SSumM: Sparse Summarization of Massive Graphs

Kyuhan Lee†
KAIST AI
kyuhan.lee@kaist.ac.kr

Hyeonsoo Jo†
KAIST AI
hsjo@kaist.ac.kr

Jihoon Ko
KAIST AI
jhoonko@kaist.ac.kr

Sungsu Lim
CNU CSE
sungsu@cnu.ac.kr

Kijung Shin†∗
KAIST AI & EE
kijungs@kaist.ac.kr

ABSTRACT

Given a graph $G$ and the desired size $k$ in bits, how can we summarize $G$ within $k$ bits, while minimizing the information loss?

Large-scale graphs have become omnipresent, posing considerable computational challenges. Analyzing such large graphs can be fast and easy if they are compressed sufficiently to fit in main memory or even cache. Graph summarization, which yields a coarse-grained summary graph with merged nodes, stands out with several advantages among graph compression techniques. Thus, a number of algorithms have been developed for obtaining a concise summary graph with little information loss or equivalently small reconstruction error. However, the existing methods focus solely on reducing the number of nodes, and they often yield dense summary graphs, failing to achieve better compression rates. Moreover, due to their limited scalability, they can be applied only to moderate-size graphs.

In this work, we propose SSumM, a scalable and effective graph summarization algorithm that yields a sparse summary graph. SSumM not only merges nodes together but also sparsifies the summary graph, and the two strategies are carefully balanced based on the minimum description length principle. Compared with state-of-the-art competitors, SSumM is (a) Concise: yields up to 11.2x smaller summary graphs with similar reconstruction error, (b) Accurate: achieves up to 4.2x smaller reconstruction error with similarly concise outputs, and (c) Scalable: summarizes 26x larger graphs while exhibiting linear scalability. We validate these advantages through extensive experiments on 10 real-world graphs.

ACM Reference Format:
Kyuhan Lee, Hyeonsoo Jo, Jihoon Ko, Sungsu Lim, and Kijung Shin. 2020. SSumM: Sparse Summarization of Massive Graphs. In Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20). ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403057

1 INTRODUCTION

Graphs are a fundamental abstraction that is widely used to represent a variety of relational datasets. As the amount of data is accumulated rapidly, massive graphs have appeared, such as (a) 3.5 billion web pages with 128 billion hyperlinks [25], (b) professional networks with more than 20 billion connections [33], and (c) social networks with hundreds of billions of connections [8].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)
August 23–27, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7998-4/20/08...
https://doi.org/10.1145/3394486.3403057

Figure 1: SSumM gives sparse and concise summary graphs. The reconstruction errors of the above summary graphs, obtained by different algorithms, are similar (±5%).

Despite the abundance of massive graphs, many existing graph-analysis tools are inapplicable to such graphs since their computational costs grow rapidly with the size of graphs. Moreover, massive graphs often do not fit in main memory, causing I/O delays over the network or to the disk.

These problems can be addressed by graph summarization, which aims to preserve the essential structure of a graph while shrinking its size by removing minor details. Given a graph $G = (V, E)$ and the desired size $k$, the objective of the graph summarization problem is to find a summary graph $\tilde{G} = (\tilde{S}, \tilde{E})$ of size $k$ from which $G$ can be accurately reconstructed. The set $\tilde{S}$ is a set of supernodes, which are distinct and exhaustive subsets of nodes in $G$, and the set $\tilde{P}$ is a set of superedges (i.e., edges between supernodes). The weight function $\omega$ assigns an integer to each superedge. Given the summary graph $\tilde{G}$, we reconstruct a graph $\hat{G}$ by connecting all pairs of nodes belonging to the source and destination supernodes of each superedge and assigning a weight, computed from the weight of the superedge, to each created edge. Note that $G$ is not necessarily the same with $\hat{G}$, and we call their similarity the accuracy of $\tilde{G}$.

Graph summarization stands out among a variety of graph-compression techniques (relabeling nodes [2, 5, 7], encoding frequent substructures with few bits [6, 13], etc.) due to the following benefits: (a) Elastic: we can reduce the size of outputs (i.e., a summary graph) as much as we want at the expense of increasing reconstruction errors. (b) Analyzable: since the output of graph summarization is also a graph, existing graph analysis and visualization tools can easily be applied. For example, [3, 19, 28] compute adjacency queries, PageRank [27], and triangle density [36] directly from summary graphs, without restoring the original graph. (c) Combinable for Additional Compression: due to the same reason, the output summary graph can be further compressed using any graph-compression techniques. While a number of graph-summarization algorithms [3, 19, 28] have been developed for finding accurate summary graphs (i.e., those with low reconstruction errors) and eventually realizing the above benefits, they share common limitations. First, their scalability is severely limited, and they cannot be applied to billion-scale graphs for which graph summarization can be extremely useful. Specifically, the largest graph to which they were applied has only
about 3 million nodes and 34 million edges, which take only about 23.8MB [3]. More importantly, existing algorithms are not effective in reducing the size in bits of graphs since they solely focus on reducing the number of nodes (see Fig. 1). Surprisingly, the size in bits of summary graphs often exceeds that of the original graphs, as reported in [28] and shown in our experiments.

To address these limitations, we propose $SSumM$ (Sparse Summarization of Massive Graphs), a scalable graph-summarization algorithm that yields concise but accurate summary graphs. $SSumM$ focuses on minimizing reconstruction errors while limiting the size in bits of the summary graph, instead of the number of nodes. Moreover, to co-optimize the compactness and accuracy, $SSumM$ carefully combines nodes and at the same time sparsifies edges. Lastly, for scalability, $SSumM$ rapidly searches promising candidate pairs of nodes to be merged. As a result, $SSumM$ significantly outperforms its state-of-the-art competitors in terms of scalability and the compactness and accuracy of outputs.

In summary, our contributions in this work are as follows:

- **Practical Problem Formulation**: We introduce a new practical variant (Problem 1) of the graph summarization problem, where the size in bits of outputs (instead of the number of supernodes) is constrained so that the outputs easily fit target storage.

- **Scalable and Effective Algorithm Design**: We propose $SSumM$ for the above problem. Compared to its state-of-the-art competitors, $SSumM$ handles up to $26\times$ larger graphs with linear scalability, and it yields up to $11.2\times$ smaller summary graphs with similar reconstruction errors (Fig. 2 and Thm. 3.4).

- **Extensive Experiments**: Throughout extensive experiments on 10 real-world graphs, we validate the advantages of $SSumM$ over its state-of-the-art competitors.

**Reproducibility**: The source code and datasets used in the paper can be found at http://dmlab.kaist.ac.kr/ssumm/.

In Sect. 2, we introduce some notations and concepts, and we formally define the problem of graph summarization within the given size in bits. In Sect. 3, we present $SSumM$, our proposed algorithm for the problem, and we analyze its time and space complexity. In Sect. 4, we evaluate $SSumM$ through extensive experiments. After discussing related work in Sect. 5, we draw conclusions in Sect. 6.

## 2 PRELIMINARIES & PROBLEM DEFINITION

We introduce some notations and concepts that are used throughout this paper. Then, we define the problem of summarizing a graph within the given size in bits. Table 1 lists some frequently-used notations, and Fig. 3 illustrates some important concepts.

### 2.1 Notations and Concepts

**Input graph**: Consider an undirected graph $G = (V, E)$ with nodes $V$ and edges $E$. Each edge $\{u, v\} \in E$ joins two distinct nodes $u \neq v \in V$. We assume that $G$ is undirected without self-loops for simplicity, while for the considered problem and our proposed algorithm can easily be extended to directed graphs with self-loops. We call nodes and edges in $G$ *subnodes* and *subedges*, respectively, to distinguish them from those in summary graphs, described below.

**Summary graph**: A summary graph $\hat{G} = (S, \hat{E}, \hat{\omega})$ of a graph $G = (V, E)$ consists of a set $S$ of supernodes, a set $P$ of superedges, and a weight function $\hat{\omega}$. The supernodes $S$ are distinct and exhaustive subsets of $V$, i.e., $\bigcup_{A \in S} A = V$ and $A \cap B = \emptyset$ for all $A \neq B \in S$. Thus, every subnode in $V$ is contained in exactly one supernode in $S$, and we denote the supernode that each subnode $v \in V$ belongs to as $V_v \in S$. Each superedge $(A, B) \in P$ connects two supernodes $A, B \in S$, and if $A = B$, then $(A, A)$ indicates the self-loop at the supernode $A \in S$. We use $\Pi_S := \{ \hat{E}\} \cup \{(A, A) : A \in S\}$ to denote the all unordered pairs of supernodes, and then $P \subseteq \Pi_S$. The weight function $\hat{\omega} : P \rightarrow \mathbb{Z}^+$ assigns to each superedge $(A, B) \in P$ the number of subedges between two subnodes belonging to $A \in S$ and $B \in S$, respectively. Let the set of such subedges as $E_{AB} := \{(u, v) \in E : u \in A, v \in B\}$. Then, $\hat{\omega}(A, B) := |E_{AB}|$ for each superedge $(A, B) \in P$. See Fig. 3 for an example summary graph.

**Reconstructed graph**: Given a summary graph $\hat{G} = (S, \hat{E}, \hat{\omega})$, we obtain a reconstructed graph $G = (V, \hat{E}, \hat{\omega})$ conventionally as in [3, 19, 28]. The set $V$ of subnodes is recovered by the union of all supernodes in $S$. The set $\hat{E}$ of subedges is defined as the set of all pairs of distinct subnodes belonging to two supernodes connected by a superedge in $P$. That is, $\hat{E} := \{(u, v) \in V \times V : u \neq v, \{V_u, V_v\} \in P\}$. The weight function $\hat{\omega} : \hat{E} \rightarrow \mathbb{R}^+$ is defined as follows:

$$\hat{\omega}(u, v) := \frac{\omega(\{V_u, V_v\})}{||V_u|| \cdot ||V_v||},$$

(1)

where $\Pi_{V_u, V_v} := \{(u, v) : u \neq v, u \in V_u, v \in V_v\}$ is the set of all possible subedges between two supernodes. That is, in Eq. (1), the denominator is the maximum number of subedges between two supernodes, and each nominator is the actual number of subedges.
Figure 3: Illustration of graph summarization. An example graph \( G \) in (a) has the adjacency matrix \( A \) in (b). From a summary graph \( \hat{G} \) in (c), we restore a graph \( \tilde{G} \), whose weighted adjacency matrix is \( \tilde{A} \) in (d). Each subnode in \( \hat{G} \) belongs to one supernode in \( \tilde{G} \), and the weight of each superedge corresponds to the number of subedges between the two supernodes. For example, since there are 3 subedges (i.e., \{1, 6\}, \{2, 3\}, \{2, 6\}) between supernodes \( V_1 \) and \( V_3 \), the weight \( \omega(\{V_1, V_3\}) \) of the superedge \( \{V_1, V_3\} \) is 3. Note that two supernodes do not have to be connected by a superedge even when there are subedges between them (see \( V_1 \) and \( V_2 \)). Eq. (1) is used for the weights of subedges in \( \hat{G} \). For example, the weight \( \hat{\omega}(\{1, 3\}) \) of the subedge \( \{1, 3\} \) in \( \hat{G} \) is \( \hat{\omega}(\{V_1, V_3\}) = \frac{3}{8} \).

2.2 Problem Definition

Now that we have introduced necessary concepts, we formally define, in Problem 1, the problem of graph summarization within the given size in bits. Then, we discuss how we measure the reconstruction error and size of summary graphs. Lastly, we compare the defined problem with the original graph summarization problem.

**Problem 1** (Graph Summarization within a Budget in Bits):

- **Given:** a graph \( G = (V, E) \) and the desired size \( k \) in bits
- **Find:** a summary graph \( \hat{G} = (\hat{S}, \hat{P}, \hat{\omega}) \)
- **to Minimize:** the reconstruction error
- **Subject to:** \( \text{Size}(\hat{G}) \leq k \).

**Reconstruction error:** The reconstruction error corresponds to the difference between the original graph \( G \) and the graph \( \hat{G} \) reconstructed from the summary graph \( \hat{G} \). While there can be many different ways of measuring the reconstruction error, as in the previous studies of graph summarization [3, 19, 28], we use the \( \ell_p \) reconstruction error \( (RE_p) \), defined as

\[
RE_p(G|\hat{G}) := \left( \sum_{i,j=1}^{V} |A(i,j) - \hat{A}(i,j)|^p \right)^{1/p},
\]

where \( A(i,j) \) is the \((i,j)\)-th entry of the matrix \( A \). Recall that \( A \) and \( \hat{A} \) are the (weighted) adjacency matrices of \( G \) and \( \hat{G} \), respectively.

**Size of summary graphs:** As in the previous studies [3, 19, 28], we define the size in bits of (summary) graphs based on the assumption that they are stored in the edge list format. Specifically, the size in bits of the input graph \( G = (V, E) \) is defined as

\[
\text{Size}(G) := 2|E| \log_2 |V|,
\]

since each of \(|E|\) subedges consists of two subnodes, each of which is encoded using \( \log_2 |V| \) bits. Note that, in order to distinguish \(|V|\) items, at least \( \log_2 |V| \) bits per item are required. Similarly, the size in bits of the summary graph \( \hat{G} = (\hat{S}, \hat{P}, \hat{\omega}) \) is defined as

\[
\text{Size}(\hat{G}) := |\hat{P}|(2 \log_2 |S| + \log_2 \omega_{\text{max}}) + |V| \log_2 |S|,
\]

where \( \omega_{\text{max}} := \max_{(A,B) \in P} \omega(A,B) \) is the maximum superedge weight. The first term in Eq. (4) is for \(|P|\) superedges, each of which consists of two supernodes and an edge weight, which are encoded using \( 2 \log_2 |S| \) bits and \( \log_2 \omega_{\text{max}} \) bits, respectively. Again, in order to distinguish \(|S|\) (or \( \omega_{\text{max}} \)) items, at least \( \log_2 |S| \) (or \( \log_2 \omega_{\text{max}} \)) bits are required for encoding each item.\(^2\) The second term in Eq. (4) is for the membership information. Each of \(|V|\) nodes belongs to a single supernode, which is encoded using \( \log_2 |S| \) bits.

**Comparison with the original problem:** Different from Problem 1, where we constrain the size in bits of a summary graph, the number of supernodes is constrained in the original graph summarization problem [19]. By constraining the size in bits, we can easily make summary graphs tightly fit in target storage (main memory, cache, etc.). On the other hand, it is not trivial to control the number of nodes so that a summary graph tightly fits in target storage. This is because how the size of summary graphs changes depending on the number of supernodes varies across datasets.

3 PROPOSED METHOD

We propose \textsc{SSumM} (Sparse Summarization of Massive Graphs), a scalable and effective algorithm for Problem 1. \textsc{SSumM} is a randomized greedy search algorithm equipped with three novel ideas.

One main idea of \textsc{SSumM} is to carefully balance the changes in the reconstruction error and size of the summary graph at each step of the greedy search. To this end, \textsc{SSumM} adapts the minimum description length principle (the MDL principle) [29] to measure both the reconstruction error and size commonly in the number of bits. Then, \textsc{SSumM} performs a randomized greedy search, aiming to minimize the total number of bits.

Another main idea of \textsc{SSumM} is to tightly combine two strategies for summarization: merging supernodes into a single supernode, and sparsifying the summary graph. Specifically, instead of creating all possible superedges as long as their weight is not zero, \textsc{SSumM} selectively creates superedges so that its cost function is minimized. \textsc{SSumM} also takes this selective superedge creation into consideration when deciding supernodes to be merged.

Lastly, \textsc{SSumM} achieves linear scalability by rapidly but effectively finding promising candidate pairs of supernodes to be merged.

In this section, we present the cost function (Sect. 3.1) and the search method (Sect. 3.2) of \textsc{SSumM}. After that, we analyze its time and space complexity (Sect. 3.3).

\(^2\)Since \( \omega \) cannot be zero in our algorithm, we need to distinguish \( \omega_{\text{max}} \) potential distinct values, i.e., \( 1, 2, \ldots, \omega_{\text{max}} \).
3.1 Cost Function in SSumM

In this subsection, we introduce the cost function, which is used at each step of the greedy search in SSumM. The cost function is for measuring the quality of candidate summary graphs by balancing the size and the reconstruction error, which is important since SSumM aims to reduce the size of the output summary graph while increasing the reconstruction error as little as possible.

For balancing the size and reconstruction error, they need to be directly comparable. To this end, we measure both in terms of the number of bits by adapting the minimum description length principle. The principle states that given data, which is the input graph \( G \) in our case, the best model, which is a summary graph \( \bar{G} \), for the data is the one that minimizes \( \text{Cost}(\bar{G}, G) \), the description length in bits of \( G \) defined as

\[
\text{Cost}(\bar{G}, G) := \text{Cost}(\bar{G}) + \text{Cost}(G|\bar{G}), \tag{5}
\]

where the description length is divided into the model cost \( \text{Cost}(\bar{G}) \) and the data cost \( \text{Cost}(G|\bar{G}) \). The model cost measures the number of bits required to describe \( \bar{G} \). The data cost measures the number of bits required to describe \( G \) given \( \bar{G} \) or equivalently to describe the difference between \( G \) and \( \bar{G} \), which is reconstructed from \( \bar{G} \). Thus, the data cost is naturally interpreted as the reconstruction error of \( \bar{G} \) in bits. Note that, if \( G = \bar{G} \) without any reconstruction error, then \( \text{Cost}(\bar{G}, G) \) becomes zero.

Eq. (5) is the cost function that SSumM uses to balance the size and reconstruction error and thus to measure the quality of candidate summary graphs. Below, we describe each term of Eq. (5) in detail, and then we divide it into the cost for each supernode.

**Model cost:** For the model cost, we use Eq. (6). It is an upper bound of Eq. (4) which measures the size of a summary graph in bits. In Eq. (6), \( \log_2 |V| (\geq \log_2 |S|) \) and \( \log_2 |E| (\geq \log_2 |\omega_{\text{max}}|) \) bits are used to distinguish supernodes and superedges, respectively. That is,

\[
\text{Cost}(\bar{G}) := |P|(2 \log_2 |V| + \log_2 |E|) + |V| \log_2 |V|. \tag{6}
\]

We divide the total model cost into the model cost for each supernode pair as follows:

\[
\text{Cost}(\bar{G}) = |V| \log_2 |V| + \sum_{\{A, B\} \in \Pi_S} \text{Cost}(E_{AB}|\bar{G}), \tag{7}
\]

where \( \text{Cost}(\bar{G}) \) is the number of bits required to exactly describe \( G \), or equivalently all subedges in \( G \), given \( \bar{G} \). As explained above, \( \text{Cost}(\bar{G}) \) is naturally interpreted as the reconstruction error of \( \bar{G} \) in bits. We divide the total data cost into the data cost for each supernode pair as follows:

\[
\text{Cost}(G|\bar{G}) = \sum_{\{A, B\} \in \Pi_S} \text{Cost}(E_{AB}|\bar{G}), \tag{8}
\]

where \( \text{Cost}(E_{AB}|\bar{G}) \) is the number of bits required to describe the subedges between the supernodes \( A \) and \( B \) (i.e., \( E_{AB} \)).

For each \( \text{Cost}(E_{AB}|\bar{G}) \), we assume a dual-encoding method to take into consideration both cases where the superedge \( \{A, B\} \) exists or not. Specifically, one between two encoding methods is used depending on whether the superedge \( \{A, B\} \) exists in \( \bar{G} \) or not. In a case where \( \{A, B\} \) exists in \( \bar{G} \), the first encoding method is used, and it optimally assigns bits to denote whether each possible subedge in \( \Pi_{AB} \) exists or not. Then, the number of bits required is tightly lower bounded by the Shannon entropy \([31]\). Thus, we define \( \text{Cost}(E_{AB}|\bar{G}) \) as

\[
\text{Cost}_{t1}(E_{AB}|\bar{G}) := -|\Pi_{AB}|(\sigma \log_2 \sigma + (1 - \sigma) \log_2(1 - \sigma)). \tag{9}
\]

where \( \sigma := \frac{|E_{AB}|}{|\Pi_{AB}|} \) is the proportion of existing subedges in \( \Pi_{AB} \).

Note that in order to compute \( \sigma \), the superedge \( \{A, B\} \) and its weight \( \omega(\{A, B\}) \) need to be retained in \( \bar{G} \).

In a case where \( \{A, B\} \) does not exist in \( \bar{G} \), the second encoding method is used, and it simply lists all existing subedges in \( E_{AB} \). Then, the number of required bits is

\[
\text{Cost}_{t2}(E_{AB}|\bar{G}) := 2|E_{AB}| \log_2 |V|. \tag{10}
\]

where \( 2 \log_2 |V| \) is the number bits required to encode an subedge. Note that, for this encoding method, the superedge \( \{A, B\} \) and its weight \( \omega(\{A, B\}) \) do not need to be retained in \( \bar{G} \).

Then, the final number of bits required to describe \( E_{AB} \) is

\[
\text{Cost}(E_{AB}|\bar{G}) := \begin{cases} 
\text{Cost}_{t1}(E_{AB}|\bar{G}) & \text{if } \{A, B\} \in P \\
\text{Cost}_{t2}(E_{AB}|\bar{G}) & \text{otherwise} 
\end{cases}. \tag{11}
\]

**Cost decomposition:** By combining Eq. (5)–Eq. (11), the total description cost \( \text{Cost}(\bar{G}, G) \) can be divided into that for each supernode pair as follows:

\[
\text{Cost}(\bar{G}, G) = |V| \log_2 |V| + \sum_{\{A, B\} \in \Pi_S} \text{Cost}_{AB}(\bar{G}, G),
\]

where \( \text{Cost}_{AB}(\bar{G}, G) \), the total description cost for each supernode pair \( \{A, B\} \in \Pi_S \), is defined as

\[
\text{Cost}_{AB}(\bar{G}, G) := \text{Cost}(\{A, B\}|\bar{G}) + \text{Cost}(E_{AB}|\bar{G}). \tag{12}
\]

Based on this cost, we also define the total description cost of each supernode \( A \) by summing the costs for the pairs containing \( A \), i.e.,

\[
\text{Cost}_{A}(\bar{G}, G) := \sum_{B \in S} \text{Cost}_{AB}(\bar{G}, G). \tag{13}
\]

Eq. (13) is used by SSumM when deciding supernodes to be merged, as described in detail in the following subsection.

**Optimal encoding given a set of supernodes:** Once a set \( S \) of supernodes is fixed, then the set \( P \) of superedges that minimizes Eq. (5) is easily obtained by minimizing Eq. (12) for each pair \( \{A, B\} \in \Pi_S \) of supernodes. That is, the superedge between each pair \( \{A, B\} \) is created if and only if it reduces Eq. (12). We let \( P^*(S) \) be the set of superedges that minimizes Eq. (5) given \( S \), and we let \( G^*(S) = (S, P^*(S), \omega) \) be the summary graph consisting of \( S \) and \( P^*(S) \). Then, minimizing Eq. (5) is equivalent to finding \( S \) that minimizes

\[
\text{Cost}^*(S) := \text{Cost}(G^*(S)) + \text{Cost}(G|G^*(S)). \tag{14}
\]

Similarly, as in Eq. (12) and Eq. (13), we let the description costs of each supernode pair \( \{A, B\} \in \Pi_S \) and supernode \( A \in S \) in \( G^*(S) \) be

\[
\text{Cost}^*_A(S) := \text{Cost}_{AB}(G^*(S), G), \tag{15}
\]

\[
\text{Cost}^*_A(S) := \sum_{B \in S} \text{Cost}^*_{AB}(S). \tag{16}
\]
SSumM: Sparse Summarization of Massive Graphs

3.2 Search Method in SSumM

Now that we have defined the cost function (i.e., Eq. (14)) for measuring the quality of candidate summary graphs, we present how SSumM performs a rapid and effective randomized greedy search over candidate summary graphs. We first provide an overview of SSumM, and then we describe each step in detail.

3.2.1 Overview (Alg. 1). Given an input graph \( G = (V, E) \), the desired size \( k \) in bits of the summary graph, and the number \( T \) of iterations, SSumM produces a summary graph \( \tilde{G} = (S, P, \omega) \). SSumM first initializes \( \tilde{G} \) to \( G \). That is, \( S = \{\{u\} : u \in V\} \) and \( P = \{\{V_u, V_v\} : (u, v) \in E\} \) (lines 1-2). Then, it repeatedly merges pairs of supernodes and sparsifies the summary graph by alternatively running the following two phases until the size of the summary graph reaches \( k \) or the number of iterations reaches \( T \):

- **Candidate generation (line 5)**: To rapidly and effectively search promising pairs of supernodes whose merger significantly reduces the cost function, SSumM first divides \( S \) into candidate sets \( S_t \) each of which consists of supernodes within 2 hops. To take more pairs of supernodes into consideration, SSumM changes \( S_t \) probabilistically at each iteration \( t \).

- **Merging and sparsification (lines 6-7)**: Within each candidate set, obtained in the previous phase, SSumM repeatedly merges two supernodes whose merger reduces the cost function most. Simultaneously, SSumM sparsifies the summary graph by selectively creating superedges adjacent to newly created supernodes. Each superedge is created only when it reduces the cost function.

After that, if the size of summary graph is still larger than the given target size \( k \), the following phase is executed:

- **Further sparsification (lines 11-12)**: SSumM further sparsifies the summary graph until its size reaches the given target size \( k \). Specifically, SSumM repeatedly removes a superedge so that the cost function is minimized.

Lastly, SSumM returns the summary graph as an output. In the following subsections, we present each phase in detail.

3.2.2 Candidate generation phase. The objective of this step is to find candidate sets of supernodes. SSumM uses the candidate sets in the next merging and sparsification phase, and specifically, it searches pairs of supernodes to be merged within each candidate set. For rapid and effective search, the candidate sets should be small, and at the same time, they should contain many promising supernode pairs whose merger leads to significant reduction in the cost function, i.e., Eq. (14).

To find such candidate sets, SSumM groups supernodes within two hops of each other. If we define the distance between two supernodes as the minimum distance between subnodes in one supernode and those in the other, merging supernodes within two hops tends to reduce the cost function more than merging those three or more hops away from each other, as formalized in Lemmas 3.1 and 3.2, where

\[
\text{Reduction}(A, B) := \text{Cost}^*_A(S) + \text{Cost}^*_B(S) - \text{Cost}^*_{A \cup B}(S) - \text{Cost}^*_{A \cup B}(S \cup \{A \cup B \} \setminus \{A, B\}) \quad (17)
\]

is the reduction of the cost function, i.e., Eq. (14), when two supernodes \( A \neq B \) in \( S \) are merged.

**Lemma 3.1 (Merger within 2 Hops).** If two supernodes \( A \in S \) and \( B \in S \) are within 2 hops, then

\[
\text{Reduction}(A, B) \leq \min(\text{Cost}^*_A(S), \text{Cost}^*_B(S)), \quad (18)
\]

and this inequality is tight.

**Lemma 3.2 (Merger outside 2 Hops).** If two supernodes \( A \in S \) and \( B \in S \) are 3 or more hops away from each other, then

\[
\text{Reduction}(A, B) \leq 2 \log_2 |V| + \log_2 |E|. \quad (19)
\]

See Appendix B for proofs of the lemmas. Empirically, for carefully chosen \( A \neq B \in S \) within two hops, \( \min(\text{Cost}^*_A(S), \text{Cost}^*_B(S)) \) and \( \text{Reduction}(A, B) \) are much larger than \( 2 \log_2 |V| + \log_2 |E| \).

To rapidly group supernodes within two hops of each other, SSumM divides the supernodes into those with the same shingles [7]. Note that, for a random bijective function \( h : V \rightarrow \{1, ..., |V|\} \), if we define the shingle of each supernode \( A \in S \) as

\[
f(A) := \min_{u \in A} \left( \min_{(u, v) \in E} h(u) \right),
\]

then two supernodes \( A \neq B \in S \) have the same shingle (i.e., \( f(A) = f(B) \)) only if \( A \) and \( B \) are within two hops. Specifically, until each candidate set consists of at most a constant (spec., 500) number of nodes, SSumM divides the supernodes using shingles recursively at most constant (spec., 10) times and then randomly. Note that computing the shingle of all supernodes takes \( O(|V| + |E|) \) time if we (1) create a random hash function \( h \), which takes \( O(|V|) \) time [16], (2) compute and store \( \min_{u \in A} \min_{(u, v) \in E} h(u) \) for every subnode \( u \in V \), which takes \( O(|V| + |E|) \) time, and (3) compute \( f(A) \) for every supernode \( A \in S \), which takes \( O(|V|) \) time.

2. If \( f(A) = f(B) \), there exist a subnode in \( A \) and a subnode in \( B \) within 1-hop of each other.
3.2.3 Merging and sparsification phase. In this phase, SSUMM searches a concise and accurate summary graph by repeatedly (1) merging two supernodes within the same candidate set into a single supernode and (2) greedily sparsifying its adjacent superedges. To this end, each candidate set $C$ obtained in the previous phase is processed as described in Alg. 2. SSUMM first finds two supernodes $A \neq B \in C$, among $\log_2 |C|$ randomly chosen supernode pairs of $C$, whose merger maximizes

$$
\text{Relative Reduction}(A, B) :=
1 - \frac{\text{Cost}_A^\star(S \cup \{A, B\}) - \text{Cost}_A^\star(S)}{\text{Cost}_B^\star(S)} = \frac{\text{RE}_{A,B}(S)}{\text{Cost}_B^\star(S)}
$$

(20)

where $t$ denotes the current iteration number. Once $A$ and $B$ are merged into $A \cup B$, all supernodes adjacent to $A$ and $B$ are removed (line 6), and then the superedges adjacent to $A \cup B$ are selectively created (or equivalently sparsified) so that the cost function given $S$ (i.e., $\text{Cost}_{A \cup B}(\widehat{G}, C)$ defined in Eq. (13)) is minimized.\(^3\) Merging two supernodes in a candidate set $C$ is repeated until the relative reduction (i.e., Eq. (20)) does not exceed the threshold $\theta(t)$, $\max(\log_2 |C|, 1)$ times in a row (lines 1, 2, and 8). Then, each of the other candidate sets is processed in the same manner.

By restricting its attention to a small number of supernode pairs in each candidate set, SSUMM significantly reduces the search space and achieves linear scalability (see Sect. 3.3). However, in our experiments, this reduction does not harm the quality of the output summary graph much due to (1) careful formation of candidate sets, (2) the adaptive threshold $\theta(t)$, and (3) robust termination with $\max(\log_2 |C|, 1)$ chances.

3.2.4 Further sparsification phase. This phase is executed only when the size of the summary graph after repeating the previous phases $T$ times still exceeds the given target size $k$ (lines 11-12 of Alg. 1). In this phase, SSUMM sparsifies the summary graph until its size $\text{Size}(\widehat{G})$ reaches $k$ as follows:

1. Compute the increase in the reconstruction error $\text{RE}_p$ after dropping each superedge from $P$.\(^4\) Note that $\text{RE}_p$ is directly used instead of the cost function. This is because the decrease in $\text{Size}(\widehat{G})$ after dropping each superedge is a constant (spec., $2\log_2 |S| + \log_2 \omega_{\text{max}}$) only for those with weight $\omega_{\text{max}}$.

2. Find the $\xi := \frac{\text{Size}(\widehat{G}) - k}{2\log_2 |S| + \log_2 \omega_{\text{max}}}$-th smallest increase in $\text{RE}_p$, and let it be $\Delta_e$.

3. For each superedge in $P$, drop it if the increase in $\text{RE}_p$ is smaller than or equal to $\Delta_e$.

\(^3\)Our implementation minimizes a tighter upper bound obtained by replacing $2\log_2 |V| + \log_2 |E|$ in $\text{Cost}_{A \cup B}(\widehat{G}, G)$ with $2\log_2 |S| + \log_2 \omega_{\text{max}}$. Moreover, it never creates superedges that increase the reconstruction error $\text{RE}_p$.

\(^4\)If we drop $(A, B)$, the increase in $\text{RE}_1$ is $(2|E_{AB}| / |I_{AB}| - 1) \cdot |E_{AB}|$, and that in $\text{RE}_2$ is $|E_{AB}| / |I_{AB}|$.

---

**Algorithm 2: Merging & Sparsification in a Candidate Set**

**Input:** (a) input graph $G = (V, E)$  
               (b) current summary graph $\widehat{G} = (S, P, \omega)$  
               (c) current iteration number $t$  
               (d) a candidate supernode set $C$

**Output:** updated summary graph $\widehat{G} = (S, P, \omega)$

1. num_skips ← 0;
2. while num_skips < $\max(\log_2 |C|, 1)$ do
   3. find a pair $(A, B)$ that maximizes Eq. (20) among $\log_2 |C|$ random pairs of supernodes in $C$;
   4. if Relative Reduction$(A, B) > \theta(t)$ then
      5. merge $A, B$ into $A \cup B$ both in $S$ and $C$; \(\triangleright \text{merge} \)
      6. remove the superedges adjacent to $A$ or $B$ from $P$;
      7. add the superedges adjacent to $A \cup B$ to $P$ selectively so that $\text{Cost}_{A \cup B}(\widehat{G}, G)$ is minimized; \(\triangleright \text{sparsify} \)
   8. num_skips ← num_skips + 1;
3. end
4. num_skips ← num_skips + 1;
5. end

Note that each step takes $O(|P|) = O(|E|)$ time, and to this end, the median-selection algorithm [4] is used in the second step.

3.3 Complexity Analysis

We analyze the time and space complexities of SSUMM. To this end, we define the neighborhood of a supernode $A \in S$ as $N_A := \{B \in S : \exists v \in A, \exists v \in B \text{ s.t. } (u, v) \in E\}$, i.e., the set of supernodes that include a subnode adjacent to any subnode in $A$. For simplicity, we assume $|V| = O(|E|)$, as in most real-world graphs.

**Time complexity:** SSUMM scales linearly with the size of the input graph, as formalized in Thm. 3.4, which is based on Lemma 3.3.

**Lemma 3.3.** The merging and sparsification phase, i.e., lines 6-7 of Alg. 1, takes $O(|E|)$ time.

**Proof.** Consider a candidate set $C \subseteq S_I$. Considering the termination condition (i.e., line 2 of Alg. 2), to merge a pair, $O(\log_2 |C| + 1)$ pairs are considered. Thus, finding the best pair among them takes $O(\log_2 |C| + 1) \cdot \max_{A \subseteq C} |N_A|$ time, and if Eq. (20) is greater than $\theta(t)$, then merging the pair and sparsifying the adjacent superedges takes additional $O(\sum_{A \subseteq C} |N_A|)$. In total, a merger takes $O(\log_2 |C| + 1) \cdot \max_{A \subseteq C} |N_A|$, and since at most $|C|$ merges take place within $C$, the time complexity of processing a candidate set $C$ (i.e., Alg. 2) is $O(|C| \cdot \log_2 |C| + 1) \cdot \sum_{A \subseteq C} |N_A|$, which is $O(\sum_{A \subseteq C} |N_A|)$ because we upper bound $|C|$ by a constant, as described in Sect. 3.2.2. Since $\sum_{C \in S_I} \sum_{A \subseteq C} |N_A| = \sum_{A \subseteq S} |N_A| \leq 2|E|$, processing all candidate sets in $S_I$ takes $O(|E|)$ time.

**Theorem 3.4 (Linear Scalability of SSUMM).** The time complexity of Alg. 1 is $O(T \cdot |E|)$.

**Proof.** The initialization phase takes $O(1)$ time per subnode and subedge and thus $O(|V| + |E|) = O(|E|)$ time in total. The candidate generation and further sparsification phases take $O(|V| + |E|) = O(|E|)$ time, as discussed in Sects. 3.2.2 and 3.2.4. The merging and
Figure 4: SSumM yields compact and accurate summary graphs, o.o.t.: out of time (>12hours), o.o.m.: out of memory (>64GB). SSumM yielded up to 11.2× smaller summary graphs with similar reconstruction error. It also achieved up to 4.2× smaller reconstruction error with similarly concise outputs.

**Table 2: Summary of the real-world datasets**

| Name                  | # Nodes | # Edges | Summary     |
|-----------------------|---------|---------|-------------|
| Ego-Facebook (EF)     | 4,039   | 88,234  | Social      |
| Caida (CA)            | 26,475  | 106,762 | Internet    |
| Email-Enron (EE)      | 36,692  | 183,831 | Email       |
| Amazon-0302 (A3)      | 262,111 | 899,792 | Co-purchase |
| DBLP (DB)             | 317,080 | 1,049,866 | Collaboration |
| SAA-Gs (linear sample)| 403,394 | 2,443,408 | Co-purchase |
| Skitter (SK)          | 1,696,415 | 11,095,298 | Internet |
| LiveJournal (LJ)      | 3,997,962 | 34,681,189 | Social |
| Web-UK-02 (W2)        | 18,483,186 | 261,787,258 | Hyperlinks |
| Web-UK-05 (W5)        | 39,454,463 | 783,027,125 | Hyperlinks |

**4 EXPERIMENTS**

We review our experiments designed for the following questions:

Q1. **Compactness & Accuracy**: Does SSumM yield more compact and accurate summary graphs than its best competitors?

Q2. **Speed**: Is SSumM faster than its best competitors?

Q3. **Scalability**: Does SSumM scale linearly with the size of the input graph? Can it handle graphs with about 1 billion edges?

Q4. **Effects of Parameters (Appendix A.2)**: How does the number of iterations $T$ affect the accuracy of summary graphs?

**4.1 Experimental Settings**

**Machines**: All experiments were conducted on a desktop with a 3.7 GHz Intel i5-9600k CPU and 64GB memory.

**Datasets**: We used the publicly available real-world graphs listed in Table 2 after removing all self-loops and direction of all edges.

**Implementations**: We implemented SSumM and $k$-Gs [19] in Java, and for S2L [28] and SAA-Gs [3], we used the implementation in C++ and Java, resp., released by the authors. In SSumM, the target summary size was set from 10% to 60% of the size of the input graph, at equal intervals. The number of iterations $T$ was fixed to 20 unless otherwise stated (see Appendix A.2 for its effects). For $k$-Gs, S2L, and SAA-Gs, the target number of supernodes was set from 10% to 60% of the number of nodes in the input graph, at equal intervals. For $k$-Gs, we used the SamplePairs method with $c = 1.0$, as suggested in [19]. For SAA-Gs and SAA-Gs (linear sample), the number of sample pairs was set to $\log n$ and $n$, resp., and the count-min sketch was used with $w = 50$ and $d = 2$.

**Evaluation Metrics**: We evaluated summary graphs in terms of accuracy, size, and quality. For accuracy, we measured $\ell_1$ and $\ell_2$ reconstruction errors, i.e., $RE_1$ and $RE_2$ (see Eq. (2)), and we normalized them by dividing them by the size of the adjacency matrix.\(^5\) For size, we used the number of bits required to store each summary graph (i.e., Eq. (4)). The quality of a summary graph is a metric for evaluating its accuracy and size at the same time. For quality, we used the number of bits required to store each summary graph (i.e., Eq. (4)). The quality of a summary graph is a metric for evaluating its accuracy and size at the same time.

\(^5\)We ignore the diagonals, and the size of the adjacency matrix is $|V| \cdot (|V| - 1)$.

\(^6\)Normalizing $X_i$ results in $(X_i - \min)(X_i - \min)$.

\(^7\)The maximum distance is $\sqrt{2}$. 

4.2 Q1. **Compactness and Accuracy**

We compared the size and $\ell_1$ reconstruction error ($RE_1$) of the summary graphs obtained by SSumM and its competitors. As seen in
4.3 Q2. Speed (Fig. 5)
We compared SSumM and its competitors in terms of speed and the quality of summary graphs. As seen in Fig. 5, **SSumM gave the best trade-off between speed and the quality of the summary** on all datasets. Specifically, SSumM was **406.6x faster** than S2L while giving summary graphs with better quality in the Amazon-0302 dataset. While SAA-Gs was faster than SSumM, SSumM gave outputs of much higher quality than SAA-Gs. SAA-Gs (linear sample) and k-Gs were slower with lower-quality outputs than SSumM, and they did not scale to large datasets, taking more than 12 hours.

4.4 Q3. Scalability (Fig. 6)
We evaluated the scalability of SSumM by measuring how its runtime changes depending on the size of the input graph. To this end, we used a number of graphs that are obtained from the Amazon-0601 and Skitter datasets by randomly sampling different numbers of nodes. As seen in Fig. 6, **SSumM scaled linearly with the size of the input graph**, as formulated in Thm. 3.4. In addition, SSumM successfully processed 26x larger datasets (with about 0.8 billion edges) than its best competitors, as seen Fig. 2(b).

5 RELATED WORK
Graph summarization have been studied extensively for various objectives, including efficient queries [19, 28, 34], compression [12, 26, 35], and visualization [10, 18, 23, 30, 32]. See [24] for a survey. Below, we focus on previous studies directly related to Problem 1.

Given the target number of supernodes, k-Gs [19] aims to minimize the reconstruction error by repeatedly merging a pair of supernodes that decrease the \( \ell_1 \) reconstruction error most among candidate pairs. While several sampling methods are proposed to reduce the number of candidate pairs from \( O(|V|^2) \) to \( O(|V|) \), k-Gs still takes \( O(|V|^2) \) time. Gs [19] aims to minimize its loss function, which takes both reconstruction error and the number of supernodes into consideration. Gs greedily merges supernodes, as in k-Gs, until the loss function increases.

S2L [28] uses geometric clustering for summarizing a graph with a given number of supernodes. Specifically, S2L considers each row (or column) in the adjacency matrix as a point in the \(|V|\)-dimensional space, and it employs \( k \)-means and \( k \)-median clustering to obtain clusters, each of which is considered as a supernode. It is shown that S2L provides a theoretical guarantee in terms of the \( \ell_p \) reconstruction error of its output summary graph.
speed up clustering, which incurs expensive computation of the pairwise distances between many high-dimensional points. S2L also adopts dimensionality reduction [11] and adaptive sampling [1] techniques. The scalability of S2L is still limited due to high memory requirements for clustering and high time complexity. Its time complexity, $O(|E| + k|V|)$, becomes $O(|V|^2)$ if $k = O(|V|)$.

SAA-Gs [3] is a more scalable algorithm for the same problem. Like $k$-Gs, SAA-Gs repeatedly merges the best supernode pair among some candidate pairs. When finding the candidate pairs, SAA-Gs uses a weighted sampling method designed to increase the probability that promising pairs are sampled. To speed up the candidate search, SAA-Gs maintains a tree storing the weights defined on each supernode, and it approximates reconstruction error using the count-min sketch [9]. Although it has lower time complexity (spec., $O(|V| \log^2 |V|)$), the scalability of SAA-Gs is limited due to its high memory requirements for maintaining the tree.

Different from the aforementioned algorithms, which focus solely on reducing the number of supernodes by merging nodes, our proposed algorithm SSUMM aims to minimize the size in bits of summary graphs by merging nodes and also sparsifying superedges.

A number of algorithms were developed for variants of the graph summarization problem [13, 17, 18, 26, 33]. As outputs, [13, 17, 26, 33] yield an unweighted summary graph and edge corrections (i.e., edges to be added to or removed from the restored graph).

6 CONCLUSION

In this work, we consider a new practical variant of the graph summarization problem where the target size is given in bits rather than the number of nodes so that outputs easily fit target storage. Then, we propose SSUMM, a fast and scalable algorithm for concise and accurate graph summarization. While balancing conciseness and accuracy, SSUMM greedily combines two strategies: merging nodes and sparsifying edges. Moreover, SSUMM achieves linear scalability by significantly but carefully reducing the search space without sacrificing the quality of outputs much. Throughout our extensive experiments on 10 real-world graphs, we show that SSUMM has the following advantages over its best competitors:

- **Concise and Accurate**: yields up to 11.2x more concise summary graphs with similar reconstruction error (Fig. 4).
- **Fast**: gives outputs of better quality up to 406.6x faster (Fig. 5).
- **Scalable**: summarizes graphs with about 0.8 billion edges (Fig. 2), scaling linearly with the size of the input graph (Thm. 3.4, Fig. 6).

**Reproducibility**: The source code and datasets used in the paper can be found at http://dmlab.kaist.ac.kr/ssumm/.

Acknowledgements

This work was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1F1A1059755) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)).

REFERENCES

[1] Ankur Aggarwal, Amit Deshpande, and Ravi Kannan. 2009. Adaptive sampling for $k$-means clustering. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer, 15–28.

[2] Alberto Apostolico and Guido Drovandi. 2009. Graph compression by BFS. Algorithms 2, 3 (2009), 1031–1044.

[3] Maham Anwar Beg, Muhammad Ahmad, Arif Zaman, and Imdadullah Khan. 2018. Scalable approximation algorithm for graph summarization. In PAKDD.

[4] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert En- dice Tarjan. 1973. Time bounds for selection. JCSS 7, 4 (1973), 448–461.

[5] Paolo Boldi and Sebastiano Vigna. 2004. The webgraph framework I: compression techniques. In WWW.

[6] Gregory Buehler and Kumar Chellapilla. 2008. A scalable pattern mining approach to web graph compression with communities. In WSDM.

[7] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessan- dro Panconesi, and Prabhakar Raghavan. 2009. On compressing social networks. In KDD.

[8] Avery Ching, Sergey Edenov, Maja Kabiljo, DionysisLogothetis, and Sambhavi Muthukrishnan. 2015. One trillion edges: graph processing at Facebook-scale. PVLDB 8, 12 (2015), 1804–1815.

[9] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms 55, 1 (2005), 58–75.

[10] Cody Dunne and Ben Shneiderman. 2013. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In SIGCHI.

[11] Piotr Indyk. 2006. Stable distributions, pseudorandom generators, embeddings, and data stream computation. JACM 53, 3 (2006), 307–323.

[12] Kiyatay Ulkan Khan, Waqas Nawaz, and Young-Koo Lee. 2014. Set-based unified approach for attributed graph summarization. In CBDCom.

[13] Kiyatay Ulkan Khan, Waqas Nawaz, and Young-Koo Lee. 2015. Set-based approxim ate approach for lossless graph summarization. Computing 97, 12 (2015), 1185–1207.

[14] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. Science 220, 4598 (1983), 671–680.

[15] Bryan Klimt and Yiming Yang. 2004. Introducing the Enron corpus. In CEAS.

[16] DE Knuth. 1969. Seminumerical Algorithms. The Art of Computer Programming, Vol. 2.

[17] Jihoon Ko, Yunbum Kook, and Kijung Shin. 2020. Incremental Lossless Graph Summarization. In KDD.

[18] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. 2014. VoG: Summarizing and understanding large graphs. In SDM.

[19] Kristen LeFevre and Evimaria Terzi. 2010. GraSS: Graph structure summarization. In SDM.

[20] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. 2007. The dynamics of viral marketing. TWEP 1, 1 (2007), 5.

[21] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In KDD.

[22] Jure Leskovec and Julian J Mac Leahy. 2012. Learning to discover social circles in ego networks. In NIPS.

[23] Yu Ru Lin, Hari Sundaram, and Aisling Kelliher. 2008. Summarization of social activity over time: People, actions and concepts in dynamic networks. In CIKM.

[24] Yike Liu, Tara Safavi, Abihlas Dighie, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. CSUR 51, 3 (2018), 62.

[25] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. 2015. The Graph Structure in the Web - Analyzed on Different Aggregation Levels. The Journal of Web Science 1, 1 (2015), 33–47.

[26] Saket Navlakha, Rajeev Rastogi, and Nithee Shrivastava. 2008. Graph summation with bounded error. In SIGMOD.

[27] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: Bringing order to the web. Technical Report. Stanford InfoLab.

[28] Matteo Riondato, David Garcia-Soriano, and Francesco Bonchi. 2017. Graph summarization with quality guarantees. DMKD 31, 2 (2017), 314–349.

[29] Jorma Rissanen. 1978. Modeling by shortest data description. Automatica 14, 5 (1978), 465–471.

[30] Neil Shah, Danai Koutra, Tianmian Zou, Brian Gallagher, and Christos Faloutsos. 2015. TimeCrunch: Interpretable dynamic graph summarization. In KDD.

[31] Claude E. Shannon and Warren Weaver. 1998. The mathematical theory of commu nication. University of Illinois Press.

[32] Zeshan Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. 2006. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. IEEE TVCG 12, 6 (2006), 1427–1439.

[33] Kijung Shin, Amol Ghoting, Myunghwan Kim, and Hema Raghavan. 2019. SWeG: Lossless and lossy summarization of web-scale graphs. In WWW.

[34] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. 2008. Efficient aggre gatesion with bounded error. In SIGMOD.

[35] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. 2011. Com pression of weighted graphs. In KDD.

[36] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In WWW.

[37] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. KADS 42, 1 (2015), 181–213.
The proofs are based on Lemmas B.1 and B.2.

**Lemma B.1.** If two supernodes $A \neq B \in S$ are merged into a single supernode $A' := A \cup B$, then

$$\text{Cost}^*_S(S) \leq \text{Cost}^*_{A'}(S'), \quad \forall C \in S \setminus \{A, B\},$$

where $S' := S \cup \{A'\} \setminus \{A, B\}$.

**Proof.** Let $\hat{C} := 2 \log_2 |V| + \log_2 |E|$. From Eqs. (11), (12), (15),

$$\text{Cost}^*_S(S') = \left\{ \begin{array}{ll} \hat{C} + \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S')) & \text{if } (A', C) \in P^*(S') \\ \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S')) & \text{otherwise.} \end{array} \right.$$  

We show that Eq. (22) holds by dividing into 4 cases as follows:

(1) **Case 1.** $(A, C) \notin P^*(S)$ and $(A', C) \notin P^*(S')$:

$$\text{Cost}^*_S(S) = 2|E_{AC}| \log_2 |V| \leq 2|E_{AC}| \log_2 |V| = \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S')) = \text{Cost}^*_S(S').$$  

(2) **Case 2.** $(A, C) \in P^*(S)$ and $(A', C) \in P^*(S')$:

Let $\sigma_{AC} := \frac{|E_{AC}|}{|AC|}$ and $\sigma_{A'C} := \frac{|E_{AC}|}{|AC|}$. Then,

$$\text{Cost}^*_S(S) = \hat{C} + \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S'))$$

$$\leq \hat{C} - |AC| \log_2 (\sigma_{AC} + 1 - \sigma_{AC}) \log_2 (1 - \sigma_{AC})$$

$$\leq \hat{C} - |AC| \log_2 (\sigma_{A'C} + 1 - \sigma_{A'C}) \log_2 (1 - \sigma_{A'C})$$

$$\leq \hat{C} + \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S')) = \text{Cost}^*_S(S').$$  

(3) **Case 3.** $(A, C) \notin P^*(S)$ and $(A', C) \in P^*(S')$:

$$\text{Cost}^*_S(S) = \leq \hat{C} + \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S')) = \text{Cost}^*_S(S'),$$

where the first inequality holds from Shannon’s source coding theorem [31].

(4) **Case 4.** $(A, C) \in P^*(S)$ and $(A', C) \notin P^*(S')$:

$$\text{Cost}^*_S(S) = \leq \hat{C} + \text{Cost}_{t_{12}}(E_{AC}\hat{G}'(S')) = \text{Cost}^*_S(S').$$

where the first inequality holds from the optimality of $P^*(S)$, and the second one can be shown as exactly in Eq. (24).
Lemma B.2. If two supernodes $A \neq B \in S$ are merged into a single supernode $A' := A \cup B$, then the following inequalities hold:

1. $\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S)) \\ \leq \mathrm{Cost}_{\ell}(E_{A'A'}|\bar{G}^*(S'))$, (25)

2. $\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S)) \\ \leq \mathrm{Cost}_{\ell}(E_{A'A'}|\bar{G}^*(S'))$, (26)

3. $\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S)) \\ \leq \mathrm{Cost}_{\ell}(E_{A'A'}|\bar{G}^*(S'))$, (27)

4. $\mathrm{Cost}_{\ell}(E_{A'A'}|\bar{G}^*(S'))$, (28)

where $S' := S \cup \{A', B\}$.

Proof. Let $C := \log_2 |V| + \log_2 |E|$. From Eqs. (11), (12), (15),

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

For case 1, use Shannon's source coding theorem [31]

First, we show Eq. (25) holds. Let $\sigma_{A,A'} := \frac{|E_{A, A'}|}{|E_{A,A'}|}$. Then, Eq. (9) and Shannon's source coding theorem [31] imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Next, we show Eq. (26) holds. Eq. (9) and $|E_{AA}| + |E_{BB}| \leq |E_{A'A'}|$ imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Finally, we show Eq. (27) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Lastly, we show Eq. (28) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

The proof is complete.

B.1 Proof of Lemma 3.1

Proof. Suppose that $A \neq B \in S$ are both 2 hops away from each other and $A' := A \cup B$. By Eq. (18) holds. Eq. (18) implies

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) \leq \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

From Eqs. (11), (12), (15), and Shannon's source coding theorem [31],

Second, we show Eq. (27) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Finally, we show Eq. (27) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Lastly, we show Eq. (28) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

The proof is complete.

B.2 Proof of Lemma 3.2

Proof. Suppose two supernodes $A \neq B \in S$ that are 3 or more hops away from each other are merged into a single supernode $A' := A \cup B$. Then, the following equalities hold:

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Lastly, we show Eq. (28) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

Lastly, we show Eq. (28) holds. The optimality of $P^*(S)$ and Eqs. (25) and (26) imply

$$\mathrm{Cost}_{\ell}(E_{AA}|\bar{G}^*(S)) + \mathrm{Cost}_{\ell}(E_{BB}|\bar{G}^*(S))$$

The proof is complete.