Social Network Analysis and Mining (2024) 14:157
https://doi.org/10.1007/s13278-024-01314-w

ORIGINAL ARTICLE

General-purpose query processing on summary graphs

Aris Anagnostopoulos[1] · Valentina Arrigoni[2] · Francesco Gullo[3] · Giorgia Salvatori[4] · Lorenzo Severini[2]

Received: 3 June 2024 / Revised: 7 July 2024 / Accepted: 22 July 2024

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 nowadays query processing on summary graphs is 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-management tasks that rely on scalable graph query processing, including social network analysis. We set the stage of this problem, by devising basic, yet principled algorithms, and thoroughly analyzing their peculiarities and capabilities of performing well in practice, 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.

Keywords Graph data management · Scalable graph processing · Graph summarization · Query processing

1 Introduction

Graphs, which are sets of entities (vertices) linked to one another (via edges), have become a ubiquitous real-world data-representation model (Aggarwal and Wang 2010a). In many domains, graphs are so large that it is hard to directly process them in their raw form. For instance, modern social networks count billions or even more vertices and edges. 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 (Besta and Hoefler 2018; Liu et al. 2018). 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 (Besta and Hoefler 2018). *Graph sparsification* (a.k.a., *graph simplification*, or *graph backboning*) reduces a graph by removing edges/vertices (Coscia and Neffke 2017; Fung et al. 2019; Zeng et al. 2021). *Graph summarization* produces a coarse-grained version of the original graph—a *summary graph*—by grouping vertices and edges into *supernodes* and *superedges* (Beg et al. 2018; Hajiabadi et al. 2021; Khan et al. 2015; Ko et al. 2020; Kumar and Efstathopoulos 2018; Lee et al. 2020; LeFevre and Terzi 2010; Liu et al. 2014; Navlakha et al. 2008; Riondato et al. 2017; Shin et al.

✉ Francesco Gullo
francesco.gullo@univaq.it

Aris Anagnostopoulos
aris@diag.uniroma1.it

Valentina Arrigoni
valentina.arrigoni@unicredit.eu

Giorgia Salvatori
g.salvatori@reply.it

Lorenzo Severini
lorenzo.severini@unicredit.eu

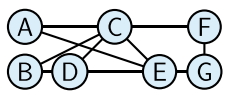
[1] Sapienza University, Rome, Italy

[2] UniCredit, Milan/Rome, Italy

[3] University of L'Aquila, L'Aquila, Italy

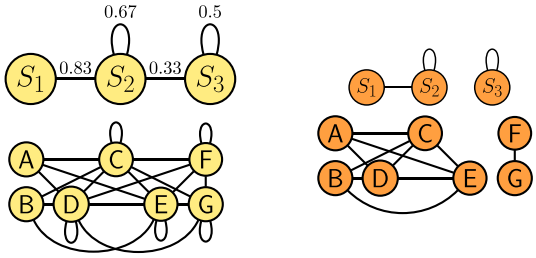
[4] Target Reply, Rome, Italy

Published online: 09 August 2024



Supernodes: $S_1 = \{AB\}$
$S_2 = \{CDE\}$ $S_3 = \{FG\}$

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



(b) Summary graphs (Def. 1) and corresponding reconstructed graphs (Def. 2) according to S2L [16] (left) and SWeG [17] (right)

Fig. 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

2019; Toivonen et al. 2011; Yong et al. 2021) (see Fig. 1).[1] Each of these graph-reduction approaches comes with its own pros and cons, effectiveness, and suitability, 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 guarantees that each vertex of the original graph forms a part of the output summary graph (through the corresponding supernode).

All these peculiarites make graph summarization a problem particularly appealing for the social-network context, where graphs to be processed are big, and keeping handling (a size-reduced version of) them with standard methodologies and data structures is highly desirable for purposes of generality and software reusability. This is eagerly required, for example, in social-network analysis, which heavily relies on well-established and acknowledged tools.

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; see Sect. 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 (Hajiabadi et al. 2021; Lee et al. 2022; Kang et al. 2022a, b; Shin et al. 2019). Conversely, existing methods that exploit the summary graph only (without reconstructing the input graph) are query-specific (Hajiabadi et al. 2021; LeFevre and Terzi 2010; Riondato et al. 2014, 2017). To the best of our knowledge, no query-processing method 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 (see Definition 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 was a normal graph. The second method interprets a summary graph as an *uncertain graph* (Khan et al. 2018), and adopts Monte-Carlo-sampling-based query processing on it.

We remark that 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. As such, *our primary contribution in this paper is not really algorithmic*. Rather, here we are mainly interested in *setting the stage of* the novel GPQPS problem. Hence, we purposely focus on basic, immediate (but anyway principled) algorithms, with the main goal of providing conceptual/experimental insights, highlighting both pros and cons.

Conversely, we point out 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

[1] Although other notions of "summary graph" do exist in the graph-summarization literature, in this work we consider the one that collapses nodes and edges into supernodes and superedges (Definition 1), which is the most common one.

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, we believe that setting the stage of GPQPS helps support the large variety of data-management applications that depend on scalable graph query processing, including social network analysis (Biafore and Nawab 2016; Fazzone et al. 2022; Galimberti et al. 2021; Jin et al. 2023; Lanciano et al. 2023; Mosa et al. 2017; ur Rehman et al. 2021). 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. Our main contributions are as follows:

- 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—mainly conceptually and experimentally—and defining a principled starting point on it for researchers and practitioners (Sect. 3).
- We devise simple algorithms for GPQPS, which are intended to be a reference for the problem and constitute a basic bar, to be raised by future work (Sect. 4).
- We set up an evaluation methodology that constitutes a benchmark testbed for this and future GPQPS studies (Sect. 5).
- We perform extensive experiments, to derive insights on the practical impact and obstacles to an effective employment of the GPQPS methods (Sects. 6–7).
- We provide nontrivial directions for further research (Sect. 8). 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.

2 Preliminaries and related work

The problem of *graph summarization* (Liu et al. 2018) 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 $w : E \to \mathbb{R}_{>0}$ 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 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 $\omega : \mathcal{E} \to \mathbb{R}_{>0}$, such that every vertex $u \in V$ is assigned to one and only one $S \in \mathcal{S}$: $\forall S \in \mathcal{S} : S \subseteq V$, $\forall S, T \in \mathcal{S}, S \neq T : S \cap T = \emptyset$, $\bigcup_{S\in\mathcal{S}} = V$. 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 (Kang et al. 2022a). 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 (Reconstructed graph) Given a summary $\mathcal{G}$ of a graph G, 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 (Liu et al. 2018). 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 1 and 2.

Lossless vs. lossy graph summarization Regardless of the formulation, graph summarization can be either *lossless* or *lossy*. In the first case, 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 (Ko et al. 2020; Navlakha et al. 2008; Shin et al. 2019; Yong et al. 2021), 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.

2.1 Size-driven graph summarization

Let $d(\mathcal{G}, G)$ be an *error* function that quantifies how good a summary $\mathcal{G}$ is for a 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 κ supernodes that minimizes $d(\mathcal{G}, G)$ (Beg et al. 2018; Lee et al. 2020; LeFevre and Terzi 2010; Liu et al. 2014; Riondato et al. 2014, 2017; Toivonen et al. 2011). Existing approaches of size-driven graph summarization differ from each other in the specific error function employed.

The ℓ_p*-reconstruction-error* is a popular error function (Beg et al. 2018; LeFevre and Terzi 2010; Riondato et al. 2014, 2017). 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:

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

The *lifted adjacency matrix* given $\mathcal{G}$ is defined as a $|V| \times |V|$ matrix $M^{\uparrow}$ whose $M^{\uparrow}[u, v]$ cell contains 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 ℓ_p-reconstruction error 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, $\mathrm{err}_p(\mathcal{G}, G) = \|M - M^{\uparrow}\|_p$.

LeFevre and Terzi (2010) deal with err_1 and propose a greedy heuristic algorithm that resembles an agglomerative hierarchical clustering using Ward's method (Ward 1963). Riondato et al. (2014, 2017) establish a connection between err_p-based graph summarization and ℓ_p^p*-clustering*, and design algorithms with constant-factor approximation guarantees for err_1 and err_2. Among Riondato *et al.*'s algorithms, the S2L one targets err_2 and employs k-median clustering, which is tackled by Lloyd's iterative approach (Lloyd 1982).

Beg et al. (2018) and Lee et al. (2020) develop techniques that are mainly focused on improving the efficiency of the aforementioned algorithms.

Other error functions adopted in the literature include *cut-norm error* (Riondato et al. 2017), *representation error* (Liu et al. 2014), and *path-based error* (Toivonen et al. 2011; Zhou et al. 2017).

2.2 Utility-driven graph summarization

This 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 (Hajiabadi et al. 2021; Khan et al. 2015; Ko et al. 2020; Kumar and Efstathopoulos 2018; Navlakha et al. 2008; Shin et al. 2019; Yong et al. 2021). Navlakha et al. (2008) adopt a utility function based on the *representation error* (see above). Shin et al. (2019) introduce SWeG, a parallel algorithm for Navlakha et al.'s formulation. Yong et al. (2021) further improve SWeG through *locality sensitive hashing* (Indyk and Motwani 1998). Khan et al. (2015) and Ko et al. (2020) devise a set-based method and an incremental algorithm, respectively, for a *lossless* variant of Navlakha et al.'s formulation. Kumar and Efstathopoulos (2018) and Hajiabadi et al. (2021) define the utility in terms of the *importance* of the edges in the reconstructed graph. The importance of an edge is computed according to any user-defined function (e.g., edge centrality).

2.3 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 (LeFevre and Terzi 2010; Riondato et al. 2014, 2017; Toivonen et al. 2011; Zhou et al. 2017). As for directed graphs, to the best of our knowledge, only Riondato et al. (2017) handle them: they propose to decompose the input adjacency matrix into the sum of a *symmetric* matrix and a *skew-symmetric* matrix, and compute a summary for each of 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.

2.4 Query processing on summary graphs

A graph summary corresponds to a succinct representation of a graph (even though possibly lossy). As such, a major

use of a summary graph is to speedup query processing on graphs.

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 when processing the target query (Hajiabadi et al. 2021; Lee et al. 2022; Kang et al. 2022a, b; Shin et al. 2019). This strategy is beneficial for space complexity, as it keeps in memory only the summary. However, it gives no speedup in runtime, as reconstructed neighborhoods are typically larger than actual neighborhoods, at least on average.[2]

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 (LeFevre and Terzi 2010; Riondato et al. 2014, 2017), triangle counting (Riondato et al. 2014, 2017), eigenvector centrality (LeFevre and Terzi 2010; Riondato et al. 2014, 2017), or, in the case of lossless summaries, PageRank and shortest path (Hajiabadi et al. 2021).

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*.

2.5 Other (marginally) related literature

Experimental evaluations Kang et al. (2022a) evaluate how useful are weights on superedges of summary graphs. They focus on err_p and the 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. (2019) 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., there is one graph-summarization method, specifically SWeG (Shin et al. 2019) (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.

[2] Experimental evidence of this claim is reported in Lee et al. (2022), for PageRank, triangle counting, breadth first search, and shortest path queries.

Other types of summary Summary graphs may have a form other than the one in Definition 1: they can describe a graph in terms of a given "vocabulary" of subgraph structures (e.g., stars, cliques, chains) (Koutra et al. 2014), or have a hierarchical structure (Lee et al. 2022). 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 (Zhou et al. 2021)); *personalized* graph summarization (Kang et al. 2022b), where summaries are tailored to a given set of target vertices; or summarization of more complex types of graph, such as dynamic/temporal graphs (Gou et al. 2019; Jiang et al. 2023; Tsalouchidou et al. 2020), or multi-relation graphs (Ke et al. 2022).

Query-specific graph summarization All the discussions so far are about summaries that are *query-agnostic*, that is, not tailored to any specific queries. There exist query-specific graph-summarization approaches too, for queries such as reachability, distance, neighborhood, community membership (Fan et al. 2012, 2021, 2022; Hernández and Navarro 2014; Maserrat and Pei 2010; Sadri et al. 2017). 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 (Besta and Hoefler 2018; Boldi et al. 2009; Boldi and Vigna 2004). 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 general approaches exist (Coscia and Neffke 2017; Serrano et al. 2009; Slater 2009), graph sparsification is mostly tailored to preserve specific properties, such as shortest paths, degree distribution, or spectral properties (Fung et al. 2019; Ahn

et al. 2012; Spielman and Teng 2011; Zeng et al. 2021; Zhou et al. 2010).

Graph clustering aims at finding groups of vertices with high intra-cluster connectivity and inter-cluster separation (Aggarwal and Wang 2010b; Schaeffer 2007). This differs from graph summarization, which groups vertices with similar connection patterns with the rest of the graph.

3 Problem statement

Graph queries A *(graph) query* Q is a computable function whose input is a graph $G = (V, E, w)$ and *(query) context* $\mathcal{C}$, and whose output is an object from a certain domain $\mathcal{O}_Q$.[3] Context $\mathcal{C}$ identifies the complementary input of the query. It may correspond to pairs or sets of vertices, subgraphs, functions, numerical values, and so on, or it can also be empty. The output of a query Q (with context $\mathcal{C}$) on a graph G is denoted by $Q(G, \mathcal{C})$, 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, diameter) have $\mathcal{C} = \emptyset$ and $\mathcal{O}_Q = \mathbb{R}$. *Node embedding* queries take a vertex u as a context $\mathcal{C}$, and output a d-dimensional numerical vector representing the embedding of u (thus, $\mathcal{O}_Q = \mathbb{R}^d$). *Inner-most core* queries have $\mathcal{C} = \emptyset$, and the output is the vertices of the k-core (Batagelj and Zaversnik 2011) of G with the highest k (thus, $\mathcal{O}_Q = 2^V$). *Reachability* queries (in directed graphs) take an ordered vertex pair (u, v) as a context $\mathcal{C}$, and output a Boolean stating whether v is reachable from u (thus, $\mathcal{O}_Q = \{\mathsf{T}, \mathsf{F}\}$). In *top-ranked centrality* queries, $\mathcal{C}$ corresponds to an integer s, and the output is the top-s—ranked vertices according to a certain centrality (thus, $\mathcal{O}_Q = 2^V$). In *community detection* queries, $\mathcal{C}$ 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, $\mathcal{O}_Q = \mathbf{B}_V$, 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., $\mathcal{O}_Q \in \{\mathbb{R}^d, 2^V, \mathbf{B}_V\}$). 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.

Answering queries from summaries 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 what we term *summary-based approximate query answers*:

Definition 3 (Summary-based approximate query answer) Given a graph G, a summary $\mathcal{G}$ of G, and a query Q on G with context $\mathcal{C}$, a *summary-based approximate answer* to Q on $\mathcal{G}$—denoted $\widetilde{Q}(G, \mathcal{G})$—is an approximation of $Q(G, \mathcal{C})$ obtained by making use only of $\mathcal{G}$.

GPQPS problem With the above notation and discussion in place, we can now state the problem that we tackle in this work:

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 $\mathcal{C}$, compute $\widetilde{Q}(G, \mathcal{G})$ that is the closest to $Q(G, \mathcal{C})$.

Simply speaking, Problem 1 asks for summary-based query answers which approximate well the true answer to the given query.

4 Algorithms

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

Definition 4 (Summary-aware query context) Given context $\mathcal{C}$ of a query Q on a graph G, the *summary-aware query context* of $\mathcal{C}$ on a summary $\mathcal{G}$ of G—denoted by $\mathcal{C}_\mathcal{G}$—is a copy of $\mathcal{C}$ where every vertex $u \in \mathcal{C}$ is replaced with the supernode S_u of $\mathcal{G}$ it belongs to.

Definition 5 (Summary-processed query answer) Let Q be a query on a graph G with context $\mathcal{C}$, $\mathcal{G}$ be a summary of G, and $\mathcal{C}_\mathcal{G}$ be the summary-aware context of $\mathcal{C}$ on $\mathcal{G}$. The *summary-processed answer* to Q on $\mathcal{G}$—denoted by $Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$—is the answer obtained by processing Q on $\mathcal{G}$ with context $\mathcal{C}_\mathcal{G}$ 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 $Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$, 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

[3] 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 (Fan et al. 2012, 2021, 2022).

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, $\widetilde{Q}(G, \mathcal{G}) = \mathbf{c} \circ Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$, 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 $Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$ 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).[4]

Definition 6 (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 (Abiteboul et al. 1987; Dalvi and Suciu 2004), an uncertain graph $\mathbb{G} = (V, E, \pi)$ is interpreted as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ deterministic graphs (*worlds*), each defined by a subset of E. Assuming independence among edge probabilities (Khan et al. 2018), 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 $\pi : \mathcal{E} \to (0, 1]$ assigning existence probabilities to superedges. Function π can be defined, for example, as the expected number of edges between two supernodes (as in err_p, see Equation (1)). However, the algorithm is independent of the definition of π. Also, no relationship is required between π and $\mathcal{G}$: π may have been (as, for example, the S2L graph-summarization method; see Sect. 2), or may have not been (e.g., the SWeG method) exploited for computing $\mathcal{G}$. Given $\mathcal{G}$ and π, 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 (Khan et al. 2018). 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 $r(S, T) \le \pi(S, T)$. On every world $\mathcal{W}$ the query is processed

Algorithm 1 Naïve-GPQPS

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

Example 1 Consider a clustering-coefficient query Q on a graph G, i.e., a query whose output is a scalar numerical value corresponding to the average clustering coefficient over all the vertices of G. For this query, $\mathcal{C} = \emptyset$, and $\mathbf{c} = [1]$. The execution of Algorithm 1 on a summary $\mathcal{G}$ of G for this clustering-coefficient query Q is as follows. First, summary-aware context information $\mathcal{C}_\mathcal{G}$ is computed from $\mathcal{C}$ and $\mathcal{G}$. As $\mathcal{C} = \emptyset$, $\mathcal{C}_\mathcal{G} = \emptyset$ as well. Then, summary-processed query answer $Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$ is computed. $Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$ corresponds to the average clustering coefficient over all the supernodes of $\mathcal{G}$. Finally, the ultimate output $\widetilde{Q}(G, \mathcal{G})$ of the algorithm is exactly equal to $Q(\mathcal{G}, \mathcal{C}_\mathcal{G})$, as the query at hand is numerical and $\mathbf{c} = [1]$.

Probabilistic-GPQPS algorithm The probabilistic interpretation of, among others, err_p (cf. Sect. 2) suggests to model a summary as an *uncertain* (or *probabilistic*) *graph*, that is, a graph whose edges are assigned a probability of existence:

[4] $2^V \subset 2^{2^V}$ slightly abuses notation: $\forall x \in 2^V, \{x\} \in 2^{2^V}$.

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			
		$\|\mathcal{S}\|$	350	500	750	$\|\mathcal{S}\|$	708	977	1168
	$\|V\|$: 4,039	$\|\mathcal{E}\|$	8462	14,390	24,327	$\|\mathcal{E}\|$	664	2157	4448
Facebook (Leskovec and Krevl 2014)	$\|E\|$: 88,234	#CCs	1	1	1	#CCs	240	18	38
	Undirected	| GCC |	100%	100%	100%	| GCC |	20%	77%	79%
	Unweighted	#self	264	280	323	#self	11	22	36
		Cpr.	86%	79%	68%	Cpr.	94%	92%	90%
		$\|\mathcal{S}\|$	500	750	1000	$\|\mathcal{S}\|$	3375	3568	3821
	$\|V\|$: 7429	$\|\mathcal{E}\|$	6835	10,350	12,768	$\|\mathcal{E}\|$	1217	1550	1997
LastFM (Leskovec and Krevl 2014)	$\|E\|$: 27788	#CCs	1	1	1	#CCs	2169	. 2076.	1966
	Undirected	| GCC |	100%	100%	100%	| GCC |	1%	11%	. 26%
	Unweighted	#self	133	176	185	#self	6	10	17
		Cpr.	58%	47%	40%	Cpr.	66%	64%	62%
		$\|\mathcal{S}\|$	1000	1500	2000	$\|\mathcal{S}\|$	22,832	23,957	24,371
	$\|V\|$: 33,696	$\|\mathcal{E}\|$	50,872	68,405	81,444	$\|\mathcal{E}\|$	10,160	28,825	44,086
Enron (Leskovec and Krevl 2014)	$\|E\|$: 180,811	#CCs	1	1	1	#CCs	14,537	7159	6056
	Undirected	| GCC |	100%	100%	100%	| GCC |	19%	64%	70%
	Unweighted	#self	231	303	397	#self	557	702	749
		Cpr.	68%	61%	54%	Cpr.	69%	60%	52%
		$\|\mathcal{S}\|$	500	750	1000	$\|\mathcal{S}\|$	–	–	–
	$\|V\|$: 6301	$\|\mathcal{E}\|$	2016	3383	5000	$\|\mathcal{E}\|$	–	–	–
Gnutella (Leskovec and Krevl 2014)	$\|E\|$: 20,777	#CCs	1	1	1	#CCs	–	–	–
	Directed	| GCC |	100%	100%	100%	| GCC |	–	–	–
	Unweighted	#self	5	6	12	#self	–	–	–
		Cpr.	60%	52%	45%	Cpr.	–	–	–
		$\|\mathcal{S}\|$	350	500	750	$\|\mathcal{S}\|$	–	–	–
	$\|V\|$: 3024	$\|\mathcal{E}\|$	35,346	55,276	82,589	$\|\mathcal{E}\|$	–	–	–
Ubuntu (Fu et al. 2019)	$\|E\|$: 144,094	#CCs	1	1	1	#CCs	–	–	–
	Undirected	| GCC |	100%	100%	100%	| GCC |	–	–	–
	Weighted	#self	67	86	110	#self	–	–	–
		Cpr.	74%	60%	41%	Cpr.	–	–	–
		$\|\mathcal{S}\|$	1000	10,000	25,000	$\|\mathcal{S}\|$	1,087,434	1,087,430	1,103,931
	$\|V\|$: 1,696,415	$\|\mathcal{E}\|$	39,064	461,826	983,129	$\|\mathcal{E}\|$	1,704,954	1,857,562	2,073,416
AS-Skitter (Leskovec and Krevl 2014)	$\|E\|$: 11,095,298	#CCs	4	38	73	#CCs	204,299	200,833	181,766
	Undirected	| GCC |	99.6%	99.7%	99.6%	| GCC |	70.1%	72.7%	75.9%
	Unweighted	#self	671	5306	10,746	#self	11,936	11,955	12,325
		Cpr.	86%	83%	79%	Cpr.	65%	64%	62%

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* (Gionis et al. 2007) (for queries returning a vertex partition).

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 $\mathcal{C}$; clustering-aggregation algorithm AGG (if $\mathcal{O}_Q=\mathbf{B}_V$)
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$(G,\mathcal{W}_i,Q,\mathcal{C})$, for all $i=1,\ldots,K$
3: **if** $\mathcal{O}_Q=\mathbb{R}^d$ **then**
4: $\quad\widetilde{Q}(G,\mathcal{G}) \leftarrow \frac{1}{K}\sum_{i=1}^{K}\widetilde{Q}(G,\mathcal{W}_i)$
5: **else if** $\mathcal{O}_Q=2^V$ **then**
6: $\quad\widetilde{Q}(G,\mathcal{G}) \leftarrow \bigcap_{i=1}^{K}\widetilde{Q}(G,\mathcal{W}_i)$
7: **else if** $\mathcal{O}_Q=\mathbf{B}_V$ **then**
8: $\quad\widetilde{Q}(G,\mathcal{G}) \leftarrow$ run AGG on $\{\widetilde{Q}(G,\mathcal{W}_i)\}_{i=1}^{K}$ Gionis et al. (2007) and Gullo et al. (2009)
9: **end if**

Example 2 Consider again the clustering-coefficient query Q of Example 1. The execution of Algorithm 2 on a summary $\mathcal{G}$ of G for this clustering-coefficient query Q is as follows. First, possible worlds $\mathcal{W}_1,\ldots,\mathcal{W}_K$ are sampled from $\mathcal{G}_\pi=(\mathcal{S},\mathcal{E},\pi)$. Then, query answer $\widetilde{Q}(G,\mathcal{W}_i)$ is computed on every world $\mathcal{W}_i$, by running Algorithm 1. Finally, the ultimate output $\widetilde{Q}(G,\mathcal{G})$ of the algorithm is equal to the average $\frac{1}{K}\sum_{i=1}^{K}\widetilde{Q}(G,\mathcal{W}_i)$ over all the various $\widetilde{Q}(G,\mathcal{W}_i)$'s, as the query at hand is numerical.

5 Experimental methodology

Datasets We experiment with public datasets, most of which are traditionally used in the graph-summarization literature (Kang et al. 2022b; Ko et al. 2020; Lee et al. 2020; Riondato et al. 2017; Shin et al. 2019), and whose characteristics are reported in 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 approach (see Sect. 2), namely size-driven S2L (Riondato et al. 2017) and utility-driven SWeG (Shin et al. 2019). 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 report results in correspondence of various different sizes of the resulting summary graphs.

We used a SWeG 's unofficial implementation (Hajiabadi et al. 2021). As for S2L, although it is capable of handling weighted/directed graphs, its official implementation (Riondato et al. 2017) 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 all the classes discussed in Sect. 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 $\in 2^V$).

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 (Blondel et al. 2008). Centrality queries include *PageRank*, and *closeness*. As for core decomposition (Batagelj and Zaversnik 2011), 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 apart from core decomposition, we used the corresponding implementations in NetworkX (Hagberg et al. 2008). For core decomposition, we used a custom implementation of the classic Batagelj and Zaversnik's algorithm (Batagelj and Zaversnik 2011) (as NetworkX's implementation does not support weighted graphs).

GPQPS methods We evaluate Naïve-GPQPS and Probabilistic-GPQPS (Algorithms 1 and 2). On S2L summaries, we test two variants of Naïve-GPQPS : "N " and "N w," which either consider (N w) or discard (N) superedge weights, and three variants of Probabilistic-GPQPS : "P," "P a," and "P e," depending on the weight considered for every superedge (S,T) in the sampled worlds: no weight (P), average weight $\overline{\omega}(S,T)$ (P a), and expected weight $pr(S,T)\cdot\overline{\omega}(S,T)$ as defined in Eq. (1) (P e). 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 variants on SWeG 's

summaries coupled with other-party superedge weightings (e.g., the one by S2L). 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: $K = 100$; π 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 (Topchy et al. 2003).

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 answer obtained via GPQPS with respect to the answer in the original graph. For community detection, we consider the *relative error* of the *modularity* (Newman and Girvan 2004) of the communities yielded by GPQPS with respect to the modularity of the communities computed in the original graph. For both queries, the *relative error* is defined as $|(q_G - q_{\mathcal{G}})/q_G|$ where q_G is the query answer in the graph G and $q_{\mathcal{G}}$ is the query answer in $\mathcal{G}$, the summary of G.

Regarding centrality, 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) $\in \{1, \ldots, |V|\}$, we take the centrality score x_g (resp. x_s) of the gth (resp. sth) 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 the *precision* $P = |g\text{-set} \cap s\text{-set}|/|s\text{-set}|$ and the *recall* $R = |g\text{-set} \cap s\text{-set}|/|g\text{-set}|$ of the s-set with respect to 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, because of its large size. Thus, on that dataset, we compute closeness on a subset of one thousand randomly sampled vertices.

As for core decomposition, we assess it similarly to centrality. 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 we measure the precision $P = |C_z \cap C^*|/|C_z|$ and the recall $R = |C_z \cap C^*|/|C^*|$, for all $z \in \{1, 2, 5\}$.

Environment We run all 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.

6 Experimental results

Storage space Table 1 shows the compression percentages of the various summaries. Those percentages attest how 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, 3, 4. and 5.

Because of lack of space, we report results with varying the summary size for clustering coefficient and community detection queries, and for the other queries we show results for one summary size only. Results with other summary sizes are in the **supplementary material** (Anagnostopoulos et al. 2024).

General remarks As expected, the accuracy and runtime overall increase as the the summary size increases. Regarding accuracy, a few exceptions may arise with the Probabilistic-GPQPS variants as well as 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 larger, worse-connected summary may lead to less effective query processing than a smaller, better-connected summary.

The Naïve-GPQPS variants are evidently faster than actual query processing on the input graph. The speedup depends on dataset and summary sizes, and expensiveness of the query. However, it is always tangible (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, P a, and P e refer to a sequential execution of them).

Clustering coefficient (Tables 2 and 3). 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

Table 2 GPQPS results for clustering coefficient and community detection queries with S2L (Riondato et al. 2017) 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
	N w	.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
	P a	.034	.025	.01	5.92	18.6	42.6	.058	.045	.052	.97	2.02	3.81
	P e	.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
	N w	.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
	P a	1.02	.857	.755	1.53	2.51	3.36	.37	.317	.332	1.01	1.56	1.93
	P e	.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
	N w	.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
	P a	.279	.309	.312	38	59.1	73.7	.318	.322	.317	6.61	10.5	20.7
	P e	.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
	N w	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
	P a	.017	.19	.3	.085	.142	.268	1.02	1.02	1	1.5	2.26	2.32
	P e	.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	1316	1208	1075	.999	2.28	3.5	.838	.733	.656	.26	.4	.86
	N w	1.18	.526	.123	11.4	19.5	28.1	.130	.093	.055	.29	.49	.76
	P	1231	1150	1058	14.9	31.2	52.6	.788	.783	.66	4.76	7.69	12.9
	P a	1.88	1.07	.509	145	244	371	.202	.117	.078	5.31	8.37	13.4
	P e	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
	N w	.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
	P a	.184	.129	.117	3.95	133	593	.734	.609	.509	47.9	91.8	216
	P e	.657	.451	.402	1.3	109	433	.666	.465	.416	96.6	104	234

"A" means actual query processing on the original graph

Table 3 GPQPS results for clustering coefficient and community detection with SWeG (Shin et al. 2019) summaries

Dataset	Method	Clustering coefficient						Community detection (modularity)					
		Relative error			Runtime (s)			Relative error			Runtime (s)		
Facebook	*#supernodes*	708	977	1.2k	708	977	1.2k	708	977	1.2k	708	977	1.2k
	#superedges	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	*#supernodes*	3.4k	3.6k	3.8k	3.4k	3.6k	3.8k	3.4k	3.6k	3.8k	3.4k	3.6k	3.8k
	#superedges	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	*#supernodes*	23k	24k	24.4k	23k	24k	24.4k	23k	24k	24.4k	23k	24k	24.4k
	#superedges	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	*#supernodes*	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M	1.1M
	#superedges	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

"A " means actual query processing on original graph

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 each dataset) there is no clear winner, as each method achieves better performance in 3 out of 6 datasets. No winner is between N and N w either: this meets literature evidence that superedge weights are useful or not, depending on the dataset (Kang et al. 2022a). A similar finding arises comparing the Probabilistic-GPQPS variants to each other. As for SWeG summaries, GPQPS the performance is high on summaries that are not too sparse (e.g., Facebook, AS-Skitter), but it drops on very sparse summaries (e.g., LastFM).

Community detection (Tables 2 and 3). 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 (Tables 4 and 5)). As expected, increasing the number s of the summary-retrieved vertices, 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 variants. On the negative side is Probabilistic-GPQPS 's results on Gnutella, explained by the nature of that dataset (directed), as mentioned previously. Again, the performance of SWeG on sparser summaries is much worse than the performance on denser summaries.

Closeness centrality (Tables 4 and 5). Results here are mostly in accordance with 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. Indeed, the closeness score is less robust than PageRank to the disappearing of paths between vertices when switching from the original graph to the summary, as we discuss in Sect. 7. An important remark here is that the efficiency results on AS-Skitter are not very reliable, because for this large dataset we compute closeness of only one thousand randomly sampled vertices. Therefore, the

Table 4 GPQPS results for centrality and core decomposition queries with S2L (Riondato et al. 2017) 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
	N w	.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
	P a	40	.39	.28	.24	.41	.16	.65	437	.32	.28	.16	.31	.08	.4	9.6	1	.04	1	.06	1	.15
	P e	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
	N w	.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
	P a	20.8	.79	.61	.51	.79	.21	.97	337	.8	.6	.51	.88	.2	.95	5.3	1	.04	1	.09	1	.15
	P e	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: 1500, #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
	N w	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
	P a	126	.6	.53	.37	.68	.18	.84	5.8k	.89	.78	.54	.98	.21	.99	43	.64	.66	.58	.69	.49	.75
	P e	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: 3383*																				
	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
	N w	.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
	P a	8.6	0	0	1	.08	.85	.39	77	.96	.24	.84	.41	.76	.6	1.3	.82	0	.86	0	.97	.02
	P e	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: 55276*																				
	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
	N w	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
	P a	125	.97	.95	.51	1	.2	1	1.9k	.6	.39	.35	.52	.17	.81	38	1	0	1	.01	.97	.01
	P e	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
	N w	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
	P a	444	.38	.27	.3	.46	.16	.69	4.5k	.46	.41	.38	.71	.1	.96	139	.81	.61	.81	.61	.81	.62
	P e	391	.37	.31	.27	.46	.16	.72	3.7k	.7	.26	.64	.65	.11	.92	126	1	0	1	0	1	0

"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 1000 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)

Table 5 GPQPS results for centrality and core decomposition queries with SWeG (Shin et al. 2019) 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: 2157*																				
	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: 3568, #superedges: 3821*																				
	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

"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 1000 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)

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, we still observe a consistent speedup of GPQPS methods (at least by Naïve-GPQPS).

Core decomposition (Tables 4 and 5). Trends and observations here are similar to the previous queries. As a key difference, for $z = 1$ or 2, the precision is mostly higher than the recall. This suggests that taking up to the two top-innermost cores of the summary does match the original innermost core, but it does not cover it entirely.

7 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 can claim that the methods we design in this work are rather powerful baselines for the emerging GPQPS problem.

Obstacles for a more 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. All these aspects are limitations for the basic GPQPS methodologies devised in this work. Among these, the first aspect (i.e., sparse/not well-connected summaries) is perhaps the most critical one, as it limits the capability of the proposed basic GPQPS methods to provide correct answers to queries (e.g., shortest-path queries) which rely on the fact that the underlying graph is connected.

Consistent gain in storage space is achieved by any tested 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 effec-

tiveness 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 lies in the query-answer aggregation strategy over the various worlds, especially for vertex-set or partitioning queries (see Sect. 8 for ideas on how to address this in future work).

No clear winner among weighted and unweighted variants of the various GPQPS methods. This confirms a literature finding (Kang et al. 2022a) that superedge weights in a summary may be beneficial or not.

8 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-management workloads on graphs. Our major goal in studying this problem is to set its stage, and stimulate and drive further research on it, by devising initial, basic methodologies. 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, such 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. More concretely, instead of the one investigated in this work, which simply takes the union of the supernodes, one can also exploit the information of superedges 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 such as core decomposition, that only depend on intra-set edges. Conversely, this could be less effective for queries such as 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. (2016) 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 worlds would be no more dependent on the form of query answer.

GPQPS on sparse summaries As mentioned in Sect. 7 one of the main obstacles for a really effective GPQPS is a summary that is overly sparse or not so well-connected. In fact, these types of summary limit the capability of the proposed basic GPQPS methods to provide correct answers to queries (e.g., shortest-path queries) which rely on the fact that the underlying graph is connected. In this regard, therefore, opportunities for future work include proper corrections to the GPQPS methodologies, or, even better, proper ways to increase the connectedness of the summaries before having them processed by GPQPS methods.

Miscellanea Other interesting directions include deriving approximation guarantees for GPQPS methods, testing GPQPS on more queries, and devising query-processing-effective methods to produce 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 is at answering queries. For instance, graph-summarization methods can focus better on reducing the sparsity of the resulting summaries, an aspect that affects GPQPS significantly, as observed in our experiments.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s13278-024-01314-w.

Acknowledgements Francesco Gullo and Lorenzo Severini were supported for this research by Project ECS 0000024 Rome Technopole—CUP B83C22002820006, "PNRR Missione 4 Componente 2 Investimento 1.5", funded by European Commission—NextGenerationEU. Aris Anagnostopoulos was supported by the ERC Advanced Grant 788893 AMDROMA, the EC H2020RIA project "SoBigData++" (871042), the PNRR MUR project PE0000013-FAIR, the PNRR MUR project IR0000013-SoBigData.it, and the MUR PRIN project

2022EKNE5K "Learning in Markets and Society". Giorgia Salvatori's work for this research was carried out while she was an intern at UniCredit.

Author contributions All the authors contributed equally to all the phases of the work.

Data availability No datasets were generated or analysed during the current study.

Reproducibility Data and code are available at https://github.com/fgullo/GPQPS

Declarations

Conflict of interest The authors declare no competing interests.

References

Abiteboul S, Kanellakis P, Grahne G (1987) On the representation and querying of sets of possible worlds. In: Proceedings of ACM international conference on management of data (SIGMOD), pp 34–48

Aggarwal CC, Wang H (2010) Managing and mining graph data, advances in database systems, vol 40. Springer, Berlin

Aggarwal CC, Wang H (2010) A survey of clustering algorithms for graph data. In: Aggarwal CC, Wang H (eds) Managing and mining graph data, advances in database systems, vol 40. Springer, Berlin, pp 275–301

Ahn KJ, Guha S, McGregor A (2012) Graph sketches: sparsification, spanners, and subgraphs. In: Proceedings of symposium on principles of database systems (PODS), pp 5–14

Anagnostopoulos A, Arrigoni V, Gullo F et al (2024) General-purpose query processing on summary graphs—supplementary material (https://github.com/fgullo/GPQPS)

Batagelj V, Zaversnik M (2011) Fast algorithms for determining (generalized) core groups in social networks. Adv Data Anal Classif (ADAC) 5(2):129–145

Beg MA, Ahmad M, Zaman A et al (2018) Scalable approximation algorithm for graph summarization. In: Proceedings of Pacific-Asia conference on advances on knowledge discovery and data mining (PAKDD), pp 502–514

Besta M, Hoefler T (2018) Survey and taxonomy of lossless graph compression and space-efficient graph representations. CoRR arXiv:abs/1806.01799

Besta M, Weber S, Gianinazzi L et al (2019) Slim Graph: practical lossy graph compression for approximate graph processing, storage, and analytics. In: Proceeedings of international conference for high performance computing, networking, storage and analysis (SC), pp 35:1–35:25

Biafore C, Nawab F (2016) Graph summarization for geo-correlated trends detection in social networks. In: Proceedings of ACM international conference on management of data (SIGMOD), pp 2247–2248

Blondel VD, Guillaume JL, Lambiotte R et al (2008) Fast unfolding of communities in large networks. J Stat Mech: Theory Exp 10:P10008

Boldi P, Vigna S (2004) The WebGraph framework I: compression techniques. In: Proceedings of world wide web conference (WWW), pp 595–602

Boldi P, Santini M, Vigna S (2009) Permuting web and social graphs. Internet Math 6(3):257–283

Coscia M, Neffke FMH (2017) Network backboning with noisy data. In: Proceedings of IEEE International conference on data engineering (ICDE), pp 425–436

Dalvi N, Suciu D (2004) Efficient query evaluation on probabilistic databases. In: Proceedings of international conference on very large data bases (VLDB), pp 864–875

Fan W, Li J, Wang X et al (2012) Query preserving graph compression. In: Proceedings of ACM international conference on Management of Data (SIGMOD), pp 157–168

Fan W, Li Y, Liu M et al (2021) Making graphs compact by lossless contraction. In: Proceedings of ACM international confernce on management of data (SIGMOD), pp 472–484

Fan W, Li Y, Liu M et al (2022) A hierarchical contraction scheme for querying big graphs. In: Proceedings of ACM international confernce on management of data (SIGMOD), pp 1726–1740

Fazzone A, Lanciano T, Denni R et al (2022) Discovering polarization niches via dense subgraphs with attractors and repulsers. Proc VLDB Endowm (PVLDB) 15(13):3883–3896

Fu X, Yu S, Benson AR (2019) Modelling and analysis of tagging networks in stack exchange communities. J Complex Netw 8(5)

Fung WS, Hariharan R, Harvey NJA et al (2019) A general framework for graph sparsification. SIAM J Comput (SICOMP) 48(4):1196–1223

Galimberti E, Ciaperoni M, Barrat A et al (2021) Span-core decomposition for temporal networks: Algorithms and applications. ACM Trans Knowl Discov Data (TKDD) 15(1):2:1–2:44

Gionis A, Mannila H, Tsaparas P (2007) Clustering aggregation. ACM Trans Knowl Discov Data (TKDD) 1(1):4

Gou X, Zou L, Zhao C et al (2019) Fast and accurate graph stream summarization. In: Proceedings of IEEE international conference on data engineering (ICDE), pp 1118–1129

Gullo F, Tagarelli A, Greco S (2009) Diversity-based weighting schemes for clustering ensembles. In: Proceedings of SIAM international conference on data mining (SDM), pp 437–448

Hagberg AA, Schult DA, Swart PJ (2008) Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th python in science conference, pp 11–15

Hajiabadi M, Singh J, Srinivasan V et al (2021) Graph summarization with controlled utility loss. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 536–546

Hernández C, Navarro G (2014) Compressed representations for web and social graphs. Knowl Inf Syst (KAIS) 40(2):279–313

Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of ACM symposium on theory of computing (STOC), pp 604–613

Jiang Z, Chen H, Jin H (2023) Auxo: a scalable and efficient graph stream summarization structure. In: Proceedings of the VLDB endowment (PVLDB) 16(6)

Jin D, Yu Z, Jiao P et al (2023) A survey of community detection approaches: from statistical modeling to deep learning. IEEE Trans Knowl Data Eng (TKDE) 35(2):1149–1170

Kang S, Lee K, Shin K (2022a) Are edge weights in summary graphs useful? A comparative study. In: Proceedings of Pacific-Asia conference on advances on knowledge discovery and data mining (PAKDD), pp 54–67

Kang S, Lee K, Shin K (2022b) Personalized graph summarization: formulation, scalable algorithms, and applications. In: Proceedings of IEEE international conference on Data Engineering (ICDE), pp 2319–2332

Ke X, Khan A, Bonchi F (2022) Multi-relation graph summarization. ACM Trans Knowl Discov Data (TKDD) 16(5):82:1–82:30

Khan A, Ye Y, Chen L (2018) On uncertain graphs. Synthesis lectures on data management, Morgan & Claypool Publishers, San Rafael

Khan K, Nawaz W, Lee Y (2015) Set-based approximate approach for lossless graph summarization. Computing 97(12):1185–1207

Ko J, Kook Y, Shin K (2020) Incremental lossless graph summarization. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 317–327
Koutra D, Kang U, Vreeken J et al (2014) VoG: summarizing and understanding large graphs. In: Proceedings of SIAM international conference on data mining (SDM), pp 91–99
Kumar KA, Efstathopoulos P (2018) Utility-driven graph summarization. Proc VLDB Endowm (PVLDB) 12(4):335–347
Lanciano T, Savino A, Porcu F et al (2023) Contrast subgraphs allow comparing homogeneous and heterogeneous networks derived from omics data. GigaScience 12
Lee K, Jo H, Ko J et al (2020) SSumM: sparse summarization of massive graphs. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 144–154
Lee K, Ko J, Shin K (2022) SLUGGER: lossless hierarchical summarization of massive graphs. In: Proceedings of IEEE international conference on data engineering (ICDE), pp 472–484
LeFevre K, Terzi E (2010) GraSS: Graph structure summarization. In: Proceedings of SIAM international conference on data mining (SDM), pp 454–465
Leskovec J, Krevl A (2014) SNAP datasets: stanford large network dataset collection. http://snap.stanford.edu/data
Liu X, Tian Y, He Q et al (2014) Distributed graph summarization. In: Proceedings of ACM international conference on information and knowledge management (CIKM), pp 799–808
Liu Y, Safavi T, Dighe A et al (2018) Graph summarization methods and applications: a survey. ACM Comput Surv (CSUR) 51(3):62:1–62:34
Lloyd SP (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–136
Maserrat H, Pei J (2010) Neighbor query friendly compression of social networks. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 533–542
Mehmood Y, Bonchi F, García-Soriano D (2016) Spheres of influence for more effective viral marketing. In: Proceedings of ACM international conference on management of data (SIGMOD), pp 711–726
Mosa MA, Hamouda A, Marei M (2017) Graph coloring and ACO based summarization for social networks. Expert Syst Appl 74:115–126
Navlakha S, Rastogi R, Shrivastava N (2008) Graph summarization with bounded error. In: Proceedings of ACM international conference on management of data (SIGMOD), pp 419–432
Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113
Riondato M, García-Soriano D, Bonchi F (2014) Graph summarization with quality guarantees. In: Proc. IEEE international conference on data mining (ICDM), pp 947–952
Riondato M, García-Soriano D, Bonchi F (2017) Graph summarization with quality guarantees. Data Min Knowl Discov (DAMI) 31(2):314–349
Sadri A, Salim FD, Ren Y et al (2017) Shrink: distance preserving graph compression. Inf Syst 69:180–193
Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64
Serrano MÁ, Boguñá M, Vespignani A (2009) Extracting the multiscale backbone of complex weighted networks. Proc Natl Acad Sci 106(16):6483–6488
Shin K, Ghoting A, Kim M et al (2019) SWeG: Lossless and lossy summarization of web-scale graphs. In: Proceedings of world wide web conference (WWW), pp 1679–1690
Slater PB (2009) A two-stage algorithm for extracting the multiscale backbone of complex weighted networks. Proc Natl Acad Sci 106(26):E66–E66
Spielman DA, Teng S (2011) Spectral sparsification of graphs. SIAM J Comput (SICOMP) 40(4):981–1025
Toivonen H, Zhou F, Hartikainen A et al (2011) Compression of weighted graphs. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 965–973
Topchy AP, Jain AK, Punch WF (2003) Combining multiple weak clusterings. In: Proc. IEEE international conference on data mining (ICDM), pp 331–338
Tsalouchidou I, Bonchi F, Morales GDF et al (2020) Scalable dynamic graph summarization. IEEE Trans Knowl Data Eng (TKDE) (TKDE) 32(2):360–373
ur Rehman S, Nawaz A, Ali T et al (2021) g-Sum: agraph summarization approach for a single large social network. EAI Endorsed Trans Scalable Inf Syst 8(32):e2
Ward JH (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244
Yong Q, Hajiabadi M, Srinivasan V et al (2021) Efficient graph summarization using weighted LSH at billion-scale. In: Proceedings of ACM international conference on management of data (SIGMOD), pp 2357–2365
Zeng Y, Song C, Ge T (2021) Selective edge shedding in large graphs under resource constraints. In: Proceedings of IEEE international conference on data engineering (ICDE), pp 2057–2062
Zhou F, Mahler S, Toivonen H (2010) Network simplification with minimal loss of connectivity. In: Proc. IEEE international conference on data mining (ICDM), pp 659–668
Zhou F, Qu Q, Toivonen H (2017) Summarisation of weighted networks. J Exp Theor Artif Intell 29(5):1023–1052
Zhou H, Liu S, Lee K et al (2021) DPGS: degree-preserving graph summarization. In: Proceedings of SIAM international conference on data mining (SDM), pp 280–288