W_M}, k_{W_M})$, such that $W_1, \ldots, W_M \subset W$ and $W_i \cap W_j = \emptyset, \forall i, j \in \{1, \ldots, M\}$.

The different conditions that must be satisfied for each of the six types of actions to be applicable are summarized in Table 2.

Next, the formulation of information content $\mathcal{IC}$ and description length $\mathcal{DL}$ of each action type is summarized in Table 3. We extend the abstract definition of description length given in the previous section (Definition 2). The description length of an action is the summation of two parts, the first of which encodes the type of action, represented by $type(\alpha)$, and the second of which encodes the details, represented by $details(\alpha)$. For all quantities where the upper limit is not known, we use the universal integer code (Rissanen, 1983), which is given by $\mathrm{L}_{\mathbb{N}}(n) = \log(2.865064) + \log(n) + \log\log(n) \ldots$ and sums over all positive terms. If the upper limit is known then we use the uniform code (Grünwald, 2007), given by $\log(n)$. Note that all logarithms are to base two.

In the description length of α, to describe the type of action we use the uniform code over all possible action types as there is no priority or bias towards any action. Thus, $\mathcal{DL}(type(\alpha)) = \mathcal{T}_a = \log(l)$, as we require $-\log\frac{1}{l}$ bits. Here, $l = 6$ as we have defined six action types above. The computation of $\mathcal{DL}(details(\alpha))$ for

Table 3 Shown are the formulation of Information Content ($\mathcal{IC}$) and Description Length ($\mathcal{DL}$) for each defined atomic change, α.

α	$\mathcal{IC}$	$\mathcal{DL}$
`Add`	$\log P^*_{\mathbf{C}\cup C}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D})$	$\mathcal{T}_a + \mathcal{T}_c + \mathcal{T}_p$
`Remove`	$\log P^*_{\mathbf{C}\setminus C}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D})$	$\mathcal{T}_a + \mathcal{T}_b$
`Update`	$\log P^*_{\mathbf{C}\setminus C\cup C'}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D})$	$\mathcal{T}_a + \mathcal{T}_b + \mathcal{T}_{c'}$
`Shrink`	$\log P^*_{\mathbf{C}\setminus C\cup C'}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D})$	$\mathcal{T}_a + \mathcal{T}_b + \mathcal{T}_{c'} + \mathcal{T}_d + \mathcal{T}_e$
`Merge`	$\log P^*_{\mathbf{C}\setminus\{C_i,C_j\}\cup C'}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D})$	$\mathcal{T}_a + 2\times\mathcal{T}_b + \mathcal{T}_{c'}$
`Split`	$\log P^*_{\mathbf{C}\setminus C\cup\{C_1,\ldots,C_\tau\}}(\mathbf{D}) - \log P^*_{\mathbf{C}}(\mathbf{D})$	$\mathcal{T}_a + \mathcal{T}_b + \mathcal{T}_f + \mathcal{T}_g + \mathcal{T}_h + \mathcal{T}_i$

$\mathcal{T}_a = \log(l)$, $\mathcal{T}_b = \log(|\mathbf{C}|)$, $\mathcal{T}_c = \mathrm{L}_{\mathbb{N}}\,(n_W - k_W + 1)$, $\mathcal{T}_{c'} = \mathrm{L}_{\mathbb{N}}(n_{W'} - k_{W'} + 1)$,
$\mathcal{T}_d = \mathrm{L}_{\mathbb{N}}(|\Psi|)$, $\mathcal{T}_e = \log(|W|) + \log(|W|-1)\cdots + \log(|W| - |\Psi| + 1)$, $\mathcal{T}_f = log(M)$,
$\mathcal{T}_g = \mathrm{L}_{\mathbb{N}}(|W_1|) + \cdots + \mathrm{L}_{\mathbb{N}}(|W_M|))$, $\mathcal{T}_h = \log(|W|) + \log(|W|-1) + \cdots + \log(|W| - x + 1)$
where $x = |W_1 \cup \cdots \cup W_M|$, $\mathcal{T}_i = \mathrm{L}_{\mathbb{N}}(n_{W_1} - k_{W_1} + 1) + \cdots + \mathrm{L}_{\mathbb{N}}(n_{W_M} - k_{W_M} + 1)$,
$\mathcal{T}_p = |W|\log(q) + (|V| - |W|)\log(1-q)$

each action type is shown in Table 3. That is, *details*(`add`) is the summation of the number of bits required to describe the set of vertices ($\mathcal{T}_p = \mathcal{DL}[(W, k_W)]$, see Equation 4, and the number of edges in the corresponding vertex-induced subgraph. Instead of describing the number of edges in a subgraph, we describe the number of edges short in a subgraph when compared to a clique of same number of vertices. That is, for a subgraph having W vertices, n_W is the maximum number of edges possible between W vertices, and k_W is the number of edges, then we describe the difference between n_W and k_W, given by $\mathcal{T}_c$. Thus, a dense subgraph with high number of edges would have smaller description length, which favours discovery of dense subgraph patterns. Note that, the hyperparameter 'q' in $\mathcal{T}_p$ can be used to influence the size of pattern (see Section 3.2).

In `remove`, `update`, `shrink`, and `split`, the index of the constraint to be removed is communicated in $\mathcal{T}_b$ bits. Similarly, in case of `merge` the index of two constraints are communicated in $2 \times \mathcal{T}_b$ bits. Since we only consider the merge of two constraints at a time, the term $\mathrm{L}_{\mathbb{N}}(|2|)$ is omitted. In addition, for all the actions except `remove` the information about the edges is communicated in $\mathcal{T}_{c'}$ bits. In case of `shrink`, terms $\mathcal{T}_d$ and $\mathcal{T}_e$ indicate the number of bits required to describes the number of vertices removed from the original pattern and the removed vertices, respectively. In `split`, the number of resulting constraints is described in $\mathcal{T}_f$ bits, each constraint in $\mathcal{T}_g$ bits, vertices in each constraint in $\mathcal{T}_h$ bits, and information about edges in each component using $\mathcal{T}_i$ bits.

Lemma 1 *For an action α, which updates a set of constraints $\mathbf{C}$ to $\mathbf{C}'$, $\mathcal{IC}(\alpha)$ as defined in Definition 1 is equal to*

$$\mathcal{IC}(\alpha) = \log P^*_{\mathbf{C}'}(\mathbf{R}) - \log P^*_{\mathbf{C}}(\mathbf{R}), \tag{11}$$

where $\mathbf{R}$ is a submatrix of $\mathbf{D}$ given by $\mathbf{R} = \mathbf{D}[W_1, \ldots, W_M; W_1, \ldots, W_M]$, such that W is the set of M vertices covered by the affected constraint(s)[2]*, C_α.*

[2] The affected constraints C_α are those constraints (both original and updated) that are affected by action α. That is, if α transforms $\mathbf{C}$ to $\mathbf{C}'$ the C_α is defined to be all constraints in either $\mathbf{C}$ or $\mathbf{C}'$ that are not in both $\mathbf{C}$ and $\mathbf{C}'$

Proof The proof is straightforward, however, for completeness we provide the following details. In Equation 8, $\log P^*_X(\mathbf{D}) = \sum_{i,j \in V} \log P^*_X(\mathbf{D}_{ij})$ is the sum over all pairs of vertices. These pairs can be categorized into three groups, which are 1) both vertices lie in W, 2) neither of the vertices lie in W, and 3) either (but not both) of the vertices lie in W. It is only in the first case that the probability is updated on performing the action α, while the rest of the probability terms remains unchanged and hence, these terms cancel out each other, i.e., $\log P^*_{\mathbf{C}'}(\mathbf{D}_{ij}) = \log P^*_{\mathbf{C}}(\mathbf{D}_{ij})$. Thus, the result follows. □

By virtue of Lemma 1, we come up with the following result.

Theorem 1 *The complexity of computing information content $\mathcal{IC}$ of an action α is $\mathcal{O}(|W|^2)$, where W is the set of vertices included in C_α.*

Proof The proof follows Equation 11 which is sum over all pair of vertices, $(i, j) : i, j \in W$. Hence, this requires a complexity of $\mathcal{O}(|W|^2)$. □

As discussed above, we solve Problem 2 by iteratively performing that action (of one of the six types defined above) with maximal $\mathcal{IG}$. Thus, we introduce the problem of online summarization of dynamic graphs (Problem 3). Hence, we heuristically unfold Problem 2 by iteratively solving Problem 3 at each step.

Problem 3 (Online Summarization of Dynamic Graphs) Given the current state s, graph snapshot G_s, corresponding adjacency matrix $\mathbf{D}_s$, and current constraint set $\mathbf{C}_s$, perform that action 'α' from the set of all possible actions having maximal information gain $\mathcal{IG}$, such that the pattern(s) obtained after performing 'α' are connected subgraph(s).

4.4 Additional Details

Prior Beliefs. We consider two different types of prior beliefs to constitute the set $\mathcal{B}$, which are direct adaptations of the beliefs proposed by van Leeuwen et al. (2016), as follows:

1. Belief-c: In this case, we model the analyst's knowledge about the total number of edges in the initial snapshot of the data. In other words, the analyst has prior knowledge about the relative edge density of the graph dataset. Solving Equations 1-3, De Bie (2011) showed that P^* turns out to be product of independent Bernoulli distributions for each random variable $a_{u,v}$ and is given by
$$P^*(\mathbf{D}) = \prod_{u<v} \frac{exp((2 \cdot \lambda) \cdot a_{u,v})}{1 + exp(2 \cdot \lambda)}. \tag{12}$$
This distribution is best represented as a matrix $P^* \in [0, 1]^{|V| \times |V|}$ with row and column indices indicating the vertices, such that $p_{u,v} = \frac{exp(2 \cdot \lambda)}{1+exp(2 \cdot \lambda)} = \rho(G_0)$ suggests the probability of $a_{u,v} = 1$, i.e., an edge between vertex u and v.
2. Belief-i: Similarly, here, the user possesses a belief about the individual degree of each vertex in a snapshot of the data. The maximum entropy distribution

turns out to be a similar product of independent Bernoulli distributions, given as

$$P^*(\mathbf{D}) = \prod_{u<v} \frac{exp((\lambda_u + \lambda_v) \cdot a_{u,v})}{1 + exp(\lambda_u + \lambda_v)}, \tag{13}$$

where $p_{u,v} = \frac{exp(\lambda_u+\lambda_v)}{1+exp(\lambda_u+\lambda_v)}$ is the probability of random variable $a_{u,v} = 1$.

Updating the background distribution. When a pattern $\theta = (W, k_W)$ is discovered (through action `add`), a constraint $C = (f_W, k_W)$ is added to the set $\mathbf{C}$, and P^* is updated using Equations 5-7 (van Leeuwen et al., 2016), where the updated P^* is given as

$$P^*(\mathbf{D}) = \prod_{u<v} {p'_{u,v}}^{a_{u,v}} \cdot \left(1 - p'_{u,v}\right)^{1-a_{u,v}}, \tag{14}$$

where

$$p'_{u,v} = \begin{cases} \frac{exp(\lambda_u+\lambda_v+\lambda_W)}{1+exp(\lambda_u+\lambda_v+\lambda_W)} & \textit{if } u, v \in W, \\ \frac{exp(\lambda_u+\lambda_v)}{1+exp(\lambda_u+\lambda_v)} & \textit{otherwise} . \end{cases} \tag{15}$$

Thus, for all pairs $(u, v) : u, v \in W$ a unique Lagrangian multiplier, λ_W is introduced (using the bisection method) upon updating the background distribution. Similarly, if multiple constraints are present in $\mathbf{C}$, then $p'_{u,v}$ is computed as $\frac{exp(\lambda_u+\lambda_v+\sum_{C\in\mathbf{C}:u,v\in W} \lambda_W)}{1+exp(\lambda_u+\lambda_v+\sum_{C\in\mathbf{C}:u,v\in W} \lambda_W)}$. Hence, it is efficient to store only the Lagrangian multipliers and compute the probability whenever required.

If a `remove` action is performed then the corresponding Lagrangian multiplier is removed from the list to update the background distribution. Similarly, for all other actions, first the corresponding Lagrangian multiplier(s) to the original constraint(s) are removed and then using Equations 5-7, new Lagrangian multiplier(s) are computed. Hence, this is an efficient way to update the background distribution.

Feasibility Constraint. In order to provide the user with a concise summary we introduce a feasibility constraints to limit the number of actions performed in each state. That is, we consider an action feasible if the *information gain* is positive, i.e., $\mathcal{IG}(\alpha) > 0$. Although, it may be altered as per user preference, this choice is motivated by MDL principle and ensures that an action always provide more information about the data than that it costs to describe the action.

5 The DSSG Algorithm

In this section, we introduce an algorithm called DSSG, of which the step by step procedure is outlined in Algorithm 1. DSSG is a heuristic approach to solve Problem 2 that works in an iterative manner, solving Problem 3 in each step. The overall procedure of DSSG can be summarized as follows.

DSSG starts with an initial graph snapshot G_0, an initial set of constraints $\mathcal{B}$ (as the analyst's prior belief), and a set of constraint $\mathbf{C}$ (which is usually $\emptyset$ initially). Given this, the maximum entropy distribution P is then computed (Line 2). For each state s (Line 3) actions are performed iteratively to solve Problem 3 (Lines 5-10). The process continues until no action can be performed (Line 14).

Algorithm 1 DSSG

```
1: procedure DSSG($G_T$, $\mathbf{C}$, $\mathcal{B}$)
2:     Compute Maximum Entropy Distribution for $G_0$ given $\mathcal{B}$ as $P$
3:     for each $G_s \in G_T$ do        ▷ here $G_T$ is a sequence of static graphs (snapshots)
4:         repeat
5:             $\mathsf{A} \leftarrow$ EvaluateAdd($G_s, P$)
6:             $\mathsf{R} \leftarrow$ EvaluateRemove($G_s, P, \mathbf{C}$)
7:             $\mathsf{U} \leftarrow$ EvaluateUpdate($G_s, P, \mathbf{C}$)
8:             $\mathsf{S} \leftarrow$ EvaluateShrink($G_s, P, \mathbf{C}$)
9:             $\mathsf{M} \leftarrow$ EvaluateMerge($G_s, P, \mathbf{C}$)
10:            $\mathsf{T} \leftarrow$ EvaluateSplit($G_s, P, \mathbf{C}$)
11:            $\mathsf{B} \leftarrow$ GetBestAction($\mathsf{A}, \mathsf{R}, \mathsf{U}, \mathsf{S}, \mathsf{M}, \mathsf{T}$)        ▷ Returns action with max. $\mathcal{IG}$
12:            if $\mathsf{B} \neq \emptyset$ then
13:                Update $\mathbf{C}$ and $P$ using $\mathsf{B}$ and Communicate $\mathsf{B}$ to the analyst
14:        until $\mathsf{B} \neq \emptyset$        ▷ Move to next snapshot if nothing is to be learned
```

Each performed action consists of an update of the background knowledge (updating P and $\mathbf{C}$) followed by communication of the performed action B, to the analyst (Line 12). An example can be seen in Figure 1, where in each state the initial and final (represented by superscript I and F respectively) set of constraints is represented by $\mathbf{C}$ (indexed by subscript $s \in [1, T]$).

The feasibility constraint comes into effect while searching for the best action to be performed in each step (Line 11). The overall best action with the maximal value of $\mathcal{IG}$ is selected and returned. If the best action violates the feasibility constraint, then null is returned and the process continues with the next graph snapshot.

The EvaluateAdd procedure is used to discover the best new subgraph pattern with maximum $\mathcal{IG}$, which is a complex problem. This can be realized by the fact that the discovery of new pattern requires the evaluation of all possible $2^{|V|}$ candidate subgraphs. Hence, we use a hill climber based search algorithm (SearchPattern, see Algorithm 2) based on the SSG algorithm (van Leeuwen et al., 2016), which is proposed for finding a subjective interesting subgraph in a

Algorithm 2 Find the most interesting pattern for addition

```
1: procedure SearchPattern($G_s$, $P$, $H^*$, $I^*$)
2:     $H \leftarrow H^*$, $I \leftarrow i^*$
3:     for $u \in Neighbors(H, G_t) \setminus W$ do        ▷ try if adding a vertex increases $I$
4:         $W' \leftarrow W \cup \{u\}$, $I' \leftarrow \mathcal{IG}(\texttt{add})$
5:         if $I' > I$ then
6:             $W \leftarrow W'$, $I \leftarrow I'$, $H \leftarrow (W', k_{W'})$
7:     if $I > I^*$ then
8:         return SearchPattern($G_s$, $P$, $H$, $I$)
9:     else
10:        for $u \in W$ do        ▷ try if removing a vertex increase $I$
11:            $W' \leftarrow W \setminus \{u\}$, $I' \leftarrow \mathcal{IG}(\texttt{add})$
12:            if $I' > I$ then
13:                $W \leftarrow W'$, $I \leftarrow I'$, $H \leftarrow (W', k_{W'})$
14:        if $I > I^*$ then
15:            return SearchPattern($G_s$, $P$, $H$, $I$)
16:        else
17:            return $(H^*, I^*)$        ▷ If nothing increases $I^*$ return the found graph pattern
```

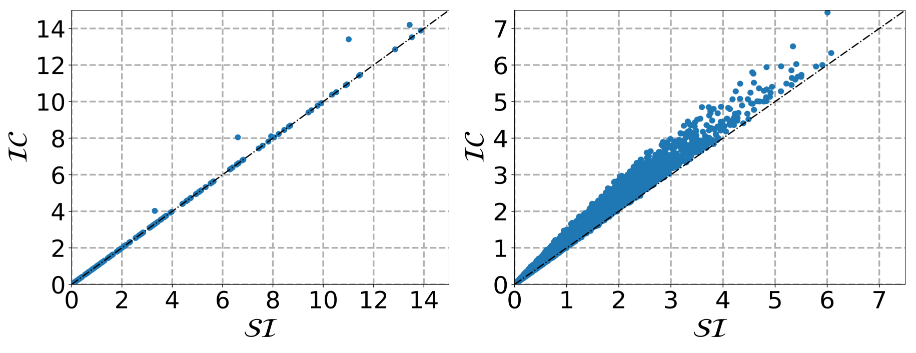


Fig. 2 Plots of $\mathcal{IC}$ vs $\mathcal{SI}$ of all connected subgraphs of a Barabási-Albert random graph of 20 vertices

static graph. This algorithm starts with a seed pattern H^* and recursively adds (Line 3-6) or removes (Line 10-13) vertices to find a pattern with a maximal value of $\mathcal{IG}$. This search stops if neither a vertex can be added nor removed (Line 17). To ensure the connectedness constraint, while adding vertices only vertices neighboring to vertices present in the pattern are checked (Line 3). As this hill climber is likely to suffer from convergence to local optima, we independently run the Algorithm 2 for a list of seed patterns (van Leeuwen et al., 2016) and select the single best pattern as search result. Further, note that the computational cost of naïvely computing $\mathcal{IG}(\texttt{add})$ at each step of the hill climber would be prohibitive, as it would require to compute a new Lagrangian multiplier to update the background distribution at each step. As this is the same problem as van Leeuwen et al. (2016) faced, we also adapt the same solution. That is, information content $\mathcal{IC}$ of a pattern $\theta = (W, k_W)$, as defined in Equation 8, is approximated by

$$\mathcal{IC}(\texttt{add}) \approx \mathcal{SI}(\theta) = n_W \cdot \mathbf{KL}\left(\frac{k_W}{n_W}||p_W\right). \tag{16}$$

We empirically show that Equation 16 is an adequate approximation of Equation 8 in Figure 2. To obtain Figure 2, we created a random graph of 20 vertices using the Barabási-Albert model and computed the values of $\mathcal{SI}$ (Equation 16) and $\mathcal{IC}$ (Equation 8) of all possible connected subgraphs, considering the two types of prior belief as discussed in Section 4.4. It is observed that for all candidate subgraphs (and for both types of prior belief) the value of $\mathcal{SI}$ is always less than or equal to $\mathcal{IC}$. Although they are not exactly equal, the correlation $r = 0.9999$ (in Figure 2a) and $r = 0.9948$ (in Figure 2b) are high enough to suggest that $\mathcal{SI}$ can be successfully used as proxy for $\mathcal{IC}$, as is also argued by van Leeuwen et al. (2016). Moreover, computing $\mathcal{SI}$ is clearly much faster than computing $\mathcal{IC}$, as it does not require updating the background distribution at each step. Hence, this allows to discover surprisingly densely connected graph patterns from snapshots of the graph in an efficient way.

EvaluateRemove and EvaluateUpdate are used to evaluate each constraint in **C** to, either `remove` or `update` a constraint, respectively. In these procedures, each constraint in **C** is independently evaluated by computing the corresponding

Algorithm 3 Find a candidate shrink pattern

```
1: procedure ShrinkPattern($H^*$, $P$, $I^*$)
2:     for $u \in W$ do
3:         $W' \leftarrow W \setminus \{u\}$, $I' \leftarrow \mathcal{IG}(\alpha)$
4:         if $I' > I$ then
5:             $W \leftarrow W'$, $I \leftarrow I'$, $H \leftarrow (W', k_{W'})$
6:     if $I > I^*$ then
7:         return ShrinkPattern($H$, $P$, $I$)
8:     else
9:         return $(H^*, I^*)$
```

$\mathcal{IG}$. To compute $\mathcal{IC}$ (as in Table 3), we update the background distribution assuming that the action would take place. Of note, the update in the background distribution is rolled back after evaluation of each constraint. Both of these method return the respective constraint with maximal $\mathcal{IG}$.

Similarly, EvaluateMerge returns two constraints (in **C**) or patterns which, when `merged`, result in a connected graph pattern with maximal $\mathcal{IG}$.

EvaluateShrink is used to evaluate each constraint in **C** for `shrink` and the reduced constraint with maximal $\mathcal{IG}$ is returned. To shrink a pattern or constraint, we use the procedure ShrinkPattern (Algorithm 3), which recursively removes vertices (Line 2-7) until no increase in $\mathcal{IG}$ is observed (Line 9).

EvaluateSplit is used find the constraint which produces maximal $\mathcal{IG}$ upon `split`. Note that, a new pattern that is the result of `split` may `shrink` in a next iteration; hence we also evaluate a possible reduction of each resulting pattern upon `split` using procedure ShrinkPattern. Thus, EvaluateSplit contains two parts: 1) first the different connected components in the original pattern are identified (each component acts as a new pattern or constraint), and 2) then each new pattern is evaluated for `shrink`.

Complexity. In a single iteration of DSSG, six different procedures are executed sequentially; hence, we discuss the complexity of each procedure. The EvaluateAdd procedure runs the hill climber SearchPattern independently, k times for k different seeds. In each iteration of this hill climber, the computation of $\mathcal{IG}$ is the most computationally expensive part, with time complexity of $\mathcal{O}(|W|^2)$ (from Theorem 1), where W is the set of vertices in a pattern. This hill climber is a direct adaptation of SSG and van Leeuwen et al. (2016) showed that this complexity can be reduced to $\mathcal{O}(|W|)$. Hence, if the number of neighbors in Algorithm 2 is (let's say) $\mathcal{N}$, then each iteration takes $\mathcal{O}(\mathcal{N}|W|)$. Thus, the worst-case complexity of running SearchPattern becomes $\mathcal{O}(\mathcal{I}\mathcal{N}|W|)$, assuming that the hill climber runs for at most $\mathcal{I}$ iterations.

In the other procedures, to evaluate each constraint in **C** requires the computation of $\mathcal{IG}$, which takes $\mathcal{O}(|W|^2)$ (from Theorem 1). Note that computing the Lagrangian multiplier corresponding to a revised constraint in **C** requires to run the bisection method, which has a complexity of $\mathcal{O}(n|W|^2)$. In this, n is the number of iterations required, computed as $\log \frac{\epsilon_0}{\epsilon}$, where ϵ is the given error or tolerance and ϵ_0 is the initial bracket size. Thus, the other procedures have a complexity of $\mathcal{O}(n|W|^2)$.

Given that the complexity of the overall algorithm strongly depends on the actual number of iterations, which cannot be computed in advance, we will instead mention empirical runtimes in the experiment section.

Table 4 Datasets along with some of their properties. Type indicates if the dataset is a Directed (D) or Undirected (U) graph, $|V|$ is the total number of nodes in the graph, $|E_S|$ is the total number of unique edges without timestamp, $|E_T|$ is the total total number of unique edges with timestamp, T is the total time period for which the edges in the graph are considered, t is the time period covered by each individual state, and $|S|$ is the total number of states considered for each dataset.

Dataset	**Type**	$\|V\|$	$\|E_S\|$	$\|E_T\|$	T	t	$\|S\|$
High-School	U	327	5 818	20 448	5 days	1 hour	41
Workplace	U	217	4 274	11 730	10 days	1 hour	91
MathOverFlow	U	24 818	187 978	231 465	6.5 years	1 quarter	26
Reuters	U	7 403	105 343	159 977	66 days	1 day	66
TheMovieDB	U	8 292	236 691	249 324	10 years	1 year	10
DBLP	U	27 400	83 509	98 330	10 years	1 year	10
WebClicks	D	80 306	90 435	231 055	22 days	1 day	22

6 Experiments and Results

In this section, we will demonstrate the efficacy of the proposed framework and corresponding algorithm, DSSG, by means of quantitative (Section 6.3) and qualitative (Section 6.5) results on seven publicly available real-world datasets (Section 6.1). We also compare the proposed method to baselines based on two recent methods for dynamic graph summarization (Section 6.4).

6.1 Datasets

In this section, we will use the following seven publicly available datasets, also summarized in Table 4.

High-School Interaction[3]: This dataset has a total timespan of 5 days. In all 9 hours of interaction is available per day, except for the first day with 5 hours, and the total timespan is segmented into 41 different states of 1 hour each.

Workplace Interaction[3]: This is an interaction network of employees at a workplace. It has a total timespan of 10 working days, where interactions for 9 hours are available for each day, except for the first day where 10 hours of interactions are available. It is segmented into 91 different states of 1 hour each. Although the interactions are instantaneous in nature, an edge exists for each interaction which occurred in a state (snapshot).

MathOverFlow[4]: This network captures the communication between users on the MathOverFlow website. A timestamped undirected edge exists between two users if one user answers another user's question, comments on another user's question, or comments on another user's answer to any question. The dataset has a total duration of 2 560 days. Here we consider a total timeperiod of 6.5 years, segmented into 26 states of 1 quarter (3 months) each. The lifespan of any edge is considered to be three months, i.e., an edge disappears 3 months after the time it appeared in the network.

[3] source: http://www.sociopatterns.org/

[4] source: https://snap.stanford.edu/data/sx-mathoverflow.html

Reuters Terror Network[5]: This dataset contains words that are present in each news article following the 9/11 terror attack. We build a network of words (as vertices) with a link between them (undirected edge) wherever they appear in the same article. The total time period considered is 66 days, with segments (snapshots) of 1 day each. In each state, the snapshot of the network contains all the words (and edges between them) if they appeared in any news article published on that day.

TheMovieDB: A network of actors (vertices) is considered, with an edge corresponding to a co-acted movie. The data is fetched using the TheMovieDB API[6]. The time period of the network is from year 2009 to 2016, and is segmented into 8 states of 1 year each. All movies in the 8 year time period having actors with popularity score more than 2 are included. Each snapshot contains edges corresponding to movies released in the same year.

DBLP: This is a co-author network, created using the DBLP[7] data of all publications in top-20 Machine Learning and Data Mining conferences[8] over a period of 10 years. The dataset is segmented into 10 states of 1 year each, adding an edge between two authors if they have co-authored at least one publication in the given year.

WebClicks: A network of click requests (directed edges) is created from referrer host to target host (nodes) for the time period between 1 November 2009 to 22 November 2009. To prune the data[9], we only consider edges with more than 25 requests in a day. Also, the network is segmented into 22 states of 1 day each. That is, the edge remains only for 1 day, given that at least 25 requests were made from referrer host to target host.

6.2 Experimental Setup

The prior belief for each of the datasets, except for the TheMovieDB dataset, used in this paper is type belief-c. For TheMovieDB type belief-i is used.

Since, we use an adaptation of the hill climber given by van Leeuwen et al. (2016), we fix the following parameters as suggested by the same article.

1. The parameter 'q' used in computation of the description length of pattern (see Table 3) is fixed at 0.01.
2. We use the 'interestingness' based 'TopK' seeding strategy with $k = 10$.

The experiments are executed on an Apple Macbook Pro 2018, with 2.3 GHz Quad-Core Intel Core i5 processor and 8GBs of RAM.

6.3 Quantitative Analysis

In this subsection, we demonstrate the performance of the proposed method on the above mentioned datasets. We evaluate the results in terms of 1) the type

[5] source: http://vlado.fmf.uni-lj.si/pub/networks/data/CRA/terror.htm
[6] source: https://www.themoviedb.org/documentation/api
[7] source: https://dblp.uni-trier.de/
[8] source: https://scholar.google.co.in/citations?view_op=top_venues&hl=en&vq=eng
[9] source: http://carl.cs.indiana.edu/data/websci2014/web-clicks-nov-2009.tgz

Table 5 Properties of the found set of patterns (or constraints) $\mathbf{C}_s$ in each state s. Minimum, median, and maximum value of each property is shown among all states in each dataset, where the number of constraints in each state is shown by $|\mathbf{C}_s|$; total number of performed actions in each state by $|\mathcal{A}|$; difference in two sets of constraints ($\mathbf{C}_{s-1}$ and $\mathbf{C}_s$) in terms of the number of edges added and removed covered by patterns in either set by $\Omega_{\mathbf{C}}$; overall changes in the dataset between two consecutive states ($s-1$ and s) in terms of number of edges added and removed by Ω_s; average of the average density of all the patterns in $\mathbf{C}_s$ by $\bar{\rho}$; compression ratio by CR; and coverage, i.e., the fraction of vertices of the dataset covered by all patterns combined. Runtime (in seconds) is the time required to process all states of the dataset, i.e., to obtain a complete solution of Problem 2.

DataSet		$\|\mathbf{C}_s\|$	$\|\mathcal{A}\|$	$\Omega_{\mathbf{C}}$	Ω_s	$\bar{\rho}$	CR	Coverage	Runtime
	min	6	6	60	326	0.4104	0.47%	10.70%	
High-School	*median*	14	12	250	819	0.6247	8.01%	28.13%	307
	max	22	16	755	1247	0.7600	25.01%	58.41%	
	min	1	1	5	35	0.5176	0.36%	2.16%	
Workplace	*median*	5	5	50	193	0.7000	1.23%	3.23%	96
	max	11	11	626	997	1.0000	36.44%	28.57%	
	min	1	1	4 151	8 223	0.0070	0.63%	2.59%	
MathOverFlow	*median*	10	7	6 147	16 703	0.0179	4.38%	10.22%	74 849
	max	23	17	14 138	24 292	0.1226	46.20%	12.56%	
	min	5	1	177	322	0.0353	0.16%	2.20%	
Reuters	*median*	18	11	796	3 827	0.2630	5.44%	4.71%	79 049
	max	32	27	6 550	13 494	0.6693	19.50%	16.52%	
	min	2	9	3 725	22 145	0.0062	12.79%	6.07%	
TheMovieDB	*median*	27	45	6 379	42 586	0.3125	21.82%	10.93%	18 908
	max	118	118	15 556	74 499	0.9968	48.30%	17.62%	
	min	2	1	345	7 959	0.6208	1.03%	2.19%	
WebClicks	*median*	6	2	539	11 759	0.6814	2.41%	3.58%	14 576
	max	7	3	3 384	12 751	0.7104	18.50%	4.24%	
	min	10	10	462	4 125	0.6207	7.49%	0.49%	
DBLP	*median*	62	68	3 599	16 705	0.6814	9.28%	2.53%	72 548
	max	142	160	6 604	29 704	0.7104	11.51%	5.13%	

of *actions* performed in each state, 2) the number of patterns (or constraints) required to summarize each state, 3) the densities of the patterns found in each state, 4) the ratio of the vertices covered by the patterns in the dataset in each state, and 5) the compression ratio between the encoding cost of the data given the initial background distribution and given the final background distribution in each state. We also showcase the feasibility of the proposed approach by presenting the time taken for online summarization of all states in each graph dataset. Table 5 presents the results and summarizes the set of found patterns for each dataset by the proposed method.

Number of patterns required to summarize each state. We observe the lowest median number of patterns, i.e., 5 for Workplace and most, i.e., 62, for DBLP. This is expected as Workplace has the smallest number of vertices and DBLP has the second most number of vertices among all considered datasets. However, WebClicks has the largest number of vertices but surprisingly very few patterns are found to summarize each state ranging between $2-7$. This is because WebClicks is sparsely dense with the number of unique edges $|E_s|$ almost equal to number of vertices $|V|$ (see Table 4). For TheMovieDB, a high number of patterns are observed in the summary of each state as TheMovieDB is relatively dense dataset.

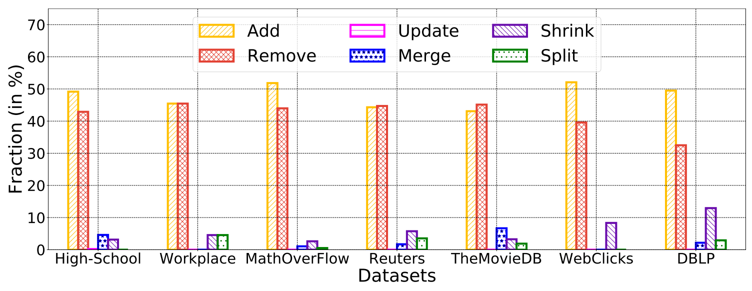


Fig. 3 The fraction of each type of action used to summarize each dataset.

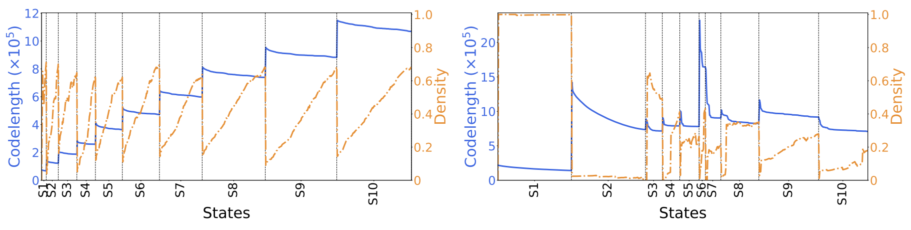
We also observe that the patterns found covers a high number of vertices despite of performing only limited actions. The largest coverage of 58.41% is observed for HighSchool and smallest of 0.49% in DBLP. In case of WebClicks, reasonable coverage in the range of 2.19% − 4.24% is observed. Hence, we conclude that depending upon the size and density of a dataset, our method adequately identifies the number of patterns required summarize each state of a dynamic graph.

Number and type of actions performed. We observe that the number of actions ($|\mathcal{A}|$) performed in each state is consistent with number of changes taking place in the network upon evolution from one state to another (i.e., total number of new edges added and old edges removed, shown by Ω_S). That is, when Ω_S is smaller, a smaller value of $|\mathcal{A}|$ is observed, and vice versa. For example, in Reuters only 1 action is performed when changes in the network are small, i.e., a total of 322 edges are either added or removed, and 27 actions are performed when the changes are much larger, i.e., 13 494 edges either appeared or disappeared from the network. The fraction of each type of action performed can be seen in Figure 3. It is found that `add` and `remove` are the two most frequently performed actions, whereas the other types of actions depend on the nature of evolution of the network. It is seen that `update` is performed only for the High-School network. For WebClicks, no `merge` or `split` actions are observed. Hence, the type of actions carried out are dependent on the topology of the network and the nature of evolution, to which our proposed algorithm effectively adapts itself.

Quality of patterns. We assess the identified patterns through average density[10] ρ and compression ratios CR[11]. Minimizing the encoding cost of the data is only one part of our objective, and we use it to signify the information contained in the patterns: the higher the compression ratio, the more information about the data is provided by the patterns. The maximum compression ratio is observed for TheMovieDB, which is 48.30%, and the minimum of 0.16% is obtained for Reuters. This is accompanied by the observed high values for the average of the average densities of all identified patterns, including the minima of 0.0062 and 0.0070 in case of TheMovieDB and MathOverFlow respectively, which are also higher than the average densities of snapshots of the data. Thus, our method finds subjectively dense and informative patterns.

[10] For a graph $G = (V, E)$, $\rho = \frac{|E|}{|V|*(|V|-1)}$ (directed) or $= \frac{2*|E|}{|V|*(|V|-1)}$ (undirected)

[11] CR is 1 minus the ratio of the encoding cost (number of bits, computed as $-\log_2 P(\mathbf{D})$) of the data given the initial background distribution and given the final background distribution.



(a) DBLP (b) TheMovieDB

Fig. 4 Codelength (blue solid line) vs average of the average densities of patterns in set $\mathbf{C}_s$ (orange dashed dotted line) vs state s. The vertical dashed lines indicates the change of state and the horizontal axis represent from left to right all iterations, where a series of actions is performed for each consecutive state.

We also observe for TheMovieDB where a more sophisticated belief, i.e., belief-i is used. That is, the background distribution closely represents a snapshot of the dataset, and with the change of state, any action would results in high compression ratios, which is also observed in Table 5.

Quality of actions performed. We next investigate the sequential approach taken in Problem 3. From the nature of the problem, it is expected that with each performed action, the codelength of the data should decrease and the average of average densities of identified set of patterns should increase. This is confirmed by Figure 4, where the codelength is found to be always decreasing and the density is mostly increasing for the DBLP and TheMovieDB networks. We also observe in Table 5, that there is a correlation between changes captured by the actions (i.e, Ω_C) and changes in the overall state (i.e., Ω_S) compared to the previous state. For TheMovieDB, we observed a relatively larger value of Ω_C, i.e., 15 556, when Ω_S is also large, i.e., 74 499; for Workplace we observed smaller value of Ω_C, i.e., 5, when Ω_S is smaller, i.e., 35. Therefore, the actions capture the changes in the graph state appropriately.

Runtime Analysis. Last, we discuss the (computation) time taken to run the experiment for each dataset. This is comprised of various factors, including the time required to compute the background distribution, executing the hill climber with different number of seeds to discover patterns, creating a candidate list for each type of atomic change to be performed, and updating the background distribution. In Table 5, the factors visibly affecting runtime are the size and density of a dataset, and the number of segments considered in a dataset. Overall, the maximum runtime of 79 049 seconds, which is approximately 22 hours for Reuters, appears practical. However, this could be further reduced upon optimization and parallelization of the proposed algorithm. Also note that all experiments have been run on a standard laptop.

6.4 Comparison with baseline methods

In Section 2 (and specifically in Table 1) we have described in detail how DSSG differs from existing methods for dynamic graph summarization, i.e., it solves a (slightly yet crucially) different problem. Specifically, unlike other methods DSSG summarizes a dynamic graph by discovering state-to-state relative changes in the form of evolving patterns and incrementally updates the analyst's knowledge after each graph snapshot. To empirically demonstrate that DSSG provides good solutions to this problem, we here compare its results to those obtained by two baseline methods adapted from TimeCrunch (TC) (Shah et al., 2015) and Scalable Dynamic Graph summarization Method (SDGM) (Tsalouchidou et al., 2020).

Baseline Methods. Next we describe how we adapt TC and SDGM to match our problem setting. For TC, at each state s we compute two summaries using TC, for 1) graph sequence $G_1, \dots, G_{s-1}$, and 2) graph sequence $G_1, \dots, G_{s-1}, G_s$. The difference between the two resulting summaries is the incremental information communicated to the analyst using two action types, namely `add`, to communicate patterns that appeared in state s, and `remove`, to communicate patterns present in state $s-1$ but not in s. These actions are encoded as with DSSG (see Table 3), except that the number of action types is two. SDGM provides a summary after each state, hence for SDGM we use the same two action types and encoding to communicate the changes between each two consecutive summaries.

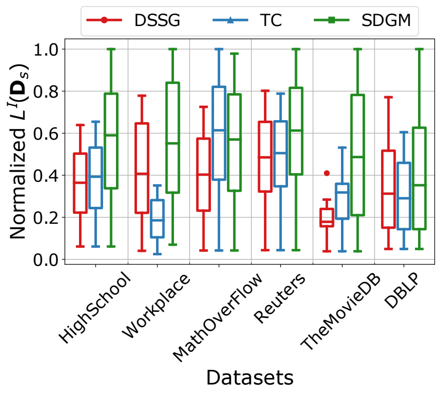
As neither TC nor SDGM considers prior knowledge, for consistency in comparison we start from the same initial background distribution as for DSSG. The background distribution is updated after each action, exactly as in our approach (Equations 5-7, 14). Whereas our approach automatically selects the number of patterns needed to summarize the changes, TC and SDGM do not. For TC we set the (maximum) number of actions[12] to be the number of patterns found by DSSG in each state. For SDGM[13], we choose the maximum number of patterns among all states by DSSG as the number of supernodes in each dataset, as it identifies a preset number of supernodes while producing a summary in each state. Note that providing this information from DSSG is potentially favorable to TC and SDGM.

Evaluation criterion. The objective of our main problem, i.e., Problem 2, is to minimize $-\log P^*_{\mathbf{C}_s}(\mathbf{D}_s) + L(\mathbf{C}_s|\mathbf{C}_{s-1})$ for each state s of a dynamic graph. Hence, we assess the there methods using this function as a measure. For simplicity, we denote the number of bits required to encode a graph snapshot $\mathbf{D}_s$ given background distribution $P^*_{\mathbf{C}_s}$, i.e., $-\log P^*_{\mathbf{C}_s}(\mathbf{D}_s)$, by $L(\mathbf{D}_s)$, and the number of bits required to encode all atomic changes, i.e., $L(\mathbf{C}_s|\mathbf{C}_{s-1})$, by $\mathcal{DL}$. In the following, we use superscripts I and F to represent respectively the Initial and Final values of a state. Apart from absolute values, we also investigate the difference between these initial and final values, which is equivalent to the total $\mathcal{IG}$ achieved by performing all actions found for a state s, as given by

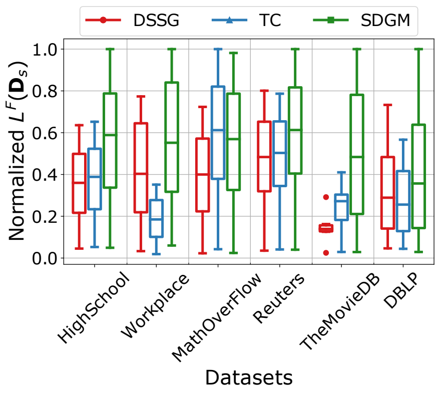
$$\begin{aligned}\mathcal{IG} &= \left(\overbrace{-\log P_{C_s^I}(\mathbf{D}_s)}^{\mathbf{L}^I(\mathbf{D}_s)} + \overbrace{L(\mathbf{C}_s^I|\mathbf{C}_{s-1}^F)}^{=0,\text{ as } \mathbf{C}_s^I = \mathbf{C}_{s-1}^F} \right) - \left(\overbrace{-\log P_{C_s^F}(\mathbf{D}_s)}^{L^F(\mathbf{D}_s)} + L(\mathbf{C}_s^F|\mathbf{C}_{s-1}^F) \right) \\ &= \underbrace{\log P_{C_s^F}(\mathbf{D}_s) - \log P_{C_s^I}(\mathbf{D}_s)}_{\mathcal{IC}} - \underbrace{L(\mathbf{C}_s^F|\mathbf{C}_{s-1}^F)}_{\mathcal{DL}}.\end{aligned}$$

[12] As recommended in Shah et al. (2015), we fix the Jaccard similarity threshold to 0.5.

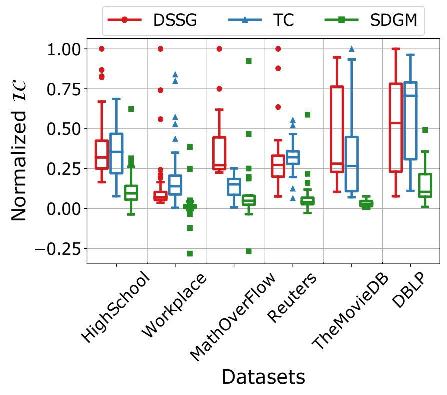
[13] Tsalouchidou et al. (2020) suggests choosing a high number of microclusters compared to the number of supernodes, therefore, the number of microclusters is chosen as 10 times the required number of supernodes/clusters. The required window size is set to 2, to take into consideration only the previous state while summarizing each state.



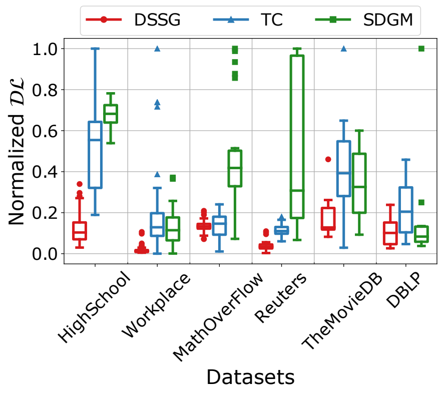
(a) Normalized $L^I(\mathbf{D}_s)$ (lower=better)



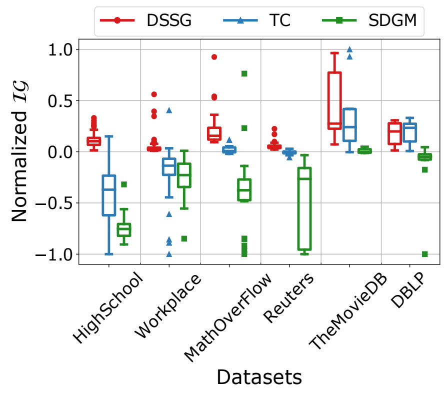
(b) Normalized $L^F(\mathbf{D}_s)$ (lower=better)



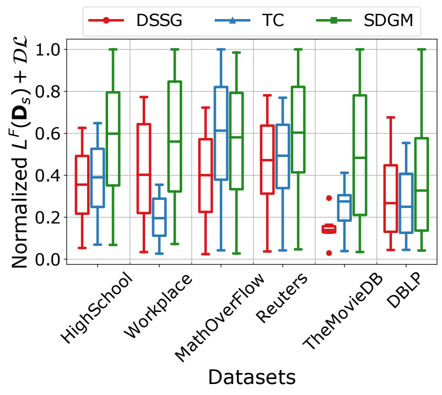
(c) Normalized $\mathcal{IC}$ (higher=better)



(d) Normalized $\mathcal{DL}$ (lower=better)



(e) Normalized $\mathcal{IG}$ (higher=better)



(f) Normalized $L^F(\mathbf{D}_s) + \mathcal{DL}$ (lower=better)

Fig. 5 Each subfigure shows the distribution of values over all states, for a component of the evaluation criterion, for six datasets and all three methods. Each measure is normalized by setting the maximum absolute value observed for a dataset by any of the methods to one.

Results. Starting with Figure 5f, depicting overall compression by means of normalized $L^F(\mathbf{D}_s)+\mathcal{DL}$ over all states, we observe that DSSG yields lower (=better) values for four datasets; TC has better results for WORKPLACE and DBLP. This shows that DSSG generally succeeds in finding better solutions to the overall problem, with TC often close and occasionally better. If we now turn our attention to the results for $\mathcal{IG}$ in Figure 5e, we observe that DSSG is the *only method that always finds actions having positive information gain.* For WORKPLACE, where TC seemed to perform better on a high level, TC finds patterns that provide negative information gain—which conflicts our problem statement, and therefore the results are suboptimal from the perspective of our problem.

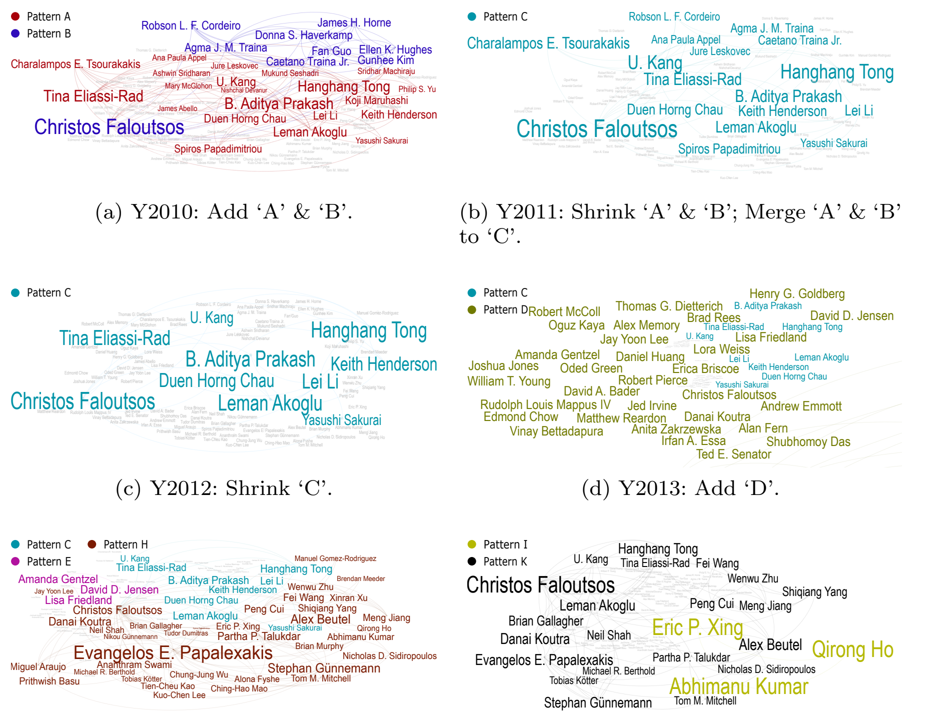
For DBLP, however, TC performs better than DSSG with regard to both $L^F(\mathbf{D}_s) + \mathcal{DL}$ and $\mathcal{IG}$. We therefore investigate the individual components of $\mathcal{IG}$, i.e., $L^I(\mathbf{D}_s)$, $L^F(\mathbf{D}_s)$, $\mathcal{IC}$, and $\mathcal{DL}$, in Figures 5a-5d respectively. In Figures 5a and 5b, we see that the encoded sizes of the data at the start and end of each state are (logically) very similar to each other, but they are also quite similar to those in Figure 5f: the size of the data is a relatively large part of the total compressed size. The (normalized) encoded sizes of the data obtained by DSSG are typically smaller than those obtained by the other methods.

When we study the distributions of information content in Figure 5c, we observe that both DSSG and TC succeed in identifying patterns with high information content. The high $\mathcal{IC}$ values for TC come at the cost of higher values for $\mathcal{DL}$ though: Figure 5d shows that the description lengths required to communicate the patterns are larger for TC than for DSSG—also for DBLP. Given that the same encoding is used for TC as for DSSG, and the number of patterns is fixed for TC, this means that the patterns found by TC are larger and often—but not always—less informative. (Also, note that DSSG is able to employ other action types, enabling compact yet informative summaries.) SDGM clearly solves a (very) different problem than DSSG, as it finds patterns with both low information content and large description lengths.

In summary, we conclude that when we adapt TC and SDGM for the problem that we consider in this paper, they perform less well than DSSG. TC finds summaries that are similarly informative yet more complex than those of DSSG. SDGM, on the other hand, generally finds complex summaries that are far less informative than those identified by the other methods. We would like to stress that this should not come as a surprise though, and should certainly not disqualify either TC or SDGM: they have been designed to solve other problems than DSSG. The above results demonstrate that our problem and approach are indeed different from those considered by TC and SDGM, corroborating our proposed approach.

6.5 Qualitative Analysis

In this subsection, we discuss how the summary created by our proposed approach can be meaningful to a domain expert. Since we provide a summary of the changes in a dataset, the effectiveness of the discovered patterns can be assessed by the



(a) Y2010: Add 'A' & 'B'.

(b) Y2011: Shrink 'A' & 'B'; Merge 'A' & 'B' to 'C'.

(c) Y2012: Shrink 'C'.

(d) Y2013: Add 'D'.

(e) Y2014: Split 'D' to 'E' & 'F'; Add 'G'; Merge 'F' & 'G' into 'H'.

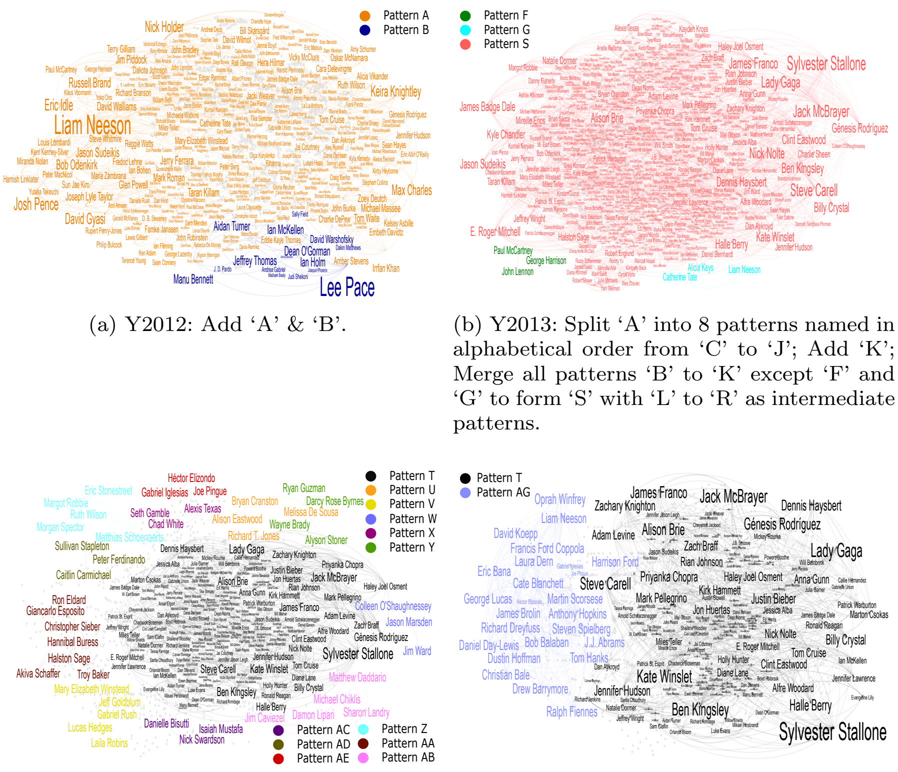
(f) Y2015: Split 'H' to 'I' & 'J'; Remove 'E'; Shrink 'C'; Merge 'J' & 'C' into 'K'.

Fig. 6 Shown is the evolution of patterns in the DBLP network from Year 2010 to 2015.

information captured in the sets of patterns and the actions performed on them. We analyze the results[14] obtained for DBLP and TheMovieDB.

DBLP. For the DBLP graph, we discuss one of the various captured chains of subgraph patterns, which demonstrates the evolution of the communities of 92 authors centered mainly around Christos Faloutsos from Year 2010 to 2015. This evolution is shown in Figure 6. Initially (Figure 6a) in Year 2010, two surprisingly dense communities shown as pattern 'A' and 'B' are discovered, where Christos Faloutsos is a common link between the two communities. These two different communities have been condensed in the following year and merged to form a single community, shown as pattern 'C' (Figure 6b) with Christos Faloutsos and U Kang being some of the prominent names. This collaboration network shrinks the next year (Figure 6c). In Year 2013 another very densely connected set of authors is discovered, shown as pattern 'D' in Figure 6d. Surprisingly, in the subsequent year, this set of authors got split into two different communities of three authors each, i.e., Lisa Friedland, David D. Jensen & Amanda Gentzel and Christos Faloutsos, Jay Yoon Lee & Danai Koutra. However, the latter set of authors got merged with a newly discovered densely connected set of authors centered

[14] In the following figures, graphs are shown such that the text size of a vertex label is proportional to its degree. That is, if vertex degree is higher then text size is larger and vice versa.



(a) Y2012: Add ‘A’ & ‘B’.

(b) Y2013: Split ‘A’ into 8 patterns named in alphabetical order from ‘C’ to ‘J’; Add ‘K’; Merge all patterns ‘B’ to ‘K’ except ‘F’ and ‘G’ to form ‘S’ with ‘L’ to ‘R’ as intermediate patterns.

(c) Y2014: Split ‘S’ into 12 patterns shown in order from ‘T’ to ‘AE’; Remove ‘F’ & ‘G’.

(d) Y2017: Add ‘AF’; Merge ‘AE’ & ‘AF’ to ‘AG’.

Fig. 7 Shown is the evolution of patterns in TheMovieDB network from Year 2012 to 2017.

around Christos Faloutsos and Evangelos E. Papalexakis, shown by pattern ‘H’ in Figure 6e. Finally, in year 2015 the two different communities where Christos Faloutsos is the common link, i.e., pattern ‘C’ and a part of pattern ‘H’, merge to form one community with Neil Shah starting the collaboration with Leman Akoglu and others. In short, we captured how the community around one author with a large number of collaborations evolve over time.

TheMovieDB. In this network, we discussed the discovered evolution of different patterns or communities of 1019 actors from Year 2012 to 2017, as shown in Figure 7. For each found pattern, we also find the associated genres using the hypergeometric test. A genre is considered to be significant if the p-value after Bonferroni Correction (with factor 19) is less than $1e-1$. During the Year 2012, two patterns ‘A’ and ‘B’ are discovered (Figure 7a). Pattern ‘A’, with significant genres Action and Comedy, includes vertices such as Liam Neeson, Josh Pence, David Gyasi and Nick Holder, all with high vertex degree. Pattern ‘B’ comprises of Sally Field and Lee Pace as high degree vertices and has Adventure and Fantasy as significant genres. In Year 2013, Pattern ‘A’ splits into 8 resulting patterns (‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘H’, ‘I’, ‘J’). This suggests that these 8 patterns represents 8 different communities of actors. Surprisingly, among these 8 patterns (which are all non-overlapping disjoint patterns), 6 patterns (excluding ‘F’ and ‘G’) got merged to form Pattern ‘S’ only after pattern ‘K’ is discovered (Figure 7b). Hence, it is found that the actors of the pattern ‘C’, ‘D’, ‘E’, ‘H’, ‘I’ and ‘J’ are indirectly connected through the actors in pattern ‘K’. Some of the notable actors of pattern ‘S’ include James Badge Dale, Kyle Chandler, Kirsten Dunst and Will Smith. Pattern ‘S’ has Romance, Crime and Western as the three significantly associated genres. Pattern ‘S’ is decomposed into 12 different patterns in Year 2014 (Figure 7c). All the 12 resulting patterns have different significantly associated genres such as, Action with pattern ‘T’, Science Fiction with ‘U’, Documentary with ‘W’, Fantasy with ‘X’, Animation & Family with ‘Y’ and ‘AE’, War with ‘Z’ & ‘AD’, Crime with ‘AA’, Drama with ‘AB’ and Comedy with ‘AC’. Most of the patterns disappear in the following two years, i.e., 2015 and 2016, except patterns ‘T’ and ‘AE’. In Year 2017, pattern ‘AE’ merges with a newly discovered pattern ‘AF’, resulting in pattern ‘AG’. Thus, pattern ‘T’ and ‘AG’ are observed in Year 2017 (Figure 7d). Some of the prominent actors in pattern ‘T’ are Sylvester Stallone, Lady Gaga, Ben Kingsley and Alison Brie. Pattern ‘AG’ includes actors like Dustin Hoffman and Oprah Winfrey and has Animation, Fantasy and Adventure as significant genres.

This case presents how the collaboration between actors evolves over time. The genres which are significantly associated to each pattern implies that our algorithm successfully identifies different and evolving subgroups (or communities) in the network.

7 Case Study: Airline Flight Network

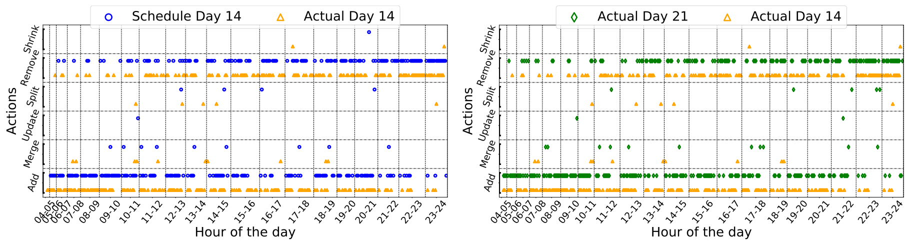
To explore how the proposed approach and algorithm could be used in a real-world scenario, we now present a case study on the US flight network[15]. Flight networks are typical examples of dynamic graphs that one would like to analyze on the fly, e.g., to detect and monitor delays as early as possible.

Dataset. We use the scheduled and actual flight operating data for the month of January 2017, with 298 airports (considered as vertices) and 450 017 flights operated in that month. The dataset has features such as scheduled departure and arrival time, along with actual departure and arrival time, for each flight. Using these features we create two types of networks: 1) in a scheduled flight network, a directed edge for a given time interval is included from origin to destination airport if at least one flight was scheduled to depart or arrive in that interval; 2) in an actual flight network, a directed edge is included between two airports if at least one flight actually departed or arrived in that interval.

For either type of network we create 31 independent instances, one for each day of January 2017. Each network is segmented into 20 sequential snapshots, or states, of one hour each (from 0400 hours to 2400 hrs, all converted to UTC -7). The motivation behind choosing one hour as the length of a snapshot is that airliners manage their operations in blocks of one hour duration each. For simplicity, we do not consider cancelled flights in this case study.

Approach. We use DSSG to independently summarize both the scheduled and actual flight network. In both the cases, we assume the analyst to have a prior belief on the number of routes scheduled to be operated from each airport in the

[15] source: https://www.transtats.bts.gov/



(a) Scheduled vs actual operations of day 14. (b) Actual operations of day 14 vs day 21.

Fig. 8 Actions (and their types) throughout the day as found by DSSG on the scheduled and actual flight networks of two days.

initial snapshot (i.e., the total number of airports from where at least one flight is arriving and the total number of airports to where at least one fight is departing). We then inspect the resulting summaries.

Summaries. As the data is large and dynamic, visualizing all patterns or the complete summary at once is not practical. Instead, Figure 8 visualizes the sequence of actions identified by our method on a given flight network, to provide a high-level overview—or *fingerprint*—of the summary. Such fingerprints can then be compared to spot deviations between the scheduled and actual dynamic networks.

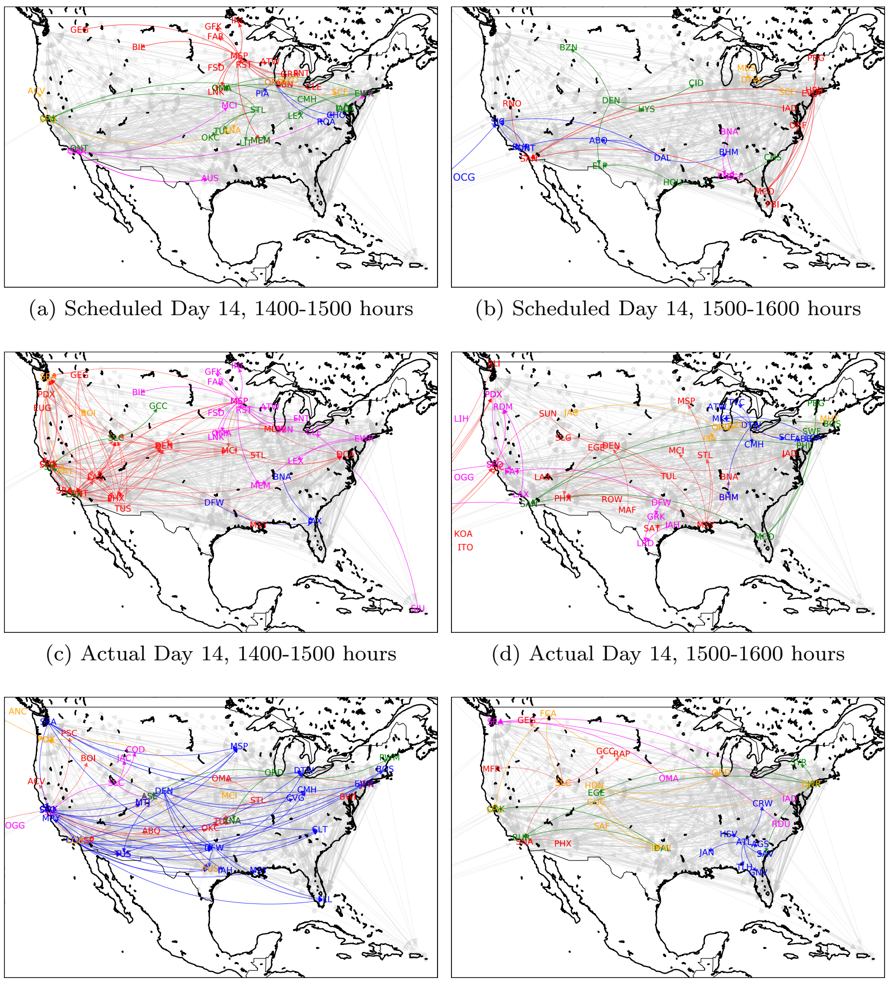
Comparing summaries. An analyst could investigate the discovered patterns (as shown in Section 6.5), but here we first investigate the differences between the obtained summaries, to learn about unexpected events (here: delays) causing the observed network to differ from the expected network. For illustrative purposes, we use the scheduled network of day 14, and actual networks of days 14 and 21.

Inspecting the fingerprints in Figure 8 shows that the actual flight network of day 14 behaves differently from both the scheduled flight network of day 14 (Figure 8a) and the actual flight network of the same day one week later (Figure 8b). For example, in the initial snapshot (0400–0500hrs) in Figure 8a, the prior distribution sufficiently described the scheduled flight network of day 14 and hence no new patterns are discovered. In the actual flight network of that day, however, two patterns are discovered for that snapshot. A closer look at the data reveals that this is caused by flights that operated either ahead of time or delayed. In Figure 8b, similar observations can be made for the actual flight networks for two days exactly on week apart. To further investigate the causes of deviations, an analyst could inspect the patterns and actions. DSSG provides a sequence of actions (descending by $\mathcal{IG}$) that an analyst could learn from, especially when supported by an environment for interactive data and pattern exploration (see Discussion).

Inspecting patterns. To further understand the differences between the flight networks, we consider two typical block hours, i.e., 1400-1500 and 1500-1600 hours. Figure 9 shows the top 5 patterns[16] with regard to information content ($\mathcal{IC}$) and for the same three different networks as above. Note that this means that we only show patterns that are newly discovered or revised in the current state.

From Figure 9a we observe that, for the *scheduled flight network* of day 14 during 1400-1500 hours, four out of the five patterns are star-shaped, with hub airports. In the first pattern (shown in red) MSP is the hub, with flights departing

[16] An analyst could of course visualize all actions or patterns in the summaries, but we only show the top 5 patterns for reasons of space and clarity.


(a) Scheduled Day 14, 1400-1500 hours (b) Scheduled Day 14, 1500-1600 hours

(c) Actual Day 14, 1400-1500 hours (d) Actual Day 14, 1500-1600 hours

(e) Actual Day 21, 1400-1500 hours (f) Actual Day 21, 1500-1600 hours

Fig. 9 The top 5 patterns with regard to information content discovered from each respective flight network. Color coding: the pattern with highest information content is shown in red, followed in order by magenta, green, blue, and orange. Labels indicate airport codes.

for airports such as ATW, LNK and MEM. Similarly, patterns 2 (magenta) and 4 (blue) have SNA and ORD as hubs, respectively. Pattern 3 (green) has STL and EWR as hubs, where flights are departing, and two other airports, OMA and OAK, where flights are arriving. These patterns indicate that a large number of flights are scheduled to depart from hubs like MSP, SNA, ORD and STL, while flights are expected to arrive at OAK and OMA. Finally, pattern 5 is a connected set of airports including SFO, XNA, ORD and SCE.

In the *actual flight network* for the same timeslot in Figure 9c, the most informative pattern is a set of densely connected airports including SLC, DEN, LAS, and SEA (shown in red). The second pattern (magenta) is similar to the most informative pattern found in the scheduled network (red in Figure 9a), with MSP as hub. Patterns 3, 4, and 5 are also star-shaped, with hubs SLC, JAX, and SEA respectively. Upon investigating the underlying data we find that patterns 1 and 3 comprise flights having a combined positive delay (flights departing and/or arriving late) of 1083 minutes and 23 minutes, respectively. This is a relevant discovery, as 1083 minutes is a large combined delay and pattern 1 was not found in the scheduled data. For pattern 2, which we did find in the scheduled network, no positive delay is observed (instead we find a combined negative 'delay' of roughly 9 minutes, which is very moderate). For patterns 4 and 5 negative delays are observed. Similar observations can be made for the block hour in Figures 9c–9d.

The fingerprints of Figure 8b already suggested that the actual flight networks of days 14 and 21 differ, and this is confirmed by the different top 5 patterns shown in Figures 9e-9f. Interestingly, none of these patterns is present in either the scheduled or actual flight network of day 14, and these patterns are also found to correspond to substantial positive and/or negative delays.

Together, these observations indicate that by comparing the summaries and patterns discovered by DSSG, an analyst can learn about sets of connected airports where structural operational deviations from the schedule occurred, which often resulted in delays. As such, this case study served to illustrate how our approach could be used in a real-world scenario where online and incremental analysis of structural changes in dynamic graphs can render valuable insights.

8 Discussion

We propose a framework for summarizing sequential datasets in an online setting. We define *information gain* using both the maximum entropy principle and minimum description length principles. This measure enables not only to quantify the informativeness of a pattern, but also of the proposed actions (or atomic changes) in our framework, which enables to capture the evolution in a graph by evolving patterns. The proposed generic framework for subjective summarization of sequential data can be further instantiated for different types of evolving datasets, such as event sequence databases. In this paper, we instantiated the proposed generic framework for dynamic (simple) graphs.

This work focuses on the discovery of an *online* summary of dynamic graphs, by iteratively identifying actions with maximum information gain. The summary of a dynamic network contains a set of subgraph patterns (or constraints) along with captured changes in those (chains of) patterns over time. The findings from the experiments performed on different networks indicate that 1) the generated summaries are informative with regard to the analyst's prior knowledge about the data, with relatively high observed compression ratios; 2) the sets of subgraph patterns identified to summarize the networks are found to be relatively dense; and 3) the discovered evolving patterns provide an informative sequence that can be further inspected and analyzed. Also, with the proposed measures of information gain and information content, we show in the airline case study that our method can be used to rank the found patterns.

We observe during the experiments that a pattern might appear regularly or sporadically in different snapshots of a dynamic network. This leads to a situation where our method learns and forgets the same pattern multiple times. However, on each occasion, our method treats the same pattern as newly acquired knowledge.

It would be interesting to identify these instances while summarizing a network over time. A way to address this limitation could be to label each subgraph pattern and explore the similarity between two subgraph patterns. Thus, similar to TimeCrunch (Shah et al., 2015), the periodicity of a pattern could be explored. Another limitation of our work is the consideration of prior belief of the analyst. In this setting, we only consider that the analyst has prior knowledge on the initial snapshot and is interested in observing the changes in the network. A different setting may consider that the analyst knows about the different snapshots of the network.

One future opportunity includes improving the scalability of the proposed framework. The runtime of the proposed algorithm is currently higher than the two methods used to compare the summaries provided by DSSG, including TimeCrunch (Shah et al., 2015) and SDGM (Tsalouchidou et al., 2020). Notably, the other two methods have a highly optimized implementation using parallel and distributed computing capabilities. For now, DSSG sequentially executes multiple procedures, including the number of independent seed runs of the hill climber. These procedures are highly independent and could be executed simultaneously. Hence, DSSG has several inherent features which may allow a parallelized implementation. This would significantly reduce the runtime and improve the scalability of the algorithm. Another future opportunity includes the development of a tool based on the proposed framework, for interactive visualization and exploration of changes identified in a dynamic network. This tool would further provide a user-friendly platform for analysts to learn how a network evolves with time.

9 Conclusion

We presented the novel problem of subjective summarization of sequential data in an online manner. As a specific instance of this generic problem, online summarization of dynamic graphs was introduced. We presented a framework to solve this problem, which has been built on the existing ideas related to maximum entropy principle, the minimum description length principle, and subjectively interesting subgraph patterns. We then introduced an efficient algorithm, called DSSG, which is followed by extensive experiments on real-world datasets. Through experimental results, we demonstrated the effectiveness of the proposed algorithm. The generated summaries are found to be informative with regard to the analyst's prior knowledge about the data. We conclude this from the observed substantial compression ratios and the fact that *compression equates learning*. We have also found different sequences of patterns, which evolved over time in a network. We also presented a case study and demonstrated a potential use of the proposed method in the airline domain. Comparison of two different summaries of the airline network, using the scheduled and the actual flight data, revealed potentially informative events. As a part of future work, it would be interesting to extend the proposed method to incorporate a feature to capture periodicity of the patterns; another is to extend this method to multigraphs, weighted graphs, and attributed graphs. Finally, as a part of our ongoing/future work, we aim to develop a tool for interactive visualization and exploration of the found patterns.

Acknowledgements This research has been conducted as part of an Indo-Dutch project titled ‘A Systems approach towards Data Mining and Predictions in Airline Operation’ (SAP-PAO), grant no.: MIT-861-MID. We gratefully acknowledge the project sponsors MeitY (India), NWO (The Netherlands), and GE Aviation (India).

References

Abello J, Resende MG, Sudarsky S (2002) Massive quasi-clique detection. In: Latin American symposium on theoretical informatics, Springer, pp 598–612

Adhikari B, Zhang Y, Bharadwaj A, Prakash BA (2017) Condensing temporal networks using propagation. In: Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, pp 417–425

Ahmed R, Karypis G (2012) Algorithms for mining the evolution of conserved relational states in dynamic networks. Knowledge and Information Systems 33(3):603–630

Ahmed R, Karypis G (2015) Algorithms for mining the coevolving relational motifs in dynamic networks. ACM Transactions on Knowledge Discovery from Data (TKDD) 10(1):1–31

Alpert CJ, Kahng AB, Yao SZ (1999) Spectral partitioning with multiple eigenvectors. Discrete Applied Mathematics 90(1-3):3–26

Araujo M, Papadimitriou S, Günnemann S, Faloutsos C, Basu P, Swami A, Papalexakis EE, Koutra D (2014) Com2: fast automatic discovery of temporal (‘comet’) communities. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, pp 271–283

Bazargan M (2016) Airline operations and scheduling. Routledge

Bendimerad A, Mel A, Lijffijt J, Plantevit M, Robardet C, De Bie T (2020) Siasminer: mining subjectively interesting attributed subgraphs. Data Mining and Knowledge Discovery 34(2):355–393

Cook DJ, Holder LB (1994) Substructure discovery using minimum description length and background knowledge. J Artif Int Res 1(1):231–255

De Bie T (2011) Maximum entropy models and subjective interestingness: an application to tiles in binary databases. Data Mining and Knowledge Discovery 23(3):407–446

Ding CH, He X, Zha H, Gu M, Simon HD (2001) A min-max cut algorithm for graph partitioning and data clustering. In: Proceedings 2001 IEEE International Conference on Data Mining, IEEE, pp 107–114

Flake GW, Tarjan RE, Tsioutsiouliklis K (2004) Graph clustering and minimum cut trees. Internet Mathematics 1(4):385–408

Galimberti E, Barrat A, Bonchi F, Cattuto C, Gullo F (2018) Mining (maximal) span-cores from temporal networks. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, pp 107–116

Goebl S, Tonch A, Böhm C, Plant C (2016) Megs: Partitioning meaningful subgraph structures using minimum description length. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, pp 889–894

Grünwald PD (2007) The minimum description length principle. MIT press

Khan A, Aggarwal C (2016) Query-friendly compression of graph streams. In: 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp 130–137

Koutra D, Kang U, Vreeken J, Faloutsos C (2014) Vog: Summarizing and understanding large graphs. In: Proceedings of the 2014 SIAM international conference on data mining, SIAM, pp 91–99

van Leeuwen M, De Bie T, Spyropoulou E, Mesnage C (2016) Subjective interestingness of subgraph patterns. Machine Learning 105(1):41–75

LeFevre K, Terzi E (2010) Grass: Graph structure summarization. In: Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, pp 454–465

Lin YR, Sun J, Sundaram H, Kelliher A, Castro P, Konuru R (2011) Community discovery via metagraph factorization. ACM Trans Knowl Discov Data 5(3), DOI 10.1145/1993077.1993081, URL `https://doi.org/10.1145/1993077.1993081`

Luce RD (1950) Connectivity and generalized cliques in sociometric group structure. Psychometrika 15(2):169–190

Matsuda H, Ishihara T, Hashimoto A (1999) Classifying molecular sequences using a linkage graph with their pairwise similarities. Theoretical Computer Science 210(2):305–325

Mokken RJ (1979) Cliques, clubs and clans. Quality & Quantity 13(2):161–173

Navlakha S, Rastogi R, Shrivastava N (2008) Graph summarization with bounded error. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM, pp 419–432

Newman ME (2006) Modularity and community structure in networks. Proceedings of the national academy of sciences 103(23):8577–8582

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

Qu Q, Liu S, Zhu F, Jensen CS (2016) Efficient online summarization of large-scale dynamic networks. IEEE Transactions on Knowledge and Data Engineering 28(12):3231–3245

Rissanen J (1983) A universal prior for integers and estimation by minimum description length. The Annals of statistics pp 416–431

Robardet C (2009) Constraint-based pattern mining in dynamic graphs. In: 2009 Ninth IEEE International Conference on Data Mining, pp 950–955

Rozenshtein P, Tatti N, Gionis A (2017) Finding dynamic dense subgraphs. ACM Transactions on Knowledge Discovery from Data (TKDD) 11(3):27

Rozenshtein P, Bonchi F, Gionis A, Sozio M, Tatti N (2018) Finding events in temporal networks: Segmentation meets densest-subgraph discovery. In: 2018 IEEE International Conference on Data Mining (ICDM), IEEE, pp 397–406

Saran D, Vreeken J (2019) Summarizing dynamic graphs using mdl. In: Proceedings of the ECMLPKDD Workshop on Graph Embedding and Mining (GEM), URL `https://publications.cispa.saarland/3002/`

Scharwächter E, Müller E, Donges J, Hassani M, Seidl T (2016) Detecting change processes in dynamic networks by frequent graph evolution rule mining. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, pp 1191–1196

Seidman SB (1983) Network structure and minimum degree. Social networks 5(3):269–287

Seidman SB, Foster BL (1978) A graph-theoretic generalization of the clique concept. Journal of Mathematical sociology 6(1):139–154

Shah N, Koutra D, Zou T, Gallagher B, Faloutsos C (2015) Timecrunch: Interpretable dynamic graph summarization. In: Proceedings of the 21th ACM

SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 1055–1064

Sun J, Faloutsos C, Faloutsos C, Papadimitriou S, Yu PS (2007) Graphscope: parameter-free mining of large time-evolving graphs. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 687–696

Tang N, Chen Q, Mitra P (2016) Graph stream summarization: From big bang to big crunch. In: SIGMOD 2016 - Proceedings of the 2016 International Conference on Management of Data, Association for Computing Machinery, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp 1481–1496, DOI 10.1145/2882903.2915223, 2016 ACM SIGMOD International Conference on Management of Data, SIGMOD 2016 ; Conference date: 26-06-2016 Through 01-07-2016

Toivonen H, Zhou F, Hartikainen A, Hinkka A (2011) Compression of weighted graphs. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 965–973

Tsalouchidou I, Bonchi F, Morales GDF, Baeza-Yates R (2020) Scalable dynamic graph summarization. IEEE Transactions on Knowledge and Data Engineering 32(2):360–373

Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M (2013) Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 104–112

Veremyev A, Prokopyev OA, Butenko S, Pasiliao EL (2016) Exact mip-based approaches for finding maximum quasi-cliques and dense subgraphs. Computational Optimization and Applications 64(1):177–214

Wu Q, Hao JK (2015) A review on algorithms for maximum clique problems. European Journal of Operational Research 242(3):693–709

You Ch, Holder LB, Cook DJ (2009) Learning patterns in the dynamics of biological networks. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA, KDD '09, p 977–986, DOI 10.1145/1557019.1557125, URL `https://doi.org/10.1145/1557019.1557125`