# General-purpose Query Processing on Summary Graphs

[Scalable Data Science]

Aris Anagnostopoulos[1] Valentina Arrigoni[2] Francesco Gullo[2] Giorgia Salvatori[3] Lorenzo Severini[2]

[1] Sapienza University, Rome, Italy — aris@diag.uniroma1.it
[2] UniCredit, Milan/Rome, Italy — firstname.lastname@unicredit.eu
[3] Target Reply, Rome, Italy — g.salvatori@reply.it

## ABSTRACT

*Graph summarization* is a well-established problem in large-scale graph data management. Its goal is to produce a *summary graph*, which is a coarse-grained version of a graph, whose use in substitution for the original graph enables downstream task execution and query processing at scale. Despite the extensive literature on graph summarization, still query processing on summary graphs is nowadays accomplished by either reconstructing the original graph, or in a query-specific manner. No general methods exist that operate on the summary graph only, with no graph reconstruction.

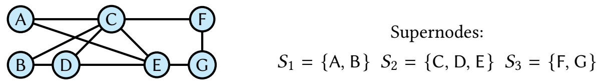
In this paper, we fill this gap, and study for the first time *general-purpose (approximate) query processing on summary graphs*. This is a new important tool to support data-science applications that rely on scalable graph query processing. We set the stage of this problem, by devising basic, yet principled algorithms for it, and thoroughly analyzing their peculiarities and capabilities of performing well in practical contexts, both conceptually and experimentally. The ultimate goal of this work is to make researchers and practitioners aware of this so-far overlooked problem, and define an authoritative starting point to stimulate and drive further research on it.

**PVLDB Reference Format:**
Aris Anagnostopoulos[1] Valentina Arrigoni[2] Francesco Gullo[2] Giorgia Salvatori[3] Lorenzo Severini[2]. General-purpose Query Processing on Summary Graphs. PVLDB, 17(X): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

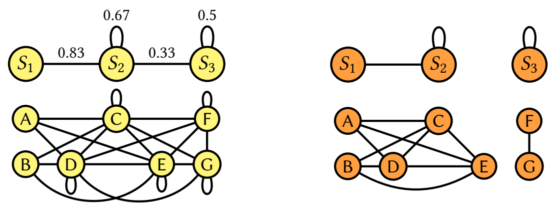
**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/fgullo/GPQPS.

## 1 INTRODUCTION

*Graphs*, that is, sets of entities (vertices) linked to one other (via edges), have become a ubiquitous real-world data representation model [2, 19, 26, 50]. In many domains, graphs are so large that it is hard to directly process them in their raw form. Thus, the problem of reducing the input graph – so as to make graph data management less time/space-demanding without changing algorithms for downstream query processing/task execution – has received considerable attention from researchers, practitioners, and in the industry [8, 59]. Generating more compact versions of a big graph has been addressed from multiple perspectives. The problem that is usually termed *graph compression* (though there is no uniformity in the nomenclature in the literature) aims at developing ad-hoc data structures (and algorithms to manipulate them) to store (and retrieve) an exact representation of a graph using the minimum possible space [8, 11, 12]. *Graph sparsification* (a.k.a., *graph simplification*, or *graph backboning*) reduces a graph by removing edges/vertices [4, 20, 28, 73, 75, 76, 82, 83]. *Graph summarization* produces a coarse-grained version of the original graph – a *summary graph* – by grouping vertices and edges into *supernodes* and *superedges* [7, 35, 47, 48, 52, 54, 56, 58, 64, 70, 74, 77, 81] (see Figure 1). Any of these graph-reduction approaches comes with its own pros/cons, effectiveness, and suitability of application, depending on the application scenario at hand. Graph summarization is mainly appealing as, unlike graph compression, it does not require to employ algorithms (and corresponding software) that have been developed ad-hoc for handling the data structures of a particular compression technique. Also, unlike graph sparsification, it comes with no loss of information, in the sense that at least every original vertex is guaranteed to be part of the output summary graph.



(a) Input graph and partitioning of its vertices into supernodes



(b) Summary graphs (Def. 2.1) and corresponding reconstructed graphs (Def. 2.2) according to S2L [70] (left) and SWeG [74] (right)

**Figure 1: Illustration of graph summarization. In S2L's reconstructed graph edge weights are omitted (for every edge $(u, v)$ they are equal to the weight of the superedge between supernodes containing $u$ and $v$). In this work, we focus on *lossy* graph summarization, where summary graphs lead to reconstructed graphs that do not necessarily correspond to the original graph.**

**Motivation.** Designing methods for producing summary graphs has been an extremely active research area. A plethora of graph-summarization methods exist, based on various design principles, and targeting different graph types and applications (cf. Section 2).

Despite this extensive literature on graph summarization, an aspect that has received limited attention is how to use graph summaries for effective and efficient *approximate* query processing. Existing *general-purpose* query-processing methods on summary graphs (methods that depend on no specific query type) *reconstruct on-the-fly* the input graph [35, 41, 42, 55, 74]. Conversely, existing methods that exploit the summary graph only (without reconstructing the input graph) are query-specific [35, 56, 69, 70]. To the best of

Proceedings of the VLDB Endowment, Vol. 17, No. X ISSN 2150-8097.
doi:XX.XX/XXX.XX

our knowledge, no query-processing methods on summary graphs exist that are general-purpose *and* use the summary graph only.

**Contributions.** In this paper, we fill this gap and study, for the first time, the problem of *general-purpose (approximate) query-processing on summary graphs* (GPQPS). We focus on input *vanilla graphs* (possibly directed and edge-weighted), and summary graphs that have been produced in a *general* (i.e., non-task-specific) and *lossy* way, and whose form is a grouping of original vertices/edges into supernodes and superedges (cf. Definition 2.1). We are interested in approximate graph query-processing methods that (1) are *independent of the query*, (2) exploit the *summary graph only*, (3) *do not access/reconstruct the input graph* (not even partially), and (4) are *agnostic* of the specific method that *generated the summary graphs*.

With these design principles in place, we devise two methods for GPQPS. The first one simply processes queries on a summary graph as if it were a normal graph. The second method interprets a summary graph as an *uncertain graph* [46], and adopts Monte-Carlo-sampling-based query processing on it. Our GPQPS methods are not meant to be sophisticated algorithms facing complex algorithmic/technical challenges. Nor are they required to advance the state of the art, and, as such, achieve good experimental results, perhaps outperforming existing methods. Rather, here we are interested in *setting the stage of* GPQPS. Hence, we purposely focus on basic, immediate (but anyway principled) algorithms, with the main goal of providing conceptual/experimental insights, on both the positive and negative side. However, the promising experimental results achieved by these simple methods attest that they are already a good basis for a satisfactorily effective tackling of GPQPS.

Conversely, we remark that providing a comprehensive comparative evaluation of existing methods (summary-graph-construction methods, graph-query-processing methods, etc.) is not our focus.

**Benefits** of this work include: (1) bringing to the attention of the graph-data-management community GPQPS, a relevant, so-far overlooked problem that focuses on *scalable graph processing*; (2) figuring out the status of the problem, by deeply investigating how and why it can(not) be addressed satisfactorily with simple methodologies; (3) stimulating and driving further research; (4) paving the way for further improving existing graph-summarization methods.

In general, setting the stage of GPQPS helps support the large variety of data-science applications that depend on scalable graph query processing [13–15, 17, 18, 25, 29, 30, 39, 53]. Among others, our study is particularly appealing in an industry setting, as the GPQPS problem enables scalable graph data management, without requiring to change the technology already in place in a company (e.g., graph databases/platforms/software libraries).

**Summary and roadmap.** To summarize, our contributions are:

- We study for the first time the problem of *general-purpose (approximate) query processing on summary graphs* (GPQPS), with the main goal of setting its stage – conceptually, algorithmically, experimentally – and defining a principled starting point on it for researchers/practitioners (Section 3).
- We devise simple algorithms for GPQPS, which are intended to be a reference for the problem and constitute a basic yet authoritative bar, to be raised by future work (Section 3).
- We set up an evaluation methodology that constitutes a benchmark testbed for this and future GPQPS studies (Section 4).
- We carry out extensive experiments on the GPQPS methods, to derive insights on their practical impact and obstacles to an effective employment of them. (Sections 5–6).
- We provide nontrivial directions for further research (Section 7). In this regard, we remark that such directions are not as high-level as the usual ones of scientific papers; rather, they are a concrete yet highly specific roadmap that we could draw thanks to all the previously listed contributions.

All in all, we believe this work can be a key landmark for a novel relevant tool for scalable data-science-oriented workloads on graphs.

## 2 PRELIMINARIES AND RELATED WORK

The problem of *graph summarization* [59] takes as input a *graph* that, in the general case, is assumed to be *directed* and *edge weighted*. Formally, we are given a triple $G = (V, E, w)$, where $V$ is a set of *vertices*, $E \subseteq V \times V$ is a set of *edges*, (i.e., *ordered* pairs of vertices), and $w\colon E \rightarrow \mathbb{R}_{>0}$ is a function assigning a positive real-valued weight to every edge. Should $G$ be *undirected*, $E$ is defined as a set of *unordered* vertex pairs. Should $G$ be *unweighted*, then $w(e) = 1$, $\forall e \in E$. A graph $G = (V, E, w)$ may alternatively, yet equivalently, be represented with its *adjacency matrix* $M \in \mathbb{R}^{|V| \times |V|}$, where $M[u, v] = w(u, v)$, if $(u, v) \in E$, $M[u, v] = 0$, otherwise, $\forall u, v \in V$.

Graph summarization relies on the key notion of *summary graph*:

*Definition 2.1 (Summary graph).* A *summary graph* (or, simply, a *summary*) of a given graph $G = (V, E, w)$ is a *directed* and (possibly) *edge-weighted* graph $\mathcal{G} = (\mathcal{S}, \mathcal{E}, \omega)$, with vertices $\mathcal{S}$, edges $\mathcal{E} \subseteq \mathcal{S} \times \mathcal{S}$, and edge-weighting function $\omega\colon \mathcal{E} \rightarrow \mathbb{R}_{>0}$, such that every vertex $u \in V$ is assigned to one and only one $S \in \mathcal{S}$: $\forall S \in \mathcal{S}\colon S \subseteq V$, $\forall S, T \in \mathcal{S}, S \neq T\colon S \cap T = \emptyset$, $\bigcup_{S \in \mathcal{S}} = V$. We term vertices and edges of a summary *supernodes* and *superedges*, respectively. The supernode to which a vertex $u \in V$ belongs is denoted by $S_u$. $E(S, T) = \{(u, v) \in E \mid u \in S, v \in T\}$ and $\omega(S, T) = \sum_{e \in E(S,T)} w(e)$ denote the edges and the overall edge weight between all the vertices of supernodes $S$ and $T$, respectively.

Simply speaking, a summary $\mathcal{G} = (\mathcal{S}, \mathcal{E}, \omega)$ of a graph $G = (V, E, w)$ represents a *partition* of the vertices in $V$ into $\mathcal{S}$ supernodes. Superedges $\mathcal{E}$ are pairs $(S, T)$ of supernodes, which are unordered if the original graph is undirected, and ordered otherwise. Superedges may possibly be assigned a real-valued weight $\omega(S, T)$. However, a summary can simply be unweighted [41]. This case is modeled by setting $\omega(S, T) = 1$, for all $(S, T) \in \mathcal{E}$. Summary graphs admit *self-loops*, that is, superedges of the form $(S, S)$. In general, a superedge $(S, T)$, along with its possible weight $\omega(S, T)$, is meant to concisely represent the information about the various edges in $G$ that connect a vertex in $S$ and some other vertex in $T$.

From a summary we can derive the so-called *reconstructed graph*:

*Definition 2.2 (Reconstructed graph).* Given a summary $\mathcal{G}$ of a graph $G = (V, E, w)$, the *reconstructed graph* is a graph $G' = (V, E', w')$ that is defined by properly exploiting $\mathcal{G}$.

The definition of reconstructed graph is voluntarily left general here, as it depends on the specific graph-summarization method. Broadly speaking, the problem of graph summarization consists in finding a summary of a given graph such as to optimize some criteria that are typically aimed at both (1) minimizing the difference between the original graph and the reconstructed graph, and (2) keeping the size of graph summarization's output low [59]. Specific formulations of graph summarization fall into two main classes: *size-driven* and *utility-driven*, which we overview below.

Figure 1 illustrates the notions in Definitions 2.1–2.2.

**Lossless vs. lossy graph summarization.** Regardless of the formulation, graph summarization can be either *lossless* or *lossy*. In the first case, it is guaranteed that the original graph can be recovered *exactly* from the output of graph summarization. To guarantee such an exactness, several graph-summarization methods yield a *correction set* together with a summary graph [48, 64, 74, 81], that is, edges to be added to or removed from the reconstructed graph so as to make it actually correspond to the input graph.

Conversely, a lossy graph-summarization output is not required to allow for exactly recovering the input graph. As such, lossy graph summarization typically outputs the summary only, with no correction set. In this work, *we focus on lossy graph summarization.* Thus, we will not address the notion of correction set further.

**Size-driven graph summarization.** Let $d(\mathcal{G}, G)$ be an *error* function that quantifies how good a summary $\mathcal{G}$ is for graph $G$, in terms of the difference between the reconstructed graph $G'$ and $G$. Given an integer $\kappa > 0$, *size-driven graph summarization* looks for a summary $\mathcal{G}$ with $\kappa$ supernodes that minimizes $d(\mathcal{G}, G)$ [7, 54, 56, 58, 69, 70, 77]. Existing approaches of size-driven graph summarization differ from each other in the specific error function employed.

*$\ell_p$-reconstruction-error* is a popular error function [7, 56, 69, 70]. Let $pr(S,T)$ and $\overline{\omega}(S,T)$ be the *probability of existence* and the *average weight* of an edge between supernodes $S$ and $T$, respectively:

$$pr(S,T) = |E(S,T)|/(|S| \cdot |T|), \quad \overline{\omega}(S,T) = \omega(S,T)/|E(S,T)|\,. \quad (1)$$

The *lifted adjacency matrix* given $\mathcal{G}$ is defined as a $|V| \times |V|$ matrix $M^{\uparrow}$ whose $M^{\uparrow}[u,v]$ cells contain the *expected weight* of an edge between the supernodes of $u$ and $v$: $M^{\uparrow}[u,v] = pr(S_u, S_v) \cdot \overline{\omega}(S_u, S_v)$. The $\ell_p$-reconstruction error $\text{err}_p$ is defined as the entry-wise $p$-norm of the difference between the adjacency matrix $M$ of the input graph and $M^{\uparrow}$, that is $\text{err}_p(\mathcal{G}, G) = \|M - M^{\uparrow}\|_p$.

LeFevre and Terzi [56] deal with $\text{err}_1$ and propose a greedy heuristic algorithm that resembles an agglomerative hierarchical clustering using Ward's method [80]. Riondato *et al.* [69, 70] establish a connection between $\text{err}_p$-based graph summarization and *$\ell_p^p$-clustering*, and devise algorithms with constant-factor approximation guarantees for $\text{err}_1$ and $\text{err}_2$. Among Riondato *et al.*'s algorithms, the S2L one targets $\text{err}_2$ and employs $k$-median clustering, which is tackled by Lloyd's iterative approach [60].

Beg *et al.* [7] and Lee *et al.* [54] devise algorithms that are mainly focused on improving the efficiency of the algorithms of [56, 69, 70].

*Other error functions* adopted in the literature include *cut-norm error* [70], *representation error* [58], and *path-based error* [77, 84].

**Utility-driven graph summarization** is the dual formulation of the size-driven counterpart: given a *utility function* $u(\mathcal{G}, G)$, which expresses how close the graph reconstructed from summary $\mathcal{G}$ is to the input graph $G$, find a summary $\mathcal{G}$ of minimum size, subject to the constraint that $u(\mathcal{G}, G)$ is no less than a given threshold [35, 47, 48, 52, 64, 74, 81]. Navlakha *et al.* [64] adopt a utility function based on the *representation error* (see above). Shin *et al.* [74] introduce SWeG, a parallel algorithm for Navlakha *et al.*'s formulation. Yong *et al.* [81] further improve SWeG through *locality sensitive hashing* [37]. Khan *et al.* [47] and Ko *et al.* [48] devise a set-based method and an incremental algorithm, respectively, for a *lossless* variant of Navlakha *et al.*'s formulation. Khumar and Efstathopoulos [52] and Hajiabadi *et al.* [35] define the utility in terms of the *importance* of the edges in the reconstructed graph.

**Summarizing weighted/directed graphs.** Graph-summarization methods typically handle unweighted and undirected graphs. A few methods (can be easily adapted to) work for edge-weighted graphs [56, 69, 70, 77, 84]. As for directed graphs, to our knowledge, only Riondato *et al.* [70] handle them: they propose to decompose the input adjacency matrix into the sum of a *symmetric* matrix and a *skew-symmetric* matrix, and compute two summaries for either such matrices. The resulting summaries still come with (constant-factor) approximation guarantees, as the theoretical properties of their algorithms carry over to skew-symmetric matrices.

**Query processing on summary graphs.** A major use of summaries is to reduce time/space of graph query processing.

Existing *general-purpose* (approximate) query-processing methods (methods that depend on no specific query type) assume that the basic primitive for graph queries corresponds to retrieving the neighborhood of a vertex. Based on this, such methods *reconstruct on-the-fly* a neighborhood from the summary, while processing the target query [35, 41, 42, 55, 74]. This strategy is beneficial for space complexity, as it keeps in memory the summary only. However, it gives no speedup in runtime, as reconstructed neighborhoods are typically larger than actual neighborhoods, at least on average.[1]

Conversely, existing methods to process graph queries using the summary only (without reconstructing the input graph) are ad-hoc defined for a few specific queries, for instance, degree [56, 69, 70], triangle counting [69, 70], eigenvector centrality [56, 69, 70], or, in the case of lossless summaries, PageRank and shortest path [35].

To the best of our knowledge, the problem of *general-purpose* query-processing on summary graphs *without reconstructing the input graph* has never been tackled. *In this paper, we fill this gap.*

**Other tangentially related literature** includes experimental evaluations with completely different goals with respect to the ones of this work [9, 41], graph-summarization methods producing summaries with a form other than the one in Definition 2.1 [51, 55], problem variants of vanilla graph summarization [32, 38, 42, 43, 79, 85], query-specific graph summarization [22–24, 36, 62, 71], and related but different problems, such as *graph compression* [8, 11, 12], *graph sparsification* [4, 20, 28, 73, 75, 76, 82, 83], *graph clustering* [3, 72]. A more detailed discussion of this literature is in [5] (Appendix A).

## 3 GENERAL-PURPOSE QUERY PROCESSING BY EXPLOITING SUMMARY GRAPHS ONLY

**Graph queries.** A *(graph) query* $Q$ is a computable function whose input is a graph $G = (V, E, w)$ and *(query) context* $C$, and whose output is an object from a certain domain $O_Q$.[2] Context $C$ identifies the complementary input of the query. It may correspond to sets/pairs of vertices, subgraphs, functions, numerical values, and

[1]Experimental evidence of this claim is reported in [55], for PageRank, triangle counting, breadth first search, and shortest path queries.

[2]In this work, we consider solely queries that operate on graphs. Thus, we hereinafter use the terms "graph query" and "query" interchangeably. Moreover, our notion of query corresponds to what is termed "query class" in some existing works [22–24].

**Algorithm 1** Naïve-GPQPS

```
Input: graph $G = (V, E, w)$; summary $\mathcal{G}$ of $G$; graph query $Q$; query context $C$;
    $\mathbf{c} \in \mathbb{R}^d$ (if $O_Q = \mathbb{R}^d$)
Output: approximate answer $\widetilde{Q}(G, \mathcal{G})$
1: $C_{\mathcal{G}} \leftarrow$ compute summary-aware context information from $C$ and $\mathcal{G}$   ▷ Def. 3.2
2: $Q(\mathcal{G}, C_{\mathcal{G}}) \leftarrow$ compute summary-processed query answer   ▷ Def. 3.3
3: if $O_Q = \mathbb{R}^d$ then
4:     $\widetilde{Q}(G, \mathcal{G}) \leftarrow \mathbf{c} \circ Q(\mathcal{G}, C_{\mathcal{G}})$
5: else if $O_Q = 2^{2^V}$ then
6:     $\widetilde{Q}(G, \mathcal{G}) \leftarrow \{\{\bigcup_{S \in \mathsf{S}} S\} \mid \mathsf{S} \in Q(\mathcal{G}, C_{\mathcal{G}})\}$
7: end if
```

so on, or it can also be empty. The output of a query $Q$ (with context $C$) on a graph $G$ is denoted by $Q(G, C)$, and is alternatively termed the *answer* of $Q$ on $G$. The answer of a query can be a Boolean, a numerical value, a set of vertices, a subgraph, a partition of the vertices, and so on. For instance, *global* queries computing numerical statistics on $G$ (e.g., number of triangles, clustering coefficient, diame.ter) have $C = \emptyset$ and $O_Q = \mathbb{R}$. *Node embedding* queries take a vertex $u$ as a context $C$, and output a $d$-dimensional numerical vector representing the embedding of $u$ (thus, $O_Q = \mathbb{R}^d$). *Inner-most core* queries have $C = \emptyset$, and the output is the vertices of the $k$-core [6] of $G$ with the highest $k$ (thus, $O_Q = 2^V$). In *top-ranked centrality* queries, $C$ corresponds to an integer $s$, and the output is the top-$s$–ranked vertices according to a certain centrality (thus, $O_Q = 2^V$). In *community detection* queries, $C$ either contains the parameters of the algorithm to be used (e.g., number of communities), or is empty if the algorithm at hand is parameterless, and the output is a partition of $V$ (thus, $O_Q = \mathbf{B}_V$, where $\mathbf{B}_V$ denotes the set of all possible partitions of $V$). In this work, we restrict our study to query answers that are either numerical or sets/partitions of vertices (i.e., $O_Q \in \{\mathbb{R}^d, 2^V, \mathbf{B}_V\}$). Adaptation of our methodologies to other forms of answer is possible with relatively low effort. Anyway, we defer a more rigorous study of this to future work.

**Problem statement.** We focus on a scenario where the answer to a query is *approximated by exploiting solely a summary* $\mathcal{G}$ of a graph $G$, *without accessing* $G$ *at all*. Also, we require query processing to be *agnostic* of both the specific query and the graph-summarization technique that has produced $\mathcal{G}$. Specifically, we are interested in:

*Definition 3.1 (Summary-based approximate query answer).* Given a graph $G$, a summary $\mathcal{G}$ of $G$, and a query $Q$ on $G$ with context $C$, a *summary-based approximate answer* to $Q$ on $\mathcal{G}$ – denoted $\widetilde{Q}(G, \mathcal{G})$ – is an approximation of $Q(G, C)$ obtained by exploiting $\mathcal{G}$ only.

The problem we tackle in this work is as follows:

Problem 1 (General-Purpose Query Processing on Summary graphs (GPQPS)). *Given a summary $\mathcal{G}$ of a graph $G$, and a query $Q$ on $G$ with context $C$, compute $\widetilde{Q}(G, \mathcal{G})$ that is the closest to $Q(G, C)$.*

Simply speaking, Problem 1 asks for summary-based query answers which approximate well the true answer to the given query. In the remainder of this section, we present algorithms for Problem 1.

**Naïve-GPQPS algorithm.** As a very first, simple method for Problem 1 we consider the one where a query $Q$ is processed on summary $\mathcal{G}$ as if it were a normal graph, with the only precaution of letting each vertex $u$ in the input graph $G$ conceptually be identified with

**Algorithm 2** Probabilistic-GPQPS

```
Input: graph $G = (V, E, w)$; summary $\mathcal{G} = (\mathcal{S}, \mathcal{E}, \omega)$ of $G$; function $\pi: \mathcal{E} \to (0, 1]$;
    integer $K$; graph query $Q$; query context $C$; clustering-aggregation algorithm
    AGG (if $O_Q = \mathbf{B}_V$)
Output: approximate answer $\widetilde{Q}(G, \mathcal{G})$
1: $\mathcal{W}_1, \ldots, \mathcal{W}_K \leftarrow$ sample $K$ possible worlds from $\mathcal{G}_\pi = (\mathcal{S}, \mathcal{E}, \pi)$
2: $\widetilde{Q}(G, \mathcal{W}_i) \leftarrow$ Naïve-GPQPS($G, \mathcal{W}_i, Q, C$), for all $i = 1, \ldots, K$   ▷ Alg. 1
3: if $O_Q = \mathbb{R}^d$ then
4:     $\widetilde{Q}(G, \mathcal{G}) \leftarrow \frac{1}{K} \sum_{i=1}^{K} \widetilde{Q}(G, \mathcal{W}_i)$
5: else if $O_Q = 2^V$ then
6:     $\widetilde{Q}(G, \mathcal{G}) \leftarrow \bigcap_{i=1}^{K} \widetilde{Q}(G, \mathcal{W}_i)$
7: else if $O_Q = \mathbf{B}_V$ then
8:     $\widetilde{Q}(G, \mathcal{G}) \leftarrow$ run AGG on $\{\widetilde{Q}(G, \mathcal{W}_i)\}_{i=1}^{K}$   ▷ [31, 33]
9: end if
```

the supernode $S_u$ of $\mathcal{G}$ it belongs to, and vice versa. To be more precise, let us introduce the following notions:

*Definition 3.2 (Summary-aware query context).* Given context $C$ of a query $Q$ on a graph $G$, the *summary-aware query context* of $C$ on a summary $\mathcal{G}$ of $G$ – denoted by $C_{\mathcal{G}}$ – is a copy of $C$ where every vertex $u \in C$ is replaced with the supernode $S_u$ of $\mathcal{G}$ it belongs to.

*Definition 3.3 (Summary-processed query answer).* Let $Q$ be a query on a graph $G$ with context $C$, $\mathcal{G}$ be a summary of $G$, and $C_{\mathcal{G}}$ be the summary-aware context of $C$ on $\mathcal{G}$. The *summary-processed answer* to $Q$ on $\mathcal{G}$ – denoted by $Q(\mathcal{G}, C_{\mathcal{G}})$ – is the answer obtained by processing $Q$ on $\mathcal{G}$ with context $C_{\mathcal{G}}$ as if $\mathcal{G}$ were a normal graph.

The simple method considered here is termed Naïve-GPQPS, and is outlined as Algorithm 1. It computes a summary-processed query answer $Q(\mathcal{G}, C_{\mathcal{G}})$, then derives a summary-based approximate query answer $\widetilde{Q}(G, \mathcal{G})$, distinguishing two cases, based on $Q$'s output domain. If the answer to $Q$ is numerical (e.g., a global statistic, such as clustering coefficient or diameter, or a property of a vertex, such as a centrality score or an embedding vector), the answer obtained on the summary is interpreted as is as the ultimate answer to the given query, up to a multiplicative factor $\mathbf{c} \in \mathbb{R}^d$, that is, $\widetilde{Q}(G, \mathcal{G}) = \mathbf{c} \circ Q(\mathcal{G}, C_{\mathcal{G}})$, where "$\circ$" denotes Hadamard (i.e., element-wise) product. Instead, if the answer is in the form of a set of subsets of vertices, then $Q(\mathcal{G}, C_{\mathcal{G}})$ will be a set of subsets of supernodes, and $\widetilde{Q}(G, \mathcal{G})$ corresponds to a set of subsets of vertices derived by taking the union of all the supernodes in each of the output supernode subsets. Note that $2^V \subset 2^{2^V}$, $\mathbf{B}_V \subset 2^{2^V}$, thus the latter covers also the special cases where $Q$'s answer is a single subset of vertices (e.g., the inner-most core) or a partition of the vertices (e.g., a community structure).[3]

**Probabilistic-GPQPS algorithm.** The probabilistic interpretation that is at the basis, among others, of $\text{err}_p$ (cf. Section 2) suggests to model a summary as an *uncertain* (or *probabilistic*) *graph*, that is, a graph whose edges are assigned a probability of existence:

*Definition 3.4 (Uncertain graph).* An *uncertain* (or *probabilistic*) *graph* is a triple $\mathbb{G} = (V, E, \pi)$, where $V$ is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $\pi: E \to (0, 1]$ is a function assigning existence probabilities to edges. According to the *possible-world* semantics [1, 21], an uncertain graph $\mathbb{G} = (V, E, \pi)$ is interpreted

[3] $2^V \subset 2^{2^V}$ slightly abuses notation: $\forall x \in 2^V$, $\{x\} \in 2^{2^V}$.

as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ possible deterministic graphs (*worlds*), each defined by a subset of $E$. Assuming independence among edge probabilities [16, 40, 44, 45, 49, 61, 66–68], the probability of observing any possible world $G = (V, E_G)$ drawn from $\mathbb{G}$ is $\Pr(G) = \prod_{e \in E_G} \pi(e) \prod_{e \in E \setminus E_G} (1 - \pi(e))$.

Our second GPQPS method, Probabilistic-GPQPS (Algorithm 2), is based on the interpretation of a summary as an uncertain graph. It takes as input a summary $\mathcal{G} = (\mathcal{S}, \mathcal{E}, \omega)$ and a function $\pi\colon \mathcal{E} \rightarrow (0, 1]$ assigning existence probabilities to superedges. Function $\pi$ can be defined, for example, as the expected number of edges between two supernodes (as in $\mathrm{err}_p$, see Equation (1)). However, the algorithm is independent of the definition of $\pi$. Also, no relationship is required between $\pi$ and $\mathcal{G}$: $\pi$ may have been (as, for example, the S2L graph-summarization method, cf. Section 2), or may have not been (like, e.g., the SWeG method) exploited for computing $\mathcal{G}$. Given $\mathcal{G}$ and $\pi$, Probabilistic-GPQPS defines an *uncertain summary graph* as $\mathcal{G}_\pi = (\mathcal{S}, \mathcal{E}, \pi)$, and adopts a consolidated Monte-Carlo–sampling query-processing methodology on uncertain graphs, which consists in processing a query on a set of random possible worlds drawn from $\mathcal{G}_\pi$, and then aggregating the answers obtained for each such worlds [46]. A possible world $\mathcal{W} = (\mathcal{S}, \mathcal{E}_{\mathcal{W}}, \omega)$ is drawn from $\mathcal{G}_\pi$ by generating a number $r(S, T) \in [0, 1]$ uniformly at random for every superedge $(S, T) \in \mathcal{E}$, and adding $(S, T)$ to $\mathcal{E}_{\mathcal{W}}$ if $r(S, T) \leq \pi(S, T)$. On every world $\mathcal{W}$ the query is processed with Naïve-GPQPS (Algorithm 1). Query-answer aggregation is performed by: averaging over all individual answers (for numerical queries), or taking the intersection of all the vertex sets obtained as an answer on each world (for queries returning a single vertex set), or via *clustering aggregation* [31, 33] (for queries returning a vertex partition).

## 4 EXPERIMENTAL METHODOLOGY

**Datasets.** We experiment with public datasets, most of which traditionally used in the graph-summarization literature [42, 48, 54, 70, 74] (Table 1). Regarding the ones composed of multiple connected components (LastFM, Enron), we retain only the giant component.

**Graph-summarization methods.** To generate summaries, we selected one representative per category of graph-summarization approaches (see Section 2), namely size-driven S2L [70] and utility-driven SWeG [74]. These two methods differ in the type of input graph and output summary: S2L handles possibly directed/weighted graphs and produces weighted summaries, whereas SWeG handles only undirected and unweighted graphs, and yields unweighted summaries. This way, we cover a reasonably complete and diverse spectrum of testbeds for our GPQPS methods.

We used SWeG's unofficial implementation available at [35]. As for S2L, though it is capable of handling weighted/directed graphs, its official implementation [70] supports unweighted and undirected graphs only. Thus, we also developed a custom S2L's implementation, and used it on weighted, directed, and smaller graphs. Instead, we used S2L's official implementation for the remaining graphs.

**Queries.** We selected queries for each of the classes discussed in Section 3: *clustering coefficient*, representative of *numerical* queries (answer $\in \mathbb{R}$); *community detection*, representative of *partitioning* queries (answer $\in \mathbf{B}_V$); *top-ranked centrality* and *core decomposition*, representatives of *vertex-set* queries (answer $\in 2^V$).

**Table 1: Characteristics of the selected datasets ($|V|$: #vertices; $|E|$: #edges) and summaries ($|\mathcal{S}|$: #supernodes; $|\mathcal{E}|$: #superedges; #CCs: #connected components; |GCC|: size of the giant connected component (%); #self: #self-loops; Cpr.: compression defined as $1 - (|V| + |\mathcal{S}| + |\mathcal{E}|)/(|V| + |E|)$ (%)).**

| | *characteristics* | | *S2L summaries* | | | *SWeG summaries* | | |
|---|---|---|---|---|---|---|---|---|
| Facebook [57] | | $\lvert\mathcal{S}\rvert$: | 350 | 500 | 750 | 708 | 977 | 1 168 |
| | $\lvert V\rvert$: 4 039 | $\lvert\mathcal{E}\rvert$: | 8 462 | 14 390 | 24 327 | 664 | 2 157 | 4 448 |
| | $\lvert E\rvert$: 88 234 | #CCs: | 1 | 1 | 1 | 240 | 18 | 38 |
| | undirected | \|GCC\|: | 100% | 100% | 100% | 20% | 77% | 79% |
| | unweighted | #self: | 264 | 280 | 323 | 11 | 22 | 36 |
| | | Cpr.: | 86% | 79% | 68% | 94% | 92% | 90% |
| LastFM [57] | | $\lvert\mathcal{S}\rvert$: | 500 | 750 | 1 000 | 3 375 | 3 568 | 3 821 |
| | $\lvert V\rvert$: 7 429 | $\lvert\mathcal{E}\rvert$: | 6 835 | 10 350 | 12 768 | 1 217 | 1 550 | 1 997 |
| | $\lvert E\rvert$: 27 788 | #CCs: | 1 | 1 | 1 | 2 169 | . 2 076. | 1 966 |
| | undirected | \|GCC\|: | 100% | 100% | 100% | 1% | 11% | . 26% |
| | unweighted | #self: | 133 | 176 | 185 | 6 | 10 | 17 |
| | | Cpr.: | 58% | 47% | 40% | 66% | 64% | 62% |
| Enron [57] | | $\lvert\mathcal{S}\rvert$: | 1 000 | 1 500 | 2 000 | 22 832 | 23 957 | 24 371 |
| | $\lvert V\rvert$: 33 696 | $\lvert\mathcal{E}\rvert$: | 50 872 | 68 405 | 81 444 | 10 160 | 28 825 | 44 086 |
| | $\lvert E\rvert$: 180 811 | #CCs: | 1 | 1 | 1 | 14 537 | 7 159 | 6 056 |
| | undirected | \|GCC\|: | 100% | 100% | 100% | 19% | 64% | 70% |
| | unweighted | #self: | 231 | 303 | 397 | 557 | 702 | 749 |
| | | Cpr.: | 68% | 61% | 54% | 69% | 60% | 52% |
| Gnutella [57] | | $\lvert\mathcal{S}\rvert$: | 500 | 750 | 1 000 | – | – | – |
| | $\lvert V\rvert$: 6 301 | $\lvert\mathcal{E}\rvert$: | 2 016 | 3 383 | 5 000 | – | – | – |
| | $\lvert E\rvert$: 20 777 | #CCs: | 1 | 1 | 1 | – | – | – |
| | directed | \|GCC\|: | 100% | 100% | 100% | – | – | – |
| | unweighted | #self: | 5 | 6 | 12 | – | – | – |
| | | Cpr.: | 60% | 52% | 45% | – | – | – |
| Ubuntu [27] | | $\lvert\mathcal{S}\rvert$: | 350 | 500 | 750 | – | – | – |
| | $\lvert V\rvert$: 3 024 | $\lvert\mathcal{E}\rvert$: | 35 346 | 55 276 | 82 589 | – | – | – |
| | $\lvert E\rvert$: 144 094 | #CCs: | 1 | 1 | 1 | – | – | – |
| | undirected | \|GCC\|: | 100% | 100% | 100% | – | – | – |
| | weighted | #self: | 67 | 86 | 110 | – | – | – |
| | | Cpr.: | 74% | 60% | 41% | – | – | – |
| AS-Skitter [57] | | $\lvert\mathcal{S}\rvert$: | 1 000 | 10 000 | 25 000 | 1 087 434 | 1 087 430 | 1 103 931 |
| | $\lvert V\rvert$: 1 696 415 | $\lvert\mathcal{E}\rvert$: | 39 064 | 461 826 | 983 129 | 1 704 954 | 1 857 562 | 2 073 416 |
| | $\lvert E\rvert$: 11 095 298 | #CCs: | 4 | 38 | 73 | 204 299 | 200 833 | 181 766 |
| | undirected | \|GCC\|: | 99.6% | 99.7% | 99.6% | 70.1% | 72.7% | 75.9% |
| | unweighted | #self: | 671 | 5 306 | 10 746 | 11 936 | 11 955 | 12 325 |
| | | Cpr.: | 86% | 83% | 79% | 65% | 64% | 62% |

Clustering coefficient refers to the average of the individual clustering coefficient of every vertex in the graph. Community detection's output is a partition of the input vertices into communities, computed with the well-established Louvain algorithm [10]. Centrality queries include *PageRank*, and *closeness*. As for core decomposition [6], we consider the vertex set corresponding to the union of the top-$z$ *inner-most cores* ($z \in \{1, 2, 5\}$). For all our queries but core decomposition, we used the corresponding implementations in `NetworkX` [34]. For core decomposition, we used a custom implementation of classic Batagelj and Zaversnik's algorithm [6] (as `NetworkX`'s implementation does not support weighted graphs).

**GPQPS methods.** We evaluate Naïve-GPQPS and Probabilistic-GPQPS (Algorithms 1–2). On S2L summaries, we test two variants of Naïve-GPQPS: "N" and "Nw," which either consider (Nw) or discard (N) superedge weights, and three variants of Probabilistic-GPQPS: "P," "Pa," and "Pe," depending on the weight considered for every superedge $(S, T)$ in the sampled worlds: no weight (P), average weight $\overline{\omega}(S, T)$ (Pa), and expected weight $pr(S, T) \cdot \overline{\omega}(S, T)$ as defined in Eq. (1) (Pe). On summaries computed with SWeG, we consider N only, as SWeG produces no superedge weights. We defer to future work the investigation of the other GPQPS methods/variants on SWeG's summaries coupled with other-party superedge weightings (e.g., S2L's one). Unless otherwise specified, for all the Naïve-GPQPS variants, we set $\mathbf{c} = |\mathcal{S}_u|$ for PageRank queries of every vertex $u$, and $\mathbf{c} = 1$ for all other numerical queries. Instead, parameters for all the Probabilistic-GPQPS variants are: $K = 100$; $\pi$ is set equal to function *pr* in Eq. (1); for clustering aggregation in vertex-set queries we use (a custom implementation of) Topchy *et al.*'s algorithm [78].

**Assessment criteria.** We assess accuracy, efficiency, and space requirements of the GPQPS methods. Efficiency is measured simply in terms of query-processing runtime. Regarding space, note that a summary requires $O(|V| + |\mathcal{S}| + |\mathcal{E}|)$ space (to store supernodes, superedges, vertex-to-supernode assignment), as opposed to $O(|V| + |E|)$ space required by the original graph. Hence, we assess the gain in space of GPQPS methods by means of compression percentage $1 - (|V| + |\mathcal{S}| + |\mathcal{E}|)/(|V| + |E|)$ (the higher, the better).

Accuracy depends on the specific query. For clustering coefficient, we measure the *relative error* of the query answer obtained via GPQPS w.r.t. the answer in the original graph. For community-detection queries, we consider the *relative error* of the *modularity* [65] of the communities yielded by GPQPS w.r.t. the modularity of the communities computed in the original graph.

Regarding centrality queries, a direct comparison of centrality scores is not really meaningful. Instead, we compare the centrality rank (ties broken randomly) of all the vertices obtained in the original graph and via GPQPS (for the centralities considered in this work, higher scores correspond to better ranks). More specifically, for integer $g$ (resp. $s$) $\in [1..|V|]$, we take the centrality score $x_g$ (resp. $x_s$) of the $g$th (resp. $s$th) vertex in the centrality ranking within the original graph (resp. via GPQPS), and define the $g$-set (resp. $s$-set) as the set of vertices with centrality score no less than $x_g$ (resp. $x_s$). To ultimately assess a centrality query, we measure *precision* $P = |g\text{-set} \cap s\text{-set}|/|s\text{-set}|$ and *recall* $R = |g\text{-set} \cap s\text{-set}|/|g\text{-set}|$ of the $s$-set w.r.t. the "ground-truth" $g$-set. In our experiments, we set $g = 100$, and vary $s \in \{100, 200, 500\}$. The rationale here is to assess to what extent the top-$g$ central vertices (according to the original graph) are part of the top-$s$ central vertices (according to GPQPS). On AS-Skitter, computing closeness for all the vertices is infeasible, due to its large size. Thus, on that dataset, we compute/evaluate closeness on a subset of 1k randomly-sampled vertices.

As for core decomposition, we assess it similarly to centrality. Specifically, we take the inner-most core $C^*$ of the graph as a ground-truth vertex set, the top-$z$ inner-most cores $\{C_z \mid z \in \{1, 2, 5\}\}$ computed by GPQPS, and ultimately measure precision $P = |C_z \cap C^*|/|C_z|$ and recall $R = |C_z \cap C^*|/|C^*|$, for all $z \in \{1, 2, 5\}$.

**Testing environment.** We run all our experiments *with no parallelization* on a single machine equipped with a Dual 20-Core Intel Xeon E5-2698 v4 2.20GHz CPU and 512GB RAM.

**Reproducibility.** This work's companion repo [5] contains the code of the GPQPS methods, of the assessment methodologies, and of our custom implementations of S2L, Batagelj and Zaversnik's core decomposition, Topchy *et al.*'s clustering aggregation, along with (links to) datasets and summaries, and all the parameters and details to generate summaries and run the experimental pipelines.

## 5 EXPERIMENTAL RESULTS

**Storage space.** We report the compression percentages of the various summaries in Table 1. Those percentages apparently attest how

**Table 2: GPQPS results for clustering coefficient and community detection queries. "A" stands for actual query processing on the original graph.**

(a) S2L [70] summaries

| | method | *clustering coefficient* | | | | | | *community detection (modularity)* | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *relative error* | | | *runtime (s)* | | | *relative error* | | | *runtime (s)* | | |
| Facebook | *#supernodes:* | 350 | 500 | 750 | 350 | 500 | 750 | 350 | 500 | 750 | 350 | 500 | 750 |
| | *#superedges:* | 8k | 14k | 24k | 8k | 14k | 24k | 8k | 14k | 24k | 8k | 14k | 24k |
| | A | – | | | 1.69 | | | – | | | .86 | | |
| | N | .263 | .21 | .138 | .121 | .261 | .548 | .124 | .101 | .038 | .06 | .11 | .24 |
| | Nw | .466 | .367 | .284 | .864 | 1.74 | 3.01 | .116 | .043 | .043 | .06 | .11 | .23 |
| | P | .034 | .025 | .01 | 1.12 | 3.3 | 7.45 | .069 | .047 | .052 | 1.11 | 1.88 | 4.09 |
| | Pa | .034 | .025 | .01 | 5.92 | 18.6 | 42.6 | .058 | .045 | .052 | .97 | 2.02 | 3.81 |
| | Pe | .207 | .161 | .128 | 6.26 | 19.4 | 43.4 | .115 | .106 | .065 | .99 | 1.83 | 4.03 |
| LastFM | *#supernodes:* | 500 | 750 | 1k | 500 | 750 | 1k | 500 | 750 | 1k | 500 | 750 | 1k |
| | *#superedges:* | 7k | 10k | 13k | 7k | 10k | 13k | 7k | 10k | 13k | 7k | 10k | 13k |
| | A | – | | | .215 | | | – | | | .61 | | |
| | N | 1.67 | 1.31 | 1.13 | .069 | .111 | .141 | .342 | .309 | .262 | .06 | .11 | .13 |
| | Nw | .002 | .075 | .048 | .263 | .396 | .452 | .329 | .299 | .268 | .16 | .12 | .29 |
| | P | 1.02 | .857 | .755 | .456 | .769 | 1.08 | .388 | .334 | .321 | .95 | 1.48 | 2.08 |
| | Pa | 1.02 | .857 | .755 | 1.53 | 2.51 | 3.36 | .37 | .317 | .332 | 1.01 | 1.56 | 1.93 |
| | Pe | .766 | .64 | .579 | 1.56 | 2.54 | 3.42 | .353 | .317 | .278 | .92 | 1.58 | 2.12 |
| Enron | *#supernodes:* | 1k | 1.5k | 2k | 1k | 1.5k | 2k | 1k | 1.5k | 2k | 1k | 1.5k | 2k |
| | *#superedges:* | 51k | 68k | 81k | 51k | 68k | 81k | 51k | 68k | 81k | 51k | 68k | 81k |
| | A | – | | | 3.97 | | | – | | | 6.58 | | |
| | N | .132 | .198 | .244 | 1.64 | 2.09 | 2.25 | .271 | .215 | .218 | .54 | .67 | 1.36 |
| | Nw | .666 | .632 | .613 | 6.87 | 8.06 | 8.16 | .23 | .201 | .18 | .51 | .77 | 1.74 |
| | P | .273 | .308 | .31 | 1.08 | 16.6 | 22.3 | .301 | .325 | .312 | 6.97 | 11.1 | 17.6 |
| | Pa | .279 | .309 | .312 | 38 | 59.1 | 73.7 | .318 | .322 | .317 | 6.61 | 10.5 | 20.7 |
| | Pe | .37 | .383 | .381 | 38.6 | 6.45 | 75.5 | .238 | .223 | .223 | 7.33 | 13.1 | 21.5 |
| Gnutella | *#supernodes:* | 500 | 750 | 1k | 500 | 750 | 1k | 500 | 750 | 1k | 500 | 750 | 1k |
| | *#superedges:* | 2k | 3k | 5k | 2k | 3k | 5k | 2k | 3k | 5k | 2k | 3k | 5k |
| | A | – | | | .128 | | | – | | | 1.13 | | |
| | N | 39.9 | 34.8 | 30.1 | .017 | .033 | .055 | .947 | .914 | .865 | .05 | .1 | .15 |
| | Nw | 1.28 | .677 | .422 | .035 | .064 | .1 | .954 | .910 | .863 | .05 | .11 | .16 |
| | P | .017 | .19 | .3 | .067 | .108 | .168 | 1.02 | 1.02 | 1.01 | 1.28 | 1.94 | 2.37 |
| | Pa | .017 | .19 | .3 | .085 | .142 | .268 | 1.02 | 1.02 | 1 | 1.5 | 2.26 | 2.32 |
| | Pe | .003 | .222 | .345 | .085 | .142 | .238 | 1.02 | 1.01 | 1.02 | 1.19 | 1.69 | 2.4 |
| Ubuntu | *#supernodes:* | 350 | 500 | 750 | 350 | 500 | 750 | 350 | 500 | 750 | 350 | 500 | 750 |
| | *#superedges:* | 35k | 55k | 83k | 35k | 55k | 83k | 35k | 55k | 83k | 35k | 55k | 83k |
| | A | – | | | 33.5 | | | – | | | 1.79 | | |
| | N | 1 316 | 1 208 | 1 075 | .999 | 2.28 | 3.5 | .838 | .733 | .656 | .26 | .4 | .86 |
| | Nw | 1.18 | .526 | .123 | 11.4 | 19.5 | 28.1 | .130 | .093 | .055 | .29 | .49 | .76 |
| | P | 1 231 | 1 150 | 1 058 | 14.9 | 31.2 | 52.6 | .788 | .783 | .66 | 4.76 | 7.69 | 12.9 |
| | Pa | 1.88 | 1.07 | .509 | 145 | 244 | 371 | .202 | .117 | .078 | 5.31 | 8.37 | 13.4 |
| | Pe | 1.77 | .982 | .439 | 146 | 246 | 379 | .153 | .114 | .069 | 4.99 | 7.94 | 14.8 |
| AS-Skitter | *#supernodes:* | 1k | 10k | 25k | 1k | 10k | 25k | 1k | 10k | 25k | 1k | 10k | 25k |
| | *#superedges:* | 39k | 462k | 983k | 39k | 462k | 983k | 39k | 462k | 983k | 39k | 462k | 983k |
| | A | – | | | 1.6k | | | – | | | 440 | | |
| | N | 1.58 | .8 | .56 | 1.54 | 53.8 | 106 | .544 | .342 | .283 | .381 | 6.32 | 14.9 |
| | Nw | .92 | .815 | .754 | 8.03 | 118 | 217 | .518 | .35 | .289 | .452 | 7.09 | 18.7 |
| | P | .44 | .191 | .155 | .417 | 30.9 | 147 | .632 | .5 | .455 | 45.1 | 92.7 | 204 |
| | Pa | .184 | .129 | .117 | 3.95 | 133 | 593 | .734 | .609 | .509 | 47.9 | 91.8 | 216 |
| | Pe | .657 | .451 | .402 | 1.3 | 109 | 433 | .666 | .465 | .416 | 96.6 | 104 | 234 |

(b) SWeG [74] summaries

| | method | *clustering coefficient* | | | | | | *community detection (modularity)* | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *relative error* | | | *runtime (s)* | | | *relative error* | | | *runtime (s)* | | |
| Facebook | *#sup.nodes:* | 708 | 977 | 1.2k | 708 | 977 | 1.2k | 708 | 977 | 1.2k | 708 | 977 | 1.2k |
| | *#sup.edges:* | 664 | 2.2k | 4.4k | 664 | 2.2k | 4.4k | 664 | 2.2k | 4.4k | 664 | 2.2k | 4.4k |
| | A | – | | | 1.69 | | | – | | | .86 | | |
| | N | .652 | .404 | .228 | .006 | .014 | .036 | .23 | .204 | .21 | .03 | .06 | .08 |
| LastFM | *#sup.nodes:* | 3.4k | 3.6k | 3.8k | 3.4k | 3.6k | 3.8k | 3.4k | 3.6k | 3.8k | 3.4k | 3.6k | 3.8k |
| | *#sup.edges:* | 1.2k | 1.6k | 2k | 1.2k | 1.6k | 2k | 1.2k | 1.6k | 2k | 1.2k | 1.6k | 2k |
| | A | – | | | .215 | | | – | | | .61 | | |
| | N | .991 | .951 | .924 | .012 | .015 | .017 | .703 | .511 | .512 | .19 | .28 | .33 |
| Enron | *#sup.nodes:* | 23k | 24k | 24.4k | 23k | 24k | 24.4k | 23k | 24k | 24.4k | 23k | 24k | 24.4k |
| | *#sup.edges:* | 10k | 29k | 44k | 10k | 29k | 44k | 10k | 29k | 44k | 10k | 29k | 44k |
| | A | – | | | 3.97 | | | – | | | 6.58 | | |
| | N | .907 | .773 | .568 | .196 | .416 | .773 | .731 | .464 | .412 | 2.39 | 2.15 | 2.74 |
| AS-Skitter | *#sup.nodes:* | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M | 1.1M |
| | *#sup.edges:* | 1.7M | 1.9M | 2.1M | 1.7M | 1.9M | 2.1M | 1.7M | 1.9M | 2.1M | 1.7M | 1.9M | 2.1M |
| | A | – | | | 1.6k | | | – | | | 440 | | |
| | N | .044 | .071 | .076 | 60.8 | 76.8 | 78.4 | .679 | .675 | .688 | 197 | 192 | 206 |

GPQPS methods (which, we recall, need to keep in memory the summary only) are able to consistently reduce the space requirements.

**Effectiveness and efficiency** results are reported in Tables 2 and 3. Because of lack of space, we report results with varying the summary size for clustering coefficient and community detection queries, while for the other queries we show results for one summary size only. Results with other summary sizes are in [5] (Appendix B).

*General remarks.* As expected, accuracies and runtimes with increasing the summary size follow in general an increasing trend. Regarding accuracy, a few exceptions arise with the Probabilistic-GPQPS variants or on SWeG summaries (e.g., clustering-coefficient relative errors on Gnutella and AS-Skitter, for both S2L and SWeG). A motivation for this relies on the degree of connectedness of a summary (in general or in the various sampled worlds): a bigger, worse-connected summary may lead to less effective query processing than a smaller, better-connected summary.

The Naïve-GPQPS variants are consistently faster than actual query processing on the input graph. The speedup clearly depends on the dataset and summary sizes, and expensiveness of the query. However, it is tangible in all the cases (up to 4 orders of magnitude).

The Probabilistic-GPQPS variants are faster than actual query processing only for large graphs and expensive queries (e.g., closeness centrality). However, these methods can still be useful for smaller graphs/inexpensive queries as (1) they lead to storage-space savings nonetheless, and (2) their computation over individual worlds may be easily parallelized (we note that the reported runtimes of P, Pa, and Pe refer to a sequential execution of them).

*Clustering coefficient (Table 2).* As for S2L summaries, in most cases, relative errors are rather low, mostly concentrated around a $[0.2, 0.3]$ range. Exceptions to this are on Ubuntu for the GPQPS methods that discard superedge weights (N, P). This is not surprising, as Ubuntu is a weighted graph. Between Naïve-GPQPS and Probabilistic-GPQPS (looking at their best-performing variants in every dataset) there is no clear winner, as each method achieves better performance in 3 out of 6 datasets. No general winner is between N and Nw either: this meets literature evidence that superedge weights are useful or not, depending on the dataset [41]. A similar finding arises when comparing the Probabilistic-GPQPS variants to each other. As for SWeG summaries, GPQPS performance is pretty/very good when the summary is not too sparse (e.g., on Facebook, AS-Skitter), while it gets poor on very sparse summaries (e.g., LastFM).

*Community detection (Table 2).* Results here are broadly in line with the ones of clustering coefficient. Error values are slightly lower, but still mostly falling into a $[0.2, 0.3]$ range. Again, the unweighted GPQPS methods (N, P) do not achieve good results on Ubuntu. A difference with respect to clustering coefficient is that GPQPS does not perform well on Gnutella (relative errors around 1). A motivation for this could be that Gnutella is a directed graph. Performance on Gnutella is not particularly good for other queries too (see next), at least for some GPQPS methods. This suggests that, perhaps, GPQPS is still not mature for handling directed graphs, at least not with the current graph-summarization methods for directed graphs.

*PageRank centrality (Table 3).* As expected, increasing the summary-retrieved vertices $s$ leads to decreasing precision and increasing recall. On S2L summaries, the GPQPS methods are particularly good at recall: for $s = 500$, the recall of the best Naïve-GPQPS and Probabilistic-GPQPS variants is $\in [0.7, 1]$ across all datasets. Precision results are worse, but still overall reasonable, and even very high on some datasets (e.g., on Ubuntu from 0.8 to 0.97, for $s = 100$). This behavior is not surprising, as it is inherent in the design principles of the evaluation for these queries, where the summary-retrieved vertex sets tend to be larger than the ground-truth sets, and this clearly favors recall, not precision. As with the previous queries, there is no clear winner or looser among the GPQPS methods/variants. On the negative side is Probabilistic-GPQPS's results on Gnutella, motivated by the nature of that dataset (directed), as said above. Also, again,

**Table 3: GPQPS results for centrality and core decomposition queries. "A" stands for actual query processing on the original graph. Metrics for centralities: runtime (s), and precision ($P$) and recall ($R$) with varying the number of top-$s$-ranked vertices in the summary (w.r.t. the "ground-truth" of top-100-ranked vertices in the original graph). Closeness-centrality results on AS-Skitter refer to a subset of 1 000 randomly-sampled vertices. Metrics for core decomposition: runtime (s), and precision ($P$) and recall ($R$) with varying the number of top-$z$ innermost cores in the summary (w.r.t. the "ground-truth" innermost core $C^*$ of the original graph).**

(a) S2L [70] summaries

| *dataset* | *method* | *PageRank centrality* | | | | | | | *closeness centrality* | | | | | | | *core decomposition* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | |
| | | | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ | | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ | | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ |
| Facebook | | *#supernodes: 500, #superedges: 14 390* | | | | | | | | | | | | | | | | | | | | |
| | A | 2.2 | – | | | | | | 23 | – | | | | | | .64 | – | | | | | |
| | N | .98 | .26 | .26 | .21 | .43 | .13 | .64 | .43 | .22 | .22 | .15 | .31 | .08 | .41 | .11 | .99 | .05 | .99 | .05 | .93 | .69 |
| | Nw | .56 | .36 | .36 | .23 | .45 | .15 | .76 | 6.7 | .18 | .18 | .15 | .31 | .08 | .41 | .13 | 1 | .12 | 1 | .12 | 1 | .12 |
| | P | 40 | .39 | .28 | .24 | .41 | .16 | .65 | 45 | .32 | .28 | .16 | .31 | .08 | .4 | 7.7 | 1 | .03 | 1 | .03 | 1 | .03 |
| | Pa | 40 | .39 | .28 | .24 | .41 | .16 | .65 | 437 | .32 | .28 | .16 | .31 | .08 | .4 | 9.6 | 1 | .04 | 1 | .06 | 1 | .15 |
| | Pe | 39 | .36 | .3 | .23 | .42 | .16 | .7 | 454 | 0 | 0 | .18 | .27 | .09 | .37 | 9.6 | 1 | .07 | 1 | .07 | 1 | .08 |
| LastFM | | *#supernodes: 750, #superedges: 10 350* | | | | | | | | | | | | | | | | | | | | |
| | A | 1.7 | – | | | | | | 61 | – | | | | | | .25 | – | | | | | |
| | N | .31 | .66 | .66 | .42 | .85 | .2 | .98 | .81 | .47 | .49 | .27 | .55 | .19 | .93 | .08 | .79 | .14 | .84 | .2 | .84 | .24 |
| | Nw | .36 | .69 | .69 | .42 | .85 | .2 | 1 | 8.7 | .11 | .11 | .09 | .22 | .13 | .68 | .09 | 1 | .04 | 1 | .06 | 1 | .09 |
| | P | 20.3 | .79 | .61 | .51 | .79 | .21 | .97 | 76 | .8 | .6 | .51 | .88 | .2 | .95 | 4 | 1 | .06 | 1 | .08 | 1 | .15 |
| | Pa | 20.8 | .79 | .61 | .51 | .79 | .21 | .97 | 337 | .8 | .6 | .51 | .88 | .2 | .95 | 5.3 | 1 | .04 | 1 | .09 | 1 | .15 |
| | Pe | 20.9 | .72 | .65 | .46 | .83 | .21 | .99 | 367 | 1 | .01 | 1 | .03 | .31 | .94 | 5.1 | 1 | .04 | 1 | .07 | 1 | .12 |
| Enron | | *#supernodes: 1 500, #superedges: 68 405* | | | | | | | | | | | | | | | | | | | | |
| | A | 24 | – | | | | | | 1.2k | – | | | | | | 4.2 | – | | | | | |
| | N | 1.7 | .56 | .56 | .35 | .7 | .17 | .87 | 8.8 | .75 | .75 | .46 | .91 | .19 | .98 | .55 | .52 | .62 | .53 | .67 | .48 | .92 |
| | Nw | 2.1 | .56 | .56 | .35 | .7 | .17 | .86 | 112 | 0 | 0 | .01 | .04 | .02 | .12 | .67 | .77 | .83 | .67 | .9 | .58 | .92 |
| | P | 123 | .6 | .53 | .37 | .68 | .18 | .84 | 610 | .89 | .78 | .54 | .98 | .21 | .99 | 32 | .65 | .65 | .59 | .68 | .59 | .74 |
| | Pa | 126 | .6 | .53 | .37 | .68 | .18 | .84 | 5.8k | .89 | .78 | .54 | .98 | .21 | .99 | 43 | .64 | .66 | .58 | .69 | .49 | .75 |
| | Pe | 130 | .59 | .55 | .36 | .69 | .18 | .86 | 6.4k | 0 | 0 | 0 | 0 | 1 | .03 | 42 | .78 | .61 | .69 | .67 | .62 | .75 |
| Gnutella | | *#supernodes: 750, #superedges: 3 383* | | | | | | | | | | | | | | | | | | | | |
| | A | .76 | – | | | | | | 42 | – | | | | | | .2 | – | | | | | |
| | N | .11 | .65 | .65 | .4 | .8 | .17 | .87 | .14 | .62 | .72 | .36 | .73 | .14 | .73 | .03 | .99 | .15 | .99 | .16 | 1 | 1 |
| | Nw | .15 | .55 | .55 | .39 | .78 | .18 | .88 | 2.9 | .03 | .05 | .02 | .05 | .02 | .09 | .03 | .91 | .01 | .94 | .01 | .96 | .02 |
| | P | 8.3 | 0 | 0 | 1 | .08 | .85 | .39 | 37 | .96 | .24 | .84 | .41 | .76 | .6 | 1.2 | .82 | 0 | .86 | 0 | .97 | .02 |
| | Pa | 8.6 | 0 | 0 | 1 | .08 | .85 | .39 | 77 | .96 | .24 | .84 | .41 | .76 | .6 | 1.3 | .82 | 0 | .86 | 0 | .97 | .02 |
| | Pe | 8.8 | 0 | 0 | 1 | .08 | .84 | .38 | 77 | 1 | .11 | .84 | .38 | .76 | .6 | 1.3 | 1 | 0 | 1 | 0 | .9 | .01 |
| Ubuntu | | *#supernodes: 500, #superedges: 55 276* | | | | | | | | | | | | | | | | | | | | |
| | A | 4.1 | – | | | | | | 471 | – | | | | | | 1.5 | – | | | | | |
| | N | 1.1 | .8 | .8 | .49 | .99 | .2 | 1 | .79 | .51 | .5 | .34 | .68 | .16 | .81 | .41 | .99 | .27 | .99 | .27 | .99 | .28 |
| | Nw | 1.5 | .96 | .96 | .5 | 1 | .2 | 1 | 28 | .04 | .04 | .06 | .12 | .14 | .68 | .46 | 1 | 0 | 1 | .01 | .97 | .01 |
| | P | 94 | .82 | .75 | .53 | .99 | .21 | 1 | 52 | .54 | .5 | .34 | .63 | .17 | .81 | 32 | .99 | .2 | .99 | .26 | .82 | .88 |
| | Pa | 125 | .97 | .95 | .51 | 1 | .2 | 1 | 1.9k | .6 | .39 | .35 | .52 | .17 | .81 | 38 | 1 | 0 | 1 | .01 | .97 | .01 |
| | Pe | 127 | .97 | .95 | .51 | 1 | .2 | 1 | 2k | 0 | 0 | 0 | 0 | .19 | .81 | 38 | 1 | 0 | 1 | .01 | .97 | .01 |
| AS-Skitter | | *#supernodes: 10 000, #superedges: 461 826* | | | | | | | | | | | | | | | | | | | | |
| | A | 66 | – | | | | | | 2.5k | – | | | | | | 180 | – | | | | | |
| | N | 2.17 | .27 | .27 | .21 | .43 | .14 | .71 | 53.4 | .42 | .42 | .37 | .75 | .1 | 1 | 4.6 | .8 | .57 | .8 | .57 | .8 | .58 |
| | Nw | 20.1 | .36 | .36 | .24 | .49 | .16 | .79 | 266 | .43 | .43 | .38 | .75 | .1 | 1 | 5.9 | 1 | 0 | 1 | 0 | 1 | 0 |
| | P | 383 | .38 | .27 | .3 | .46 | .16 | .69 | 630 | .46 | .41 | .38 | .71 | .1 | .96 | 125 | 1 | .06 | 1 | .06 | 1 | .07 |
| | Pa | 444 | .38 | .27 | .3 | .46 | .16 | .69 | 4.5k | .46 | .41 | .38 | .71 | .1 | .96 | 139 | .81 | .61 | .81 | .61 | .81 | .62 |
| | Pe | 391 | .37 | .31 | .27 | .46 | .16 | .72 | 3.7k | .7 | .26 | .64 | .65 | .11 | .92 | 126 | 1 | 0 | 1 | 0 | 1 | 0 |

(b) SWeG [74] summaries

| *dataset* | *method* | *PageRank centrality* | | | | | | | *closeness centrality* | | | | | | | *core decomposition* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | |
| | | | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ | | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ | | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ |
| F.book | | *#supernodes: 977, #superedges: 2 157* | | | | | | | | | | | | | | | | | | | | |
| | A | 2.2 | – | | | | | | 23 | – | | | | | | .64 | – | | | | | |
| | N | .12 | .13 | .13 | .07 | .13 | .03 | .13 | .54 | .14 | .14 | .09 | .17 | .03 | .17 | .02 | .97 | .99 | .97 | .99 | .94 | 1 |
| L.FM | | *#supernodes: 3 568, #superedges: 3 821* | | | | | | | | | | | | | | | | | | | | |
| | A | 1.7 | – | | | | | | 61 | – | | | | | | .25 | – | | | | | |
| | N | .18 | .01 | .01 | 0 | .01 | 0 | .02 | .17 | 0 | 0 | 0 | 0 | 0 | 0 | .03 | .71 | .05 | .45 | .11 | .38 | 1 |
| Enron | | *#supernodes: 23 957, #superedges: 28 825* | | | | | | | | | | | | | | | | | | | | |
| | A | 24 | – | | | | | | 1.2k | – | | | | | | 4.2 | – | | | | | |
| | N | 1.9 | .67 | .67 | .34 | .67 | .13 | .67 | 209 | .37 | .37 | .23 | .46 | .12 | .58 | .51 | .02 | .48 | .02 | .48 | .02 | 1 |
| AS-Sk. | | *#supernodes: 1 087 430, #superedges: 1 857 562* | | | | | | | | | | | | | | | | | | | | |
| | A | 66 | – | | | | | | 2.5k | – | | | | | | 180 | – | | | | | |
| | N | 9.9 | .65 | .65 | .39 | .78 | .19 | .95 | 712 | .18 | .18 | .09 | .18 | .08 | .39 | 30 | 1 | .34 | 1 | .34 | .89 | .84 |

performance of SWeG on sparser summaries is much worse than the performance on denser summaries.

*Closeness centrality (Table 3).* Results here are mostly in accordance with the ones on PageRank, with an overall slight decrease in the various precision and recall scores. This is not surprising, because closeness centrality is a harder query for GPQPS. In fact, closeness is less robust than PageRank to the disappearing of paths between vertices when switching from the original graph to the summary. An important remark here is that the efficiency results on AS-Skitter are not really reliable, because for this large dataset we compute closeness of 1k (randomly sampled) vertices only. Therefore, the comparison to what comes from summary is not fair, as for our assessment we need the closeness of a variable-size set of supernodes, that is all the supernodes that contain the sampled vertices. Despite this, still a consistent speedup of GPQPS methods is observed (at least by Naïve-GPQPS).

*Core decomposition (Table 3).* Trends and observations here are similar to the previous queries. A key difference is that, for $z = 1$ or 2, precision is mostly higher than recall. This suggests that taking up to 2 top-innermost cores of the summary does match the innermost core of the graph, but it does not cover it entirely.

## 6 SUMMARY OF MAIN FINDINGS

**Promising effectiveness.** Overall, the basic GPQPS methods introduced in this work achieve fair accuracy results. Particularly good performance is observed for clustering coefficient and community detection queries, in terms of recall for centrality queries, and precision for core decomposition. All in all, we believe we can claim that the methods we design in this work are rather powerful baselines for the emerging GPQPS problem.

**Obstacles for a really effective GPQPS.** Main exceptions to satisfactorily effective GPQPS are when: (1) weighted graphs are handled with unweighted summaries or with GPQPS methods that discard superedge weights (at least for some queries); (2) handling directed graphs; (3) summaries are overly sparse or not well-connected.

**Consistent gain in storage space** is achieved by any GPQPS method.

**Drastic speedup by Naïve-GPQPS.** It depends on the dataset and the query, but, typically, it is consistent (i.e., orders of magnitude).

**Speedup by Probabilistic-GPQPS** is appreciable for large datasets or expensive queries. However, the usefulness of this method is not limited to those cases as (1) it gives storage space saving nevertheless, and (2) speedup in smaller graphs or less expensive queries can still be achieved by parallelizing its computation over the various worlds (straightforward parallelization).

**Increasing summary size** corresponds to an increase of effectiveness and a decrease of speedup, which perfectly meets common sense. A few exceptions to effectiveness increase are when the summaries are not well-connected.

**Naïve-GPQPS vs. Probabilistic-GPQPS: no clear winner**. These two methods are mostly comparable to one another, with each one (slightly) outperforming the other depending on the dataset and the query. We conjecture that a major reason why Probabilistic-GPQPS performs similarly to Naïve-GPQPS despite it is more sophisticated is the query-answer aggregation strategy over the various worlds, especially for vertex-set or partitioning queries (see Section 7 for ideas on how to address this issue in future work).

**No clear winner among weighted and unweighted variants** of the various GPQPS methods. This confirms a literature finding [41] that superedge weights in a summary may be beneficial or not.

## 7 FUTURE DIRECTIONS AND CONCLUSION

In this paper, we introduce *general-purpose (approximate) query processing on summary graph* (GPQPS), a new tool to support scalable data-science workloads on graphs. Our major goal in studying this problem is to set its stage, and stimulate and drive further research on it. Possible, concrete future directions are as follows.

**Refine GPQPS methods in general.** The GPQPS methods presented here are basic ones: they can and should be improved, from several perspectives. A possible refinement consists in understanding the relationship between the coarser-grained structure of a summary and the finer-grained structure of the original graph, so as to derive proper correction terms to be used for numerical queries. As an example, this way one may discover that the clustering coefficient of the summary should be multiplied by a certain factor in order to actually match the clustering coefficient of the input graph.

As for vertex-set or partitioning queries, an effective direction could be to refine the strategies for deriving a set of vertices out of a set of supernodes. In concrete, instead of the one investigated in this work, which simply takes the union of the supernodes, one can exploit the information of superedges too to derive a more representative vertex set. Another idea could be to rerun the query on the union of the supernodes. In particular, this could intuitively work well for queries (e.g., core decomposition) that only depend on intra-set edges. Conversely, this could be less effective for queries like community detection, where inter-set edges matter as well.

**Refine Probabilistic-GPQPS algorithm.** Probabilistic-GPQPS (Algorithm 2) is particularly prone to improvement. For instance, one could define more sophisticated strategies of vertex-set aggregation than simple vertex-set intersection. A concrete idea here could be to compute *aggregation spheres* out of the vertex sets produced in various worlds, taking inspiration by what Mehmood *et al.* [63] do for the problem of influence maximization.

Another refinement could be to perform *summary aggregation* over the possible worlds, instead of query-answer aggregation. A major advantage of this is that aggregation over the possible worlds would be no more dependent on the form of the query answer.

**Miscellanea.** Other interesting directions include deriving theoretical approximation guarantees for GPQPS methods; experimentally investigating GPQPS on more queries; devising query-processing-effective methods to produce graph summaries. The last one is particularly appealing, as, so far, graph-summarization methods have been defined by considering mostly the adherence of the reconstructed graph to the original graph, and agnostically to how good a resulting summary could be at answering queries. For instance, graph-summarization methods can focus better on reducing the sparsity of the resulting summaries, which is an aspect that affects GPQPS significantly, as observed in our experiments.

## REFERENCES

[1] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. 1987. On the representation and querying of sets of possible worlds. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 34–48.

[2] Charu C. Aggarwal and Haixun Wang (Eds.). 2010. *Managing and Mining Graph Data*. Advances in Database Systems, Vol. 40. Springer.

[3] Charu C. Aggarwal and Haixun Wang. 2010. A Survey of Clustering Algorithms for Graph Data. In *Managing and Mining Graph Data*, Charu C. Aggarwal and Haixun Wang (Eds.). Advances in Database Systems, Vol. 40. Springer, 275–301.

[4] K. Jin Ahn, S. Guha, and A. McGregor. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *Proc. of Symp. on Principles of Database Systems (PODS)*. 5–14.

[5] Aris Anagnostopoulos, Valentina Arrigoni, Francesco Gullo, Giorgia Salvatori, and Lorenzo Severini. 2023. General-purpose Query Processing on Summary Graphs. https://github.com/fgullo/GPQPS. *Supplemental Material and Experimental Artifacts* (2023).

[6] Vladimir Batagelj and Matjaz Zaversnik. 2011. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification (ADAC)* 5, 2 (2011), 129–145.

[7] Maham Anwar Beg, Muhammad Ahmad, Arif Zaman, and Imdadullah Khan. 2018. Scalable Approximation Algorithm for Graph Summarization. In *Proc. of Pacific-Asia Conf. on Advances on Knowledge Discovery and Data Mining (PAKDD)*. 502–514.

[8] Maciej Besta and Torsten Hoefler. 2018. Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations. *CoRR* abs/1806.01799 (2018).

[9] Maciej Besta, Simon Weber, Lukas Gianinazzi, Robert Gerstenberger, Andrey Ivanov, Yishai Oltchik, and Torsten Hoefler. 2019. Slim Graph: practical lossy graph compression for approximate graph processing, storage, and analytics. In *Proc. of Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*. 35:1–35:25.

[10] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008.

[11] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. 2009. Permuting Web and Social Graphs. *Internet Mathematics* 6, 3 (2009), 257–283.

[12] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph framework I: compression techniques. In *Proc. of World Wide Web Conf. (WWW)*. 595–602.

[13] Francesco Bonchi, Ilaria Bordino, Francesco Gullo, and Giovanni Stilo. 2016. Identifying Buzzing Stories via Anomalous Temporal Subgraph Discovery. In *Proc. of IEEE/WIC/ACM Int. Conf. on Web Intelligence (WI)*. 161–168.

[14] Francesco Bonchi, Ilaria Bordino, Francesco Gullo, and Giovanni Stilo. 2019. The importance of unexpectedness: Discovering buzzing stories in anomalous temporal graphs. *Web Intelligence* 17, 3 (2019), 177–198.

[15] Francesco Bonchi, Francesco Gullo, and Andreas Kaltenbrunner. 2018. Core Decomposition of Massive, Information-Rich Graphs. In *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*, Reda Alhajj and Jon G. Rokne (Eds.).

[16] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1316–1325.

[17] Ilaria Bordino and Francesco Gullo. 2018. Network-based Receivable Financing. In *Proc. of ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 2137–2145.

[18] Ilaria Bordino, Francesco Gullo, and Giacomo Legnaro. 2021. Advancing Receivable Financing via a Network-Based Approach. *IEEE Transactions on Network Science and Engineering (TNSE)* 8, 2 (2021), 1328–1337.

[19] Michele Coscia. 2021. The Atlas for the Aspiring Network Scientist. *CoRR* abs/2101.00863 (2021).

[20] M. Coscia and F. M. H. Neffke. 2017. Network Backboning with Noisy Data. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 425–436.

[21] Nilesh Dalvi and Dan Suciu. 2004. Efficient query evaluation on probabilistic databases. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB)*. 864–875.

[22] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. 2012. Query preserving graph compression. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 157–168.

[23] Wenfei Fan, Yuanhao Li, Muyang Liu, and Can Lu. 2021. Making Graphs Compact by Lossless Contraction. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 472–484.

[24] Wenfei Fan, Yuanhao Li, Muyang Liu, and Can Lu. 2022. A Hierarchical Contraction Scheme for Querying Big Graphs. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 1726–1740.

[25] Adriano Fazzone, Tommaso Lanciano, Riccardo Denni, Charalampos E. Tsourakakis, and Francesco Bonchi. 2022. Discovering Polarization Niches via Dense Subgraphs with Attractors and Repulsers. *Proc. of the VLDB Endowment (PVLDB)* 15, 13 (2022), 3883–3896.

[26] Alan M. Frieze, Aristides Gionis, and Charalampos E. Tsourakakis. 2013. Algorithmic techniques for modeling and mining large graphs (AMAzING). In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1523.

[27] Xiang Fu, Shangdi Yu, and Austin R Benson. 2019. Modelling and analysis of tagging networks in Stack Exchange communities. https://www.cs.cornell.edu/~arb/data/stack-exchange-tags/. *Journal of Complex Networks* 8, 5 (2019).

[28] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. 2019. A General Framework for Graph Sparsification. *SIAM Journal on Computing (SICOMP)* 48, 4 (2019), 1196–1223.

[29] Edoardo Galimberti, Francesco Bonchi, Francesco Gullo, and Tommaso Lanciano. 2020. Core Decomposition in Multilayer Networks: Theory, Algorithms, and Applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14, 1 (2020), 11:1–11:40.

[30] Edoardo Galimberti, Martino Ciaperoni, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2021. Span-core Decomposition for Temporal Networks: Algorithms and Applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 1 (2021), 2:1–2:44.

[31] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. 2007. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 4.

[32] Xiangyang Gou, Lei Zou, Chenxingyu Zhao, and Tong Yang. 2019. Fast and Accurate Graph Stream Summarization. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 1118–1129.

[33] Francesco Gullo, Andrea Tagarelli, and Sergio Greco. 2009. Diversity-Based Weighting Schemes for Clustering Ensembles. In *Proc. of SIAM Int. Conf. on Data Mining (SDM)*. 437–448.

[34] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. https://networkx.org/. In *Proc. of the 7th Python in Science Conf.* 11–15.

[35] Mahdi Hajiabadi, Jasbir Singh, Venkatesh Srinivasan, and Alex Thomo. 2021. Graph Summarization with Controlled Utility Loss. https://github.com/MahdiHajiabadi/GSCIS_TBUDS. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 536–546.

[36] Cecilia Hernández and Gonzalo Navarro. 2014. Compressed representations for web and social graphs. *Knowledge and Information Systems (KAIS)* 40, 2 (2014), 279–313.

[37] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proc. of ACM Symp. on Theory of Computing (STOC)*. 604–613.

[38] Zhiguo Jiang, Hanhua Chen, and Hai Jin. 2023. Auxo: A Scalable and Efficient Graph Stream Summarization Structure. *Proc. of the VLDB Endowment (PVLDB)* 16, 6 (2023).

[39] Di Jin, Zhizhi Yu, Pengfei Jiao, Shirui Pan, Dongxiao He, Jia Wu, Philip S. Yu, and Weixiong Zhang. 2023. A Survey of Community Detection Approaches: From Statistical Modeling to Deep Learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 2 (2023), 1149–1170.

[40] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. 2011. Distance-Constraint Reachability Computation in Uncertain Graphs. *Proc. of the VLDB Endowment (PVLDB)* 4, 9 (2011), 551–562.

[41] Shinhwan Kang, Kyuhan Lee, and Kijung Shin. 2022. Are Edge Weights in Summary Graphs Useful? - A Comparative Study. In *Proc. of Pacific-Asia Conf. on Advances on Knowledge Discovery and Data Mining (PAKDD)*. 54–67.

[42] Shinhwan Kang, Kyuhan Lee, and Kijung Shin. 2022. Personalized Graph Summarization: Formulation, Scalable Algorithms, and Applications. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 2319–2332.

[43] Xiangyu Ke, Arijit Khan, and Francesco Bonchi. 2022. Multi-relation Graph Summarization. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 16, 5 (2022), 82:1–82:30.

[44] Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. 2014. Fast Reliability Search in Uncertain Graphs. In *Proc. EDBT Conf.* 535–546.

[45] Arijit Khan, Francesco Bonchi, Francesco Gullo, and Andreas Nufer. 2018. Conditional Reliability in Uncertain Graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 30, 11 (2018), 2078–2092.

[46] Arijit Khan, Yuan Ye, and Lei Chen. 2018. *On Uncertain Graphs*. Morgan & Claypool Publishers.

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

[48] Jihoon Ko, Yunbum Kook, and Kijung Shin. 2020. Incremental Lossless Graph Summarization. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 317–327.

[49] George Kollios, Michalis Potamias, and Evimaria Terzi. 2013. Clustering large probabilistic graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 25, 2 (2013), 325–336.

[50] Danai Koutra and Christos Faloutsos. 2017. *Individual and Collective Graph Mining: Principles, Algorithms, and Applications.* Morgan & Claypool Publishers.

[51] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. 2014. VoG: Summarizing and Understanding Large Graphs. In *Proc. of SIAM Int. Conf. on Data Mining (SDM)*. 91–99.

[52] K. Ashwin Kumar and Petros Efstathopoulos. 2018. Utility-Driven Graph Summarization. *Proc. of the VLDB Endowment (PVLDB)* 12, 4 (2018), 335–347.

[53] Tommaso Lanciano, Aurora Savino, Francesca Porcu, Davide Cittaro, Francesco Bonchi, and Paolo Provero. 2023. Contrast subgraphs allow comparing homogeneous and heterogeneous networks derived from omics data. *GigaScience* 12 (2023).

[54] Kyuhan Lee, Hyeonsoo Jo, Jihoon Ko, Sungsu Lim, and Kijung Shin. 2020. SSumM: Sparse Summarization of Massive Graphs. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 144–154.

[55] Kyuhan Lee, Jihoon Ko, and Kijung Shin. 2022. SLUGGER: Lossless Hierarchical Summarization of Massive Graphs. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 472–484.

[56] Kristen LeFevre and Evimaria Terzi. 2010. GraSS: Graph Structure Summarization. In *Proc. of SIAM Int. Conf. on Data Mining (SDM)*. 454–465.

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

[58] Xingjie Liu, Yuanyuan Tian, Qi He, Wang-Chien Lee, and John McPherson. 2014. Distributed Graph Summarization. In *Proc. of ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 799–808.

[59] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph Summarization Methods and Applications: A Survey. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 62:1–62:34.

[60] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–136.

[61] Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. 2020. In and Out: Optimizing Overall Interaction in Probabilistic Graphs under Clustering Constraints. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1371–1381.

[62] Hossein Maserrat and Jian Pei. 2010. Neighbor query friendly compression of social networks. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 533–542.

[63] Yasir Mehmood, Francesco Bonchi, and David García-Soriano. 2016. Spheres of Influence for More Effective Viral Marketing. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 711–726.

[64] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. 2008. Graph summarization with bounded error. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 419–432.

[65] Michelle E. J. Newman and Mark Girvan. 2004. Finding and evaluating community structure in networks. *Physical Review E* 69, 2 (2004), 026113.

[66] Panos Parchas, Francesco Gullo, Dimitris Papadias, and Francesco Bonchi. 2014. The pursuit of a good possible world: extracting representative instances of uncertain graphs. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 967–978.

[67] Panos Parchas, Francesco Gullo, Dimitris Papadias, and Francesco Bonchi. 2015. Uncertain Graph Processing through Representative Instances. *ACM Transactions on Database Systems (TODS)* 40, 3 (2015), 20:1–20:39.

[68] Michalis Potamias, Francesco Bonchi, Aris Gionis, and George Kollios. 2010. $k$-Nearest Neighbors in uncertain graphs. *Proc. of the VLDB Endowment (PVLDB)* 3, 1 (2010), 997–1008.

[69] Matteo Riondato, David García-Soriano, and Francesco Bonchi. 2014. Graph summarization with quality guarantees. In *Proc. IEEE Int. Conf. on Data Mining (ICDM)*. 947–952.

[70] Matteo Riondato, David García-Soriano, and Francesco Bonchi. 2017. Graph summarization with quality guarantees. https://github.com/rionda/graphsumm. *Data Mining and Knowledge Discovery (DAMI)* 31, 2 (2017), 314–349.

[71] Amin Sadri, Flora D. Salim, Yongli Ren, Masoomeh Zameni, Jeffrey Chan, and Timos Sellis. 2017. Shrink: Distance preserving graph compression. *Information Systems* 69 (2017), 180–193.

[72] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer Science Review* 1, 1 (2007), 27–64.

[73] M. Á. Serrano, M. Boguñá, and A. Vespignani. 2009. Extracting the multiscale backbone of complex weighted networks. *Proc. National Academy of Sciences* 106, 16 (2009), 6483–6488.

[74] Kijung Shin, Amol Ghoting, Myunghwan Kim, and Hema Raghavan. 2019. SWeG: Lossless and Lossy Summarization of Web-Scale Graphs. In *Proc. of World Wide Web Conf. (WWW)*. 1679–1690.

[75] P. B. Slater. 2009. A two-stage algorithm for extracting the multiscale backbone of complex weighted networks. *Proc. National Academy of Sciences* 106, 26 (2009), E66–E66.

[76] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral Sparsification of Graphs. *SIAM Journal on Computing (SICOMP)* 40, 4 (2011), 981–1025.

[77] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. 2011. Compression of weighted graphs. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 965–973.

[78] Alexander P. Topchy, Anil K. Jain, and William F. Punch. 2003. Combining Multiple Weak Clusterings. In *Proc. IEEE Int. Conf. on Data Mining (ICDM)*. 331–338.

[79] Ioanna Tsalouchidou, Francesco Bonchi, Gianmarco De Francisci Morales, and Ricardo Baeza-Yates. 2020. Scalable Dynamic Graph Summarization. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 32, 2 (2020), 360–373.

[80] Joe H. Ward. 1963. Hierarchical Grouping to Optimize an Objective Function. *J. Amer. Statist. Assoc.* 58, 301 (1963), 236–244.

[81] Quinton Yong, Mahdi Hajiabadi, Venkatesh Srinivasan, and Alex Thomo. 2021. Efficient Graph Summarization using Weighted LSH at Billion-Scale. In *Proc. of ACM Int. Conf. on Management of Data (SIGMOD)*. 2357–2365.

[82] Yiling Zeng, Chunyao Song, and Tingjian Ge. 2021. Selective Edge Shedding in Large Graphs Under Resource Constraints. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*. 2057–2062.

[83] F. Zhou, S. Mahler, and H. Toivonen. 2010. Network Simplification with Minimal Loss of Connectivity. In *Proc. IEEE Int. Conf. on Data Mining (ICDM)*. 659–668.

[84] Fang Zhou, Qiang Qu, and Hannu Toivonen. 2017. Summarisation of weighted networks. *Journal of Experimental and Theoretical Artificial Intelligence* 29, 5 (2017), 1023–1052.

[85] Houquan Zhou, Shenghua Liu, Kyuhan Lee, Kijung Shin, Huawei Shen, and Xueqi Cheng. 2021. DPGS: Degree-Preserving Graph Summarization. In *Proc. of SIAM Int. Conf. on Data Mining (SDM)*. 280–288.

## APPENDIX

**Table 4: GPQPS results for core decomposition. "A" stands for actual query processing on the original graph. Metrics: runtime (s), and precision ($P$) and recall ($R$) with varying the number of top-$z$ innermost cores in the summary (w.r.t. the "ground-truth" innermost core $C^*$ of the original graph).**

(a) S2L [70] summaries

| *data* | *meth.* | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* |
| Facebook | | *sup.nodes/edges: 350, 8.5k* | | | | | | | *sup.nodes/edges: 500, 14.4k* | | | | | | | *sup.nodes/edges: 750, 24.3k* | | | | | | |
| | A | .64 | | | – | | | | .64 | | | – | | | | .64 | | | – | | | |
| | N | .06 | 1 | .03 | 1 | .03 | .94 | 1 | .11 | .99 | .05 | .99 | .05 | .93 | .69 | .19 | 1 | .11 | .96 | .57 | .94 | 1 |
| | Nw | .07 | 1 | .04 | 1 | .05 | 1 | .24 | .13 | 1 | .12 | 1 | .12 | 1 | .12 | .23 | 1 | .12 | 1 | .14 | 1 | .17 |
| | P | 3.3 | 1 | .01 | 1 | .01 | 1 | .12 | 7.7 | 1 | .03 | 1 | .03 | 1 | .03 | 16 | 1 | .1 | 1 | .11 | 1 | .12 |
| | Pa | 4.2 | 1 | .06 | 1 | .12 | 1 | .23 | 9.6 | 1 | .04 | 1 | .06 | 1 | .15 | 19 | 1 | .18 | 1 | .2 | 1 | .26 |
| | Pe | 4.8 | 1 | .03 | 1 | .03 | 1 | .04 | 9.6 | 1 | .07 | 1 | .07 | 1 | .08 | 19 | 1 | .07 | 1 | .15 | 1 | .18 |
| LastFM | | *sup.nodes/edges: 500, 6.8k* | | | | | | | *sup.nodes/edges: 750, 10.4k* | | | | | | | *sup.nodes/edges: 1k, 12.8k* | | | | | | |
| | A | .25 | | | – | | | | .25 | | | – | | | | .25 | | | – | | | |
| | N | .05 | .82 | .09 | .73 | .41 | .38 | 1 | .08 | .79 | .14 | .84 | .2 | .84 | .24 | .11 | .8 | .33 | .63 | .67 | .38 | 1 |
| | Nw | .06 | 1 | .08 | 1 | .09 | 1 | .11 | .09 | 1 | .04 | 1 | .06 | 1 | .09 | .12 | 1 | .25 | .99 | .27 | .99 | .29 |
| | P | 2.5 | 1 | .05 | 1 | .1 | .95 | .14 | 4 | 1 | .06 | 1 | .08 | 1 | .15 | 6.1 | 1 | .11 | 1 | .17 | .99 | .27 |
| | Pa | 3.3 | 1 | .06 | 1 | .11 | 1 | .18 | 5.3 | 1 | .04 | 1 | .09 | 1 | .15 | 7.2 | 1 | .09 | 1 | .18 | .99 | .29 |
| | Pe | 3.2 | 1 | .07 | 1 | .09 | 1 | .12 | 5.1 | 1 | .04 | 1 | .07 | 1 | .12 | 7.4 | 1 | .07 | 1 | .18 | 1 | .23 |
| Enron | | *sup.nodes/edges: 1k, 50.1k* | | | | | | | *sup.nodes/edges: 1.5k, 68.4k* | | | | | | | *sup.nodes/edges: 2k, 81.4k* | | | | | | |
| | A | 4.2 | | | – | | | | 4.2 | | | – | | | | 4.2 | | | – | | | |
| | N | .39 | .05 | .94 | .03 | 1 | .02 | 1 | .55 | .52 | .62 | .53 | .67 | .48 | .92 | .67 | .39 | .85 | .39 | .86 | .4 | .92 |
| | Nw | .48 | .81 | .71 | .8 | .73 | .75 | .77 | .67 | .77 | .83 | .67 | .9 | .58 | .92 | .83 | .63 | .91 | .63 | .91 | .51 | .95 |
| | P | 21 | .78 | .56 | .77 | .63 | .76 | .71 | 32 | .65 | .65 | .59 | .68 | .59 | .74 | 43 | .42 | .7 | .44 | .8 | .39 | .94 |
| | Pa | 27 | .66 | .64 | .45 | .8 | .39 | .91 | 43 | .64 | .66 | .58 | .69 | .49 | .75 | 57 | .36 | .71 | .37 | .77 | .29 | .93 |
| | Pe | 27 | .82 | .48 | .81 | .53 | .8 | .72 | 42 | .78 | .61 | .69 | .67 | .62 | .75 | 56 | .45 | .59 | .44 | .62 | .47 | .75 |
| Gnutella | | *sup.nodes/edges: 500, 2k* | | | | | | | *sup.nodes/edges: 750, 3.4k* | | | | | | | *sup.nodes/edges: 1k, 5k* | | | | | | |
| | A | .2 | | | – | | | | .2 | | | – | | | | .2 | | | – | | | |
| | N | .02 | .99 | .12 | 1 | 1 | 1 | 1 | .03 | .99 | .15 | .99 | .16 | 1 | 1 | .04 | 1 | .22 | 1 | .22 | 1 | 1 |
| | Nw | .02 | .91 | .01 | .93 | .01 | .95 | .01 | .03 | .91 | .01 | .94 | .01 | .96 | .02 | .05 | .92 | .01 | .93 | .01 | .98 | .03 |
| | P | .7 | 1 | 0 | 1 | 0 | .9 | .01 | 1.2 | .82 | 0 | .86 | 0 | .97 | .02 | 1.6 | .88 | 0 | .88 | 0 | .97 | .03 |
| | Pa | .87 | 1 | 0 | 1 | 0 | .9 | .01 | 1.3 | .82 | 0 | .86 | 0 | .97 | .02 | 2 | .88 | 0 | .88 | 0 | .97 | .03 |
| | Pe | .88 | 1 | 0 | 1 | 0 | .88 | 0 | 1.3 | 1 | 0 | 1 | 0 | .9 | .01 | 2.1 | .8 | 0 | .83 | 0 | .92 | .01 |
| Ubuntu | | *sup.nodes/edges: 350, 35.3k* | | | | | | | *sup.nodes/edges: 500, 55.3k* | | | | | | | *sup.nodes/edges: 750, 82.6k* | | | | | | |
| | A | 1.5 | | | – | | | | 1.5 | | | – | | | | 1.5 | | | – | | | |
| | N | .26 | .8 | .78 | .8 | .78 | .84 | 1 | .41 | .99 | .27 | .99 | .27 | .99 | .28 | .65 | .99 | .21 | .99 | .35 | .84 | 1 |
| | Nw | .3 | 1 | 0 | 1 | .01 | .97 | .01 | .46 | 1 | 0 | 1 | .01 | .97 | .01 | .76 | 1 | 0 | 1 | .01 | .97 | .01 |
| | P | 21 | .98 | .08 | .78 | .66 | .8 | .79 | 32 | .99 | .2 | .99 | .26 | .82 | .88 | 52 | .72 | .49 | .73 | .51 | .79 | .74 |
| | Pa | 23 | 1 | 0 | 1 | .01 | .97 | .01 | 38 | 1 | 0 | 1 | .01 | .97 | .01 | 65 | 1 | 0 | 1 | .01 | .97 | .01 |
| | Pe | 24 | 1 | 0 | 1 | .01 | .97 | .01 | 38 | 1 | 0 | 1 | .01 | .97 | .01 | 71 | 1 | 0 | 1 | .01 | .97 | .01 |
| AS-Skitter | | *sup.nodes/edges: 1k, 39k* | | | | | | | *sup.nodes/edges: 10k, 462k* | | | | | | | *sup.nodes/edges: 25k, 983k* | | | | | | |
| | A | 180 | | | – | | | | 180 | | | – | | | | 180 | | | – | | | |
| | N | .61 | .83 | .74 | .84 | .75 | .84 | .75 | 4.64 | .8 | .57 | .8 | .57 | .8 | .58 | 11.1 | .78 | .47 | .78 | .48 | .78 | .48 |
| | Nw | .62 | 1 | 0 | 1 | 0 | 1 | 0 | 5.94 | 1 | 0 | 1 | 0 | 1 | 0 | 13.2 | 1 | 0 | 1 | 0 | 1 | 0 |
| | P | 34.6 | 1 | .13 | 1 | .15 | 1 | .16 | 125 | 1 | .06 | 1 | .06 | 1 | .07 | 377 | 1 | .03 | 1 | .03 | 1 | .03 |
| | Pa | 32.1 | .85 | .87 | .86 | .88 | .86 | .9 | 139 | .81 | .61 | .81 | .61 | .81 | .62 | 414 | .78 | .47 | .78 | .48 | .78 | .48 |
| | Pe | 29.4 | 1 | 0 | 1 | 0 | 1 | 0 | 126 | 1 | 0 | 1 | 0 | 1 | 0 | 395 | 1 | 0 | 1 | 0 | 1 | 0 |

(b) SWeG [74] summaries

| *data* | *meth.* | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | | *runt.* | $z=1$ | | $z=2$ | | $z=5$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* |
| F.book | | *sup.nodes/edges: 708, 664* | | | | | | | *sup.nodes/edges: 977, 2.2k* | | | | | | | *sup.nodes/edges: 1.2k, 4.4k* | | | | | | |
| | A | .64 | | | – | | | | .64 | | | – | | | | .64 | | | – | | | |
| | N | .01 | 1 | .49 | .9 | .51 | .94 | 1 | .02 | .97 | .99 | .97 | .99 | .94 | 1 | .04 | 1 | .9 | .97 | .91 | .94 | .92 |
| L.FM | | *sup.nodes/edges: 3.4k, 1.2k* | | | | | | | *sup.nodes/edges: 3.6k, 1.6k* | | | | | | | *sup.nodes/edges: 3.8k, 2k* | | | | | | |
| | A | .25 | | | – | | | | .25 | | | – | | | | .25 | | | – | | | |
| | N | .03 | .43 | .07 | .3 | .28 | .38 | 1 | .03 | .71 | .05 | .45 | .11 | .38 | 1 | .04 | .93 | .06 | .36 | .44 | .38 | 1 |
| Enron | | *sup.nodes/edges: 22.8k, 10.2k* | | | | | | | *sup.nodes/edges: 24k, 28.8k* | | | | | | | *sup.nodes/edges: 24.4k, 44.1k* | | | | | | |
| | A | 4.2 | | | – | | | | 4.2 | | | – | | | | 4.2 | | | – | | | |
| | N | .2 | .17 | .05 | .07 | .05 | .01 | .08 | .51 | .02 | .48 | .02 | .48 | .02 | 1 | .7 | .02 | .67 | .02 | .67 | .02 | 1 |
| AS-Sk. | | *sup.nodes/edges: 1.1M, 1.7M* | | | | | | | *sup.nodes/edges: 1.1M, 1.9M* | | | | | | | *sup.nodes/edges: 1.1M, 2.1M* | | | | | | |
| | A | 180 | | | – | | | | 180 | | | – | | | | 180 | | | – | | | |
| | N | 29 | 1 | .56 | 1 | .56 | .89 | .84 | 30 | 1 | .34 | 1 | .34 | .89 | .84 | 32 | 1 | .25 | 1 | .25 | .89 | .87 |

## A OTHER RELATED LITERATURE

**Experimental evaluations.** Kang *et al.* [41] evaluate how useful weights on superedges of summary graphs are. They focus on err$_p$ and size of the reconstructed graph, as well as PageRank and vertex-proximity queries. Unlike our work, those queries are evaluated by reconstructing on-the-fly the input graph from the summary.

Besta *et al.* [9] devise a programming model, a framework, and a query-processing evaluation for what they term *lossy graph compression*. By this term they mean "*any scheme that removes some parts of graphs*", thus something that goes beyond graph summarization. Among the techniques tested by Besta *et al.* is one graph-summarization method, specifically SWeG [74] (see above). However, the query-processing evaluation performed by Besta *et al.* on SWeG is carried out by *recostructing the whole graph from SWeG's summary*. This differs from the evaluation we perform in this work, which operates on the summary without reconstructing the graph.

**Other types of summary.** Summary graphs may have a form other than the one in Definition 2.1: they can describe a graph in terms of a given "vocabulary" of subgraph structures (e.g., stars, cliques, chains) [51], or have a hierarchical structure [55]. Those alternative types of summary are out of the scope of this work.

**Problem variants** of vanilla graph summarization include *meta-graph summarization*, whose goal is to apply meta-methods on top of basic graph-summarization methods, so that the resulting summaries exhibit properties that do not otherwise hold (e.g., vertex degree preservation in superedges [85]); *personalized* graph summarization [42], where summaries are tailored to a given set of target vertices; or summarization of more complex types of graph, such as dynamic/temporal graphs [32, 38, 79], or multi-relation graphs [43].

**Query-specific graph summarization.** All the discussions so far are about summaries that are *query-agnostic*, i.e., not tailored to any specific queries. There exist query-specific graph-summarization approaches too, for queries like reachability, distance, neighborhood, community membership [22–24, 36, 62, 71]. However, query-specific graph summarization is not a focus of this work.

**Related problems.** There exist problems that are similar in spirit to graph summarization, while still being different. The one that is usually termed *graph compression* (though there is no uniformity in the nomenclature in the literature) is concerned with developing data structures (and algorithms to manipulate them) to store (and retrieve) an exact representation of a graph using the minimum possible space [8, 11, 12]. A major difference with respect to graph summarization is that graph compression is not general: one needs to stick to the algorithms (and corresponding software) that have been ad-hoc developed for handling the data structures of that particular compression technique. Further differences include the fact that, typically, graph compression is lossless and operates at a lower level of graph representation (e.g., at a bit-level).

*Graph sparsification* (or *graph simplification*, or *graph backboning*) consists in reducing a graph by removing edges/vertices. As such, graph sparsification differs from graph summarization as it may completely discard parts of the graph in its output, while graph summarization guarantees that at least every vertex is part of the output summary. Another difference is that, although a few generalist approaches exist [20, 73, 75], graph sparsification is mostly tailored to preserve specific properties, such as shortest paths, degree distribution, spectral properties [4, 28, 76, 82, 83].

*Graph clustering* aims at finding groups of vertices that exhibit high intra-cluster connectivity and inter-cluster separation [3, 72]. This is different from graph summarization, which instead groups vertices with similar connection patterns with the rest of the graph.

**Table 5: GPQPS results for centrality queries. "A" stands for actual query processing on the original graph. Metrics: runtime (s), and precision ($P$) and recall ($R$) with varying the number of top-$s$-ranked vertices in the summary (w.r.t. the "ground-truth" of top-100-ranked vertices in the original graph). Closeness-centrality results on AS-Skitter refer to a subset of 1 000 randomly-sampled vertices.**

(a) PageRank centrality, S2L [70] summaries

| *data* | *meth.* | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* |
| Facebook | | *sup.nodes/edges: 350, 8.5k* | | | | | | | *sup.nodes/edges: 500, 14.4k* | | | | | | | *sup.nodes/edges: 750, 24.3k* | | | | | | |
| | A | 2.2 | – | | | | | | 2.2 | – | | | | | | 2.2 | – | | | | | |
| | N | .35 | .29 | .29 | .19 | .38 | .13 | .63 | .98 | .26 | .26 | .21 | .43 | .13 | .64 | .83 | .25 | .25 | .17 | .35 | .12 | .59 |
| | Nw | .34 | .32 | .32 | .22 | .45 | .16 | .78 | .56 | .36 | .36 | .23 | .45 | .15 | .76 | .88 | .33 | .33 | .2 | .4 | .14 | .69 |
| | P | 20 | .35 | .27 | .24 | .35 | .16 | .61 | 40 | .39 | .28 | .24 | .41 | .16 | .65 | 71 | .33 | .25 | .22 | .35 | .13 | .58 |
| | Pa | 21 | .35 | .27 | .24 | .35 | .16 | .61 | 40 | .39 | .28 | .24 | .41 | .16 | .65 | 72 | .33 | .25 | .22 | .35 | .13 | .58 |
| | Pe | 21 | .35 | .29 | .26 | .43 | .18 | .73 | 39 | .36 | .3 | .23 | .42 | .16 | .7 | 72 | .32 | .27 | .21 | .36 | .14 | .67 |
| LastFM | | *sup.nodes/edges: 500, 6.8k* | | | | | | | *sup.nodes/edges: 750, 10.4k* | | | | | | | *sup.nodes/edges: 1k, 12.8k* | | | | | | |
| | A | 1.7 | – | | | | | | 1.7 | – | | | | | | 1.7 | – | | | | | |
| | N | .22 | .65 | .65 | .43 | .87 | .19 | .97 | .31 | .66 | .66 | .42 | .85 | .2 | .98 | .36 | .7 | .7 | .43 | .87 | .19 | .96 |
| | Nw | .26 | .66 | .66 | .43 | .86 | .19 | .97 | .36 | .69 | .69 | .42 | .85 | .2 | 1 | .43 | .7 | .7 | .45 | .89 | .2 | .98 |
| | P | 14.7 | .8 | .57 | .47 | .79 | .22 | .97 | 20.3 | .79 | .61 | .51 | .79 | .21 | .97 | 26 | .79 | .65 | .5 | .83 | .22 | .96 |
| | Pa | 15 | .8 | .57 | .47 | .79 | .22 | .97 | 20.8 | .79 | .61 | .51 | .79 | .21 | .97 | 27 | .79 | .65 | .5 | .83 | .22 | .96 |
| | Pe | 15.5 | .76 | .61 | .45 | .79 | .21 | .97 | 20.9 | .72 | .65 | .46 | .83 | .21 | .99 | 28 | .76 | .66 | .47 | .85 | .21 | .97 |
| Enron | | *sup.nodes/edges: 1k, 50.1k* | | | | | | | *sup.nodes/edges: 1.5k, 68.4k* | | | | | | | *sup.nodes/edges: 2k, 81.4k* | | | | | | |
| | A | 24 | – | | | | | | 24 | – | | | | | | 24 | – | | | | | |
| | N | 1.2 | .54 | .54 | .35 | .71 | .17 | .87 | 1.7 | .56 | .56 | .35 | .7 | .17 | .87 | 2.2 | .58 | .58 | .36 | .72 | .17 | .87 |
| | Nw | 1.4 | .53 | .53 | .34 | .68 | .17 | .87 | 2.1 | .56 | .56 | .35 | .7 | .17 | .86 | 2.8 | .57 | .57 | .36 | .73 | .17 | .87 |
| | P | 80 | .6 | .5 | .35 | .66 | .18 | .85 | 123 | .6 | .53 | .37 | .68 | .18 | .84 | 161 | .61 | .55 | .38 | .7 | .18 | .85 |
| | Pa | 82 | .6 | .5 | .35 | .66 | .18 | .85 | 126 | .6 | .53 | .37 | .68 | .18 | .84 | 160 | .61 | .55 | .38 | .7 | .18 | .85 |
| | Pe | 83 | .57 | .52 | .35 | .67 | .18 | .88 | 130 | .59 | .55 | .36 | .69 | .18 | .86 | 167 | .59 | .56 | .37 | .72 | .18 | .87 |
| Gnutella | | *sup.nodes/edges: 500, 2k* | | | | | | | *sup.nodes/edges: 750, 3.4k* | | | | | | | *sup.nodes/edges: 1k, 5k* | | | | | | |
| | A | .76 | – | | | | | | .76 | – | | | | | | .76 | – | | | | | |
| | N | .08 | .62 | .62 | .4 | .8 | .18 | .9 | .11 | .65 | .65 | .4 | .8 | .17 | .87 | .15 | .69 | .69 | .41 | .81 | .18 | .89 |
| | Nw | .1 | .55 | .55 | .38 | .76 | .18 | .92 | .15 | .55 | .55 | .39 | .78 | .18 | .88 | .19 | .61 | .61 | .4 | .8 | .18 | .88 |
| | P | 5.8 | 1 | .01 | .9 | .09 | .19 | .92 | 8.3 | 0 | 0 | 1 | .08 | .85 | .39 | 11 | 0 | 0 | 1 | .06 | .92 | .33 |
| | Pa | 5.9 | 1 | .01 | .9 | .09 | .19 | .92 | 8.6 | 0 | 0 | 1 | .08 | .85 | .39 | 11 | 0 | 0 | 1 | .06 | .92 | .33 |
| | Pe | 6.1 | 1 | .01 | .89 | .08 | .19 | .92 | 8.8 | 0 | 0 | 1 | .08 | .84 | .38 | 12 | 1 | .01 | 1 | .05 | .92 | .33 |
| Ubuntu | | *sup.nodes/edges: 350, 35.3k* | | | | | | | *sup.nodes/edges: 500, 55.3k* | | | | | | | *sup.nodes/edges: 750, 82.6k* | | | | | | |
| | A | 4.1 | – | | | | | | 4.1 | – | | | | | | 4.1 | – | | | | | |
| | N | .66 | .79 | .79 | .5 | 1 | .19 | 1 | 1.1 | .8 | .8 | .49 | .99 | .2 | 1 | 1.8 | .8 | .8 | .49 | .99 | .2 | 1 |
| | Nw | .94 | .95 | .95 | .5 | 1 | .19 | 1 | 1.5 | .96 | .96 | .5 | 1 | .2 | 1 | 2.3 | .97 | .97 | .5 | 1 | .2 | 1 |
| | P | 58 | .83 | .74 | .52 | 1 | .2 | 1 | 94 | .82 | .75 | .53 | .99 | .21 | 1 | 151 | .84 | .72 | .53 | .99 | .2 | 1 |
| | Pa | 79 | .97 | .94 | .51 | 1 | .2 | 1 | 125 | .97 | .95 | .51 | 1 | .2 | 1 | 190 | .98 | .96 | .51 | 1 | .2 | 1 |
| | Pe | 79 | .97 | .94 | .51 | 1 | .19 | 1 | 127 | .97 | .95 | .51 | 1 | .2 | 1 | 192 | .98 | .96 | .51 | 1 | .2 | 1 |
| AS-Skitter | | *sup.nodes/edges: 1k, 39k* | | | | | | | *sup.nodes/edges: 10k, 462k* | | | | | | | *sup.nodes/edges: 25k, 983k* | | | | | | |
| | A | 66 | – | | | | | | 66 | – | | | | | | 66 | – | | | | | |
| | N | .84 | .34 | .34 | .27 | .54 | .15 | .74 | 2.17 | .27 | .27 | .21 | .43 | .14 | .71 | 34.7 | .29 | .29 | .26 | .51 | .13 | .66 |
| | Nw | .85 | .35 | .36 | .28 | .57 | .16 | .82 | 20.1 | .36 | .36 | .24 | .49 | .16 | .79 | 3.79 | .45 | .45 | .3 | .6 | .16 | .79 |
| | P | 212 | .33 | .19 | .33 | .43 | .26 | .71 | 383 | .38 | .27 | .3 | .46 | .16 | .69 | 738 | .47 | .4 | .32 | .53 | .18 | .74 |
| | Pa | 243 | .33 | .19 | .33 | .43 | .26 | .71 | 444 | .38 | .27 | .3 | .46 | .16 | .69 | 707 | .47 | .4 | .32 | .53 | .18 | .74 |
| | Pe | 222 | .33 | .23 | .32 | .46 | .23 | .69 | 391 | .37 | .31 | .27 | .46 | .16 | .72 | 727 | .5 | .46 | .35 | .57 | .17 | .76 |

(b) Closeness centrality, S2L [70] summaries

| *data* | *meth.* | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* |
| Facebook | | *sup.nodes/edges: 350, 8.5k* | | | | | | | *sup.nodes/edges: 500, 14.4k* | | | | | | | *sup.nodes/edges: 750, 24.3k* | | | | | | |
| | A | 23 | – | | | | | | 23 | – | | | | | | 23 | – | | | | | |
| | N | .24 | .23 | .23 | .14 | .29 | .09 | .44 | .43 | .22 | .22 | .15 | .31 | .08 | .41 | 1.97 | .35 | .35 | .18 | .36 | .09 | .47 |
| | Nw | 3 | .19 | .2 | .11 | .22 | .08 | .41 | 6.7 | .18 | .18 | .15 | .31 | .08 | .41 | 26 | .15 | .19 | .12 | .25 | .09 | .44 |
| | P | 19 | .26 | .22 | .15 | .27 | .09 | .4 | 45 | .32 | .28 | .16 | .31 | .08 | .4 | 166 | .35 | .32 | .18 | .34 | .1 | .47 |
| | Pa | 127 | .26 | .22 | .15 | .27 | .09 | .4 | 437 | .32 | .28 | .16 | .31 | .08 | .4 | 1.2k | .35 | .32 | .18 | .34 | .1 | .47 |
| | Pe | 142 | .25 | .16 | .14 | .21 | .09 | .37 | 454 | 0 | 0 | .18 | .27 | .09 | .37 | 1.4k | 1 | .01 | .41 | .07 | .1 | .43 |
| LastFM | | *sup.nodes/edges: 500, 6.8k* | | | | | | | *sup.nodes/edges: 750, 10.4k* | | | | | | | *sup.nodes/edges: 1k, 12.8k* | | | | | | |
| | A | 61 | – | | | | | | 61 | – | | | | | | 61 | – | | | | | |
| | N | .29 | .56 | .59 | .32 | .65 | .18 | .91 | .81 | .47 | .49 | .27 | .55 | .19 | .93 | 1.3 | .42 | .44 | .26 | .53 | .11 | .55 |
| | Nw | 3.3 | .15 | .15 | .15 | .33 | .18 | .91 | 8.7 | .11 | .11 | .09 | .22 | .13 | .68 | 13 | .05 | .05 | .06 | .13 | .07 | .36 |
| | P | 32 | .74 | .61 | .47 | .84 | .19 | .91 | 76 | .8 | .6 | .51 | .88 | .2 | .95 | 125 | .85 | .7 | .54 | .92 | .21 | .97 |
| | Pa | 147 | .74 | .61 | .47 | .84 | .19 | .91 | 337 | .8 | .6 | .51 | .88 | .2 | .95 | 617 | .85 | .7 | .54 | .92 | .21 | .97 |
| | Pe | 158 | 1 | .02 | 1 | .14 | .21 | .91 | 367 | 1 | .01 | 1 | .03 | .31 | .94 | 666 | 0 | 0 | 1 | .03 | .8 | .72 |
| Enron | | *sup.nodes/edges: 1k, 50.1k* | | | | | | | *sup.nodes/edges: 1.5k, 68.4k* | | | | | | | *sup.nodes/edges: 2k, 81.4k* | | | | | | |
| | A | 1.2k | – | | | | | | 1.2k | – | | | | | | 1.2k | – | | | | | |
| | N | 3.8 | .75 | .76 | .45 | .92 | .19 | .97 | 8.8 | .75 | .75 | .46 | .91 | .19 | .98 | 14 | .74 | .76 | .45 | .9 | .19 | .97 |
| | Nw | 56 | .02 | .03 | .02 | .04 | .08 | .4 | 112 | 0 | 0 | .01 | .04 | .02 | .12 | 185 | .01 | .02 | .01 | .02 | .02 | .13 |
| | P | 263 | .91 | .78 | .54 | .98 | .2 | .98 | 610 | .89 | .78 | .54 | .98 | .21 | .99 | 1.1k | .89 | .82 | .53 | .98 | .21 | .99 |
| | Pa | 2.5k | .91 | .78 | .54 | .98 | .2 | .98 | 5.8k | .89 | .78 | .54 | .98 | .21 | .99 | 10k | .89 | .82 | .53 | .98 | .21 | .99 |
| | Pe | 2.7k | 0 | 0 | 0 | 0 | .95 | .21 | 6.4k | 0 | 0 | 0 | 0 | 1 | .03 | 11k | 0 | 0 | 0 | 0 | 1 | .01 |
| Gnutella | | *sup.nodes/edges: 500, 2k* | | | | | | | *sup.nodes/edges: 750, 3.4k* | | | | | | | *sup.nodes/edges: 1k, 5k* | | | | | | |
| | A | 42 | – | | | | | | 42 | – | | | | | | 42 | – | | | | | |
| | N | .06 | .64 | .72 | .29 | .73 | .14 | .73 | .14 | .62 | .72 | .36 | .73 | .14 | .73 | .93 | .76 | .79 | .3 | .8 | .12 | .8 |
| | Nw | .94 | .04 | .04 | .03 | .08 | .14 | .73 | 2.9 | .03 | .05 | .02 | .05 | .02 | .09 | 5.7 | .05 | .08 | .04 | .1 | .03 | .15 |
| | P | 12 | 1 | .21 | .92 | .46 | .81 | .55 | 37 | .96 | .24 | .84 | .41 | .76 | .6 | 83 | 1 | .2 | .89 | .42 | .8 | .67 |
| | Pa | 22 | 1 | .21 | .92 | .46 | .81 | .55 | 77 | .96 | .24 | .84 | .41 | .76 | .6 | 181 | 1 | .2 | .89 | .42 | .8 | .67 |
| | Pe | 23 | 1 | .22 | .92 | .46 | .72 | .58 | 77 | 1 | .11 | .84 | .38 | .76 | .6 | 181 | 1 | .06 | .95 | .36 | .79 | .67 |
| Ubuntu | | *sup.nodes/edges: 350, 35.3k* | | | | | | | *sup.nodes/edges: 500, 55.3k* | | | | | | | *sup.nodes/edges: 750, 82.6k* | | | | | | |
| | A | 471 | – | | | | | | 471 | – | | | | | | 471 | – | | | | | |
| | N | .32 | .45 | .46 | .28 | .56 | .15 | .77 | .79 | .51 | .5 | .34 | .68 | .16 | .81 | 1.2 | .52 | .51 | .36 | .71 | .18 | .88 |
| | Nw | 12 | .06 | .06 | .09 | .18 | .15 | .77 | 28 | .04 | .04 | .06 | .12 | .14 | .68 | 65 | 0 | 0 | 0 | 0 | .06 | .3 |
| | P | 25 | .5 | .46 | .29 | .56 | .15 | .77 | 52 | .54 | .5 | .34 | .63 | .17 | .81 | 102 | .53 | .5 | .36 | .67 | .18 | .88 |
| | Pa | 821 | .37 | .21 | .21 | .32 | .16 | .77 | 1.9k | .6 | .39 | .35 | .52 | .17 | .81 | 4.6k | .74 | .48 | .47 | .73 | .18 | .85 |
| | Pe | 882 | 0 | 0 | 0 | 0 | .16 | .77 | 2k | 0 | 0 | 0 | 0 | .19 | .81 | 5k | 0 | 0 | 0 | 0 | .56 | .3 |
| AS-Skitter | | *sup.nodes/edges: 1k, 39k* | | | | | | | *sup.nodes/edges: 10k, 462k* | | | | | | | *sup.nodes/edges: 25k, 983k* | | | | | | |
| | A | 2.5k | – | | | | | | 2.5k | – | | | | | | 2.5k | – | | | | | |
| | N | .46 | .5 | .5 | .4 | .83 | .1 | 1 | 53.4 | .42 | .42 | .37 | .75 | .1 | 1 | 115 | .34 | .34 | .32 | .63 | .1 | 1 |
| | Nw | 3.89 | .5 | .5 | .39 | .79 | .1 | 1 | 266 | .43 | .43 | .38 | .75 | .1 | 1 | 894 | .33 | .33 | .32 | .64 | .1 | 1 |
| | P | 11.3 | .52 | .5 | .4 | .79 | .1 | 1 | 630 | .46 | .41 | .38 | .71 | .1 | .96 | 3.3k | .35 | .32 | .33 | .61 | .2 | .95 |
| | Pa | 58.7 | .52 | .5 | .4 | .79 | .1 | 1 | 4.5k | .46 | .41 | .38 | .71 | .1 | .96 | 21.8k | .35 | .32 | .33 | .61 | .2 | .95 |
| | Pe | 25.1 | .97 | .28 | .66 | .43 | .09 | .76 | 3.7k | .7 | .26 | .64 | .65 | .11 | .92 | 20.1k | .56 | .24 | .54 | .52 | .11 | .96 |

(c) PageRank centrality, SWeG [74] summaries

| *data* | *meth.* | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* |
| F.book | | *sup.nodes/edges: 708, 664* | | | | | | | *sup.nodes/edges: 977, 2.2k* | | | | | | | *sup.nodes/edges: 1.2k, 4.4k* | | | | | | |
| | A | 2.2 | – | | | | | | 2.2 | – | | | | | | 2.2 | – | | | | | |
| | N | .06 | .03 | .04 | .02 | .04 | .01 | .05 | .12 | .13 | .13 | .07 | .13 | .03 | .13 | .19 | .16 | .16 | .09 | .17 | .03 | .17 |
| L.FM | | *sup.nodes/edges: 3.4k, 1.2k* | | | | | | | *sup.nodes/edges: 3.6k, 1.6k* | | | | | | | *sup.nodes/edges: 3.8k, 2k* | | | | | | |
| | A | 1.7 | – | | | | | | 1.7 | – | | | | | | 1.7 | – | | | | | |
| | N | .16 | 0 | 0 | 0 | .01 | 0 | .01 | .18 | .01 | .01 | 0 | .01 | 0 | .02 | .2 | .01 | .01 | .01 | .01 | 0 | .01 |
| Enron | | *sup.nodes/edges: 22.8k, 10.2k* | | | | | | | *sup.nodes/edges: 24k, 28.8k* | | | | | | | *sup.nodes/edges: 24.4k, 44.1k* | | | | | | |
| | A | 24 | – | | | | | | 24 | – | | | | | | 24 | – | | | | | |
| | N | .81 | .12 | .12 | .06 | .12 | .02 | .12 | 1.9 | .67 | .67 | .34 | .67 | .13 | .67 | 2.1 | .76 | .76 | .38 | .76 | .15 | .76 |
| AS-Sk. | | *sup.nodes/edges: 1.1M, 1.7M* | | | | | | | *sup.nodes/edges: 1.1M, 1.9M* | | | | | | | *sup.nodes/edges: 1.1M, 2.1M* | | | | | | |
| | A | 66 | – | | | | | | 66 | – | | | | | | 66 | – | | | | | |
| | N | 12.5 | .64 | .64 | .4 | .8 | .19 | .96 | 9.9 | .65 | .65 | .39 | .78 | .19 | .95 | 13.2 | .64 | .64 | .39 | .78 | .18 | .92 |

(d) Closeness centrality, SWeG [74] summaries

| *data* | *meth.* | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | | *runt.* | $s=100$ | | $s=200$ | | $s=500$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* | | *P* | *R* | *P* | *R* | *P* | *R* |
| F.book | | *sup.nodes/edges: 708, 664* | | | | | | | *sup.nodes/edges: 977, 2.2k* | | | | | | | *sup.nodes/edges: 1.2k, 4.4k* | | | | | | |
| | A | 23 | – | | | | | | 23 | – | | | | | | 23 | – | | | | | |
| | N | .04 | .02 | .03 | .02 | .04 | .01 | .06 | .54 | .14 | .14 | .09 | .17 | .03 | .17 | .94 | .16 | .16 | .09 | .18 | .04 | .22 |
| L.FM | | *sup.nodes/edges: 3.4k, 1.2k* | | | | | | | *sup.nodes/edges: 3.6k, 1.6k* | | | | | | | *sup.nodes/edges: 3.8k, 2k* | | | | | | |
| | A | 61 | – | | | | | | 61 | – | | | | | | 61 | – | | | | | |
| | N | .02 | 0 | 0 | 0 | 0 | 0 | .01 | .17 | 0 | 0 | 0 | 0 | 0 | 0 | .95 | .05 | .05 | .03 | .05 | .01 | .05 |
| Enron | | *sup.nodes/edges: 22.8k, 10.2k* | | | | | | | *sup.nodes/edges: 24k, 28.8k* | | | | | | | *sup.nodes/edges: 24.4k, 44.1k* | | | | | | |
| | A | 1.2k | – | | | | | | 1.2k | – | | | | | | 1.2k | – | | | | | |
| | N | 18 | .06 | .06 | .03 | .06 | .01 | .06 | 209 | .37 | .37 | .23 | .46 | .12 | .58 | 270 | .66 | .66 | .35 | .7 | .14 | .7 |
| AS-Sk. | | *sup.nodes/edges: 1.1M, 1.7M* | | | | | | | *sup.nodes/edges: 1.1M, 1.9M* | | | | | | | *sup.nodes/edges: 1.1M, 2.1M* | | | | | | |
| | A | 2.5k | – | | | | | | 2.5k | – | | | | | | 2.5k | – | | | | | |
| | N | 663 | .24 | .24 | .12 | .24 | .09 | .45 | 712 | .18 | .18 | .09 | .18 | .08 | .39 | 850 | .19 | .19 | .1 | .19 | .06 | .3 |

## B FURTHER EXPERIMENTAL RESULTS

Tables 4–5 show GPQPS results for core decomposition, PageRank centrality, and closeness centrality queries, with varying the summary size.