Structural Summarization of Semantic Graphs Using Quotients

Ansgar Scherp
Ulm University, Germany

David Richerby
University of Essex, UK

Till Blume
Ernst and Young Research, Germany

Michael Cochez
Vrije Universiteit Amsterdam, The Netherlands
Elsevier Discovery Lab, The Netherlands

Jannik Rau
Ulm University, Germany

Abstract

Graph summarization is the process of computing a compact version of an input graph while preserving chosen features of its structure. We consider semantic graphs where the features include edge labels and label sets associated with a vertex. Graph summaries are typically much smaller than the original graph. Applications that depend on the preserved features can perform their tasks on the summary, but much faster or with less memory overhead, while producing the same outcome as if they were applied on the original graph.

In this survey, we focus on structural summaries based on quotients that organize vertices in equivalence classes of shared features. Structural summaries are particularly popular for semantic graphs and have the advantage of defining a precise graph-based output. We consider approaches and algorithms for both static and temporal graphs. A common example of quotient-based structural summaries is bisimulation, and we discuss this in detail. While there exist other surveys on graph summarization, to the best of our knowledge, we are the first to bring in a focused discussion on quotients, bisimulation, and their relation. Furthermore, structural summarization naturally connects well with formal logic due to the discrete structures considered. We complete the survey with a brief description of approaches beyond structural summaries.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; General and reference → Surveys and overviews

Keywords and phrases graph summarization, quotients, stratified bisimulation

Digital Object Identifier 10.4230/TGDK.1.1.12

Category Survey

Funding *Ansgar Scherp*: Co-funded by the CodeInspector project (No. 504226141) of the DFG, German Research Foundation.
Michael Cochez: Partially funded by the Graph-Massivizer project, funded by the Horizon Europe programme of the European Union (grant 101093202).

Received 2023-07-03 **Accepted** 2023-11-17 **Published** 2023-12-21

Editors Aidan Hogan, Ian Horrocks, Andreas Hotho, and Lalana Kagal

Special Issue Trends in Graph Data and Knowledge

Transactions on Graph Data and Knowledge, Vol. 1, Issue 1, Article No. 12, pp. 12:1–12:25
TGDK Transactions on Graph Data and Knowledge
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Representing data as a graph is increasingly popular [45, 48], though the idea dates back to at least the 1960s [5]. The strength of graphs as a data representation lies in their general applicability and their focus on relationships between data points rather than just the properties of the individual data points [45]. The same graph formalism can be used in various domains ranging from relations in social networks, drug and protein interactions, connections between terminals in a telecommunications network, pages on the World Wide Web, linked data on the Semantic Web, and many others [45].

1.1 What are Semantic Graphs and Why Graph Summarization?

We are agnostic to the specific representation of graphs. We use the umbrella term *semantic graph* for the kinds of graphs considered in this work. We assume that a graph is a collection of vertices connected by directed edges. The vertices and/or edges may be labeled and one may perform some semantic inference on them, e. g., generalizations and specializations [16].

▶ **Definition 1** (Semantic Graph). *A* semantic graph *is a finite, directed, labeled graph* $G = (V, E, \ell_V, \ell_E)$*. Here* $E \subseteq V \times V$ *is the edge relation, and the functions* $\ell_V : V \to \Sigma_V$ *and* $\ell_E : E \to \Sigma_E$ *label the vertices and edges, respectively.*

We also refer to the value $\ell_V(v)$ as the *type set* of a vertex, and we use the function ℓ_E to define the *property set* of a vertex, which is defined over its outgoing edges. Some variants of graph summarization also consider incoming edges (we discuss these in Section 3).

▶ **Definition 2** (Property Set). *The* property set *of a vertex* v *in a semantic graph* G *is the set* $\{\ell_E(v, w) \mid (v, w) \in E\}$ *of labels on its outgoing edges. We abuse notation and write* $\ell_E(v)$ *for the property set of* v*.*

A semantic graph may be represented as a Resource Description Framework (RDF) graph [24], labeled property graph (LPGs), or some other approach. RDF graphs do not directly support vertex labels, but `rdf:type` edges can be used to simulate these, so RDF graphs and LPGs can be transformed into one another [8]. A comprehensive overview of different kinds of semantic graphs is given by Hogan et al. [48] under the term "Knowledge Graphs". The term was coined in 2012 by Google as part of its knowledge representation and web search service extended by contextual knowledge such as mapping queries to persons, companies, etc.

Classical graph algorithms focus on finding structures such as shortest paths or minimum spanning trees, or invariants such as treewidth or chromatic number. In the context of graph-structured data, the focus of algorithms shifts from the graph *per se* to the data it represents. Typical tasks on such graphs include mere querying of the data, but also estimating cardinalities for queries in graph databases [77], subgraph-based indices for data search [59], data modeling recommendation [87], schema induction [99], data exploration [80], data visualization [41], and related entity retrieval [22].

The motivation for graph summaries lies in the growing size of semantic graphs. As the graphs can be extremely large, tasks become computationally expensive and might require a large amount of memory. Structural *graph summaries* have been developed as useful abstractions of large graphs to solve tasks more efficiently. A summary of a graph G is a smaller graph S, which retains the information from G that is required to perform the desired tasks, but which discards information that is not needed, and which may represent the retained information more compactly.

Example

Suppose we wish to count solutions to queries on a graph database G such as finding all students studying the same course, or all books in the same genre by the same set of authors. Many vertices in the graph G may be equivalent from the point of view of the queries. Hence, we can summarize the graph by merging each set X of equivalent vertices into a single vertex v_X that is labeled with the cardinality of X. In doing this, we lose the ability to answer certain queries that could be answered on the original database. We can no longer distinguish between vertices that have been merged, and we no longer know their identities. But we can still answer our class of counting queries, and we can do so by processing a much smaller graph – graph summaries are typically orders of magnitude smaller than the original graph, measured in numbers of edges [19].

Summary graphs can be constructed in several ways. The survey by Čebirić et al. [19] classifies existing techniques into structural, pattern-mining, statistical, and hybrid approaches. A broad overview of these summarization approaches can be found there. In this paper, we focus on structural approaches based on quotients, due to their versatility and popularity for summarizing semantic graphs. The idea here is to partition the vertices into equivalence classes, assigning each vertex to exactly one equivalence class. These equivalence classes are used as the vertices of the summary. A structural summary is again a graph that can be used to answer a given application task exactly [19], i. e., as if it were executed on the original graph. Structural graph summaries defined using quotients [20] are closely related to k-bisimulation [53,88]. Many summary approaches for semantic graphs are in fact stratified bisimulations.

Such structural summaries are "lossless" with respect to the features defined in a summary model: selected features of the original graph are accurately preserved to allow the task to be exactly computed on the summary. The features preserved by the graph summary are defined in the so-called *graph summary model*. Summaries can also be "lossy", only allowing the task to be approximated on the summary.

1.2 What is a Task, a Graph Summary, and a Graph Summary Model?

Since different works and communities deal with graph summarization from different perspectives, we first need to provide a high-level clarification of the basic concepts. This shall provide the reader with an intuition about the nature of graph summaries and how they are used. The basic definitions are of a *task* (in the context of a graph summary) as well as the *graph summary* and its *model*, based on Blume, Scherp, and Richerby [10].

▶ **Definition 3** (Task). *Given a graph G, a task T applied on G is a parameterized function $T_\Psi : G \to Y$ that maps the input graph G to a task-specific range of values Y.*

The set of parameters Ψ and the output of the function T, (the range Y) are specific to a given application domain.

Example Task

For cardinality estimations of queries on the graph G the range Y is $\mathbb{N}^+$ and the parameter Ψ is the graph-based query Q for which the cardinality is to be estimated [77].

▶ **Definition 4** (Graph Summary Model). *A graph summary model is a tuple* $M = (\mathrm{EQR}, \Phi, \mathrm{PAY})$*, where* EQR *specifies an equivalence relation on graph vertices,* Φ *are model parameters, and* PAY *is a set of task-specific payload functions. They are computed during summarization and their output is included in the summary.*

Structural approaches summarize a graph G w.r.t. an equivalence relation $\mathrm{EQR} \subseteq V \times V$ defined on the vertices V of G [14, 19]. The vertices $V(S)$ of the resulting summary graph S correspond to equivalence classes of the equivalence relation EQR, and to equivalence classes of subsidiary equivalence relations used in the definition of EQR – see Figure 1.

Example Summary Models

Example graph summary models are the attribute-based collection [18], where the EQR defines vertex equivalence based on having the same set of labels of the outgoing edges. In contrast, the class-based collection [18] is defined by an EQR that groups vertices sharing the same vertex labels. There are also summary models that require both attribute and class equivalence [22, 59].

The model parameters Φ are applied to control the output. Φ can, e.g., limit the maximum summary size (in terms of number of vertices or edges), the weights assigned to some graph elements (preference for certain edges or vertices), or the minimum support of subgraph structures summarized by S [19]. PAY describes what information must be stored about the summarized vertices to allow the task T to be answered using the summary.

Example Payload Functions

In a data search task, such as in the first example, the set PAY contains one payload function to compute cardinalities of the search results. Another commonly used payload function for data search is computing the set of data sources (URIs where the results can be found on the web) [43].

Different summary models serve different tasks. While a structural summary is a lossless representation of the input graph with respect to the features defined in a summary model, different summaries (with different payloads) will be needed for different tasks. Thus, the choice of features and the tasks have to be aligned. Finally, based on the graph summary model, we define the *Graph Summary*:

▶ **Definition 5** (Graph Summarization). *A graph summarization is a parameterized function* $C_M \colon G \to S$ *that computes a representation of the input graph based on a graph summary model* $M = (\mathrm{EQR}, \Phi, \mathrm{PAY})$*. The summary* S *preserves (selected) features of* G *in accordance with* M*. Note that* S *also contains the output of the payload functions* PAY*.*

For the full definition of a graph summary with respect to a graph summary model, see [10].

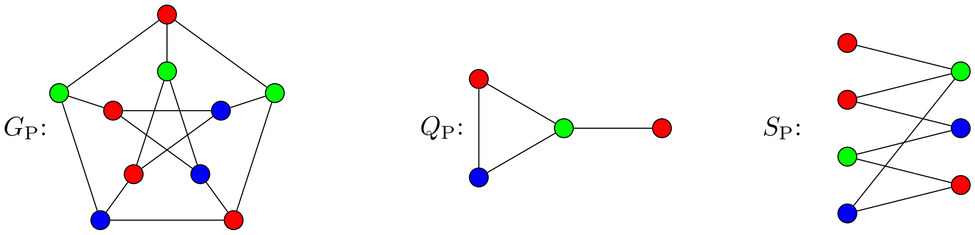

Figure 1 A colored Petersen graph G_{P} (left) quotiented (center, Q_{P}; see Section 3) and summarized (right, S_{P}). The graph summary model defines two vertices of G_{P} to be equivalent if they have the same color and are adjacent to the same set of colors.

Example Summary of the Petersen Graph

In Figure 1, the semantic graph G_{P} to be summarized is a colored version of the Petersen graph. In this case, the graph is undirected and has vertex labels (the colors) but no edge labels. We use a graph summary model that defines two vertices of G_{P} to be equivalent if they have the same color and are adjacent to the same set of colors. There are four equivalence classes: the red vertex with only green neighbors (the top vertex), the three red vertices with green and blue neighbors, the three green vertices with red and blue neighbors, and the three blue vertices with red and green neighbors.

In the summary graph S_{P}, these “primary” equivalence classes are on the left-hand side; the vertices on the right are “secondary” equivalence classes of the “same color” relation, which is used to define equivalence of the neighbors. In this example, there is no payload, but payload would be stored as labels of additional vertices connected to the primary equivalence classes.

1.3 What is *Not* a Graph Summary!

Graph summarization is distinct from other related concepts, some of which also use the term *summarization*, but to mean something different.

The goal of *compression* is to allow the original graph to be exactly recovered (in the case of lossless compression, for example, RDF HDT [38]), or approximately recovered (lossy compression). Thus, compression must retain (or approximately retain) all information in the graph. Another term in this context is “corrections” [58, 89]. Again, the goal is to (incrementally) compute a lossless compression of a graph by determining a so-called corrections set, i. e., a set of edges that must be added or removed to reconstruct the original graph [58, 89]. Another work on graph compression is by Hajiabadi et al. [44], who propose an approach to reconstruct a graph exactly (lossless) or with a small error (lossy). In contrast to compression and correction, summarization only retains the information that is needed for specific tasks.

An intermediate operation between summarization and compression is graph *contraction* [34], which uses the graph-theoretic concept of contraction minors [25]. Contraction makes graphs smaller by replacing regular structures such as cliques and paths with “supernodes”. Each supernode is annotated with a payload and a mechanism for recovering the original graph. This can also be done hierarchically, to produce even smaller graphs [35]. Compared to summarization, this only operates on parts of the graph; on the other hand, the entire graph can be recovered if needed, so the contracted graph is not task-specific (albeit that tasks that can be performed using the payload of the supernodes, without expanding out the graph, will run faster).

Further related research areas are graph transformation systems, graph rewriting, and model-driven engineering [30]. These approaches have in common that they have rules and mappings to manipulate input graphs to a desired output graph, e. g., to merge adjacent vertices by preserving (hyper-)edge structures [57, 81]. In general, the difference between structured graph summaries and the communities of graph transformation and graph rewriting is as follows: The goal of summarization is to provide concise representations of the input graph G while preserving specific, defined features (e. g., structural features like which edges are attached to a vertex). The focus of graph rewriting is to perform operations on the graph to transform G from one state to another via rewrite rules. Nonetheless, graph summarization is an example of graph rewriting in the most general sense.

1.4 Structure

The article is organized as follows: First, we consider applications of structural graph summaries in Section 2. In Section 3, we introduce features and models of structural graph summaries for static graphs that are based on quotients.

Bisimulation is a versatile and popular technique among structural graph summarization models, which we discuss in further detail in Section 4. Logics are a natural partner of structural summaries and this connection is discussed in Section 5. We consider graph summarization approaches for temporal graphs in Section 6.

Finally, in Section 7, we briefly discuss alternative approaches to graph summaries that are not based on graph quotients, such as pattern mining, and statistical approaches. We conclude this article with a brief reflection, as well as an outlook on future directions and open questions.

2 Applications of Structural Graph Summaries

Many different structural graph summaries have been developed to solve different tasks [6, 18, 22, 23, 41, 53, 59, 71, 72, 77, 87, 88, 91, 95]. In the following, we describe four common applications of structural (semantic) graph summaries in detail. These applications are typical applications for managed graph database systems but also for the decentralized Semantic Web [1].

These applications showcase the universal nature of structural summaries. We connect the applications with the basic notion of graph summarization by providing the specific definitions of the graph summary model M being used in each application. These are the equivalence relation EQR its parameters Ψ and equally important the set of payload functions PAY (see Definition 4).

2.1 Semantic Entity Retrieval

One application of structural graph summaries is to find semantically related entities in the Semantic Web [22]. Entities on the Semantic Web are represented using vertices, each identified with a unique IRI [29], and labeled edges indicate relations between them. Entities can have a set of types, each of which is indicated by an edge labeled `rdf:type` to a vertex representing that type. This graph data is typically stored as an RDF graph [1].

If two entities share the same set of RDF types and RDF properties (i.e., labels of outgoing edges other than `rdf:type`), we can say they are semantically related [22]. Hence, we can create a summary where the equivalence relation EQR represents that; i. e., it puts vertices in the same partition if their RDF types and set of RDF properties are the same.

This structural graph summary summarizes vertices (entities), based on such structural subgraph features. To find semantically related entities, we need to memorize the vertex identifiers of each summarized vertex in the computed structural graph summary, this index gets created by a function from PAY and is part of the output of the graph summary. We can now use that index for immediate retrieval of related vertices since all semantically related vertices are summarized together.

A variation of this task would be when the graph is dynamic. In that case, there is an interesting trade-off between recomputing the index, which can be costly, versus updating it in place, which requires more index information to be kept and leads to a more complex implementation.

2.2 Cardinality Computation

Cardinality computation is often desired in databases [77, 84]. For graph databases, graph summaries can be used as an index to look up how many vertices will be returned by certain queries. If the graph summary is smaller than the original graph, as it usually is, the query runs faster on the graph summary than on the database. This task is related to the semantic entity retrieval task described above. However, for this purpose, we only need to memorize the number of summarized vertices rather than all vertex identifiers. Hence, EQR would be the same, but the function in PAY would only compute the counts, rather than keeping the full index.

A graph summary that memorizes the number of summarized vertices enables fast implementations of query size estimation [77]. Analogously to the semantically related entities task above, expensive re-computation of the graph summary from scratch when the database changes can yield an unwanted performance overhead, and hence a way to deal with online updates is necessary.

Knowledge bases often have a data schema or ontology that defines how entities should be modeled. Structural graph summaries can help determine how many entities strictly follow that schema, match the schema partially, or even contradict the schema. The stored numbers of summarized vertices can be used as an indicator of data completeness [84]. It is often desired to evaluate the evolution of data quality over time [84]; also in that case an incremental update mechanism is needed.

2.3 Data Source Search

As part of *data source search*, one needs to find (sub)graphs in the Semantic Web that match a given schema structure [43]. Structural graph summaries can be used as an index that memorizes the location of summarized vertices on the Web. This is illustrated in Figure 2.

The EQR of the graph summary model in this scenario is defined such that it summarizes vertices that have the same set of RDF types, and are connected by edges with the same labels to target vertices with the same set of types, known as SchemEX [59]. The set of payload functions PAY is in this case a cardinality count and memorizing the data source URIs.

With a graph query, the structural graph summary is queried to get the URLs of relevant data sources. Then, the data sources are accessed to download the graphs matching the query. Search systems like LODatio [43], LODeX [6], Loupe [71], and LODatlas [80] rely on structural graph summaries to offer a search for relevant data sources or exploration of data sources.

To implement this task, we need to memorize the locations where each summarized vertex appears, which is computed by a function in PAY. As the data on the Web changes [51], the summaries need to be updated as well. In contrast to the previous two tasks, for data source search we neither memorize vertex identifiers nor the number of summarized vertices but only their location.

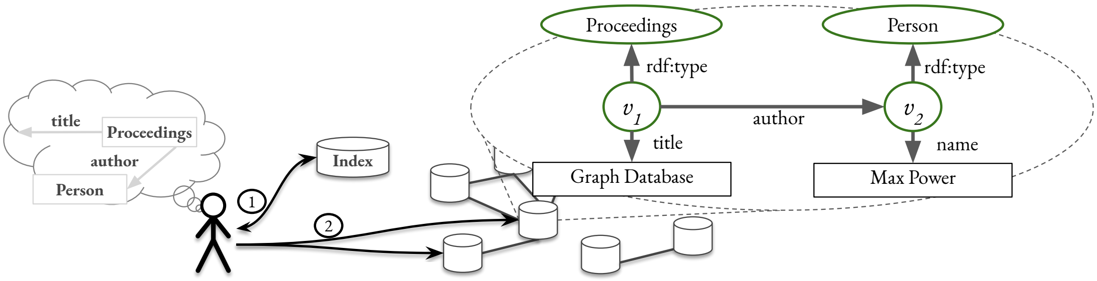

Figure 2 Finding data sources on the Web using an index based on graph summaries (from [8]). A structural query is executed over an index to identify relevant data sources (1). Subsequently, the data sources are accessed to retrieve actual vertices (2).

2.4 Training Graph Neural Networks

Another application of graph summaries can be found in recent work by Generale, Blume, and Cochez [39], who address scalability issues when training graph neural networks (GNNs) on larger graphs. They suggest training the GNN on a summary, rather than on the original graph. The summaries used in this case are actually quotient graphs (see Section 3), using either k-bisimulation (see Section 4) or direct vertex attributes for EQR.

The machine learning task solved with this GNN is the prediction of vertex types. In the normal training setting, the training set is a subset of the vertices of known type. The model parameters are then optimized such that the outputs of the GNN, when passed through a classifier, predict the correct vertex type for the test vertices. To evaluate whether the model works, one uses it to predict the types of vertices outside the training set and compares these predictions to the correct answer. An issue while training on the summary is that multiple vertices from the original graph are in the same equivalence class and hence mapped to the same vertex in the summary. Now, the difficulty is that we do not have a clear label for this vertex; it is not the case that all nodes in the equivalence class have the same label, since this is not taken into account when summarizing. The authors [39] suggest using a weighted multi-label classification task during training where the labels are weighted by their frequency. To compute this label, we need to collect the frequency of the labels of the vertices of each equivalence class, which is one of the functions in PAY.

After training on the summary graph, the weights of the model are transferred back to the original graph, where inference is performed to predict the types of vertices from the test set. Inference can be done on the full graph because it is much faster than training and requires much less memory. In some cases, this method of training can not only provide reasonable results, but also the results can be improved if the model is further trained on the original graph. This two-stage process can give better results than training only on the original graph, without summarizing. To transfer the model weights, we include the index with the equivalence classes as part of PAY.

A follow-up work by Bollen et al. [13] provides a theoretical foundation for the earlier work. They prove that it is possible to create a specific graph summary that for a specific class of GNN performs the same message-passing steps as on the original graph. In effect, this means that the same outcome is obtained at inference time. The prior work is not covered by this proof because the summary did not retain information about the cardinality of edges in the quotient graph, and therefore the conditions for equivalence are not met. Error bounds for the approximation remain open.

3 Structural Summarization of Static Graphs based on Quotients

In Section 1, we introduced structural graph summaries as a condensed representation of a graph based on some summary model. Our focus is on quotient-based summaries. For context, we first define graph quotients.

▶ **Definition 6** (Graph Quotient). *Consider a labeled graph G, and an equivalence relation $\equiv$ on G's vertices. The* quotient *of G w.r.t. $\equiv$ is the graph Q defined as follows. The vertices are the equivalence classes of $\equiv$. Q has an edge labeled p from class c to class c' iff G contains an edge labeled p from some vertex in c to some vertex in c'.*

Example

Recall Figure 1. There, vertices of the colored Petersen graph G_P are defined to be equivalent if they have the same color and their neighbors have the same set of colors. This leads to four equivalence classes among G_P's vertices: the blue vertices (which are all adjacent to both red and green), the green vertices (which are all adjacent to red and blue), and what we will call type-1 (adjacent to green and blue) and type-2 red vertices (adjacent only to green). These are the four vertices of the quotient Q_P, with the type-1 red vertex on the left and the type-2 on the right. The edges can be read off from the descriptions of the equivalence classes, noting that each green vertex is adjacent to a blue vertex, a type-1 red and the type-2 red.

Quotient-based summaries are constructed along similar lines, but the summary stores relationships between multiple equivalence relations, instead of just between the classes of a single one. For example, the equivalence relation described above for the colored Petersen graph G_P defines the overall equivalence on vertices with reference to a second equivalence relation: that which only considers the vertex's color and not its neighbors. Blume, Richerby, and Scherp [10] define a language, FLUID, for specifying such combinations of equivalence relations. We omit the details here, as the following intuition suffices.

The input semantic graph is summarized using a *primary equivalence relation*, which is defined in FLUID by a logic-like expression that combines *secondary equivalence relations*. The secondary equivalence relations may themselves be defined by combining other secondary equivalence relations.

▶ **Definition 7** (Summary Graph). *The summary S of a graph G w.r.t. a summary model M has a vertex for each equivalence class of each equivalence relation used in the definition of* EQR *in M.* Primary *and* secondary vertices *correspond to primary and secondary equivalence relations, respectively. Between these vertices, edges are added similarly to quotient graphs. Suppose equivalence relation R is defined by combining equivalence relations, one of which is R'. Then there is an edge labeled p from class c of R to class c' of R' in the summary graph if and only if there is an edge labeled p in the input graph from some vertex in c to some vertex in c'.*

This can be seen in Figure 1. The primary vertices are on the left of S_P (equivalence classes of "same color and same colored neighbors") and the secondary vertices ("same color") are on the right. All edges are between a primary vertex and a secondary vertex.

In this approach to structural graph summarization, each vertex v of the input graph G is represented by exactly one primary vertex p_v in the summary graph S. The structure around v in G that is used to determine v's equivalence class is represented by the edges between p_v and secondary vertices, and edges between the secondary vertices. Structure around v that is not used to determine v's equivalence class is not represented in the summary.

Bisimulation is a very common feature among structural graph summarization approaches, but often the works on structural summarization using quotients and bisimulation do not explicitly refer to each other. We discuss bisimulation in detail in Section 4, where we explain its relation to summary models introduced in this section.

There are many structural graph summary models based on quotients, defining the features that shall be captured by the summary, often targeted for solving one specific task [6, 22, 43, 71, 77, 80, 87, 91, 92]. Based on Blume et al. [10], we analyze existing structural graph summaries with respect to the captured schema structure, i. e., what features of the input graph are used to summarize vertices. This analysis complements existing surveys covering graph summaries [14, 19, 55, 66] and the taxonomy is summarized in Figure 3. We distinguish features that only use triple information (triple features), features that define how features of multiple vertices are combined (subgraph features), and features that define explicit semantic rules such as joining and inference (semantic rule features). Each group of features adds another level of complexity, i. e., intuitively, the computational complexity of computing summaries grows when features of different groups are used. There is no single graph summary model that supports all features. However, we see common combinations of features. In the following, we summarize the graph summary models along with the identified features.

Other features in quotient-based summaries include dependent compression [50], which summarizes vertices v_1 and v_2 if and only if v_1 is adjacent only to v_2, or vice versa [50]. In another variant of dependent compression, a set of vertices is grouped if they are connected to the same set of other vertices in G [65].

3.1 Triple Features

Triple features are solely based on outgoing triples of vertices. A triple corresponds to a directed edge between two vertices, namely the subject connecting to the object, which is labeled with the predicate.

Edge Labels

To compute the equivalence of two vertices s and s' of G, we compare the triples where the subject is s with those where the subject is s'. The most commonly used feature in structural graph summaries is using properties to compute the schema of vertices. More specifically, for each vertex s in the data graph the set of edge labels $\ell_E(s)$ is compared. For example, the graph summary model attribute-based collections, due to Campinas et al. [18], solely relies on the sets of edge labels to compute the graph summary. If vertices s and s' share the same property set, i. e., $\ell_E(s) = \ell_E(s')$, they are considered equivalent, so are summarized together.

Vertex Labels

Another common feature is using the vertices' labels to compute the summary. Here, for each vertex s in the data graph, the so-called type set $\ell_V(s)$ is compared. If vertices s and s' share the same type set, i. e., $\ell_V(s) = \ell_V(s')$, they are considered equivalent. For example, class-based collections (again due to Campinas et al. [18]) is a summary model, which uses only the vertices' label sets to compute the schema. These are used, along with the attribute-based collections described above, to implement a query recommendation system for SPARQL,[1] that facilitates working with heterogeneous datasets, especially when the schema structure is unknown.

[1] https://www.w3.org/TR/sparql11-overview/

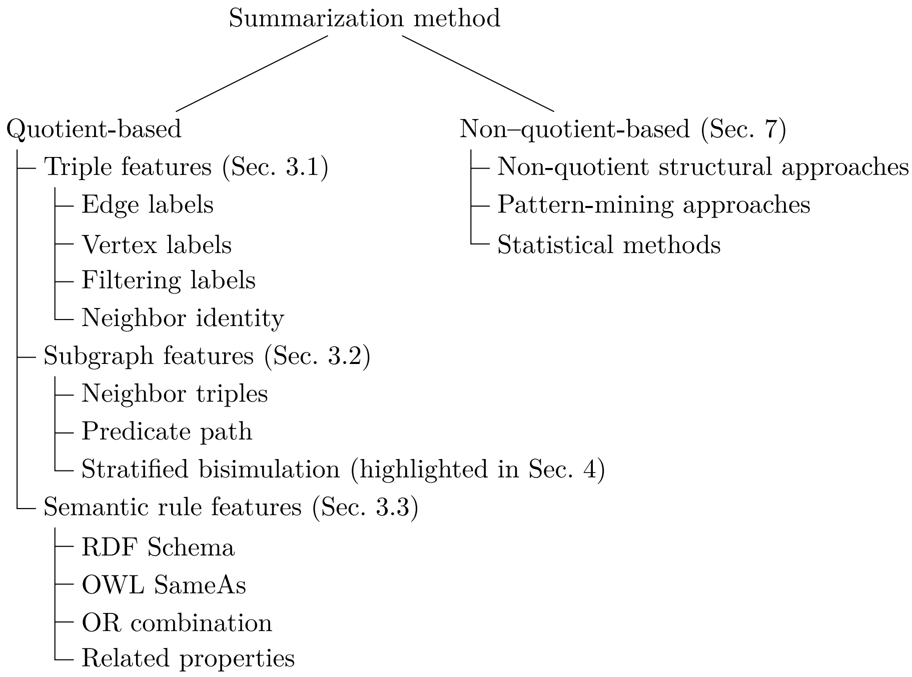
Summarization method
Quotient-based
Triple features (Sec. 3.1)
Edge labels
Vertex labels
Filtering labels
Neighbor identity
Subgraph features (Sec. 3.2)
Neighbor triples
Predicate path
Stratified bisimulation (highlighted in Sec. 4)
Semantic rule features (Sec. 3.3)
RDF Schema
OWL SameAs
OR combination
Related properties
Non–quotient-based (Sec. 7)
Non-quotient structural approaches
Pattern-mining approaches
Statistical methods

Figure 3 The taxonomy of features that are used in structural graph summarization, based on Čebirić et al. [19]. These features are described in detail in the sections indicated, and can be summarized in both static and temporal graphs. Section 3 introduces the different features of quotient-based methods. Section 4 discusses stratified bisimulation methods in detail, as they are most prominent among quotient-based methods. Section 5 further reflects on quotient-based summaries and their relation to logics, which are by their nature orthogonal to the taxonomy. Quotient-based summaries for temporal graphs are discussed in Section 6. The overview of summarization methods is complemented with a brief discussion of non–quotient-based summarization methods in Section 7.

Filtering Labels

Tran et al. [95] proposed the feature of label parameterization for graph summaries. With the label parameterization, only a subset of all edge labels is used to compute the schema. More precisely, one defines a set of predicates P_l, the so-called label set, which are ignored when determining the equivalence of vertices. Tran et al.'s graph summary combines property sets $\ell_E(s)$ with label sets. Furthermore, they combine this with k-bisimulation (see Section 4 on k-bisimulation).

Neighbor Identity

The final triple feature uses the identity of outgoing neighbors $\Gamma^+(s) = \{v \mid (s,v) \in E\}$ to determine the equivalence of vertices. It appears that no existing graph summary summarizes vertices solely by comparing the neighbor identities. However, SemSets [22] summarize vertices that share the same outgoing predicates, which are linked to the same vertices. To check if vertices s and s' are equivalent under SemSets, all triples where s or s' are the subject vertices are compared. For each triple $(s,p,o) \in G$ there has to be a triple $(s',p,o) \in G$, and vice versa. Thus, they combine neighbor vertex identifiers $\Gamma^+(s)$ with predicate paths (see below).

3.2 Subgraph Features

The neighbor vertex identifier is the most direct approach to incorporate neighbor information and leads to a wider range of summary models that consider neighbor information, e. g., vertices in $\Gamma^+(s)$. We classify features as subgraph features when they combine triple features of multiple vertices.

Neighbor Triples

SchemEX [59], SchemEX+U+I [11], ABSTAT [91], LODeX [6], and Loupe [71] summarize vertices s and s' based on having a common type set and common edge labels linking to vertices with the same type sets. This means that, to compute the schema of one vertex s, also the type sets of outgoing neighbors $\Gamma^+(s)$ are required to be equivalent, i. e., we compare neighbor triples. In contrast to SemSets [22], these approaches use not the neighbor vertex identifiers $\Gamma^+(s)$ but the type set $\ell_V(o)$ for each $o \in \Gamma^+(s)$. SchemEX [59], SchemEX+U+I [11], ABSTAT [91], LODeX [6], and Loupe [71] combine type sets $\ell_V(s)$, property sets $\ell_E(s)$, and neighbor type sets $\ell_V(o)$ for $o \in \Gamma^+(s)$ using predicate paths, introduced next. This mapping can aid in recommending related queries and generally for finding relevant data sources [43].

Predicate Path

Almost all analyzed graph summaries that use neighbor information combine the schema structures using predicate paths, i. e., they compare which predicates link to which neighbors. Predicate paths are compared based on the edge labels and type sets that appear along paths. For example, SchemEX [59], SchemEX+U+I [11], ABSTAT [91], LODeX [6], and Loupe [71] consider which property links to which type set. TermPicker [87] follows a different strategy to integrate the schema of neighboring vertices. TermPicker summarizes vertices s based on having the same type set $\ell_V(s)$, the same property set $\ell_E(s)$, and the same set of types among the neighbors, $\{\ell_V(o) \mid o \in \Gamma^+(s)\}$. Consequently, TermPicker's graph summaries compress all type sets of all neighbors into a single type set. Thus, TermPicker's graph summaries do not contain information about which *specific* property linked to which neighbor.

Stratified Bisimulation

Many graph summaries compute the schema of vertices by taking into account the schema of neighbors over multiple hops [53, 59, 83, 95]. This is commonly defined as a bisimulation. Bisimulation operates on state transition systems and defines an equivalence relation over states [86]. Two states are equivalent (or bisimilar) if they change into equivalent states with the same type of transition. Inductively, this means that applying an arbitrary sequence of transitions to two bisimilar states will result in bisimilar states. Interpreting a labeled graph as a representation of a state transition system allows us to apply bisimulation on graph data to discover structurally equivalent parts.

In practice, many graph summary models define a **stratified k-bisimulation**, e. g., [53,59,83, 95]. When states are k-bisimilar, applying any sequence of k transitions to them will result in equivalent states, but applying more than k transitions may lead to inequivalent states. Thus, stratified bisimulation only considers paths of lengths up to k when determining equivalence. This increases the chance that two vertices are considered equivalent.

Some graph summaries combine the feature of using only incoming or only outgoing properties with the k-bisimulation feature [23, 72, 88]. This is referred to as **backward k-bisimulation** and **forward k-bisimulation**, respectively [41]. The T-index of Milo and Suciu [72] supports path queries in semi-structured databases. This summarizes vertices s based on having the same set of incoming property-paths, i. e., they use k-bisimulation only on incoming property sets $\ell^-_E(s)$. Consens et al. [23] propose a structural graph summary model to support navigational SPARQL queries, so-called Extended Property Paths (EPPs). They summarize vertices s based on having the same set of outgoing property-paths, i. e., they use k-bisimulation only on outgoing property sets $\ell_E(s)$. In addition, for each hop, the type sets $\ell_V(s)$ have to be equivalent.

Stratified k-bisimulation is very popular in the summary models found in the literature. Often, however, it is not referred to as such and/or the summary model is considering only the case of the 1-bisimulation. Due to its importance for structural graph summarization, we discuss and reflect on the k-bisimulation feature in detail in Section 4.

3.3 Semantic Rule Features in Graph Summaries

The last group of features for structural graph summaries defines explicit semantic rules. It deals with RDF Schema (RDFS) reasoning, OWL's `owl:sameAs` reasoning, as well as inference on property sets in an OR-like fashion and via the inclusion of related properties.

RDF Schema

Several semantic structural graph summaries use RDF Schema inference to enhance their summaries. ABSTAT [91] exploits RDF Schema type hierarchies to compute so-called minimal patterns. They select the minimal number of types, i. e., they only keep the most specific types from the RDF Schema type hierarchy. Goasdoué et al. [41] exploit RDF Schema type hierarchies, property hierarchies, and RDF Schema domain and RDF Schema range. With domain and range, types for the subject vertex and the object vertex can be inferred.

OWL's SameAs

SchemEX+U+I [11] also uses the full RDF Schema inference but also exploits the semantics of the `owl:sameAs` property. This property is part of W3C's OWL [70], which is heavily used in the context of RDF graphs. The `owl:sameAs` property defines an equivalence relation [70, Section 4.2], intended to identify vertices that represent the same real-world entity. To compute the schema structure of one vertex v, the schema structures of all vertices v' in the weakly connected components in an `owl:sameAs`-labeled subgraph of G are merged (see Ding et al. [26] for details on `owl:sameAs` networks).

OR Combination

Goasdoué et al. [41] define the Weak Summary using an OR-like combination. In the Weak Summary, two vertices s and s' are equivalent if they have the same outgoing property set and/or the same incoming property set. This is not necessarily transitive, so the transitive closure is taken as the equivalence relation. The authors also define a Typed Weak Summary, which combines the Weak Summary based on properties with vertex types. There is also a variant of a Strong Summary, which does not consider the OR-like combination. Details can be found in Goasdoué et al. [41], see also the discussion in Blume et al. [10].

Related Properties

Goasdoué et al. [41] also propose to include property relations. Two properties p and p' are source-related if they co-occur in any property set $\ell_E(s)$ of any vertex s and they are target-related if they co-occur in any incoming property set $\ell^-_E(s)$ of any vertex s (i. e., the set $\{\ell_E(v, s) \mid (v, s) \in E\}$).

4 Structural Graph Summarization by k-Bisimulation

We introduced bisimulation in Section 3.2 as a means of defining equivalence of vertices in a graph. Bisimulations are specific kinds of equivalence relations that classify vertices v and w as equivalent if, for each edge (v, v') with label p, there is an edge (w, w'), also with label p, where w' is, recursively, equivalent to v'. *Complete bisimulation* extends this recursively to all distances from v and w, whereas k-bisimulation only requires equivalence out to distance k. Forward bisimulation considers outgoing edges to determine equivalence, as described above; backward bisimulation is analogous but uses incoming edges; backward-forward bisimulation uses both. In addition to edge labels, bisimulation may also be based on vertex labels or both, but this makes no principal difference.

In complete bisimulation, for the vertices v and w to be bisimilar, their in-/out-neighbors must be bisimilar as well. This recursive definition is, essentially, an equivalence relation defined in terms of itself. Since there is only one equivalence relation, this naturally lends itself to a quotient representation, as exemplified in Section 2.4. However, stratified bisimulations are more commonly used for graph summarization, and lead naturally to quotient-based graph summaries, as we discuss below.

4.1 Stratified Bisimulation to Paths of Length k

As real-world graphs are quite heterogeneous, there may be only a few bisimilar vertices [19] if we consider full bisimulation. Thus, k-stratified bisimulation is often used, restricting the paths to length k. This increases the possibility that two vertices are bisimilar and, overall, reduces the size of the summary.

A k-bisimulation on a graph G considers features a distance at most k from a vertex that is to be decided equivalent to another vertex [85]. Formally, this can be defined for the forward bisimulation on the outgoing edges as follows (based on [85]). Backward bisimulation is defined similarly.

▶ **Definition 8** (Stratified Forward Bisimulation based on Edge Labels). *The* forward k-bisimulation *$\approx^k_{\text{fw}} \subseteq V \times V$ with $k \in \mathbb{N}$ is defined as follows:*

- *$u \approx^0_{\text{fw}} v$ for all $u, v \in V$,*
- *$u \approx^{k+1}_{\text{fw}} v$ iff $u \approx^k_{\text{fw}} v$ and, for every edge (u, u'), there exists an edge (v, v') with the same label such that $u' \approx^k_{\text{fw}} v'$, and vice-versa.*

For bisimulation with vertex labels, we modify the base case of the definition so that $u \approx^0_{\text{fw}} v$ iff u and v have the same labels. Note that stratified bisimulation defines a hierarchy of equivalence relations $\approx^0_{\text{fw}}, \approx^1_{\text{fw}}, \ldots \approx^k_{\text{fw}}$, in contrast to complete bisimulation, which defines a single equivalence relation, recursively in terms of itself. Thus, stratified bisimulation is best suited to summarization, rather than quotienting. When producing a summary for k-bisimulation, the relation $\approx^k_{\text{fw}}$ will be the primary equivalence relation, and $\approx^0_{\text{fw}}, \ldots, \approx^{k-1}_{\text{fw}}$ are secondary.

Bisimulation stratified to paths of length k is a popular technique to compute structural graph summaries, though often $k = 1$ is used. We give examples in Table 1 and discuss them in detail in this section. Note that TermPicker [87] is a relaxed version of bisimulation. Conventional bisimulation requires the same edge label to the same type of neighbor, whereas TermPicker just requires the same edge labels and the same neighbor types, without the correlation.

4.2 Examples of Stratified Bisimulation for Graph Summarization

Efficient algorithms for bisimulation have been developed by Paige and Tarjan [78], Kaushik et al. [53], Dovier et al. [27], Schätzle et al. [88], and others.

Table 1 Example graph summary models based on bisimulation.

Summary model	Depth	Labels used	Direction	Application
Class collection [18]	0	vertex	forward	query recommendation
Attribute collection [18]	1	edge	forward	query recommendation
LODex [6], Loupe [71]	1	vertex and edge	forward	data exploration
SchemEX [43, 59]	1	vertex and edge	forward	data search
TermPicker [87]	1	vertex and edge*	forward	modeling recommendation
Tran et al. [95]	k	edge	backward-forward	entity search
$A(k)$-index [53]	k	vertex	backward	general purpose
T-index [72]	k	vertex	backward	data indexing
Schätzle et al. [88]	k	edge	forward	general-purpose

(* relaxed version of bisimulation; see text)

A notion of k-bisimulation w.r.t. graph indices is introduced by seminal works such as the k-RO index [76] and the T-index summaries [72]. Milo and Suciu's T-index [72], the $A(k)$-Index by Kaushik et al. [53], and others summarize graphs using backward k-bisimulation. Qun et al. [83] extend the $A(k)$-Index to a $D(k)$-Index, which is also based on bisimulation but focuses on query optimization. To this end, the $D(k)$-Index dynamically adapts its structure according to the current query load. Another work using stratified bisimulation is by Fan et al. [33], for reachability and graph pattern matching on large graphs.

Conversely, the k-RO index, the Extended Property Paths of Consens et al. [23], the SemSets model of Ciglan et al. [22], Buneman et al.'s RDF graph alignment [17], and the work of Schätzle et al. [88] are based on forward k-bisimulation. Buneman et al. use forward k-bisimulation to summarize the union of two consecutive versions $G_{\text{union}} = G_1 \cup G_2$ of an RDF graph with respect to k-bisimulation, which puts vertices to be aligned in the same partition. As well as to k-bisimulation, they use a similarity measure to further refine the initial k-bisimulation partition, as it does not capture all vertices to be aligned. The focus of their work is the optimization of the alignment process so that every node pair (v_1, v_2), with $v_1 \in G_1$ and $v_2 \in G_2$, which have to be aligned is identified and not the construction of a k-bisimulation-based partition of G. Schätzle et al. compute a forward k-bisimulation on RDF graphs in sequential and distributed settings [88]. For a small synthetic dataset ($\sim 1M$ RDF-triples) the sequential algorithm slightly outperforms the distributed one; for larger datasets, the distributed algorithm clearly outperforms the sequential one.

Tran et al. compute a structural index for graphs based on backward-forward k-bisimulation [95]. Moreover, they parameterize their notion of bisimulation to a forward-set L_1 and a backward-set L_2, so that only labels $l \in L_1$ are considered for forward bisimulation and labels $l \in L_2$ for backward bisimulation. However, similar to Buneman et al., the particular focus of their work is not the construction of the bisimulation partition. Rather, they evaluate how one can efficiently optimize query processing on semi-structured data using an index graph based on bisimulation.

There are also structural summarization approaches that determine vertex equivalence only based on local information ($k \leq 1$). As shown in Table 1, many of the summary models introduced in Section 3.2 are actually very shallow bisimulations.

4.3 Distributed and Parallel Bisimulation

Luo et al. [67, 68] examine structural graph summarization by forward k-bisimulation in a distributed, external-memory model. They empirically observe that, for values of $k > 5$, the summary graph's partition blocks change little or not at all. Therefore they state that, for summarizing a graph with respect to k-bisimulation, it is sufficient to summarize up to a value of $k = 5$.

Martens et al. [69] introduce a parallel bisimulation algorithm for massively parallel devices such as GPU clusters. Their approach is tested on a single GPU with 24 GB RAM, which limits its use on large datasets. Nonetheless, their proposed blocking mechanism could be combined with our vertex-centric approach to further improve performance.

5 Structural Graph Summarization and Logics

Logics are a natural framework for defining properties of graphs, queries on graphs, and transformations between graphs. More specifically, logics can be used to define equivalence relations on the set of vertices V of a graph, partitioning V into disjoint sets [10]. Thus, logics provide a natural framework for defining structural graph summaries. The partitions and the relationships between them are interpreted as summaries of the vertices, either in the sense of a quotient or a structural graph summary. The question is what logics should be used to define graph summaries.

Since Fagin's discovery [31] that existential second-order logic defines exactly the properties of graphs in the complexity class **NP**, the field of descriptive complexity [49] has sought to understand the relationship between the features of a logical language (e.g., the logical operators and quantifiers it contains) and the computational complexity of the graph properties it can define. A paradigmatic tension in descriptive complexity theory is the trade-off between a logic's expressive power and the computational cost of evaluating formulas [49, 64]. Thus, we seek logics that are expressive enough to define interesting summaries but not so expressive that summaries cannot be computed in a reasonable time. To allow summarization of large graphs, it is essential that formulas can be evaluated in polynomial time. For web-scale graphs, we need even more efficient evaluation, which can be obtained by syntactically restricting the formulas that can be written, e.g., by restricting the number of variables in formulas or using guarded fragments of the logics. For example, evaluating a formula like $\exists y\, P(y)$ requires searching the whole graph for a vertex satisfying the predicate P, whereas, for a given vertex x, the guarded formula $\exists y\, (E(x, y) \wedge P(y))$ only requires us to search among x's neighbors. We note that existing graph summaries are typically expressible using guarded formulas, such as the neighborhood feature in FLUID [10] or constraints on requiring certain vertex labels to appear with specific edge labels [99].

First-order logic (FO) is powerful but can also be evaluated efficiently. However, from analyzing the existing graph summary approaches, we see that extensions to FO are needed to express more complex summaries. For example, extensions with counting quantifiers [49, 64] can count vertices and edges, and define vertex equivalences based on the number of neighbors of particular kinds, rather than just the existence of such neighbors. This is required for pattern mining methods with a min-supp threshold or summarizing vertices based on the number of edges having the same label. Counting quantifiers and their expressive power have been extensively studied in descriptive complexity [49, 64] and similar ideas have also been introduced in graph pattern matching [36]. One can also express iterative constructs such as loops – as required for computing bisimulations – by extending FO with fixed-point operators [49] or other recursive mechanisms such as those in Datalog [21].

6 Summarization of Temporal Graphs

Existing structural graph summarization algorithms are often designed and/or evaluated using static graphs only [6, 22, 66, 77, 87, 91]. Few quotient-based structural graph summaries are designed for evolving graphs [40, 59].

There are two ways in which one could consider summarizing temporal graphs. First, we may have a summary model that is not aware of time, and desire an algorithm that can update the summary as the data graph changes, considering the changes as a sequence of versions of the graph. Second, the summary model itself may depend on temporal features of the graph – that is, whether two vertices are in the same equivalence class may depend on their history, rather than just on their current state. The second approach might use temporal logics to define the summary model [2, 62].

The literature predominantly uses the first approach. Thus, we discuss algorithms that efficiently recompute quotient-based summaries as a graph changes over time. Subsequently, we present related approaches to incremental graph indexing and schema discovery. Indices are often based directly on quotients, often as bisimulations. Schema discovery classifies vertices according to their properties, naturally defining equivalence classes that can be quotiented.

6.1 Incremental Graph Summarization

Konrath et al. [59] compute their graph summary over a stream of vertex-edge-vertex triples. They can deal with the addition of new vertices and edges to the graph but not the deletion of vertices or edges, or modification of their labels. Similarly, Goasdoué et al. [40] present the summarization tool RDFQuotient, which only supports iterative additions of vertices and edges to their structural graph summaries, and does not handle deletions. Thus, these approaches are not suited to updating structural summaries of evolving graphs. Goasdoué et al. also do not support payload information, which is needed for tasks such as cardinality estimations and data search. The purpose of their summaries is to visualize them to a human viewer. Finally, Blume, Richerby, and Scherp [9] propose an incremental algorithm to update graph summaries that also takes deletions and payload into account. Experiments on benchmark datasets show that using the incremental algorithm is beneficial even if up to half of the graph has changed from one version to the next.

6.2 Incremental Subgraph Indices

Often, graph databases use path indices, tree indices, and subgraph indices [46]. A seminal approach to computing subgraph indices is DataGuide [42], implemented in the Lore database management system (DBMS) [102]. A DataGuide is a graph index built incrementally while executing queries on an XML database. It indicates to the query engine if and how a specific path defined in the query can be reached. To this end, a DataGuide represents all possible paths between two vertices in an XML file. While DataGuides operate on semi-structured data in terms of XML trees, a guide is essentially quotienting the graph.

Tran et al. [95, 96] took up this idea and applied it to quotienting RDF graphs. Representative Objects (ROs) by Nestorov et al. [76] take up the ideas of DataGuides and are also implemented in the Lore DBMS with focus on path queries, query optimization, and schema discovery. While the Full ROs capture a description of the global structure of the graph, the authors also introduce a notion of a k-RO. which only considers paths of length up to k. These are examples of bisimulation in graph summarization.

We now consider incremental subgraph indices based on frequent pattern mining. These techniques group graph patterns, similarly to structural summarization. For example, Yuan et al. [104] (see also extensions in [52, 105]) propose an index based on mining frequent and discriminative features in subgraphs. Their algorithm minimizes index lookups for a given query and regroups subgraphs based on newly added features.

A work directly based on quotients is Qiao et al. [82] who compute an index of isomorphic subgraphs in an unlabeled, undirected graph G. The goal is to find the set of subgraphs in G that are isomorphic to a given query pattern. The result is a compression of the original graph that can be used to answer, e. g., cardinality queries regarding subgraphs. This is for static graphs, but the algorithm of Fan et al. [32] can deal with graph changes for the subgraph isomorphism problem. Their incremental computation of an index for isomorphic subgraphs is closely related to structural graph summarization but, unlike in summarization, the graph pattern p is an input to the algorithm, not the output.

Min et al. [73] propose an algorithm for continuous subgraph matching using a summary-like data structure that stores the intermediate results between a query graph and a dynamic data graph. They consider undirected graphs where only vertices are labeled. Dynamic graphs are updated through a sequence of edge insertions and edge deletions. The TipTap [74] algorithm computes approximations of the frequent subgraphs on up to k vertices w.r.t. a given threshold. This is similar to a quotient, but vertices may appear in multiple subgraphs. It does this to count occurrences of subgraphs in large, evolving graphs, modeled as a stream of updates on an existing graph. Tesseract [7] is a distributed framework for executing general graph mining algorithms on evolving graphs. It uses a vertex-centric approach to distribute updates to different workers. It assumes that most changes affect only local graphs, so few duplicate updates need to be detected. Tesseract supports the quotient-like ideas of k-clique enumeration, graph keyword search, motif counting, and frequent subgraph mining.

Another area of incremental subgraph indices considers an evolving set of queries over a static data graph. Duong et al. [28] propose a streaming algorithm using approximate pattern matching to determine subgraph isomorphisms. They use k-bisimulation to determine equivalent subgraphs and store them in an index. However, this index is computed offline for a static graph only and their algorithm considers a stream of graph queries as input.

6.3 Incremental Schema Discovery

Another area related to quotient-based summarization is incremental schema discovery. Vertices with the same schema are naturally equivalence classes that can be quotiented. Völker and Niepert mine logical patterns in the Web Ontology Language from static RDF graphs [99]. Wang et al. [100] incrementally discover attribute-based schemata from JSON documents. The schema is computed incrementally as more documents are processed. Baazizi et al. [4] also compute schemata from JSON objects, focusing on optional and mandatory attributes.

In addition to document-oriented formats like JSON, schema discovery is also used for graph data. For example, XStruct [47] follows a heuristic approach to incrementally extract the XML schema of XML documents. However, such schema discovery approaches cannot deal with modifications or deletions of nodes in the XML tree. Other schema discovery approaches focus on generating (probabilistic) dataset descriptions. Kellou-Menouer and Kedad [54] apply density-based hierarchical clustering on vertex and edge labels in a graph database. This computes profiles that can be used to visualize the schema of the graph. Recently, Bouhamoum et al. [15] used density-based clustering to extract schema information from an RDF graph and incrementally update the schema when new RDF instances arrive. While the work can deal with additions, the deletion of edges and vertices is not considered.

7 Non-Quotient Graph Summaries

The methods for structural graph summarization discussed so far focus on analyzing the graph based on pre-defined structural features such as paths and subgraphs encountered in the graph [10, 19]. These structural graph summaries are based on quotient graphs [19] (see Section 3).

There are also structural summary models that are not formed from quotient graphs, which we discuss below. Subsequently, we consider approaches for summarization based on pattern mining and statistical approaches. The organization of the methods in these categories is taken from Čebirić et al. [20].

7.1 Non-Quotient Structural Graph Summaries

Non-quotient summaries do not use equivalence relations to summarize a graph. Rather, the summary graph is composed of vertex summaries vs, which group together vertices v of the original graph G according to certain criteria [19].

The main difference from quotient summaries discussed above is that, in the non-quotient summaries, a vertex v can belong to zero, one, or multiple vertex summaries vs. In contrast, in quotient summaries every vertex v has exactly one corresponding vertex summary vs, which is the equivalence class of v under $\sim$ [19].

Early work on non-quotient summarization includes that of Goldman and Widom [42], who created a vertex summary vs for every labeled path in the original graph G. A vertex v of G is associated with a vertex summary vs if it is reachable by the corresponding label path. The summary graph is used as a path index, as well as a tool for understanding the schema structure in semi-structured databases, and hence finds application in query formulation and query optimization. Revisiting the summarization tool SchemEX [59], its first layer – the *RDF class layer* – consists of vertex summaries vs_{c_j} representing all the classes c_j present in the input RDF graph G. A vertex v of G is associated with a vertex summary vs_{c_j}, iff v is of the corresponding type c_j. Since a vertex v can have multiple types $c_{j_1}, c_{j_2}, \ldots$, it is possible that v is associated with several vertex summaries $vs_{c_{j_1}}, vs_{c_{j_2}}, \ldots$ and therefore the index's RDF class layer is considered a non-quotient summary. Kellou-Menouer and Kedad [54] perform schema extraction by using density-based clustering to establish a partition of the vertices based on type profiles. For each type T_j, a *type profile* $TP_j = \{(label_1, \alpha_1), (label_2, \alpha_2), \ldots\}$ is constructed, consisting of tuples of edge labels for outgoing edges (v, w) and incoming edges (w, v), with $v \in T_j$. The associated probabilities α_i denote how likely it is that a vertex $v \in T_j$ has an edge with the respective $label_i$. If a type profile TP_j contains all entries $(label_i, \alpha_i)$ of another type profile TP_k and every α_i is greater than a certain threshold θ (e.g., $\theta = 0.6$), then the vertices in T_k are added to T_j to create *overlapping classes*. Clustering can be found in more structural non-quotient approaches [56, 63, 75, 98].

Other structural graph summarization methods that do not use quotients are based on structural measures such as centrality. They identify the most important vertices, cliques, and others, and connect them in the summary [19]. The difference from quotient-based summaries is that some graph vertices may not be represented in the summary, i.e., the summaries are approximate. Examples of summary methods using structural features are [79,97,107]. These guide the summaries using vertex centrality measures [12] such as vertex (in/out) degree, betweenness (how often a vertex lies on the shortest path between two other vertices), (k, h)-cores, and the well-known information retrieval measures PageRank and HITS (based on eigenvalue analysis over vertices), as well as further measures of vertex centrality such as those applied by Pappas et al. [79].

7.2 Pattern-Mining Approaches for Graph Summarization

Pattern-mining approaches identify frequent patterns in the input graph G, which are then used to construct the summary graph SG [19]. Song et al. [90] construct *d-summaries* to summarize a knowledge graph G. A summary P, which is a graph pattern found in G, is considered a d-summary, iff all the summary vertices $u \in P$ are d-similar (R_d) to all their respective original vertices $v \in V$. Informally, uR_dv iff (1) u and v share the same label and (2) for every neighbor $u' \in P$ of u connected over an edge with label p there exists a respective neighbor $v' \in V$ connected via the same edge label and $u'R_{d-1}v'$. Their definition of d-similarity is very similar to k-bisimulation (Section 4) and mainly differs in the domain on which it is defined, namely summary vertices and original vertices.

Pattern mining methods for graph summarization discover frequently occurring patterns in the data [19]. Various algorithms exist for graph pattern mining, based either on the well-known Apriori principle (e. g., [61]) or pattern-growth algorithms (e. g., [103]). These define a minimum support (min-supp) threshold over subgraphs $X \subseteq G$. Only patterns that are frequent enough are included in the summary. Pattern mining methods are approximate summaries of the input graph G, as they do not include subgraphs that occur infrequently (thus, the summarization function is no longer homomorphic). However, setting min-supp $= 1$ usually produces a lossless summary, equivalent to a structural summary computation, as no subgraphs are omitted. Pattern mining methods also have interesting features such as automatically mining specific types of subgraphs like cliques, (bipartite) cores, stars, and chains [37, 60]. While star-shaped subgraphs and chains are in principle also supported by quotient summaries, the difference here is that the selection of edges in star patterns and the length of the chains is determined in a data-driven way, rather than being pre-defined in a summary model. Finally, some pattern mining methods are also approximate because they use approximate methods such as locality-sensitive hashing (LSH) to assign graph vertices to the summary [65]. This is an inaccuracy introduced by the LSH function but not a characteristic of the underlying frequent pattern mining algorithms.

7.3 Statistical Methods for Graph Summarization

Statistical methods for graph summarization summarize the contents of a graph quantitatively such as by counting occurrences of edge labels or computing histograms over the labels [19] and define further constraints on the summary models. For example, in k-SNAP [94], the number k of summary vertices in a summary graph can be specified by the user, which controls the size of the summaries. The summarization operation k-SNAP [93] minimizes a function based on occurrences of user-selected edge labels to produce a summary graph SG, which contains exactly k vertex summaries. In its top-down approach, it starts by partitioning the graph based on user-selected vertex attributes. Afterward, the algorithm splits elements (vertex summaries) of the partition based on the aforementioned function, until the partition's size is k. Combining the first step, partitioning vertices by label, and the second step, minimizing a function that considers edge labels, k-SNAP can be considered a hybrid approach, combining structural and statistical concepts. CANAL is an extension of k-SNAP that supports numeric edge attributes [106]. Summarizing edges labeled with numeric values is approached by bucketing the values into predefined categories. Thus, the problem of supporting unbounded numerical values is reduced to summarizing graphs with discrete categories only.

8 Conclusion and Outlook

8.1 Conclusion

We delved into the domain of graph summarization, a process aimed at generating concise representations of input graphs while preserving specific structural attributes. Particularly, we focus on structural graph summaries that can be applied to semantic graphs, i. e., labeled graphs such as in the RDF or provided as labeled property graphs.

Different approaches and algorithms have been developed to address graph summarization. Our focus has been on exploring the state-of-the-art methods for structural graph summarization based on quotients. We have examined the relationship of structural summarization with other pertinent fields, like the well-known k-bisimulation. A noteworthy observation is the natural connection between structural summarization and logics, owing to the discrete structures under consideration.

Finally, there are also hybrid approaches for graph summarization. They combine features of quotient-based structural and non-quotient statistical and pattern mining techniques [19]. One such example is the combination of bisimulation with clustering [3]. Hierarchical complete-link clustering is applied over vertex features to group the vertices. Similar, Wang et al. cluster paths in a graph to compute an approximate summary [101].

8.2 Outlook

This research contributes to a deeper understanding of graph summarization techniques and opens avenues for future advancements in this domain. As directions of future research, we see:

1. Multi summaries: Existing works summarize a graph with respect to a single, defined summary model only. Multi-summaries compute multiple condensed representations of the input graph at once, stored in a joined data structure.
2. Summaries on temporal graphs: while there are already graph summarization approaches for temporal graphs, including incremental summarization [9], there is still a lot of work to be done. For example, we are currently missing approaches for summarization of rapidly evolving graphs such as social media graphs.
3. Task-specific learned summaries: Graph summarization addresses different application needs. Example applications are outlined in Section 2. Interesting future work would be to automatically learn which features of a summary model are most relevant to a task (in general) or to the workload of a task (e. g., the kinds of queries executed in the data search scenario).
4. Exploiting modern hardware such as GPUs: We have already briefly reflected on parallel and distributed computation of graph summaries in Section 4.3. First steps have been taken on using GPUs but there is still a lack of research in this direction.
5. `PyGraphSum`: Each graph summarization model and algorithm typically comes with its own implementation, datasets, and evaluation measures. Comparing different algorithms and methods is difficult and cumbersome, as a common standard library for efficient distributed summarization of static and temporal graphs is missing. A standardized library that is used among industry and researchers alike will contribute not only to more transparency and comparability of the different approaches but also accelerate research in the field by facilitating reuse.

References

1 Dean Allemang and James A. Hendler. *Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL, Second Edition.* Morgan Kaufmann, 2011. URL: http://www.elsevierdirect.com/product.jsp?isbn=9780123859655.

2 James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.

3 Anas Alzogbi and Georg Lausen. Similar structures inside RDF-graphs. In *Workshop on Linked Data on the Web*. CEUR-WS.org, 2013. URL: http://ceur-ws.org/Vol-996/papers/ldow2013-paper-05.pdf.

4 M. A. Baazizi, H. Ben Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani. Schema inference for massive JSON datasets. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT*, pages 222–233. OpenProceedings.org, 2017. doi:10.5441/002/EDBT.2017.21.

5 Charles W. Bachman. Data structure diagrams. *Data Base*, 1(2):4–10, 1969. doi:10.1145/1017466.1017467.

6 Fabio Benedetti, Sonia Bergamaschi, and Laura Po. Exposing the underlying schema of LOD sources. In *Web Intelligence (WI)*, pages 301–304. IEEE, 2015. doi:10.1109/WI-IAT.2015.99.

7 Laurent Bindschaedler, Jasmina Malicevic, Baptiste Lepers, Ashvin Goel, and Willy Zwaenepoel. Tesseract: distributed, general graph pattern mining on evolving graphs. In *European Conf. on Comp. Systems (EuroSys)*, pages 458–473. ACM, 2021. doi:10.1145/3447786.3456253.

8 Till Blume. *Semantic structural graph summaries for evolving and distributed graphs.* PhD thesis, University of Ulm, Germany, 2022. URL: https://nbn-resolving.org/urn:nbn:de:bsz:289-oparu-46050-1.

9 Till Blume, David Richerby, and Ansgar Scherp. Incremental and parallel computation of structural graph summaries for evolving graphs. In

Int. Conf. on Inform. and Knowledge Management (CIKM), pages 75–84. ACM, 2020. doi:10.1145/3340531.3411878.

10 Till Blume, David Richerby, and Ansgar Scherp. FLUID: A common model for semantic structural graph summaries based on equivalence relations. *Theoretical Computer Science*, 854:136–158, 2021. doi:10.1016/J.TCS.2020.12.019.

11 Till Blume and Ansgar Scherp. Indexing data on the web: A comparison of schema-level indices for data search. In *Database and Expert Systems Applications (DEXA)*, pages 277–286, 2020. doi:10.1007/978-3-030-59051-2_18.

12 Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Math.*, 10(3-4):222–262, 2014. doi:10.1080/15427951.2013.865686.

13 Jeroen Bollen, Jasper Steegmans, Jan Van den Bussche, and Stijn Vansummeren. Learning graph neural networks using exact compression. In *Proceedings of the 6th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 8:1–8:9. ACM, 2023. doi:10.1145/3594778.3594878.

14 A. Bonifati, S. Dumbrava, and H. Kondylakis. Graph summarization. *CoRR*, abs/2004.14794, 2020. arXiv:2004.14794.

15 Redouane Bouhamoum, Zoubida Kedad, and Stéphane Lopes. Incremental schema discovery at scale for RDF data. In *The Semantic Web - 18th International Conference, ESWC*, volume 12731 of *Lecture Notes in Computer Science*, pages 195–211. Springer, 2021. doi:10.1007/978-3-030-77385-4_12.

16 D. Brickley and R.V. Guha. RDF Schema 1.1, 2014. URL: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

17 Peter Buneman and Slawek Staworko. RDF graph alignment with bisimulation. *Proc. VLDB Endow.*, 9(12):1149–1160, 2016. doi:10.14778/2994509.2994531.

18 Stéphane Campinas, Thomas Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing RDF graph summary with application to assisted SPARQL formulation. In *Database and Expert Systems Applications (DEXA)*, pages 261–266. IEEE, 2012. doi:10.1109/DEXA.2012.38.

19 Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. Summarizing semantic graphs: a survey. *VLDB J.*, 28(3):295–327, 2019. doi:10.1007/S00778-018-0528-3.

20 Šejla Čebirić, François Goasdoué, and Ioana Manolescu. A framework for efficient representative summarization of RDF graphs. In *Int. Semantic Web Conf. (ISWC)*. CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-1963/paper512.pdf.

21 S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (but never dared to ask). *Trans. Knowl. Data Eng.*, 1(1):146–166, 1989. doi:10.1109/69.43410.

22 Marek Ciglan, Kjetil Nørvåg, and Ladislav Hluchý. The SemSets model for ad-hoc semantic list search. In *World Wide Web Conf. (WWW)*, pages 131–140, 2012. doi:10.1145/2187836.2187855.

23 Mariano P. Consens, Valeria Fionda, Shahan Khatchadourian, and Giuseppe Pirrò. S+EPPs: Construct and explore bisimulation summaries, plus optimize navigational queries; all on existing SPARQL systems. *PVLDB*, 8(12):2028–2031, 2015. doi:10.14778/2824032.2824128.

24 Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225.

25 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 5th edition, 2016. doi:10.1007/978-3-662-53622-3.

26 Li Ding, Joshua Shinavier, Zhenning Shangguan, and Deborah L. McGuinness. SameAs networks and beyond: Analyzing deployment status and implications of owl:sameAs in Linked Data. In *Int. Semantic Web Conf. (ISWC)*, pages 145–160. Springer, 2010. doi:10.1007/978-3-642-17746-0_10.

27 Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004. doi:10.1016/S0304-3975(03)00361-X.

28 Chi Thang Duong, Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. Efficient streaming subgraph isomorphism with graph neural networks. *VLDB Endow.*, 14(5):730–742, 2021. doi:10.14778/3446095.3446097.

29 M. Dürst and M. Suignard. RFC 3987 int.ized resource identifiers (IRIs), 2005. URL: https://www.ietf.org/rfc/rfc3987.txt.

30 Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel. Graph transformations and model-driven engineering. In *Graph Transformations and Model-Driven Engineering*, pages 1–5. Springer, 2010. doi:10.1007/978-3-642-17322-6_1.

31 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Computation*, volume 7, pages 43–73, 1974.

32 W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *Management of Data (SIGMOD)*, pages 925–936. ACM, 2011. doi:10.1145/1989323.1989420.

33 Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. Query preserving graph compression. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 157–168. ACM, 2012. doi:10.1145/2213836.2213855.

34 Wenfei Fan, Yuanhao Li, Muyang Liu, and Can Lu. Making graphs compact by lossless contraction. In *SIGMOD '21: International Conference on Management of Data*, pages 472–484. ACM, 2021. doi:10.1145/3448016.3452797.

35 Wenfei Fan, Yuanhao Li, Muyang Liu, and Can Lu. A hierarchical contraction scheme for querying big graphs. In *SIGMOD '22: International Conference on Management of Data*, pages 1726–1740. ACM, 2022. doi:10.1145/3514221.3517862.

36 Wenfei Fan, Yinghui Wu, and Jingbo Xu. Adding counting quantifiers to graph patterns. In *Management of Data (SIGMOD)*, pages 1215–1230. ACM, 2016. doi:10.1145/2882903.2882937.

37 Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. A survey of community search over big graphs. *VLDB J.*, 29(1):353–392, 2020. doi:10.1007/S00778-019-00556-X.

38 Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutierrez, Axel Polleres, and Mario Arias. Binary RDF representation for publication and exchange (HDT). *J. Web Semant.*, 19:22–41, 2013. doi:10.1016/J.WEBSEM.2013.01.002.

39 Alessandro Generale, Till Blume, and Michael Cochez. Scaling R-GCN training with graph summarization. In *Companion of The Web Conference 2022*, pages 1073–1082. ACM, 2022. doi:10.1145/3487553.3524719.

40 François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. Incremental structural summarization of RDF graphs. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 566–569. OpenProceedings.org, 2019. doi:10.5441/002/EDBT.2019.57.

41 François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. RDF graph summarization for first-sight structure discovery. *VLDB J.*, 29(5):1191–1218, 2020. doi:10.1007/S00778-020-00611-Y.

42 Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann, 1997. URL: http://www.vldb.org/conf/1997/P436.PDF.

43 Thomas Gottron, Ansgar Scherp, Bastian Krayer, and Arne Peters. LODatio: using a schema-level index to support users in finding relevant sources of linked data. In *Knowledge Capture (K-CAP)*, pages 105–108. ACM, 2013. doi:10.1145/2479832.2479841.

44 Mahdi Hajiabadi, Venkatesh Srinivasan, and Alex Thomo. Dynamic graph summarization: Optimal and scalable. In *IEEE International Conference on Big Data*, pages 545–554. IEEE, 2022. doi:10.1109/BIGDATA55660.2022.10020422.

45 William L. Hamilton. *Graph Representation Learning.* Morgan and Claypool, 2020. URL: https://www.cs.mcgill.ca/~wlh/grl_book/.

46 Wook-Shin Han, Jinsoo Lee, Minh-Duc Pham, and Jeffrey Xu Yu. iGraph: A framework for comparisons of disk-based graph indexing techniques. *Proceedings of the International Conference on Very Large Data Bases (VLDB) Endowment*, 3(1):449–459, 2010. doi:10.14778/1920841.1920901.

47 Jan Hegewald, Felix Naumann, and Melanie Weis. XStruct: Efficient schema extraction from multiple and large XML documents. In *Proceedings of the International Conference on Data Engineering Workshops (ICDE)*, page 81. IEEE Computer Society, 2006. doi:10.1109/ICDEW.2006.166.

48 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, and others. Knowledge graphs. *CoRR*, abs/2003.02320, 2020. arXiv:2003.02320.

49 Neil Immerman. *Descriptive Complexity.* Springer, 1999. doi:10.1007/978-1-4612-0539-5.

50 Xiaowei Jiang, Xiang Zhang, Feifei Gao, Chunan Pu, and Peng Wang. Graph compression strategies for instance-focused semantic mining. In *Chinese Semantic Web Symposium/Chinese Web Science Conf.*, pages 50–61. Springer, 2013. doi:10.1007/978-3-642-54025-7_5.

51 Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing linked data dynamics. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, volume 7882 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2013. doi:10.1007/978-3-642-38288-8_15.

52 A. Kansal and F. Spezzano. A scalable graph-coarsening based index for dynamic graph databases. In *Int. Conf. on Information and Knowledge Management (CIKM)*, pages 207–216. ACM, 2017. doi:10.1145/3132847.3133003.

53 Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, and Ehud Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *Int. Conf. on Data Engineering (ICDE)*, pages 129–140. IEEE, 2002. doi:10.1109/ICDE.2002.994703.

54 K. Kellou-Menouer and Z. Kedad. Schema discovery in RDF data sources. In *Int. Conf. on Conceptual Modeling (ER)*, volume 9381 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2015. doi:10.1007/978-3-319-25264-3_36.

55 Arijit Khan, Sourav S. Bhowmick, and Francesco Bonchi. Summarizing static and dynamic big graphs. *VLDB Endowment*, 10(12):1981–1984, 2017. doi:10.14778/3137765.3137825.

56 Kifayat-Ullah Khan, Waqas Nawaz, and Young-Koo Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015. doi:10.1007/S00607-015-0454-9.

57 Aleks Kissinger, Alex Merry, and Matvey Soloviev. Pattern graph rewrite systems. In *Int. Workshop on Developments in Computational Models (DCM)*, pages 54–66, 2012. doi:10.4204/EPTCS.143.5.

58 Jihoon Ko, Yunbum Kook, and Kijung Shin. Incremental lossless graph summarization. In *Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 317–327. ACM, 2020. doi:10.1145/3394486.3403074.

59 M. Konrath, T. Gottron, S. Staab, and A. Scherp. SchemEX - efficient construction of a data catalogue by stream-based indexing of linked data. *J. Web Semant.*, 16:52–58, 2012. doi:10.1016/J.WEBSEM.2012.06.002.

60 Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Summarizing and understanding large graphs. *Stat. Anal. Data Min.*, 8(3):183–202, 2015. doi:10.1002/SAM.11267.

61 Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Int. Conf. on Data Mining (ICDM)*, pages 313–320. IEEE, 2001. doi:10.1109/ICDM.2001.989534.

62 Kostis Kyzirakos, Manos Karpathiotakis, Konstantina Bereta, George Garbis, Charalampos Nikolaou, Panayiotis Smeros, Stella Giannakopoulou, Kallirroi Dogani, and Manolis Koubarakis. The spatiotemporal RDF store Strabon. In *Advances in Spatial and Temporal Databases - 13th International Symposium*, volume 8098 of *Lecture Notes in Computer Science*, pages 496–500. Springer, 2013. doi:10.1007/978-3-642-40235-7_35.

63 Kristen LeFevre and Evimaria Terzi. Grass: Graph structure summarization. In *SIAM International Conference on Data Mining, SDM*, pages 454–465. SIAM, 2010. doi:10.1137/1.9781611972801.40.

64 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.

65 Xingjie Liu, Yuanyuan Tian, Qi He, Wang-Chien Lee, and John McPherson. Distributed graph summarization. In *Int. Conf. on Information and Knowledge Management (CIKM)*, pages 799–808. ACM, 2014. doi:10.1145/2661829.2661862.

66 Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Comput. Surv.*, 51(3):62:1–62:34, 2018. doi:10.1145/3186727.

67 Yongming Luo, George H. L. Fletcher, Jan Hidders, Paul De Bra, and Yuqing Wu. Regularities and dynamics in bisimulation reductions of big graphs. In *First International Workshop on Graph Data Management Experiences and Systems, GRADES*, page 13. CWI/ACM, 2013. doi:10.1145/2484425.2484438.

68 Yongming Luo, George H. L. Fletcher, Jan Hidders, Yuqing Wu, and Paul De Bra. External memory k-bisimulation reduction of big graphs. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13*, pages 919–928. ACM, 2013. doi:10.1145/2505515.2505752.

69 Jan Martens, Jan Friso Groote, Lars B. van den Haak, Pieter Hijma, and Anton Wijs. A linear parallel algorithm to compute bisimulation and relational coarsest partitions. In *Formal Aspects of Component Software (FACS)*, volume 13077 of *LNCS*, pages 115–133. Springer, 2021. doi:10.1007/978-3-030-90636-8_7.

70 D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language, 2014. [Online, accessed: December 6, 2023]. URL: https://www.w3.org/TR/2004/REC-owl-features-20040210/.

71 Nandana Mihindukulasooriya, María Poveda-Villalón, Raúl García-Castro, and Asunción Gómez-Pérez. Loupe – an online tool for inspecting datasets in the linked data cloud. In *Int. Semantic Web Conf. (ISWC)*. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1486/paper_113.pdf.

72 Tova Milo and Dan Suciu. Index structures for path expressions. In *Int. Conf. Database Theory (ICDT)*, pages 277–295. Springer, 1999. doi:10.1007/3-540-49257-7_18.

73 Seunghwan Min, Sung Gwan Park, Kunsoo Park, Dora Giammarresi, Giuseppe F. Italiano, and Wook-Shin Han. Symmetric continuous subgraph matching with bidirectional dynamic programming. *Proc. VLDB Endow.*, 14(8):1298–1310, 2021. doi:10.14778/3457390.3457395.

74 Muhammad Anis Uddin Nasir, Cigdem Aslay, Gianmarco De Francisci Morales, and Matteo Riondato. Tiptap: Approximate mining of frequent k-subgraph patterns in evolving graphs. *ACM Trans. on Knowledge Discovery from Data*, 15(3), apr 2021. doi:10.1145/3442590.

75 Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *ACM SIGMOD International Conference on Management of Data*, pages 419–432. ACM, 2008. doi:10.1145/1376616.1376661.

76 Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *Proceedings 13th Intl Conference on Data Engineering*, pages 79–90. IEEE Computer Society, 1997. doi:10.1109/ICDE.1997.581741.

77 T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Int. Conf. on Data Engineering (ICDE)*, pages 984–994. IEEE, 2011. doi:10.1109/ICDE.2011.5767868.

78 Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.

79 Alexandros Pappas, Georgia Troullinou, Giannis Roussakis, Haridimos Kondylakis, and Dimitris Plexousakis. Exploring importance measures for summarizing RDF/S KBs. In *The Semantic Web ESWC*, pages 387–403. Springer, 2017. doi:10.1007/978-3-319-58068-5_24.

80 Emmanuel Pietriga, Hande Gözükan, Caroline Appert, Marie Destandau, Sejla Čebirić, François Goasdoué, and Ioana Manolescu. Browsing linked data catalogs with LODAtlas. In *Int. Semantic Web Conf. (ISWC)*, pages 137–153. Springer, 2018. doi:10.1007/978-3-030-00668-6_9.

81 Detlef Plump. Essentials of term graph rewriting. *Electron. Notes Theor. Comput. Sci.*, 51:277–289, 2001. doi:10.1016/S1571-0661(04)80210-X.

82 Miao Qiao, Hao Zhang, and Hong Cheng. Subgraph matching: on compression and computation. *Proceedings of the International Conference on Very Large Data Bases (VLDB) Endowment*, 11(2):176–188, 2017. doi:10.14778/3149193.3149198.

83 Chen Qun, Andrew Lim, and Kian Win Ong. D(k)-index: An adaptive structural summary for graph-structured data. In *Management of Data (SIGMOD)*, pages 134–144. ACM, 2003. doi:10.1145/872757.872776.

84 Mohammad Rashid, Giuseppe Rizzo, Nandana Mihindukulasooriya, Marco Torchiano, and Óscar Corcho. KBQ - A tool for knowledge base quality assessment using evolution analysis. In *Proceedings of Workshops and Tutorials of the International Conference on Knowledge Capture (K-CAP)*, volume 2065 of *CEUR Workshop Proceedings*, pages 58–63. CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-2065/paper13.pdf.

85 Jannik Rau, David Richerby, and Ansgar Scherp. Computing k-bisimulations for large graphs: A comparison and efficiency analysis. In *Graph Transformation - 16th International Conference, ICGT 2023*, volume 13961 of *Lecture Notes in Computer Science*, pages 223–242. Springer, 2023. doi:10.1007/978-3-031-36709-0_12.

86 Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4):15:1–15:41, 2009. doi:10.1145/1516507.1516510.

87 Johann Schaible, Thomas Gottron, and Ansgar Scherp. TermPicker: Enabling the reuse of vocabulary terms by exploiting data from the Linked Open Data cloud. In *The Semantic Web (ESWC)*, pages 101–117. Springer, 2016. doi:10.1007/978-3-319-34129-3_7.

88 Alexander Schätzle, Antony Neu, Georg Lausen, and Martin Przyjaciel-Zablocki. Large-scale bisimulation of RDF graphs. In *Semantic Web Information Management Workshop*, pages 1–8. ACM, 2013. doi:10.1145/2484712.2484713.

89 Kijung Shin, Amol Ghoting, Myunghwan Kim, and Hema Raghavan. SWeG: Lossless and lossy summarization of web-scale graphs. In *World Wide Web Conf. (WWW)*, pages 1679–1690. ACM, 2019. doi:10.1145/3308558.3313402.

90 Qi Song, Yinghui Wu, and Xin Luna Dong. Mining summaries for knowledge graph search. In *ICDM*, pages 1215–1220. IEEE, 2016. doi:10.1109/ICDM.2016.0162.

91 Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. ABSTAT: ontology-driven linked data summaries with pattern minimalization. In *The Semantic Web (ESWC)*, pages 381–395, 2016. doi:10.1007/978-3-319-47602-5_51.

92 Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. Estimating the cardinality of conjunctive queries over RDF data using graph summarisation. In *Proceedings of the 2018 World Wide Web Conference*, pages 1043–1052. ACM, 2018. doi:10.1145/3178876.3186003.

93 Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In *ACM SIGMOD International Conference on Management of Data, SIGMOD*, pages 567–580. ACM, 2008. doi:10.1145/1376616.1376675.

94 Yuanyuan Tian and Jignesh M. Patel. Interactive graph summarization. In *Link Mining: Models, Algorithms, and Applications*, pages 389–409. Springer, 2010. doi:10.1007/978-1-4419-6515-8_15.

95 Thanh Tran, Günter Ladwig, and Sebastian Rudolph. Managing structured and semistructured RDF data using structure indexes. *Trans. Knowl. Data Eng.*, 25(9):2076–2089, 2013. doi:10.1109/TKDE.2012.134.

96 Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *Proceedings of the 25th International Conference on Data Engineering*, pages 405–416. IEEE Computer Society, 2009. doi:10.1109/ICDE.2009.119.

97 Georgia Troullinou, Haridimos Kondylakis, Evangelia Daskalaki, and Dimitris Plexousakis. Ontology understanding without tears: The summarization approach. *Semantic Web*, 8(6):797–815, 2017. doi:10.3233/SW-170264.

98 Octavian Udrea, Andrea Pugliese, and V. S. Subrahmanian. GRIN: A graph based RDF index. In *AAAI*, pages 1465–1470. AAAI Press, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-232.php.

99 Johanna Völker and Mathias Niepert. Statistical schema induction. In *Extended Semantic Web Conf. (ESWC)*, pages 124–138. Springer, 2011. doi:10.1007/978-3-642-21034-1_9.

100 Lanjun Wang, Oktie Hassanzadeh, Shuo Zhang, Juwei Shi, Limei Jiao, Jia Zou, and Chen Wang. Schema management for document stores. *Proceedings of the International Conference on Very Large Data Bases (VLDB) Endowment*, 8(9):922–933, 2015. doi:10.14778/2777598.2777601.

101 Qiu Yue Wang, Jeffrey Xu Yu, and Kam-Fai Wong. Approximate graph schema extraction for semi-structured data. In *Advances in Database Technology (EDBT)*, pages 302–316. Springer, 2000. doi:10.1007/3-540-46439-5_21.

102 Jennifer Widom. Data management for XML: research directions. *IEEE Data Eng. Bull.*, 22(3):44–52, 1999. URL: http://sites.computer.org/debull/99sept/jennifer.ps.

103 Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *Int. Conf. on Data Mining (ICDM)*, pages 721–724. IEEE, 2002. doi:10.1109/ICDM.2002.1184038.

104 D. Yuan, P. Mitra, H. Yu, and C. L. Giles. Iterative graph feature mining for graph indexing. In *Int. Conf. on Data Engineering (ICDE)*, pages 198–209. IEEE, 2012. doi:10.1109/ICDE.2012.11.

105 D. Yuan, P. Mitra, H. Yu, and C. L. Giles. Updating graph indices with a one-pass algorithm. In *Management of Data (SIGMOD)*, pages 1903–1916. ACM, 2015. doi:10.1145/2723372.2746482.

106 Ning Zhang, Yuanyuan Tian, and Jignesh M. Patel. Discovery-driven graph summarization. In *Int. Conf. on Data Engineering (ICDE)*, pages 880–891. IEEE, 2010. doi:10.1109/ICDE.2010.5447830.

107 Xiang Zhang, Gong Cheng, and Yuzhong Qu. Ontology summarization based on RDF sentence graph. In *World Wide Web (WWW)*, pages 707–716. ACM, 2007. doi:10.1145/1242572.1242668.