Personalized Graph Summarization: Formulation, Scalable Algorithms, and Applications

Shinhwan Kang, Kyuhan Lee, and Kijung Shin
Kim Jaechul Graduate School of AI, KAIST, Seoul, South Korea
{shinhwan.kang, kyuhan.lee, kijungs}@kaist.ac.kr

"Everything is related to everything else, but near things are more related than distant things" - **The First Law of Geography** (Tobler 1970).

***Abstract*—Are users of an online social network interested equally in all connections in the network? If not, how can we obtain a summary of the network personalized to specific users? Can we use the summary for approximate query answering?**

As massive graphs (e.g., online social networks, hyperlink networks, and road networks) have become pervasive, graph compression has gained importance for the efficient processing of such graphs with limited resources. Graph summarization is an extensively-studied lossy compression method. It provides a summary graph where nodes with similar connectivity are merged into supernodes, and a variety of graph queries can be answered approximately from the summary graph.

In this work, we introduce a new problem, namely *personalized graph summarization*, where the objective is to obtain a summary graph where more emphasis is put on connections closer to a given set of target nodes. Then, we propose PEGASUS, a linear-time algorithm for the problem. Through experiments on six real-world graphs, we demonstrate that PEGASUS is (a) *Effective:* node-similarity queries for target nodes can be answered significantly more accurately from personalized summary graphs than from non-personalized ones of similar size, (b) *Scalable:* it summarizes graphs with up to one billion edges, and (c) *Applicable to distributed multi-query answering*: it successfully replaces graph partitioning for communication-free multi-query processing.

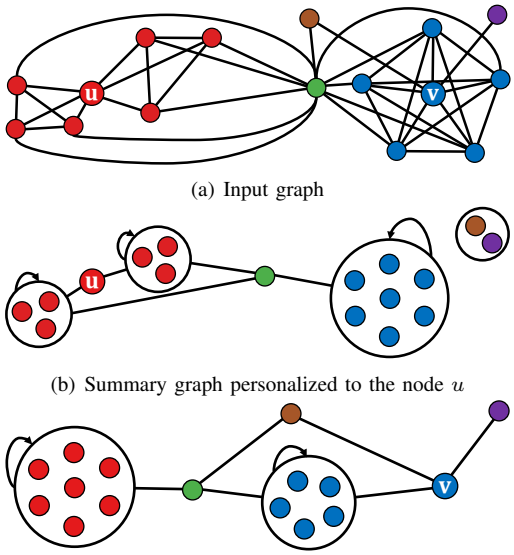
***Index Terms*—Graph summarization, Graph compression, Personalization, Graph query answering**

I. INTRODUCTION

A graph is an abstract data structure, and it naturally represents a wide range of data, including hyperlink networks [1], [2], online social networks [3], [4], collaboration networks [5], and co-purchasing networks [6]. Such real-world graphs grow rapidly as data modeled by them are accumulated at an unprecedented pace.

As a result, many real-world graphs are too large to fit in main memory, while real-time processing of various complex graph queries requires them to be resident in main memory of a single machine. Real-time answering of complex queries on graphs often requires fast random access into memory, and thus, if graphs are disk-resident and/or distributed across multiple machines, answering such queries incurs significant I/O overhead, preventing real-time processing.

As a promising approach to address the above challenge, *graph summarization* [4], [7]–[12] has received much attention among many graph-compression techniques [1], [3], [13]–[17]. Its objective is to compress a given graph G in a lossy way



(a) Input graph

(b) Summary graph personalized to the node u

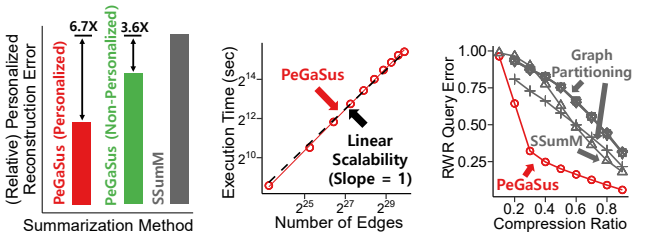
(c) Summary graph personalized to the node v

Fig. 1. **An illustrating example of personalized graph summarization.** From the input graph (top), two summary graphs are obtained. Supernodes group nodes, and each superedge between two supernodes indicates dense connections across the nodes in them. Similarly, the self-loop of each supernode indicates dense connections within the supernode. Note that one summary graph (middle) is personalized to the the node u, accurately preserving the edges near u. The other (bottom) is personalized to the the node v, which focus relatively more on the edges near v.

while satisfying a given space budget, which is typically the amount of main memory in a machine. The output, which we call *summary graph*, is in a form of a graph where each node indicates a group of nodes in G and each edge between two groups indicates the presence of edges between a large fraction of pairs of nodes in the two groups. Note that we use the term "graph summarization" to refer to this specific way of compression throughout this paper, while the term has been used to refer to a number of related but different concepts [18].

A key benefit of graph summarization is that a variety of graph queries can be directly be approximately answered from the output, in addition to many other benefits, including the interpretability of its output, the scalability its solvers, and its combinability with other graph-compression techniques.[1] This is because, the neighborhood query (i.e., retrieving the neighbors of a given node) can be approximately answered directly from a summary graph, and many graphs algorithms, including node degrees [10], clustering coefficients [10],

[1]Since an output is in the form of a graph, it can be further compressed using any graph-compression techniques.



(a) Effective (b) Scalable (c) Applicable

Fig. 2. **PEGASUS is effective, linearly scalable, and applicable to distributed multi-query answering.** (a) PEGASUS successfully personalizes summary graphs, (b) PEGASUS is linear in the number of edges, scaling to one billion edges. (c) PEGASUS answers RWR queries accurately in a distributed but "communication-free" manner. See Sect. V for details.

eigenvector centrality [11], hops between nodes, and random walk with restart, access graphs only by the neighborhood query (see Appendix A for examples).

In this work, we introduce a new problem called personalized graph summarization. It is motivated by the fact that people often have different levels of interest in different parts of a graph, following Tobler's first law of geography [19]: *"everything is related to everything else, but near things are more related than distant things."* For example, users in online social networks are more interested in connections of their close friends than in those of strangers. Moreover, travelers navigating a road network are more interested in the roads near them than in those far from them.

Given a graph G, a set of target nodes T, and a space budget k, *personalized graph summarization* is to find a summary graph of G that is personalized to T while satisfying the budget k. We formulate it as an optimization problem whose objective, namely *personalized error*, weighs error differently depending on the distance between the location of error and T. Specifically, the weight of error at closer locations is larger, requiring the output summary graph to be focused more on parts closer to target nodes T (see Fig. 1 for an example).

Our algorithmic contribution is to design PEGASUS (**Pe**rsonalized **G**raph **Sum**marization with **S**calability), a linear-time algorithm for the personalized summarization problem. It is largely based on SSUMM, which is the state-of-the-art algorithm [7] for non-personalized graph summarization, with an improved balance between exploration and exploitation. Through extensive experiments, we demonstrate that PEGASUS provides personalized summary graphs (see Fig. 2(a)) from which three types of node-similarity queries for target nodes are answered significantly more accurately than from non-personalized summary graphs of similar size obtained by previous solvers [7], [9]–[11].

Application to Distributed Multi-Query Answering. Much effort has been made to minimize the communication between machines when answering queries on graphs distributed over multiple machines [20]–[23]. We demonstrate that PEGASUS can be useful for "communication-free" distributed multi-query answering. Specifically, we assume approximate multi-query answering by multiple machines that act independently without I/O overhead over the network. To this end, using PEGASUS, we produce multiple personalized summary graphs with focuses on different regions of the input graph with a mapping function from nodes to summary graphs. The summary graphs are loaded in different machines, and we assign each query to machines on which the queries can be processed accurately, based on the mapping function. We demonstrate through experiments that three types of node-similarity queries are answered more accurately without communication from multiple summary graphs than from partitions of graphs (see Fig. 2(c)).

TABLE I
FREQUENTLY-USED SYMBOLS AND THEIR DEFINITIONS.

Symbol	Definition
$G = (V, E)$	input graph with nodes V and edges E
$\{u, v\} \in E$	edge between nodes $u \in V$ and $v \in V$
$A^{(G)}$	adjacency matrix of G
$\overline{G} = (S, P)$	summary graph with supernodes S, superedges P
$S_u \in S$	supernode containing the node $u \in V$
$\{A, B\} \in P$	superedge between supernodes $A \in S$ and $B \in S$
Π_S	set of possible unordered pairs of supernodes
$\hat{G} = (V, \hat{E})$	reconstructed graph with nodes V, edges $\hat{E}$
$A^{(\hat{G})}$	reconstructed adjacency matrix of $\hat{G}$
$T \subseteq V$	target node set
α	degree of personalization
k	desired size in bits of the output summary graph
$A \cup B$	supernode into which supernodes A and B are merged
θ	threshold for adaptive thresholding
L	list for adaptive thresholding
β	parameter for adaptive thresholding
$\{C_1, \ldots, C_q\}$	candidate groups
t_{max}	maximum number of iterations

We summarize our contribution as follows:

- **Problem Formulation:** We introduce a new problem, personalized graph summarization, and demonstrate the usefulness of personalizing summary graphs (see Fig. 2(a)).
- **Algorithm Design:** We propose PEGASUS, a linear-time algorithm for the problem. We show empirically that it scales to a graph with one billion edges (see Fig. 2(b)).
- **Extensive Experiments:** Using six real-world graphs, we exhibit the effectiveness of PEGASUS and its applicability to distributed multi-query answering (see Fig. 2(c)).

The source code and the datasets are available at [24].

In Sect. II, we introduce some concepts and formally define the personalized graph summarization problem. In Sect. III, we present PEGASUS. In Sect. IV, we discuss its application to distributed multi-query answering. In Sect. V, we evaluate PEGASUS. After reviewing related studies in Sect. VI, we draw a conclusion in Sect. VII.

II. CONCEPTS & PROBLEM DEFINITION

In this section, we introduce basic concepts and formalize our new problem, namely the personalized graph summarization problem. The frequently-used notations are listed in Table I.

A. Basic Concepts

Input Graph. An input graph $G = (V, E)$ consists of a set of *nodes* $V = \{1, 2, ..., |V|\}$ and a set of *edges* $E \subseteq \binom{V}{2}$. We assume that G is undirected without self-loops for simplicity. The *adjacency matrix* $A^{(G)} \in \mathbb{R}^{|V| \times |V|}$ of G encodes the connectivity in G as follows:

$$A^{(G)}_{uv} := \begin{cases} 1, & \text{if } \{u, v\} \in E, \\ 0, & \text{otherwise,} \end{cases} \quad \forall u \in V, \forall v \in V.$$

Summary Graph. A *summary graph* $\overline{G} = (S, P)$ of $G = (V, E)$ consists of a set of *supernodes* S and a set of *superedges* P, and the set S is a partition of V. That is, supernodes are disjoint sets whose union is V, and equivalently, each node belongs to exactly one supernode. We use $S_u \in S$ to denote the supernode containing each node $u \in V$. The summary graph $\overline{G}$ is undirected and it may have self-loops. Thus, $P \subseteq \Pi_S$ holds where $\Pi_S := \binom{S}{2} \cup \{\{A, A\} : A \in S\}$ denotes the set of all possible unordered pairs of supernodes. We use $\{A, B\} \in P$ to denote the superedge that joins $A \in S$ and $B \in S$.

Reconstructed Graph. From a summary graph $\overline{G} = (S, P)$, we can reconstruct a graph $\hat{G} = (V, \hat{E})$ with the set of nodes V and the set of reconstructed edges $\hat{E} \subseteq \binom{V}{2}$. In the reconstructed graph $\hat{G}$, there exists an edge between nodes u and v (i.e., $\{u, v\} \in \hat{E}$) if and only if the supernodes containing them (i.e., S_u and S_v) are adjacent in $\overline{G}$ (see Fig. 3 for an example). That is, the *reconstructed graph* $\hat{G}$ from $\overline{G}$ is defined as the graph whose adjacency matrix satisfies

$$A^{(\hat{G})}_{uv} = \begin{cases} 1, & \text{if } u \neq v \text{ and } \{S_u, S_v\} \in P, \\ 0, & \text{otherwise,} \end{cases} \quad \forall u \in V, \forall v \in V. \tag{1}$$

Note that, if a self-loop $\{A, A\} \in P$, all pairs of nodes in A become adjacent in $\hat{G}$.

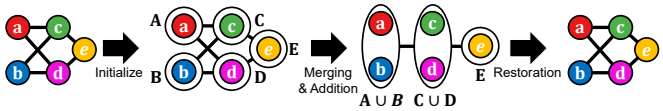
(Non-personalized) Graph Summarization. Given (a) a graph G and (b) a budget on the number of supernodes [11] or the number of bits to encode $\overline{G}$ [7], *graph summarization* aims to find the summary graph $\overline{G}$ to minimize the difference between G and the reconstructed graph $\hat{G}$ (e.g., Manhattan distance between their adjacency matrix), while satisfying the budget. Note that a variety of graph queries can be answered directly from the $\overline{G}$ without reconstructing $\hat{G}$ entirely (see Appendix A for details).

B. Problem Formulation

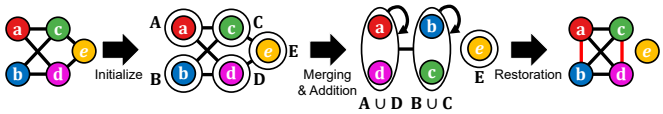
It is easy to imagine cases where people (e.g., users in online social networks and travelers navigating a road network) have different levels of interest in different parts of a graph. With this intuition, we generalize graph summarization [7], [11] to *personalized graph summarziation*, where we aim to find a summary graph personalized to given target nodes, as formalized in Problem 1 (see Fig. 1 for an example).

Problem 1 (Personalized Graph Summarization)**.**
- ***Given:***
  - *a graph* $G = (V, E)$
  - *a set of target nodes* $T (\subseteq V)$
  - *a size budget* $k > 0$
- ***Find:*** *a summary graph* $\overline{G} = (S, P)$
- ***to Minimize:*** *personalized error* $RE^{(T)}(\overline{G})$
- ***Subject to:*** $Size(\overline{G}) \leq k$.

Below, we describe $RE^{(T)}(\overline{G})$ and $Size(\overline{G})$.



(a) Example with exact reconstruction



(b) Example with reconstruction error

Fig. 3. **Two illustrating examples of graph summarization.** Note that merging supernodes with similar connectivity (e.g., A and B) yields a more concise and accurate summary graph than merging those with dissimilar connectivity (e.g., A and D).

Personalized Error. We design *peronsalized error* as a weighted reconstruction error in the form of

$$RE^{(T)}(\overline{G}) := \sum_{u=1}^{|V|} \sum_{v=1}^{|V|} W^{(T)}_{uv} |A^{(G)}_{uv} - A^{(\hat{G})}_{uv}|. \tag{2}$$

Based on the first law of geography [19], which states that *"everything is related to everything else, but near things are more related than distant things,"* we design the personalized weight $W^{(T)}_{uv}$ on each node pair $\{u, v\}$ so that it depends on the number of hops between them and the target node set T as follows:

$$W^{(T)}_{uv} := \frac{\alpha^{-(D(u,T)+D(v,T))}}{Z}, \tag{3}$$

where $D(u, T) := \min_{t \in T} \#hops(u, t)$ is the minimum number of hops between u and any target node, Z is the constant that makes the average weight 1,[2] and $\alpha \geq 1$ is a constant that controls the degree of personalization. That is, the closer an edge is to target nodes, the larger its weight is. In other words, we aim to find a summary graph of G with greater focuses on parts closer to target nodes.

Graph Size. For the size $Size(\overline{G})$ of a summary graph $\overline{G}$, we use the number of bits to encode $\overline{G} = (S, P)$ as in [7]. We assume an encoding method that requires $2 \cdot \log_2 |S|$ bits to encode each superedge (i.e., $\log_2 |S|$ bits per incident supernode) and $\log_2 |S|$ bits to encode the supernode membership of each node $v \in V$. That is,

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

Similarly, the size of an input graph $G = (V, E)$ in bits is

$$Size(G) := 2|E| \log_2 |V|. \tag{5}$$

III. Proposed Methods: PeGaSus

In this section, we present PeGaSus (**Pe**rsonalized **Ga**ph **Su**mmarization with **S**calability), our proposed algorithm for Problem 1. We first outline PeGaSus. Then, we present the cost function based on which PeGaSus performs greedy search. After that, we describe the detailed procedures of greedy search. After making a comparison with SSumM [7], which

[2] $Z := \frac{\left(\sum_{u=1}^{V} \alpha^{-D(u,T)}\right)^2 - \left(\sum_{v=1}^{V} \alpha^{-2 \times D(v,T)}\right)}{|V|(|V|-1)}$ is the average personalized weight over all possible pairs of nodes.

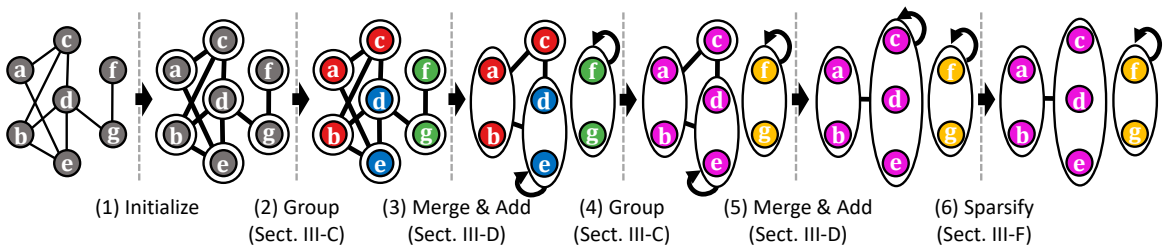


Fig. 4. **A pictorial description of PEGASUS with an illustrative toy example.** Supernodes are colored according the groups that they belong to (i.e., shingles) at each iteration. Note that supernodes are merged only among those in the same group.

PEGASUS is largely based on, we analyze the time and space complexity. It should be noted that PEGASUS is a heuristic without approximation guarantees.[3]

A. Overview (Alg. 1)

We provide the pseudocode of PEGASUS in Alg. 1 and an illustrative example in Fig. 4. Given an input graph $G = (V, E)$, a size budget k, a target node set T, the degree of personalization α, a parameter for adaptive thresholding β, and the maximum number of iterations t_{max}, PEGASUS produces a summary graph $\overline{G} = (S, P)$ personalized for T. First, it initializes the supernode set S so that each node $u \in V$ composes a singleton supernode S_u (line 1). For each edge $\{u, v\} \in E$, we create a superedge between supernodes S_u and S_v (see Fig. 3 for examples). Then, PEGASUS updates $\overline{G}$ by repeating the following steps until its size meets the budget or the maximum number of iterations is reached (line 3).

- **Candidate Generation (line 4, Sect. III-C)**: PEGASUS divides S into groups of supernodes with similar connectivity whose merger is likely to reduce the personalized cost (i.e., our objective function described in Sect. III-B). The groups are updated in every iteration.
- **Merging and Addition (line 6-7, Sect. III-D)**: Within each group, PEGASUS repeats merging supernodes and selectively adding superedges incident to the merged supernode.

During the above process, PEGASUS balances exploitation and exploration using the threshold θ, and θ is updated adaptively (line 8), as described in detail in Sect. III-E. If the size of $\overline{G}$ still exceeds the budget k, PEGASUS sparsifies $\overline{G}$ until the budget is met (lines 12-13), as described in detail in Sect. III-F.

B. Personalized Cost Function

In this subsection, we introduce the cost function that PEGASUS uses while performing greedy search.

Total Cost. In order to decide a pair of supernodes to be merged, both the size (i.e., Eq. (4)) of summary graphs after mergers and the personalized error (i.e., Eq. (2)) needs to be taken into consideration. We define the *personalized cost* of each summary graph $\overline{G}$ as

$$Cost^{(T)}(\overline{G}) := Size(\overline{G}) + \log|V| \cdot RE^{(T)}(\overline{G}). \quad (6)$$

Note that $\log|V| \cdot RE^{(V)}(\overline{G})$ equals the number of bits required to encode the difference between the input graph G and the graph $\hat{G}$ reconstructed from $\overline{G}$; and $\log|V| \cdot RE^{(T)}(\overline{G})$ can be regarded as its personalized version.[4] Since both the size and the personalized error are interpreted as the number of bits, aiming to minimize their sum in Eq. (6) aligns with the minimum description length (MDL) principle [25].

[3] Most graph-summarization algorithms [7], [9], [11] are heuristic without approximation guarantees. While S2L [10] guarantees a constant approximation ratio, empirically, SSUMM outperforms S2L [7].

Algorithm 1: Overview of PEGASUS

Input: (1) input graph $G = (V, E)$, (2) size budget k, (3) target node set T, (4) degree of personalization α, (5) parameter for adaptive thresholding β, (6) max. number of iterations t_{max},
Output: summary graph $\overline{G} = (S, P)$

1 $S \leftarrow \{\{u\} : u \in V\}$; $P \leftarrow \{\{\{u\}, \{v\}\} : \{u, v\} \in E\}$
2 $t \leftarrow 1$; $\theta \leftarrow 0.5$; $L \leftarrow$ an empty list
3 **while** $t \leq t_{max}$ and $Size(\overline{G}) > k$ **do**
4 $C \leftarrow$ generate candidate groups ▷ Sect. III-C
5 **for each** group $C_i \in C$ **do**
6 greedily merge nodes in C_i with the threshold θ (update S, P, and L) ▷ Sect. III-D
7 **end**
8 $\theta \leftarrow \lfloor \beta \times |L| \rfloor$-th largest entry in L ▷ Sect. III-E
9 $L \leftarrow$ an empty list; $t \leftarrow t + 1$
10 **end**
11 **if** $Size(\overline{G}) > k$ **then**
12 sparsify $\overline{G}$ further ▷ Sect. III-F
13 **end**
14 **return** $\overline{G} = (S, P)$

Cost Decomposition. We define the cost for each unordered pair of supernodes $\{A, B\} \in \Pi_S$ (see Sect. II-A for Π_S) as

$$Cost^{(T)}_{AB}(\overline{G}) := 2\log_2|S| \cdot \mathbb{1}_P(\{A, B\}) + \log_2|V| \cdot RE^{(T)}_{AB}(\overline{G}), \quad (7)$$

where $\mathbb{1}_P(\{A, B\}) = 1$ if $\{A, B\} \in P$ and 0 otherwise; and $RE^{(T)}_{AB}(\overline{G}) := \sum_{u \in A}\sum_{v \in B} W^{(T)}_{uv}|A^{(G)}_{uv} - A^{(\hat{G})}_{uv}|$ is the personalized error occurred between A and B. Then, the personalized cost in Eq. (6) is decomposed into the cost for each supernode pair as follows:

$$Cost^{(T)}(\overline{G}) = |V|\log_2|S| + \sum_{\{A,B\} \in \Pi_S} Cost^{(T)}_{AB}(\overline{G}). \quad (8)$$

Additionally, we define the cost for each supernode $A \in S$ as

$$Cost^{(T)}_{A}(\overline{G}) := \sum_{B \in S} Cost^{(T)}_{AB}(\overline{G}). \quad (9)$$

Cost Reduction. Let $merge(A, B; \overline{G})$ be the summary graph obtained if supernodes $A \in S$ and $B \in S$ are merged in the

[4] $\sum_{u=1}^{|V|}\sum_{v=1}^{|V|}|A^{(G)}_{uv} - A^{(\hat{G})}_{uv}|$ entries in the adjacency matrices are flipped, and due to symmetry, only a half of them needs to be specified. Encoding the location (i.e., row and column) of each of them requires $2 \cdot \log|V|$ bits.

summary graph $\overline{G} = (S, P)$ (see Sect. III-D for details), Then, the reduction in the personalized cost in Eq. (6) is

$$\Delta Cost^{(T)}(A, B; \overline{G}) := Cost_A^{(T)}(\overline{G}) + Cost_B^{(T)}(\overline{G}) - Cost_{AB}^{(T)}(\overline{G}) - Cost_{A\cup B}^{(T)}(merge(A, B; \overline{G})), \quad (10)$$

where $(Cost_A^{(T)}(\overline{G}) + Cost_B^{(T)}(\overline{G}) - Cost_{AB}^{(T)}(\overline{G}))$ is the sum of costs for supernodes A and B in $\overline{G}$ (after removing duplicates), and $Cost_{A\cup B}^{(T)}(merge(A, B; \overline{G}))$ is the cost for the merged supernode $A \cup B$ after merging A and B. Based on $\Delta Cost^{(T)}(A, B; \overline{G})$ and $(Cost_A^{(T)}(\overline{G}) + Cost_B^{(T)}(\overline{G}) - Cost_{AB}^{(T)}(\overline{G}))$, we define the relative cost reduction as

$$\Delta\overline{Cost}^{(T)}(A, B; \overline{G}) := \frac{\Delta Cost^{(T)}(A, B; \overline{G})}{Cost_A^{(T)}(\overline{G}) + Cost_B^{(T)}(\overline{G}) - Cost_{AB}^{(T)}(\overline{G})}. \quad (11)$$

As described in the following subsection, PEGASUS uses Eq. (11) when deciding supernodes to be merged.

For two nodes with dissimilar connectivity patterns, if they are distant from target nodes (that is, if the personalized weights of their incident edges are small), the absolute cost reduction in Eq. (10) can be small, while the relative cost reduction in Eq. (11) should be large. Thus, when Eq. (10) is used, such nodes can be easily merged (myoypically even when there exist pairs of nodes with similar connectivity patterns), and thus summary graphs with large personalized error can be obtained. We demonstrate experimentally in the online appendix [24] that using Eq. (11) results in better summary graphs from which queries can be answered accurately, compared to using the reduction in Eq. (10).

C. Candidate Generation

In this subsection, we describe the candidate generation step, which accelerates PEGASUS by reducing the search space. PEGASUS groups supernodes with similar connectivity (so that only pairs of nodes within the same group are considered to be merged) based on the following grounds:

- It is impractical to consider all pairs of supernodes, whose number is $\binom{S}{2}$, whenever deciding a node pair to be merged.
- Uniform sampling is likely to result in pairs of supernodes whose merger does not reduce the personalized cost much.
- Generally, if supernodes with similar connectivity are merged, encoding their connectivity together using superedges incurs little reconstruction error. For example, in Fig. 3, merging supernodes A and B (and C and D), which share exactly the same neighbors does not incur any reconstruction error, while merging A and D (and B and C) with dissimilar connectivity leads to missing or incorrect edges.

In order to group supernodes with similar connectivity, we refer to the fact that the probability of two nodes having the same shingle [26] is equal to the jaccard similarity of their neighbor sets. Specifically, we extend the notion of shingles to supernodes and define the shingle of each supernode $U \in S$ as

$$F(U) := \min_{u\in U}\left(\min_{v\in N_u\cup\{u\}} f(v)\right), \quad (12)$$

Algorithm 2: Merging and Addition Step

Input: (1) current summary graph $\overline{G} = (S, P)$, (2) candidate group C_i, (3) adaptive threshold θ, (4) target node set T, (5) degree of personalization α, (6) input graph $G = (V, E)$, (7) current list L
Output: updated summary graph $\overline{G}$ and list L

1 $\#fails \leftarrow 0$
2 **while** $|C_i| > 1$ *and* $\#fails \leq \log_2 |C_i|$ **do**
3 $I \leftarrow$ sample $|C_i|$ pairs of supernodes in C_i
4 $\{A, B\} \leftarrow \underset{\{X,Y\}\in I}{\operatorname{argmax}} \Delta\overline{Cost}^{(T)}(A, B; \overline{G})$ ▷ Eq. (11)
5 **if** $\Delta\overline{Cost}^{(T)}(A, B; \overline{G}) \geq \theta$ **then**
6 remove A and B from S and C_i
7 add $A \cup B$ to S and C_i
8 remove superedges incident to A or B from P
9 add superedges incident to $A \cup B$ to P selectively to minimize $Cost_{A\cup B}^{(T)}(\overline{G})$ ▷ Eq. (9)
10 $\#fails \leftarrow 0$
11 **else**
12 add $\Delta\overline{Cost}^{(T)}(A, B; \overline{G})$ to L ▷ Eq. (11)
13 $\#fails \leftarrow \#fails + 1$
14 **end**
15 **end**
16 **return** $\overline{G} = (S, P)$ and L

where N_u is the set of neighboring nodes of $u \in V$ in the input graph G and $f : V \to \{1, 2, ..., |V|\}$ is a uniform random hash function. Note that two supernodes are more likely to have the same shingle if their members' connectivities are more similar.

Example 1 (Shingle). *In Fig. 3, suppose that $f(a) = 5$, $f(b) = 4$, $f(c) = 3$, $f(d) = 2$, and $f(e) = 1$ in the input graph (left-most). After initialization, $F(A) = \min(f(a), f(c), f(d)) = 2$, and $F(B) = \min(f(b), f(c), f(d)) = 2$. Similarly, $F(C) = F(D) = F(E) = 1$. After merging A with B and C with D, $F(A \cup B) = \min(f(a), f(b), f(c), f(d)) = 2$ and similarly $F(C \cup D) = 1$. Note that supernodes with similar connecitivity (e.g., A and B; and C and D) have the same shingle.*

Then, the supernodes with the same shingle are grouped together as a candidate group. PEGASUS further divides each candidate group recursively by repeating the above process at most a constant (spec., 10) times. After that PEGASUS ensures that the size of each candidate group is at most a constant (spec., 500) by randomly dividing larger groups. We use $C = \{C_1, C_2, \ldots, C_q\}$ to denote the candidate groups, which are used in a later step (see Sect. III-D). In different iterations, PEGASUS draws a hash function f using different random seeds, and as a result, obtains different candidate groups to further explore the search space.

D. Merging and Addition (Alg. 2)

In this subsection, we describe how PEGASUS merges supernodes within each candidate group C_i (see Sect. III-C) in a greedy manner and creates superedges accordingly. See Alg. 2 for the pseudocode. PEGASUS first draw $|C_i|$ pairs of supernodes uniformly at random within C_i (line 3). Then, it chooses a pair, which we denote by $\{A, B\}$, that maximizes the relative reduction in the personalized cost (i.e., Eq. (11) in Sect. III-B)

(line 4) If the relative reduction $\Delta\overline{Cost}^{(T)}(A, B; \overline{G})$ is at least the threshold θ (see Sect. III-E for how to decide θ) (line 5), then the chosen pair is merged (lines 6-7). When A and B are merged, the superedges incident to A or B are removed from P as they are no longer valid (line 8). For the merged supernodes $A \cup B$, PEGASUS adds superedges incident to it selectively to P so that the personalized cost $Cost^{(T)}_{A\cup B}(\overline{G})$ (see Eq. (9)) for $A \cup B$ is minimized, while fixing all non-incident superedges (line 9). It should be noted that, since each supernode is s set of subnodes, merging the two supernodes A and B results in $A \cup B$ (i.e., the union of the two sets). Specifically, PEGASUS adds each potential superedge $\{(A\cup B), X\}$ to P if and only if it decreases the personalized cost $Cost^{(T)}_{(A\cup B)X}(\overline{G})$ (see Eq. (7)) for it. The detailed procedure with the time complexity are provided in Lemma 1 and its proof. We denote the updated $\overline{G}$ by $merge(A, B; \overline{G})$.

Lemma 1. *For any supernodes $A, B \in S$ in a summary graph $\overline{G}$, the time complexity of updating $\overline{G}$ to $merge(A, B; \overline{G})$ is*

$$O\left(\sum_{u\in A} |N_u| + \sum_{v\in B} |N_v|\right),$$

where N_x is the set of neighbors of a node $x \in V$ in the input graph G.

Proof. A proof can be found in the online appendix [24]. □

For each candidate group C_i, PEGASUS repeats the above process until (a) only one supernode is left in the group or (b) it fails to merge supernodes $\log(|C_i|)$ times in a row. That is, if the relative reduction by a chosen pair is smaller than the threshold θ, $\log(|C_i|)$ times in a row, PEGASUS concludes that no promising supernode pairs are left in C_i. The relative reduction smaller than θ is stored in a list L (line 12), which is later used to adjust θ, as described in the following subsection.

E. Adaptive Thresholding

In this subsection, we present how PEGASUS adjusts the threshold θ for relative reduction, which is used in line 5 of Alg. 2. The threshold θ balances exploitation and exploration. Specifically, if θ becomes smaller, supernode pairs are merged more aggressively within the candidate groups in the current iteration. However, if θ becomes larger, PEGASUS merge pairs less aggressively, considering the possibility that better pairs can be found within the candidate groups in future iterations. Recall that PEGASUS obtains different candidate groups in different iterations.

PEGASUS intializes θ to 0.5 and adjusts it adpatively based on the relative reductions stored in L (line 12 of Alg. 2). Specifically, PEGASUS sets θ for the next iteration to the $\lfloor \beta \times |L| \rfloor$-th largest entry in L and then clears L (lines 8-9 of Alg. 1). The larger the parameter β is, the faster θ decreases, with a greater emphasis on exploitation. The empirically effect of β is shown in Sect. V-E. Recall that relative reductions in L are smaller than θ for the current iteration (see Sect. III-D). Hence, PEGASUS gradually decreases θ over iterations, gradually putting more emphasis on exploitation.

F. Further Sparsification

If the size of the summary graph $\overline{G}$ stills exceeds the budget k after t_{max} iterations (line 11 of Alg. 1), PEGASUS greedily drops superedges in P (line 12) until the budget is met. Specifically, superedges dropped in increasing order of $Cost^{(T)}_{AB}(\overline{G})$ (Eq. (7) in Sect. III-B), where $\{A, B\} \in P$ denotes a dropped superedge.

G. Comparison with SSUMM [7]

PEGASUS is largely based on SSUMM [7], which is a state-of-the-art algorithm for non-personalized graph summarization, with the following major differences:

- **Personalized Cost Function (Sect. III-B)**: PEGASUS is based a new cost function to balance the size of summary graphs and the personalized error when deciding supernode pairs to be merged.
- **Adaptive Thresholding (Sect. III-E)**: In order to balance exploitation and exploration, PEGASUS controls the threshold θ adaptively based on statistics collected at runtime, while SSUMM relies on a fixed rule.[5]

Notably, the experiments in Sect. V-B show that PEGASUS outperforms SSUMM in non-persoalized cases (i.e., when $T = V$), as well as in personalized cases,

H. Complexity Analysis

In this section, we analyze the time and space complexity of PEGASUS. We assume $|V| = O(|E|)$ for simplicity.

Time Complexity: PEGASUS scales linearly with the size of the input graph, as formalized in Theorem 1.

Theorem 1 (Linear Scalability of PEGASUS)**.** *The time complexity of Alg. 1 is $O(t_{max} \cdot |E|)$.*

Proof. We focus on showing that the differences of PEGASUS from SSUMM, whose time complexity is $O(t_{max} \cdot |E|)$ (see Theorem 3.4 of [7]), does not increase the time complexity. According to Lemma 1, computing the cost reduction for each supernode pair (based on which supernode pairs to be merged are chosen) takes $O(\sum_{u\in A} |N_u| + \sum_{v\in B} |N_v|)$ time, as in SSUMM. Regarding adaptive thresholding, the $\lfloor \frac{|L|}{10} \rfloor$-th largest entry in L (line 8 of Alg. 1) can be found in $O(|L|)$ time by the "median of medians" algorithm [27]. Since the number of failures cannot exceed $\sum_{i=1}^{q}(|C_i| \log |C_i|) = O(|V| \cdot \max_{i=1}^{q} \log |C_i|) = O(|V|)$, and thus $O(|L|) = O(|V|)$ holds. Recall that, as described in Sect. III-C, the size of each candidate group is at most a constant, and thus $O(\max_{i=1}^{q} \log |C_i|) = O(1)$. Thus, t_{max} updates of the threshold θ take $O(t_{max} \cdot |V|) = O(t_{max} \cdot |E|)$ time in total. □

Space Complexity: PEGASUS retains an input graph $G = (V, E)$, a summary graph $\overline{G} = (S, P)$, a hash function $f : V \to \{1, 2, ..., |V|\}$, a shingle function $F : S \to \{1, 2, ..., |V|\}$, and a list L. Additionally, it retains intermediate results of size $O(|V| + |E|)$ (see the online appendix [24] for details). Since $|S| \leq |V|$, $|P| \leq |E|$, and $|L| = O(|V|)$ (see the proof of Theorem 1), the space complexity is $O(|V| + |E|) = O(|E|)$.

[5] $\theta(t) = (1 + t)^{-1}$ if $t < t_{max}$ and 0 otherwise.

Algorithm 3: Application of PEGASUS to Distributed Multi-Query Answering

Input: (1) input graph G, (2) machines $\{M_1, M_2, \ldots, M_m\}$
(3) size budget k, (4) degree of personalization α,
(5) parameter for adaptive thresholding β
(6) max. number of iterations t_{max},

Preprocessing:

$\{V_1, V_2, ..., V_m\} \leftarrow$ GraphPartitioning(G, m)
for each $V_i \in \{V_1, V_2, ..., V_m\}$ **do**
  $\overline{G}_i \leftarrow$ PEGASUS$(G, k, V_i, \alpha, \beta, t_{max})$
  Load $\overline{G}_i$ into the main memory of M_i

Distributed Multi-Query Answering:

for each arrived query on a node q **do**
  Find i where $q \in V_i$
  Assign the query to M_i, which answers the query independently using $\overline{G}_i$

IV. APPLICATION: "COMMUNICATION-FREE" DISTRIBUTED MULTI-QUERY ANSWERING

In this section, we discuss an application of PEGASUS to distributed multi-query answering.

Background. Real-time processing of various complex graph queries requires fast random access into memory. Thus, if graphs are distributed across multiple machines, answering such queries incurs significant communication overhead, preventing real-time processing. As a result, much effort has been made to minimize communication overhead when answering queries on graphs distributed over multiple machines [20]–[23].

Intuition. PEGASUS can be utilized for "communication-free" distributed multi-query answering. Specifically, multiple personalized summary graphs with different target node sets are obtained by PEGASUS. Then, different summary graphs are loaded on different machines, which act independently without communication. If multiple queries are given, each query is assigned to a machine on which it can be processed accurately.

Procedure. Assume m machines each of which has main memory of size k are available. We first divide the node set V into m subsets using the Louvain method [28], while any graph-partitioning method (e.g., [29]–[35]) can be used instead. Let the m subsets of nodes be V_1, V_2, ..., V_m. For each node set V_i, we create a summary graph $\overline{G}_i$ personalized to V_i within the budget k using PEGASUS, and $\overline{G}_i$ is loaded into the main memory of the i-th machine. Given multiple graph queries, we assign each query on a node q to the i-th machine satisfying $q \in V_i$, and the machine answers the query using $\overline{G}_i$ without communicating with other machines. Since $\overline{G}_i$ is personalized to q, $\overline{G}_i$ is expected to maintain much information relevant to q, and thus the answer from it is expected to be accurate. We confirm this expectation experimentally in Section V-F. We provide the pseudocode in Alg. 3.

Potential Alternatives. As a potential alternative for communication-free distributed multi-query processing, m overlapping subgraphs of size k can be distributed over the main memory of m machines for query answering. In our experiments in Sect. V-F, we first partition the node set V into m subsets using graph-partitioning methods and compose each i-th subgraph of size k (see Eq. (5)) using the edges closest to each i-th subset. Each query on a node u is assigned to the i-th machine where $u \in V_i$.

TABLE II
SUMMARY OF SIX REAL-WORLD GRAPHS AND ONE SYNTHETIC GRAPH.

Name	# Nodes	# Edges	Summary
LastFM-Asia (LA) [36]	7,624	27,806	Social
Caida (CA) [37]	26,475	53,381	Internet
DBLP (DB) [38]	317,080	1,049,866	Collaboration
Amazon0601 (A6) [6]	403,364	2,443,311	Co-purchase
Skitter (SK) [37]	1,694,616	11,094,209	Internet
Wikipedia (WK) [39]	3,174,745	103,310,688	Hyperlinks
Synthetic (ST) [40]	10,000,000	1,000,000,000	BA Model

V. EXPERIMENTS

In this section, we review our experiments to answer Q1-Q4.

- **Q1. Effectiveness**: Does PEGASUS provide personalized summary graphs?
- **Q2. Scalability**: Does PEGASUS scale linearly with the number of edges in the input graph?
- **Q3. Comparison with the State of the Art**: Does PEGASUS provide better summary graphs to target nodes than the best non-personalized graph summarization methods?
- **Q4. Effect of Parameters**: How do α and β affect the output summary graphs?
- **Q5. Application**: Is PEGASUS useful for communication-free distributed multi-query answering?

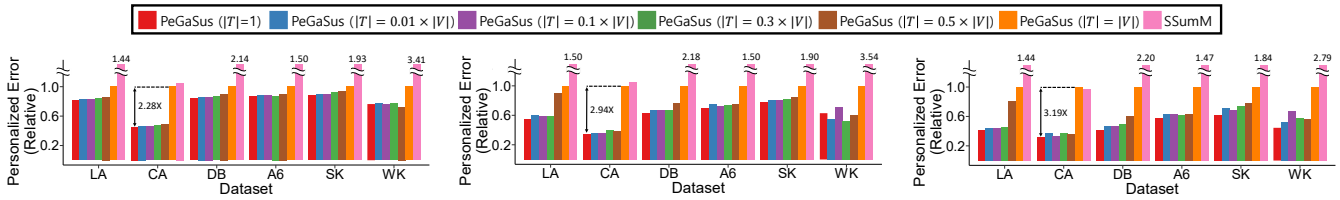
A. Experimental settings

Machines. We performed our experiments on a desktop with AMD Ryzen 7 3700X CPU and 128GB memory.

Datasets. We used six real-world graphs summarized in Table II. We removed all self-loops and edge directions in them and used only the largest connected components.

Implementations. We implemented PEGASUS, SSUMM [7], and K-GRASS [11] in Java. We used the implementations SAAGS [9] and S2L [10] provided by the authors, and they were implemented in Java and C++, respectively. We set the size budget from 10% to 90% of the size in bits of the input graph at the same interval and set the maximum number of iterations to 20 for PEGASUS and SSUMM; and we set it from 10% to 90% of the number of supernodes at the same interval for SAAGS, K-GRASS, and S2L. For PEGASUS, we set α to 1.25 and β to 0.1 unless otherwise stated. For K-GRASS, we used the $SamplePairs$ method with $c = 1.0$, as suggested in [11]. For SAAGS, we set the number of sample pairs to $\log n$ and the used the count-min sketch with $w = 50$ and $d = 2$. For S2L, we used the type of reconstruction error as $L1$ without dimensionality reduction. For graph partitioning in Sect. V-F, we implemented the Louvain method [28] in Java and used BLP [41] and SHP (SHPI, SHPII, and SHPKL) [42] implemented in NumPy by the authors of [43]. We set the maximum number of iterations to 10 and used 8 shards.

Node-Similarity Queries. We considered the following three types of node-similarity queries:



(a) $\alpha = 1.25$ (b) $\alpha = 1.5$ (c) $\alpha = 1.75$

Fig. 5. **PEGASUS provides personalized summary graphs.** The relative personalized error tends to decrease as we reduce the size of the target node set (i.e., the summary graph is more focused) and grow the degree of personalization α.

- *Random Walk with Restart* (RWR) [44]: The RWR score of a node w.r.t. a query node q is the stationary probability that a random walker stays at the node when it repeatedly restarts at q. We set the restarting probability to 0.05.
- *Length of the Shortest Path* (HOP): The HOP of a node w.r.t. a query noede q is the length of shortest paths from q to the node. If there is no path between them, we used the length of longest path in the given (sub)graph as the HOP.
- *Penalized Hitting Probability* (PHP) [45], [46]: The PHP of a node u w.r.t. a query node q is defined as

$$\text{PHP}_u := \begin{cases} 1 & \text{if } u = q, \\ c \sum_{v \in N_u} \left(\frac{w_{uv}}{w_u} \cdot \text{PHP}_v \right) & \text{if } i \neq q, \end{cases}$$

where N_u is the set of neighbors of u, w_{uv} is the weight of the edge $\{u, v\}$ (which is 1 if the graph is unweighted), and w_u is the weighted degree of u. We set c to 0.95.

The queries can be answered directly from a summary graph, as described in Appendix A.

Evaluation Measures. We measured compression rates in bits. That is, the *compression rate* of a summary graph $\overline{G}$ of a graph G is $\frac{Size(\overline{G})}{Size(G)}$. As in [7], for the size in bits of weighted summary graphs with the maximum weight ω_{max}, we used

$$|P|(2 \log_2 |S| + \log_2 \omega_{max}) + |V| \log_2 |S|.$$

For each query q, we measured the accuracy of an approximate answer vector $\hat{x} \in \mathbb{R}^{|V|}$ by comparing it with the ground-truth answer vector $x \in \mathbb{R}^{|V|}$ in two ways:

- *Symmetric Mean Absolute Percentage Error* (SMAPE) [47] (the lower, the better):

$$\text{SMAPE}(x, \hat{x}) := \sum_{u \in V} \frac{|x_u - \hat{x}_u|}{|x_u| + |\hat{x}_u|},$$

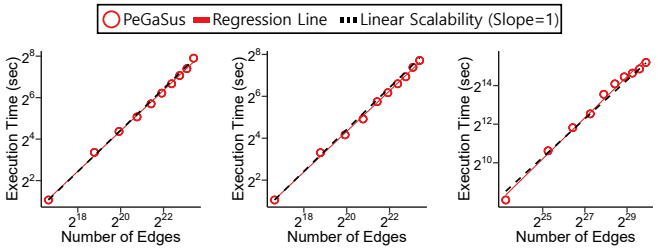
If $x_u = \hat{x}_u = 0$, 0 is used instead of $\frac{|x_u - \hat{x}_u|}{|x_u| + |\hat{x}_u|}$. Note that SMAPE is well defined even when $x_u = 0$ or $\hat{x}_u = 0$.

- *Spearman Correlation Coefficients* (SC) [48] (the higher the better): It measures the Pearson correlation coefficient between the rankings of the entries of x and the rankings of the entries of $\hat{x}$. It compares rankings, which are more important than absolute values in many graph applications.

We report the average when multiple query nodes were used.

B. Q1. Effectiveness of PEGASUS (Fig. 5)

We demonstrate that PEGASUS provides summary graphs personalized to target nodes. To this end, we randomly chose a test node u in the input graph and sampled $|T|$ target nodes by breadth first search (BFS) from u. Then, while varying the degree of personalization α and the target node set T, we measured the personalized error at u (i.e., Eq. (2) with $T = \{u\}$) relative to that in non-personalized cases (i.e., when $T = V$). Fig. 5 shows the result averaged over three test nodes when the size budget was set so that the compression ratio is 0.5. The relative personalized error tended to decrease as we decreased the size of the target node set (i.e., the summary graph is more focused to i) and increased the degree of personalization. **These results indicate that summary graphs are personalized effectively by PEGASUS.** Notably, even in non-personalized cases (i.e., when $T = V$), PEGASUS outperformed SSUMM, as discussed in Sect. III-G.



(a) Skitter ($|T|$=100) (b) Skitter ($|T|=\frac{|V|}{2}$) (c) Synthetic ($|T|$=100)

Fig. 6. **PEGASUS exhibits linear scalability.** PEGASUS scales linearly with the edge count, **to one billion edges**, regardless of the target node number. Fig. 2(b) shows the result on the synthetic dataset when $|T| = \frac{|V|}{2}$.

C. Q2. Scalability of PEGASUS (Fig. 6)

We show the linear scalability of PEGASUS. To this end, we measured how the execution time depend on the number of edges in the input graph, while fixing $|T|$ to 100 or $\frac{|V|}{2}$. We obtained the induced subgraphs of different sizes by randomly sampling different numbers of nodes ranging from 10% to 100% at the same interval from the the Skitter dataset and a synthetic graph ($|V| = 10^7, |E| = 10^9$). The synthetic graph was generated by the the Barabási-Albert model [40], which reflects several structural properties of real-world graphs. As seen in Figs. 2(b) and 6, **PEGASUS scaled linearly with the number of edges**, regardless of the target node number.

D. Q3. Comparison with the State of the Art (Figs. 7-8)

We compare PEGASUS with state-of-the-art non-personalized graph summarization algorithms in terms of (a) the accuracy of query answers, (b) the conciseness of summary graphs, and (c) speed. To this end, we sampled 100 query nodes by breadth first search (BFS) from a random node, used them as the target node set T.

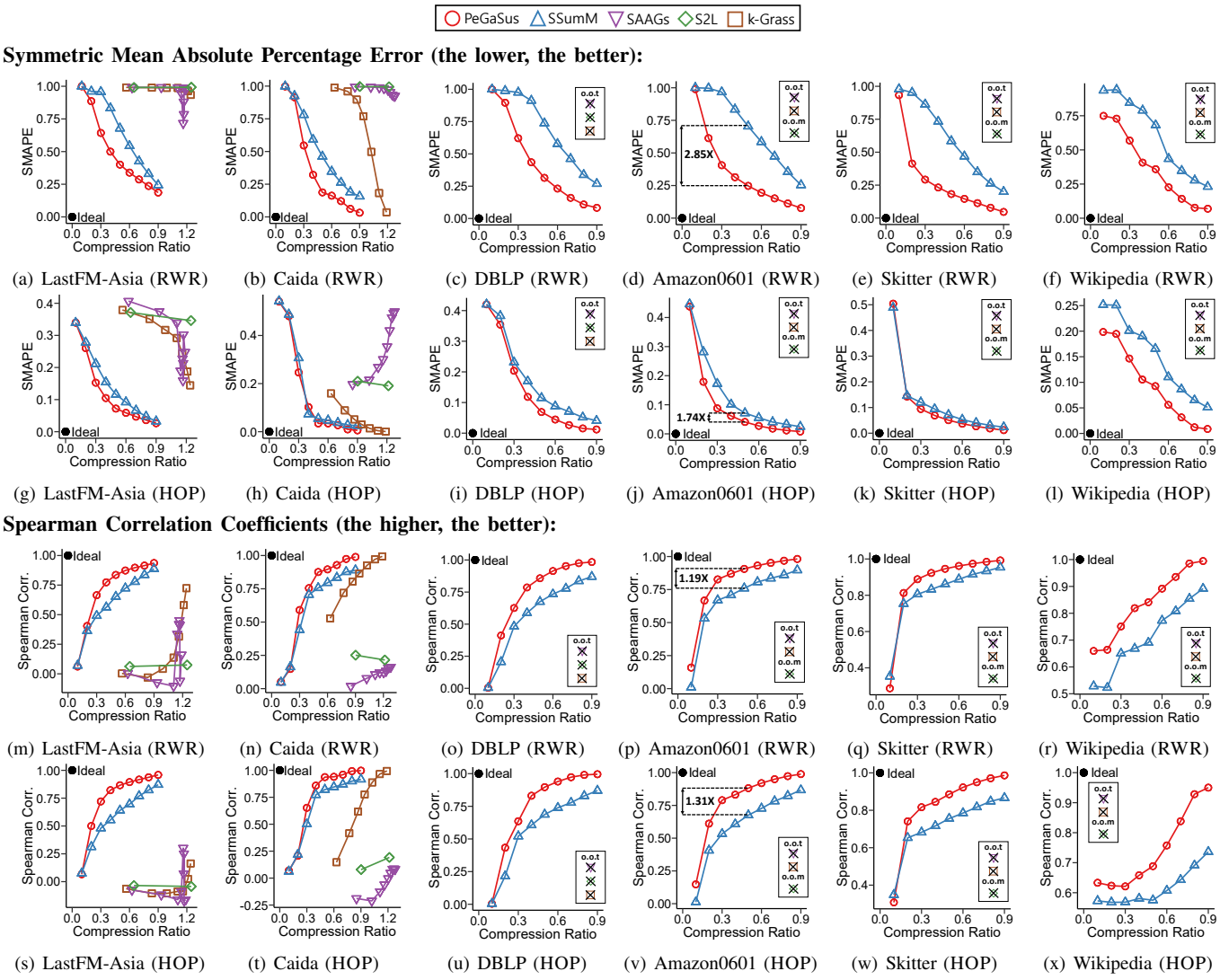


Fig. 7. **PEGASUS gives summary graphs where queries on target nodes are answered most accurately.** o.o.t: out of time (> 48hours). o.o.m: out of memory (> 128GB). The degree of personalization α is fixed to 1.25, and the size of the target node set is fixed to 100.

As seen in Fig. 7, **RWR and HOP queries were answered significantly more accurately from personalized summary graphs obtained by PEGASUS** than from non-personalized ones obtained by all other algorithms. We obtained similar results for PHP queries, as reported in the online appendix [24]. For example, for the Amazon0601 dataset, queries were answered up to $2.85\times$ and $1.31\times$ more accurate in terms of SMAPE and SC (see Sect. V-A), respectively, when the compression rate was 0.5.

As shown in Fig. 8, **PEGASUS was one of the most scalable algorithms, and queries were processed rapidly on its output** since it selectively adds superedges (see Sect. III-D). Query processing took much longer on dense summary graphs obtained by K-GRASS, S2L, and SAAGS, which add superedges without selection.

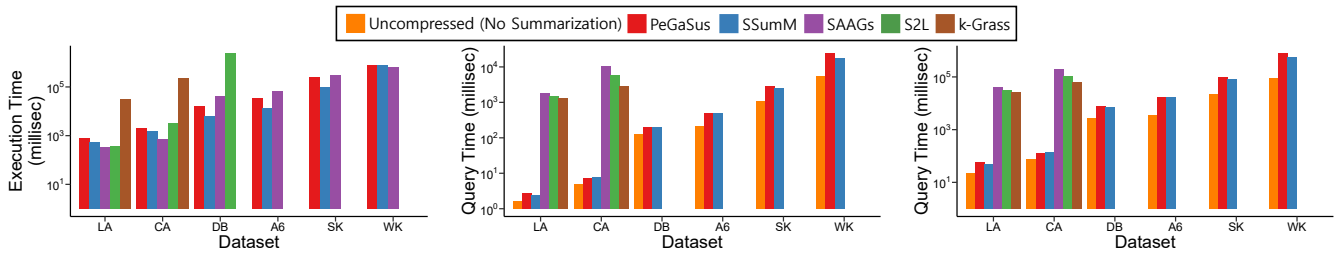
E. Q4. Effects of Parameters (Figs. 9-11)

We analyze how the accuracy of the answers of node-similarity queries (see Sect. V-A) obtained from personalized summary graphs depends on parameters α and β.

Effect of α. As seen in Fig. 9, the answers were most accurate when the degree of personalization α was moderate (spec., when it was 1.25 or 1.5); and they were less accurate when α was higher and more global information was lost. Specifically, $\alpha = 1.5$ led to the best accuracy in the Wikipedia dataset, whose effective diameter[6] is remarkably small despite its huge size. In the all other datsets, $\alpha = 1.25$ led to the best accuracy.

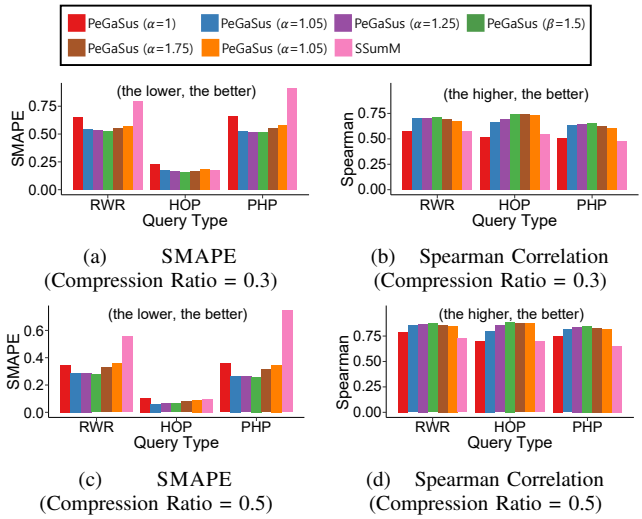
In order to systematically analyze the relation between α and the effective diameter, we generated 5 synthetic graphs of the same size and edges but with different effective diameters using the Watts-Strogatz model [49]. Specifically, we changed the rewiring probability from 0 to 0.1 (spec., we tried 0, 0.0001, 0.001, 0.01, and 0.1), while fixing the number of nodes to $1,000$ and the number of edges to $10,000$, and as a result the effective diameter varied from 3.71 to 44.95. **As seen in Fig. 10, the best-performing degree of personalization α decreased as the effective diameter increased.** This is because, when the effective diameter is large, a large fraction of

[6]We used the 90-percentile effective diameter [37], i.e., the minimum number of hops such that 90% of nodes pairs are within the hops from each other.



(a) Summarization Time (b) Query Time: Breadth First Search (c) Query Time: Random Walk with Restart

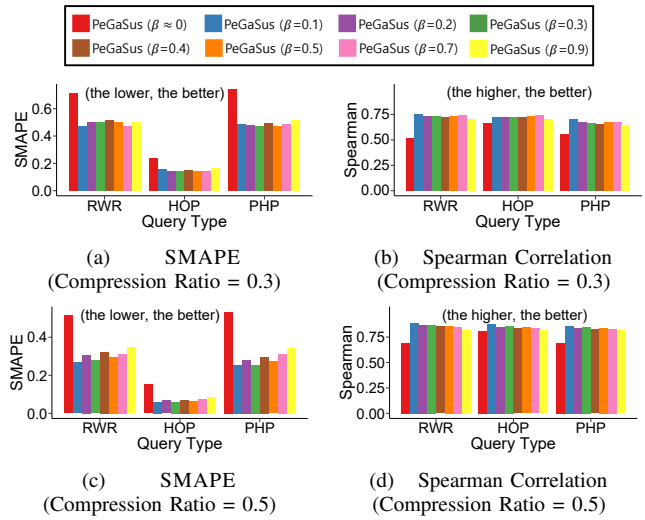
Fig. 8. **PEGASUS is scalable, and it provides sparse summary graphs on which queries are processed efficiently.** S2L and K-GRASS run out of time ($>$ 48 hours) or out of memory ($>$ 128 GB) for large datasets; and SAAGs produces dense summary graphs where queries run out of time ($>$ 48 hours). The compression ratio is 0.5, and it takes almost the same time to process RWR queries and PHP queries.



(a) SMAPE (Compression Ratio = 0.3) (b) Spearman Correlation (Compression Ratio = 0.3)

(c) SMAPE (Compression Ratio = 0.5) (d) Spearman Correlation (Compression Ratio = 0.5)

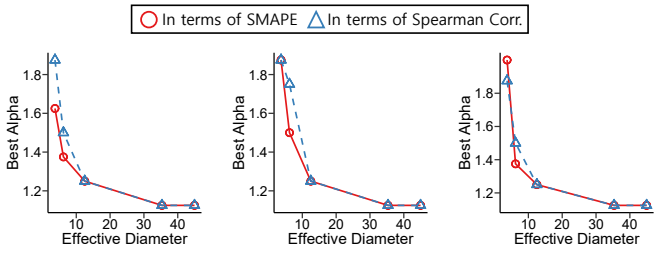
Fig. 9. **Queries on target nodes can be answered more accurately from personalized summary graphs** ($\alpha > 1$) than from non-personalized ones ($\alpha = 1$). The results are averaged over all datasets. Note that answers are most accurate when the degree of personalization α is moderate.



(a) SMAPE (Compression Ratio = 0.3) (b) Spearman Correlation (Compression Ratio = 0.3)

(c) SMAPE (Compression Ratio = 0.5) (d) Spearman Correlation (Compression Ratio = 0.5)

Fig. 11. **Queries are answered most accurately when $\beta = 0.1$**, in the majority of the cases. The largest entry in L (see Sect. III-E) is chosen when $\beta \approx 0$. The reported results are averaged over all datasets.



(a) RWR (b) HOP (c) PHP

Fig. 10. **The best-performing α decreases as the effective diameter of the input graph increases.** The best-performing alphas are chosen based on each evaluation measure (i.e., SMAPE and SC) averaged over all datasets when a compression ratio is 0.3.

edges are distant from a target node set, and large α understates their weight, which depends on the distance, too much.

Effect of β. As seen in Fig. 11, in the majority of cases, $\beta = 0.1$ **resulted in best accuracy**. The accuracy was not sensitive to β as long as β was not too close to 0 or 1.

F. Q5. Application of PEGASUS (Fig. 12)

We demonstrate that PEGASUS can be useful for distributed multi-query answering. We assumed eight machines and applied PEGASUS and five graph partitioning algorithms [28], [41], [42] to communication-free distributed multi-query processing, as described in Sect. IV. We randomly chose 100 query nodes in the Wikipedia dataset and 500 query nodes in the others dataset; and the averaged results are reported. **Recall that summary graphs are not personalized only for query nodes.** Their target node sets together cover all nodes.

As seen in Fig. 12, in almost all settings, **RWR and HOP queries were answered significantly more accurately from personalized summary graphs obtained by PEGASUS than from subgraphs obtained by graph partitioning.** We obtained similar results for PHP queries, as reported in the online appendix [24]. For example, in the Caida dataset, answers were up to 2.52× and 1.25× more accurate in terms of SMAPE and SC (see Sect. V-A), respectively, when the compression rate was 0.5. A shortcoming of our approach is that processing queries took longer on summary graphs than on subgraphs, which are uncompressed (see Fig. 8).

VI. RELATED WORKS

Graph Summarization: While the term "graph summarization" refers to a specific way of compressing graphs [7], [10], [11] in this work, it has been used also for a wide range of different but related concepts, as surveyed in [18]. The most similar one is a lossless graph-compression technique [4], [8], [12], [50], [51] where the input graph is encoded "losslessly" by a summary graph, positive edge corrections, negative edge corrections, and optionally a hierarchy of nodes. Among

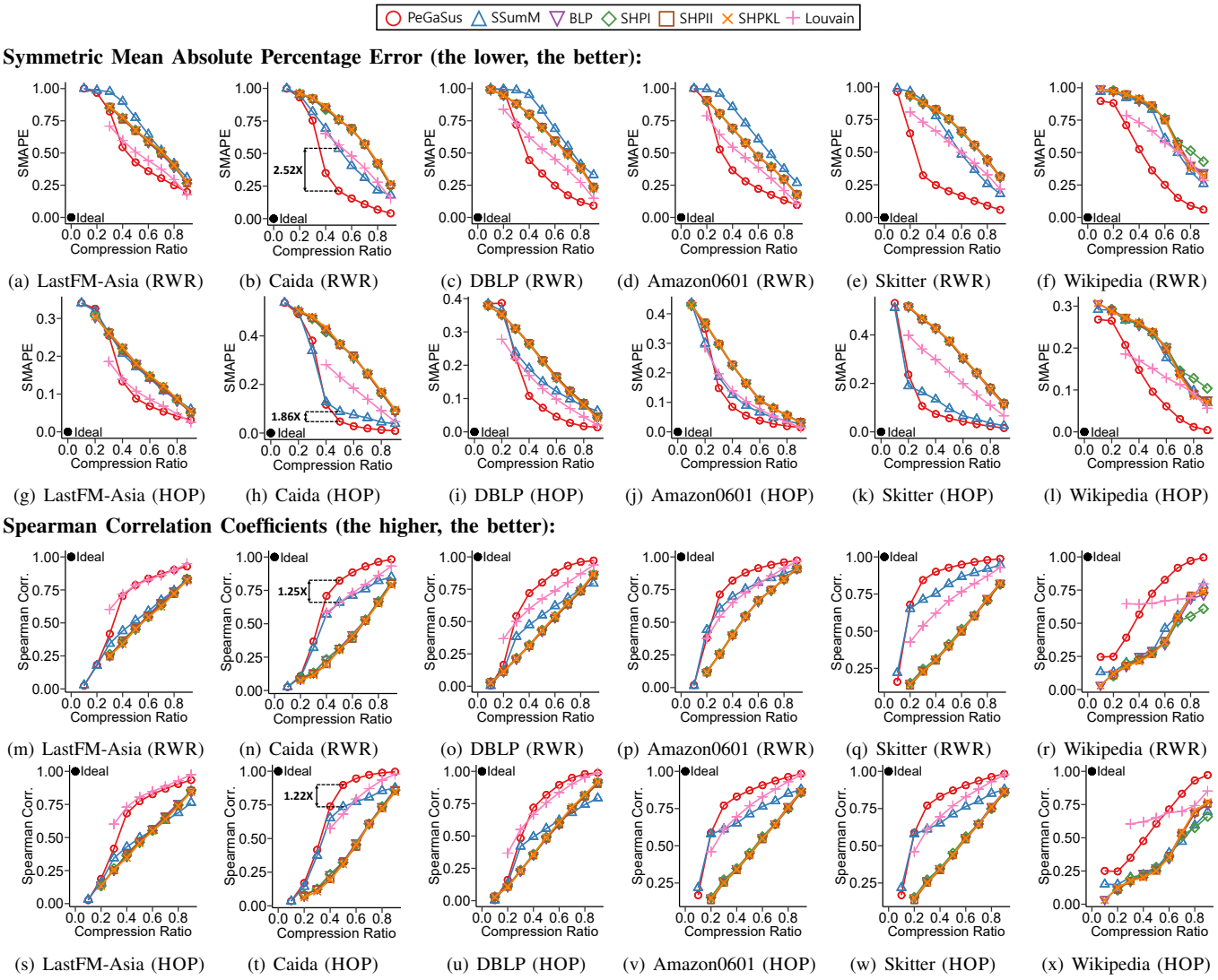


Fig. 12. **PEGASUS is useful for "communication-free" distributed multi-query processing.** Queries are answered more accurately from distributed personalized summary graphs (PEGASUS) than from non-personalized summary graphs (SSUMM) or distributed subgraphs (the others). The degree of personalization α is fixed to 1.25. In some datasets, compression rates cannot be lowered further due to imbalance among graph partitions.

recent lossy graph summarization methods, UDS [52] uses memoization for summarizing a graph into the given number of supernodes while ensuring the proposed utility function does not drop below a certain threshold. T-BUDs [53] use the minimum spanning tree of the two-hop graph, outperforming UDS in terms of speed and memory efficiency. Among recent lossless graph summarization methods, LDME [54] reduces the search space of supernode pairs to be merged and accelerates the computation of saving and the update of encoding. Fan et al. [55] proposed a lossless graph contraction scheme that can be adapted for several types of graph queries (e.g., traingle counting and shortest distance), while PEGASUS focuses on the neighborhood query, which is the key building block of many graph algorithms, as discussed in Appendix A.

Safavi et al. [56] aimed to summarize a knowledge graph to capture facts preferred by a single user, who is not necessarily a node in the graph, based on the user's past queries. The summary is a 'subgraph' that captures local information and discards the others, while PEGASUS produces a summary graph (i.e., graphs consisting of supernodes) that summarizes the entire graph with varying resolutions, as seen in Fig. 1.

Graph Partitioning: Graph partitioning is to decompose a graph into subgraphs by partitioning nodes into groups for certain goals (e.g., to minimize normalized cuts). Many approaches, including label propagation [41], [43], [57], local search [42], and eigen decomposition [58], have been developed for various applications, including VLSI circuit placement [59], sparse matrix factorization [60], and storage sharding [42], [61], [62]. As our competitors, we use those for balanced partitioning [41], [42] and a hierarchical node clustering algorithm [28].

Applications: Distributed Query Processing System: Distributed graph query processing systems can be divided into two types depending on whether multiple queries can be executed concurrently. Some systems [63]–[67] are designed to handle one specific job at a time over the entire graph (e.g., PageRank and graph partitioning). They follow a vertex or edge-centric [68] scatter-gather model with batch processing. The others [20], [21], [69], where a graph storage and a query

processor exist on each machine, facilitate graph traversal over small areas in the graph. However, if each query spans over nodes on the boundary of graph partitions, a large amount of inter-machine communication is inevitable, which eventually slows down the query processing time. Several multi-queriable distributed SPARQL engines [70], [71] were developed to handle queries on RDF graphs. Mayer et al. [72] proposed Q-graph for multi-query graph analysis that considers user-centric graph applications. EdgeFrame [73], a graph-specialized Spark DataFrame, caches the edge structure of a graph in compressed form on all workers in the cluster, which circumvents the inherent communication bottlenecks of worst-case optimal join (WCOJ) [22], [74], [75] on distributed graphs. Different from the previous studies, we consider completely removing inter-machine communications, at the expense of exactness, using personalized summary graphs, as an application of PEGASUS to distributed multi-query processing (see Sects. IV and V-F).

Graph Visualization: Our work is also related to graph visualization where the objective is to provide a pictorial representation of the nodes and edges where users focus the most. Rafiei [76] and Ellis et al. [77] use sampling and magnification to focus on a specific part of a graph. GMine [78] obtains hierarchical communities and offers multi-resolution graph exploration. It extracts a subgraph of interest based on the initial set of target nodes. While these tools focus on small parts of graph for visualizations, PEGASUS summarizes the entire graph with for compression and query processing.

VII. Conclusion

In this paper, we introduce the problem of finding summary graphs personalized to given target nodes (Problem 1), motivated by the observations that people have different levels of interest in different parts of graphs. We formulate the problem as an optimization problem, and we propose PEGASUS, a linear-time algorithm (Theorem 1) that successfully summarizes a graph with one billion edges (Fig. 6). Through extensive experiments using six real-world graphs and three types of node-similarity queries, we show that PEGASUS provides summary graphs from which queries on target nodes are answered significantly more accurately than from non-personalized summary graphs obtained by state-of-the-art graph summarization methods (Fig. 7). We also demonstrate the effectiveness of PEGASUS for communication-free distributed multi-query answering. Specifically, we show that queries are answered more accurately from distributed personalized summary graphs than from distributed subgraphs (Fig. 12). The source code and the data are available at [24] for reproducibility.

Appendix

A. Query Answering

As described in Alg. 4, the approximate neighbors of a given node $u \in V$ can be retrieved directly from a summary graph $\overline{G}$. That is, the *neighborhood query* can be answered efficiently on $\overline{G}$, without restoring the entire graph. A wide range of graph algorithms (e.g., BFS, DFS, Dijkstra's, and PageRank) access graphs only through neighborhood queries, and thus also can be executed directly on $\overline{G}$. Alg. 5 and Alg. 6 describe how to answer RWR and HOP queries on $\overline{G}$. Answers of PHP queries, which are used in Sect. V, can be computed from those of RWR queries (see [79] for details). In Sect. V, on weighted summary graphs, RWR and HOP queries were processed considering superedge weights.

Algorithm 4: getNeighbors($\overline{G}$, q)

Input: (1) summary graph $\overline{G} = (S, P)$, (2) query node q
Output: approximate neighbor $\hat{N}_q$ of q in $\hat{G}$

$\hat{N}_q \leftarrow \emptyset$
$\bar{N}_{S_q} \leftarrow$ neighbors of S_q in $\overline{G}$ (i.e., $\forall A$ where $\{A, S_q\} \in P$)
for each $A \in \bar{N}_{S_q}$ **do**
  $\hat{N}_q \leftarrow \hat{N}_q \cup A$
end
$\hat{N}_q \leftarrow \hat{N}_q \setminus \{q\}$
return $\hat{N}_q$

Algorithm 5: Number of Hops (HOPS) on $\overline{G}$

Input: (1) summary graph $\overline{G} = (S, P)$, (2) query node q
Output: distance vector $d \in \mathbb{R}^{|V|}$

$Q \leftarrow \emptyset$; Q.insert(q);
$d \leftarrow -\mathbf{1}$; $d_q \leftarrow 0$ ▷ 1 = one vector of size |V|
while $Q \neq \emptyset$ **do**
  $u \leftarrow Q$.pop()
  for each $v \in$ getNeighbors($\overline{G}$, u) **do**
    if $d_v = -1$ **then**
      Q.insert(w); $d_v \leftarrow d_u + 1$
return d

Algorithm 6: Random Walk with Restart (RWR) on $\overline{G}$

Input: (1) summary graph $\overline{G} = (S, P)$,
(2) random walk probability p, (3) query node q
Output: RWR score vector $r^{new} \in \mathbb{R}^{|V|}$

$V \leftarrow \bigcup_{A \in S} A$
$r^{old} \leftarrow \mathbf{0}$ ▷ 0 = zero vector of size |V|
$r^{new} \leftarrow \frac{1}{|V|} \cdot \mathbf{1}$ ▷ 1 = one vector of size |V|
while $r^{new} \neq r^{old}$ **do**
  $r^{old} \leftarrow r^{new}$; $r^{new} \leftarrow \mathbf{0}$
  for each $u \in V$ **do**
    $\hat{N}_u \leftarrow$ getNeighbors($\overline{G}$, u)
    $r^{new}_v \leftarrow r^{new}_v + \frac{1}{|\hat{N}_u|} r^{old}_u, \forall v \in \hat{N}_u$
  $r^{new} \leftarrow p \cdot r^{new}$
  $r^{new}_q \leftarrow (1 - p \cdot \sum_{v \in V} r^{new}_v)$
return r^{new}

References

[1] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *WWW*, 2004.
[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Stanford InfoLab, Tech. Rep., 1999.
[3] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita, "Compressing graphs and indexes with recursive graph bisection," in *KDD*, 2016.
[4] K. Shin, A. Ghoting, M. Kim, and H. Raghavan, "Sweg: Lossless and lossy summarization of web-scale graphs," in *WWW*, 2019.

[5] J. J. Ramasco, S. N. Dorogovtsev, and R. Pastor-Satorras, "Self-organization of collaboration networks," *Physical review E*, vol. 70, no. 3, p. 036106, 2004.

[6] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *TWEB*, vol. 1, no. 1, p. 5, 2007.

[7] K. Lee, H. Jo, J. Ko, S. Lim, and K. Shin, "Ssumm: Sparse summarization of massive graphs," in *KDD*, 2020.

[8] K. U. Khan, W. Nawaz, and Y.-K. Lee, "Set-based approximate approach for lossless graph summarization," *Computing*, vol. 97, no. 12, pp. 1185–1207, 2015.

[9] M. A. Beg, M. Ahmad, A. Zaman, and I. Khan, "Scalable approximation algorithm for graph summarization," in *PAKDD*, 2018.

[10] M. Riondato, D. García-Soriano, and F. Bonchi, "Graph summarization with quality guarantees," *DMKD*, vol. 31, no. 2, pp. 314–349, 2017.

[11] K. LeFevre and E. Terzi, "Grass: Graph structure summarization," in *SDM*, 2010.

[12] J. Ko, Y. Kook, and K. Shin, "Incremental lossless graph summarization," in *KDD*, 2020.

[13] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *KDD*, 2009.

[14] G. Buehrer and K. Chellapilla, "A scalable pattern mining approach to web graph compression with communities," in *WSDM*, 2008.

[15] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *SIGMOD*, 2012.

[16] W. Henecka and M. Roughan, "Lossy compression of dynamic, weighted graphs," in *FiCloud*, 2015.

[17] I. Tsalouchidou, F. Bonchi, G. D. F. Morales, and R. Baeza-Yates, "Scalable dynamic graph summarization," *TKDE*, vol. 32, no. 2, pp. 360–373, 2018.

[18] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *CSUR*, vol. 51, no. 3, pp. 1–34, 2018.

[19] W. R. Tobler, "A computer movie simulating urban growth in the detroit region," *Economic Geography*, vol. 46, no. sup1, pp. 234–240, 1970.

[20] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *SIGMOD*, 2013.

[21] M. Sarwat, S. Elnikety, Y. He, and M. F. Mokbel, "Horton+ a distributed system for processing declarative reachability queries over partitioned graphs," *PVLDB*, vol. 6, no. 14, pp. 1918–1929, 2013.

[22] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra, "Worst-case optimal join algorithms," *JACM*, vol. 65, no. 3, pp. 1–40, 2018.

[23] A. Khan, G. Segovia, and D. Kossmann, "On smart query routing: for distributed graph querying with decoupled storage," in *ATC*, 2018.

[24] "Supplementary materials: Appendix, code, datasets," https://anonymous.4open.science/r/ICDE22-PeGaSus-A23C/README.md, 2021.

[25] P. D. Grünwald, *The minimum description length principle*. MIT press, 2007.

[26] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.

[27] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, R. E. Tarjan *et al.*, "Time bounds for selection," *Journal of Computer and System Sciences*, vol. 7, no. 4, pp. 448–461, 1973.

[28] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *JSTAT*, vol. 2008, no. 10, p. P10008, 2008.

[29] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

[30] W. W. Hager, D. T. Phan, and H. Zhang, "An exact algorithm for graph partitioning," *Mathematical Programming*, vol. 137, no. 1, pp. 531–556, 2013.

[31] W. W. Hager and Y. Krylyuk, "Graph partitioning and continuous quadratic programming," *SIAM Journal on Discrete Mathematics*, vol. 12, no. 4, pp. 500–523, 1999.

[32] A. Felner, "Finding optimal solutions to the graph partitioning problem with heuristic search," *Annals of Mathematics and Artificial Intelligence*, vol. 45, no. 3, pp. 293–322, 2005.

[33] L. Brunetta, M. Conforti, and G. Rinaldi, "A branch-and-cut algorithm for the equicut problem," *Mathematical Programming*, vol. 78, no. 2, pp. 243–263, 1997.

[34] W. E. Donath and A. J. Hoffman, "Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices," *IBM Technical Disclosure Bulletin*, vol. 15, no. 3, pp. 938–944, 1972.

[35] ——, "Lower bounds for the partitioning of graphs," in *Selected Papers Of Alan J Hoffman: With Commentary*. World Scientific, 2003, pp. 437–442.

[36] B. Rozemberczki and R. Sarkar, "Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models," in *CIKM*, 2020.

[37] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD*, 2005.

[38] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *KAIS*, vol. 42, no. 1, pp. 181–213, 2015.

[39] J. Kunegis, "Konect: the koblenz network collection," in *WWW*, 2013.

[40] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[41] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *WSDM*, 2013.

[42] I. Kabiljo, B. Karrer, M. Pundir, S. Pupyrev, A. Shalita, A. Presta, and Y. Akhremtsev, "Social hash partitioner: a scalable distributed hypergraph partitioner," *arXiv preprint arXiv:1707.06665*, 2017.

[43] A. Awadelkarim and J. Ugander, "Prioritized restreaming algorithms for balanced graph partitioning," in *KDD*, 2020.

[44] H. Tong, C. Faloutsos, and J.-Y. Pan, "Random walk with restart: fast solutions and applications," *KAIS*, vol. 14, no. 3, pp. 327–346, 2008.

[45] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei, "Evaluating geo-social influence in location-based social networks," in *CIKM*, 2012.

[46] Z. Guan, J. Wu, Q. Zhang, A. Singh, and X. Yan, "Assessing and ranking structural correlations in graphs," in *SIGMOD*, 2011.

[47] P. Goodwin and R. Lawton, "On the asymmetry of the symmetric mape," *International journal of forecasting*, vol. 15, no. 4, pp. 405–408, 1999.

[48] C. Spearman, "The proof and measurement of association between two things." 1961.

[49] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[50] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *SIGMOD*, 2008.

[51] K. Lee, J. Ko, and K. Shin, "Slugger: Hierarchical summarization of massive graphs," in *ICDE*, 2022.

[52] K. A. Kumar and P. Efstathopoulos, "Utility-driven graph summarization," *PVLDB*, vol. 12, no. 4, 2018.

[53] M. Hajiabadi, J. Singh, V. Srinivasan, and A. Thomo, "Graph summarization with controlled utility loss," in *KDD*, 2021.

[54] Q. Yong, M. Hajiabadi, V. Srinivasan, and A. Thomo, "Efficient graph summarization using weighted lsh at billion-scale," in *SIGMOD*, 2021.

[55] W. Fan, Y. Li, M. Liu, and C. Lu, "Making graphs compact by lossless contraction," in *SIGMOD*, 2021.

[56] T. Safavi, C. Belth, L. Faber, D. Mottin, E. Müller, and D. Koutra, "Personalized knowledge graph summarization: From the cloud to your pocket," in *ICDM*, 2019.

[57] M. Bae, M. Jeong, and S. Oh, "Label propagation-based parallel graph partitioning for large-scale graph data," *IEEE Access*, vol. 8, pp. 72 801–72 813, 2020.

[58] M. E. Newman, "Spectral methods for community detection and graph partitioning," *Physical Review E*, vol. 88, no. 4, p. 042822, 2013.

[59] C. J. Augeri and H. H. Ali, "New graph-based algorithms for partitioning vlsi circuits," in *ISCAS*, 2004.

[60] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," in *Supercomputing*, 1995.

[61] L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha, "Distributed data placement to minimize communication costs via graph partitioning," in *SSDBM*, 2014.

[62] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *PVLDB*, vol. 3, no. 1-2, pp. 48–57, 2010.

[63] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *ICDM*, 2009.

[64] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD*, 2010.

[65] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *SSDBM*, 2013.

[66] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *PVLDB*, vol. 5, no. 8, 2012.

[67] G. Wang, W. Xie, A. J. Demers, and J. Gehrke, "Asynchronous large-scale graph processing made easy," in *CIDR*, 2013.
[68] S. Hong, S. Salihoglu, J. Widom, and K. Olukotun, "Simplifying scalable graph processing with a domain-specific language," in *CGO*, 2014.
[69] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, "Green-marl: a dsl for easy and efficient graph analysis," in *ASPLOS*, 2012.
[70] J. Huang, D. J. Abadi, and K. Ren, "Scalable sparql querying of large rdf graphs," *PVLDB*, vol. 4, no. 11, pp. 1123–1134, 2011.
[71] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, and G. Lausen, "S2rdf: Rdf querying with sparql on spark," *PVLDB*, vol. 9, no. 10, 2016.
[72] C. Mayer, R. Mayer, J. Grunert, K. Rothermel, and M. A. Tariq, "Q-graph: preserving query locality in multi-query graph processing," in *GRADES-NDA*, 2018.
[73] P. Fuchs, P. Boncz, and B. Ghit, "Edgeframe: Worst-case optimal joins for graph-pattern matching in spark," in *GRADES-NDA*, 2020.
[74] D. Olteanu and J. Závodnỳ, "Size bounds for factorised representations of query results," *TODS*, vol. 40, no. 1, pp. 1–44, 2015.
[75] P. Koutris, P. Beame, and D. Suciu, "Worst-case optimal algorithms for parallel query processing," in *ICDT*, 2016.
[76] D. Rafiei, "Effectively visualizing large networks through sampling," in *VIS*, 2005.
[77] G. Ellis and A. Dix, "The plot, the clutter, the sampling and its lens: occlusion measures for automatic clutter reduction," in *AVI*, 2006.
[78] J. F. Rodrigues Jr, H. Tong, A. J. Traina, C. Faloutsos, and J. Leskovec, "Gmine: a system for scalable, interactive graph visualization and mining," in *VLDB*, 2006.
[79] Y. Wu, R. Jin, and X. Zhang, "Fast and unified local search for random walk based k-nearest-neighbor query in large graphs," in *SIGMOD*, 2014.