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 jihoonko@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 11.2× smaller summary graphs with similar reconstruction error, ***(b) Accurate:*** achieves up to 4.2× smaller reconstruction error with similarly concise outputs, and ***(c) Scalable:*** summarizes 26× 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 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

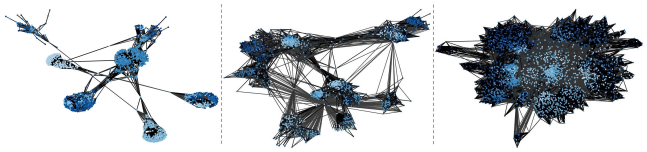
1 INTRODUCTION

Graphs are a fundamental abstraction that is widely used to represent a variety of relational datasets. As the underlying data are 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].

*Equal Contribution. †Corresponding author.

KDD '20, August 23–27, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403057



SSumM (338KB) S2L (1185KB) SAA-Gs (2343KB)

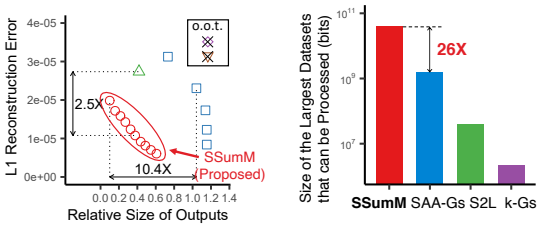
Figure 1: SSumM gives sparse and concise summary graphs. The reconstruction errors of the above summary graphs, obtained by different algorithms, are similar (±5%).

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* $\overline{G} = (S, P, \omega)$ of size k from which G can be accurately reconstructed. The set S is a set of *supernodes*, which are distinct and exhaustive subsets of nodes in G, and the set P is a set of *superedges* (i.e., edges between supernodes). The weight function ω assigns an integer to each superedge. Given the summary graph $\overline{G}$, 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 $\hat{G}$ is not necessarily the same with G, and we call their similarity the *accuracy* of $\overline{G}$.

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



(a) Compactness and Accuracy (b) Scalability

Figure 2: Advantages of SSumM. Compared to its state-of-the-art competitors, SSumM yields more compact and accurate summary graphs, and it successfully processes a 26× larger dataset with 0.8 billion edges. See Sect. 4 for details.

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 (**S**parse **Sum**marization of **M**assive 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**× larger graphs with linear scalability, and it yields up to **11.2**× 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.

Table 1: Symbols and Definitions.

Symbol	Definition
Symbols for the problem definition (Sect. 2)	
$G = (V, E)$	input graph with subnodes V and subedges E
A	adjacency matrix of G
$\overline{G} = (S, P, \omega)$	summary graph with supernodes S, superedges P, and a weight function ω
V_u	supernode with the subnode u
Π_S	set of all unordered pairs of supernodes
E_{AB}	set of subedges between the supernodes A and B
Π_{AB}	set of all possible subedges between the supernodes A and B
$\hat{G} = (V, \hat{E}, \hat{\omega})$	reconstructed graph with subnodes V, subedges $\hat{E}$, and a weight function $\hat{\omega}$
$\hat{A}$	weighted adjacency matrix of $\hat{G}$
k	desired size in bits of the output summary graph
Symbols for the proposed algorithm (Sect. 3)	
T	given number of iterations
$\theta(t)$	threshold at the t-th iteration
$\mathcal{S}_t$	set of candidate sets at the t-th iteration

2.1 Notations and Concepts

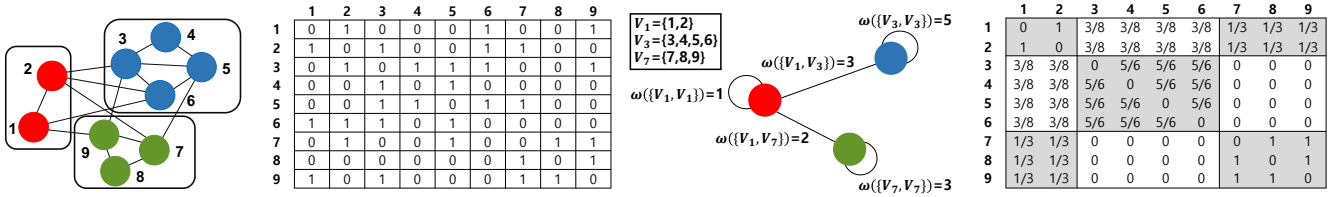
Input graph: Consider an undirected graph $G = (V, E)$ with nodes V and edges E. Each edge $\{u, v\} \in E$ joins two distinct nodes $u \neq v \in V$. We assume that G is undirected without self-loops for simplicity, while the considered problem and our proposed algorithm can easily be extended to directed 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* $\overline{G} = (S, P, \omega)$ of a graph $G = (V, E)$ consists of a set S of *supernodes*, a set P of *superedges*, and a weight function ω. 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 $\{A, B\} = \{A, A\}$ indicates the self-loop at the supernode $A \in S$. We use $\Pi_S := \binom{S}{2} \cup \{\{A, A\} : A \in S\}$ to denote the all unordered pairs of supernodes, and then $P \subseteq \Pi_S$. The weight function $\omega : P \to \mathbb{Z}^+$ 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, $\omega(\{A, B\}) := |E_{AB}|$ for each superedge $\{A, B\} \in P$. See Fig. 3 for an example summary graph.

Reconstructed graph: Given a summary graph $\overline{G} = (S, P, \omega)$, we obtain a *reconstructed graph* $\hat{G} = (V, \hat{E}, \hat{\omega})$ 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 $\hat{\omega} : \hat{E} \to \mathbb{R}^+$ is defined as follows:

$$\hat{\omega}(\{u, v\}) := \frac{\omega(\{V_u, V_v\})}{|\Pi_{V_u V_v}|} = \frac{|E_{V_u V_v}|}{|\Pi_{V_u V_v}|}, \tag{1}$$

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



(a) Input graph G (b) Adjacency matrix A (c) Summary graph $\overline{G}$ (d) Reconstructed adjacency matrix $\hat{A}$

Figure 3: Illustration of graph summarization. An example graph G in (a) has the adjacency matrix A in (b). From a summary graph $\overline{G}$ in (c), we restore a graph $\hat{G}$, whose weighted adjacency matrix is $\hat{A}$ in (d). Each subnode in G belongs to one supernode in $\hat{G}$, 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_3 and V_7). 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 $\frac{\omega(\{V_1,V_3\})}{|\Pi_{V_1 V_3}|} = \frac{3}{8}$.

between two supernodes. Note that the graph $\hat{G}$ reconstructed from $\overline{G}$ is not necessarily the same with the original graph G, and we discuss how to measure their difference in the following section.

Adjacency matrix: We use A to denote the adjacency matrix of the original graph G, and we use $\hat{A}$ to denote the weighted adjacency matrix of a reconstructed graph $\hat{G}$. See Fig. 3 for examples.

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 $\overline{G} = (S, P, \omega)$
- **to Minimize:** the reconstruction error
- **Subject to:** $Size(\overline{G}) \leq k$.

Reconstruction error: The *reconstruction error* corresponds to the difference between the original graph G and the graph $\hat{G}$ reconstructed from the summary graph $\overline{G}$. 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 ℓ_p *reconstruction error* (RE_p), defined as

$$RE_p(G|\overline{G}) := \left(\sum_{i=1}^{|V|} \sum_{j=1}^{|V|} |A(i,j) - \hat{A}(i,j)|^p \right)^{1/p}, \tag{2}$$

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

$$Size(G) := 2|E| \log_2 |V|, \tag{3}$$

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 $\overline{G} = (S, P, \omega)$ is defined as

$$Size(\overline{G}) := |P|(2 \log_2 |S| + \log_2 \omega_{max}) + |V| \log_2 |S|, \tag{4}$$

where $\omega_{max} := \max_{\{A,B\} \in P} \omega(\{A, B\})$ 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 $\log_2 \omega_{max}$ bits, respectively. Again, in order to distinguish $|S|$ (or ω_{max}) items, at least $\log_2 |S|$ (or $\log_2 \omega_{max}$) bits are required for encoding each item.[1] 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 graphs 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 SSumM (**S**parse **Sum**marization of **M**assive Graphs), a scalable and effective algorithm for Problem 1. SSumM is a randomized greedy search algorithm equipped with three novel ideas.

One main idea of 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, 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, SSumM performs a randomized greedy search, aiming to minimize the total number of bits.

Another main idea of 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, SSumM selectively creates superedges so that its cost function is minimized. SSumM also takes this selective superedge creation into consideration when deciding supernodes to be merged.

Lastly, 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 SSumM. After that, we analyze its time and space complexity (Sect. 3.3).

[1] Since ω cannot be zero in our algorithm, we need to distinguish ω_{max} potential distinct values, i.e., $\{1, 2, \ldots, \omega_{max}\}$.

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 $\overline{G}$, for the data is the one that minimizes $Cost(\overline{G}, G)$, the description length in bits of G defined as

$$Cost(\overline{G}, G) := Cost(\overline{G}) + Cost(G|\overline{G}), \tag{5}$$

where the description length is divided into the model cost $Cost(\overline{G})$ and the data cost $Cost(G|\overline{G})$. The model cost measures the number of bits required to describe $\overline{G}$. The data cost measures the number of bits required to describe G given $\overline{G}$ or equivalently to describe the difference between G and $\hat{G}$, which is reconstructed from $\overline{G}$. Thus, the data cost is naturally interpreted as the reconstruction error of $\overline{G}$ in bits. Note that, if $G = \hat{G}$ without any reconstruction error, then $Cost(G|\overline{G})$ 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 $\log_2 |E|$ ($\geq \log_2 \omega_{max}$) bits are used to distinguish supernodes and superedges, respectively. That is,

$$Cost(\overline{G}) := |P|(2\log_2 |V| + \log_2 |E|) + |V| \log_2 |V|. \tag{6}$$

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

$$Cost(\overline{G}) = |V| \log_2 |V| + \sum\nolimits_{\{A,B\} \in \Pi_S} Cost(\{A, B\}|\overline{G}), \tag{7}$$

where $Cost(\{A, B\}|\overline{G}) := \mathbb{1}(\{A, B\} \in P) \times (2\log_2 |V| + \log_2 |E|)$ is the model cost for each supernode pair $\{A, B\} \in \Pi_S$.

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

$$Cost(G|\overline{G}) = \sum\nolimits_{\{A,B\} \in \Pi_S} Cost(E_{AB}|\overline{G}), \tag{8}$$

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

For each $Cost(E_{AB}|\overline{G})$, 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 $\overline{G}$ or not. In a case where $\{A, B\}$ exists in $\overline{G}$, the first encoding method is used, and it optimally assigns bits to denote whether each possible subedge in Π_{AB} exists or not. Then, the number of bits required is tightly lower bounded by the Shannon entropy [31]. Thus, we define $Cost(E_{AB}|\overline{G})$ as

$$Cost_{(1)}(E_{AB}|\overline{G}) := -|\Pi_{AB}|(\sigma \log_2 \sigma + (1-\sigma)\log_2(1-\sigma)), \tag{9}$$

where $\sigma := \frac{|E_{AB}|}{|\Pi_{AB}|}$ is the proportion of existing subedges in Π_{AB}. Note that in order to compute σ, the superedge $\{A, B\}$ and its weight $\omega(\{A, B\})$ need to be retained in $\overline{G}$.

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

$$Cost_{(2)}(E_{AB}|\overline{G}) := 2|E_{AB}| \log_2 |V|, \tag{10}$$

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 $\overline{G}$.

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

$$Cost(E_{AB}|\overline{G}) := \begin{cases} Cost_{(1)}(E_{AB}|\overline{G}) & \text{if } \{A, B\} \in P \\ Cost_{(2)}(E_{AB}|\overline{G}) & \text{otherwise.} \end{cases} \tag{11}$$

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

$$Cost(\overline{G}, G) = |V| \log_2 |V| + \sum\nolimits_{\{A,B\} \in \Pi_S} Cost_{AB}(\overline{G}, G),$$

where $Cost_{AB}(\overline{G}, G)$, the total description cost for each supernode pair $\{A, B\} \in \Pi_S$, is defined as

$$Cost_{AB}(\overline{G}, G) := Cost(\{A, B\}|\overline{G}) + Cost(E_{AB}|\overline{G}). \tag{12}$$

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.,

$$Cost_A(\overline{G}, G) := \sum\nolimits_{B \in S} Cost_{AB}(\overline{G}, G). \tag{13}$$

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 $P^\star(S)$ be the set of superedges that minimizes Eq. (5) given S, and we let $\bar{G}^\star(S) = (S, P^\star(S), \omega)$ be the summary graph consisting of S and $P^\star(S)$. Then, minimizing Eq. (5) is equivalent to finding S that minimizes

$$Cost^\star(S) := Cost(\bar{G}^\star(S)) + Cost(G|\bar{G}^\star(S)). \tag{14}$$

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 $\bar{G}^\star(S)$ be

$$Cost^\star_{AB}(S) := Cost_{AB}(\bar{G}^\star(S), G), \tag{15}$$

$$Cost^\star_A(S) := \sum\nolimits_{B \in S} Cost^\star_{AB}(S). \tag{16}$$

Algorithm 1: Overview of SSumM

Input: (a) input graph: $G = (V, E)$
(b) the number of iterations: T
(c) desired size of $\overline{G}$: k
Output: summary graph: $\overline{G} = (S, P, \omega)$

1 $S \leftarrow \{\{u\} : u \in V\}$; ▷ initialize $\overline{G}$ to G
2 $P \leftarrow \{\{V_u, V_v\} \in \Pi_S : \{u, v\} \in E\}$;
3 $t \leftarrow 1$; ▷ t: iteration
4 **while** $t \leq T$ **and** $k < Size(\overline{G})$ **do**
5 generate candidate sets $\mathcal{S}_t \subseteq 2^S$; ▷ Sect. 3.2.2
6 **for each** *candidate set* $C \in \mathcal{S}_t$ **do**
7 greedily merges supernodes within $C \subseteq S$ and adds new superedges selectively ; ▷ Sect. 3.2.3
8 **end**
9 $t \leftarrow t + 1$;
10 **end**
11 **if** $Size(\overline{G}) > k$ **then**
12 greedily drops superedges from P so that $Size(\overline{G}) \leq$ k ; ▷ Sect. 3.2.4
13 **end**
14 **return** $\overline{G} = (S, P, \omega)$

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 $\overline{G} = (S, P, \omega)$. SSumM first initializes $\overline{G}$ to G. That is, $S = \{\{u\} : u \in V\}$ and $P = \{\{V_u, V_v\} \in S \times S : \{u, v\} \in E\}$ (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 $\mathcal{S}_t$ each of which consists of supernodes within 2 hops. To take more pairs of supernodes into consideration, SSumM changes $\mathcal{S}_t$ 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

$$\begin{aligned} Reduction(A, B) := Cost^{\star}_{A}(S) + Cost^{\star}_{B}(S) - Cost^{\star}_{AB}(S) \\ - Cost^{\star}_{A \cup B}(S \cup \{A \cup B\} \setminus \{A, B\}) \end{aligned} \tag{17}$$

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*

$$Reduction(A, B) \leq \min(Cost^{\star}_{A}(S), Cost^{\star}_{B}(S)), \tag{18}$$

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*

$$Reduction(A, B) \leq 2 \log_2 |V| + \log_2 |E|. \tag{19}$$

See Appendix B for proofs of the lemmas. Empirically, for carefully chosen $A \neq B \in S$ within two hops, $\min(Cost^{\star}_{A}(S), Cost^{\star}_{B}(S))$ and $Reduction(A, B)$ 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

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

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.[2] 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 $\min(h(u), \min_{(u,v) \in E} h(v))$ 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 $f(A) = f(B) = h(v)$, there exist a subnode in A and a subnode in B within 1-hop of v.

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

$$Relative\_Reduction(A,B) := 1 - \frac{Cost^{\star}_{A\cup B}(S \cup \{A \cup B\} \setminus \{A,B\})}{Cost^{\star}_{A}(S) + Cost^{\star}_{B}(S) - Cost^{\star}_{AB}(S)}, \tag{20}$$

which is the reduction of the cost function (i.e., Eq. (14)) due to the merger of A and B over the current cost of describing the superedges adjacent to A and B (line 4). Then, if Eq. (20) exceeds a threshold (line 4), A and B are merged into a single supernode $A \cup B$ (line 5). Inspired by simulated annealing [14] and SWeG [33], we let the threshold decrease over iterations as follows:

$$\theta(t) := \begin{cases} (1+t)^{-1} & \text{if } t < T \\ 0 & \text{if } t = T, \end{cases} \tag{21}$$

where t denotes the current iteration number. Once A and B are merged into $A \cup B$, all superedges adjacent A or 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., $Cost_{A\cup B}(\overline{G}, G)$ 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 $Size(\overline{G})$ reaches k as follows:

(1) Compute the increase in the reconstruction error RE_p after dropping each superedge from P.[4] Note that RE_p is directly used instead of the cost function. This is because the decrease in $Size(\overline{G})$ after dropping each superedge is a constant (spec., $2\log_2 |S| + \log_2 \omega_{max}$) only except for those with weight ω_{max}.
(2) Find the $\xi := \lceil \frac{Size(\overline{G})-k}{2\log_2 |S| + \log_2 \omega_{max}} \rceil$-th smallest increase in RE_p, and let it be Δ_ξ.
(3) For each superedge in P, drop it if the increase in RE_p is smaller than or equal to Δ_ξ.

[3] Our implementation minimizes a tighter upper bound obtained by replacing $2\log_2 |V| + \log_2 |E|$ in $Cost_{A\cup B}(\overline{G}, G)$ with $2\log_2 |S| + \log_2 \omega_{max}$. Moreover, it never creates superedges that increase the reconstruction error RE_p.

[4] If we drop $\{A, B\}$, the increase in RE_1 is $(2|E_{AB}|/|\Pi_{AB}| - 1) \cdot |E_{AB}|$, and that in RE_2^2 is $|E_{AB}|^2/|\Pi_{AB}|$.

Algorithm 2: Merging & Sparsification in a Candidate Set

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

```
1  num_skips ← 0 ;
2  while num_skips < max(log2 |C|, 1) do
3      find a pair {A, B} that maximizes Eq. (20) among log2 |C|
         random pairs of supernodes in C ;
4      if Relative_Reduction(A, B) > θ(t) then
5          merge A, B into A ∪ B both in S and C ;          ▷ merge
6          remove the superedges adjacent to A or B from P ;
7          add the superedges adjacent to A ∪ B to P selectively
             so that Cost_{A∪B}(Ḡ, G) is minimized ;        ▷ sparsify
8          num_skips ← 0 ;
9      end
10     else
11         num_skips ← num_skips + 1 ;
12     end
13 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 $\bar{N}_A := \{B \in S : \exists u \in A,\ \exists v \in B \text{ s.t. } \{u, v\} \in E\}$, 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 $C \in \mathcal{S}_t$. Considering the termination condition (i.e., line 2 of Alg. 2), to merge a pair, $O(\log_2^2 |C|+1)$ pairs are considered. Thus, finding the best pair among them takes $O((\log_2^2 |C| + 1) \cdot \max_{A\in C} |\bar{N}_A|)$ time, and if Eq. (20) is greater than $\theta(t)$, then merging the pair and sparsifying the adjacent superedges takes additional $O(\sum_{A\in C} |\bar{N}_A|)$ time. In total, a merger takes $O((\log_2^2 |C|+1)\cdot\sum_{A\in C} |\bar{N}_A|)$ time, and since at most $|C|$ merges take place within C, the time complexity of processing a candidate set C (i.e., Alg. 2) is $O(|C| \cdot (\log_2^2 |C| + 1) \cdot \sum_{A\in C} |\bar{N}_A|)$, which is $O(\sum_{A\in C} |\bar{N}_A|)$ because we upper bound $|C|$ by a constant, as described in Sect. 3.2.2. Since $\sum_{C\in\mathcal{S}_t} \sum_{A\in C} |\bar{N}_A| = \sum_{A\in S} |\bar{N}_A| \leq 2|E|$, processing all candidate sets in $\mathcal{S}_t$ 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

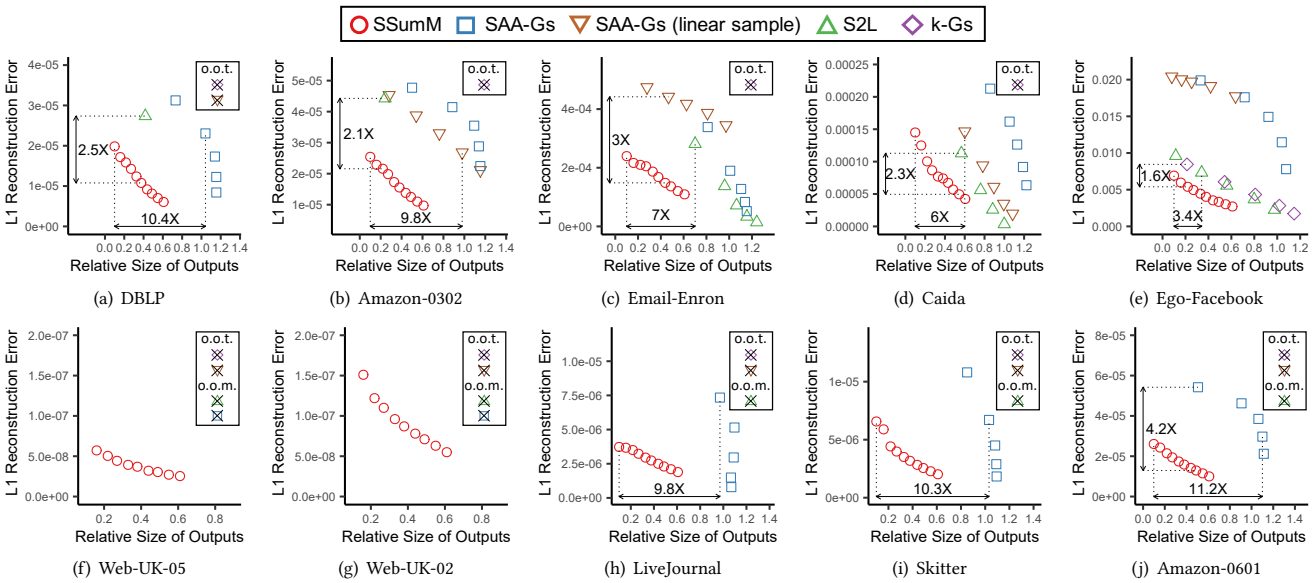


Figure 4: SSumM yields compact and accurate summary graphs. o.o.t.: out of time (>12hours). o.o.m.: out of memory (>64GB). SSumM yielded up to 11.2× smaller summary graphs with similar reconstruction error. It also achieved up to 4.2× smaller reconstruction error with similarly concise outputs.

sparsification phase also takes $O(|E|)$ time, as proven in Lemma 3.3. Since each phase is repeated at most T times, the total time complexity of Alg. 1 is $O(T \cdot |E|)$. □

Space complexity: SSumM (i.e., Alg. 1) needs to maintain (1) the input graph $G = (V, E)$, (2) the summary graph $\overline{G} = (S, P, \omega)$, (3) the neighborhood $\bar{N}_A$ of each supernode $A \in S$, and (4) a random hash function $h(v)$ for each subnode $v \in V$. Since $|S| \leq |V|$, $|P| \leq |E|$, and $\sum_{A \in S} |\bar{N}_A| \leq 2|E|$, its memory requirements are $O(|E|)$.

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 the 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 $d = 2$.

Table 2: Summary of the read-world datasets

Name	# Nodes	# Edges	Summary
Ego-Facebook (EF) [22]	4,039	88,234	Social
Caida (CA) [21]	26,475	106,762	Internet
Email-Enron (EE) [15]	36,692	183,831	Email
Amazon-0302 (A3) [20]	262,111	899,792	Co-purchase
DBLP (DB) [37]	317,080	1,049,866	Collaboration
Amazon-0601 (A6) [20]	403,394	2,443,408	Co-purchase
Skitter (SK) [21]	1,696,415	11,095,298	Internet
LiveJournal (LJ) [37]	3,997,962	34,681,189	Social
Web-UK-02 (W2) [5]	18,483,186	261,787,258	Hyperlinks
Web-UK-05 (W5) [5]	39,454,463	783,027,125	Hyperlinks

Evaluation Metrics: We evaluated summary graphs in terms of accuracy, size, and quality. For accuracy, we measured ℓ_1 and ℓ_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 (1) measured the reconstruction error RE_1 and size of the summary graphs obtained by all competitors, (2) normalized both so that they are between 0 and 1 in each dataset,[6] and (3) computed $\sqrt{\text{normalized size}^2 + \text{normalized reconstruction error}^2}$, i.e., the euclidean distance from the ideal quality.[7] All evaluation metrics were averaged over 5 iterations.

4.2 Q1. Compactness and Accuracy

We compared the size and ℓ_1 reconstruction error (RE_1) of the summary graphs obtained by SSumM and its competitors. As seen in

[5] We ignore the diagonals, and the size of the adjacency matrix is $|V| \cdot (|V| - 1)$.
[6] Normalizing X_i results in $(X_i - \min_j X_j)/(\max_j X_j - \min_j X_j)$.
[7] The maximum distance is $\sqrt{2}$.

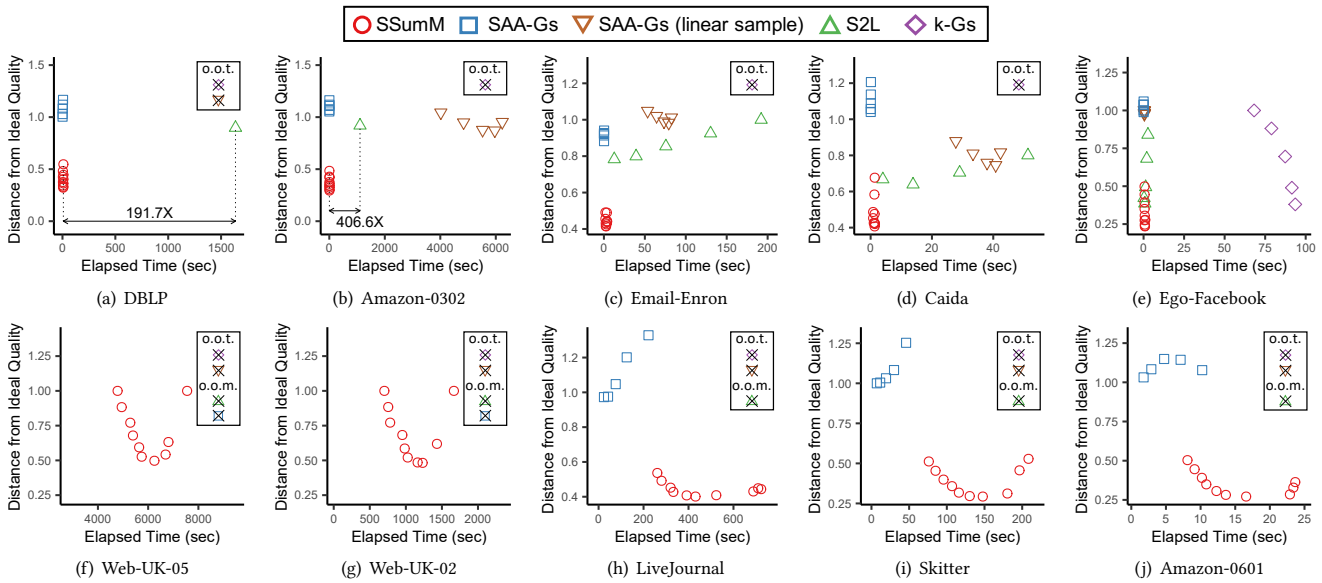


Figure 5: SSumM is fast with high-quality summary graphs. o.o.t.: out of time (>12hours). o.o.m.: out of memory (>64GB). SSumM was up to 406.6× faster than the competitors with outputs of better quality. Only SSumM scaled to the largest datasets.

Fig. 4, **SSumM yielded the most compact and accurate summaries** in all the considered datasets. Specifically, SSumM gave a **11.2× smaller summary graph** with similar or smaller RE_1 than its competitors in the Amazon-0601 dataset. It also gave a summary graph with **4.2× smaller** RE_1 but similar or smaller sizes than its competitors in the Amazon-0601 dataset. We obtained consistent results when RE_2 was used instead of RE_1 (see Appendix A.1).

Note in Fig. 4 that competitors often gave summary graphs whose (relative) size is greater than 1. That is, they failed to reduce the size in bits of the input graph since they focused solely on reducing the number of nodes. On the contrary, SSumM always gives a summary graph whose size does not exceed a given size in bits.

In Fig. 1, we visually compared the summary graphs obtained by different methods in the Ego-Facebook dataset. While they have similar ℓ_1 reconstruction error (spec., $(5.9 \pm 0.3) \times 10^{-3}$), one provided by SSumM is more concise with fewer edges.

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 **406.6× 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 Skitter and Livejournal 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

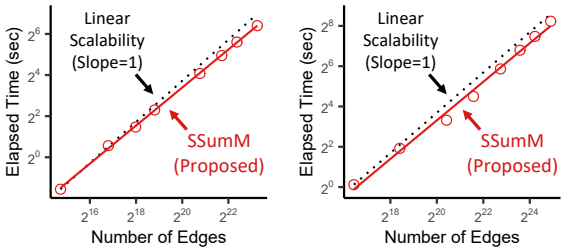


Figure 6: SSumM is scalable. SSumM scaled linearly with the number of edges in the input graph.

successfully processed 26× 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 ℓ_1 reconstruction error by repeatedly merging a pair of supernodes that decrease the ℓ_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 $O(|V|^3)$ 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 ℓ_p reconstruction error of its output summary graph. To

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 κ-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, 35]. 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 **11.2× more concise summary graphs** with similar reconstruction error (Fig. 4).
- **Fast**: gives outputs of better quality up to **406.6×** 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] Ankit 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 Endre 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 Buehrer 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, Alessandro Panconesi, and Prabhakar Raghavan. 2009. On compressing social networks. In *KDD*.
[8] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi 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] Kifayat Ullah Khan, Waqas Nawaz, and Young-Koo Lee. 2014. Set-based unified approach for attributed graph summarization. In *CBDCom*.
[13] Kifayat Ullah Khan, Waqas Nawaz, and Young-Koo Lee. 2015. Set-based approximate 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. Seminymerical 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. *TWEB* 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 Mcauley. 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, Abhilash Dighe, 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 Nisheeth Shrivastava. 2008. Graph summarization 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 García-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, Tianmin 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 communication.* University of Illinois Press.
[32] Zeqian 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 aggregation for graph summarization. In *SIGMOD*.
[35] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. 2011. Compression 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. *KAIS* 42, 1 (2015), 181–213.

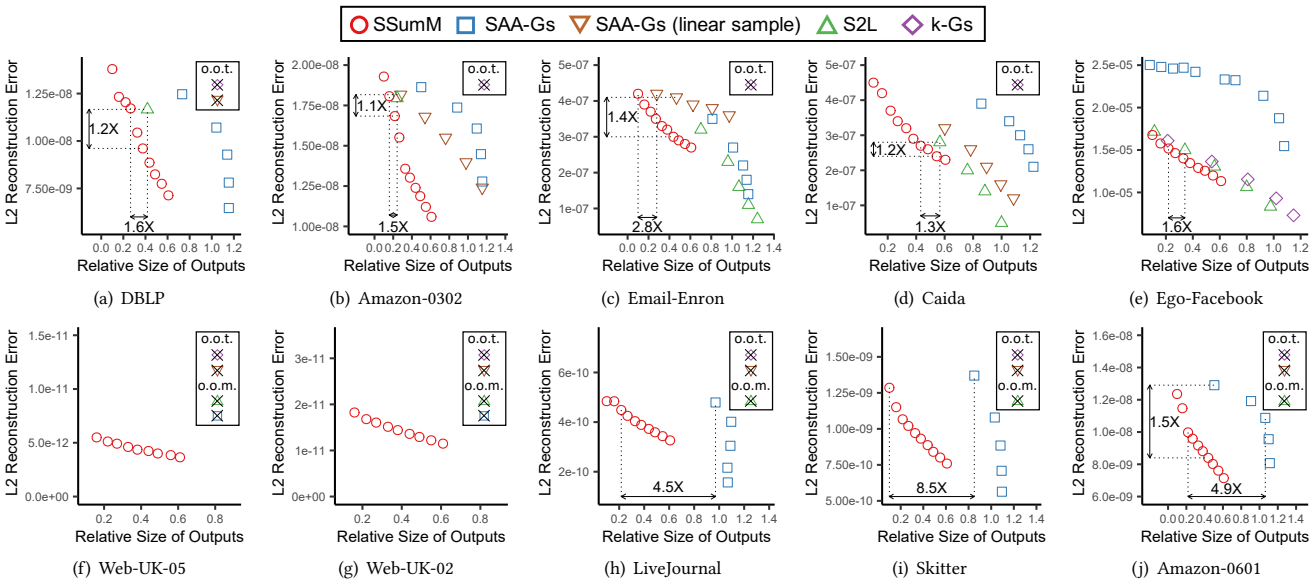


Figure 7: SSumM yields compact and accurate summaries. o.o.t.: out of time (>12hours). o.o.m.: out of memory (>64GB). Specifically, SSumM yielded up to 8.5× smaller summary graphs with similar reconstruction error (spec., RE_2). It also achieved up to 1.5× smaller reconstruction error with similarly concise outputs.

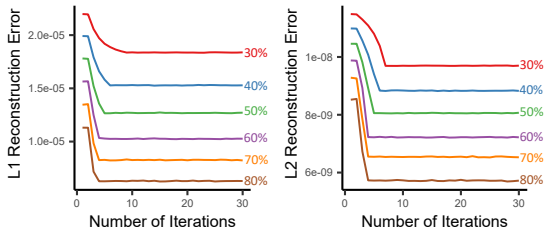


Figure 8: The effects of the iteration number T in SSumM. Regardless of the target size, the reconstruction error of the output summary graph converged within 20 iterations.

A APPENDIX: EXTRA EXPERIMENTS

A.1 Compactness and Accuracy (Fig. 7)

We compared the size and ℓ_2 reconstruction error (RE_2) of the summary graphs obtained by SSumM and its competitors in Fig. 7. As in Sect. 4.2, where RE_1 was used, SSumM consistently produced more concise and accurate summary graphs than its competitors.

A.2 Effects of Parameters (Fig. 8)

We measured how the number of iteration T in SSumM affects the reconstruction error of its summary graph in the Amazon-0601 dataset by changing the target size of summary graph evenly from 30% to 80%. As seen in Fig. 8, the reconstruction error decreased over iterations and eventually converged. As the target size decreased, more iterations were needed for convergence. In all settings, however, 20 iterations were enough for convergence.

B APPENDIX: PROOFS

In this section, we provide proofs of Lemmas 3.1 and 3.2 in Sect. 3.2.2. 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*

$$Cost^{\star}_{AC}(S) \leq Cost^{\star}_{A'C}(S'), \quad \forall C \in S \setminus \{A, B\}, \tag{22}$$

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

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

$$Cost^{\star}_{A'C}(S') = \begin{cases} \bar{C} + Cost_{(1)}(E_{A'C}|\bar{G}^{\star}(S')) & \text{if } \{A', C\} \in P^{\star}(S') \\ Cost_{(2)}(E_{A'C}|\bar{G}^{\star}(S')) & \text{otherwise.} \end{cases}$$

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

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

$$Cost^{\star}_{AC}(S) = Cost_{(2)}(E_{AC}|\bar{G}^{\star}(S)) = 2|E_{AC}|\log_2 |V|$$
$$\leq 2|E_{A'C}|\log_2 |V| = Cost_{(2)}(E_{A'C}|\bar{G}^{\star}(S')) = Cost^{\star}_{A'C}(S'). \tag{23}$$

(2) **Case 2.** $\{A, C\} \in P^{\star}(S)$ and $\{A', C\} \in P^{\star}(S')$:
Let $\sigma_{AC} := \frac{|E_{AC}|}{|\Pi_{AC}|}$ and $\sigma_{A'C} := \frac{|E_{A'C}|}{|\Pi_{A'C}|}$. Then,

$$Cost^{\star}_{AC}(S) = \bar{C} + Cost_{(1)}(E_{AC}|\bar{G}^{\star}(S))$$
$$= \bar{C} - |\Pi_{AC}|(\sigma_{AC}\log_2 \sigma_{AC} + (1-\sigma_{AC})\log_2(1-\sigma_{AC}))$$
$$\leq \bar{C} - |\Pi_{AC}|(\sigma_{A'C}\log_2 \sigma_{A'C} + (1-\sigma_{A'C})\log_2(1-\sigma_{A'C}))$$
$$\leq \bar{C} - |\Pi_{A'C}|(\sigma_{A'C}\log_2 \sigma_{A'C} + (1-\sigma_{A'C})\log_2(1-\sigma_{A'C}))$$
$$= \bar{C} + Cost_{(1)}(E_{A'C}|\bar{G}^{\star}(S)) = Cost^{\star}_{A'C}(S'), \tag{24}$$

where the first inequality holds by Shannon's source coding theorem [31].

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

$$Cost^{\star}_{AC}(S) = Cost_{(2)}(E_{AC}|\bar{G}^{\star}(S)) \leq \bar{C} + Cost_{(1)}(E_{AC}|\bar{G}^{\star}(S))$$
$$\leq \bar{C} + Cost_{(1)}(E_{A'C}|\bar{G}^{\star}(S)) = Cost^{\star}_{A'C}(S'),$$

where the first inequality holds from the optimality of $P^{\star}(S)$, and the second one can be shown as exactly in Eq. (24).

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

$$Cost^{\star}_{AC}(S) = \bar{C} + Cost_{(1)}(E_{AC}|\bar{G}^{\star}(S)) \leq Cost_{(2)}(E_{AC}|\bar{G}^{\star}(S))$$
$$\leq Cost_{(2)}(E_{A'C}|\bar{G}^{\star}(S)) = Cost^{\star}_{A'C}(S'),$$

where the first inequality holds from the optimality of $P^{\star}(S)$, and the second one can be shown as exactly in Eq. (23). □

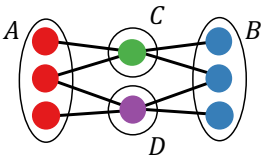


Figure 9: An example pair of supernodes $\{A, B\}$ which are 2 hops away from each other.

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:*

$$(1)\quad Cost_{(1)}(E_{AA}|\bar{G}^\star(S)) + Cost_{(1)}(E_{BB}|\bar{G}^\star(S)) \leq Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')), \tag{25}$$

$$(2)\quad Cost_{(2)}(E_{AA}|\bar{G}^\star(S)) + Cost_{(2)}(E_{BB}|\bar{G}^\star(S)) \leq Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S')), \tag{26}$$

$$(3)\quad Cost^\star_{AA}(S) + Cost^\star_{BB}(S) \leq \bar{C} + Cost^\star_{A'A'}(S'), \tag{27}$$

$$(4)\quad Cost^\star_{AA}(S) \leq Cost^\star_{A'A'}(S'), \tag{28}$$

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

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

$$Cost^\star_{A'A'}(S') = \begin{cases} \bar{C} + Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')) & \text{if } \{A', A'\} \in P^\star(S') \\ Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S')) & \text{otherwise.} \end{cases} \tag{29}$$

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

$$\begin{aligned} &Cost_{(1)}(E_{AA}|\bar{G}^\star(S)) + Cost_{(1)}(E_{BB}|\bar{G}^\star(S)) \\ &\leq -(|\Pi_{AA}| + |\Pi_{AB}|)\cdot(\sigma_{A'A'}\log_2\sigma_{A'A'} + (1-\sigma_{A'A'})\log_2(1-\sigma_{A'A'})) \\ &\leq -|\Pi_{A'A'}|(\sigma_{A'A'}\log_2\sigma_{A'A'} + (1-\sigma_{A'A'})\log_2(1-\sigma_{A'A'})) \\ &= Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')), \end{aligned}$$

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

$$\begin{aligned} &Cost_{(2)}(E_{AA}|\bar{G}^\star(S)) + Cost_{(2)}(E_{BB}|\bar{G}^\star(S)) \\ &= 2 * (|E_{AA}| + |E_{BB}|)\log_2|V| \leq 2 * |E_{A'A'}|\log_2|V| \\ &= Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S')). \end{aligned}$$

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

$$\begin{aligned} &Cost^\star_{AA}(S) + Cost^\star_{BB}(S) \\ &\quad\leq 2\bar{C} + Cost_{(1)}(E_{AA}|\bar{G}^\star(S)) + Cost_{(1)}(E_{BB}|\bar{G}^\star(S)) \\ &\quad\leq 2\bar{C} + Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')) \end{aligned} \tag{30}$$

$$\begin{aligned} &Cost^\star_{AA}(S) + Cost^\star_{BB}(S) \\ &\quad\leq Cost_{(2)}(E_{AA}|\bar{G}^\star(S)) + Cost_{(2)}(E_{BB}|\bar{G}^\star(S)) \\ &\quad\leq Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S')). \end{aligned} \tag{31}$$

The optimality of $P^\star(S')$ and Eqs. (30) and (31) imply

$$\begin{aligned} &Cost^\star_{AA}(S) + Cost^\star_{BB}(S) \\ &\quad\leq \min(2\bar{C} + Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')), Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S'))) \\ &\quad\leq \bar{C} + Cost^\star_{A'A'}(S'). \end{aligned}$$

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

$$\begin{aligned} Cost^\star_{AA}(S) &\leq \bar{C} + Cost_{(1)}(E_{AA}|\bar{G}^\star(S)) \\ &\leq \bar{C} + Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')), \end{aligned} \tag{32}$$

$$\begin{aligned} Cost^\star_{AA}(S) &\leq Cost_{(2)}(E_{AA}|\bar{G}^\star(S)) \\ &\leq Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S')). \end{aligned} \tag{33}$$

The optimality of $P^\star(S')$ and Eqs. (29), (32), and (33) imply

$$\begin{aligned} &Cost^\star_{AA}(S) \\ &\leq \min(\bar{C} + Cost_{(1)}(E_{A'A'}|\bar{G}^\star(S')), Cost_{(2)}(E_{A'A'}|\bar{G}^\star(S'))) \\ &\leq Cost^\star_{A'A'}(S'). \end{aligned}$$

□

B.1 Proof of Lemma 3.1

Proof. Suppose two $A \neq B \in S$ that are within 2 hops are merged into a single supernode $A' := A \cup B$, and without loss of generality, $Cost^\star_A(S) \geq Cost^\star_B(S)$. We let $S' := S \cup \{A'\} \setminus \{A, B\}$.

We first show that Eq. (18) holds. From Eqs. (22) and (28),

$$\begin{aligned} &Cost^\star_A(S) - Cost^\star_{AB}(S) = Cost^\star_{AA}(S) + \sum\nolimits_{C\in S\setminus\{A,B\}} Cost^\star_{AC}(S) \\ &\leq Cost^\star_{A'A'}(S') + \sum\nolimits_{C\in S\setminus\{A,B\}} Cost^\star_{A'C}(S') = Cost^\star_{A'}(S'). \end{aligned} \tag{34}$$

Eq. (17), Eq. (34), and $Cost^\star_A(S) \geq Cost^\star_B(S)$ imply Eq. (18).

Now, we show that Eq. (18) is tight. That is, we show that there exists $A \neq B \in S$ within 2 hops where

$$Reduction(A, B) = \min(Cost^\star_A(S), Cost^\star_B(S)). \tag{35}$$

Fig. 9, where A and B are 2 hops away from each other, provides such an example. In the example,

$$Cost^\star_A(S) = Cost^\star_B(S) = 2\cdot(2\log_2|V| + \log_2|E|) = Cost^\star_{A'A'}(S'),$$

and $Cost^\star_{AB}(S) = 0$. Hence,

$$\begin{aligned} Reduction(A, B) &= 2\cdot(2\log_2|V| + \log_2|E|) \\ &= \min(Cost^\star_A(S), Cost^\star_B(S)). \end{aligned}$$

□

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:

$$Cost^\star_{AB}(S) = 0, \tag{36}$$

$$Cost^\star_{AC}(S) = 0 \;\; or \;\; Cost^\star_{BC}(S) = 0, \quad \forall C \in S \setminus \{A, B\}. \tag{37}$$

Eqs. (22), (36), and (37) imply

$$Cost^\star_{AC}(S) + Cost^\star_{BC}(S) \leq Cost^\star_{A'C}(S'), \quad \forall C \in S \setminus \{A, B\}, \tag{38}$$

where $S' := S \cup \{A'\} \setminus \{A, B\}$. Then, Eqs. (27), (36) and (38) imply

$$\begin{aligned} &Cost^\star_A(S) + Cost^\star_B(S) - Cost^\star_{AB}(S) \\ &\quad= Cost^\star_{AA}(S) + Cost^\star_{BB}(S) + Cost^\star_{AB}(S) \\ &\qquad+ \sum\nolimits_{C\in S\setminus\{A,B\}} (Cost^\star_{AC}(S) + Cost^\star_{BC}(S)) \\ &\quad\leq \bar{C} + Cost^\star_{A'A'}(S') + \sum\nolimits_{C\in S\setminus\{A,B\}} Cost^\star_{A'C}(S') \\ &\quad= \bar{C} + Cost^\star_{A'}(S'), \end{aligned} \tag{39}$$

where $\bar{C} := 2\log_2|V| + \log_2|E|$. Eqs. (39) and (17) imply Eq. (19). □