SsAG: Summarization and Sparsification of Attributed Graphs

SARWAN ALI, Georgia State University, Atlanta, USA
MUHAMMAD AHMAD, Lahore University of Management Sciences, Lahore, Pakistan
MAHAM ANWER BEG, Lahore University of Management Sciences, Lahore, Pakistan
IMDAD ULLAH KHAN, Lahore University of Management Sciences, Lahore, Pakistan
SAFIULLAH FAIZULLAH, Islamic University, Madinah, KSA
MUHAMMAD ASAD KHAN, Hazara University, Mansehra, Pakistan

Graph summarization has become integral for managing and analyzing large-scale graphs in diverse real-world applications, including social networks, biological networks, and communication networks. Existing methods for graph summarization often face challenges, being either computationally expensive, limiting their applicability to large graphs, or lacking the incorporation of node attributes. In response, we introduce SsAG, an efficient and scalable lossy graph summarization method designed to preserve the essential structure of the original graph.

SsAG computes a sparse representation (summary) of the input graph, accommodating graphs with node attributes. The summary is structured as a graph on supernodes (subsets of vertices of $G$), where weighted superedges connect pairs of supernodes. The methodology focuses on constructing a summary graph with $k$ supernodes, aiming to minimize the reconstruction error (the difference between the original graph and the graph reconstructed from the summary) while maximizing homogeneity with respect to the node attributes. The construction process involves iteratively merging pairs of nodes.

To enhance computational efficiency, we derive a closed-form expression for efficiently computing the reconstruction error (RE) after merging a pair, enabling constant-time approximation of this score. We assign a weight to each supernode, quantifying their contribution to the score of pairs, and utilize a weighted sampling strategy to select the best pair for merging. Notably, a logarithmic-sized sample achieves a summary comparable in quality based on various measures. Additionally, we propose a sparsification step for the constructed summary, aiming to reduce storage costs to a specified target size with a marginal increase in RE.

Empirical evaluations across diverse real-world graphs demonstrate that SsAG exhibits superior speed, being up to $17 \times$ faster, while generating summaries of comparable quality. This work represents a significant advancement in the field, addressing computational challenges and showcasing the effectiveness of SsAG in graph summarization.

CCS Concepts: • Computing methodologies → Information extraction;

Additional Key Words and Phrases: Graph summarization, attributed graphs, graph sparsification


1 INTRODUCTION

Graph analysis is a fundamental task in various research fields such as social networks analysis, bioinformatics, internet of things, and so on [7, 37]. Graphs with millions of nodes and billions of edges are ubiquitous in many applications. The magnitude of these graphs poses significant computational challenges for graph processing. A practical solution is to compress the graph into a summary that retains the essential structural information of the original graph. Processing and analyzing the summary is significantly faster and reduces the storage and communication overhead. It also makes visualization of very large graphs possible [43].

Graph summarization plays a pivotal role in drawing insights from a social or information network while preserving users’ privacy [9]. Summarization has been successfully applied to identify critical nodes for immunization to minimize the infection spread in the graph [27]. The summary of a graph helps efficiently estimate the combinatorial trace of the original graph, which is used to select nodes for immunization [1]. Moreover, summarization techniques help generate descriptors of graphs i.e., vectors representation (also called embeddings) for efficient graph analysis [10]. Summaries of attributed graphs are used for targeted publicity campaigns, nodes clustering, de-anonymization, and nodes attribute prediction [12, 14].

Given an undirected attributed graph $G = (V, E, \mathcal{A})$, where $V$ and $E$ are sets of nodes and edges, respectively. $\mathcal{A}(v_i)$ is the attribute value for $v_i \in V$. Formally, $\mathcal{A}$ is a function mapping each node $v_i$ to one of the possible attribute values, i.e., $\mathcal{A} : V \mapsto \{a_1, a_2, \ldots, a_l\}$. For $k \in \mathbb{Z}^+$, a summary of $G$, $S = (V_S, E_S, \mathcal{A}_S)$ is a weighted graph on $k$ supernodes. $V_S = \{V_1, \ldots, V_k\}$ is a partition of $V$. Each $V_i$ has two associated integers, $n_i = |V_i|$ and $e_i = \{(u, v) : u, v \in V_i, (u, v) \in E\}$. An edge $(V_i, V_j) \in E_S$ (called a superedge) has weight $e_{ij}$, where $e_{ij}$ is the number of edges in the bipartite subgraph induced between $V_i$ and $V_j$, i.e., $e_{ij} = \{(u, v) : u \in V_i, v \in V_j, (u, v) \in E\}$. Each supernode $V_i$ has a $l$-dimensional feature vector that maintains the distribution of attribute values of nodes in $V_i$, i.e., $\mathcal{A}_S[i]|p| = |\{u_j \in V_i : \mathcal{A}(u_j) = a_p\}, 1 \leq p \leq l$.

Given a summary $S$ of $G$, the original graph can be approximately reconstructed from $S$. The reconstructed graph $G'$ is represented by the expected adjacency matrix, $A'$ defined as following (see Figure 1 for an example):

$$
A'(u, v) = \begin{cases} 
0 & \text{if } u = v \\
\frac{e_{ij}(n_i)}{n_j} & \text{if } u, v \in V_i \\
\frac{e_{ij}/n_i}{n_j} & \text{if } u \in V_i, v \in V_j
\end{cases}.
$$

(1)

The (unnormalized) $\ell_p$-reconstruction error, $RE_p$ of a summary $S$ of a graph $G$ is the $p$th norm of the error matrix ($A - A'$), where $A'$ is the expected adjacency matrix of $G'$. Note that $G'$ is an approximate reconstruction of $G$ from $S$. Formally, $RE_p$ of a summary $S$ of graph $G$ is:

$$
RE_p(G|S) = RE_p(A|A') = \left( \sum_{i=1}^{n} \sum_{j=1}^{n} |A(i, j) - A'(i, j)|^p \right)^{1/p},
$$

(2)

where $G|S$ denotes the approximate reconstruction of $G$ and $A|A'$ is the approximation of $A$ based on $S$. $RE_p$ quantifies the disagreements between $A$ and $A'$ constructed from $S$.

The attribute values for nodes are approximated from the feature vectors of supernodes in $S$. For a vertex $v_j \in V_i$, the probability that attribute value of $v_j$ is $a_p$ is $\mathcal{A}_S[i]|p|/n_i$. A summary $S$ is
Fig. 1. Figure for constructing expected adjacency matrix $A'$. A graph $G$ with gender = male(M)/female(F) attribute on nodes. $S$ on three supernodes is a summary of $G$. Each supernode $V_i$ has an attribute feature vector, $A_i$, with the counts of males and females $V_i$.

$$RE(G|S) = 16.83$$ and purity($S$) = 0.67. The expected adjacency matrix $A'$ reconstructed from $S$ using Equation (1).

homogeneous with respect to $A$ if nodes in $V_i, i \leq k$ have the same attribute values. We use the purity of the partition $V_S = \{V_1, \ldots, V_k\}$ as a measure of the homogeneity of $S$. The purity of $S$, $purity(S)$ is computed as $purity(S) = \frac{1}{n} \sum_{i=1}^{k} \max(\mathcal{A}_i)$.

The primary research objective of this work is to devise an efficient and scalable algorithm for the following dual optimization problem:

**Problem 1.** Given a graph $G = (V, E, \mathcal{A})$ and a positive integer $k \leq n$, find a summary $S$ with $k$ supernodes such that $RE(G|S)$ is minimum and $purity(S)$ is maximum over all choices of $S$.

**Summarization Solution Sketch and Computational Challenges:** As the number of possible summaries (vertex set partitions) is exponential, computing the “best” summary is challenging. A well-known method, GrASS [20], uses an agglomerative approach to form a summary of the given graph. Initially, each node is considered a supernode, and in every iteration, two supernodes are merged until the desired number of supernodes are formed. Selecting a pair of nodes to merge and computing the error incurred after merging a pair are computationally expensive steps. At iteration $t$, let $n(t)$ be the number of supernodes in summary, and the possible number of pairs of supernodes is $\binom{n(t)}{2}$. GrASS selects the best pair from a random sample of $O(n(t))$ pairs to reduce the search space. The selection and merging of the best pair take $O(n(t))$ time. The overall runtime to summarize the graph on $n$ nodes is $O(n^3)$. However, s2l [31] represents each node by an $n$-dimensional vector and applies vector-clustering to find supernodes. The runtime of s2l is $O(n^2t)$, where $t$ is the number of iterations. This runtime is still infeasible for large graphs. A recent and scalable method, SSumM [19], summarizes a graph to minimize reconstruction error (RE) and keeps the summary size within a fixed size limit. However, none of these methods incorporate node attributes for summarization.

This article proposes SsAG, a lossy summarization approach that incorporates both the graph topology and node attributes. SsAG constructs the summary by iteratively merging a pair of supernodes. We define a score function for pairs, a function of graph topology and attribute information, that quantifies the RE and purity after merging a pair. For computational efficiency, we approximate this score using a closed-form expression and storing a constant number of extra variables at each node. We assign a weight to each node to estimate the contribution of the node in the score of pairs and select nodes with probability proportional to their weights. We choose the “best pair” for merging from a weighted random sample of nodes. Using weighted sampling and approximate score, SsAG efficiently constructs a summary of comparable quality with logarithmic-sized samples. The overall runtime of SsAG to compute a summary on $k$ supernodes is $O(n \log^2 n)$. 

ACM Trans. Knowl. Discov. Data., Vol. 18, No. 6, Article 141. Publication date: April 2024.
Furthermore, we use sparsification to reduce the storage size of the summary to a target budget with negligible impact on RE. The impact on RE is shown in Section 6 for different datasets on the proposed and baseline methods.

**Remark 1.** Note that SsAG is different from attributed graph embedding, as it focuses on graph summarization, which involves creating a concise and informative representation of the original graph. In contrast, attributed graph embedding typically aims to map nodes into a continuous vector space while preserving their structural and attribute relationships. SsAG retains the essential structure of the graph in a summary, allowing for efficient analysis, while embedding methods transform nodes into vectors for downstream tasks.

The main contributions of this article are as follows:

- We quantify the **goodness** of pairs of nodes for merging that incorporates graph topology and node attribute values. Our score can be efficiently approximated by storing $O(1)$ extra information at each node.
- We give weights to nodes that measure the contribution of a node in the score of pairs. A sample of $O(\log n)$ nodes selected with probability proportional to this weight yields results comparable to a linear-sized sample.
- We implemented a data structure for sampling nodes with probability proportional to their weights in $O(\log n)$ time. Inserting, deleting, and updating nodes weights are accomplished in $O(\log n)$ time.
- With $O(1)$ score approximation, $O(\log n)$-time node sampling with probability proportional to their (dynamic weights), and a logarithmic-sized sample, SsAG takes $O(n \log^2 n)$ time to construct a summary.
- Experimental evaluation on several real-world networks shows that SsAG significantly outperforms GraSS and s2l in terms of runtime and scalability to large graphs while producing comparable quality summaries. We show that SsAG is up to $17\times$ faster than s2l while being comparable in terms of the reconstruction error.
- SsAG produces homogeneous summaries with respect to attributes. Moreover, with sparsification, the summaries produced by SsAG have low storage requirements compared to the current state-of-the-art approach of SSumM.

**Our Previous Work:** This manuscript is an extension of the earlier work [3], which describes the approximate scheme for the graph summarization based on the graph structure only. In this article, we extend the idea of Reference [3] to incorporate the node attributes along with the graph topology to make a summary. We also provide detailed proofs and analyses of the summarization algorithm. Moreover, we present a sparsification approach to drop superedges to reduce the summary size while having a negligible impact on the reconstruction error. Last, we extensively evaluate SsAG based on various evaluation measures adopted in the literature.

The remaining article is organized as follows: In Section 2, we discuss previous work on graph summarization. The proposed method and its proof of correctness are presented in Section 3. In Section 4, we give runtime analysis, the space complexity of the obtained summaries, and query computation based on a summary. The description of the dataset, baseline models, and evaluation metrics is given in Section 5. We report the results and baseline comparison in Section 6. We conclude the article in Section 7.

## 2 RELATED WORK

Graph summarization and compression have been studied for a wide array of problems and have applications in diverse domains. It is widely used in clustering, classification, community...
detection, outlier identification, network immunization, and so on. There are two main types of graph summarization methods in literature: lossless and lossy. In lossless summarization, the exact reconstruction is achieved by storing some extra information in the form of edge corrections along with the summary [25]. The edge corrections include the edges to be inserted (positive edge corrections) and edges to be deleted (negative edge corrections) from the reconstructed version of the graph. A scalable summarization approach summarizes sets of similar nodes that are found using locality-sensitive hashing [14].

In lossy graph summarization, some detailed information is compromised to reduce the size and space complexity of the original graph. Reconstruction error [20], cutnorm error [31], and error in query answering are some of the widely used quality measures of a lossy summary. Reconstruction error is the norm of the error matrix (difference between the adjacency matrices of the original and the reconstructed version of the graph) [20]. cutnorm error is defined as the maximum absolute difference between the sum of the weights of the edges between any two sets of vertices [31]. Similarly, accuracy in answers to various types of graph queries indicates the quality of the summary. For partitioning nodes into supernodes, an agglomerative approach is used in Reference [20] that greedily merges pairs of nodes to minimize the $l_1$-reconstruction error. This approach is very simple and achieves great summarization quality, but it does not scale to large graphs. Even after subsampling [20], the approach scales only to graphs of the order of a few thousand. A weighted sampling scheme is proposed in Reference [3] that can be applied to large-scale graphs. Other sampling methods are also discussed in Reference [9] for large graph analysis. Authors in Reference [29] propose a sampling algorithm that considers the network edge weights. A comparison of different sampling strategies for the processing of large graphs is also presented in Reference [6]. Authors in Reference [30] propose an idea of using stochastic graphs for the analysis of large graphs where the weights associated with the edges are random variables.

Note that various types of graphs exist in different domains. Summarization of different types of graphs is applied to get useful results. In attributed graphs, nodes have certain associated attributes (properties) [2, 4]. The lossless summarization of attributed graphs is described in Reference [15], which identifies the sets of nodes having similar neighborhood and attribute values for merging. Locality-sensitive hashing is used to select nodes having similar neighborhood and attribute values. Moreover, to construct a summary with approximate homogeneous neighborhood information and attribute values using an entropy model is described in Reference [24]. In addition to this, the summarization of attributed graphs based on a selected set of attributes is described in Reference [39]. The work also presents an operation to allow users to drill down to a larger summary for more details and roll up to a concise summary with fewer details. Another line of work for attributed graphs finds a compact subgraph of the desired number of nodes having query attribute values [12]. A survey cited in Reference [42] discusses various summarization techniques for attributed graphs. Authors in Reference [4] propose a method for attributed graph clustering using a modified random walk with a restart. A method for community detection is proposed in Reference [5] for attributed graphs using a matrix factorization-based approach.

Compression of edge-weighted graphs using locality-sensitive hashing while preserving the edge weights is described [13]. Furthermore, compression of node and edge-weighted graphs is described such that the weights on the path between two nodes in the summary graph should be similar to that in the original graph [44]. The article also aims to preserve more information related to nodes with high weights. Another closely related area is of influence graph in which nodes have influence over other nodes. The influence graph summarization takes into account the influence of nodes on other nodes in the summary graph [35].

Dynamic graphs, where nodes or edges evolve over time, are also prevalent these days. An approach discusses the summarization of a dynamic graph based on connectivity and communication.
ALGORITHM 1: AGGLOMERATIVE-SUMMARIZATION($G = (V, E, A), k$, target size)

1: $S_0 ← G$
2: $t ← 1$
3: while $S_{t-1}$ has more than $k$ nodes do
4:   Select the ‘best’ pair ($V_a, V_b$) of (super)nodes in $S_{t-1}$ \(\Rightarrow\) T-1 and T-2
5:   Merge $V_a$ and $V_b$ in $S_{t-1}$ to get $S_t$ \(\Rightarrow\) T-3
6:   Sparsify $S_t$ to the target size \(\Rightarrow\) T-4

among nodes [40]. The work creates summaries of the dynamic graph over a fixed-sized sliding window. MoSSo, a lossless approach, incrementally updates the summary in response to the deletions or additions of edges [16]. A summarization framework that captures the dynamic nature of dynamic graphs is described in Reference [28].

Webgraph summarization improves the performance of search engines [34]. They are efficiently compressed by exploiting the link structure of the web. Permuting the nodes in a web graph such that similar nodes are placed together produces improved compression results. Parallel methods are also devised to summarize massive web graphs spread over multiple machines [36].

Summarization of graph streams such that to approximately answer the queries on the graph stream is discussed in Reference [38]. The real-time summarization of massive graph streams is done using the count-min sketch approach to preserve the structural information of the graph [11].

Several methods use graph features as building blocks (vocabulary) for graph representation. VoG (Vocabulary-based summarization of graphs) summarizes graphs based on the substructures like cliques, chains, stars, and bipartite cores [17]. Graph compression based on communities identified based on central nodes and hubs is studied in Reference [22]. See Reference [23] for a survey of graph summarization techniques.

Many summarization approaches result in very dense summaries. These summaries have $k$ supernodes, but the storage cost of superedges is very high; sometimes, it even surpasses that of the original graph. A recent scalable state-of-the-art approach, SSumM [19], summarizes the graph minimizing both the reconstruction error and summary density simultaneously. In this work, we utilized the notion of sparsification of SSumM to reduce the summary size to a target size. Reference [33] proposed a framework to find a sparse summary of knowledge graphs preserving only the most relevant information. Authors in Reference [26] propose a model for movie summarization by constructing a sparse movie graph by identifying important turning points in movies. The movie summarization model highlights different graph topologies for different movie genres.

3 PROPOSED SOLUTION

In this section, we present the details of the proposed solution, SsAG for Problem 1. The goal is to construct a summary graph on $k$ supernodes that have the minimum reconstruction error $RE$ and maximum homogeneity of attribute values. After constructing the summary, we sparsify it to reduce the storage cost to the target size.

Given a graph $G = (V, E, A)$ on $n$ nodes, an integer $k$ and target storage size, SsAG produces a summary graph $S = (V_S, E_S, A_S)$ on $k$ supernodes with storage size at most the given target. We give a general overview of SsAG in Algorithm 1. $S$ is iteratively constructed in an agglomerative fashion. Initially, each node is a supernode, and in each iteration, a pair of supernodes are merged. Denote by $S_t$ the summary after iteration $t$ with $(n - t)$ supernodes. In iteration $t$, Algorithm 1 selects a pair of nodes in $S_{t-1}$ for merging that results in the least $RE(G|S_t)$ and the maximum homogeneous merged supernode.
Let \( n(t) \) be the number of supernodes in \( S_t \) and \( s_t^{(a,b)} \) be the graph obtained after merging \( V_a \) and \( V_b \) in iteration \( t \). We identify the following key tasks in Algorithm 1.

**T-1: Efficient Score Computation of a Pair of Nodes:** While constructing a summary, we select a pair of supernodes for merging such that the resulting supernode has maximum homogeneity and the supergraph has the minimum reconstruction error. For a given pair \((V_a, V_b)\) naive computation of only \( RE(G|S_t^{(a,b)}) \) from Equation (2) takes \( O(n^2) \) time.

**T-2: Selection of the Best Pair of Nodes for Merging:** By Lemma 1 the score of a pair can be estimated in constant time, however, the search space is quadratic, since there are \( \binom{n(t)}{2} \) candidate pairs. This poses a significant hurdle to the scalability of Algorithm 1 to large graphs.

**T-3: Efficient Merging of a Pair of Nodes:** The next task is to merge the selected pair. For a summary with the adjacency list representation, naively merging a pair of nodes requires traversal of the whole graph in the worst case.

**T-4: Sparsification of Summary Graph:** In many cases, the resulting summaries on \( k \) supernodes are dense and even surpass the original graph in storage cost. The last task is to sparsify the summary to the given target size.

Next, we describe in detail how SsAG accomplishes each of these tasks. Each supernode \( V_a \in V_S \) stores \( n_a \) and \( e_a \). We store the weighted adjacency list of \( S \), i.e., for edge \((V_a, V_b) \in E_S\) its weight \( e_{ab} \) is stored at nodes \( V_a \) and \( V_b \) in the list.

### 3.1 T-1: Efficient Score Computation of a Pair of Nodes

We define a score for a pair that can be efficiently estimated with some extra bookkeeping. We need to select a pair of nodes \((V_a, V_b) \in \binom{V_{S_{t-1}}}{2}\) such that \( RE(G|S_t^{(a,b)}) \) is minimum. We denote \( score_t^{RE}(a, b) \) as the score of \((V_a, V_b)\) based on graph topology, which quantifies the increase in \( RE \) after merging the pair. Note that for fixed \( S_{t-1} \), minimizing \( RE(G|S_t^{(a,b)}) \) is equivalent of maximizing \( RE(G|s^{(a,b)}_t) - RE(G|S_t) \).

Each supernode \( V_a \) has an attribute vector \( A^a_S \), which records the distribution of attribute values of nodes in \( V_a \). Based on the attribute information, we compute the score of \( V_a \) and \( V_b \), \( score_t^A(a, b) \) as \( \max(A^a_S \oplus A^b_S) \), where \( \oplus \) denotes the element-wise addition of \( A^a_S \) and \( A^b_S \). Using a weight parameter \( \alpha \in [0, 1] \), we assign weight to the attribute and graph structure similarity and define the score of \((V_a, V_b)\) as

\[
score_t(a, b) = \alpha \times score_t^{RE}(a, b) + (1 - \alpha) \times score_t^A(a, b)
\]

\[
= \alpha \left( RE(G|s^{(a,b)}_t) - RE(G|S_t) \right) + (1 - \alpha)\max(A^a_S \oplus A^b_S).
\]  

(3)

We normalize \( RE(G|s^{(a,b)}_t) \) and \( \max(A^a_S \oplus A^b_S) \) by \( n^2 \) and \((n_a + n_b)\), respectively, to bring them in the same scale of \([0, 1]\). To compute \( score_t^{RE}(a, b) \), we need to efficiently evaluate the \( RE \) of \( S_t^{(a,b)} \).

**Lemma 1.** Given a summary \( S_t \) with constant extra space per node, we can

(i) evaluate \( score_t(a, b) \) in \( O(n(t)) \) time

(ii) approximate \( score_t(a, b) \) in constant time and space

Note that the score of a pair of nodes consists of two factors (Equation (3)). We discuss the computation of each factor below; together, they constitute proof of Lemma 1.

**Note 1.** At every supernode \( V_a \), we store integers \( n_a \) and \( e_a \), the number of nodes in \( V_a \), and the number of edges with both endpoints in \( V_a \). Moreover, at \( V_a \), we also store a real number \( D_a = \sum_{i=1}^{n(t)} e^2_{ai}/n_i \), i.e., the sum of squares of weights of superedges incident on \( V_a \). We can update
$D_a$ in constant time after merging any two nodes $V_x, V_y \neq V_a$. After merging, we traverse neighbors list of $V_x$ and $V_y$ for $V_a$, subtract $\varepsilon_{xa/n_x}$ and $\varepsilon_{ya/n_y}$ from $D_a$ and add $(\varepsilon_{xa}^{x} + \varepsilon_{ya}^{y})/n_x + n_y$ to $D_a$.

### 3.1.1 $\text{score}_{RE}(a, b)$ Computation

We derive a closed-form expression to efficiently compute $RE(G|S)$, which is the total error incurred in the estimation of $A$ from $S$ only. We first calculate the contribution of $V_a \in V_S$ in $RE(G|S)$ and then extend it to a general expression that sums the contribution of all the supernodes in $RE(G|S)$. Let $V_a = \{v_{a1}, v_{a2}, \ldots, v_{an_a}\},$ where $v_{aj} \in V, a \leq k, j \leq n_a.$ $V_a$ contributes to all the entries/edges in $A'$, which have one or both the endpoints in $V_a$. We predict the presence of each of the possible internal edge (edge with both the endpoints in $V_a$) as $e_a/\binom{n_a}{2}$. However, there are $e_a$ edges in $V_a$ and thus $2e_a$ corresponding entries in $A$ have value 1. The error at these $2e_a$ entries in $A$ and $A'$ is $2e_a(1 - e_a/\binom{n_a}{2})$. The error at the remaining $2[\binom{n_a}{2} - e_a]$ positions is $2[\binom{n_a}{2} - e_a]/\binom{n_a}{2}$. For a superedge $(V_a, V_b) \in E_S, V_a, V_b \in V_S, a, b \leq k, a \neq b,$ we predict the presence of an edge $(u, v), v \in V_a, u \in V_b, u, v \in V$ as $e_{ab}/n_an_b$. Doing the similar calculation as above, the contribution of $V_a$ in the error at the entries $(u, v), u \in V_a, v \in V_b$ is $e_{ab}(1 - e_{ab}/n_an_b) + (n_an_b - e_{ab})(e_{ab}/n_an_b)$.

Thus, the total error introduced by supernode $V_a$ is $2e_a(1 - e_a/\binom{n_a}{2}) + 2[\binom{n_a}{2} - e_a]/\binom{n_a}{2} + e_{ab}(1 - e_{ab}/n_an_b) + (n_an_b - e_{ab})(e_{ab}/n_an_b)$. Simplifying this, we get that the total error accumulated in reconstructing $A$ using summary $S$ with $k$ supernodes is

$$RE(G|S) = RE(A|A') = \sum_{i=1}^{k} 4e_i - 4e_i^2/\binom{n_i}{2} + \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} 2e_{ij} - 2e_{ij}^2/n_in_j. \tag{4}$$

Using Equation (4), $\text{score}_{i}^{RE}(a, b)$ can be computed using Equation (5).

$$\text{score}_{i}^{RE}(a, b) = RE(G|S_{i-1}) - RE(G|S_{i})$$

$$= 4e_a + 4e_b - \frac{4e_a^2}{\binom{n_a}{2}} - \frac{4e_b^2}{\binom{n_b}{2}} + \sum_{i=1}^{k} 4e_i - 4e_i^2/\binom{n_i}{2}$$

$$+ 2 \left( \frac{2e_{ab}}{n_an_b} \right) + \sum_{i=1}^{k} 4e_{ai} - \frac{4e_{ai}^2}{n_an_i} + \sum_{i=1}^{k} 4e_{bi}$$

$$- \frac{4e_{bi}^2}{n_bn_i} + \sum_{i,j=1, i \neq a, b}^{k} 2e_{ij} - 2e_{ij}^2/n_in_j - 4(e_a + e_b + e_{ab})$$

$$+ \frac{4(e_a + e_b + e_{ab})^2}{\binom{n_a+n_b}{2}} - \sum_{i=1}^{k} 4e_i - 4e_i^2/\binom{n_i}{2}$$

$$- 4 \sum_{i \neq a, b}^{k} \left( e_{ai} + e_{bi} \right) - \frac{(e_{ai} + e_{bi})^2}{n_a + n_b} + \sum_{i,j \neq a, b}^{k} 2e_{ij} - 2e_{ij}^2/n_in_j$$

$$- \frac{4e_a^2}{\binom{n_a}{2}} - \frac{n(t)4e_{ai}^2}{n_an_i} + \frac{4e_{ab}^2}{n_an_b} - \frac{n(t)4e_{bi}^2}{n_bn_i}$$

$$+ \frac{4(e_a + e_b + e_{ab})^2}{\binom{n_a+n_b}{2}} + \frac{4}{n_a + n_b} \sum_{i=1}^{n(t)} \left( \frac{e_{ai}}{n_i} + \frac{e_{bi}}{n_i} + 2e_{ai}e_{bi} \right). \tag{5}$$
Remark 2. Given \( n_a, n_b, e_a, e_b, e_{ab}, D_a \) and \( D_b \), we can compute all the terms of \( \text{score}_{t}^{RE}(a, b) \) in constant time in Equation (5), except for the last summation that requires a traversal of adjacency lists of \( V_a \) and \( V_b \).

The last summation in Equation (5), \( \sum_{i=1}^{n(t)} 2e_a e_b / n_i \), is the inner product of two \( n(t) \) dimensional vectors \( u \) and \( v \), where the \( i \)th coordinate of \( u \) is \( e_i / \sqrt{m} \) (\( v \) is similarly defined). It takes \( O(n(t)) \) space to store these vectors and \( O(n(t)) \) time to compute \( \text{score}_{t}^{RE}(a, b) \). However, we can approximate \( \langle u, v \rangle = u \cdot v \) using count-min sketch [8], which takes parameters \( \epsilon \) and \( \delta \). Let \( w = 1/\epsilon \) and \( d = \log 1/\delta \), a count-min sketch is represented by a \( d \times w \) matrix. Using a randomly chosen hash function \( h_i, 1 \leq i \leq d \) such that \( h_i : \{1 \ldots n(t)\} \rightarrow \{1 \ldots w\} \), \( i \)th entry of \( u \) is hashed to \( \{j, h_i(j)\} \) in the matrix. Let \( \langle \hat{u}, \hat{v} \rangle \) be the estimate using \( h_i \), and set \( \langle \hat{u}, \hat{v} \rangle \rangle_i = \arg \min_i \langle \hat{u}, \hat{v} \rangle_i \).

Theorem 1 (c.f [8] Theorem 2). For \( 0 < \epsilon, \delta < 1 \), let \( \langle \hat{u}, \hat{v} \rangle \rangle_i \) be the estimate for \( \langle u, v \rangle \) using the count-min sketch. Then

\[
- \langle \hat{u}, \hat{v} \rangle \rangle_i \geq \langle u, v \rangle
\]

\[
- \Pr[\langle \hat{u}, \hat{v} \rangle < \langle u, v \rangle + \epsilon \|u\|_1 \|v\|_1) \geq 1 - \delta
\]

The computational cost of \( \langle \hat{u}, \hat{v} \rangle \rangle_i \) is \( O(\epsilon^{-1} \log 1/\delta) \) and updating the sketch after a merge takes \( O(\log 1/\delta) \) time.

Hence, it takes constant time to closely approximate \( \text{score}_{t}^{RE}(a, b) \) for a pair \( (a, b) \) in \( S_{t-1} \). Note that the bounds on time and space complexity, though constants are quite loose in practice.

3.1.2 \( \text{score}_{t}^{R}(a, b) \) Computation. \( \text{score}_{t}^{R}(a, b) \) computation for \( V_a \) and \( V_b \) is equivalent of element-wise addition of \( \mathcal{A}_a^{d} \) and \( \mathcal{A}_b^{d} \), which are the feature vectors containing the count of attribute values of nodes in \( V_a \) and \( V_b \), respectively. \( \text{score}_{t}^{R}(a, b) \) is the maximum value of the resultant feature vector. Note that the length of \( \mathcal{A}_a \) is equal to \( l \), the number of unique values of the attribute. So, the element-wise addition of \( l \)-dimensional feature vectors and then finding the maximum value of the resultant feature vector takes \( O(l) \) time, where \( l \) is a fixed constant (very small in practice).

3.2 T-2: Selection of the Best Pair of Nodes for Merging

As in Reference [20], we select the best pair out of a random sample; however, Reference [20] takes a sample of linear size, which is still computationally prohibitive for large graphs. We select nodes that are more likely to constitute high-scoring pairs. To this end, we define weights of nodes that essentially measure the contribution of a node to the score of the pair made up of this node. The weight, \( w(a) \) of \( V_a \) is given by

\[
w(a) = \begin{cases} 
\frac{-1}{f(a)} & \text{if } f(a) \neq 0 \\
0 & \text{otherwise}
\end{cases}
\]

where \( f(a) = -\frac{4e_a^2}{n_a^2} - \sum_{i=1}^{n(t)} n_a n_i \frac{\Delta a_i^2}{n_a^2} \).

We sample nodes based on their weights, so nodes with higher weights are more likely to be sampled. This implies that pairs formed from these nodes will also have higher scores, i.e., will be of better quality. Using weighted sampling, a sample of size \( O(\log n) \) outperforms a random sample of size \( O(n) \). Recall that \( n_a \) and \( e_a \), are stored at each supernode \( V_a \). Using these variables, we update the weight of a given node in constant time. Let \( W = \sum_{i=1}^{n(t)} w(i) \) denote the aggregate of node weights, the probability of selecting a node \( V_a \) is \( w(a)/W \). In a given iteration, weighted sampling takes linear time but in the case of dynamic graphs, where nodes and edge weights change over time, weighted sampling is a challenging task. As in each iteration of summarization, we merge a
Structure : Node

```
int NodeId   // contains the unique ID of each vertex
double NodeWeight // stores weight of each node
Node *LeftChild  // stores link to left child
Node *RightChild // stores link to right child
Node *ParentNode // stores pointer to parent node
```

ALGORITHM 3: buildSamplingTree\((V, W, start, end)\)

1: if \(V[start] = V[end]\) then
2:   leaf ← makenode()
3:   leaf.NodeWeight ← \(W[start]\)
4:   leaf.NodeId ← \(V[start]\)
5:   return leaf
6: else
7:   mid ← \(\frac{start + end}{2}\)
8:   leftChild ← makenode()
9:   leftChild ← buildSamplingTree\((V, W, start, mid)\)
10:  rightChild ← makenode()
11:  rightChild ← buildSamplingTree\((V, W, mid + 1, end)\)
12:  parent ← makenode()
13:  parent.NodeWeight ← leftChild.NodeWeight + rightChild.NodeWeight
14:  leftChild.ParentNode ← parent
15:  rightChild.ParentNode ← parent
16:  return parent
```

A pair of nodes and as a result of which, the weights of some other nodes also change. To address these issues, we design a data structure \(T\) for weighted sampling with the following properties:

**Lemma 2.** We implement \(T\) as a binary tree with following properties:

(i) initially populating \(T\) with \(n\) nodes can be done in \(O(n)\) time;

(ii) randomly sampling a node with probability proportional to its weight can be done in \(O(\log n)\) time;

(iii) inserting, deleting, and updating weight of a node in \(T\) can be done in \(O(\log n)\) time.

**Data Structure for Sampling:** We implement \(T\) as a balanced binary tree and a leaf node in \(T\) represents a supernode in the graph. A leaf node also contains the weight and id of the supernode. Each internal node stores the sum of the weights of its children and resultantly, the weight of the root sums up to \(\sum_{i=1}^{n} w(i)\). Algorithm 3 presents the construction of \(T\), and we also show the structure of a tree node. The height of \(T\) is \([\log n]\) and it takes \(O(n)\) time to construct the tree. We designed \(T\) independently, but later found out that it has been known to the statistics community since 1980 [41].

A node \(V_a\) is sampled from \(T\) with probability \(\frac{w(a)}{W}\) using Algorithm 4, which takes as input the root node and a uniform random number \(r \in [0, W]\). As \(T\) is a balanced binary tree, the sampling of a node takes \(O(\log n)\) time (length of the path from the root to the node). To update the weight at a leaf, we begin at the leaf (using the stored pointer) and change the weight of that leaf. Following the parent pointers, we update the weights of internal nodes to the new sum of the weights of children.
We do sparsification (deletion of superedges) to reduce the storage size of \( S \). The storage cost of \( S \) with \( k \) supernodes is given as in Reference [19]:

\[
\text{cost}(S) = |E_S| \left( 2 \log_2 k + \log_2 \left( \max(e_{ab}) \right) \right) + n \log_2 k. \tag{7}
\]

Here, \( E_S \) is the set of superedges in \( S \), and \( \max(e_{ab}) \) is the maximum weight of a superedge in \( S \). Note that for any \( S \) with \( k \) supernodes, \( |V| \log_2 k \) will remain fixed, and the storage cost depends on the number of superedges in \( S \). Moreover, each superedge takes a constant number of bits, and the change in the cost after dropping a superedge will remain constant; however, the change in reconstruction error varies for each superedge. The increase in reconstruction error after dropping a superedge \( (V_a, V_b), a, b \leq k, a \neq b \) is \( 2 \left( \frac{e_{ab}}{n_a n_b} - 1 \right) e_{ab} \). To do sparsification, we sort superedges based on the increase in reconstruction error and drop the desired number of edges that result in the minimum increase in reconstruction error. To reduce the storage size of \( S \) to a certain target size, the number of superedges to be dropped is computed as \( \frac{\text{size}(S)-\text{target size}}{2\log_2 k + \log_2 \left( \max(e_{ab}) \right)} \). Sparsifying the summary takes \( O(2k^2 \log k) \) time in the worst case as it sorts the superedges in \( S \) by weight and then selects the desired number of superedges for deletion.

4 SSAG PERFORMANCE ANALYSIS AND SUMMARY-BASED QUERY ANSWERING

In this section, we analyze the time and space complexity of SsAG. We also discuss how to compute answers to common graph analysis queries using only the summary.
ALGORITHM 5: SsAG($G = (V, E, \mathcal{A})$, $k$, $w$, $d$, size$_t$)

1: $S \leftarrow G$
2: $T \leftarrow \text{buildsamplingtree}(V, W, 1, n)$ \hfill \triangleright \text{Lemma 2}
3: while $|V_S| > k$ do
4:  $\text{pairs} \leftarrow \text{getsample}(T, s)$ \hfill \triangleright \text{Uses Equation (3) and Theorem 1}
5:  $\text{pairScores} \leftarrow \text{computeapproxscore}(\text{pairs}, w, d)$
6:  $(x, y) \leftarrow \text{MAX}(\text{pairScores})$ \hfill \triangleright \text{select the pair $(x, y)$ having maximum score}
7:  $\text{MERGE}(x, y)$ \hfill \triangleright \text{Lemma 3}
8:  for each neighbor $u \in \{N(x) \cup N(y)\}$ do
9:     $\text{updateweight}(u, T)$
10: if $\text{size}(S) > \text{size}_t$ then
11:     $\text{delCount} \leftarrow \frac{\text{size}(S) - \text{size}_t}{\frac{\text{size}(S) - \text{size}_t}{2\log_k k + \log_2(\max(e_{ab}))}}$ \hfill \triangleright \text{count of superedges to be deleted from } S
12:     $\text{SPARSMIFY}(S, \text{delCount})$

4.1 Runtime Analysis of SsAG

Algorithm 5 is our main summarization algorithm that takes as input $G$, integer $k$ (target summary size), $s$ (sample size), $w$, $d$ (where $w = 1/\epsilon$ and $d = \log 1/\delta$ are parameters for count-min sketch) and size$_t$, the target storage cost of summary in bits. See also Figure 2 for the flow diagram of SsAG.

As stated earlier, each node $V_a$ has a variable $D_a$, using which we can initialize the weight array in $O(n)$ using Equation (6). By Lemma 2, we populate $T$ in $O(n)$ time and sample $s$ nodes from $T$ in $O(s \log n)$ time (Line 3). Score approximation of a pair of nodes takes constant time by Lemma 1, and it takes $O(\Delta)$ time to merge a pair by Lemma 3 (Line 6), where $\Delta$ is the maximum degree in $G$. As updating and deleting the weights in $T$ takes $O(\log n)$ time and the while loop makes $n - k + 1$ iterations, the runtime to construct $S$ on $k$ nodes is $O((n - k + 1)(s \log n + \Delta \log n))$. Typically, $k$ is a fraction of $n$, $k \in O(n)$ and we consider $s$ to be $O(\log n)$ or $O(\log^2 n)$. Generally, real-world graphs are very sparse ($\Delta$, the worst-case upper bound, is constant). As mentioned earlier, sparsification takes $O(k^2 \log k)$ time, however, $k$ is a small fraction of $n$ and generally, $k^2 < n$. The overall complexity of SsAG is $O(n \log^2 n)$ or $O(n \log^3 n)$ for sample size of $O(\log n)$ or $O(\log^2 n)$, respectively.

4.2 Space Complexity of Storing the Summary

We give the space complexity of storing the summary $S$. Note that $S$ has $k$ supernodes and weights on both its nodes and edges. First, we need to store the mapping of $n$ nodes in $G$ to the $k$ supernodes in $S$. This is essentially a function that maps each node in $G$ to one of the possible $k$ locations. This
mapping takes $O(n \log k)$ bits. For each supernode $V_i$, we maintain two extra variables $n_i$ and $e_i$. The cost to store these variables at nodes is $O(k \log n_i + k \log \binom{n_i}{2}) = O(k \log n)$ bits. In addition to this, weights at superedges are also stored, which takes $O(k^2 \log n)$ bits in total. So, the overall space complexity of storing $S$ is $O(n \log k + k \log n + k^2 \log n)$.

### 4.3 Summary-based Query Answering

A measure to assess the quality of a summary $S$ is the accuracy in answering queries about $G$ using only $S$. We give expressions to answer queries using $S$ efficiently. We consider widely used queries at different granularity levels [20, 31]. The node-level queries include degree, eigenvector-centrality, and attribute of a node. Degree and eigenvector-centrality query seek the number of edges incident on the query node and the node’s relative importance in the graph. The attribute query answers the attribute value of the query node. We also discuss the adjacency query that asks whether an edge exists between the given pair of nodes. Finally, we compute the triangle density query, a graph-level query, which gives the fraction of triangles in the graph.

**Node Degree Query** Given a node $u \in V$, degree of $u$ can be estimated using only $S$ as $deg'(u) = \sum_{i,j=1}^{n} A'(u,j)$. We can compute $deg'(v)$ in $O(k)$ time using the extra information stored at supernodes as $deg'(v) = \frac{1}{n_i} (2e_i + \sum_{j=1,j \neq i}^{k} e_i)$.

**Node Eigenvector-centrality Query** Based on $S$, the estimate for eigenvector-centrality of a node $v$ is $p'(v) = \frac{deg'(v)}{|E|} \ast [20]$. As we compute $deg'(v)$ in $O(k)$ time, we can compute eigenvector-centrality in $O(k)$ time.

**Node Attribute Query** For a node $u$, the attribute query asks for the value of $A(u)$. From $S$, we answer this query approximately as follows. For $u \in V_i$, the approximate answer $A'(u)$ is $a_u$ with probability $A'_i[u] / n_i$.

**Adjacency Query** Given two nodes $u, v \in V$, using $S$ the estimated answer to the query whether $(u, v) \in E$ is $A'(u, v)$.

**Graph Triangle Density Query** Let $t(G)$ be the number of triangles in $G$. $t(G)$ can be estimated from $S$ by counting (i) the expected number of triangles in each supernode, (ii) the expected number of triangles with one vertex in a supernode and the remaining two vertices in another supernode, (iii) the expected number of triangles with the three vertices in three different supernodes. Let $\pi_i = e_i/\binom{n_i}{3}$ and $\pi_{ij} = e_{ij}/n_in_j$, then the estimate for $t(G)$, is

$$t'(G) = \sum_{i=1}^{k} \left( \binom{n_i}{3} \pi_i^3 + \sum_{j=i+1}^{k} \binom{n_i}{2} \pi_{ij} \left( \binom{n_j}{2} n_j \pi_j \right) + \binom{n_j}{2} n_i \pi_j \right) + \sum_{i=j+1}^{k} n_i n_j n_i n_j \pi_{ij} \pi_{ij} \pi_{ij} \pi_{ij} \right).$$

Let $T^a_i$, $T^b_i$, $T^c_i$, and $T^d_i$ be the number of triangles made with three vertices inside $V_i$, with one vertex in $V_i$ and two in another $V_j$, with two vertices in $V_i$ and one in another $V_j$, and with one vertex in $V_i$ and two vertices in two distinct other supernodes $V_j$ and $V_l$, respectively. The number of each type of these triangles is given as:

$$T^a_i = \binom{n_i}{3} \pi_i^3, \quad T^b_i = \sum_{j=1,j \neq i}^{k} n_i \binom{n_j}{2} \pi_{ij}^2 \pi_j,$$

ACM Trans. Knowl. Discov. Data., Vol. 18, No. 6, Article 141. Publication date: April 2024.
\[ T_i^c = \sum_{j=1 \atop j \neq i}^k \binom{n_i}{2} n_j \pi_i^2 \pi_{ij} \quad \text{and} \]
\[ T_i^d = \sum_{j=1 \atop j \neq i}^k \sum_{l=1 \atop l \neq i}^k n_i n_j n_l \pi_{ij} \pi_{jl} \pi_{il}. \]

Keeping in view the over-counting of each type of triangle, our estimate for \( t(G) \) is
\[ \sum_{i=1}^k T_i^a + T_i^b/2 + T_i^c/2 + T_i^d/3. \]

Computing this estimate takes \( O(k^2) \) time.

Overall, in terms of the properties preservation, SsAG preserves crucial properties such as graph topology and node attributes and aims to minimize the reconstruction error while maintaining the homogeneity of attributes within supernodes. However, it may not capture fine-grained node embeddings or exact attribute values due to its focus on creating a concise and interpretable summary. The tradeoff allows SsAG to efficiently handle large-scale graphs and produce summaries that balance structural and attribute information.

5 EXPERIMENTAL SETUP

This section describes the dataset, implementation details, evaluation metrics, and parameter values used for experimentation. We also describe the setup briefly for results comparison with baseline and SOTA methods. We perform our experiments on an Intel(R) Core i5 with 2.4 GHz processor and 8 GB memory using Java programming language. The experiments were performed on the Windows 10 operating system. The compiler used for the programming is Netbeans. We made the code available online for reproducibility.

5.1 Dataset Description and Statistics

We perform experiments on real-world benchmark graphs. We treat all the graphs as undirected and unweighted. The order of the graphs ranges from a few thousand to a few million. The brief statistics and description of the graphs used in the experiments are given in Table 1.

5.2 Evaluation and Comparison Setup

We compare SsAG with GraSS [20] and s2L [31] in terms of computational efficiency and quality of summary. Furthermore, we make a comparison with SsumM [19] to show the impact of sparsification.

5.2.1 GraSS [20]. GraSS generates a summary with the goal to minimize the \( l_1 \)-reconstruction error (RE). GraSS uses the agglomerative approach to generate a sequence of summaries on \( i \) supernodes for \( k \leq i \leq n \) of a graph on \( n \) nodes. The total worst-case running time of GraSS is \( O(n^3) \). Thus, it is feasible for graphs with a few thousand nodes. We compare the runtime and RE of SsAG with GraSS only the Facebook dataset.

5.2.2 s2L [31]. s2L uses the Euclidean-space clustering algorithm to generate a summary with \( k \) supernodes. s2L minimizes the reconstruction error and evaluates summaries also on cutnorm error, storage cost, and accuracy in query answers. We compare our summaries with s2L on all

1https://www.dropbox.com/sh/gh05kux3x04qp4r/AAAhVOf-RGdWFA2yv11V7Mzia?dl=0
Table 1. Statistics and Type of Datasets Used in Experiments

| Name            | \( |V| \) | \( |E| \) | Network Description                                                                 |
|-----------------|--------|--------|----------------------------------------------------------------------------------|
| Caltech36 [32]  | 769    | 16,656 | Facebook100 graph induced by Caltech users Nodes: Users, Edges: Friendships Node attributes: Gender, High school status |
| Political Blogs [18]| 1,224  | 16,718 | Hyperlink Network Nodes: Political blogs, Edges: Hyperlinks Nodes attribute: Political view (liberal/conserv.) |
| Facebook [21]   | 4,039  | 88,234 | Anonymized subgraph of Facebook Nodes: Facebook Users, Edges: Friendships |
| Email [21]      | 36,692 | 183,831| Online Social Network Nodes: Email addresses, Edges: Email exchange |
| Stanford [21]   | 281,903| 1,992,636| Web Network Nodes: Stanford webpages, Edges: Hyperlinks |
| Amazon [21]     | 403,394| 2,443,408| Co-purchasing network Nodes: Products, Edges: Co-purchased products |
| YouTube [21]    | 1,157,828| 2,987,624| Online Social Network Nodes: YouTube users, Edges: Friendships |
| Skitter [21]    | 1,696,415| 11,095,298| Internet Topology Nodes: Autonomous systems, Edges: Links |
| Wiki-Talk [21]  | 2,394,385| 4,659,565| Online Social Network Nodes: Wikipedia users, Edges: Edits on pages |

these four measures. We also compare the runtimes of s2l and SsAG. In comparison with s2l, we report results for summary graphs with \( k \in \{100, 500\} \) supernodes for smaller datasets. For larger graphs (on more than 100,000 nodes), we make summaries on \( k \in \{1000, 2000\} \) supernodes. These numbers are chosen without any quality bias. We report results for SsAG with the sample size parameter \( s = 5 \log n(t) \), where \( n(t) \) is the number of supernodes in the summary. Note that increasing \( s \) improves the quality of the summary but increases the computational cost. To evaluate the impact of sample size on the performance of SsAG, we also report its results with \( s \in \{\log n(t), 5 \log n(t), \log^2 n(t)\} \). We compute the approximate score using the count-min sketch approach with width \( w = 200 \).

5.2.3 SSumM [19]. Given the target summary size in bits (rather than a target number of supernodes), SSumM merges pairs of nodes that minimize the RE and result in a sparse summary. SSumM scores pairs based on both these objectives. The score function can be efficiently evaluated and is the building block of the greedy summarization strategy. Since the sparsity is an input parameter, we compare the RE in summaries of a fixed target size obtained using SSumM and SsAG.

5.2.4 Purity of Summaries of Attributed Graphs. Since SsAG incorporates attribute information along with the graph structure, we summarize widely used attributed graphs using SsAG. We demonstrate the tradeoff between purity and reconstruction error using the user-set parameter \( \alpha \). Note that in the current work, we only consider nominal attributes at nodes. SsAG can readily be extended to incorporate numeric attributes.

5.2.5 Scalability of SsAG. We show that SsAG is scalable to larger real-world graphs on which the competitor methods can not be applied. We ran SsAG on graphs on more than 1 million nodes and reported RE and runtime for varying values of parameters.

6 RESULTS AND DISCUSSION
This section reports the results of the proposed model and its comparison with the baseline and SOTA methods. We first compare the results of SsAG with the baseline method, GRASS. Second, we give a detailed report on how SsAG performs compared to s2l by the evaluation metrics mentioned above. We also evaluate the sparsification phase in SsAG and demonstrate the effect of incorporat-
Fig. 3. Reconstruction error of SsAG and GraSS for varying $k$ on Facebook graph. **Runtime to build a summary with $k = 500$ of SsAG and GraSS is 0.33 and 11.16 seconds, respectively.**

ing node attributes on reconstruction error. Finally, we demonstrate that SsAG is scalable to very large graphs. In all experiments, we set the (sketch depth) parameter $d = 2$ in Algorithm 5.

### 6.1 Comparison with GraSS

We compare the normalized RE (scaled by $n \times n$, the number of entries in the adjacency matrix) with GraSS that only works for graphs with a few thousand nodes. We show comparison only on the Facebook graph (4,039 vertices). Figure 3 depicts that RE increases with decreasing number of supernodes in summary ($k$) for both SsAG and GraSS, with SsAG slightly outperforming GraSS. However, SsAG is more than 33× faster than GraSS, as it took 0.33 seconds to build the summary, while GraSS took 11.16 seconds. These results are for sample size $s = 5 \log n(t)$ in SsAG. For other values of $s$, SsAG exhibits the expected trend; with increasing sample size, RE decreases at the cost of running time (see Section 6.3).

### 6.2 Comparison with s2l

In this section, we report the results of an extensive comparison of SsAG and s2l based on RE, cut-norm Error, Storage Cost (KB), Average Degree Error, Triangle Density Error, and Computation Time. Table 2 shows the results for 4 datasets, namely, Facebook, Email, Stanford, and Amazon. To show the approximation quality of score computation, we report results for exact score computation using Equation (5) and Theorem 1. We used $w = \frac{1}{\epsilon} = 200$; other values show a similar trend. Results show that, especially for large graphs, the count-min sketch closely approximates the exact score of a pair for merging. The rows corresponding to SsAG with exact score computation are listed with $w = \Box$. These results are for sample size $s = 5 \log n(t)$ in SsAG. We demonstrate the effect of different sample sizes on RE and runtime in Section 6.3. We also provide % improvement for SsAG (with $w = \Box$) from s2l for each metric using the expression:

$$\% \text{ increase} = \frac{\text{SsAG value} - \text{s2l value}}{\text{s2l value}} \times 100$$

**Runtime** Computational efficiency is the most salient feature and prominent achievement of SsAG. Table 2 shows that the runtime of SsAG is significantly smaller in all settings compared to s2l but results in a slightly larger reconstruction error. As expected, estimating pairs’ scores with sketch takes less time at the cost of a negligible increase in RE. Observe that on the Email dataset, SsAG is $\sim 17\times$ faster than s2l, while RE of SsAG is only 0.0026% worse than that of s2l. Similarly, for Stanford dataset, SsAG is $\sim 5\times$ more efficient while the difference in error is only $2 \times 10^{-5}$.

**Cutnorm Error** The cutnorm error in the summaries produced by SsAG is less than those by s2l for all graphs except for the small Facebook graph (when $k = 100$). However, cutnorm error for
Table 2. Comparison of SsAG with s2l on Different Evaluation Metrics

| G | k | Method | w | RE Cutnorm Error | Storage Cost (KB) | Avg. Degree Error | Triangle Density Error | Time (sec) |
|---|---|--------|---|------------------|------------------|------------------|------------------------|------------|
| Facebook | 100 | SsAG | 200 | 1.65E-2 | 3.64E-3 | 7.86 | 16.74±21 | 0.52 | 0.19 |
| | | s2l | N/A | 1.62E-2 | 4.10E-3 | 8.65 | 17.18±20 | 0.49 | 0.32 |
| | | | | 1.60E-2 | 3.05E-3 | 6.57 | 9.89±12 | 0.30 | 1.45 |
| | | | | -52.83 | -34.43 | -31.66 | -73.71 | 63.33 | 77.93 |
| | 500 | SsAG | 200 | 1.35E-2 | 3.60E-3 | 62.5 | 11.99±15 | 0.30 | 0.22 |
| | | s2l | N/A | 1.32E-2 | 2.48E-3 | 81.00 | 11.22±12 | 0.28 | 0.37 |
| | | | | 8.61E-3 | 2.85E-3 | 39.71 | 7.21±8 | -0.32 | 4.68 |
| | | | | -52.83 | -34.43 | -31.66 | -73.71 | 63.33 | 77.93 |
| Email | 100 | SsAG | 200 | 5.28E-4 | 1.91E-4 | 44.34 | 6.79±25 | 0.80 | 1.96 |
| | | s2l | N/A | 5.26E-4 | 1.39E-4 | 46.45 | 6.31±14 | -0.76 | 2.57 |
| | | | | 5.28E-4 | 2.40E-4 | 37.99 | 6.12±16 | -0.77 | 45.94 |
| | | | | -52.83 | -34.43 | -31.66 | -73.71 | 63.33 | 77.93 |
| | 500 | SsAG | 200 | 5.18E-4 | 1.38E-4 | 153.36 | 5.71±19 | -0.65 | 2.79 |
| | | s2l | N/A | 5.18E-4 | 2.21E-4 | 158.74 | 5.15±10 | -0.63 | 4.1 |
| | | | | 4.89E-4 | 2.12E-4 | 112.34 | 4.79±12 | -0.73 | 55.4 |
| | | | | -52.83 | -34.43 | -31.66 | -73.71 | 63.33 | 77.93 |
| Stanford | 1,000 | SsAG | 200 | 7.16E-5 | 3.49E-5 | 860.11 | 7.69±41 | -0.68 | 89.43 |
| | | s2l | N/A | 7.10E-5 | 3.77E-5 | 876.4 | 8.06±42 | -0.67 | 108.78 |
| | | | | 5.37E-5 | 6.30E-5 | 374.04 | 5.11±36 | -0.26 | 305.19 |
| | | | | -32.22 | -40.1 | -134.31 | 57.73 | 157.69 | 64.36 |
| | 2,000 | SsAG | 200 | 6.91E-5 | 3.16E-5 | 1417.6 | 7.38±40 | -0.64 | 80.6 |
| | | s2l | N/A | 6.87E-5 | 5.18E-5 | 1462.5 | 7.60±38 | -0.63 | 109.36 |
| | | | | 4.56E-5 | 6.80E-5 | 449.29 | 4.06±10 | -0.23 | 425.95 |
| | | | | -47.74 | -23.82 | -225.51 | 87.19 | 173.91 | 74.33 |
| Amazon | 1,000 | SsAG | 200 | 5.98E-5 | 3.04E-5 | 537.66 | 5.63±13 | -1.00 | 496.17 |
| | | s2l | N/A | 5.97E-5 | 3.77E-5 | 537.29 | 5.64±13 | -1.00 | 632.35 |
| | | | | 5.98E-5 | 3.04E-5 | 537.66 | 5.63±13 | -1.00 | 496.17 |
| | | | | -47.74 | -23.82 | -225.51 | 87.19 | 173.91 | 74.33 |
| | 2,000 | SsAG | 200 | 5.96E-5 | 2.84E-5 | 648.02 | 5.57±12 | -0.99 | 793.57 |
| | | s2l | N/A | 5.95E-5 | 2.88E-5 | 638.55 | 5.60±13 | -0.99 | 1,096.01 |
| | | | | 5.81E-5 | 3.80E-5 | 584.29 | 5.13±9 | -0.92 | 1,096.01 |
| | | | | -2.41 | 24.21 | -9.29 | -9.16 | -7.61 | 1.77 |

Observe that SsAG is better by Cutnorm error than s2l for all but the smallest dataset (Facebook with k = 100). Moreover, with a slight degradation in performance SsAG significantly outperforms s2l by the runtime. Sample size s = 5 log n(t) and scores of pairs are computed exactly (w = □) and approximated with count-min sketch width w = 200. The best values are shown in bold. We also show percentage (%) improvement of SsAG (w = □) from s2l using Equation (8).

SsAG is better for the Facebook dataset when we have k = 500. We observe the trend that SsAG substantially outperforms s2l for small k. We achieved up to 42% improvement in cutnorm error compared to s2l (for Email dataset when w = □).

Storage Cost Table 2 gives the space consumption of summaries produced by SsAG and s2l. The summaries produced by SsAG have higher storage cost compared to those made by s2l. Considering the storage size of the summary in terms of the percentage of the size of the original graph, the difference in summary sizes of the two approaches is not very significant. For the Amazon dataset, the summaries on 1,000 supernodes by s2l and SsAG take 4.58% and 4.83% space of the original...
Fig. 4. Impact of sample size on RE (subfigures (a) and (c)) and runtime (subfigures (b) and (d)), as shown in their y-axis for Facebook and Stanford datasets. We take sample size \( s \in \{ \log n(t), 5 \log n(t), \log^2 n(t), \sqrt{n(t)} \} \), where \( n(t) \) is the number of supernodes in the summary. Increasing \( s \) results in a significant increase in runtime. However, there is relatively less reduction in RE.

**6.3 Impact of Sample Size on RE and Runtime**

We show the impact of sample size on RE and the time taken to compute the summary. We evaluate SsAG with \( s \in \{ \log n(t), 5 \log n(t), \sqrt{n(t)} \} \) and \( w = 200 \) to show the impact of \( s \) on the quality of the summary and runtime. In Figure 4, it is evident that the error decreases with the increase in \( s \). However, the benefit in quality is not proportional to the increase in computational cost. We show the results for Facebook and Stanford datasets; other datasets show the same trend regarding the quality of summary and computational time with varying \( s \).

**6.4 Sparsification and Comparison with SSumM**

In this section, we report RE of summaries of a fixed size obtained by SsAG and SSumM. Figure 5 shows the RE of summaries with varying relative sizes \( \frac{\text{summary size}}{\text{original graph size}} \) of three graphs. For the Stanford graph, after sparsification, the summary size is reduced to 5% of the original graph size and achieves RE of \( 4.5 \times 10^{-5} \). This is less than the RE of the summary of the corresponding size by s2l. Compared to SSumM, summaries obtained using SsAG are similar in quality. The difference in RE of summaries is not very significant (the y-axis represents values at \( 10^{-5} \) and \( 10^{-4} \) scale). For the Stanford graph, the maximum absolute difference in RE of SsAG and SSumM at a relative size of 0.4 is 0.003%. On other graphs, the difference in RE is significantly smaller.

**6.5 Summarization of Attributed Graphs: RE vs. Purity**

SsAG also incorporates node attributes for summarization along with the graph structure. We report the purity of summaries obtained by SsAG using \( \alpha \in [0, 1] \) as a weight parameter for the
Fig. 5. Reconstruction error of SsAG and SSumM at varying relative sizes (summary size / original graph size) of summary graphs on three datasets. The results are reported on a scale of $10^{-5}$ and $10^{-4}$ and the max. absolute difference in RE of SsAG and SSumM (Stanford dataset) at relative size $= 0.4$ is 0.003%.

Fig. 6. Summarization of attributed graphs Caltech36 and Political Blogs, where nodes in Caltech36 have two attributes (status and gender) while Political blogs have one attribute (affiliation). The left y-axis shows reconstruction error while the right y-axis shows the purity of summary with $k$ supernodes. Results are reported with $\alpha \in \{0, 0.5, 1\}$; $\alpha = 0.5$ gives equal weight to the graph topology and node attributes.

graph topology and attribute information ($\alpha = 1$ means summarization only based on graph structure while $\alpha = 0$ mean summarization only based on attributes). We report the results in Figure 6 for $\alpha \in \{0, 0.5, 1\}$ that show that with increasing weight of attributes, SsAG yields summaries with high purity and a minimal increase in RE.

6.6 Scalability of SsAG
To demonstrate the scalability of SsAG, we run SsAG on graphs on more than 1 million nodes. Table 3 reports quality and runtime only for SsAG on large graphs, since s2l is not applicable to graphs at this scale. SsAG can compute the summaries of large datasets in a few minutes while s2l fails to compute the summaries on these datasets even after running for a whole day. These results are for sample size $s = 5 \log n(t)$. Note that increasing the value of $w$ does not improve the quality of summaries of large graphs.

7 CONCLUSION
We propose SsAG, a sampling-based efficient approximate method that considers both the graph structure and node attribute information for summarization. We build the summary by iteratively merging pairs of nodes. The pair is selected based on the score quantifying the reconstruction error resulting after merging it. We approximate the score in constant time with theoretical
Table 3. Reconstruction Error (RE) and Runtime (Seconds) to Compute Summaries of Large Graphs Using SsAG on Different Datasets

|          | Skitter | Wiki-Talk | YouTube |
|----------|---------|-----------|---------|
|          | $|V| = 1,696,415$ | $|V| = 2,394,385$ | $|V| = 1,157,828$ |
|          | $|E| = 11,095,298$ | $|E| = 4,659,565$ | $|E| = 2,987,624$ |
| $k \times 10^3$ | $w$ | RE | Time(s) | RE | Time(s) | RE | Time(s) |
|----------|-------|-----|--------|-----|--------|-----|--------|
| 10       | 50    | 2.29E-6  | 521.43 | 1.56E-6 | 311.10 | 3.34E-6 | 207.38 |
|          | 100   | 2.23E-6  | 516.03 | 1.51E-6 | 328.19 | 3.22E-6 | 222.22 |
|          | □     | 2.14E-6  | 649.82 | 1.44E-6 | 319.95 | 3.09E-6 | 242.58 |
| 50       | 50    | 2.11E-6  | 481.40 | 1.23E-6 | 285.89 | 2.50E-6 | 184.67 |
|          | 100   | 1.96E-6  | 480.85 | 1.21E-6 | 299.98 | 2.38E-6 | 195.85 |
|          | □     | 1.88E-6  | 524.94 | 1.20E-6 | 329.39 | 2.37E-6 | 215.87 |
| 100      | 50    | 1.91E-6  | 436.84 | 1.09E-6 | 266.15 | 1.97E-6 | 160.11 |
|          | 100   | 1.70E-6  | 445.27 | 1.09E-6 | 276.32 | 1.94E-6 | 167.67 |
|          | □     | 1.66E-6  | 486.88 | 1.09E-6 | 303.44 | 1.94E-6 | 183.64 |
| 250      | 50    | 1.38E-6  | 332.27 | 9.07E-7 | 223.65 | 1.30E-6 | 103.25 |
|          | 100   | 1.24E-6  | 350.47 | 9.05E-7 | 232.39 | 1.30E-6 | 107.79 |
|          | □     | 1.23E-6  | 376.89 | 9.03E-7 | 256.05 | 1.30E-6 | 118.73 |

The approximate score is computed using count-min sketch width $w \in \{50, 100, 200\}$ and exact score computation ($w = \Box$).

guarantees using the closed-form expression. Experimental results on several benchmark datasets show that our technique is comparable in quality and performs better than competitor methods in efficiency and runtime. Moreover, SsAG also incorporates attributed graphs and produces highly homogeneous summaries. Furthermore, our sparsification method greatly reduces the size of the summary graph without significantly impacting the reconstruction error. In the future, we hope to evaluate SsAG on edge attributed graphs for experiments, along with more real-world graphs having node attributes.

REFERENCES

[1] Muhammad Ahmad, Juvaria Tariq, Mudassir Shabbir, and Imdadullah Khan. 2017. Spectral methods for immunization of large networks. Australas. J. Inf. Syst. 21 (2017), 1–18.

[2] Sarwan Ali, Muhammad Shakeel, Imdadullah Khan, Safiullah Faizullah, and Muhammad Khan. 2021. Predicting attributes of nodes using network structure. ACM Trans. Intell. Syst. Technol. 12, 2 (2021), 1–23.

[3] Maham Beg, Muhammad Ahmad, Arif Zaman, and Imdadullah Khan. 2018. Scalable approximation algorithm for graph summarization. In Pacific-Asia Conference on Knowledge Discovery and Data Mining. 502–514.

[4] Kamal Berahmand, Yuefeng Li, and Yue Xu. 2023. DAC-HPP: Deep attributed clustering with high-order proximity preserve. Neural Comput. Appl. 35, 34 (2023), 24493–24511.

[5] Kamal Berahmand, Mehrnoush Mohammadi, Farid Saberi-Movahed, Yuefeng Li, and Yue Xu. 2022. Graph regularized nonnegative matrix factorization for community detection in attributed networks. IEEE Trans. Netw. Sci. Eng. 10, 1 (2022), 372–385.

[6] Alexandru Boicea, Ciprian-Octavian Truică, Florin Rădulescu, and Elena-Cristina Buşe. 2018. Sampling strategies for extracting information from large data sets. Data Knowl. Eng. 115 (2018), 1–15.

[7] Chonghao Chen, Fei Cai, Xuejun Hu, Wanyu Chen, and Honghui Chen. 2021. HHGN: A hierarchical reasoning-based heterogeneous graph neural network for fact verification. Inf. Process. Manag. 58, 5 (2021), 102659:1–102659:14.
[8] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: The count-min sketch and its applications. *J. Algor.* 55, 1 (2005), 58–75.

[9] Yingan Cui, Xue Li, Junhui Li, Huajun Wang, and Xiaogang Chen. 2022. A survey of sampling method for social media embeddedness relationship. *Comput. Surv.* 55, 4 (2022), 1–39.

[10] Zohair Hassan, Madassir Shabbir, Imadullah Khan, and Waseem Abbas. 2020. Estimating descriptors for large graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 779–791.

[11] Arijit Khan and Charu Aggarwal. 2017. Toward query-friendly compression of rapid graph streams. *Soc. Netw. Anal. Min.* 7, 1 (2017), 23:1–23:19.

[12] Abeer Khan, Lukasz Golab, Mehdi Kargar, Jaroslaw Szlichta, and Morteza Zibohayat. 2020. Compact graph discovery in attributed graphs and social networks. *Inf. Process. Manag.* 57, 2 (2020), 102054.

[13] Kifayat-Ullah Khan, Batjargal Dolgorsuren, Nguyen Tu, Waqas Nawaz, and Young-Koo Lee. 2017. Faster compression methods for a weighted graph using locality sensitive hashing. *Inf. Sci.* 421 (2017), 257–253.

[14] Kifayat-Ullah Khan, Waqas Nawaz, and Young-Koo Lee. 2015. Set-based approximate approach for lossless graph summarization. *Computing* 97, 12 (2015), 1185–1207.

[15] Kifayat-Ullah Khan, Waqas Nawaz, and Young-Koo Lee. 2017. Set-based unified approach for summarization of a multi-attributed graph. *World Wide Web* 20, 3 (2017), 543–570.

[16] Jihoon Ko, Yunbum Kook, and Kijung Shin. 2020. Incremental lossless graph summarization. In *International Conference on Knowledge Discovery and Data Mining*, 317–327.

[17] Danai Koutra, U. Kang, Jilles Vreeken, and Christos Faloutsos. 2014. VOG: Summarizing and understanding large graphs. In *SIAM International Conference on Data Mining*, 91–99.

[18] Jérôme Kunegis. 2013. Konect: The Koblenz network collection. In *International Conference on Data Mining*, 1343–1350.

[19] Kyuhan Lee, Hyeonsoo Jo, Jihoon Ko, Sungsu Lim, and Kijung Shin. 2020. SSumM: Sparse summarization of massive graphs. In *International Conference on Knowledge Discovery and Data Mining*, 144–154.

[20] Kristen LeFevre and Evimaria Terzi. 2010. GraSS: Graph structure summarization. In *SIAM International Conference on Data Mining*, 454–465.

[21] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. Retrieved from http://snap.stanford.edu/data

[22] Yongsub Lim, U. Kang, and Christos Faloutsos. 2014. SlashBurn: Graph compression and mining beyond caveman communities. *IEEE Trans. Knowl. Data Eng.* 26 (2014), 3077–3089.

[23] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *Comput. Surv.* 51, 3 (2018), 62:1–62:34.

[24] Zheng Liu, Jeffrey Xu Yu, and Hong Cheng. 2012. Approximate homogeneous graph summarization. *J. Inf. Process.* 20, 1 (2012), 77–88.

[25] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. 2008. Graph summarization with bounded error. In *International Conference on Management of Data*, 419–432.

[26] Pinelopi Papalampidi, Frank Keller, and Mirella Lapata. 2021. Movie summarization via sparse graph construction. In *Association for the Advancement of Artificial Intelligence*.

[27] B. Prakash, Jilles Vreeken, and Christos Faloutsos. 2012. Spotting culprits in epidemics: How many and which ones? In *International Conference on Data Mining*, 11–20.

[28] Qiang Qu, Siyuan Liu, Feida Zhu, and Christian Jensen. 2016. Efficient online summarization of large-scale dynamic networks. *IEEE Trans. Knowl. Data Eng.* 28, 12 (2016), 3231–3245.

[29] Alireza Rezvanian and Mohammad Reza Meybodi. 2016. Sampling algorithms for weighted networks. *Soc. Netw. Anal. Min.* 6 (2016), 1–22.

[30] Alireza Rezvanian and Mohammad Reza Meybodi. 2017. Sampling algorithms for stochastic graphs: A learning automata approach. *Knowl.-based Syst.* 127 (2017), 126–144.

[31] Matteo Riondato, David Garcia-Soriano, and Francesco Bonchi. 2017. Graph summarization with quality guarantees. *Data Min. Knowl. Discov.* 31, 2 (2017), 314–349.

[32] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *AAAI*. Retrieved from https://networkrepository.com

[33] Tara Safavi, Caleb Belth, Lukas Faber, Davide Mottin, Emmanuel Müller, and Danai Koutra. 2019. Personalized knowledge graph summarization: From the cloud to your pocket. In *IEEE International Conference on Data Mining*, 528–537.

[34] Hojin Seo, Kisung Park, Yongkoo Han, Hyunwook Kim, Muhammad Umair, Kifayat Ullah Khan, and Young-Koo Lee. 2020. An effective graph summarization and compression technique for a large-scaled graph. *J. Supercomput.* 76 (2020), 7906–7920.

[35] Lei Shi, Hanghang Tong, Jie Tang, and Chuan Lin. 2015. VEGAS: Visual influence graph summarization on citation networks. *IEEE Trans. Knowl. Data Eng.* 27, 12 (2015), 3417–3431.
[36] Kijung Shin, Amol Ghoting, Myunghwan Kim, and Hema Raghavan. 2019. SWeG: Lossless and lossy summarization of web-scale graphs. In International Conference on World Wide Web. 1679–1690.

[37] Chenguang Song, Kai Shu, and Bin Wu. 2021. Temporally evolving graph neural network for fake news detection. Inf. Process. Manag. 58, 6 (2021), 102712:1–102712:18.

[38] Nan Tang, Qing Chen, and Prasenjit Mitra. 2016. Graph stream summarization: From big bang to big crunch. In International Conference on Management of Data. 1481–1496.

[39] Yuanyuan Tian, Richard Hankins, and Jignesh Patel. 2008. Efficient aggregation for graph summarization. In International Conference on Management of Data. 567–580.

[40] Ioanna Tsalouchidou, Gianmarco Morales, Francesco Bonchi, and Ricardo Baeza-Yates. 2016. Scalable dynamic graph summarization. In International Conference on Big Data. 1032–1039.

[41] C. Wong and Malcolm Easton. 1980. An efficient method for weighted sampling without replacement. SIAM J. Comput. 9, 1 (1980), 111–113.

[42] Jinguo You, Quping Pan, Wei Shi, Zhipeng Zhang, and Jianhua Hu. 2013. Towards graph summary and aggregation: A survey. In Social Media Retrieval and Mining. Springer.

[43] Jinbo Zhang, Sourav S. Bhowmick, Hong H. Nguyen, Byron Choi, and Feida Zhu. 2015. DaVinci: Data-driven visual interface construction for subgraph search in graph databases. In International Conference on Data Engineering. 1500–1503.

[44] Fang Zhou, Qiang Qu, and Hannu Toivonen. 2017. Summarisation of weighted networks. J. Experim. Theoret. Artif. Intell. 29, 5 (2017), 1023–1052.

Received 4 July 2023; revised 3 January 2024; accepted 3 March 2024