Summarizing Labeled Multi-Graphs

Dimitris Berberidis[1] Pierre J. Liang[2] Leman Akoglu[1] ✉

[1] Carnegie Mellon University, Heinz College of Information Systems and Public Policy
{dbermper, lakoglu}@andrew.cmu.edu
[2] Carnegie Mellon University, Tepper School of Business
liangj@tepper.cmu.edu

Abstract. Real-world graphs can be difficult to interpret and visualize beyond a certain size. To address this issue, graph summarization aims to simplify and shrink a graph, while maintaining its high-level structure and characteristics. Most summarization methods are designed for homogeneous, undirected, simple graphs; however, many real-world graphs are *ornate*; with characteristics including node labels, directed edges, edge multiplicities, and self-loops. In this paper we propose TG-SUM, a *versatile* yet rigorous graph summarization model that (to the best of our knowledge, for the first time) can handle graphs with *all* the aforementioned characteristics (and *any* combination thereof). Moreover, our proposed model captures basic sub-structures that are prevalent in real-world graphs, such as cliques, stars, etc. Experiments demonstrate that TG-SUM facilitates the visualization of real-world complex graphs, revealing interpretable structures and high-level relationships. Furthermore, TG-SUM achieves better trade-off between compression rate and running time, relative to existing methods (only) on comparable settings.

Keywords: graph summarization · super graph · labeled multi-graph

1 Introduction

Given a directed labeled multi-graph G, how can we construct a small summary graph g that reflects the high-level structures and relationships in G? How can we find a succinct g that is yet an accurate representation, which requires a small amount of corrections to recover the original G? With the advent of technology, not only the size but also the complexity of real-world graphs have grown immensely. Today graph data often contains node labels, multi-edges, etc. Graph summarization aims to find high-level structural patterns and most salient information in large complex graphs to enable efficient storage, processing, visualization and interpretation.

A large body of existing graph summarization techniques is for *plain* graphs with homogeneous unlabeled nodes [14,24,16,15,23,26,20]. However there exist numerous real-world graphs with multiple node labels; including transaction networks containing nodes (i.e. accounts) of various types (cash, revenue, expense,

etc.) or heterogeneous graphs such as publication records among entities of various types (paper, author, venue, etc.). We refer to both kinds as node-labeled, or simply *labeled graphs*. Moreover, a vast majority of prior work are for summarizing *simple* [14,24,16,15,20,28,19,25,9] *undirected* [14,24,16,15,23,26,20,28,25] graphs, whereas the edges in real-world graphs may repeat (e.g., multiple transactions between two accounts, multiple exchanges between two email addresses, etc.) which are called *multi-graphs*. As is the case for transaction and email graphs, among others, the edges can also be *directed*.

In this work we propose (to the best of our knowledge; see, Table **??**) the *first* method called TG-SUM, for *multi*-Type (i.e. node-labeled) *multi*-Graph SUMmarization, with directed edges and possible self-loops. (See Sec. 2, and a recent comprehensive survey [18].) Besides, TG-SUM is *versatile* in that it can also handle graphs with any combination of those properties (i.e., (un)directed, plain/labeled, simple/multi- or weighted graphs).

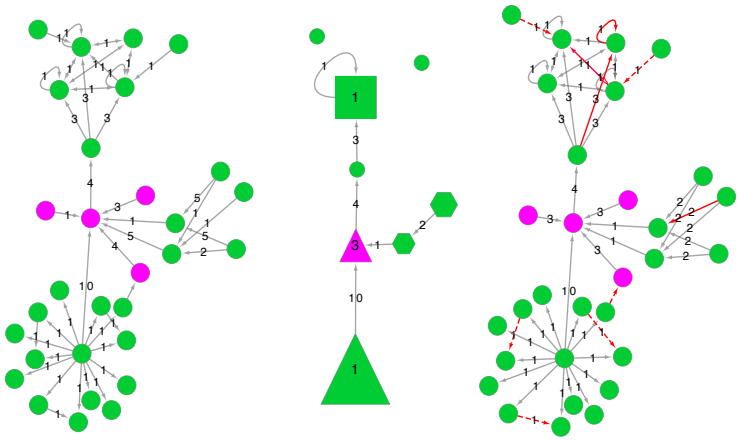

Fig. 1: (best in color) Ex. input graph (left), its summary/super-graph (middle), and the decompressed graph (right) w/ edge corrections in red, where dashed - - (solid —) are edges that need to be added (removed) for lossless reconstruction. See text for description of the scalars, node color, size, and shape.

Our goal is to output a small yet representative summary that facilitates the visualization, by which, improves the understanding of the overall structure of an input graph. To this end, we model a *summary graph (or super-graph)* as a collection of labeled super-nodes and weighted super-edges. As illustrated in Fig. 1, we merge structurally similar nodes of the same type/label (depicted by color) into super-nodes (size reflecting the number of constituent nodes). Super-nodes capture prevalent structural constructs found in real-world graphs, such as stars and cliques [15] (depicted by shape). A super-node is also marked with a scalar (i.e., weight), representative of the edge multiplicities among its nodes.

A super-edge is placed between two super-nodes whose constituent nodes are sufficiently well-connected, and is also marked with a scalar (i.e., weight) that best represents the edge multiplicities inbetween.

We aim to construct a small summary graph, which accurately reflects the input graph. Here, succinctness and accuracy are in trade-off; the coarser the summary graph, the more information about the original graph is lost. We design a novel two-part *lossless* encoding scheme, describing (*i*) the summary graph and (*ii*) the corrections required to reconstruct the input graph losslessly. Treating the total number of encoding bits as a cost function, we design algorithms to find a summary with a small total cost. In summary, our main contributions are:

- The first method for Summarizing LMDS-Graphs.
- A Novel Super-graph Model, in Section 3.1
- A Novel Two-part Lossless Encoding Scheme, in Section 3.2.
- Efficient Search Algorithms, in Section 4.
- Extensive experiments on real-world graphs, in Section 5

Reproducibility: Source code for TG-SUM and all public-domain datasets are shared at `https://bit.ly/3d4vogt`.

2 Related Work

Graph *summarization* and graph *compression* techniques, while related, exhibit a key distinction. The former typically aims to simplify an input graph into a coarser one, while reflecting its prominent structure. On the other hand, the latter aims at reducing the storage requirements of a graph, often enabling speedy querying, while maintaining a certain level of query accuracy [2,6,7,13,17]. (See [5] for a recent survey.) In this work we focus on graph summarization, with a goal to extract a simplified overview of key structural patterns within an input graph. Most graph summarization techniques are designed for unlabeled, undirected, and simple graphs without edge multiplicities, weights, or self-loops [24,14,16,15]. Closely-related are graph-pooling methods used within graph neural networks to gradually reduce the dimension of the layers; see. e.g. [27]. Riondato *et al.*[23] and Toivonen *et al.*[26] are some of the few summarization methods that can accommodate weighted edges, but not labeled nodes or directed edges. Among the methods that can handle graphs with multiple node labels, CoSum [28], Liu *et al.*[19], and SNAP [25] build a coarser graph by only merging the nodes of the same label into super-nodes. Differently, Subdue [9] replaces frequent subgraphs that potentially contain different labels with a super-node, which makes the interpretation of the summary graph harder. Closest to our work is the approach by Navlakha *et al.*[20], which iteratively merges nodes into super-nodes as long as the description length of the input graph decreases. Thanks to its simple model and algorithm, it can be modified to handle labeled graphs, specifically by restricting the node merges to same-label nodes. However, its model is unable, nor is it trivial to modify, to accommodate edge weights/multiplicities. All in all, there is *no existing work* that can summarize labeled multi-graphs

– with labeled nodes, directed and multi-edges and self-loops. (See [18] for an extended survey and Table 1 therein.) While our TG-SUM is the first of its kind, it is *versatile* in that it can also accommodate graphs with any combination of those properties. Besides input graph properties, prior work can also be classified w.r.t. the properties of the summary. Here, we focus on summaries where the output is itself a (coarser) graph, called the summary or super graph. VoG [15] identifies key sub-structures (stars, (near)cliques, etc.) however does not provide any super-edges, i.e., its summary graph is disconnected. Second, the summary may be lossy; including only the coarse summary graph [23,26,28,19,25,9], or lossless; consisting of both the summary and the corrections necessary to fully reconstruct the input graph [24,14,16,15,20]. Finally, a desired characteristic of a summary is multi-granularity; where the coarseness or resolution of the summary graph can be adjusted on demand [16,23,26,19,25,9,20], via appropriately altering some of the model parameters. Notably, TG-SUM exhibits all of these three properties: super graph output, lossless and multi-resolution summary.

3 Graph Summary Design and Encoding

3.1 Summary and Decompression

Given a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$ with edge multiplicities $m(e) \in \mathbb{N}, \forall e \in \mathcal{E}$, node labels/types $\ell(u) \in \mathcal{T}, \forall u \in \mathcal{V}$, and self-loops, we define the *summary* and *decompressed* graphs as follows.

3.1.1 Summary graph (or super-graph) model Let $\mathcal{G}_\mathrm{s} = \{\mathcal{V}_\mathrm{s}, \mathcal{E}_\mathrm{s}\}$ be the sets of super-nodes and directed super-edges that define the summary graph topology. Each super-node $v \in \mathcal{V}_\mathrm{s}$ is annotated by four components: (i) its label $\ell(v)$ (depicted by color), (ii) the number $|\mathcal{S}_v|$, $\mathcal{S}_v \subset \mathcal{V}$, of nodes it contains (depicted by size), (iii) the *glyph* $\mu(v) \in \mathcal{M}$ it represents (depicted by shape), and (iv) the *representative* multiplicity $m(v)$ of the edges it summarizes (depicted as a scalar inside the glyph). For each super-edge $e \in \mathcal{E}_s$, we let $m(e)$ be the *representative* multiplicity of the edges it captures, depicted as a scalar on the super-edge. (We describe how to find the "representative" multiplicity of a set of edges in Sec. 4.2.)

Fig. 1 (left) and (middle) respectively depict an example input graph and its corresponding summary graph. Apart from unmerged simple nodes that are depicted as plain circles, the set $\mathcal{M}$ of possible glyphs that TG-SUM supports contains: 1) `Clique` (square), 2) `In-star` (triangle), 3) `Out-star` (triangle), and 4) `Disconnected` set (hexagon). Such structures are commonly found in real-word graphs [15]. For instance, a clique can represent a tightly-knit group of friends in a social network, while an out-star can capture spam-like activity in an email or call network. Moreover, using glyphs has been shown to yield easily interpretable visualizations [10].

3.1.2 Decompression The summary graph $\mathcal{G}_s$ decompresses *uniquely* and *unambiguously* into $\mathcal{G}' = \text{dec}(\mathcal{G}_s) = \{\mathcal{V}, \mathcal{E}'\}$ according to simple and intuitive rules (e.g., Fig. 1 (right)). First, every super-node expands to the set of nodes it contains, all of which also inherit the super-node's label. The nodes are then connected according to the super-node's glyph: for out(in)-stars a node defined as the hub points to (is pointed by) all other nodes, for cliques all possible directed edges are added between the nodes, and for disconnected sets no edges are added. Moreover, a super-node self-loop expands to self-loops on every node it contains. On the other hand, super-edges expand to sets of edges that have the same direction.

Apart from enabling a clear interpretation of a given summary, the decompression rules help quantify how well the summary represents the original graph. For example, the pink triangle with representative multiplicity 3 in Fig. 1 (middle) expands to an in-star with all edges having multiplicity 3 as shown in Fig. 1 (right). While the topology is perfectly captured (pink nodes form a perfect in-star), the expanded multiplicities are not always equal to the original ones. On the other hand, expanding the green triangle perfectly captures the edge multiplicities (all are 1), but only approximates the topology, as the original green subgraph also contains some edges between the spokes of the hub node.

3.2 Model Encoding

Following the two-part Minimum Description Length paradigm [12], we aim to identify a summary graph $\mathcal{G}_s$ that minimizes the total description cost of the full graph, that is,

$$\mathcal{G}_s^* := \arg\min_{\mathcal{G}_s} \quad L(\mathcal{G}_s) \;+\; L(\mathcal{G}|\mathcal{G}'), \quad \text{s.t.} \quad \mathcal{G}' = \text{dec}(\mathcal{G}_s) \tag{1}$$

where $L(\mathcal{G}_s)$ measures the number of bits required to encode the summary graph, and $L(\mathcal{G}|\mathcal{G}')$ the bits needed to encode the corrections (or extra-information) for reconstructing the original graph $\mathcal{G}$ from the (uniquely and unambiguously) decompressed $\mathcal{G}'$. These costs can be quantified as follows.

3.2.1 Encoding the summary graph We first encode the size of the summary graph $L_{\mathbb{N}}(|\mathcal{V}_s|)$, and the number of labels $L_{\mathbb{N}}(|\mathcal{T}|)$.[3] For each super-node, $\log_2 |\mathcal{T}|$ bits are used to record its label, $\log_2 |\mathcal{M}|$ for its glyph, $L_{\mathbb{N}}(|\mathcal{S}_v|)$ for its size, $\mathcal{L}_{\mathbb{N}}(m(v))$ for the within-glyph representative multiplicity, $\log_2(|\mathcal{V}_s|)$ for the number of super-nodes in $\mathcal{G}_s$ that it points to, and $\log_2 \binom{|\mathcal{V}_s|}{|\mathcal{N}(v)|}$ to identify the specific set of super-nodes it points to, where $\mathcal{N}(v)$ denotes the set of direct (out)neighbors. For each super-edge, $L_{\mathbb{N}}(m(e))$ bits are used for the representative multiplicity. In total, the number of bits required to encode a summary

[3] $L_{\mathbb{N}}(k) = 2\log k + 1$ bits are required to encode an arbitrarily large natural number k, using the variable-length prefix-free encoding; see, Ex. 2.4 in [12].

graph is given as

$$L(\mathcal{G}_\mathrm{s}) = L_\mathbb{N}(|\mathcal{V}_\mathrm{s}|) + L_\mathbb{N}(|\mathcal{T}|) + \sum_{v \in \mathcal{V}_\mathrm{s}} L_\mathrm{SNODE}(v) \;, \tag{2}$$

where

$$L_\mathrm{SNODE}(v) = \log_2 |\mathcal{T}| + \log_2 |\mathcal{M}| + L_\mathbb{N}(|\mathcal{S}_v|) + \mathcal{L}_\mathbb{N}(m(v)) + \log_2(|\mathcal{V}_\mathrm{s}|) + \log_2 \binom{|\mathcal{V}_\mathrm{s}|}{|\mathcal{N}(v)|} + \sum_{z \in \mathcal{N}(v)} L_\mathbb{N}(m(v,z)) \tag{3}$$

3.2.2 Encoding the corrections For the overall cost for corrections, we first compute the number of bits used to correct the *topology* of the expanded (i.e., decompressed) graph, followed by the number of bits needed to represent the true *multiplicities*. Regarding the topology, we first map the expanded nodes back to the original node-set $\mathcal{V}$. This costs $L_\mathrm{MAP}(v) = \log_2 \binom{|\mathcal{V}|}{|\mathcal{S}_v|} + \mathbb{1}_{\{\mu(v)=\mathrm{STAR}\}} \log_2 |\mathcal{S}_v|$ bits per super-node v (the latter term identifying the hub of a star). Subsequently, we have two types of edge corrections: Either adding edges that exist in the original graph but not in the expanded graph (i.e., positive corrections) or removing edges from the expanded graph because they do not exist in the original graph (i.e., negative corrections).

These costs are compactly encoded for every expanded super-edge and every expanded super-node (glyph), using the binomial encoding $L(\mathcal{E}_\mathrm{COR}) = L_\mathbb{N}(|\mathcal{E}_\mathrm{COR}|) + \log_2 \binom{|\mathcal{E}^\mathrm{max}_\mathrm{COR}|}{|\mathcal{E}_\mathrm{COR}|}$, where $\mathcal{E}_\mathrm{COR}$ denotes the possible set of corrections (positive or negative), and $\mathcal{E}^\mathrm{max}_\mathrm{COR}$ the largest set that $\mathcal{E}_\mathrm{COR}$ can possibly be. For example, for positive edge corrections in a disconnected set, we have $\mathcal{E}^\mathrm{max}_\mathrm{COR} = \mathcal{S}_v \times \mathcal{S}_v$, and similarly for negative edge corrections in a clique. For super-edges, corrections are computed according to the decompression rules (see Sec. 3.1). For the (few) edges in the original graph between super-nodes v and z that are not represented by a super-edge, the corrections are always positive, and $\mathcal{E}^\mathrm{max}_\mathrm{COR} = \mathcal{S}_v \times \mathcal{S}_z$.

The binomial encoding arises from using the uniform code over all the lexicographically ordered *subsets* of possible corrections. An alternative to this, as suggested in [16] and [23], would be to encode each correction *individually* using an optimal prefix code. Then, interpreting $p = |\mathcal{E}_\mathrm{COR}|/|\mathcal{E}^\mathrm{max}_\mathrm{COR}|$ as the "probability" of each correction, we would need $L_\mathrm{ENTR} = H(\mathrm{Ber}[p]) \cdot |\mathcal{E}^\mathrm{max}_\mathrm{COR}|$ bits, where $H(\cdot)$ is the Shannon entropy, and $\mathrm{Ber}[p]$ is a Bernoulli with parameter p. Denoting $|\mathcal{E}_\mathrm{COR}| = n'$ and $|\mathcal{E}^\mathrm{max}_\mathrm{COR}| = n$, we can show that our binomial encoding is more efficient.

Theorem 1. *It holds that* $L_\mathrm{ENTR} \geq \log_2 \binom{n}{n'}$, *for all* $n > n' > 0$.

Proof. See Appendix.

Theorem 1 establishes that the binomial encoding always gives a more compact measure of information required for corrections. Having corrected the edge topology, we compute the cost of correcting the edge multiplicities. Since any

edge e not included in a glyph or super-edge does not have a representative multiplicity, its multiplicity correction is encoded by $L_{\mathbb{N}}(m(e))$, encoding its true value. The reason for using $L_{\mathbb{N}}(\cdot)$ to encode multiplicities is the fact that, for most real graphs, multiplicities follow a power-law distribution. Since the vast majority has small values, $L_{\mathbb{N}}(\cdot)$ will generally be a more "compact" encoding compared to a uniform code based on the maximum multiplicities. For expanded super-nodes and super-edges with representative multiplicity m, we obtain the cost of correcting the multiplicities as

$$L_{\text{DIFF}}(\mathcal{E}_{\text{sup}}, m) = \sum_{e \in \mathcal{E}_{\text{sup}}} \ell_{\text{diff}}(m(e), m), \tag{4}$$

where $\mathcal{E}_{\text{sup}}$ in this context is the set of all edges contained in said super-node or super-edge, and

$$\ell_{\text{diff}}(m', m) = \begin{cases} 1\,, & m = m' \\ 2\log_2(|m - m'|) + 3\,, & m \neq m' \end{cases}, \tag{5}$$

bits are needed to encode the *difference* between a true multiplicity m' and its representative m. Note that, since $L_{\mathbb{N}}(\cdot)$ only holds for natural numbers (see footnote[3]), one extra bit is required to indicate whether the difference is 0, and one more for the sign of the difference.

4 Graph Summary Search

The discrete optimization problem in (1) has a very large set of feasible solutions, and needs to be approximated efficiently. Towards this goal, we follow a two-step process, where we first generate a list of (possibly overlapping) groups of nodes, which we term *candidate* node-sets (see Sec. 4.1), and then decide which ones to merge into super-nodes. These candidates have varying size and quality (i.e., structural-similarity). Larger candidates with low quality compress the graph more (reduced $L(\mathcal{G}_{\text{s}})$), but also typically require more corrections (increased $L(\mathcal{G}|\mathcal{G}')$). Clearly, the best candidates have both high quality and large size. For this reason, we first sort the candidate sets in descending order with respect to the product of their size and quality. We then process the sorted list from top to bottom, and merge the candidate sets into super-nodes, updating the summary graph accordingly (see Sec. 4.2). To ensure the quality of summarization, we only monitor the overall total cost, and only commit to a given candidate if $\Delta_{\text{cost}} = \text{Cost\_After} - \text{Cost\_Before} < 0$. This offers two benefits: (1) We avoid the cumbersome process of merging nodes in pairs (i.e. two at a time) and instead merge *in groups*, and (2) We achieve ability to summarize at *multiple resolutions*. The overview is given in Algorithm 1.

4.1 Candidate Set Generation

4.1.1 Measuring candidate quality To quantify a candidate set's quality, we first need to define a proper metric of structural node similarity. For undirected graphs, the Jaccard similarity between two nodes v and v' is given as

Algorithm 1: TG-SUM: Summarizing Labeled Multi-Graphs

Input: directed labeled multi-graph $\mathcal{G}$
Construct candidate node-sets (Sec. 4.1);
Sort candidates w.r.t. (size $\times$ quality);
for *every candidate set in list* **do**
  Merge unmarked nodes in set and decide glyph (Sec. 4.2.1);
  Decide super-edges (Sec. 4.2.2) ;
  Compute representative multiplicities (Sec. 4.2.3);
  Mark candidate node-set as merged;
  if $\Delta_{\text{cost}} < 0$ **then**
    Commit to merged super-node and its super-edges;
Return summary graph $\mathcal{G}_\text{s}$;

$J^U(v, v') = \frac{|\mathcal{N}^U(v) \cap \mathcal{N}^U(v')|}{|\mathcal{N}^U(v) \cup \mathcal{N}^U(v')|}$, and simply measures the proportion of common neighbors that they share. Naïvely using $J^U(\cdot, \cdot)$ on directed graphs is straightforward by ignoring the directions of the edges, however, it may yield misleading results by often over-estimating the true node similarity. To mitigate such inconsistencies, we introduce the following extension of Jaccard that may also accommodate directed graphs, by taking into account the similarity of both *I*ncoming and *O*utgoing edges.

Definition 1. *The* directed Jaccard *similarity between any two pair of nodes v, v' of a directed graph is given as*

$$J^D(v, v') = \frac{|\mathcal{N}^I(v) \cap \mathcal{N}^I(v')| + |\mathcal{N}^O(v) \cap \mathcal{N}^O(v')|}{|\mathcal{N}^I(v) \cup \mathcal{N}^I(v')| + |\mathcal{N}^O(v) \cup \mathcal{N}^O(v')|} \tag{6}$$

First, it can easily be observed that for undirected graphs, $J^D(v, v') = J^U(v, v')$, since $\mathcal{N}^I(v) = \mathcal{N}^O(v) = \mathcal{N}^U(v)$. Note however, that in our example, $J^D(v, v')$ becomes 0 for all cross-pairs between $\{B, C, D\}$ and $\{E, F\}$, effectively creating two separate groups. In general for directed graphs, $J^D(v, v')$ will be more "informed" than $J^U(v, v')$, typically yielding lower similarity scores. We then define

Definition 2. *Any set $\mathcal{C} \subseteq \mathcal{V}$, is* $t-$bounded *if* $\quad J^D(v, v') \geq t \quad \forall (v, v') \in \mathcal{C} \times \mathcal{C}$.

We use the $t-$bounded-ness of a candidate to serve as a pessimistic valuation of its quality. In addition, given that we are interested in a *collection* of candidate sets, we would like the sets to be *non-redundant* defined as follows.

Definition 3. *Let $\mathcal{C}_\text{S}$ be a collection of candidate sets, each one accompanied by a bound t. We call $\mathcal{C}_\text{S}$* non-redundant*, if for any $\mathcal{C} \in \mathcal{C}_\text{S}$ that is $t-$bounded, there exists no $t'-$bounded $\mathcal{C}' \in \mathcal{C}_\text{S}$, such that $t' \geq t$ and $\mathcal{C} \subset \mathcal{C}'$.*

Simply put, non-redundancy ensures that none of the candidate sets is a strict subset of another set of higher or equal quality.

4.1.2 Incremental LSH To group nodes according to their similarity, we first utilize Locality Sensitive Hashing (LSH) [4]. Specifically for every node v, we generate a set of r *minhash* signatures

$$h_j(v) := \min_{z \in \mathcal{N}^D(v)} f_j(z) \qquad \forall j = 1, \ldots, r \tag{7}$$

where f_j's are independent and uniform hash functions (see, e.g., [4] for implementation details), and $\mathcal{N}^D(v) := \mathcal{N}^I(v) \| \mathcal{N}^O(v)$ is the concatenated adjacency list of node v that includes all incoming and outgoing neighbors separately. It can then be shown that $\Pr\{h_j(v) = h_j(v')\} = J^D(v, v')$; that is, two nodes share a minhash signature with probability proportional to their directed Jaccard similarity. Since the r hash functions are independent, it follows that $\Pr\{\mathbf{h}(v) = \mathbf{h}(v')\} = \big(J^D(v, v')\big)^r$, where $\mathbf{h}(v) := [h_1(v), \ldots, h_r(v)]^T$ is the $r-$length minhash signature vector of node v. If the nodes are hashed into buckets according to their r minhash signatures, the equality gives the probability that two nodes hash into the same bucket. By collecting b hash-tables corresponding to b bands of r minhash signatures, the probability that v and v' hash to the same bucket at least once is

$$\Pr\{\mathbf{h}_i(v) = \mathbf{h}_i(v') \; \exists i = 1, \ldots, b\} = 1 - \Big(1 - \big(J^D(v, v')\big)^r\Big)^b \tag{8}$$

Interestingly, for sufficiently large r and b, the RHS expression in Eq. (8) when viewed as a function of $\big(J^D(v, v')\big)$ approximates a step function around the threshold $t = \big(\frac{1}{b}\big)^{\frac{1}{r}} \in (0, 1]$, meaning that with high probability v and v' will belong in a $t-$bounded set. To avoid repeating the entire process for different values of b, we incrementally generate and add more bands of minhash node signatures, that in turn hash nodes into new buckets. The new buckets are then merged with any overlapping existing buckets, gradually coalescing into larger clusters that are *approximately* $t-$bounded, with $t = \big(\frac{1}{b}\big)^{\frac{1}{r}}$ decreasing as b increases. This is exactly how we obtain larger candidate sets, albeit of lower quality, incrementally by the addition of new bands.

4.1.3 Filtered LSH While the incremental LSH described in the previous section efficiently guides the process of forming candidate sets, merged buckets are not guaranteed to be $t-$bounded due to the false alarm probability. For this purpose, we maintain an undirected similarity graph $\mathcal{G}_{\text{sim}}$, where an edge (v, v') is *guaranteed* to appear if and only if $J^D(v, v') \geq t$. Intuitively, $\mathcal{G}_{\text{sim}}$ serves as a data structure where large $t-$bounded candidates appear as maximal cliques. As new LSH buckets appear and clusters are updated, we compute $J^D(v, v')$ for newly coalesced pairs of nodes (v, v'), and add the latter as an edge to $\mathcal{G}_{\text{sim}}$ if $J^D(v, v') \geq t$. If the threshold is not satisfied, the computed value $J^D(v, v')$ is not discarded, but cached into a max-heap since it may satisfy a lower t in one of the subsequent iterations as b is increased.

As mentioned earlier, candidate sets are collected as maximal cliques in $\mathcal{G}_{\text{sim}}$. To ensure that the set of candidates is *non-redundant* (cf. Def.n 3), we maintain

for every node the size of the maximum clique that it has been found to belong in. Every time a new clique is discovered, we update the maximum-sizes for all the nodes it contains using the clique's size. As new edges are added to $\mathcal{G}_{\text{sim}}$, we examine every node for newly emerged cliques, and we rely on the heuristic in [22] to prune the search by avoiding the evaluation of cliques that cannot exceed the size of the previously-found maximum clique.

4.2 Merging Candidates: Glyphs, Super-edges, Multiplicities

Every time a candidate set $\mathcal{C}$ is tested, we deploy subroutines that efficiently update the summary graph, by making decisions regarding (1) the glyph that will be assigned to the merged set of nodes, (2) super-edges that emerge (or disappear) due to the merging, as well as (3) representative multiplicities for the set of edges summarized by the glyph and its super-edges.

4.2.1 Glyph decision rules To preserve super-node label homogeneity, a candidate set that contains nodes of different labels is first split into same-label subsets. Each subset is merged into a separate super-node using the procedure described below. Hereafter, the term candidate set refers to such a label-homogeneous subset. For the glyph decision, we first identify the number of directed edges $E_{\mathcal{C}}$ that are included in the subgraph induced on nodes that corresponds to $\mathcal{C}$ in the candidate set. Consequently, if $E_{\mathcal{C}} \geq |\mathcal{C}|(|\mathcal{C}|-1)/2$, i.e., at least half of all possible directed edges are present, then we decide `Clique` since it most likely is the best glyph in terms of number of edge corrections. For sparsely-edged candidate sets that do not pass the clique threshold, we proceed to test for the presence of stars. If there is a suitable out-/in-star present in $\mathcal{C}$, then its hub will be the highest out/in-degree node in $\mathcal{C}$. We use the followin proxy correction cost for encoding an in-star

$$\text{Cost}_{\text{IN}} = (|\mathcal{C}| - 1 - d^I_{\max}) + (E_{\mathcal{C}} - d^I_{\max}) \,, \tag{9}$$

and similarly Cost_{OUT} for out-star using $d^O_{\max}$. Intuitively, the first term of Eq. (9) is the number of edges that will have to be removed from the full decompressed star, while the second part is the number of edges that cannot be "explained" by the star and will have to be added. We then compare Cost_{IN} and Cost_{OUT} with $E_{\mathcal{C}}$, i.e., the number of edges that will have to be added if we decide that $\mathcal{C}$ is a disconnected set. If only Cost_{IN} (or only Cost_{OUT}) is smaller than $E_{\mathcal{C}}$, then we decide `In-star` (or `Out-star`). If both Cost_{IN} and Cost_{OUT} are smaller than $E_{\mathcal{C}}$, then we choose the smallest of the two. Finally, if neither Cost_{IN} nor Cost_{OUT} are smaller than $E_{\mathcal{C}}$, we decide $\mathcal{C}$ is a `Disconnected` set.

4.2.2 Super-edge decision rule Having decided the glyph of $\mathcal{C}$, we merge any outgoing and incoming edges and/or super-edges into "bundles" of edges and their corresponding multiplicities. We then obtain the topology-based correction costs of merging or not merging each bundle into a super-edge (recall Sec. 3.2).

If the total cost (topology and multiplicities) of representing each bundle of edges with a super-edge is lower than the cost of *not* representing it, then the corresponding super-edge (and its representative multiplicity) is added to the summary.

4.2.3 Finding representative multiplicities For every newly-formed super-node as well as each potential super-edge, we find the representative multiplicity m^* as $m^* := \arg\min_m \quad L_{\text{DIFF}}(\mathcal{E}_{\text{sup}}, m)$ where $L_{\text{DIFF}}(\mathcal{E}_{\text{sup}}, m)$ is defined in Eq. (4), and $\mathcal{E}_{\text{sup}}$ is the set of all edges contained in a given super-node or bundled by a super-edge. Although this 1D-optimization problem is not convex, we find that the dichotomous search algorithm [8] finds the optimal solution in most cases, and runs in $\mathcal{O}(|\mathcal{E}_{\text{sup}}| \log_2 R)$ time, where $R = \max_{e \in \mathcal{E}_{\text{sup}}} m(e) - \min_{e \in \mathcal{E}_{\text{sup}}} m(e)$, i.e., the dynamic range of multiplicities.

5 Experiments

5.1 Setup

We experimented with real-world graphs of a wide variety of sizes and characteristics, including a senator-to-senator network extracted from the 2009-2010 US Congress dataset [3] and the Political Blogs network [1], both with political affiliation labels; the Cora and Citeseer citation networks [11] that are labeled by publication venue; and finally, transaction networks from 3 (anonymous) corporations that we collaborated with. See Table 1 for a summary of network characteristics. There is *no existing method* for LMDS-graph summarization, thus

Table 1: Real-world graphs used in experiments. $*$ depicts naturally directed graphs that are typically treated as undirected. For `SH, HW, KD` #labels is given for EB/FS labeling.

Name	#nodes	#m-edges	#labels	**L**bl.	**D**ir.	**M**ult.	**S**-loop
`senate`	0.1K	2.4K	2	✔			
`polblogs`	1.5K	19K	2	✔	*		
`cora`	2.7K	10.6K	6	✔	*		
`citeseer`	3.3K	9.2K	7	✔	*		
`SH trans`	0.25K	301K	11/27	✔	✔	✔	✔
`HW trans`	0.32K	268K	11/60	✔	✔	✔	✔
`KD trans`	2.3K	648K	10/29	✔	✔	✔	✔

we compare only under simplified settings, w.r.t. running time and compression rate. Moreover, TG-SUM is only comparable to lossless methods. We modify the *Randomized* algorithm of Navlakha *et al.* [20] to accommodate node labels and edge directions, and compare on all graphs, ignoring the edge multiplicities.

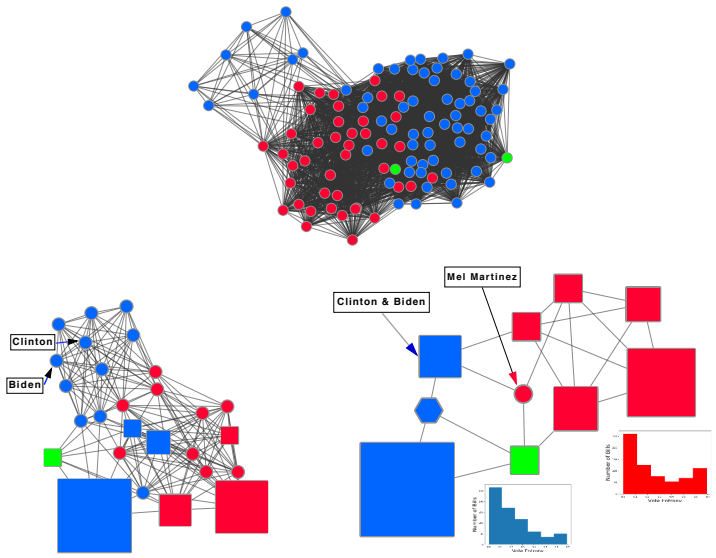

Fig. 2: (left) original US Senate graph, (middle) high resolution ($b = 2$) summary, (right) low resolution ($b = 5$) summary.

5.2 Qualitative Evaluation: TG-SUM at Work

The US Senate dataset contains the (positive or negative) votes of 108 senators for 696 congressional bills. The senators are labeled as Republicans (red), Democrats (blue), or independent (green). We construct an undirected graph where two senators are connected by an edge if the cosine similarity of their votes is larger than 0.3. The graph is plotted in Fig. 2, along with two summaries at different resolutions, leading to the following observations. Interestingly, while most democratic senators eventually form a clique, there is a smaller group of East coast senators, including prominent Democratic figures such as Joe Biden, Hillary Clinton, and Ted Kennedy that do not merge with the main body and form their own separate clique. Furthermore, this clique of Democrats is directly linked to certain Republicans, such as the Florida-based Mel Martinez, who has most recently opposed Trump openly and explicitly expressed his preference for Joe Biden[4]. The second observation is that Republican senators overall exhibit a more fragmented voting behavior, splitting into multiple cliques of comparable size. This is corroborated by computing the entropies of the votes for all the bills, for Democrats and Republicans separately. Intuitively, bills with high entropy indicate a low degree of agreement on the subject. By plotting the histograms of the voting entropies (see Fig. 2 (right)) for the two groups, it becomes apparent that Republican votes exhibit higher entropy (median = 0.21) than Democrats (median = 0.16).

[4] `https://bit.ly/3qwc9zu`

5.3 Quantitative Evaluation: Evaluating Financial Accounts Labeling

In this section, we show how we employed TG-SUM to quantitatively address a domain-specific problem, specifically, evaluating a pre-existing *labeling*, i.e., the set of types pre-assigned to the nodes in an accounting network that connects business accounts via credit/debit transaction relations.A business entity's Chart of Accounts (COA) lists, and also pre-assigns a label to, each distinct account used in its ledgers. Such labeling helps companies prepare their aggregate financial statements (FS). For example, the FS caption "Cash and Cash Equivalents" is used to describe the total sum of all liquid assets tracked in a number of accounts; e.g., currencies, checking accounts, etc. In the US, FS captions are not uniform across corporations. In fact, the data we have from 3 different companies (anonymized as SH, HW, and KD in Table 1) each contains different FS captions.

How suitable is a given FS labeling? Can a different labeling be shown to be quantitatively better than another?

To this end, our collaborator (an accounting expert) designed a new labeling (referred as EB for economic bookkeeping), relabeling the accounts based on their primary economic nature. Specifically, EB organizes them into operating versus financing and long- versus short-term accounts. Expert knowledge suggests that EB improves over FS captions by categorizing the accounts such that accounts of the same label should "behave" similarly in the system. This behavior can be discerned from the real-world usage data, in particular the transactions graph, where accounts are connected through credit/debit relations. Under a more suitable labeling, the accounts with the same label should have more structural similarity and yield better compression. To compare EB vs. FS, we employ

Table 2: Evaluating account labelings in financial networks

Dataset	Labeling	Shuffled	Actual	norm. gain (%)
SH	EB	0.28	0.32	**5.6** %
	FS	0.25	0.27	2.7 %
HW	EB	0.36	0.47	**17.0** %
	FS	0.16	0.27	13.0 %
KD	EB	0.33	0.42	**13.7** %
	FS	0.31	0.39	12.0 %

TG-SUM on each graph using one or the other labeling separately, and record the compression rate. Next, we shuffle the labels (within each setting) randomly, and employ TG-SUM again. "Shuffled" and "Actual" compression rates are reported in Table 2 (the former averaged over 20 random shuffles). EB rates are higher—this is not surprising as EB has fewer labels as compared to FS (See $|\mathcal{T}|$ in Table 1),

and hence TG-SUM has higher degree of freedom to merge nodes on EB-labeled graphs. As such, Actual values are not directly comparable. What is comparable is the difference from Shuffled, that is, how much the labeling can improve on top of the random assignment of the *same* set of labels. Here, the absolute difference is always equal or larger for EB. However, even the absolute difference of compression rates is not fair to compare—it is harder to compress a graph that has been compressed quite a bit even further. For EB, Shuffled rates are already high. Improving over Shuffled even by the same amount proves EB superior to FS. Therefore, we report the normalized gain; defined as (Actual−Shuffled) / (1−Shuffled), which shows that our expert-designed EB labeling is better, for the aforementioned reasons.

5.4 Quantitative Evaluation: Compression rate, Running time, Scalability

Quantitatively, we measure summarization performance in terms of both (1) running time, as well as (2) the size reduction achieved in terms of bits (including bits required for correction). Specifically, upon obtaining the total number of bits (as given by the encoding scheme of each method), we measure the compression ratio as Compress Ratio $= \frac{\text{Bits\_Before}-\text{Bits\_After}}{\text{Bits\_Before}} \in [0, 1)$, that is the fraction of the encoding cost that has been reduced by summarization. We compare with the Navlakha algorithm [20], which we modified to handle edge directions and node labels. We run TG-SUM by gradually increasing b, to increase the number of candidate sets and obtain multi-resolution summaries. A larger number of candidates is expected to yield higher compression ratio, albeit at the cost of increased running time—hence enabling the user to choose a suitable trade-off in practice.

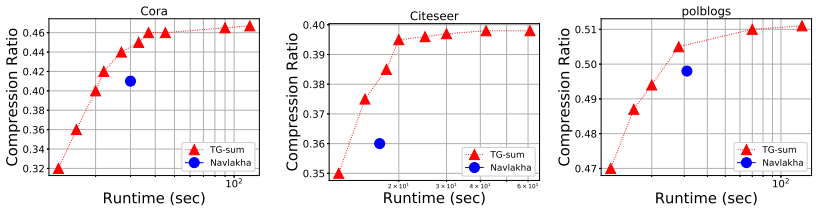

Fig. 3: Compression ratio vs. runtime on undirected graphs

Results are given in Fig. 3, where TG-SUM remarkably outperforms the alternative in terms of compression ratio in almost all cases. In absolute terms, it achieves roughly 30–60% compression across these various real-world graphs with up to hundreds of thousands of edges and tens of distinct labels. Finally, we measure the scalability of TG-SUM by first generating an increasing size synthetic directed k-out graph, where nodes are incrementally added and connected

to $k = 10$ of the existing nodes, simulating a preferential attachment process. Results in Fig. 4 (left) show that, unlike the modified Navlakha, the runtime of TG-SUM grows in a near-linear fashion.

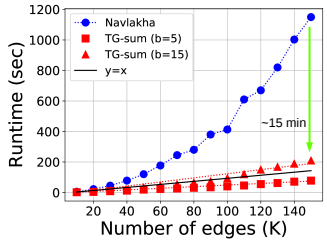

Fig. 4: TG-SUM complexityscales linearly with the number of edges.

6 Conclusion

We introduced TG-SUM, a versatile graph summarization algorithm that (for the first time) can handle *directed*, *node-labeled*, *multi*-graphs with possible self-loops (or *any* combination). Built on a novel encoding scheme, TG-SUM seeks to minimize the total encoding cost of (*i*) a summary graph, and (*ii*) the corrections to reconstruct the input graph losslessly. It efficiently finds structurally-similar nodes to create super-nodes of larger sizes incrementally, producing multi-resolution summaries. Extensive experiments show that TG-SUM (1) provides insights into the high-level structure of real-world graphs, (2) achieves better trade-off between compression and runtime relative to baselines (only) on comparable settings, and (3) scales linearly in the number of edges.

Acknowledgements This work has been sponsored by the U.S. National Science Foundation CAREER 1452425 and the PwC Risk and Regulatory Services Innovation Center at Carnegie Mellon University. Any conclusions expressed in this material are those of the author and do not necessarily reflect the views, expressed or implied, of the funding parties.

References

1. Adamic, L.A., Glance, N.: The political blogosphere and the 2004 us election: divided they blog. In: Proceedings of the 3rd Intl. Workshop on Link Discovery. pp. 36–43 (2005)
2. Adler, M., Mitzenmacher, M.: Towards compressing web graphs. In: DCC. IEEE Comp. Soc. (2001)

3. Akoglu, L.: Quantifying political polarity based on bipartite opinion networks. In: ICWSM (2014)
4. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM pp. 117–122 (2008)
5. Besta, M., Hoefler, T.: Survey and taxonomy of lossless graph compression and space-efficient graph representations (2018), arxiv:1806.01799
6. Boldi, P., Vigna, S.: The webgraph framework i: compression techniques. In: WWW. pp. 595–602 (2004)
7. Buehrer, G., Chellapilla, K.: A scalable pattern mining approach to web graph compression with communities. In: WSDM. pp. 95–106. ACM (2008)
8. Chong, E.K., Zak, S.H.: An introduction to optimization. John Wiley & Sons (2004)
9. Cook, D.J., Holder, L.B.: Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research **1**, 231–255 (1993)
10. Dunne, C., Shneiderman, B.: Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In: SIGCHI. pp. 3247–3256 (2013)
11. Giles, C.L., Bollacker, K.D., Lawrence, S.: Citeseer: An automatic citation indexing system. In: Proc. of the Conf. on Digital Libraries. pp. 89–98 (1998)
12. Grünwald, P.D.: The Minimum Description Length Principle. The MIT Press, Cambridge, MA (2007)
13. Kang, U., Faloutsos, C.: Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In: ICDM. pp. 300–309 (2011)
14. Khan, K.U., Nawaz, W., Lee, Y.K.: Set-based approximate approach for lossless graph summarization. Computing **97**(12) (2015)
15. Koutra, D., Kang, U., Vreeken, J., Faloutsos, C.: Summarizing and understanding large graphs. Statistical Analysis and Data Mining **8**(3), 183–202 (2015)
16. LeFevre, K., Terzi, E.: Grass: Graph structure summarization. In: SDM. pp. 454–465. SIAM (2010)
17. Liakos, P., Papakonstantinopoulou, K., Sioutis, M.: Pushing the envelope in graph compression. In: CIKM. pp. 1549–1558 (2014)
18. Liu, Y., Safavi, T., Dighe, A., Koutra, D.: Graph summarization methods and applications: A survey. ACM Comput. Surv. **51**(3) (2018)
19. Liu, Z., Yu, J.X., Cheng, H.: Approximate homogeneous graph summarization. Info. and Media Tech. **7**(1), 32–43 (2012)
20. Navlakha, S., Rastogi, R., Shrivastava, N.: Graph summarization with bounded error. In: SIGMOD. pp. 419–432. ACM (2008)
21. Oughtred, R., Stark, C., Breitkreutz, B.J., Rust, J., Boucher, L., Chang, C., Kolas, N., O'Donnell, L., Leung, G., McAdam, R., et al.: The biogrid interaction database **47**(D1), D529–D541 (2019)
22. Pattabiraman, B., Patwary, M.M.A., Gebremedhin, A.H., Liao, W.k., Choudhary, A.: Fast algorithms for the maximum clique problem on massive sparse graphs. In: Intl Wksp. AMWG (2013)
23. Riondato, M., García-Soriano, D., Bonchi, F.: Graph summarization with quality guarantees. Data Mining and Knowledge Discovery **31**(2), 314–349 (2017)
24. Shin, K., Ghoting, A., Kim, M., Raghavan, H.: Sweg: Lossless and lossy summarization of web-scale graphs. In: WWW. pp. 1679–1690. ACM (2019)
25. Tian, Y., Hankins, R., Patel, J.: Efficient aggregation for graph summarization. In: SIGMOD (2008)

26. Toivonen, H., Zhou, F., Hartikainen, A., Hinkka, A.: Compression of weighted graphs. In: KDD (2011)
27. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: NeurIPS. pp. 4800–4810 (2018)
28. Zhu, L., Ghasemi-Gol, M., Szekely, P., Galstyan, A., Knoblock, C.A.: Unsupervised entity resolution on multi-type graphs. In: Intl Semantic Web Conf. pp. 649–667 (2016)