Summarizing and Understanding Large Graphs

Danai Koutra
School of Computer Science
Carnegie Mellon University
danai@cs.cmu.edu

U Kang
Department of Computer Science
KAIST
ukang@cs.kaist.ac.kr

Jilles Vreeken
Max Planck Institute for Informatics
and Saarland University
jilles@mpi-inf.mpg.de

Christos Faloutsos
School of Computer Science
Carnegie Mellon University
christos@cs.cmu.edu

Abstract

How can we succinctly describe a million-node graph with a few simple sentences? Given a large graph, how can we find its most 'important' structures, so that we can summarize it and easily visualize it? How can we measure the 'importance' of a set of discovered subgraphs in a large graph? Starting with the observation that real graphs often consist of stars, bipartite cores, cliques and chains, our main idea is to find the most succinct description of a graph in these 'vocabulary' terms. To this end, we first mine candidate subgraphs using one or more graph partitioning algorithms. Next, we identify the optimal summarization using the Minimum Description Length (MDL) principle, picking only those subgraphs from the candidates that together yield the best lossless compression of the graph—or, equivalently, that most succinctly describe its adjacency matrix.

Our contributions are three-fold: (a) *formulation*: we provide a principled encoding scheme to identify the vocabulary type of a given subgraph for six structure types prevalent in real-world graphs, (b) *algorithm*: we develop VoG, an efficient method to approximate the MDL-optimal summary of a given graph in terms of local graph structures, and (c) *applicability*: we report an extensive empirical evaluation on multi-million-edge real graphs, including Flickr and the Notre Dame web graph.

1 Introduction

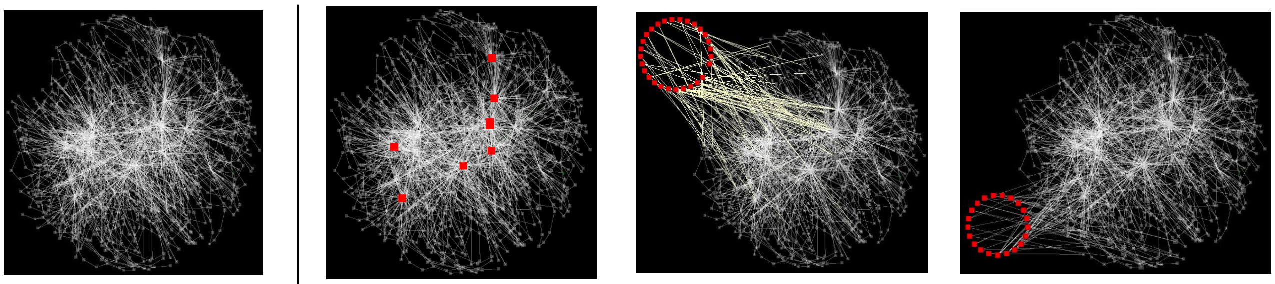
Given a large graph, such as the Facebook social network, what can we say about its structure? As most real graphs, the edge distribution will likely follow a power law [21], but apart from that, is it random? If not, how can we efficiently and in simple terms summarize which parts of the graph stand out, and how? The focus of this paper is exactly finding short summaries for large graphs, in order to gain a better understanding of their characteristics.

Why not apply one of the many community detection, clustering or graph-cut algorithms that abound in the literature [11, 15, 29, 38, 48], and summarize the graph in terms of its communities? The answer is that these algorithms do not quite serve our goal. Typically they detect numerous communities without explicit ordering, so a principled selection procedure of the most "important" subgraphs is still needed. In addition to that, these methods merely return the discovered communities, without characterizing them (e.g., clique, star), and, thus, do not help the user gain further insights in the properties of the graph.

In this paper, we propose VoG, an efficient and effective method to summarize and understand large real-world graphs. In particular, we aim at understanding graphs *beyond* the so-called "cavemen" networks that only consist of well-defined, tightly-knit clusters, which are known as cliques and near-cliques in graph terms.

The first insight is to best *describe* the structures in a graph using an enriched set of "vocabulary" terms: cliques and near-cliques (which are typically considered by community detection methods), and also stars, chains and (near) bipartite cores. The reasons we chose these "vocabulary" terms are: (a) (near-) cliques are included, and so our method works fine on "cavemen" graphs, and (b) stars [40], chains [58] and bipartite cores [30, 48] appear very often, and have semantic meaning (e.g., factions, bots) in the tens of real networks we have seen in practice (e.g., IMDB movie-actor graph, co-authorship networks, netflix movie recommendations, US Patent dataset, phonecall networks).

The second insight is to *formalize* our goal using the Minimum Description Length (MDL) principle [51] as a lossless



(a) *Original Wikipedia* `Controversy` *graph—plotted using the 'spring embedded' layout [26]. No structure stands out.*

(b) VOG*: 8 out of the 10 most informative structures are stars (their centers in red—Wikipedia editors, heavy contributors etc.).*

(c) VOG*: The most informative bipartite graph — 'edit war' — warring factions (one of them, in the top-left red circle), changing each-other's edits.*

(d) VOG*: the second most informative bipartite graph, another 'edit war', this one between vandals (bottom left circle of red points) vs. responsible editors (in white).*

Figure 1: *[Best viewed in color.]* VOG*: summarization and understanding of the most informative, from an information theoretic point of view, structures of the Wikipedia* `Controversy` *graph. Nodes stand for Wikipedia contributors and edges link users who edited the same part of the article. Without* VOG*, in 1a, no clear structures stand out.* VOG *spots stars in 1b (Wikipedia editors and other heavy contributors), and bipartite graphs in 1c and 1d (reflecting 'edit wars', i.e., editors reverting others' edits). Specifically, 1c shows the dispute between the two parties about a controversial topic and 1d shows vandals (red circles) vs responsible Wikipedia editors.*

compression problem. That is, by MDL we define the best summary of a graph as the set of subgraphs that describes the graph most succinctly, i.e., compresses it best, and, thus, helps a human understand the main graph characteristics in a simple, non-redundant manner. A big advantage is that our approach is *parameter-free*, as at any stage MDL identifies the best choice: the one by which we save most bits.

Informally, we tackle the following problem:

PROBLEM 1. (GRAPH SUMMARIZATION - INFORMAL)

- **Given**: *a graph*
- **Find**: *a set of possibly overlapping subgraphs*
- ***to most succinctly describe*** *the given graph, i.e., explain as many of its edges in as simple possible terms,*
- *in a* **scalable** *way, ideally linear on the number of edges.*

and our contributions can be summarized as:

1. **Problem Formulation:** We show how to formalize the intuitive concept of graph understanding using principled, information theoretic arguments.
2. **Effective and Scalable Algorithm:** We design VOG which is near-linear on the number of edges.
3. **Experiments on Real Graphs:** We empirically evaluate VOG on several real, public graphs spanning up to millions of edges. VOG spots interesting patterns like 'edit wars' in the Wikipedia graphs (Fig. 1).

This paper builds upon and expands on [33]. There are several differences from its earlier version: (i) We provide more details about the problem formulation and our algorithm, analyzing its time complexity, and giving illustrative examples and figures (Sections 3,4); (ii) We provide detailed analysis of the mined summaries and findings on real graphs (Section 5); (iii) We extended our discussion section to explain more questions that the reader may have (Section 6), and also included qualitative comparisons between our method and state-of-the-art techniques in the related work (Section 7).

The roadmap for this paper is as follows. First, Section 2 gives the overview and motivation of our approach. Next, in Sections 3 and 4 we respectively present the problem formulation and describe our method in detail. We empirically evaluate VOG in Section 5 using qualitative and quantitative experiments on a variety of real graphs. We discuss its implications and limitations in Section 6, and cover related work in Section 7. In Section 8 we round up with conclusions.

2 Proposed Method: Overview and Motivation

Before we give our two main contributions in the next sections – the problem formulation, and the search algorithm –, we first provide the high-level outline of VOG, which stands for *Vocabulary-based summarization of Graphs*:

(a) We use MDL to formulate a quality function: a collection M of structures (e.g., a star here, cliques there, etc) is as good as its description length $L(G, M)$. Hence, any subgraph or set of subgraphs has a quality score.

(b) We give an efficient algorithm for characterizing candidate subgraphs. In fact, we allow *any* subgraph discovery heuristic to be used for this, as we define our framework in general terms and use MDL to identify the structure *type* of the candidates.

(c) Given a candidate set $\mathcal{C}$ of promising subgraphs, we show how to mine informative summaries, removing redundancy by minimizing the compression cost.

VoG results in a list M of, possibly overlapping subgraphs, sorted in importance order (compression gain). Together these succinctly describe the main connectivity of the graph.

The motivation behind VoG is that people cannot easily understand large graphs that appear as a clutter of nodes and edges when visualized. On the other hand, a handful of simple structures are easily understood, and often meaningful. Next we give an illustrating example of VoG, where the most 'important' vocabulary subgraphs that constitute a Wikipedia article's (graph) summary are semantically interesting.

Illustrating Example: In Fig. 1 we give the results of VoG on the Wikipedia `Controversy` graph; the nodes are editors, and editors share an edge if they edited the same part of the article. Figure 1a shows the graph using the spring-embedded model [26]. No clear pattern emerges, and thus a human would have hard time understanding this graph. Contrast that with the results of VoG. Figures 1b–1d depict the same graph, where we highlight the most important structures (i.e., structures that save the most bits) discovered by VoG. The discovered structures correspond to behavioral patterns:

- *Stars → admins (+ vandals)*: in Fig. 1b, with red color, we show the centers of the most important "stars": further inspection shows that these centers typically correspond to administrators who revert vandalisms and make corrections.
- *Bipartite cores → edit wars*: Figs. 1c and 1d give the two most important near-bipartite-cores. Manual inspection shows that these correspond to *edit wars*: two groups of editors reverting each others' changes. For clarity, we denote the members of one group by red nodes (left), and hi-light the edges to the other group in pale yellow.

3 Problem Formulation

In this section we describe the first contribution, the MDL formulation of graph summarization. To enhance readability, we list the most frequently used symbols in Table 1.

In general, the Minimum Description Length principle (MDL) [52], is a practical version of Kolmogorov Complexity [39], which embraces the slogan *Induction by Compression*. For MDL, this can be roughly described as follows. Given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ minimizes

$$L(M) + L(\mathcal{D} \mid M) \ ,$$

where

- $L(M)$ is the length in bits of the description of M, and
- $L(\mathcal{D} \mid M)$ is the length, in bits, of the description of the data when encoded using the information in M.

This is called two-part or *crude* MDL, as opposed to *refined* MDL, where model and data are encoded together [24]. We use two-part MDL because we are specifically interested in the model: those graph connectivity structures that together best describe the graph. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases.

Without loss of generality, we here consider undirected graphs $G(\mathcal{V}, \mathcal{E})$ of $n = |\mathcal{V}|$ nodes, and $m = |\mathcal{E}|$ edges, with no self-loops. Our theory can be straightforwardly generalized to directed graphs—and similarly so for weighted graphs, has an expectation or is willing to make an assumption on the distribution of the edge weights.

To use MDL for graph summarization, we need to define what our models $\mathcal{M}$ are, how a model $M \in \mathcal{M}$ describes data, and how we encode this in bits. We do this next. It is important to note that to ensure fair comparison, MDL requires descriptions to be lossless, and, that in MDL we are only concerned with the optimal description *lengths* — not actual instantiated code words — and hence do not have to round up to the nearest integer.

3.1 MDL for Graph Summarization. As models M, we consider ordered lists of graph structures. We write Ω for the set of graph structure *types* that are allowed in M, i.e., that we are allowed to describe (parts of) the input graph with. We

Table 1: *Description of major symbols.*

Notation	**Description**
$G(\mathcal{V}, \mathcal{E})$	graph
$\mathbf{A}$	adjacency matrix of G
$\mathcal{V}$, $n = \lvert\mathcal{V}\rvert$	node-set and number of nodes of G respectively
$\mathcal{E}$, $m = \lvert\mathcal{E}\rvert$	edge-set and number of edges of G respectively
fc, nc	*full* clique and *near* clique respectively
fb, nb	*full* bipartite core and *near* bipartite core respectively
st	star graph
ch	chain graph
Ω	vocabulary of structure types, e.g., $\Omega \subseteq \{fc, nc, fr, nr, fb, nb, ch, st\}$
$\mathcal{C}_x$	set of all candidates structures of type $x \in \Omega$
$\mathcal{C}$	set of all candidate structures, $\mathcal{C} = \cup_x \mathcal{C}_x$
M	a model for G, essentially a list of node sets with associated structure types
$s, t \in M$	structures in M
$area(s)$	edges of G (= cells of $\mathbf{A}$) described by s
$\lvert S\rvert$, $\lvert s\rvert$	cardinality of set S and number of nodes in s respectively
$\lvert\lvert s\rvert\rvert$, $\lvert\lvert s\rvert\rvert'$	number of existing, resp. non-existing edges within the area of $\mathbf{A}$ that s describes
$\mathbf{M}$	approximation of adjacency matrix $\mathbf{A}$ deduced by M
$\mathbf{E}$	error matrix, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$
$\oplus$	exclusive OR
$L(G, M)$	number of bits to describe model M, and G using M
$L(M)$	number of bits to describe model M
$L(s)$	number of bits to describe structure s

will colloquially refer to Ω as our *vocabulary*. Although in principle any graph structure type can be a part of the vocabulary, we here choose the 6 most common structures in real-world graphs ([30, 48, 58]) that are well-known and understood by the graph mining community: *full* and *near* cliques (fc, nc), *full* and *near* bi-partite cores (fb, nb), stars (st), and chains (ch). Compactly, we have $\Omega = \{fc, nc, fb, nb, ch, st\}$. We will formally introduce these types after formalizing our goal.

Each structure $s \in M$ identifies a patch of the adjacency matrix $\mathbf{A}$ and describes how it is connected (Fig 2). We refer to this patch, or more formally the edges $(i, j) \in \mathbf{A}$ that structure s describes, as $area(s, M, \mathbf{A})$, where we omit M and $\mathbf{A}$ whenever clear from context.

We allow overlap between structures: nodes may be part of more than one structure. We allow, for example, cliques to overlap. Edges, however, are described on a first-come-first-serve basis: the first structure $s \in M$ to describe an edge (i, j) determines the value in $\mathbf{A}$. We do not impose constraints on the amount of overlap; MDL will decide for us whether adding a structure to the model is too costly w.r.t. the number of edges it helps to explain.

Let $\mathcal{C}_x$ be the set of all possible subgraphs of up to n nodes of type $x \in \Omega$, and $\mathcal{C}$ the union of all of those sets, $\mathcal{C} = \cup_x \mathcal{C}_x$. For example, $\mathcal{C}_{fc}$ is the set of all possible full cliques. Our model family $\mathcal{M}$ then consists of all possible permutations of all possible subsets of $\mathcal{C}$ – recall that the models M are *ordered* lists of graph structures. By MDL, we are after the $M \in \mathcal{M}$ that best balances the complexity of encoding both $\mathbf{A}$ and M.

Our general approach for transmitting the adjacency matrix is as follows. First, we transmit the model M. Then, given M, we can build the approximation $\mathbf{M}$ of the adjacency matrix, as defined by the structures in M; we simply iteratively consider each structure $s \in M$, and fill out the connectivity of $area(s)$ in $\mathbf{M}$ accordingly. As M is a summary, it is unlikely that $\mathbf{M} = \mathbf{A}$. Still, in order to fairly compare between models, MDL requires an encoding to be lossless. Hence, besides M, we also need to transmit the error matrix $\mathbf{E}$, which encodes the error w.r.t. $\mathbf{A}$. We obtain $\mathbf{E}$ by taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$, i.e., $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$. Once the recipient knows M and $\mathbf{E}$, the full adjacency matrix $\mathbf{A}$ can be reconstructed without loss.

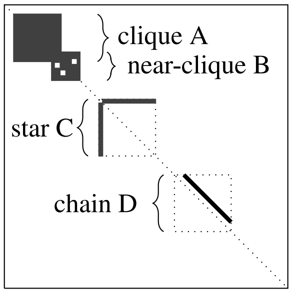


Figure 2: *Illustration of our main idea on a toy adjacency matrix:* VoG *identifies* overlapping *sets of nodes, that form vocabulary subgraphs (cliques, stars, chains, etc). With overlap,* VoG *allows for soft clustering of nodes, as in clique A and near-clique B. Stars look like inverted L shapes (e.g., star C). Chains look like lines parallel to the main diagonal (e.g., chain D).*

With this in mind, we have as our main score

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

where $L(M)$ and $L(\mathbf{E})$ are the numbers of bits that describe the structures, and the error matrix $\mathbf{E}$ respectively. We note that $L(\mathbf{E})$ maps to $L(\mathcal{D} \mid M)$, introduced in Sec. 3. That is, it corresponds to the length, in bits, of the description of the data when encoded using the information in M. The formal definition of the problem we tackle in this paper is defined as follows.

PROBLEM 2. (MINIMUM GRAPH DESCRIPTION PROBLEM) *Given a graph G with adjacency matrix A, and the graph structure vocabulary Ω, by the MDL principle we are after the smallest model M for which the total encoded length*

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

is minimal, where $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the error matrix, and $\mathbf{M}$ is an approximation of $\mathbf{A}$ deduced by M.

Next, we formalize the encoding of the model and the error matrix.

3.2 Encoding the Model.

For the encoded length of a model $M \in \mathcal{M}$, we have

$$L(M) \quad = \quad \underbrace{L_{\mathbb{N}}(|M| + 1) + \log \binom{|M| + |\Omega| - 1}{|\Omega| - 1}}_{\text{\# of structures, in total, and per type}} + \underbrace{\sum_{s \in M} \big(-\log \Pr(x(s) \mid M) + L(s) \big)}_{\text{per structure, in order, type and details}} \quad .$$

First, we transmit the total number of structures in M using $L_{\mathbb{N}}$, the MDL optimal encoding for integers ≥ 1 [52]. Next, by an index over a weak number composition, we optimally encode the number of structures of each type $x \in \Omega$ in model M. Then, for each structure $s \in M$, we encode its type $x(s)$ with an optimal prefix code [16], and finally its structure.

To compute the encoded length of a model, we need to define $L(s)$ per graph structure type in our vocabulary.

3.2.1 Cliques

To encode a *full clique*, a set of fully-connected nodes as a *full clique*, we first encode the number of nodes, and then their ids

$$L(fc) = \underbrace{L_{\mathbb{N}}(|fc|)}_{\text{\# of nodes}} + \underbrace{\log \binom{n}{|fc|}}_{\text{node ids}} \quad .$$

For the number of nodes we re-use $L_{\mathbb{N}}$, and we encode their ids by an index over an ordered enumeration of all possible ways to select $|fc|$ nodes out of n. As M generalizes the graph, we do not require that *fc is* a full clique in G. If only few edges are missing, it may still be convenient to describe it as such. Every missing edge, however, adds to the cost of transmitting $\mathbf{E}$.

As long as they stand out from the background distribution, less dense or *near*-cliques can be as interesting as full-cliques. We encode these as follows

$$L(nc) \quad = \quad \underbrace{L_{\mathbb{N}}(|nc|)}_{\text{\# of nodes}} + \underbrace{\log \binom{n}{|nc|}}_{\text{node ids}} + \underbrace{\log(|area(nc)|)}_{\text{\# of edges}} + \underbrace{||nc||l_1 + ||nc||'l_0}_{\text{edges}} \quad .$$

We first transmit the number and ids of nodes as above, and then identify which edges are present and which are not using optimal prefix codes. We write $||nc||$ and $||nc||'$ for resp. the number of present and missing edges in $area(nc)$. Then, $l_1 = -\log((||nc||/(||nc|| + ||nc||'))$, and analogue for l_0, are the lengths of the optimal prefix codes for resp. present and missing edges. The intuition is that the more dense (sparse) a near-clique is, the cheaper encoding its edges will be. Note that this encoding is exact; no edges are added to $\mathbf{E}$.

3.2.2 Bipartite Cores Bipartite cores are defined as non-empty, non-intersecting sets of nodes, A and B, for which there are edges only *between* the sets A and B, and not *within*.

The encoded length of a full bipartite core *fb* is

$$L(fb) = \underbrace{L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|)}_{\text{cardinality of} A \text{ resp. } B} + \underbrace{\log\binom{n}{|A|, |B|}}_{\text{node ids in } A \text{ and } B} \quad ,$$

where we encode the size of A, B, and then the node ids.

Analogue to cliques, we also consider near bi-partite cores, nb, where the core is not (necessarily) fully connected. To encode a near bi-partite core we have

$$L(nb) \quad = \quad \underbrace{L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|)}_{\text{cardinality of} A \text{ resp. } B} + \underbrace{\log\binom{n}{|A|, |B|}}_{\text{node ids in } A \text{ and } B} + \underbrace{\log(|area(nb)|)}_{\text{number of edges}} + \underbrace{||nb||l_1 + ||nb||'l_0}_{\text{edges}} \quad .$$

3.2.3 Stars A star is specific case of the bipartite core that consists of a single node (hub) in A connected to a set B of at least 2 nodes (spokes). For $L(st)$ of a given star st we have

$$L(st) = \underbrace{L_{\mathbb{N}}(|st| - 1)}_{\text{number of spokes}} + \underbrace{\log n}_{\text{id of hub node}} + \underbrace{\log\binom{n-1}{|st| - 1}}_{\text{ids of spoke nodes}} \quad ,$$

where $|st| - 1$ is the number of spokes of the star. To identify the member nodes, we first identify the hub out of n nodes, and then the spokes from the remaining nodes.

3.2.4 Chains A chain is a list of nodes such that every node has an edge to the next node, i.e. under the right permutation of nodes, $\mathbf{A}$ has only the super-diagonal elements (directly above the diagonal) non-zero. As such, for the encoded length $L(ch)$ for a chain ch we have

$$L(ch) = \underbrace{L_{\mathbb{N}}(|ch| - 1)}_{\text{\# of nodes in chain}} + \underbrace{\sum_{i=0}^{|ch|} \log(n - i)}_{\text{node ids, in order of chain}} \quad ,$$

where we first encode the number of nodes in the chain, and then their ids in order. Note that $\sum_{i=0}^{|ch|} \log(n - i) \leq |ch| \log n$, and hence by MDL is the better (i.e., as it is more efficient) way of the two to encode the member nodes of a chain.

3.3 Encoding the Error Next, we discuss how we encode the errors made by $\mathbf{M}$ with regard to $\mathbf{A}$, store this information in the *error* matrix $\mathbf{E}$. There exist many different approaches for encoding the errors—amongst which appealing at first glance is to simply identify all node pairs. However, it is important to realize that the more efficient our encoding is, the less spurious 'structure' will be discovered.

We hence follow [44] and encode $\mathbf{E}$ in two parts, $\mathbf{E}^+$ and $\mathbf{E}^-$. The former corresponds to the area of $\mathbf{A}$ that M does model, and for which $\mathbf{M}$ includes superfluous edges. Analogue, $\mathbf{E}^-$ consists of the area of $\mathbf{A}$ not modeled by M, for which $\mathbf{M}$ lacks edges. We encode these separately as they are likely to have different error distributions. Note that as we know near cliques and near bipartite cores are encoded exactly, we ignore these areas in $\mathbf{E}^+$. We encode the edges in $\mathbf{E}^+$ and $\mathbf{E}^-$ similarly to how we encode near-cliques, and have

$$\begin{aligned} L(\mathbf{E}^+) &= \log(|\mathbf{E}^+|) + ||\mathbf{E}^+||l_1 + ||\mathbf{E}^+||'l_0 \\ L(\mathbf{E}^-) &= \underbrace{\log(|\mathbf{E}^-|)}_{\text{\# of edges}} + \underbrace{||\mathbf{E}^-||l_1 + ||\mathbf{E}^-||'l_0}_{\text{edges}} \quad . \end{aligned}$$

That is, we first encode the number of 1s in $\mathbf{E}^+$ (resp. $\mathbf{E}^-$), after which we transmit the 1s and 0s using optimal prefix codes of length l_1 and l_0. We choose to use prefix codes over a binomial for practical reasons, as prefix codes allow us to easily and efficiently calculate accurate local gain estimates in our algorithm, without sacrificing much encoding efficiency (typically < 1 bit in practice).

Size of the Search Space. Clearly, for a graph of n nodes, the search space $\mathcal{M}$ we have to consider for solving the Minimum Graph Description Problem is enormous, as it consists of *all* possible permutations of the collection $\mathcal{C}$ of *all* possible structures over the vocabulary Ω. Unfortunately, it does not exhibit trivial structure, such as (weak) (anti)monotonicity, that we could exploit for efficient search. Further, Miettinen and Vreeken [45] showed that for a directed graph finding the MDL optimal model of only full-cliques is NP-hard. Hence, we resort to heuristics.

4 VoG: Summarization Algorithm

Now that we have the arsenal of graph encoding based on the vocabulary of structure types, Ω, we move on to the next two key ingredients: finding good candidate structures, i.e., instantiating $\mathcal{C}$, and then mining informative graph summaries, i.e., finding the best model M. An illustration of the algorithm is given in Fig. 3. The pseudocode of VoG is given in Algorithm 1, and the code is available for research purposes at `www.cs.cmu.edu/~dkoutra/SRC/VoG.tar`.

4.1 Step 1: Subgraph Generation.

Any combination of clustering and community detection algorithms can be used to decompose the graph into subgraphs, which need not be disjoint. These techniques include, but are not limited to *Cross-asssociations* [11], *Subdue* [15], SlashBurn [40], *Eigenspokes* [48], and *METIS* [29].

4.2 Step 2: Subgraph Labeling.

Given a subgraph from the set of clusters or communities discovered in the previous step, we search for the structure $x \in \Omega$ that best characterizes it, with no or some errors (e.g., perfect clique, or clique with some missing edges, encoded as error).

4.2.1 Step 2.1: Labeling Perfect Structures.

First, the subgraph is tested against the vocabulary structure types (full clique, full bipartite core, star and chain) for error-free match. The test for *clique* or *chain* is based on its degree distribution. Specifically, if all the nodes in the subgraph of size n have degree $n-1$, then it is a clique. Similarly, if all the nodes have degree 2 except for two nodes with degree 1, the subgraph is a chain. On the other hand, a subgraph is *bipartite* if the magnitudes of its maximum and minimum eigenvalues are equal. To find the node ids in the two node sets, A and B, we use Breadth First Search (BFS) with node coloring. We note that a star is a special case of a bipartite graph, where one set consists of only one node. If one of the node sets has size 1, then the given substructure is encoded as *star*.

4.2.2 Step 2.2: Labeling Approximate Structures.

If the subgraph does not have a "perfect" structure (i.e., it is *not* a full clique, full bipartite core, star, or chain), the search continues for the vocabulary structure type that, in MDL terms, best approximates the subgraph. To this end, we encode the subgraph as each of the 6 candidate vocabulary structures, and choose the structure that has the lowest encoding cost.

Let m^* be the graph model with only one subgraph encoded as structure $\in \Omega$ (e.g., clique) and the additional edges included in the error matrix. For reasons of efficiency, instead of calculating the full cost $L(G, m^*)$ as the encoding cost of each subgraph representation, we estimate the *local* encoding cost $L(m^*) + L(\mathbf{E}^+_{m^*}) + L(\mathbf{E}^-_{m^*})$, where $\mathbf{E}^+_{m^*}$ and $\mathbf{E}^-_{m^*}$ encode the incorrectly modeled, and unmodeled edges respectively (Sec. 3). The challenge of the step is to efficiently identify the role of each node in the subgraph (e.g., hub/spoke in a star, member of set A or B in a near-bipartite core, order of nodes in chain) for the MDL representation. We elaborate on each structure next.

- **Clique:** This representation is straightforward, as all the nodes have the same structural role. All the nodes are members of the clique or the near-clique. For the full clique, the missing edges are stored in a *local* error matrix, $\mathbf{E}_{fc}$, in order to obtain an estimate of the global encoding cost $L(fc) + L(\mathbf{E}^+_{fc}) + L(\mathbf{E}^-_{fc})$. For near-cliques we ignore $\mathbf{E}_{nc}$, and, so, the encoding cost is $L(nc)$.
- **Star:** Representing a given subgraph as a near-star is straightforward as well. We find the highest degree node (in case of a tie, we choose one randomly), and set it to be the hub of the star, and identify the rest nodes as the peripheral nodes — which are also referred to as spokes. The additional or missing edges are stored in the local Error matrix, $\mathbf{E_{st}}$. The MDL cost of this encoding is computed as $L(st) + L(\mathbf{E}^+_{st}) + L(\mathbf{E}^-_{st})$.
- **Bipartite core:** In this case, the problem of identifying the role of each node reduces to finding the maximum bipartite graph, which is known as max-cut problem, and is NP-hard. The need of a scalable graph summarization algorithm

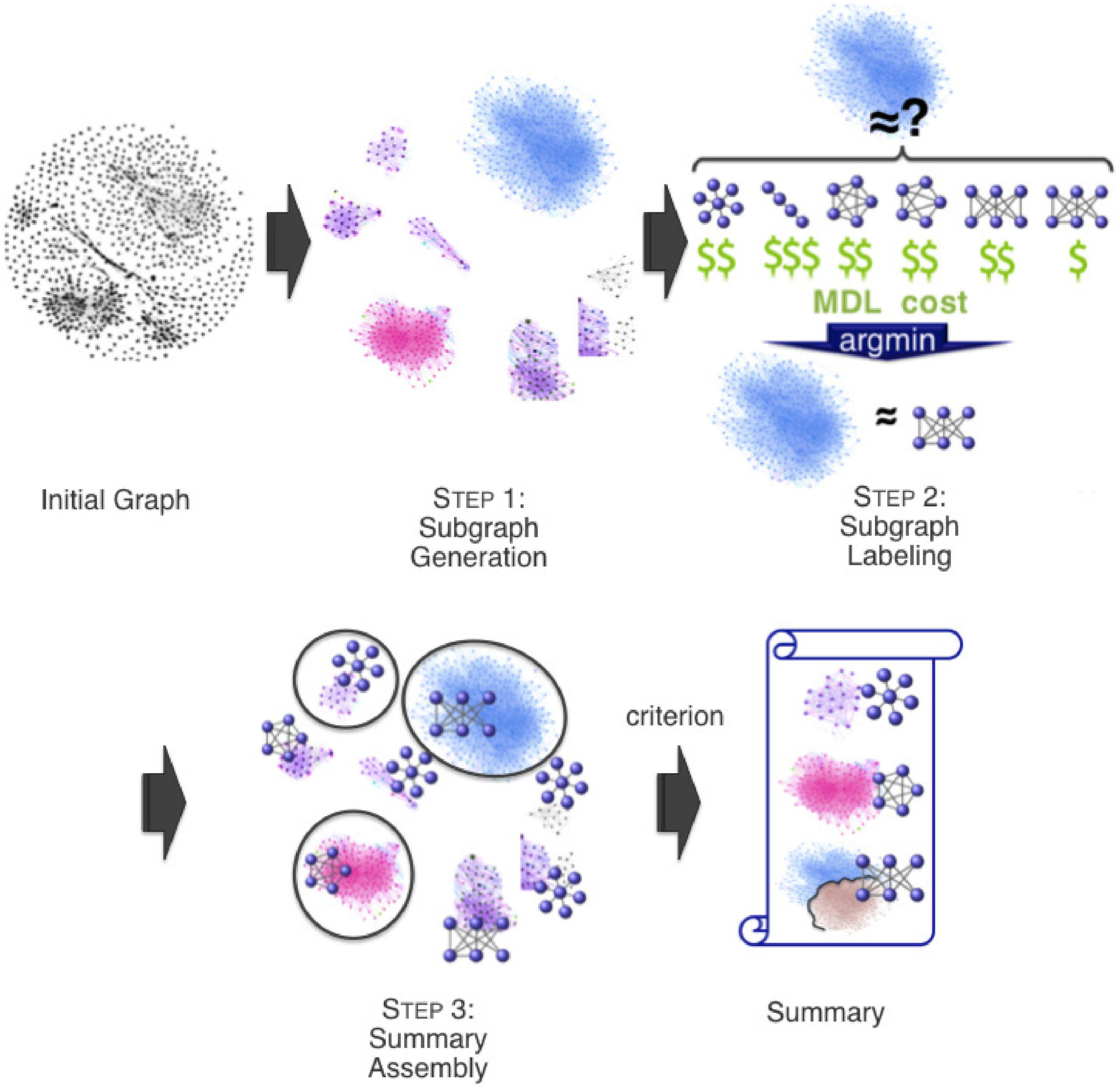


Figure 3: *Illustration of* VOG *step-by-step.*

makes us resort to approximation algorithms. In particular, finding the maximum bipartite graph can be reduced to semi-supervised classification. We consider two classes which correspond to the two node sets, A and B, of the bipartite graph, and the prior knowledge is that the highest-degree node belongs to A, and its neighbors to B. To propagate these classes/labels, we employ Fast Belief Propagation (FaBP) [35] assuming heterophily (i.e., connected nodes belong to different classes). For near-bipartite cores $L(\mathbf{E}_{nb}^{+})$ is omitted.

- **Chain:** Representing the subgraph as a chain reduces to finding the longest path in it, which is also NP-hard. We therefore, employ the following heuristic. Initially, we pick a node of the subgraph at random, and find its furthest node using BFS (temporary start). Starting from the latter and by using BFS again, we find the subsequent furthest node (temporary end). We then extend the chain by local search. Specifically, we consider the subgraph from which all the nodes that already belong to the chain, except for its endpoints, are removed. Then, starting from the endpoints we employ again BFS. If new nodes are found during this step, they are added in the chain (rendering it a near-chain with few loops). The nodes of the subgraph that are not members of this chain are encoded as error in $\mathbf{E}_{ch}$.

After representing the subgraph as each of the vocabulary structures x, we employ MDL to choose the representation with the minimum (local) encoding cost, and add the structure to the candidate set, $\mathcal{C}$. Finally, we associate the candidate

Algorithm 1 VoG

```
Input: graph $G$
Step 1: Subgraph Generation. Generate candidate – possibly overlapping – subgraphs using one or more graph decomposition methods.
Step 2: Subgraph Labeling. Characterize the type of each subgraph $x \in \Omega$ using MDL, identify the type $x$ as the one that minimizes the local encoding cost. Populate the candidate set $\mathcal{C}$ accordingly.
Step 3: Summary Assembly. Use the heuristics Plain, Top10, Top100, Greedy'nForget (Sec. 4.3) to select a non-redundant subset from the candidate structures to instantiate the graph model $M$. Pick the model of the heuristic with the lowest description cost.
return graph summary $M$ and its encoding cost.
```

structure with its encoding benefit: the savings in bits for encoding the subgraph by the minimum-cost structure type, instead of leaving its edges unmodeled and including them in the error matrix.

4.3 Step 3: Summary Assembly. Given a set of candidate structures, $\mathcal{C}$, how can we efficiently induce the model M that is the best graph summary? The exact selection algorithm, which considers all the possible ordered combinations of the candidate structures and chooses the one that minimizes the cost, is combinatorial, and cannot be applied to any non-trivial candidate set. Thus, we need heuristics that will give a fast, approximate solution to the description problem. To reduce the search space of all possible permutations, we attach to each candidate structure a quality measure, and consider them in order of decreasing quality. The measure that we use is the encoding benefit of the subgraph, which, as mentioned before, is the number of bits that are gained by encoding the subgraph as structure x instead of noise. Our constituent heuristics are:

- Plain*:* The baseline approach gives as graph summary all the candidate structures, i.e., $M = \mathcal{C}$.
- Top-k*:* Selects the top-k candidate structures as sorted according to decreasing quality.
- Greedy'nForget*:* Considers each structure in $\mathcal{C}$ sequentially, sorted by descending quality, and iteratively includes each in M: as long as the total encoded cost of the graph does not increase, keeps the structure in M, otherwise it removes it. Greedy'nForget continues this process until all the structures in $\mathcal{C}$ have been considered. This heuristic is more computationally demanding than the plain or top-k heuristics, but still handles large sets of candidates structures efficiently.

VoG employs all the heuristics and by MDL picks the overall best graph summarization, or equivalently, the summarization with the minimum description cost.

4.4 Toy Example. To illustrate how VoG works, we give an example on a toy graph. We apply VoG on the synthetic `Cavemen` graph of 841 nodes and 7 547 edges, which as shown in Fig. 4 consists of two cliques separated by two stars. The leftmost and rightmost cliques consist of 42, and 110 nodes respectively; the big star (2nd structure) has 800 nodes, and the small star (3rd structure) 91 nodes. Here is how VoG works step-by-step:

Step 1: The *raw* output of the decomposition algorithm consists of the subgraphs corresponding to the stars, the full left-hand and right-hand cliques, as well as subsets of these nodes.
Step 2: Through MDL, VoG correctly identifies the type of these structures.
Step 3: Finally, via Greedy'nForget it automatically finds the true 4 structures without redundancy, and drops the structures that consist of subsets of nodes.

The corresponding model requires 36% fewer bits than the 'empty' model, where the graph edges are encoded as noise. We note that one bit gain already corresponds to twice the likelihood.

4.5 Time Complexity of VoG. For a graph $G(\mathcal{V}, \mathcal{E})$ of $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, the time complexity of VoG depends on the runtime complexity of the algorithms that compose it, namely the decomposition algorithm, the subgraph labeling, the encoding scheme $L(G, M)$ of the model, and the structure selection (summary assembly).

For the *decomposition* of the graph, we use SlashBurn which is near-linear on the number of edges of real graphs [40]. The *subgraph labeling* algorithms in Sec. 4 are carefully designed to be linear on the number of edges of the input subgraph.

When there is no overlap between the structures in M, the complexity of calculating the *encoding scheme* $L(G, M)$ is $O(m)$. When there is overlap, the complexity is bigger: assume that s, t are two structures $\in M$ with overlap, and t has higher quality than s, i.e., t comes before s in the ordered list of structures. Finding how much 'new' structure (or area in $\mathbf{A}$)

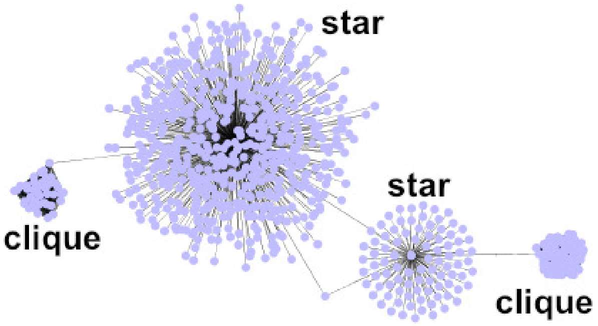


Figure 4: *Toy graph:* VoG *saves 36% in space, by successfully discovering the two cliques and two stars that we chained together.*

s explains relative to t costs $O(|M|^2)$. Thus, in the case of overlapping subgraphs, the complexity of computing the encoding scheme is $O(|M|^2 + m)$. As typically $|M| \ll m$, in practice we have $O(m)$.

As far as the *selection method* is concerned, the TOP-K heuristic that we propose has complexity $O(k)$. The GREEDY'NFORGET heuristic has runtime $O(|\mathcal{C}| \times o \times m)$, where $|\mathcal{C}|$ is the number of structures identified by VoG, and o the time complexity of $L(G, M)$.

5 Experiments

In this section, we aim to answer the following questions:

Q1. Are the real graphs structured, or random and noisy?

Q2. What structures do the graph summaries consist of, and how can they be used for understanding?

Q3. Is VoG scalable and able to efficiently summarize large graphs?

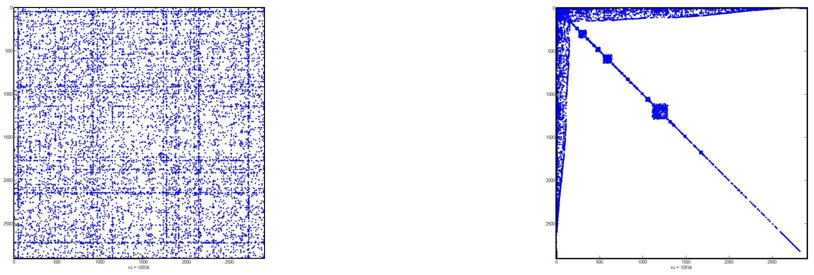
The graphs we use in the experiments along with their descriptions are summarized in Table 2. `Controversy` is a co-editor graph on a known Wikipedia controversial topic (name withheld for obvious reasons), where the nodes are users and edges mean that they edited the same sentence. `Chocolate` is a co-editor graph on the 'Chocolate' article. The descriptions of the other datasets are given in Table 2.

Table 2: *Summary of graphs used.*

Name	Nodes	Edges	Description
`Flickr` [23]	404 733	2 110 078	Friendship social network
`WWW-Barabasi` [56]	325 729	1 090 108	WWW in nd.edu
`Epinions` [56]	75 888	405 740	Trust graph
`Enron` [19]	80 163	288 364	Enron email
`AS-Oregon` [7]	13 579	37 448	Router connections
`Wikipedia-Controversy`	1 005	2 123	Co-edit graph
`Wikipedia-Chocolate`	2 899	5 467	Co-edit graph

Graph Decomposition. In our experiments, we modify SLASHBURN [40], a node reordering algorithm, to generate candidate subgraphs. The reasons we use SLASHBURN are (a) it is scalable, and (b) it is designed to handle graphs *without* "cavemen" structure. We note that VoG would only benefit from using the outputs of additional decomposition algorithms.

SLASHBURN is an algorithm that reorders the nodes so that the resulting adjacency matrix has clusters or patches of non-zero elements. The idea is that removing the top high-degree nodes in real world graphs results in the generation of many small-sized disconnected components (subgraphs), and one giant connected component whose size is significantly smaller compared to the original graph. Specifically, it performs two steps iteratively: (a) It removes top high degree nodes from the original graph; (b) It reorders the nodes so that the high-degree nodes go to the front, disconnected components to back, and the giant connected component (GCC) to the middle. During the next iterations, these steps are performed on the giant connected component. A good node-reordering method will reveal patterns, as well as large empty areas. as shown in Figure 5 on the Wikipedia `Chocolate` network.

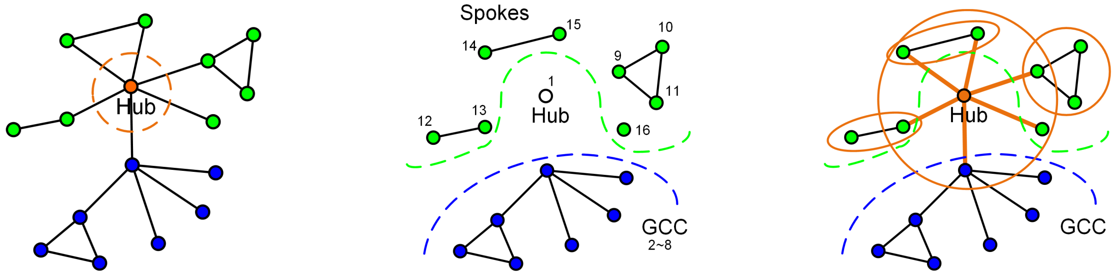


(a) *Original.*

(b) *After re-ordering.*

Figure 5: *Adjacency matrix before and after node-ordering on the Wikipedia* `Chocolate` *graph. Large empty (and dense) areas appear, aiding the graph decomposition step of* VoG *and the discovery of candidate structures.*

In this paper, SLASHBURN is modified to decompose the input graph. In more details, we focus on the first step of the algorithm, which removes the high degree node by "burning" its edges. This step is depicted for a toy graph in Fig. 6b, where the green dotted line shows which edges were "burnt". Then, the hub with its egonet, which consists of the hub's one hop away neighbors and the connections between them, form the first candidate structures. Moreover, the connected components with size greater or equal to two and smaller than the size of the GCC, consist additional candidate structures (see Fig. 6c). In the next iteration, the same procedure is applied to the giant connected component, yielding this way a set of candidate structures. We use MDL to determine the best-fitting type per discovered candidate structure.



(a) *Initial toy graph.*

(b) SLASHBURN *on the toy graph.*

(c) *Candidate structures (in circles).*

Figure 6: *Illustration of the graph decomposition and the generation of the candidate structures.*

5.1 Q1: Quantitative Analysis. In this section we apply VoG to the real datasets of Table 2, and evaluate the achieved description cost, and edge coverage, which are indicators of the discovered structures. The evaluation is done in terms of savings w.r.t. the base encoding (ORIGINAL) of the adjacency matrix of a graph with an empty model M.

Although we refer to the description cost of the summarization techniques, we note that compression itself is *not* our goal, but our *means* for identifying structures important for graph understanding or attention routing. This is also why it does not make sense to compare VoG against standard matrix compression techniques: whereas VoG has the goal of describing a graph with intelligible structures, specialized algorithms may exploit any statistical correlations to save bits.

We compare two summarization approaches: (a) ORIGINAL: The whole adjacency matrix is encoded as if it contains no structure; that is, $M = \emptyset$, and all of $\mathbf{A}$ is encoded through $L(\mathbf{E}^-)$; and (b) VoG, our proposed summarization algorithm with the three selection heuristics (PLAIN, TOP10 and TOP100, GREEDY'NFORGET[1]). We ignore very small structures; the candidate set $\mathcal{C}$ includes subgraphs with at least 10 nodes, except for the Wikipedia graphs where the size threshold is set to 3 nodes. Among the summaries obtained by the different heuristics, we choose the one that yields the smallest description length.

Table 3 presents the summarization cost of each technique w.r.t. the cost of the ORIGINAL approach, as well as the fraction of the edges that remains unexplained. The lower the ratios (i.e., the lower the obtained description length), the more

[1] By carefully designing the GREEDY'NFORGET heuristic to exploit memoization, we are able to efficiently compute the best structures within the candidate set. Although restricting our search space to a small number of candidate structures –ranked in decreasing order of quality– can yield faster results, we report results on the whole search space.

Table 3: *[Lower is better.] Quantitative analysis of* VoG *with different summarization heuristics:* Plain, Top10, Top100, *and* Greedy'nForget. *The first column,* Original, *presents the cost, in bits, of encoding the adjacency matrix with an empty model M. For different heuristics we show the relative number of bits needed to describe the adjacency matrix. In the last four columns, we give the fraction of edges that are* not *explained by the structures in the model, M, that each heuristic finds. The lowest description cost is in bold.*

		VoG							
		Compression				**Unexplained edges**			
Graph	**Original** (bits)	**Plain**	**Top10**	**Top100**	**Greedy'n Forget**	**Plain**	**Top10**	**Top100**	**Greedy'n Forget**
`Flickr`	35 210 972	**81%**	99%	97%	95%	4%	72%	39%	36%
`WWW-Barabasi`	18 546 330	**81%**	98%	96%	85%	3%	62%	51%	38%
`Epinions`	5 775 964	82%	98%	95%	**81%**	6%	65%	46%	14%
`Enron`	4 292 729	**75%**	98%	93%	**75%**	2%	77%	46%	6%
`AS-Oregon`	475 912	72%	87%	79%	**71%**	4%	59%	25%	12%
`Chocolate`	60 310	96%	96%	93%	**88%**	4%	70%	35%	27%
`Controversy`	19 833	98%	94%	96%	**87%**	5%	51%	12%	31%

structure is identified. For example, VoG-Plain describes `Flickr` with only 81% of the bits of the Original approach , and explains all but 4% of the edges, which means that 4% of the edges are not encoded by the structures in M.

Observation 1. *Real graphs do have structure;* VoG, *with or without structure selection, achieves better compression than the* Original *approach that assumes no structure.*

Greedy'nForget finds models M with fewer structures than Plain and Top100–which is important for graph understanding and guiding attention to few structures–, and often obtains (much) more succinct graph descriptions. This is due to its ability to identify structures that are informative *with regard to what it already knows*. In other words, structures that highly overlap with ones already selected into M will be much less rewarded than structures that explain unexplored parts of the graph.

5.2 Q2: Qualitative Analysis.

In this section, we showcase how to use VoG and interpret its output.

5.2.1 Graph Summaries.

How does VoG summarize real graphs? Which are the most frequent structures? Table 4 shows the summarization results of VoG for different structure selection techniques.

Table 4: *Summarization of graphs by* VoG *(for different heuristics). The most frequent structures are the stars (st) and near-bipartite cores (nb). For each graph and selection technique (heuristic), we provide the frequency of each structure type: 'st' for star, 'nb' for near-bipartite cores, 'fc' for full cliques, 'fb' for full bipartite-cores, 'ch' for chains, and 'nc' for near-cliques.*

	Plain						**Top10**		**Top100**				**Greedy'nForget**			
Graph	**st**	**nb**	**fc**	**fb**	**ch**	**nc**	**st**	**nb**	**st**	**nb**	**fb**	**ch**	**st**	**nb**	**fc**	**fb**
`Flickr`	24 385	3 750	281	9	-	3	10	-	99	1	-	-	415	-	-	1
`WWW-Barabasi`	10 027	1 684	487	120	26	-	9	1	83	14	3	-	403	7	-	16
`Epinions`	5 204	528	13	-	-	-	9	1	99	1	-	-	2 738	-	8	-
`Enron`	3 171	178	3	11	-	-	9	1	99	1	-	-	2 323	3	3	2
`AS-Oregon`	489	85	-	4	-	-	10	-	93	6	1	-	399	-	-	-
`Chocolate`	170	58	-	-	17	-	9	1	87	10	-	3	101	-	-	-
`Controversy`	73	21	-	1	22	-	8	2	66	17	1	16	35	-	-	-

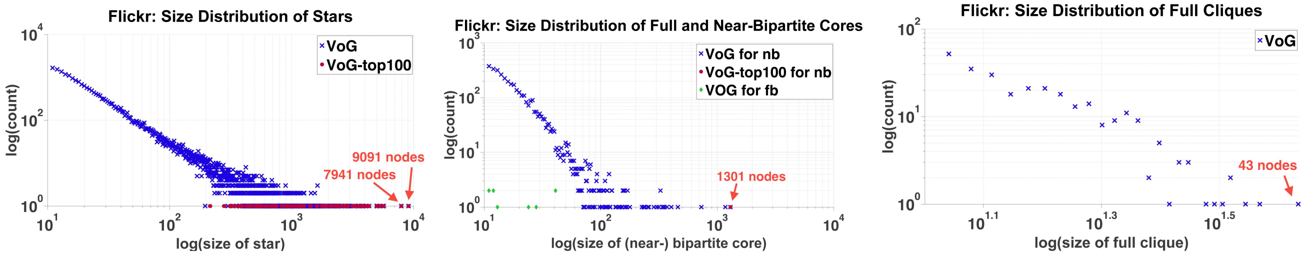
Observation 2. *The summaries of all the selection heuristics consist mainly of stars, followed by near-bipartite cores. In some graphs, like* `Flickr` *and* `WWW-Barabasi`, *there is a significant number of full cliques.*

From Table 4 we also observe that GREEDY'NFORGET drops uninteresting structures, and reduces the graph summary. Effectively, it filters out the structures that explain edges already explained by structures in model M.

In order to gain a better understanding of the structures that VOG finds, in Figures 7 and 8 we give the size distributions of the most frequent structures in the `Flickr` social network, the `WWW-Barabasi` web graph and the `Enron` email network.

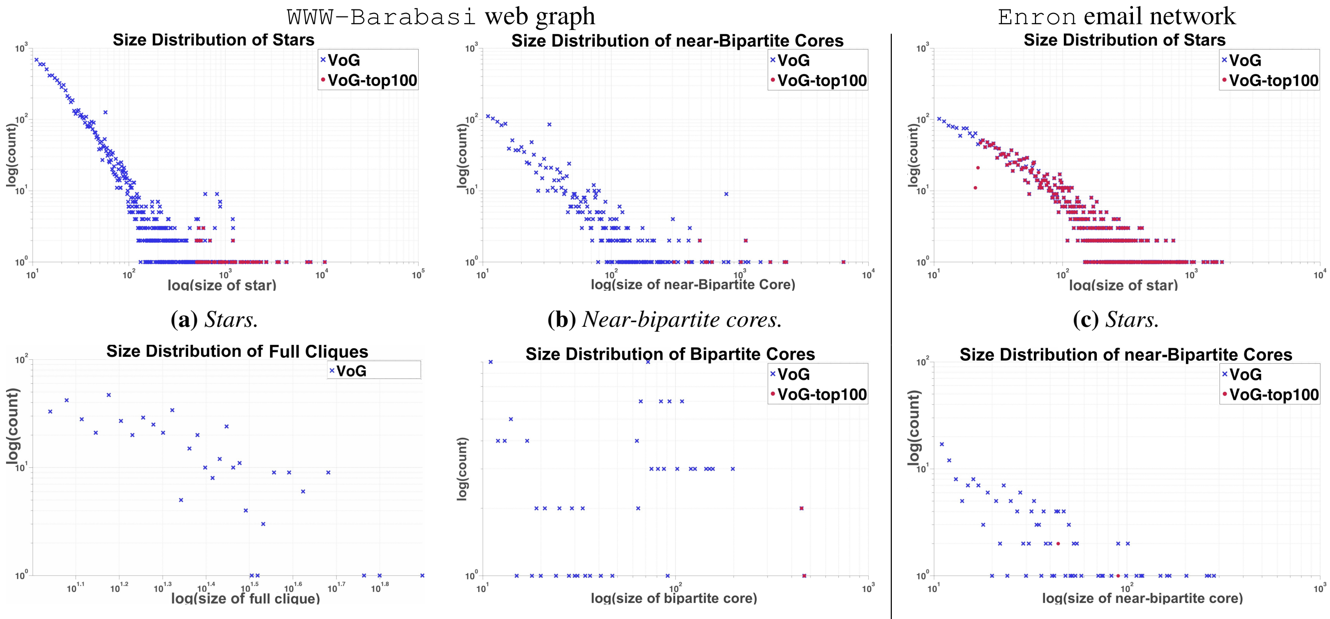
OBSERVATION 3. *The size distribution of the stars and near-bipartite cores follows a power law.*

Moreover, the distribution of the size of the full cliques in `Flickr` follows a power law as well. As far as the distribution of the size of full cliques and bipartite cores in `WWW-Barabasi` is concerned, there is no clear pattern. In Figures 7 and 8, we denote with blue crosses the size distribution of the structures discovered by VOG-PLAIN, and with red circles the size distribution for the structures found by VOG with the TOP100 heuristic.



(a) *Stars.* **(b)** *Bipartite and near-bipartite cores.* **(c)** *Full cliques.*

Figure 7: `Flickr`*: The size of the stars, the near-bipartite cores and full cliques follows the power law distribution. Distribution of the size of the most informative, from an information theoretic point of view, structures by* VOG *(blue crosses) and* VOG-TOP100 *(red circles).*



(a) *Stars.* **(b)** *Near-bipartite cores.* **(c)** *Stars.*

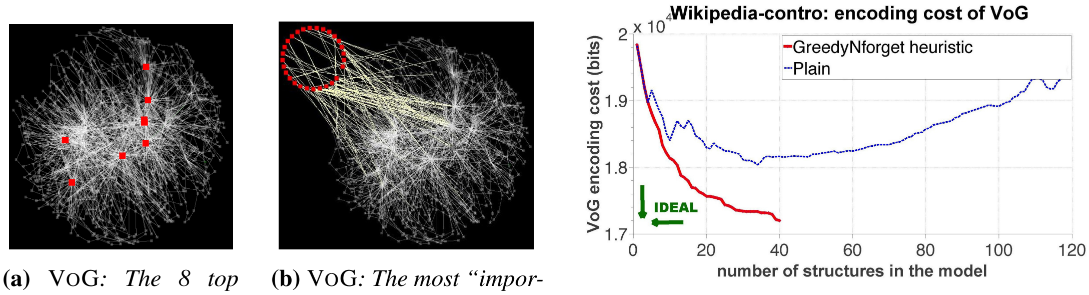
(d) *Full cliques.* **(e)** *Bipartite cores.* **(f)** *Near-bipartite cores.*

Figure 8: *The distribution of the size of the most 'interesting', from the MDL point of view, structures (stars, near-bipartite cores) follow the power law distribution both in the* `WWW-Barabasi` *web graph (left) and the* `Enron` *email network (right). The distribution of structures discovered by* VOG *and* VOG-*TOP100 are denoted by blue crosses and red circles respectively.*

5.2.2 Graph Understanding.

Are the 'important' structures found by VOG semantically meaningful? For sense-making, we analyze the discovered subgraphs in the non-anonymized real datasets `Controversy`, `Chocolate` and `Enron`.

Wikipedia–Controversy. Figs. 1 and 9(a-b) illustrate the original and VoG-based visualization of the `Controversy` graph. The VoG-Top10 summary consists of 8 stars and 2 near-bipartite cores (see also Table 4). The 8 star configurations correspond mainly to administrators, such as "Future_Perfect_at_sunrise", who do many minor edits in various parts of the article and also revert vandalisms. The most interesting structures VoG identifies are the near-bipartite cores, which reflect: (i) the conflict between the two parties, and (ii) an "edit war" between vandals and administrators or loyal Wikipedia users.



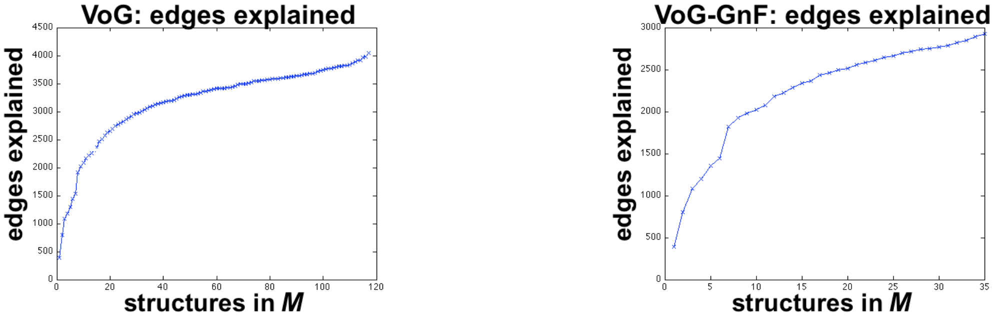
(a) VoG*: The 8 top "important" stars whose centers are denoted with red rectangles.*

(b) VoG*: The most "important" bipartite graph (node set A denoted by the circle of red points).*

(c) *Effectiveness of* Greedy'nForget *(in red). Encoding cost of* VoG *vs. number of structures in the model, M.*

Figure 9: *The* VoG *summary of the* `Controversy` *graph, and effectiveness of the* Greedy'nForget *heuristic. The top 10 structures of the graph summary of* VoG *consist of 8 stars (Fig. 9a) and 2 bipartite graphs (Fig. 9b, 1d). Figure 9c shows the encoding cost for the* Plain *(dotted blue line) and* Greedy'nForget *(solid red line) heuristics.* Greedy'nForget *minimizes the encoding cost by greedily selecting from a sorted, in decreasing quality order, set of structures the ones that reduce the cost. Thus,* Greedy'nForget *leads to better encoding costs and smaller summaries (here only 40 are chosen) than* Plain *(~120 structures).*

In Fig. 9c, the encoding cost of VoG is given as a function of the selected structures. The dotted blue line corresponds to the cost of the Plain encoding, where the structures are added sequentially in the model M, in decreasing order of quality (local encoding benefit). The solid red line maps to the cost of the Greedy'nForget heuristic. Given that the goal is to summarize the graph in the most succinct way, and at the same time achieve low encoding cost, Greedy'nForget is effective. Finally, in Fig. 10 we consider the models M with increasing number of structures (in decreasing quality order) and show the number of edges that each one explains. Specifically, Fig. 10a refers to the models that consist of ordered subsets of all the structures (~120) that VoG-Plain discovered and Fig. 10b refers to models incorporating ordered subsets of the ~35 structures kept by VoG-Greedy'nForget. We note that the slope in Fig. 10a is steep for the first ~35 structures, and then increases with small rate, which means that the new structures that are added in the model M explain few new edges (diminishing returns). On the other hand, the edges explained by the structures discovered by VoG-Greedy'nForget increase with higher rate, which signifies that VoG-Greedy'nForget drops uninteresting structures that explain edges already explained by structures in model M.

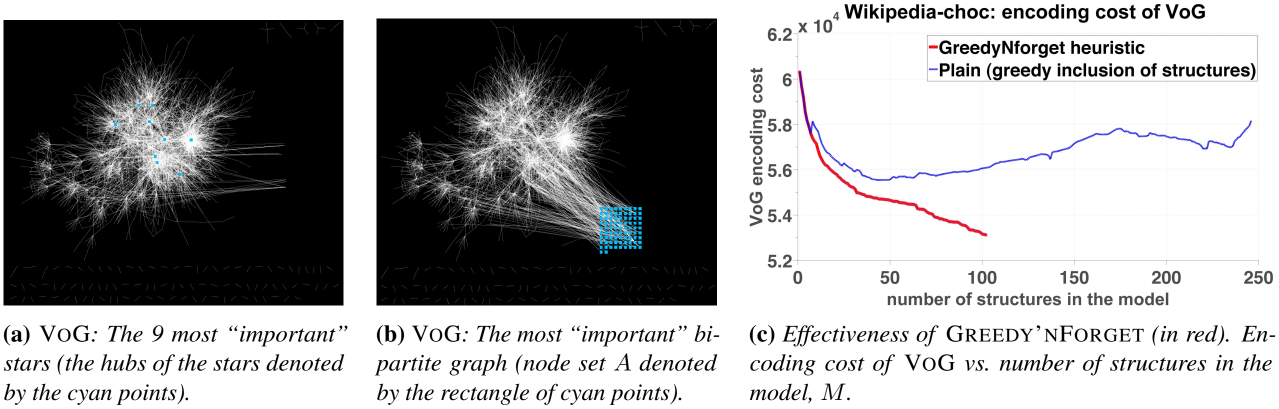


(a) VoG.

(b) VoG-GnF.

Figure 10: *Wikipedia-*`Controversy`*:* VoG-GnF *successfully drops uninteresting structures that explain edges already explained by structures in model M. Edges explained by models with increasing number of structures generated by* VoG *and* VoG-GnF.

Wikipedia–Chocolate The visualization of Wikipedia `Chocolate` is similar to the visualization of the `Controversy` and is given in Fig. 11 for completeness. As shown in Table 4, the Top10 summary of `Chocolate`



(a) VoG*: The 9 most "important" stars (the hubs of the stars denoted by the cyan points).*

(b) VoG*: The most "important" bipartite graph (node set A denoted by the rectangle of cyan points).*

(c) *Effectiveness of* GREEDY'NFORGET *(in red). Encoding cost of* VoG *vs. number of structures in the model, M.*

Figure 11: VoG*: summarization of the structures of the Wikipedia* `Chocolate` *graph. The top-10 structures of the* VoG *graph summary consist of 9 stars (11a) and one bipartite graph (11b). In (11c) the* GREEDY'NFORGET *(red line) heuristic reduces the encoding cost by keeping about 100 most important of 250 identified structures. The blue line corresponds to the encoding cost of greedily adding the identified structures in decreasing order of encoding benefit.*

contains 9 stars and 1 near-bipartite core. The center of the highest ranked star corresponds to "Chobot", a Wikipedia bot that fixes interlanguage links, and thus touches several, possibly unrelated parts of a page. Other stars have as hubs administrators, who do many minor edits, as well as heavy contributors. The near-bipartite core captures the interactions between possible vandals and administrators (or Wikipedia contributors) who were reverting each other's edits resulting in temporary (semi-) protection of the webpage. Figure 11c illustrates the original and VoG-based visualization of the who-edits-whose-text graph for the article on chocolate.

Enron. The TOP10 summary for Enron has nine stars and one near-bipartite core. The centers of the most informative stars are mainly high ranking officials (e.g., Kenneth Lay with two email accounts, Jeff Skilling, Tracey Kozadinos). As a note, Kenneth Lay was long-time Enron CEO, while Jeff Skilling had several high-ranking positions in the company, including CEO and managing director of Enron Capital & Trade Resources. The big near-bipartite core in Fig. 12 is loosely connected to the rest of the graph, and represents the email communication about an extramarital affair, which was broadcast to 235 recipients. The small bipartite graph depicted in the same spy plot captures the email activity of several employees about a skiing trip on New Year.

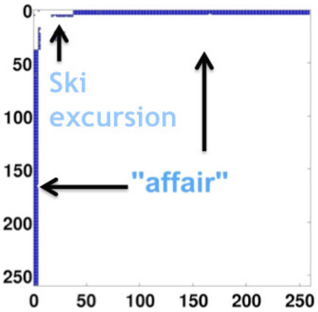


Figure 12: `Enron`*: Adjacency matrix of the top near-bipartite core found by* VoG*, corresponding to email communication about an "affair", as well as for a smaller near-bipartite core found by* VoG *representing email activity regarding a skiing trip.*

5.3 Q3: Scalability of VoG

In Fig. 13, we present the runtime of VoG with respect to the number of edges in the input graph. For this purpose, we induce subgraphs of Notre Dame dataset (`WWW-Barabasi`) for which we give the dimensions in Table 5. We ran the experiments on a Intel(R) Xeon(R) CPU 5160 at 3.00GHz, with 16GB memory. The structure identification is implemented in Matlab, while the selection process in Python. A discussion about the runtime of VoG is also given in the supplementary material.

OBSERVATION 4. *All the steps of* VoG *are designed to be scalable. Figure 13 shows the complexity is $O(m)$, i.e.,* VoG *is near-linear on the number of edges of the input graph.*

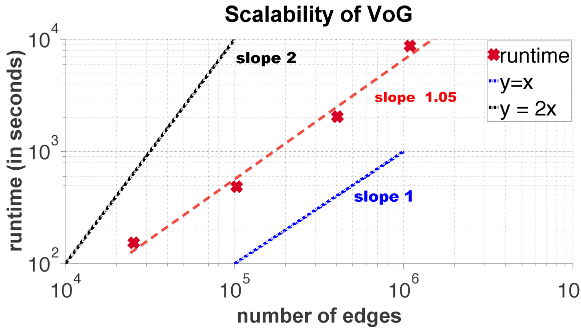


Figure 13: VoG *is near-linear on the number of edges. Runtime, in seconds, of* VoG *(*PLAIN*) vs. number of edges in graph. For reference we show the linear and quadratic slopes.*

Name	Nodes	Edges
`WWW-Barabasi-50k`	49 780	50 624
`WWW-Barabasi-100k`	99 854	205 432
`WWW-Barabasi-200k`	200 155	810 950
`WWW-Barabasi-300k`	325 729	1 090 108

Table 5: *Scalability: Induced subgraphs of* `WWW-Barabasi`.

6 Discussion

The experiments show that VoG successfully solves an important open problem in graph understanding: how to find a succinct summary for a large graph. Here we address some questions, that a reader may have.

Question 1: Why does VoG use the chosen vocabulary structures consisting of stars, (near-) cliques, (near-) bipartite cores and chains, and not other structures?
Answer 1: We noticed that these structures appear very often, in tens of real graphs, (e.g., in patent citation network, in phone-call networks, in netflix recommendation system, etc), while they also have semantic meaning, such as factions or popular entities. Moreover, these graph structures are well-known and conceptually simple, making the summaries that VoG discovers easily interpretable.

Question 2: What if a new structure such as “loops” proves to be frequent in real graphs? How can VoG handle this case?
Answer 2: VoG can easily be extended to handle new vocabulary terms. The key insight for the vocabulary term encodings in Section 3 is to encode the necessary information as succinctly as possible. In fact, as by MDL we can straightforwardly compare two or more model classes, MDL will immediately tell us whether a vocabulary set $\mathcal{V}_1$ is better than a vocabulary set $\mathcal{V}_2$: the one that gives the best compression cost for the graph wins!

Question 3: Why fix the vocabulary terms beforehand, why not automatically determine the most appropriate vocabulary for a given graph?
Answer 3: The reasons are simple: scalability and interpretability. For a vocabulary term to be useful, it needs to be easily understood by the user. This also relates to why we define our own encoding and optimization algorithm, instead of using off-the-shelf general purpose compressors based on Lempel-Ziv (such as gzip) or statistical compressors (such as PPMZ); these provide state-of-the-art compression by exploiting complex statistical properties of the data, making their models very complex to understand. Local-structure based summaries, on the other hand, are much more easily understood.

Frequent-patterns have been proven to be interpretable and powerful building blocks for data summarization [32, 57, 61]. While a powerful technique, spotting frequent subgraphs has the notoriously-expensive subgraph isomorphism problem in the inner loop. This aside, published algorithms on discovering frequent subgraphs (e.g., [62]), are not applicable here, since they expect the nodes to have labels (e.g., carbon atom, oxygen atom etc.) whereas we focus on large unlabeled graphs.

Question 4: Why would you focus on compression, since your goal is pattern discovery and understanding?
Answer 4: Compression is *not* our goal; it is only our means to find good patterns. High compression ratios are exactly a sign that we discovered many redundancies (i.e., patterns) which can be explained in simple terms (i.e., structures), and thus, we understand the input graph better.

Question 5: Why use SLASHBURN for the graph decomposition?
Answer 5: We use SLASHBURN to generate candidate subgraphs, because it is scalable, and designed to handle graphs *without* "cavemen" structure. However, *any* graph decomposition method could be used instead, or even better a combination of decomposition methods could be applied. We conjecture that the more graph decomposition methods provide the candidate structures for VOG, the better the resulting summary will be. Essentially, there is no correct graph partitioning technique, since each one of them works by optimizing a different goal. MDL, which is indispensable component of VOG, will be able to discover the best structures among the set of candidate structures.

Question 6: What if you already know a few subgraph structures of the graph, can VOG take this into account?
Answer 6: Yes. If you already know that certain nodes form a clique, star, etc., it is trivial to adapt VOG to use this as its base model M (as opposed to the empty model). When describing the graph, VOG will then only report those structures that best describe the remainder of the graph.

Question 7: Some edge distributions are more or less likely to generate certain structures, can VOG be extended such that it can take specific edge distributions into account and only report structures that stand out from such a distribution?
Answer 7: Yes! In this paper we aim to assume as little as necessary for the edge distribution, such that VOG is both parameter-free and non-parametric at its core. However, as long as you can calculate the probability of an adjacency matrix, $P(\mathbf{E})$, we can trivially define $L(\mathbf{E}) = -\log P(\mathbf{E})$. Hence, for instance, if your distribution would have a higher clustering coefficient (that is, dense areas are more likely), the cost for having a dense area in **E** will be relatively low, and hence VOG will only report structures that stand out from this (assumed) background distribution. Recent work by Araujo et al. [6] explores discovering communities that exhibit a different hyperbolic —power-law degree distributed— connectivity than the background distribution. It will be interesting to extend VOG with hyperbolic distributions for both subgraphs, as well as for encoding the error matrix.

Question 8: What if your vocabulary terms do not explicitly show in my graph, but other, more complex ones do?
Answer 8: A core property of MDL is that it identifies the model in a model class that best describes your graph regardless of whether the *true* model is in that class. More simply put, we will return the model that gives the most succinct description of your graph in the vocabulary terms at hand: our model class. In this example, we will give a more crude description of the graph structure than would be ideal. This means we have to spend more bits than ideal, which means we are guarded against overfitting. For the theory behind MDL for model selection, see [24].

Question 9: VOG does not explicitly encode the linkage between structures. Can it give high-level insight in how the structures in a summary are connected?
Answer 9: Yes. We allow nodes to participate in multiple structures, and such nodes are exactly those that implicitly 'link' two structures. For example, a node can be part of a clique, as well as the starting-point of a chain, therewith 'linking' the clique and the chain. The linkage structure of the summary can hence be trivially extracted by inspecting whether the node sets of structures in the summary overlap. It may depend on the task whether one prefers to show the high level linkage of the structure, or give the details per structure in the summary.

7 Related Work

Work related to VOG comprises Minimum Description Length based approaches, as well as graph partitioning and visualization.

7.1 MDL and Data Mining Faloutsos and Megalooikoumou [20] argue that as many data mining problems are related to summarization and pattern discovery, they are intrinsically related to Kolmogorov complexity. Kolmogorov complexity [39] identifies the shortest lossless algorithmic description of a dataset, and provides sound theoretical foundations for both identifying the optimal model for a dataset, and defining what structure is. While not computable, it can be practically implemented by the Minimum Description Length principle [24,51]—lossless compression. Examples of applications in data mining include clustering [14], classification [36], discovering communities in matrices [10], model order selection in matrix factorization [43,44], outlier detection [3,55], pattern set mining [57,61], finding sources of infection in large graphs [49], and for making sense of selected nodes in graphs [4]—just to name a few.

We are, to the best of our knowledge, the first to employ MDL with the goal of summarizing a graph with a vocabulary beyond simple rectangles, as well as with allowing overlap between structures.

7.2 Graph Compression and Summarization. Boldi [9] studied the compression of web graphs using the lexicographic localities; Chierichetti et al. [13] extended it to the social networks; Apostolico et al. [5] used BFS for compression. Maserrat et al. [42] used multi-position linearizations for neighborhood queries. Feng et al. [22] encode edges per triangle using a lossy encoding. SLASHBURN [40] exploits power-law behavior of real world graphs, addressing the 'no good cut' problem [38]. Tian et al. [59] present an attribute-based graph summarization technique with non-overlapping and covering node groups; an automation of this method is given by Zhang et al. [63]. Toivonen et al. [60] use a node structural equivalence based approach for compressing weighted graphs.

An alternative way of "compressing" a graph is by sampling nodes or edges from it [25, 37]. The goal of sampling is to obtain a graph of smaller size, which maintains properties of the initial graph, such as the degree distribution, the size distribution of connected components, the diameter, or latent properties including the community structure [41] (i.e., the graph sample contains nodes from all the existing communities). Although graph sampling may allow better visualization [50], unlike VOG, it cannot detect graph structures and may need additional processing in order to make sense of the sample.

None of the above provide summaries in terms of connectivity structures over non-trivial sub-graphs. Also, we should stress that we view compression not as the goal, but as the *means* to identify summaries that help us to understanding the graph in simple terms, i.e., to discover sets of informative structures that explain the graph well. Furthermore, our VOG is designed for large scale block based matrix vector multiplication where each square block is stored independently from each other for scalable processing in distributed platforms like MAPREDUCE [17]. The above mentioned works are not designed for this purpose: the information of the outgoing edges of a node is tightly inter-connected to the outgoing edges of its predecessor or successor, making them inappropriate for square block based distributed matrix vector multiplication.

7.3 Graph Partitioning. Assuming away the 'no good cut' issue, there are countless graph partitioning algorithms: Koopman and Siebes [31, 32] summarize multi-relational data, or, heavily attributed graphs. Their method assumes the adjacency matrix is already known, as it aims at describing the node attribute-values using tree-shaped patterns.

SUBDUE [15] is a famous frequent-subgraph based summarization scheme. It iteratively replaces the most frequent subgraph in a labeled graph by a meta-node, which allows it to discover small lossy descriptions of labeled graphs. In contract, we consider unlabeled graphs. Moreover, as our encoding is lossless, we can by MDL fairly compare between radically different models and model classes. Navlakha et al. [46] follow a similar approach to Cook and Holder [15], by iteratively grouping nodes that see high inter-connectivity. Their method is hence confined to summarizing a graph in terms of non-overlapping cliques and bipartite cores. In comparison, the work of Miettinen and Vreeken [43, 44] is closer to ours, even though they discuss MDL for Boolean matrix factorization. For directed graphs, such factorizations are in fact summaries in terms of possibly overlapping full cliques. With VOG we go beyond a single-term vocabulary, and, importantly, we can detect and reward explicit structure within subgraphs.

Chakrabarti et al. [10] proposed the cross-association method, which provides a hard clustering of the nodes into groups, effectively looking for near-cliques. Papadimitriou et al. [47] extended this to hierarchies, again of hard clusters. Finally, Rosvall and Bergstrom [53] propose information-theoretic approaches for community detection, again hard-clustering the nodes of the graph. With VOG we allow nodes to participate in multiple structures, and can summarize graphs in terms of *how* subgraphs are connected, beyond identifying that they are densely connected.

7.4 Graph Visualization. Apolo [12] is a graph tool used for attention routing. The user picks a few seeds nodes and Apolo interactively expands their vicinities enabling this way sense-making. An anomaly detection system for large graphs, OPAvion [1], mines graph features using Pegasus [28]), spots anomalies by employing OddBall [2], and lastly interactively visualizes the anomalous nodes via Apolo. In [54], Shneiderman proposes simply scaled density plots to visualize scatter plots, [8] presents random and density sampling techniques for datasets with several thousands of points, while NetRay [27] focuses on informative visualizations of the spy, distribution and correlation plots of web-scale graphs. Dunne and Shneiderman [18] introduce the idea of motif simplification to enhance network visualization. Some of these motifs are part of our vocabulary, but VOG also allows for near-structures, which are common in real-world graphs.

What sets VOG apart: Unlike VOG, none of the above methods meet all the following specifications: (a) gives a soft clustering, (b) is scalable, (c) has a large vocabulary of graph primitives (beyond cliques/cavemen-graphs) and (d) is parameter-free. Moreover, the graph visualization techniques focus on anomalous nodes or how to visualize the patterns of the whole graph. In contrast, VOG attempts to find not specific nodes, but informative structures that summarize well the graph, and therefore provides a small set of nodes and edges that are worth (by the MDL principle) visualizing. The preliminary version of VOG is described in [34].

8 Conclusion

We studied the problem of succinctly describing a large graph in terms of connectivity structures. Our contributions are:

- **Problem Formulation:** We proposed an information theoretic graph summarization technique that uses a carefully chosen vocabulary of graph primitives (Sec. 3).
- **Effective and Scalable Algorithm:** We gave VoG, an effective method which is near-linear on the number of edges of the input graph (Sec. 5.3).
- **Experiments on Real Graphs:** We discussed interesting findings like exchanges between Wikipedia vandals and responsible editors on large graphs, and also analyzed VoG quantitatively as well as qualitatively (Sec. 1, 5).

Future work includes extending the VoG vocabulary to more complex graph structures that we know appear in real graphs, such as core-peripheries, (bipartite core whose one set also forms a clique), and so-called "jellyfish" (cliques of stars), as well as implementing VoG in the distributed computing framework like Map-Reduce.

Acknowledgments

The authors would like to thank Niki Kittur and Jeffrey Rzeszotarski for sharing the Wikipedia datasets. Jilles Vreeken is supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government. U Kang is partly supported by the IT R&D program of MSIP/IITP of Korea. [10044970, Development of Core Technology for Human-like Self-taught Learning based on Symbolic Approach]. Funding was provided by the U.S. ARO and DARPA under Contract Number W911NF-11-C-0088, by DTRA under contract No. HDTRA1-10-1-0120, by the National Science Foundation under Grant No. IIS-1217559, and by ARL under Cooperative Agreement Number W911NF-09-2-0053. This work is also partially supported by an IBM Faculty Award, a Google Focused Research Award, and a Yahoo Research Alliance Gift. The views and conclusions are those of the authors and should not be interpreted as representing the official policies, of the U.S. Government, or other funding parties, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] L. Akoglu, D. H. Chau, U. Kang, D. Koutra, and C. Faloutsos. Opavion: mining and visualization in large graphs. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Scottsdale, AZ*, 2012.

[2] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hyderabad, India*, 2010.

[3] L. Akoglu, H. Tong, J. Vreeken, and C. Faloutsos. COMPREX: Compression based anomaly detection. In *Proceedings of the 21st ACM Conference on Information and Knowledge Management (CIKM), Maui, HI*. ACM, 2012.

[4] L. Akoglu, J. Vreeken, H. Tong, N. Tatti, and C. Faloutsos. Mining connection pathways for marked nodes in large graphs. In *Proceedings of the SIAM International Conference on Data Mining (SDM), Austin, TX*. SIAM, 2013.

[5] A. Apostolico and G. Drovandi. Graph compression by bfs. *Algorithms*, 2(3):1031–1044, 2009.

[6] M. Araujo, S. Günnemann, G. Mateos, and C. Faloutsos. Beyond blocks: Hyperbolic community detection. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Nancy, France*, pages 50–65. Springer.

[7] As-oregon dataset. http://topology.eecs.umich.edu/data.html.

[8] E. Bertini and G. Santucci. By chance is not enough: Preserving relative density through non uniform sampling. In *Proceedings of the Information Visualisation*, 2004.

[9] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *International World Wide Web Conference*, 2004.

[10] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, pages 79–88, 2004.

[11] D. Chakrabarti, Y. Zhan, D. Blandford, C. Faloutsos, and G. Blelloch. Netmine: New mining tools for large graphs. In *SDM Workshop on Link Analysis, Counter-terrorism and Privacy*, 2004.

[12] D. H. Chau, A. Kittur, J. I. Hong, and C. Faloutsos. Apolo: interactive large graph sensemaking by combining machine learning and visualization. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, 2011.

[13] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 219–228, 2009.

[14] R. Cilibrasi and P. Vitányi. Clustering by compression. *IEEE Transactions on Information Technology*, 51(4):1523–1545, 2005.

[15] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[16] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.

[17] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI), San Francisco, CA*, Dec. 2004.

[18] C. Dunne and B. Shneiderman. Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 3247–3256. ACM, 2013.

[19] Enron dataset. http://www.cs.cmu.edu/ enron.

[20] C. Faloutsos and V. Megalooikonomou. On data mining, compression and Kolmogorov complexity. *Data Mining and Knowledge Discovery*, 15(1):3–20, 2007.

[21] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM), Barcelona, Spain*, pages 251–262, 1999.

[22] J. Feng, X. He, N. Hubig, C. Böhm, and C. Plant. Compression-based graph mining exploiting structure primitives. In *Proceedings of the 13th IEEE International Conference on Data Mining (ICDM), Dallas, TX*, pages 181–190. IEEE, 2013.

[23] Flickr. http://www.flickr.com.

[24] P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[25] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani. Metropolis algorithms for representative subgraph sampling. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 283–292, Washington, DC, USA, 2008. IEEE Computer Society.

[26] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[27] U. Kang, J.-Y. Lee, D. Koutra, and C. Faloutsos. Net-ray: Visualizing and mining billion-scale graphs. In *Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Tainan, Taiwan*, 2014.

[28] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM), Miami, FL*, 2009.

[29] G. Karypis and V. Kumar. Multilevel-way hypergraph partitioning. In *Proceedings of the 36th Conference on Design Automation(DAC),New Orleans, LA*, pages 343–348, 1999.

[30] J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The web as a graph: Measurements, models and methods. In *Proceedings of the 5th Annual International Conference (COCOON),Tokyo, Japan*, 1999.

[31] A. Koopman and A. Siebes. Discovering relational items sets efficiently. In *Proceedings of the 8th SIAM International Conference on Data Mining (SDM), Atlanta, GA*, pages 108–119, 2008.

[32] A. Koopman and A. Siebes. Characteristic relational patterns. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 437–446, 2009.

[33] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. VoG: Summarizing graphs using rich vocabularies. In *Proceedings of the SIAM International Conference on Data Mining (SDM), Philadelphia, PA*. SIAM.

[34] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. VoG: Summarizing and Understanding Large Graphs. In *Proceedings of the SIAM International Conference on Data Mining (SDM), Philadelphia, PA*, 2014.

[35] D. Koutra, T.-Y. Ke, U. Kang, D. H. Chau, H.-K. K. Pao, and C. Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Athens, Greece*, pages 245–260, 2011.

[36] M. van Leeuwen, J. Vreeken, and A. Siebes. Compression picks the item sets that matter. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany*, pages 585–592, 2006.

[37] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, New York, NY, USA, 2006. ACM.

[38] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web (WWW), Beijing, China*, pages 695–704, 2008.

[39] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

[40] Y. Lim, U. Kang, and C. Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1–1, 2014.

[41] A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *Proceedings of the 19th International Conference on World Wide Web (WWW), Raleigh, NC*, pages 701–710, New York, NY, USA, 2010. ACM.

[42] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, 2010.

[43] P. Miettinen and J. Vreeken. MDL4BMF: Minimum description length for Boolean matrix factorization. *ACM Transactions on Knowledge Discovery from Data*, 8(4):1–30.

[44] P. Miettinen and J. Vreeken. Model order selection for Boolean matrix factorization. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 51–59. ACM, 2011.

[45] P. Miettinen and J. Vreeken. MDL4BMF: Minimum description length for Boolean matrix factorization. Technical Report MPI-I-2012-5-001, Max Planck Institute for Informatics, 2012.

[46] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 419–432, 2008.

[47] S. Papadimitriou, J. Sun, C. Faloutsos, and P. S. Yu. Hierarchical, parameter-free community discovery. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Antwerp, Belgium*, pages 170–187, 2008.

[48] B. A. Prakash, M. Seshadri, A. Sridharan, S. Machiraju, and C. Faloutsos. Eigenspokes: Surprising patterns and community structure in large graphs. *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hyderabad, India*, 2010.

[49] B. A. Prakash, J. Vreeken, and C. Faloutsos. Spotting culprits in epidemics: How many and which ones? In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM), Brussels, Belgium*. IEEE, 2012.

[50] D. Rafiei and S. Curial. Effectively visualizing large networks through sampling. In *16th IEEE Visualization Conference (VIS 2005), 23-28 October 2005, Minneapolis, MN, USA*, page 48, 2005.

[51] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

[52] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[53] M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences*, 104(18):7327–7331, 2007.

[54] B. Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, 2008.

[55] K. Smets and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 804–815. SIAM, 2011.

[56] Snap. http://snap.stanford.edu/data/index.html.

[57] N. Tatti and J. Vreeken. The long and the short of it: Summarizing event sequences with serial episodes. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Beijing, China*. ACM, 2012.

[58] L. Tauro, C. Palmer, G. Siganos, and M. Faloutsos. A simple conceptual model for the internet topology. *Proceeding of the Global Communications Conference Exhibition and Industry Forum, San Antonio, Texas*, 2001.

[59] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 567–580, 2008.

[60] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 965–973, 2011.

[61] J. Vreeken, M. van Leeuwen, and A. Siebes. KRIMP: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[62] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *IEEE International Conference on Data Mining*, Los Alamitos, CA, 2002. IEEE Computer Society Press.

[63] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach, CA*, pages 880–891, 2010.