11-2020

A survey of typical attributed graph queries

Yanhao WANG

Yuchen LI
Singapore Management University, yuchenli@smu.edu.sg

Ju FAN

Chang YE
Singapore Management University, changye.2020@phdcs.smu.edu.sg

Mingke CHAI

Citation

1

A survey of typical attributed graph queries

Yanhao Wang[1], Yuchen Li[2], Ju Fan[3], Chang Ye[2], Mingke Chai[3]

[1] Department of Computer Science, University of Helsinki, Helsinki, 00560, Finland

[2] School of Information Systems, Singapore Management University, Singapore, 178902, Singapore

[3] Key Lab of Data Engineering and Knowledge Engineering (DEKE) and Information School, Renmin University of China, Beijing, 100872, China

Published in World Wide Web, 2020 November, advance online

https://doi.org/10.1007/s11280-020-00849-0

Abstract

Graphs are commonly used for representing complex structures such as social relationships, biological interactions, and knowledge bases. In many scenarios, graphs not only represent topological relationships but also store the attributes that denote the semantics associated with their vertices and edges, known as attributed graphs. Attributed graphs can meet demands for a wide range of applications, and thus a variety of queries on attributed graphs have been proposed. However, these diverse types of attributed graph queries have not been systematically investigated yet. In this paper, we provide an extensive survey of several typical types of attributed graph queries. We propose a taxonomy of attributed graph queries based on query inputs and outputs. We summarize the definitions of queries that fall into each category and present a fine-grained classification of queries within each category by analyzing the semantics and algorithmic motivations behind these queries. Moreover, we discuss the insights of how existing studies address the technical challenges of query processing and outline several promising future research directions.

Keywords

Attributed graph, Knowledge base, Query definition, Query processing, Taxonomy, Survey

1 Introduction

Due to the rising complexity of data generated in the big data era, numerous applications have ubiquitously used graphs for storing complex information. Generally, a graph consists of a set of vertices (a.k.a. nodes) representing the entities and edges representing the relationships between entities. Furthermore, many real-world graphs known as attributed

Electronic supplementary material

The online version of this article (https://doi.org/10.1007/s11280-020-00849-0) contains supplementary material, which is available to authorized users.

Corresponding author: Ju Fan fanj@ruc.edu.cn

graphs associate the vertices and edges with attributes, e.g., types, numbers, and texts[1], to capture rich knowledge embedded in the graph structure. To extract information from attributed graphs, an extensive number of queries with different semantics have been proposed to meet the demand for a diverse range of applications. Some examples are listed as follows.

- **Social networks** can be seen as attributed graphs: users are vertices and their profiles are vertex attributes; user relationships are edges and the information about the relationships (e.g., type and strength) constitutes edge attributes. For relationship discovery, one may issue a query to find how two persons A and B are connected, e.g., "What is the shortest path from A to B through online interactions?. To analyze the relationships between different user groups, we may need a summary to answer a question like "What are the differences between men and women in making online friends?". For friend suggestion, we may need to evaluate the proximity of two persons, e.g., "How likely is it that A will be connected with B?".
- **Biological networks**, such as metabolic networks, are also attributed graphs, where a vertex represents a compound and a directed edge between two compounds means that one compound can be transformed into another one through a chemical reaction. The properties of compounds are vertex attributes while the conditions and enzymes of chemical reactions are edge attributes. One fundamental query is whether there exists a chain of reactions that can transform a compound to another one under some conditions (called *pathway finding*). In addition, given a graph pattern that represents an interesting pattern of reactions, researchers may further explore the graphs to search for similar patterns, which is referred as *subgraph pattern matching*.
- **RDF** is a common data model for schema-free information like knowledge bases. RDF data can be seen as attributed graphs, where each RDF triple (sub, pred, obj) is a directed edge from subject sub to object obj with the relation indicated by predicate pred. The types of subjects, objects, and predicates are attributes. A vertex can have different attributes in different triples and two vertices may be connected by several edges of different attributes. One can issue *keyword queries* or even *natural language questions* to RDF graphs for knowledge extraction, e.g., "John F. Kennedy, successor" or "Who was the successor of John F. Kennedy?" to find the U.S. president after John F. Kennedy, i.e., "*Lyndon B. Johnson*".

Challenges of Attributed Graph Queries Compared with queries on general graphs, attributed graph queries pose unique challenges due to the inherent heterogeneity of attributed graphs. First, the semantics of attributed graphs are significantly different from the underlying general graphs. Applying queries that do not take attributes into account to attributed graphs could lead to semantic inconsistency. For example, general subgraph pattern matching only considers the topology mapping but ignores the attribute mapping. For the team formation problem, we consider not only the relationships between persons but also their roles in the team. In this case, one should use attributed query patterns to guarantee the semantic consistency. Second, attributed graph queries allow more diverse types of inputs, e.g., aggregation conditions, keywords, and nature language questions, which capture richer semantics associated with attributes to satisfy the demands in real-world

[1]Graphs associated with spatial and temporal attributes are often referred to as *spatial-temporal graphs* [12, 82]. Since spatial-temporal graph queries are typically orthogonal to the attributed graph queries we investigate, we omit detailed discussions on them in this survey.

applications. Third, attributed graph queries often face greater algorithmic challenges due to the high order information captured by attributes. For example, the index sizes for reachability queries may grow exponentially on attributed graphs due to the combinatorial nature of attribute constraints.

To address the challenges, extensive efforts have been made in the past decades to advance the field of attributed graph data management. This article systematically surveys existing studies on attributed graph queries. We provide a taxonomy of attributed graph queries by categorizing existing queries based on their inputs and outputs. To help readers better understand each type of queries, we present their motivations and demonstrate the application scenarios where a attributed graph query is used. Moreover, we discuss the insights of how existing studies address the technical challenges of query processing and outline several promising future research directions. Our main goal is to promote the rich literature developed on this topic to a wider audience, especially for end-users of graph databases. By providing the taxonomy, we aim to ease the job for end-users on identifying appropriate queries to address their application requirements in practice.

Taxonomy of Attributed Graph Queries We now introduce our taxonomy, which classifies attributed graph queries into two general categories: *structured* and *unstructured* queries. Structured queries have pre-defined query format, such as paths, subgraphs and SQL-like operators. End-users who are familiar with the graph schema can extract valuable information by employing the low-level structured queries. In contrast, unstructured queries do not require any prior knowledge about the graph schema, and allow end-users to issue free-form queries, e.g., node proximity, keywords, and natural language questions, that naturally express the query semantics. In this survey, we propose a fine-grained taxonomy that further dissects structured and unstructured queries into six major categories, which are listed as follows:

- **Path query:** the inputs are vertices (and/or) constraints; the outputs are paths. We further divide path queries into *reachability query*, *shortest-path query*, and *regular path query* according to query objectives.
- **Subgraph pattern query:** the inputs and outputs are both subgraphs. Given a *query graph* and *data graph(s)*, a subgraph pattern query returns the subgraphs that best match the query graph from data graph(s). Based on the matching criteria, subgraph pattern queries can be divided into *exact match query* (a.k.a. *subgraph isomorphism*), *approximate match query*, and *extended match query*. As exact and approximate match queries have been extensively surveyed [65, 66, 99, 109, 130, 137, 202], we focus on reviewing existing studies on *extended match query*, which extends exact or approximate match queries with different contexts.
- **Aggregate query:** the inputs are aggregation conditions; the outputs are aggregate graphs/values. We have two subcategories, namely, *graph OLAP* and *egocentric aggregate query*. The classification is based on that the aggregation graph is generated for the entire graph or a local subgraph, respectively.
- **Similarity search:** the input are vertices; the output is a set of vertices with high similarities to query vertices. Typically, the similarity of two vertices is measured by a score in the range [0, 1]. According to similarity measures, we further classify existing methods into *path-based approaches* and *graph embedding-based approaches*.
- **Keyword search:** the inputs are a set of keywords; the outputs are top-k substructures which are the most relevant to query keywords. We further classify existing methods

by the substructures to represent query semantics, i.e., *tree-based* and *subgraph-based semantics*.

– **Natural language query:** the inputs are natural language questions; the outputs are substructures that can match the semantics of input questions. According to the schemes for question translation, we classify existing methods into *query-graph-based approach* that translates questions into structured queries for extracting answers and *end-to-end approach* that directly extracts subgraphs as answers without intermediate steps.

An illustrative example for our taxonomy is shown in Figure 1. We present an attributed graph with several types of vertices (i.e., *person, music, movie*) and edges (i.e., "*friend, supervision, spouse*" between persons; "*like*" between person and music/movie; "*theme_music*" between music and movie). Then, we show a use case of each query category in our taxonomy. We give an exemplar input of each query category and illustrate its result(s) on the attributed graph.

Differences from Prior Surveys There have been several surveys on graph queries. Subgraph pattern queries are extensively surveyed in [65, 66, 130, 202]. Lee et al. [109] and Ma et al. [137] experimentally evaluate the performance of different subgraph isomorphism algorithms. Katsarou et al. [99] compare the performance and scalability of different index structures for graph queries. Yu and Cheng [215] provide a survey of graph reachability queries. Keyword search on graph data is reviewed in [189]. Sommer [170] reviews the techniques on shortest-path queries in static graphs. Shi et al. [166] summarize existing studies on heterogeneous information network (HIN) analysis, where similarity search on HINs is included. Fang et al. [60] provide a survey of community search on graphs. Natural language question answering on graphs is surveyed in [41, 81]. Our survey is different from existing ones from three aspects. First, existing surveys focus on a specific type of graph queries but we introduce a taxonomy of queries and summarize each query category in our taxonomy

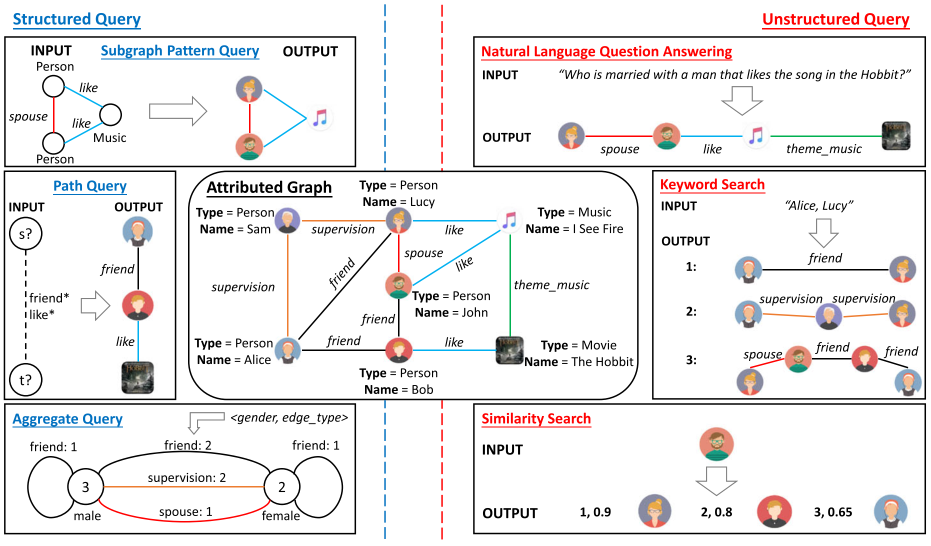


Figure 1 An illustration of our taxonomy for attributed graph queries

systematically. In this way, our survey would be helpful to end-users to identify appropriate queries for their application scenarios. Second, for query types that are well-studied on both attributed and non-attributed graphs, e.g., reachability and shortest-path queries, we highlight their diversity in semantics and unique challenges in algorithm design on attributed graphs. Third, some existing surveys are outdated and a large amount of recent literature is not covered.

Graph query languages are surveyed in [2, 195]. They investigate declarative languages, e.g., SPARQL [74], Cypher[2], and Gremlin[3], that provide high-level interfaces for expressive and efficient querying of attributed graphs. However, the focus of this paper is different from them: we consider primitive and low-level queries on attributed graphs, some of which form the foundations of high-level declarative languages. Moreover, the queries that involve rich semantics, i.e., *similarity search*, *keyword search* and *natural language question answering*, have not been well supported by existing declarative graph query languages.

Organization The remainder of this article is organized as follows. Section 2 defines the basic concepts used in the article. From Sections 3 to 8, we summarize each category of attributed graph queries in our taxonomy, respectively. Finally, we conclude the article in Section 9.

Before moving on to the subsequent section, we first summarize existing works in each category of our taxonomy in Table 1.

2 Preliminary

We first give a formal definition of *attributed graph*.

Definition 1 (Attributed Graph) An attributed graph $G = (V, E, \Sigma, L)$ is a graph where (1) V is the set of vertices and $v \in V$ is a vertex in G; (2) $E \subseteq V \times V$ is the set of edges and $e = (u, v) \in E$ is an edge between two vertices u, v; (3) Σ is the domain of attributes; (4) L is the function that assigns attributes to vertices and edges. Specifically, we use $l(v)$ and $l(e)$ to represent the attributes of vertex v and edge e respectively.

In this paper, we consider three types of attributed graphs with different types of attributes: (1) *labeled graph*: each vertex/edge has a single categorical attribute (known as *label*) that indicates the vertex/edge type; (2) *multi-labeled graph*: each vertex/edge has multiple categorical attributes (e.g., *keywords*); (3) *property graph*: both vertices and edges have multi-dimensional attributes in different domains (e.g., categorical, numerical, and so on). Moreover, several query types are based on *edge-weighted graphs*. For these graphs, there is an additional function $w : E \rightarrow \mathbb{R}_{\geq 0}$ assigns a weight $w(e)$ to each edge denoting the weight of edge e. We consider a query can be performed on either one large graph G (called a *data graph*) or a collection of n graphs $\mathbb{G} = \{G_1, \ldots, G_n\}$ (called a *graph database*).

Next, we introduce the notations of *path* and *subgraph* on attributed graphs.

[2]https://www.opencypher.org/
[3]https://tinkerpop.apache.org/gremlin.html

Table 1 Classification of existing works on attributed graph queries (we use "*" to denote surveys)

Category	Structured	Subcategory	List of Works
Path query	Y	Reachability query	[51, 54, 94, 151, 169, 185, 243]
		Shortest-path query	[10, 11, 18, 40, 75, 118, 157, 228]
		Regular path query	[4, 5, 19, 39, 79, 103, 119, 138–140, 148, 149, 186, 192, 199, 229]
Subgraph pattern query	Y	Exact match query	[15, 20, 37, 73, 156, 162, 181, 200, 204]
		Approximate match query	[32, 33, 46, 52, 53, 63, 64, 80, 100, 101, 111, 122, 136, 161, 175–177, 191, 197, 205, 208, 225, 232, 236, 238, 241]
		Extended match query	[45, 55, 67, 88–90, 144, 150, 153, 155, 163, 210, 221, 226, 227, 233, 234]
Aggregate query	Y	Graph OLAP	[13, 25, 154, 193, 214, 231]
		Egocentric aggregation	[49, 143, 203]
Similarity search	N	Path/subgraph-based	[35, 69, 77, 104, 105, 141, 142, 164, 165, 167, 173, 180, 187, 188, 217, 222, 223] [62, 87, 235]
		Graph embedding-based	[126–129, 184] [23, 68, 190]*
Keyword search	N	Tree-based	[1, 14, 43, 83, 84, 95, 97, 108, 116, 121, 133, 134, 168, 209]
		Subgraph-based	[26, 27, 48, 57, 59, 61, 72, 86, 96, 98, 108, 110, 112, 123, 179, 207, 239, 240]
Natural language query	N	Query graph-based	[8, 38, 78, 131, 132, 147, 158, 182, 183, 206, 212, 242]
		End-to-end	[29, 44, 211] [41, 81]*

Definition 2 (Path) A path P in graph G is a sequence $\langle v_0, e_1, v_1, \ldots, v_{i-1}, e_i, v_i, \ldots, e_m, v_m\rangle$ for any $m > 0$ (m is the length of P), where $v_i \in V$ for $i \in [0, m]$ and $e_i = (v_{i-1}, v_i) \in E$ for $i \in [1, m]$. Specifically, P is a simple path if it does not have duplicate vertices in P.

The edge-attribute set $L_e(P)$ of P is denoted by the union of its edges' attributes, i.e., $L_e(P) = \bigcup_{e\in P} l(e)$. Similarly, we can define the vertex-attribute set $L_v(P)$ of P by $L_v(P) = \bigcup_{v\in P} l(v)$. More generally, the attribute set $L(P)$ of P is the union of its edge-attribute set and vertex-attribute set, i.e., $L(P) = L_v(P) \cup L_e(P)$. For edge-weighted graphs, the weight of P is defined as $w(P) = \sum_{e\in P} w(e)$.

Definition 3 (Subgraph) A graph $S(V_s, E_s, \Sigma, L_s)$ is a subgraph of graph G ($S \subseteq G$) if (1) $V_s \subseteq V$, (2) $E_s \subseteq E$, (3) $\forall e = (u, v) \in E_s$, $u, v \in V_s$, and (4) $\forall v \in V_s, \forall e \in E_s$, $l_s(v) = l(v)$ and $l_s(e) = l(e)$.

Especially, a subtree T of G ($T \subseteq G$) is a tree-structured subgraph of G.

3 Path query

In this section, we review path queries whose inputs are vertices and/or constraints and outputs are paths. Based on the differences in query objectives, we further divide path queries into three subcategories. The simplest case is *reachability query* that returns whether a vertex can reach another vertex through any path satisfying some attribute constraint. Furthermore, a *shortest-path query* returns the path that has the minimal weight and satisfies some attribute constraint between two vertices in a (weighted) data graph. The most general case is *regular path query* that enumerates all pairs of vertices that are connected by a path satisfying a regular expression constraint in a data graph. In Sections 3.1–3.3, we will review existing work on each subcategory of path queries, respectively.

3.1 Reachability query

In this subsection, we summarize existing studies on *reachability queries*, one of the most fundamental graph queries, in the context of attributed graphs. A reachability query on an attributed graph asks whether there exists any path satisfying some attribute constraint from a source vertex to a destination vertex. There are many real-world applications of reachability queries on attributed graphs. In social networks, it can be used for relationship discovery: for two persons A and B, it finds if A can reach B with an attribute constraint. For example, it checks whether A reaches B via common friends (i.e., the edge type is 'friend_of'). In metabolic networks, it can be applied to pathway finding: for two compounds C and D, it finds if C can be transformed to D by the action of enzyme X. For instance, a query can ask whether C can reach D via a path whose attribute set is '$\{X\}$'. In addition, reachability query is also a building block for more complex queries, e.g., regular path queries, and declarative graph query languages, e.g., SPARQL [74].

Generally, existing literature on reachability queries is mostly specific for labeled graphs. Next, we will classify them into two categories according to the constraint type: (1) simple label constraint and (2) regular expression constraint.

Label constraint reachability query First, we formally define the label constraint reachability (LCR) query on attributed graphs [94, 151, 185, 243].

Definition 4 (LCR) For a data graph G, two vertices $u, v \in V$, and a label set $C \subseteq \Sigma$, if there is a path P from vertex u to vertex v such that $L(P) \subseteq C$, we say v is C-reachable from u. For a source vertex u, a destination vertex v, and a label set C, a LCR query asks whether v is C-reachable from u.

Example 1 Let us consider the labeled graph in Figure 2. For a reachability query without any constraint, vertex 6 is reachable from vertex 1 because there are two paths $P_1 = \langle 1, a, 2, c, 5, a, 6\rangle$ and $P_2 = \langle 1, a, 2, a, 3, b, 4, a, 5, a, 6\rangle$ between vertex 1 and 6. However, for a LCR query with $C_1 = \{a\}$, vertex 6 is not C_1-reachable from vertex 1 because $L(P_1), L(P_2) \not\subseteq C_1$. For another LCR query with $C_2 = \{a, c\}$, vertex 6 is C_2-reachable from vertex 1 since $L(P_1) \subseteq C_2$.

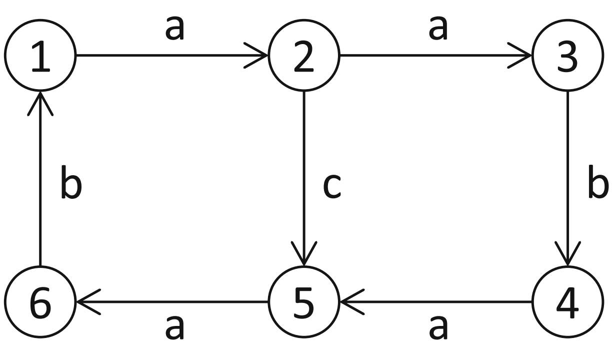


Figure 2 An example for reachability query on a labeled graph

Two basic approaches to LCR processing are (1) online DFS/BFS with the label constraint for search space pruning and (2) precomputing the transitive closure (TC) matrix of the graph. The prior approach does not build indices and incurs significant query overheads. The latter one can answer any reachability query instantly but takes a large amount of time and space to build and store the full TC matrix. Moreover, the TC must contain label information for LCR query processing. And there could be $O(2^{|\Sigma|})$ possible labels in a labeled graph. Therefore, maintaining the full TC matrix would be prohibitive for large graphs. To strike the balance between index size and query efficiency, different schemes have been proposed for index compression. First of all, several methods propose to partition the data graph into local subgraphs based on different schemes such as spanning trees [94] and strongly connected components [243]. A TC matrix is maintained for each local subgraph and used for queries between any two vertices in the same local subgraph. To answer a LCR query between vertices in different local subgraphs, online DFS/BFS and local TC matrices will be used jointly. Then, Valstar et al. [185] propose a landmark indexing for LCR queries. It selects a small number of landmarks and precomputes the TC of each landmark. To answer a LCR query, it first conducts a label-respecting BFS from the source vertex to a landmark and then exploits the information maintained for the landmark to get a shortcut to the target vertex. Recently, Peng et al. [151] propose a label-constrained 2-hop indexing for LCR queries. Compare with the aforementioned methods, it shows both lower worst-case query time and better empirical performance.

Regular expression constraint reachability query Several studies [51, 54] use regular expressions to generalize the label constraints for reachability queries.

Definition 5 (RECR) For a data graph G, two vertices u, v, and a regular expression R that specifies the class of admissible label sequences, a RECR query asks whether there exists a path P from u to v such that the concatenation of labels in P matches R.

Example 2 We also consider the labeled graph in Figure 2. Given a regular expression $\mathsf{R}_1 = a^*b^*$, vertex 6 is not reachable from vertex 1 because the paths from vertex 1 to 6 do not match R_1. Conversely, vertex 1 is reachable from vertex 6 because the label sequence of path $P_3 = \langle 1, b, 6 \rangle$ is $\langle b \rangle \in \mathsf{R}_1$.

The basic approach to RECR query processing is online BFS. During BFS, an automation derived from the regular expression is used to prune paths whose label sequences do not satisfy the regular expression. Moreover, bi-directional [51] and MapReduce-based [54] BFS schemes are proposed for acceleration.

3.2 Shortest-path query

In this subsection, we review another fundamental graph query, namely *shortest-path query*, on attributed graphs. A shortest-path query on an attributed graph asks for the path with the minimal weight among all the paths that satisfy an attribute constraint from the source vertex to the destination vertex. Its most important application is *route planning*. In a transportation network, each vertex represents a position and each edge denotes a route between two positions. Each route is labeled by its type while the estimated traveling time of a road is its weight. A user may request the shortest path from the source to the destination with restrictions, e.g., avoiding toll roads. In addition, shortest-path queries can also be used for relationship discovery in social networks and pathway finding in metabolic networks.

Label constraint shortest-path query First, we formally define the label constraint shortest-path (LCSP) query on attributed graphs [18, 75, 157].

Definition 6 (LCSP) Let G be a weighted data graph. For two vertices $u, v \in G$ and a label set $C \subseteq \Sigma$, a LCSP query returns a path P from u to v s.t. $L(P) \subseteq C$ and $w(P)$ is minimal.

Example 3 Figure 3 gives a labeled graph for the illustration of shortest-path query. We assume the weights of all edges are equal to 1. First, the result of an unconstrained shortest-path query from vertex 1 to 8 returns $P_1 = \langle 1, a, 2, a, 8\rangle$ with $w(P_1) = 2$. Second, for a LCSP with $C_1 = \{b\}$, the result becomes $P_2 = \langle 1, b, 3, b, 7, b, 4, b, 8\rangle$ with $w(P_2) = 4$ because P_2 is the only path satisfying C_1. Finally, for a LCSP with $C_2 = \{b, c\}$, $P_3 = \langle 1, b, 3, c, 4, b, 8\rangle$ with $w(P_3) = 3$ is returned since P_3 has the minimal weight among all valid paths.

Similar to LCR queries, the idea of LCSP query processing is to extend the Dijkstra's algorithm and different indices from non-attributed to attributed graphs. The main challenge

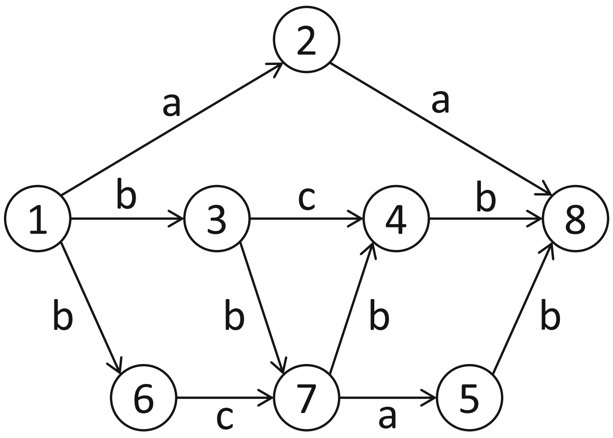


Figure 3 An example for shortest-path query on a labeled graph

is also the huge number (i.e., $2^{|\Sigma|}$) of possible constraints. For LCSP query processing, a general framework [18, 75, 157] is to first partition a data graph into subgraphs such that each subgraph contains all edges with the same label. Then, an index for non-attributed graphs can be built on the subgraph for one label. Finally, sub-paths that contain only one label can be efficiently computed by using these indices and then concatenated into the shortest path.

Regular expression constraint shortest-path query The shortest-path query on attributed graphs with regular expression constraints is one basis of *route planning* and *journal planning*. Intuitively, regular expressions are able to represent more general types of user preferences and constraints than simple label constraints. We give a formal definition of regular expression constraint shortest-path (RECSP) query [10, 11, 40].

Definition 7 (RECSP) For a weighted data graph G, two vertices u, v, and a regular expression R that specifies the class of admissible label sequences, a RECSP query returns a path P from u to v such that the concatenation of labels in P matches R and $w(P)$ is minimal among all matched paths.

Example 4 We continue with the example in Figure 3. For a regular expression $\mathsf{R}_1 = b^*cb^*$, we can find two paths matching R_1: $P_1 = \langle 1, b, 3, c, 4, b, 8\rangle$ and $P_2 = \langle 1, b, 6, c, 7, b, 4, b, 8\rangle$. Obviously, we have $w(P_1) < w(P_2)$ and P_1 is returned as the result.

Existing methods for RECSP extend the Dijkstra's algorithm by constructing automata of regular expressions for path pruning. Theoretically, Barrett et al. [11] prove that the formal-language-constrained shortest path query is solvable in polynomial time as long as the formal language is context-free. Efficient indices for shortest-path queries on non-attributed graphs, e.g., ALT [10] and contraction hierarchies [40], have also been extended for RECSP. A special case of RECSP query called *correlation constraint shortest path* (CCSP) query is studied in [228]. Compared with RECSP, CCSP relaxes the restriction on the ordering of labels. A hybrid relation encoding (HyRE) method, which encodes both topological and label information in a compact way, is proposed for CCSP query processing.

Top-*k* shortest-path query Liang et al. [118] propose a variation of shortest-path query, namely, top-k shortest path query: for a weighted data graph G, two vertices u, v, and a predefined label sequence C, a top-k shortest path query asks to find the k loopless paths with the minimum weights among all paths matching C from u to v. Different from shortest-path queries, the top-k shortest path query is NP-hard. They propose a prophetic heuristic algorithm that combines graph preprocessing with the A* search algorithm for approximate top-k shortest path query processing.

3.3 Regular path query

In this subsection, we review existing work on regular path queries (RPQs), which are more general than reachability and shortest-path queries. RPQ is one of the central components of graph query languages, e.g., XPath, SPARQL, Cypher, and so on. It is also crucial for navigational queries on XML documents and knowledge graphs.

First, we give the general formulation of the regular path query (RPQ) [139].

Definition 8 (RPQ) Given a data graph G and a regular expression R, a RPQ enumerates each pair of vertices connected by a simple path such that the concatenation of labels along the path matches R.

RECR and RECSP are special cases of RPQ as they only check whether a valid path exists or find the path with the minimal weight for given vertices and regular expressions while RPQ enumerates all valid paths.

Example 5 Figure 4 depicts a knowledge graph that shows the predecessor and father relationships among seven British monarchs. A RPQ $Q_1 = (s?, F, t?)$ enumerates five pairs of vertices connected by an edge of type 'is_father_of', e.g., $\langle 2, F, 1\rangle$ and $\langle 3, F, 4\rangle$.

Although RECR and RECSP are solvable in polynomial time [11], the complexity of RPQ is NP-hard [139]. Despite the theoretical hardness, RPQ is tractable for many classes of regular languages in practice. Interested readers can refer to [2, 195] for extensive surveys on the theoretical results of RPQ. There are several methods for processing RPQs. A general approach for RPQ evaluation is to convert the regular expression into an automaton and then to perform BFS/DFS on the graph while using the automaton for search space pruning. Several label- and query- oriented optimization strategies for RPQ processing is proposed in [103, 148]. Wadhwa et al. [186] propose a random walk-based sampling algorithm to approximate RPQ processing with theoretical guarantees. Pacaci et al. [149] study the problem of evaluating persistent RPQs incrementally on streaming graphs.

Extensions of RPQ Provenance-aware RPQs [39, 192, 199] refer to RPQs that not only return the pairs of vertices connected by any matched simple paths but also need to generate the provenance why a pair of vertices is returned. Specifically, a provenance-aware RPQ retrieves a *subgraph* consisting of all the simple paths matching R. Recent studies have supported provenance-aware RPQs in real-world systems. Dey et al. [39] provide a scheme for provenance-aware RPQs by using relational query engines. Xin et al. [199] and Wang et al. [192] integrate provenance-aware RPQs into Pregel.

Several studies focus on extending the semantics of RPQs. Liptchinsky et al. [119] propose an extension of RPQ that retrieves a group of vertices based on group structural characteristics and relations to other vertices or groups. They integrate this extension into graph query languages based on RPQ. Bai et al. [5] define a query language G-path that extends RPQs by expressing the constraints on a variety of attribute types (e.g., categorical, numerical, boolean) in property graphs.

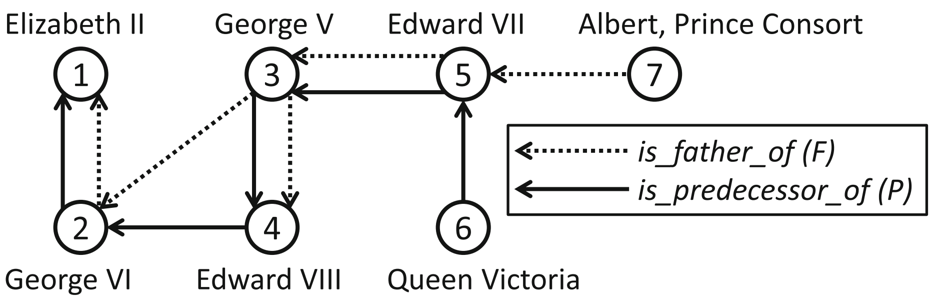


Figure 4 An example for RPQ on a knowledge graph in [199]

In recent studies [79, 229], regular expressions have been generalized to context-free grammars to express the constraints in path queries. Such context-free path queries (CFPQs) increase the expressive power of RPQs as context-free grammars are generalizations of regular expressions. Similar to RPQ, CFPQ is also in general NP-complete [79, 229]. Existing approaches to CFPQ processing are heuristic methods based on matrix multiplication [4] or grammar parse table [138].

"Reverse" regular path queries Meng et al. [140] and Bonifati et al. [19] investigate the "reverse" problem of RPQs: given exemplar pairs of vertices provided by a user, a reverse RPQ aims to find a RPQ whose outputs are what the user expects. Meng et al. [140] define the similarity measure between exemplar pairs and regular paths and select the most relevant regular path. Bonifati et al. [19] propose to learn a RPQ from both positive and negative examples. They formalize the notion of learnability and introduce an interactive method for path discovery with the help of users.

3.4 Summary

Path queries are one of the most fundamental class of graph queries. We classify three standard forms of path queries, i.e., reachability query, shortest-path query, and regular path query. Naturally, the approaches to processing path queries on attributed graphs are mostly extended from those for non-attributed graphs, with additional strategies to handle the attribute constraints.

Despite the proliferation of researches on the standard forms of path queries, open challenges still exist. First, most of the path queries are defined on static graphs where both structures and attributes do not change over time. Real-world graphs, e.g., social networks and communication networks, are highly dynamic and keep evolving. To track the changes of relationships in real time, it is essential to extend path queries to dynamic graphs, which is still largely unexplored yet. Second, extremely large attributed graphs with billions of vertices become prevalent in the big data era. It is a great challenge to answer path queries on these huge attributed graphs. Interesting future directions may include processing path queries in a distributed environment when the graph cannot fit in a single machine and using modern hardware, e.g. GPUs and FPGA, for query acceleration. Third, some types of path queries, e.g., k-disjoint shortest paths [71] and k-dissimilar paths [34], that are extensively studied on general graphs have not yet been considered on attributed graphs. It is interesting to support them on attributed graphs. Fourth, most of the path queries assume the constraints (i.e., path patterns) are predefined or provided by users. Nevertheless, users may not be able to identify appropriate query patterns. Therefore, another possible direction is to combine path query with path pattern discovery. Although there have been a few attempts along this direction [19, 118, 140], the area is still immature.

4 Subgraph pattern query

Subgraph pattern matching is one of the most important and extensively studied areas for graph query processing. Unlike path queries in the previous section, the input of subgraph pattern matching is a query graph (i.e., *pattern*) and it aims to search for similar subgraphs to the query pattern from a *data graph* or a *graph database*.

There are numerous real-world problems that can be modeled as the subgraph pattern matching problem [21, 24, 50, 85, 106, 120, 224]. For example, subgraph pattern matching in graph databases is a common problem in chemical- and bio- informatics[85, 224], where compounds and proteins are represented as graphs. Scientists show that two proteins with structural similarity indicate a high chance of similar functionalities. By searching the parts of proteins, we hope to identify the proteins with a similar function. Likewise, in collaboration networks where each node denotes a person labeled with her expertise and each edge indicates the collaboration relationship between two persons, subgraph pattern matching is employed for the team formation problem where a selected team should satisfy certain skill requirements as well as collaboration requirements between members in the team [106].

A central problem in subgraph pattern matching is to determine the similarity function between the query pattern and a candidate subgraph. A straightforward definition of the similarity function is *graph isomorphism*. It leads to the well-studied subgraph isomorphism problem [15, 20, 37, 73, 156, 162, 181, 204]. Formally,

Definition 9 (Graph Isomorphism) a graph $G = (V, E, \Sigma, L)$ is isomorphic to a query graph $Q = (V_q, E_q, \Sigma, L_q)$ if there is a bijection $f : V \mapsto V_q$ such that $\forall e = (u, v) \in E$ it holds that $f(e) = (f(u), f(v)) \in E_q$ and $l(u) = l_q(f(u))$ and $l(v) = l_q(f(v))$ and $l(e) = l_q(f(e))$.

Graph isomorphism ensures the complete topological information through a bijective map between the pattern and candidates, and can be applied to scenarios where one must find out subgraphs which are exact copies (called *instances*) of the query pattern. For example, extracting exact copies of a program structure is a key operation for designing pattern detection in the software engineering domain [120].

Although isomorphism-based pattern matching is effective in some applications, it is widely known in the literature that it suffers from two major drawbacks. First, the decision version of subgraph isomorphism is NP-complete [36]. For some applications, one needs to enumerate all instances w.r.t. the query graph [174, 230], which incurs prohibitively high time complexity and is not scalable for querying large graphs. Second, the requirement of bijective map for subgraph isomorphism could be too strict in many cases. For example, one may not be able to find any matched instance even if there are similar enough candidates but are not isomorphic to the pattern. The problem becomes more evident once the data source is noisy [220].

To overcome the challenges, huge efforts have been made to relax subgraph isomorphism in two major categories. The first category summarizes a line of works [33, 46, 53, 100, 101, 161, 175–177, 191, 197, 205, 208, 225, 232, 236, 238] that design similarity functions which measure the "proximity" between the query graph and candidate subgraphs. Only top-k candidates that are the most similar to the query graph or candidates whose similarity scores exceed a certain threshold are returned as query results. The second category relaxes the bijective map to less restrictive *relations* [32, 52, 63, 64, 80, 111, 122, 136, 241]. They permit mappings from a node in the query graph to a group of nodes in the data graph and/or mappings from an edge in the query graph to a path in the data graph.

Since there have been several extensive surveys on subgraph isomorphism [99, 109, 137], subgraph similarity search [65, 66, 130], and relation-based relaxations [63, 64, 136], we focus on surveying a broader range which extends the aforementioned subgraph pattern queries by incorporating additional ranking functions to find subgraphs that consider diversity [210], skyline [234], and a number of other contexts [45, 89, 135, 144, 155].

4.1 Extended subgraph pattern queries

All the subgraph pattern queries that have been introduced so far find matches mostly based on how *similar* the candidate matches are to a query pattern. There are several existing studies that try to extend beyond this scope and propose extended queries that consider additional contexts for determining the query results. In this section, we review several extended queries popular in the literature, i.e., representative query, diversified query, skyline query, why-not query, and supergraph query.

Representative query As previously discussed, isomorphism could be too strict that subgraph matches are hardly found. Thus, approximate subgraph matching selects subgraph candidates which are relevant to the query graph based on the certain similarity function. One of the most commonly used similarity function is graph edit distance [67, 221], which is formally defined as follows:

Definition 10 (Graph Edit Distance) Given a set of edit operations, the graph edit distance between two graphs G_1 and G_2 can be defined as $d(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in P(G_1,G_2)} \sum_{i=1}^{k} c(e_i)$ where $P(G_1, G_2)$ denotes the set of edit paths transforming G_1 into G_2 and $c(e) \geq 0$ is the cost of graph edit operation e.

Although subgraph matching by graph edit distance can produce enough results. However, the results are often heavily redundant, which overwhelm the end users. Thus representative query is proposed to restrict the number of matches by outputting the results that best represent of all possible matches.

The representativeness of a matched subgraph is based on θ-neighborhood defined as the following.

Definition 11 (θ-Neighborhood) Given a graph database $\mathbb{G}$, a query pattern Q, the θ-neighborhood of a graph $G \in \mathbb{G}$, denoted as $N_\theta(G)$, contains all matched graphs within a distance threshold θ from G, i.e., $N_\theta(G) = \{G' \in L_q | d(G, G') \leq \theta\}$ where $d(G, G')$ is the graph edit distance, and L_q is the set of graphs from $\mathbb{G}$ with a distance threshold θ from Q. The representativeness $\pi_\theta(S)$ of a set of graphs S is defined as the proportion of relevant graphs represented by S, i.e., $\pi_\theta(S) = \frac{|\bigcup_{G\in S} N_\theta(G)|}{|L_q|}$.

Definition 12 (Representative Subgraph Query) Given a graph database $\mathbb{G}$, a query pattern Q, and a budget k, the top-k representative subgraph query computes a set of graph S^* such that $S^* = \arg\max_{S\subseteq L_q:|S|=k}\{\pi_\theta(S)\}$ where $L_q \subseteq \mathbb{G}$ is the set of graphs from $\mathbb{G}$ within a distance threshold from Q, and $\pi_\theta(S)$ is the representativeness of S.

Example 6 Figure 5 is an example for reducing redundant results in a representative query. Given a query Q and the data graph G (each node has a label), four subgraphs of G are within edit distance 1 from Q, namely G_1, G_2, G_3 and G_4, which forms L_q. However, the matched graphs are redundant, e.g., G_1, G_2 and G_3 as they have a large overlap. More specifically, assuming $\theta = 1$ and we have $N_1(G_1) = N_1(G_2) = N_1(G_1) = \{G_1, G_2, G_3\}$. Hence, if G_1 and G_2 are selected as the result, the representativeness is $\pi_1(\{G_1, G_2\}) = \frac{3}{4}$. The optimal result selects $\{G_1, G_4\}$ where and representativeness is $\pi_1(\{G_1, G_4\}) = 1$.

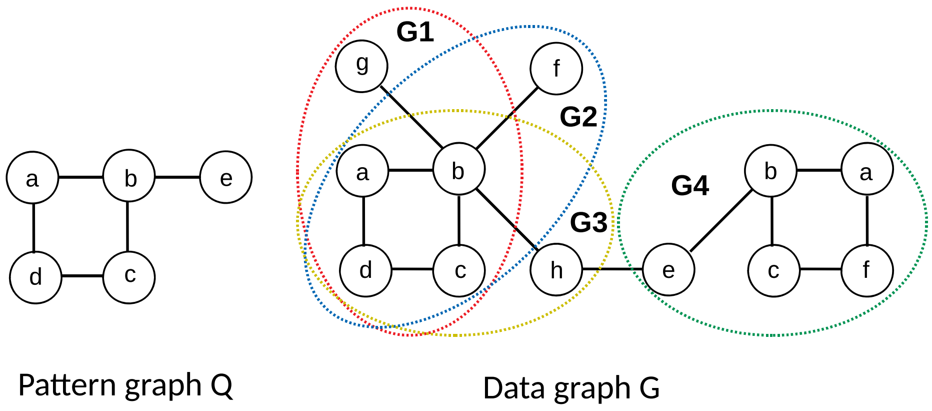


Figure 5 An example for representative query

Ranu et al. [155] propose the aforementioned measure to capture the representativeness of the result set, and prove the top-k representative query is NP-hard. Furthermore, they propose an index structure called NB-Index by indexing the θ-neighborhoods of the graph database to speedup query processing.

Diversified query Like representative query, diversified query [153] also aims to find a set of matches that are both similar to the query and *not* similar to each other.

Existing literature studies two types of diversified queries for subgraph pattern matching. Fan et al. [55] first propose a query that, given a data graph G, a pattern Q, and an output node $u_o \in Q$, finds a set of k vertices (denoted as S) in the data graph that matches u_o in terms of relation-based matching from G to Q. The diversified version of the objective function that optimizes for S is defined as:

$$F(S) = (1-\lambda)\sum_{v_i \in S}\delta_r(u_o, v_i) + \frac{2\lambda}{k-1}\sum_{v_i, v_j \in S, i<j}\delta_d(v_i, v_j)$$

where $\delta_r(u_o, v_i)$ represents the number of subgraph matches from Q to G that matches v_i to u_o, $\delta_d(v_i, v_j)$ denotes the number of distinct subgraph matches that v_1 or v_2 matches to u_o, and λ is a parameter set by users. In other words, $\delta_r(u_o, v_i)$ measures how similar between u_o and v_i whereas $\delta_d(v_i, v_j)$ penalizes if the subgraph matches which include v_i and v_j have a significant overlap. Fan et al. [55] show that finding the diversified results is NP-complete, and present an approximation algorithm called TopKDiv, as well as a heuristic algorithm called TopKDH for this problem.

Yang et al. [210] study another type of diversified top-k subgraph query in the term of maximum k-coverage. In particular, for a data graph G and a pattern Q, it selects a set of k subgraphs that are isomorphic to Q such that S covers the largest number of vertices in the data graph. Yang et al. [210] propose an approximation algorithm named DSQL, which achieves an approximation ratio of $0.25(1 + \max(\frac{1}{k}, \frac{1}{z}))$ where z is the number of vertices in the query graph.

Skyline query Subgraph skyline queries raise when there are numerical values associated with the vertices in the graph. A motivating scenario for this query is finding the points of interests (POIs) [150]. Suppose a traveler wants to visit a neighborhood that is close to a park and a mall. In this scenario, locations can be represented as nodes and distances between them can be modeled as edge weights in a neighborhood graph. There exists two neighborhoods A and B that both satisfy this query, but A has shorter distance to park, whereas B is closer to mall. Thus, both A and B should be returned since neither A nor B dominates each other in terms of their distances to park and mall. The skyline query is formalized as follows.

Definition 13 (Dominant Entity) Given two numeric entities u and v in a graph G with the numeric attribute set A, u dominates v, denoted by $u \preceq v$, if (1) for each attribute a_i, $\sigma(u, a_i) \leq \sigma(v, a_i)$; (2) there exists at least one attribute a_j such that $\sigma(u, a_j) < \sigma(v, a_j)$.

Definition 14 (Subgraph Domination) Given two subgraphs G_1 and G_2 of G, G_1 dominates G_2 if (1) G_1 is isomorphic to G_2 via a bijective function f without considering numeric values. (2) $v_i \preceq f(v_i)$ for each numeric entity $v_i \in G_1$. (3) There exists at least one numeric entity $v_j \in G_1$ such that $v_j \prec f(v_j)$.

Definition 15 (Subgraph Skyline) A subgraph $G^* \subseteq G$ is in the subgraph skyline if G^* is isomorphic to the query graph Q and is not dominated by any subgraphs $G' \subseteq G$.

Subgraph skyline query aims to find the subgraph matches that are numerically significant. Zheng et al. [233, 234] address three challenges in answering subgraph skyline query: (1) dynamic skyline computation; (2) efficient querying on graphs; (3) reducing expensive storage cost. In order to tackle these challenges, they partition the data space into grids and compute skyline grid by grid. Lastly, they give an algorithm based on feature encoding to answer the query. Pande et al. [150] study how to identify POIs in a city through subgraph skyline queries, and simplify the problem of finding k skyline subgraphs to identifying k-minimum spanning tree.

Why-not query On many occasions, the users cannot find desirable results by only providing a query graph to existing query processing engines. Why-not query is thus introduced based on the assumption that undesirable results are due to imperfect query inputs. For example, an initial subgraph pattern query submitted by a user may miss some important subgraphs that are expected by the user [89]. Therefore, a why-not query [89] finds a better query graph from the set of graphs returned by the initial query and another set of missing graphs added by the users.

Definition 16 (Why-not Query) Given a collection of graphs $\mathbb{G} = \{G_1, \ldots, G_k, G_{k+1}, \ldots, G_n\}$, where $\mathbb{G}_1 = \{G_1, \ldots, G_k\}$ is retrieved from a query graph Q and $\mathbb{G}_2 = \{G_{k+1}, \ldots, G_n\}$ is the set of missing graphs. A why-not query finds a new query graph Q^* satisfying that $Q^* = \arg\min_{Q'} \lambda_m(Q', \mathbb{G})$ where $\lambda_m(Q', \mathbb{G}) = \max\{\lambda(Q', G_i) | G_i \in \mathbb{G}\}$ and $\lambda(Q', G_i) = |E(Q')| + |E(G_i)| - 2|E(mcs(Q', G_i))|$. $mcs(Q', G_i)$ is the maximum common subgraph of Q' and G_i [22].

Why-not query is first proposed for improving the keyword search experience in relational databases [178]. Islam et al. [89] expand the why-not query to subgraph pattern matching. We note that why-not query requires to compute the maximum common subgraph (MCS), which is NP-hard [221]. To make the query tractable, Islam et al. employ a two-phase approach, i.e., candidate generation and selection, to approximate MCS computation for selecting the alternative query graphs.

Supergraph query Subgraph pattern query is to retrieve subgraphs in the data graphs which match a query graph. In contrast, supergraph query is defined as a reverse of subgraph pattern query. If graph G' is a subgraph of graph G, then G is a supergraph of G'. Hence, supergraph query retrieves all graphs in a database such that the query graph is a supergraph of them. Supergraph query could be used in a number of real-world applications [163, 226]. For example, chemists may want to predict possible properties of a new molecule by identifying particular substructures of it. In this case, the chemists can process a supergraph query using this molecule as query graph on the database of known substructures. Next, we introduce several related works on supergraph query.

Zhang et al. [226] study how to answer supergraph query efficiently. They adopt a graph encoding method called GVCode and propose an index structure GPTree based on GVCode to speed up supergraph query processing. Shang et al. [163] focus on supergraph similarity search. They transform MCS detection into supergraph detection problem. Zhang et al. [227] study the supergraph search problem with a probability threshold. In this setting, each edge in the database has an occurrence probability. The goal is to retrieve all the data graphs that is probabilistic subgraph of the query graph over the given threshold.

Query input assistance Inputting a query graph structure could be complicated and confusing for end users. There are several works proposed to ease this burden by providing user input assistance. Several studies [90, 144] devise solutions for users to input exemplar results as a query to the graph database. Meanwhile, interactive systems are built to facilitate users in exploring a large knowledge graph and forming meaningful queries [45, 88]. Furthermore, due to the evolution of using machine learning techniques for natural language processing (NLP), a plethora of researches have been developed that allow users to input queries in natural language, which will be discussed in Section 8.

4.2 Summary

In this section, we have surveyed the subgraph pattern queries. The literature of this field represents decades of work by researches from diverse areas. Most of the existing work aims to strike a balance between the quality of the matching results and the performance overhead. Moreover, the matching condition between a query pattern and a candidate subgraph only considers topology and attribute values. Thus, there appear to be future research opportunities in the following directions:

- There have been recent surge of attributed graph embedding techniques that map a subgraph structure to low-dimensional vectors [23, 68, 190]. It offers opportunities to provide richer query semantics and better performance over traditional subgraph pattern matching. For example, given one query pattern, one can embed the query pattern into a query vector which is used to find the nearest neighbors as results among vectors that are embedded from a number of subgraphs in the data graph.
- Many applications consider the temporal behavior of a graph pattern, e.g., emails and instant messages in a communication network. For these temporal networks, timestamps are attached to all edges thus it implies the order of events occurs in the network. To extend existing subgraph pattern matching queries, it is rather interesting to match

a query pattern on a temporal network which considers the temporal order of the edges in the query pattern [160, 171].

5 Aggregate query

In this section, we summarize existing studies on aggregate queries in attributed graphs. The objective of aggregate queries is to generate summaries of input graphs to enable multi-perspective analytics with varying granularities. We consider two kinds of aggregate queries: (1) *graph online analytical processing* (Graph OLAP) that generates a summary of the data graph based on a given aggregate condition; (2) *egocentric aggregate query* that computes a summary for the neighborhood subgraph induced by a query vertex.

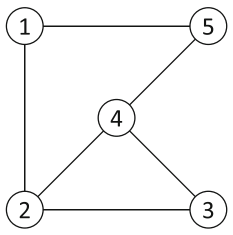
5.1 Graph OLAP

Online analytical processing (OLAP) is a powerful tool for multi-dimensional analysis of attributed graphs. OLAP can help us interactively view and analyze graph data from different perspectives and with multiple granularities. Specifically, graph OLAP returns an *aggregate graph* by grouping the vertices and aggregating the edge weights of the input graph(s) according to user-specified conditions.

Graph OLAP helps users draw insights from attributed graphs for information discovery and decision making. Take the bibliography network as an example, let us consider a graph of coauthor relations between authors. An OLAP operation on vertex attribute ‘affiliation’ generates a summary graph where the authors in the same institution is aggregated into one vertex with the coauthor frequencies as edge weights. In this way, we can provide an overview of the collaborations among institutions. Furthermore, we can further slice the summary graph by edge attribute ‘year’ for analyzing the trend of collaborations over time.

Existing studies generalize the concepts and operations of OLAP in data warehouses to graph data, e.g., *cube* that constructs all possible combinations of aggregate attributes in the graph, *roll-up* that provides an overview of the graph by aggregation, *drill-down* that navigates the graph in more detailed levels of granularity, and *slicing/dicing* that extracts a specific subset of data for analysis.

Chen et al. [25] first introduce online analytical processing (OLAP) on graph data. They propose the graph OLAP framework that defines the multi-dimensional and multi-level analysis on attributed graphs. Furthermore, they categorize OLAP queries into two types, namely *informational OLAP* (I-OLAP) where the aggregation is performed on attributes with the topology of data graph remaining unaffected and *topological OLAP* (T-OLAP) where the topology of data graph is changed by aggregation. Zhao et al. [231] formally propose the *Graph Cube* model to support OLAP queries on attributed graphs. Moreover, they introduce the Crossboid OLAP query that analyzes the relations between users grouped by different attributes. Wang et al. [193] further extend the *Graph Cube* model to the *Hyper Graph Cube* model by aggregating the graphs on both vertex and edge attributes at different granularities. They propose a parallel graph OLAP system, namely, Pagrol, which is implemented based on MapReduce and deployed on Apache Hadoop.



(a) Graph structure

ID	Gender	Profession	Salary
1	male	professor	\$75,000
2	female	doctor	\$300,000
3	male	engineer	\$65,000
4	male	engineer	\$280,000
5	female	professor	\$180,000

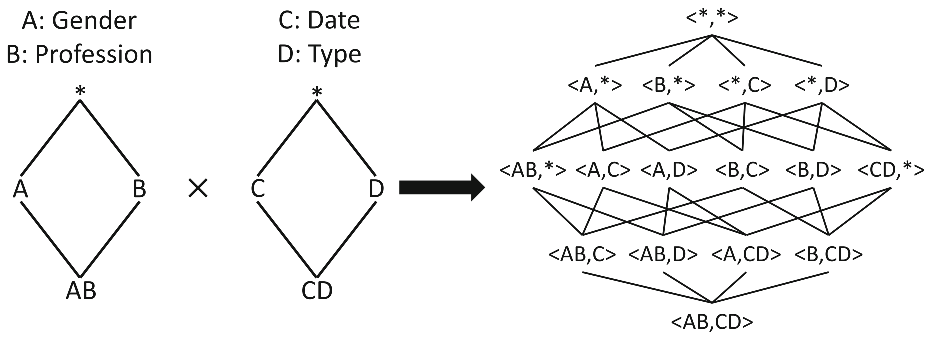
(b) Vertex attribute table

sV	tV	Date	Type
1	2	2008	family
1	5	2011	colleague
2	3	2011	friend
2	4	2011	friend
3	4	2008	friend
4	5	2009	family

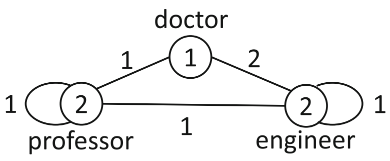
(c) Edge attribute table

Figure 6 An example for aggregate query on a property graph

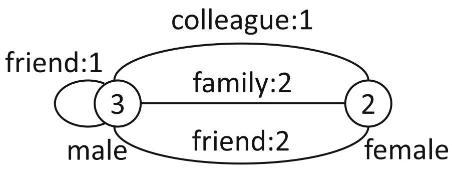
Example 7 Let us consider the property graph in Figure 6. We show the lattice for Hyper Graph Cube is the Cartesian product of the vertex attribute lattice and the edge attribute lattice in Figure 7a. We provide the aggregate graph on attribute ⟨B:Profession⟩ where COUNT(∗) is the aggregate function in Figure 7b. The aggregate graph has 3 vertices for 3 professions in the vertex attribute table. The weight of each vertex is the number of persons with each profession. In addition, the edge weights are the numbers of edges between vertices with different professions. Furthermore, we give an aggregate hyper graph on attribute ⟨A:Gender,D:Type⟩ in Figure 7c. The vertices in the data graph are grouped into two vertices by gender: *male* (3) and *female* (2). Additionally, the edges are grouped into 4 weighted edges by type. For example, "*family:2*" means that there are 2 edges of type "family" between males and females.



(a) Hyper cube lattice



(b) Aggregate graph on $\langle B, *\rangle$



(c) Aggregate hyper graph on $\langle A, D\rangle$

Figure 7 Examples for hyper graph cube and aggregate graphs

The other studies on graph OLAP are listed as follows. Qu et al. [154] introduce two novel techniques named T-Distributiveness and T-Monotonicity for efficient T-OLAP processing. Yin et al. [214] develop the HMGraph OLAP framework for multi-dimensional and multi-labeled graphs. They introduce two operations named *rotate* and *stretch* to analyze the relations between different types of entities. Beheshti et al. [13] extend SPARQL to support multi-dimensional computation for OLAP in attributed graphs and deploy their system on Apache Hadoop.

5.2 Egocentric aggregate query

Graph OLAP focuses on analyzing the properties of the entire graph from the global perspective. In this subsection, we consider the *egocentric aggregate query* which studies the *local* properties of query vertices by analyzing their neighborhood subgraphs. Typically, an egocentric aggregate query is restricted to the query vertex v's h-hop neighborhoods that are reachable from v via a path of length at most h. The objective is to provide the value of an aggregate function within the neighborhood subgraph.

Yan et al. [203] first investigate the egocentric aggregate queries on attributed graphs. Given a function $f(u) \in [0, 1]$ that measures the individual relevance of a vertex u to a given query and an aggregation function $F(u)$ that is defined as the sum/average relevance of the h-hop neighbors of a vertex u to the query, a neighborhood aggregation query aims to find the top-k nodes with the highest scores of $F(u)$.

Mondal and Deshpande [143] and Fan et al. [49] give a general definition of egocentric aggregate query on attributed graphs. Given an aggregate function F and a query vertex v, an egocentric aggregate query returns the aggregate value $F(v)$ that is computed on the h-hop neighborhoods $N_h(v)$ of v. Both of them consider that the definitions could be extended to the dynamic setting where both graph structures and attributes may change over time. Here is an example for egocentric aggregate query.

Example 8 We consider the query vertex 4 in Figure 6 and the aggregate function $F = \mathsf{AVG}(\mathsf{Salary})$. The aggregation is conducted on 1-hop neighborhoods. We can see the 1-hop subgraph induced by vertex 4 contains vertices 2,3,4,5. According to the vertex attribute table, we can compute the result of this egocentric aggregate query is $\mathsf{AVG}(\mathsf{Salary}) = \$206,250$.

Mondal and Deshpande [143] propose an aggregation overlay graph based index that shares pre-computed partial aggregate values among vertices w.r.t. a given aggregate function for egocentric aggregate query processing. Fan et al. [49] propose the Dense Block Index for egocentric aggregate query processing. Both indices are designed for the dynamic setting where both graph topology and attributes may change over time.

5.3 Summary

In this section, we have reviewed two types of aggregate queries on attributed graphs, i.e., graph OLAP and egocentric aggregate query. Both query types aim to summarize an attributed graph for analytics with multiple views. Graph OLAP focuses on the global properties of data graphs whereas egocentric aggregate query discovers the local properties of query vertices.

Independent of the literature on aggregate queries, *graph summarization* [125] has been extensively studied as an important problem in graph mining. Similar to aggregate query, the objective of graph summarization is also to generate a small summary graph of the input graph. The difference is that graph summarization aggregates or simplifies the input graph based on the topology property whereas aggregate query drives summary based on the vertex and edge attributes. Therefore, an interesting direction is to combine the merits of aggregate query and graph summarization: acquiring an overall picture of a graph from the perspectives of both attributes and topology.

6 Similarity search

In this section, we summarize existing studies on similarity search in *heterogeneous information network* (HIN), a special type of attributed graphs where both vertices and edges have multiple labels. Specifically, vertex labels denote the entity types while edge labels represent different types of relationships between entities. For a query vertex v, a similarity search returns the vertices that are highly similar to v. In different scenarios, the returned vertices are either restricted to be the same type as v or a type different from v. In Figure 8, we illustrate a bibliographic network. There are three vertex labels representing authors (circles), papers (squares), and venues (stars). There are four edge labels between vertices: (1) *colleague* relation between two authors; (2) *authored_by* relation between an author and a paper; (3) *cite* relation between two paper; and (4) *published_on* relation between a paper and a venue.

Similarity search is a fundamental problem in real-world applications. For example,

– In social networks, it is used to discover highly similar persons for *friend suggestion* and *link prediction*.
– In e-commerce networks, it finds the proximities between customers and products for *recommendation*.
– In bibliography networks, it evaluates the similarities of authors for *classification* and *community detection*.

The core of similarity search is the *similarity measure* for evaluating the proximity between vertices. For non-attributed graphs, personalized PageRank [92] (PPR) and SimRank [91] are the most common similarity measures. However, they are entirely based on the graph structures: PPR evaluates the probability starting from a query vertex to a target vertex by randomly walking with restart; SimRank evaluates the similarity of two vertices by their neighbors' similarities. They do not consider the case that vertex and edge labels have different semantic meanings. Exploiting the semantic information behind the vertex and edge labels is crucial for similarity evaluation in attributed graphs. Therefore, similarity measures in non-attributed graphs may not be adequate for attributed graphs since the semantics of vertices and edges are ignored. In the following, we will not discuss the similarity measures in non-attributed graphs and focus on the similarity measures in attributed graphs.

Generally, existing methods for similarity search in attributed graphs can be divided into two subcategories according to *similarity measures*: (1) *Path-based approach*: evaluate the similarities based on the label-constrained paths (a.k.a meta-paths) between vertices; and (2) *Graph embedding-based approach*: embed vertices into a low-dimensional vector space and using the similarity measures for vectors to evaluate the vertex similarities.

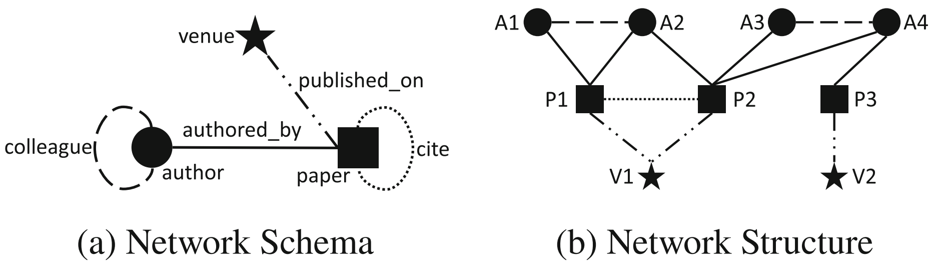


Figure 8 An example for bibliography network

6.1 Path-based similarity search

PathSim Sun et al. [173] propose a seminal work for similarly search in attributed graphs, which is a meta path-based similarity measure called PathSim. PathSim can only evaluate the similarities between vertices with the same label. A meta-path is denoted as $\mathcal{P} = \langle L_{v_0}, L_{e_1}, L_{v_1}, \cdots, L_{e_m}, L_{v_m} \rangle$ where L_{v_i} and L_{e_i} specify the labels of the i-th vertex and edge in the path respectively. We can see two vertices may be connected by multiple meta-paths with different semantic meanings, which leads to different similarity scores.

Example 9 Let us take the bibliography network in Figure 8 as an example. In Figure 9a, we give two examples for the meta-path APA (author-paper-author) that captures the co-author relationships between authors. Here we ignore the edge labels since there is only one type of edges between authors and papers. A path $\langle A1 \rightarrow P1 \rightarrow A2 \rangle$ is an APA path, which means that $A1$ and $A2$ are co-authors on paper $P1$. A meta-path $VPAPV$ (venue-paper-author-paper-venue) identifies the venues with a large number of common authors. Note the edge labels are also omitted because of non-ambiguity. In Figure 9b, we show that a path $\langle V1 \rightarrow P2 \rightarrow A4 \rightarrow P3 \rightarrow V2 \rangle$ is an example for the $VPAPV$ path. It represents two venues $V1$ and $V2$ share the same author $A4$.

Given the notion of meta-paths, PathSim [173] is defined to evaluate the similarity between two vertices of the same type.

Definition 17 (PathSim) Given a symmetric meta-path $\mathcal{P}$, the PathSim between two vertices x, y is defined by:

$$sim(u, v) = \frac{2 \times |\{P_{u \rightsquigarrow v} : P_{u \rightsquigarrow v} \in \mathcal{P}\}|}{|\{P_{u \rightsquigarrow u} : P_{u \rightsquigarrow u} \in \mathcal{P}\}| + |\{P_{v \rightsquigarrow v} : P_{v \rightsquigarrow v} \in \mathcal{P}\}|}$$

where $P_{u \rightsquigarrow v}$ is an instance of $\mathcal{P}$ between u and v, $P_{u \rightsquigarrow u}$ is between u and u, and $P_{v \rightsquigarrow v}$ is between v and v.

Here a meta-path $\mathcal{P}$ is symmetric if $\mathcal{P}$ and its inverse path $\mathcal{P}^{-1}$ are identical.

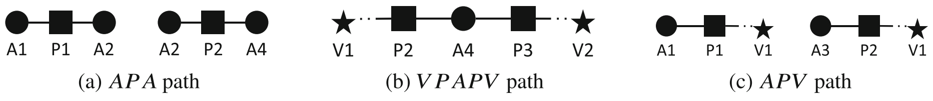


Figure 9 Examples for meta-paths

Example 10 We compute the PathSim score between $A2$ and $A3$ in Figure 8b using the APA path. The number of APA paths between $A2$ and $A3$ is 1 because they collaborate in one paper $P2$. Moreover, the number of APA paths between $A2$ and $A2$ is 2 while the number of APA paths between $A3$ and $A3$ is 1. Therefore, the PathSim score between $A2$ and $A3$ is $sim(A2, A3) = \frac{2\times 1}{2+1} = 0.67$.

PathSim evaluates the similarity by counting the number of a meta-path between two vertices. PathSim has two important properties as a similarity measure: *symmetricity*, i.e., $sim(u, v) = sim(v, u)$, and *self-maximum*, i.e., $sim(u, u) = 1$. Finally, as PathSim scores can be efficiently computed by vector-matrix multiplication, efficient algorithms have been proposed for online top-k PathSim similarity search.

Extensions of PathSim Following the seminal work of Sun et al., there have been a proliferation of studies on improving the effectiveness and efficiency of PathSim.

One drawback of PathSim is that it can only be used for evaluating the similarity of two vertices with the same label. To address this limitation, Shi et al. [164, 165] propose HeteSim to evaluate the similarity between two vertices with different labels. HeteSim is a path-constrained version of SimRank. It calculates the meeting probability of two vertices along a certain meta-path. Moreover, the meta-paths used for HeteSim can be non-symmetric so as to evaluate the similarity of any two vertices. For example, we show an APV (author-paper-venue) meta-path in Figure 9c that can model the similarity between an author and a venue: an author is more relevant to a venue if she publishes more papers on that venue. HeteSim is also semi-metric and has the *symmetric* and *self-maximum* properties. We can utilize existing methods for SimRank computation with meta-path-based pruning to compute the HeteSim scores.

Another limitation of PathSim is that it assumes meta-paths are manually provided by domain experts, which is not impractical for schema-rich networks (e.g., RDF graphs) with thousands of vertex and edge types. Wang et al. [188] propose a relation-based similarity measure called RelSim for schema-rich networks. They ask users to provide example pairs of vertices with high proximities and automatically detect the latent semantic relation (LSR) implied by the query. They devise an optimization model to efficiently learn LSR through linear programming and perform similarity search using RelSim based on the learned LSR.

PathSim and its extensions typically require matrix chain multiplication for similarity evaluation, which brings high computation overheads and scalability problems on massive graphs. To address this issue, Gu et al. [69] propose a random path sampling-based measure called RSSim for scaling up PathSim. They prose a Monte Carlo simulation to approximate RSSim scores between vertices. Zhang et al. [222] propose the Panther$_m$ framework for top-k meta-path similarity search in massive graphs using random sampling. The results of both studies show that their random sampling based approaches could achieve compelling speedups over matrix chain multiplication without significantly affecting the accuracy for the similarity scores.

Many approaches introduce richer semantics into PathSim so as to improve the effectiveness of similarity evaluation. Yu et al. [217] investigate the user-guided similarity search problem. In addition to input a query vertex, the user should provide one or more vertices of the same type as guidance. U et al. [180] introduce supportive information (e.g., the number of citations in bibliography networks) into PathSim. He et al. [77] incorporate transitive similarity and temporal dynamics into PathSim. Meng et al. [141] propose a similarity measure called AvgSim that averages the similarity scores computed by a given meta-path and its reverse path. Wang et al. [187] propose the distant meta-path similarity to capture the semantics between two distant (isolated) vertices, which is evaluated by the meta-path similarities of their neighborhood vertices with the same label.

Other path-based methods Lao and Cohen [104, 105] propose a Path-Constrained Random Walk (PCRW) model for proximity search. PCRW restricts the random walkers to follow a particular sequence of labels away from the query vertices and adopts a supervised learning method to estimate path weights. Shi et al. [167] study path-based relevance measure from a probabilistic perspective. They establish the probabilistic interpretation of existing path-based relevance measures and propose to model cross-meta-path synergy. They propose a novel relevance measure based on a generative model, which is data-driven and tailored for each network, and develop an inference algorithm for the proposed measure. Conte et al. [35] propose a q-gram based similarity measure. For any vertex u, they consider all the simple paths of length less than q landing in u. Each such simple path will give rise to a q-gram by concatenating the labels of vertices along the path. Then, all these q-grams constitutes a multiset $L(u)$ of q-grams for vertex u. To compare the similarity of vertex u and v, they adopt the Bray-Curtis similarity index $BC(u, v)$ defined on $L(u)$ and $L(v)$. Zhang et al. [223] attempt to represent the inter- and intra-type relationships in a unified manner with a general relationship matrix (GRM) indicating the importance of relationship between any two labels. Meng et al. [142] propose a method for diversified top-k similarity search by combining diversification of vertex coverage with a dissimilarity constraint.

From meta-paths to meta-subgraphs Fang et al. [62] propose a meta-subgraph-based approach to semantic proximity search. Meta-subgraphs are more general than meta-paths since it can jointly model multiple common structures between vertices. Each meta-subgraph can be seen as a nonlinear combination of meta-paths, and is thus more expressive. Furthermore, they propose a supervised method that can automatically identify the characteristic meta-subgraph from the examples of query and answer vertices. Finally, they design an efficient meta-subgraph matching algorithm for similarity computation. Huang et al. [87] propose to use meta subgraph to measure the proximity between vertices. A meta-subgraph is a directed acyclic graph (DAG) of vertex labels with edge labels connecting in between. The basic idea for similarity measure is to enumerate the instances of the meta subgraph from the query vertex and the most relevant vertices have higher chances to be covered by the instances. Due to the high computational cost of subgraph matching, they propose efficient indices and matching algorithms for similarity computation. Zhou et al. [235] propose a stratified meta-structure which can be constructed automatically and capture rich semantics. They design a similarity measure called SMSS based on stratified meta-structure.

6.2 Graph embedding-based similarity search

Graph embedding is one of the most popular areas for graph representation learning. It encodes the graph data into a low dimensional space that maximally preserves the graph topology as well as the attribute information. In recent years, with the advent of deep neural networks, there have been a surge of interest in graph embedding and its applications. As our focus is *graph query*, we do not discuss the approaches to *graph embedding* here. Interested readers can refer to [23, 68, 190] for extensive surveys of graph embedding techniques. Next, we will introduce the relationship of graph embedding and similarity search.

Graph vertex embedding is an *indirect* approach to evaluating the similarities of vertices [23]. In Figure 10, we illustrate how graph vertex embedding works: each vertex of the input graph is mapped to a vector in the embedding space and vertices with higher proximities are closer to each other in the embedding space. In this way, we can evaluate the similarity of two vertices by the distance between their embedding vectors, e.g., the Euclidean distance and other similarity measures for vectors.

Embedding-based similarity search can *implicitly* encode graph structures and thus lacks an *explicit* explanation of why two vertices are similar in the embedding space. Moreover, most of the embedding approaches are designed for downstream machine learning tasks on graphs, e.g., link prediction, vertex classification, knowledge completion, and so on. Recently, there are a few attempts on *proximity embedding* [126–129] that *directly* embeds the connections between two vertices into vectors to compute the similarity score. Compared with indirect embedding approaches, proximity embedding could achieve better accuracies on evaluating the similarities between vertices.

6.3 Summary

In this section, we have surveyed existing literature on similarity search for attributed graphs. Recent years have witnessed the success of path-based approaches to similarity search. Despite the prevalence of path-based approaches, they suffer from three drawbacks. First, most of these approaches ask users to provide explicit path patterns (e.g., meta-paths) for similarity search. Second, paths may be insufficient to capture the rich semantics between vertices. Third, the semantic meanings are merely represented by simple vertex and edge labels but richer and multi-dimensional attribute information is not considered yet. Despite the efforts to address these drawbacks [62, 87, 188], it still requires future studies to improve the effectiveness of path-based approaches from the above three aspects.

With the advent of deep learning techniques, graph embedding has attracted emerging research interest recently. However, how graph embedding can be directly used for similarity search is still largely unexplored. Furthermore, existing embedding methods require heavy computation and thus cannot support online query processing. Therefore, another possible direction is to improve the efficiency of graph embedding techniques so that they can be applied to similarity search for online query processing. Finally, embedding-based approaches suffer from the lack of human interpretable explanation. It is a promising direction to explore new graph embedding approaches that can explain the proximity relations between vertices in the embedding space.

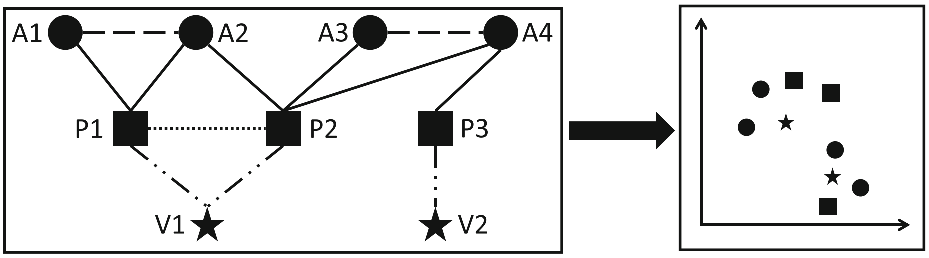


Figure 10 An illustration of graph vertex embedding

7 Keyword search

Keyword search over attributed graphs is a user-friendly and flexible query mechanism, powered by the integration of database (DB) and information retrieval (IR) techniques. It has various real-world applications, some representatives of which are presented as follows.

– **Keyword search on relational databases (RDB):** Structured query language (SQL) is a classical and powerful tool for querying RDB. However, it is not easy for users to formulate SQL queries due to complex SQL syntax and the requirement of a thorough understanding on database schema. To alleviate the problem, keyword search over RDB is extensively studied in the last decades [1, 6, 30, 48, 110, 121, 133, 134]. These approaches formalize RDB as an attributed graph and return tuples, which are typically joined from multiple relations, as results to a keyword search. As there may exist many tuples relevant to the query, some IR-style scoring functions are applied to rank the results.
– **Keyword search on knowledge graph (RDF):** Resource Description Framework (RDF) is initially introduced for conceptual description of Web information in a *triple* form (sub, pred, obj). Recently, RDF has attracted much interest from both academic and industrial communities due to the prevalence of large-scale knowledge bases, such as DBPedia [3], Yago [172], Freebase [17], and Google Knowledge Graph[4]. RDF data can be naturally formalized as attributed graphs. Many approaches are proposed for keyword queries on RDF-based attributed graphs [42, 56, 72, 108, 146, 179, 207].
– **Keyword search on other graph-structured data:** There are studies on keyword search over XML documents [9, 70, 201] (tree-structured data), uncertain graphs [219], and other data types [93, 152, 196, 218, 237]. As they are less relevant to attributed graphs, we will not discuss them and encourage the interested readers to refer to other surveys [107, 189].

There are existing survey papers on keyword search over relational databases [216], XML data [107] and graph data [189]. However, they focus on specialized context, such as relational and graph data, while we review the existing works in the perspective of the more general attributed graph. Moreover, we focus on reviewing how the approaches formalize *query semantics* as substructures of the attributed graph. Based on this, we classify existing methods into two categories, namely *tree-based query semantics* (Section 7.1) and *subgraph-based query semantics* (Section 7.2).

7.1 Tree-based query semantics

In tree-based query semantics, an answer to a keyword search is defined by a tree extracted from the attributed graph with respect to the query. Formally, given an attributed graph $G = (V, E, \Sigma, L)$ and a set $Q = \{q_1, q_2, \ldots, q_l\}$ of l keywords, an answer is defined as a subtree T of G where the vertices in T contain all or a portion of keywords in Q.

7.1.1 Minimal total tree based semantics

The pioneer literature on keyword search over RDB, such as DBXplorer [1] and Discover [84], considers the *minimal total tree* based semantics defined as follows.

[4]https://developers.google.com/knowledge-graph/

Definition 18 (Minimal Total Tree Semantics) Given an attributed graph G and a query Q, a minimal total tree T^{M} is defined as a subtree of G satisfying the following two conditions: (1) *total*: each keyword $q_i \in Q$ must be contained in at least one vertex of T^{M}; and (2) *minimal*: any subtree of T^{M} is not total.

Based on this semantics, DBXplorer [1] and Discover [84] introduce the *minimal total joining network of tuples* (MTJNT) to model an answer to query Q.

Example 11 Figure 11 illustrates an example database where Figure 11a and 11b show the schema and underlying tuples, respectively. Figure 11c illustrates the corresponding attributed graph. Given two keywords "john, search", the subtree $\langle a_1 - p_1 \rangle$ is an MTJNT, while $\langle a_1 - p_1 - p_6 \rangle$ is not because it does not satisfy the *minimal* condition.

Ranking issues of the MTJNTs are discussed in [83, 121, 133, 134] by introducing various scoring functions, which can be classified into the following two categories.

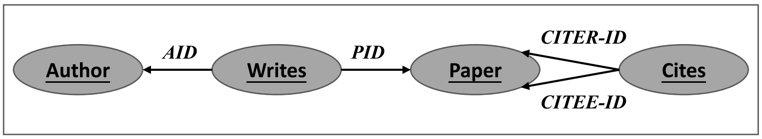
1. *IR-style scoring functions:* These functions assign a score $score(T^{\mathrm{M}}, Q)$ to an MTJNT T^{M} w.r.t. a query Q. To compute the score, some approaches [83, 121] consider each vertex v in T^{M} individually and compute $score(T^{\mathrm{M}}, Q) = \sum_{v \in T^{\mathrm{M}}} score(v, Q)$. From this equation, we can see that the essential issue is how to capture the *relevance* of a vertex v with respect to the keywords in Q. This issue can be addressed by using IR-style scoring schemes, such as the classic TF-IDF scheme. In the TF-IDF scheme, TF captures the number of times each keyword $q \in Q$ appears in the keyword (label) set of vertex v (which could be further normalized by the size of keyword set), and IDF of q is the inverse of the number of times q appears in all vertices in G. Moreover, other approaches [133, 134] consider vertices in T^{M} jointly by considering each tree T^{M} as a *virtual document* and computing $score(T^{\mathrm{M}}, Q)$ as the relevance between Q and the virtual document rather than individual tuples.
2. *Structural scoring functions:* The intuition of structural scoring function is to prefer a tree with more *compact* structure. A straightforward way is to use the size of T^{M}, i.e., the number of vertices in T^{M}, to penalize large trees. However, as it will undesirably affect moderate-size trees as well, Luo et al. [133, 134] introduce a more sophisticated scheme for size penalization.

To compute all the MTJNTs, existing approaches [1, 83, 84, 121, 133, 134] reply on the schema graph in Figure 11a. They formalize candidate network (CN) as a minimal total tree over the schema graph, e.g., Paper - Writes - Author, and generate a *complete* and *duplication-free* set of CNs with size not exceeding a pre-defined threshold. Then, they evaluate the generated CNs by issuing the corresponding SQL queries to obtain the MTJNTs.

7.1.2 Steiner tree based semantics

The query semantics discussed in the previous subsection is confined by the *schema* of graph G, which may not be applied in more general *schema-free* graphs. Therefore, many approaches [14, 43, 95, 97, 108, 116, 168, 209] are proposed for supporting keyword search over graph, and consider a *Steiner tree* based query semantics.

Definition 19 (Steiner Tree Semantics) Given an attributed graph G and a query Q, let S_i denote the set of vertices containing keyword $q_i \in Q$. An answer to Q is a rooted tree



(a) Database schema

Author

AID	*NAME*
a1	John
a2	Tom
a3	Jim
a4	Lucy

Cites

CITER-ID	*CITEE-ID*
p3	p1
p6	p1
p6	p5
p5	p2

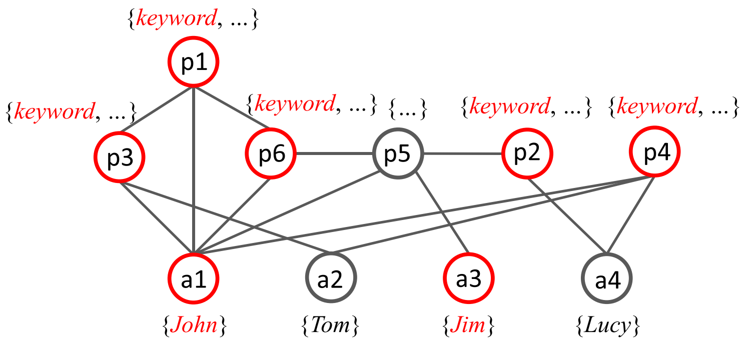
Writes

PID	*AID*
p1	a1
p2	a1
p3	a1
p5	a1
p6	a1
p3	a2
p4	a2
p5	a3
p2	a4
p4	a4

Paper

PID	*TITLE*	*BOOKTITLE*
p1	Keyword Search over DB	DB Systems
p2	NLP Techniques in Keyword Search	IR Research
p3	Keyword Query on Knowledge Graph	DB Systems
p4	IR Keyword Query over Graph Data	DB & IR
p5	QA in Knowledge Bases	IR Research
p6	Keyword Search over RDF Data	DB Systems

(b) Database tuples



(c) Data graph

Figure 11 An example for keyword search over relational databases

T^{S} that contains at least one node from each S_i, and the weight $w(T^{\mathrm{S}})$ is $\sum_{e \in E(T^{\mathrm{S}})} w(e)$, where $E(T^{\mathrm{S}})$ is the edge set of the tree and $w(e)$ is the weight of each edge e in the tree[5].

[5]Note that the original papers also consider the edge direction, e.g., an edge in database graph G_{DB} with a direction from foreign key to primary key. For ease of presentation, we omit the edge direction in this subsection.

Under this semantics, the problem of finding an optimal tree T^{S} with the smallest weight $w(T^{\mathrm{S}})$ is the classic *minimum Steiner tree* problem, which is proved as NP-complete. Existing approaches have proposed various weighting schemes. The seminal works BANKS-I [14] and BANKS-II [95] measure both node and edge weights.

– **Edge weights:** BANKS-I [14] and BANKS-II [95] associate a *pre-defined* weight to each edge e in the graph G. Thus, given an answer tree T^{S}, they define a score $s(T^{\mathrm{S}}, q_i)$ as the sum of edge weights on the path from root of T^{S} to the leaf containing keyword q_i, and then compute an aggregated score $s_E = \sum_{q_i \in Q} s(T^{\mathrm{S}}, q_i)$.
– **Node weights:** Node weights are introduced to capture node prestige, and several random walk-based techniques, such as PageRank can be applied. Then, an aggregated score s_N is defined as the weight summation of leaf nodes and answer root of T^{S}.

By combining both edge and node weights, BANKS-I [14] and BANKS-II [95] introduce score $s(T) = s_E \cdot s_N^{\lambda}$ where λ is a tuning parameter to balance importance of edge and node scores, which is usually set to 0.2.

Example 12 Consider three keywords {keyword, john, jim} over the attributed graph shown in Figure 11c, where vertices that contain keywords are highlighted. It is not difficult to find vertex p_5 that has the shortest path to one vertex in each S_i and thus forms an answer tree. The semantics of the tree is "a john's paper containing keyword that cites a jim's paper". Given weighting schemes discussed previously, a score $s(T)$ can be measured by considering weights of both edges and nodes.

There have been many approaches [14, 43, 95, 97, 108, 116, 168, 209] proposed for efficient minimum Steiner tree computation. The classic algorithm is *backward search* in BANKS-I [14]. The basic idea is to start from the "keyword vertices", i.e., $S = \bigcup_{q_i \in Q} S_i$. Then, it runs $|S|$ instances of the Dijkstra's shortest path algorithm starting from each vertex in S in a *reverse* direction. The goal is to find the common vertices which have a shortest path to at least one vertex in each S_i. Answer trees can then be formed with the common vertices as roots and keyword vertices as leaves. One drawback of backward search is that it would explore an "unnecessarily" large number of vertices, and thus results in high exploration overheads. Considering our example query {keyword, john, jim}, keyword is popular that occurs in many vertices and a_1 is a "hub" vertex that connects many other vertices. Therefore, backward search may end up visiting nearly all the vertex in the graph. Obviously, a better exploration strategy is to start from "less popular" vertex a_3. Then, after explored p_5, we start a *forward exploration* from p_5 and can quickly find an answer tree. Based on this intuition, BANKS-II [95] improves backward search and introduces *bidirectional search* exploration. BLINKS [76] exploits distance precomputation and graph partitioning to make bidirectional search more efficient. Graph summarization is further introduced for search space pruning in [108]. Dynamic programming approaches [43, 116] are proposed for exact minimum Steiner tree computation. Although they achieve good performance empirically, they may suffer from exponential running time in the worst case. Recently, parallelization [209] and hub labeling [168] are also considered for minimum Steiner tree computation.

7.2 Subgraph-based query semantics

In the subgraph-based semantics, an answer to a keyword search is defined as a (possibly cyclic) subgraph extracted from the data graph G that contains all or a subset of query keywords. This section reviews two subgraph-based semantics, r-radius Steiner graph and community-based semantics. Lastly, we discuss query graph assembly approaches that return query graphs instead of subgraphs of G.

7.2.1 r-Radius Steiner graph based semantics

Li et al. [110] propose an r-radius Steiner graph-based semantics w.r.t. a query Q. The definition of r-radius Steiner graph is based on the notions of *centric distance* and *radius*. Given a graph G and a vertex v in G, the centric distance of v in G is the maximum of the shortest distances from v to any vertices in G, and the radius of G is defined as the minimum of the centric distances of all the vertices in G. A graph G is an r-radius graph whose radius is *exactly* r. The r-radius Steiner graph is formally defined as below.

Definition 20 (r-Radius Steiner Graph) Given a query Q and a r-radius graph $G^{\mathtt{r}}$, a vertex v in $G^{\mathtt{r}}$ is called a content node if v directly contains the keywords in Q, while vertex s is called a Steiner node if there exists two content nodes u and v such that s on the simple path between u and v. An r-radius Steiner graph $G^{\mathtt{S}}$ is defined as a subgraph of $G^{\mathtt{r}}$ that contains the Steiner nodes and associated edges.

Li et al. [110] consider both IR-style and structural compactness-based scoring functions. The IR-style function utilizes the TF-IDF scheme that measures keywords in Q and an r-radius Steiner graph $G^{\mathtt{S}}$, while the structural compactness-based function prefers a $G^{\mathtt{S}}$ where content nodes are closely related in $G^{\mathtt{S}}$ with shorter paths.

Example 13 Considering the example graph in Figure 11c, the centric distance of a_1 is 3, which corresponds to the path to p_4. By considering centric distances of other vertices, we know that radius of the graph is $r = 3$. Then, given two keywords "rdf, john", we can find a_1 and p_6 as content nodes, and p_5 is a Steiner node as it exists in a simple path between a_1 and p_6. Thus, the subgraph containing a_1, p_5 and p_6 and their associated edges is a r-Radius Steiner Graph. On the contrary, a Steiner tree semantics discussed previously only takes $\langle a_1 - p_6 \rangle$ as the result, which conveys less information of the r-radius Steiner graph.

To efficiently compute r-radius Steiner graphs for a keyword search, Li et al. [110] propose an approach that consists of both offline and online stages. In the offline stage, the approach extracts r-radius graphs from graph G by using the adjacency matrix of G, and constructs an inverted index that maps each keyword q to the set of r-radius graphs containing q. As there may be a huge number of r-radius graphs and these graphs may overlap with each other, it only considers the *maximal r-radius graph* which is not a subgraph of other r-radius graphs. Given an online query Q, it first uses the constructed index to obtain all the r-radius graphs that contain all or a portion of keywords. Then, for each of such r-radius graphs, the approach generates an r-radius Steiner graph by removing the non-Steiner nodes. Finally, it ranks the generated r-radius Steiner graphs. To handle large graphs, Li et al. [110] also propose a graph partition algorithm that clusters the r-radius graphs and partitions the whole graph based on the clusters.

7.2.2 Community based semantics

The r-radius Steiner graph semantics discussed in the previous subsection does not consider the internal denseness of the returned subgraph, which is an important property under many circumstances, such as epidemic prevention and information spreading [31]. To this end, motivated by the community search over non-attributed graphs [194] (see [60] for a survey), another line of works discusses the community based semantics for keyword search in attributed graphs. Various models for attributed community have been proposed.

Kargar et al. [96, 98] use the r-clique to model the attributed communities. Vertices within an r-clique are densely connected with each other, and are associated with query keywords. Formally, r-cliques are defined as below.

Definition 21 (r-Clique Semantics) Given an attributed graph G and a query Q, a vertex v is called a *content node* if v contains at least one keyword in Q. An r-clique, denoted by G^{C} is a set of content nodes that together cover all keywords in Q while the shortest distance between any content nodes is no larger than r.

To effectively rank r-cliques, a weighting scheme that considers the pairwise similarity between any two vertices in an r-clique is proposed. Formally, let $\{v_1, v_2, \ldots, v_h\}$ denote the set of nodes in an r-clique G^{C}. The weight of G^{C} is calculated as $\sum_{i=1}^{h}\sum_{j=i+1}^{h} dist(v_i, v_j)$ where $dist(v_i, v_j)$ is the shortest distance between nodes v_i and v_j. Theoretically, Kargar et al. [96] prove that finding an r-clique with the smallest weight is NP-hard. They first introduce an exact brand-and-bound algorithm for this problem. Since the exact algorithm is slow when the number of keywords is large, they propose a 2-approximation algorithm to find r-cliques in polynomial time. They further propose an $(l-1)$-approximation algorithm that runs l times faster than the 2-approximation algorithm, where l is the number of keywords, while achieving similar quality of results. In [27], two heuristic approaches are proposed to find r-cliques for multiple keyword queries in a batch manner. Furthermore, an extension of clique-based communities in attributed graphs is considered in [112]. Li et al. [112] propose the notion of *keyword-based correlated networks* that extend the r-clique semantics to discover the correlations between different cliques for a set of keywords.

The clique-based community models can return densely internal connected communities w.r.t. query keywords, a strong constraint is imposed to the distance between any two vertices within the result, thus clique-based models tend to find very small communities and miss interesting communities with relatively large size. To this end, some other models that impose more relaxed constraints are proposed.

The d-CORE based community model are proposed by Fang et al. [57, 59], which is formally defined as below.

Definition 22 (d-CORE Semantics) Given a data graph G, a positive integer d, a query node v, and a set of keywords Q, a d-CORE $G^{\mathtt{core}}$ is a subgraph of G satisfying the following properties: (1) $G^{\mathtt{core}}$ is connected and contains v; (2) for each node u of $G^{\mathtt{core}}$, the degree of u in $G^{\mathtt{core}}$ is at least d; (3) the size of $L(G^{\mathtt{core}}, Q)$ is maximal where $L(G^{\mathtt{core}}, Q) = \bigcap_{u \in G^{\mathtt{core}}}(L(u) \cap Q)$ is the set of keywords in Q shared by all nodes in $G^{\mathtt{core}}$.

The community search problem based on d-CORE requires to enumerate all the d-COREs of G. In [57, 59], Fang et al. propose a Core Label tree (CL-tree) index that organizes d-COREs and keywords into a tree structure for efficient online community search. Very recently, core-based community models have been considered for heterogeneous information networks [61], where both vertices and edges are associated with attributes.

It is easy to see that no explicit constraint is imposed to the distance between vertices within a d-CORE, and for any d, the distance between vertices can be any large. Thus, the d-CORE model tends to find communities with large sizes. However, this could lead to another issue called the free-rider effect [198] and include undesired vertices in the result community.

To this end, other models based on d-TRUSS is proposed. The d-TRUSS ensures the internal denseness through high order structures, i.e. triangles, in the network and can impose a relaxed constraint on the distance between vertices. The formally definition is presented as below.

Definition 23 (d-TRUSS Semantics) Given a data graph G, and a positive integer d, a d-TRUSS is a subgraph $G^{\mathtt{truss}}$ of G satisfies the following properties: (1) $G^{\mathtt{truss}}$ is connected; (2) for each edge $e = (v_1, v_2)$ in $G^{\mathtt{truss}}$, e is endorsed by at least $d - 2$ common neighbors of v_1 and v_2.

The diameter of a connected d-TRUSS with n vertices is bounded by $\left\lfloor \frac{2n-2}{d} \right\rfloor$. Based on the concept of d-TRUSS, Huang and Lakshmanan [86] and Zhu et al. [239, 240] propose the (d, r)-TRUSS and minimum dense truss based community semantics for keyword search on attributed graphs, respectively.

Definition 24 ((d, r)-TRUSS Semantics) Given a data graph G, positive integers d, r, a set of query nodes V_q, and a set of keywords Q, a (d, r)-TRUSS $G^{\mathtt{truss+}}$ is a subgraph of G satisfies the following properties: (1) $G^{\mathtt{truss+}}$ is a d-TRUSS containing all nodes in V_q; (2) for each vertex v in $G^{\mathtt{truss+}}$, the maximum length of a shortest path from v to any query node $v_q \in V_q$ is at most r; (3) $score(G^{\mathtt{truss+}}, Q)$ is maximal among all subgraphs satisfying Properties (1) and (2), where $score(G^{\mathtt{truss+}}, Q) = \sum_{q \in Q} \frac{|V_q \cap V(G^{\mathtt{truss+}})|^2}{V(G^{\mathtt{truss+}})}$ and V_q is the set of nodes in G containing keyword q.

Definition 25 (MINIMUM DENSE TRUSS Semantics) Given a data graph G and a set of keywords Q, the MINIMUM DENSE TRUSS $G^{\mathtt{truss*}}$ is a subgraph of G satisfies the following properties: (1) For each keyword $q \in Q$, there exists a vertex $v \in V(G^{\mathtt{truss*}})$ such that $q \in l(v)$; (2) $G^{\mathtt{truss*}}$ is the d-TRUSS such that d is maximized among all subgraphs satisfying Property (1) and $|V(G^{\mathtt{truss*}})|$ is minimized.

Theoretically, the problems of finding (d, r)-TRUSS and minimum dense truss are both NP-hard [86]. Huang and Lakshmanan [86] propose an index that integrates *structural trussness* with *attributed trussness*, together with greedy heuristic algorithms, to compute an approximate (d, r)-TRUSS efficiently. Zhu et al. [239, 240] propose an hybrid index together with a top-down search algorithm that can return an exact MINIMUM DENSE TRUSS. But the proposed algorithm could run in exponential time. Very recently, truss-based community model is extended to directed graphs [123]. Since the problem is still NP-hard, an index based on truss decomposition together with efficient 2-approximation algorithms are proposed for community search on directed graphs.

All the models discussed above impose strict topological constraint on the community and will miss interesting results with more complex structure. Recently, Chen et al. [26] propose the concept of *contextual community*. Instead of ensuring the structure cohesiveness through strict topological constraint, the contextual community is the subgraph that maximizes the contextual density. Formally, the *contextual community* is defined as below.

Definition 26 (CONTEXTUAL COMMUNITY Semantics) Given a data graph G and a set of keywords Q, the contextual community $G^{\texttt{ctx}}$ is the connected subgraph of G that maximizes the score function $\rho(G^{\texttt{ctx}}) = \frac{\sum_{\Delta \in \text{Tri}(G^{\texttt{ctx}})} TS(\Delta, Q) + \sum_{e \in E(G^{\texttt{ctx}})} ES(e, Q)}{|V(G^{\texttt{ctx}})|}$ where $\text{Tri}(G^{\texttt{ctx}})$ is the set of triangles in $G^{\texttt{ctx}}$ and $E(G^{\texttt{ctx}})$ is the set of edges in $G^{\texttt{ctx}}$. The edge contextual score for an edge $e = (u, v)$ is defined as $ES(e, Q) = |Q \cap l(u)| + |Q \cap l(v)|$, and the triangle contextual score for an triangle $\Delta = (u, v, w)$ is defined as $TS(\Delta, Q) = \sum_{e \in \{(u,v),(u,w),(v,w)\}} ES(e, Q)$.

To obtain contextual communities efficiently, Chen et al. [26] first propose an exact algorithm based on an iterative optimization framework and then propose a $\frac{1}{3}$-approximation algorithm to further improve the efficiency.

In addition to the aforementioned works, other related studies focus on community search over graphs with different types of attributes. For example, [28, 58] deal with spatial graphs where each vertex is associated with a location. The attributed community search over spatial graphs finds the community that is densely connected in terms of network topology, and vertices within the community should also be spatially close to each other. Liu et al. [124] further propose a vertex-centric attributed community that takes into account both spatial information and keywords associated with vertices. Influential community search [16, 113, 115] has also been studied. It uses the influence of each vertex as the attribute and its goal is to find communities with high outer influence, which is useful for applications such as advertisement placement. Temporal community search, which aims to find communities that are both topologically and temporally cohesive, is studied in [117]. A combination of community search with skyline search on multi-valued graphs is studied in [114]. It aims to find communities that are not dominated by any other community in terms of multi-dimensional numerical attributes. Nevertheless, these works are less relevant to keyword search on attributed graphs, and thus we do not elaborate on them in this survey. Interested readers may refer to [60] for more details.

7.2.3 Query graph assembly

Most approaches discussed so far directly return subgraphs containing keywords as answers. On the contrary, some approaches [48, 72, 108, 179, 207] focus on assembling *query graphs* as intermediate results. Then, they interpret each query graph as a structured query (e.g., SPARQL for RDF data and SQL for RDB) to retrieve the final results. Note that if no structural query language is available for the attributed graph G, the query graph can retrieve the answers in G using subgraph pattern matching techniques.

Tran et al. [179] propose a query graph assembly approach over the RDF data. Given an RDF-based knowledge graph as illustrated in Figure 12a, the approach first extracts a *summary graph*, which is essentially the schema of the whole graph. Given a keyword query, the approach "augments" keywords to the summary graph.

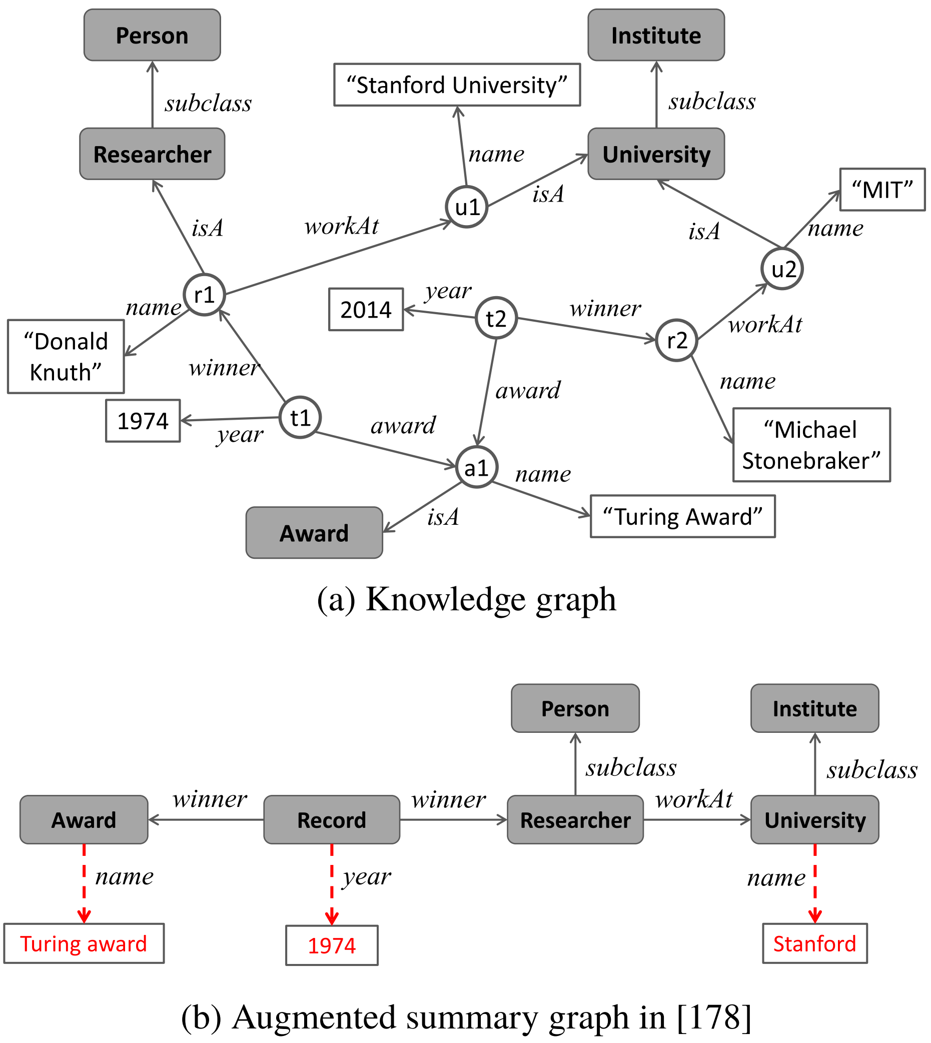


Figure 12 An example for keyword search over knowledge graphs

Example 14 Figure 12a shows an example of RDF-based knowledge graph, where each vertex represents an *entity* (e.g., r_1 and u_1), a *type* (e.g., Researcher and Institute) or a *literal* (e.g., "Donald Knuth" and "MIT"), and each edge represents a triple in RDF, say (r_1, *name*, "Donald Knuth"). Figure 12b shows such an example for summary graph with the *type vertices* (in gray color) as well as their associated edges. Given an example query "turing, stanford, 1974", the keywords turing, stanford and 1974 are augmented to the name of AWARD, the name of UNIVERSITY and the year of AWARDRECORD respectively.

It is not difficult to see that an augmented summary graph corresponds to a query graph that can be interpreted to a SPARQL query for retrieving the final results. Tran et al. [179] introduce the scoring functions to quantify the goodness of augmented summary graphs that considers path length, popularity and keyword matching scores. Moreover, for efficiently generating query graphs, Tran et al. [179] devise index structures that map keywords to type vertices and develop graph exploration algorithms using the aforementioned backward and bidirectional search to generate top-k query graphs.

Some follow-ups, such as [72, 108, 207] propose more sophisticated techniques, such as graph summarization [108], path pattern index [207] and graph embedding based scoring functions [72]. Fan et al. [48] propose an interactive query graph generation approach, called SQL Suggestion for RDB. The approach is user-friendly and can suggest SQL queries efficiency as users type in keywords. It can support a broad range of SQL queries, including aggregations, grouping and ranges.

7.3 Summary

In this section, we have surveyed a wide range of techniques for keyword query on attributed graphs. Most of the existing works aim to extract the *query semantics* behind the simple query keywords, and focus on the following aspects.

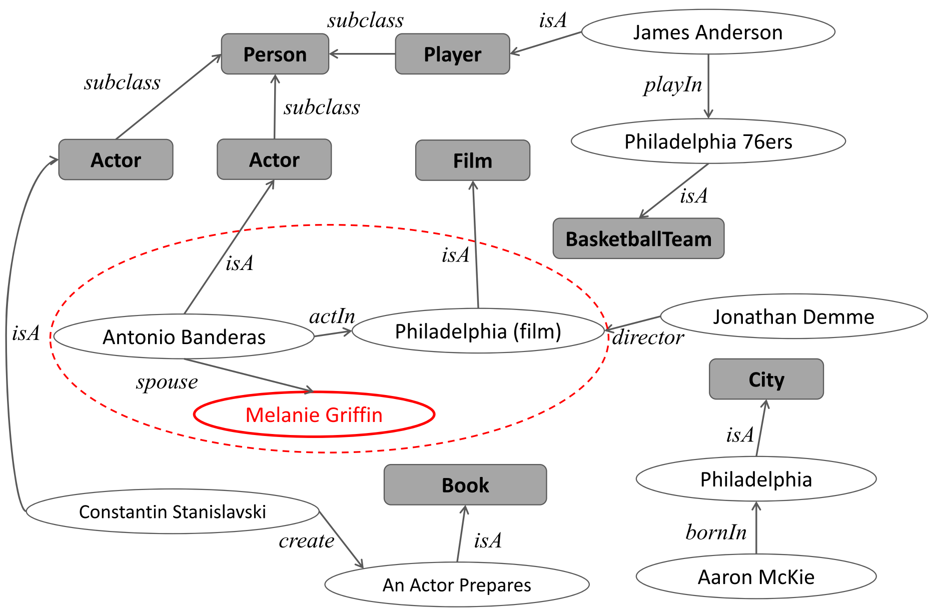
– **Answer formalization:** Existing methods define various structures to formalize an answer w.r.t. to a keyword query, such as minimal total tree, Steiner tree, r-radius Steiner graph, r-Clique, d-CORE, and (d, r)-TRUSS. Answer formalization is a core component in keyword query over attributed graphs.
– **Ranking function:** As a keyword query may be relevant to many substructures, ranking functions are extensively studied. Most of the existing methods borrow ideas from the IR community to apply scoring functions, such as TF-IDF and more sophisticated schemes.
– **Top-k algorithm:** To give online gratification to users, existing methods study efficient top-k algorithms. This is a very challenging task due to the complex answer structures and ranking functions. Existing works focus on devising effective index structures and bound estimation techniques to quickly compute top-k answers.

By applying the above techniques, most keyword query approaches may focus on effective mechanism for balancing the trade-off between *usability* and *expressiveness*. Thus, there appears to be future research opportunities in the following directions. First, many deep learning models have been adopted in the IR community, and have shown superiority over traditional scoring functions. Thus, it may be promising to borrow these models for better capturing relevance between keywords and answer structures. Second, most of the existing works do not consider the feedback from end-users. Thus, it calls for an *interactive* mechanism to ask users to help refine the results. Third, as attributed graphs become increasingly large, many memory-based solutions need to be re-designed.

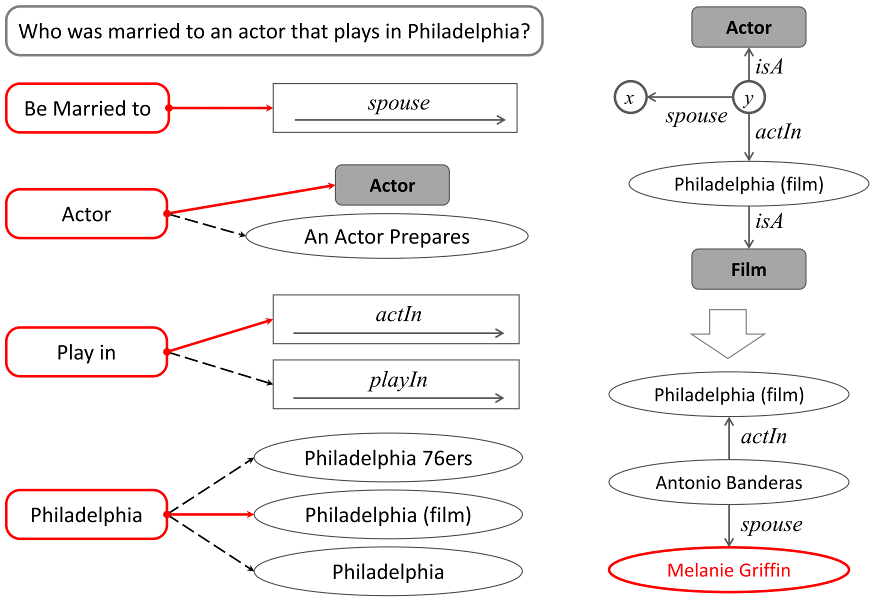
8 Natural language question answering

Recently, *natural language question answering* (NLQA) over attributed graphs has attracted much attention. Unlike keyword search in the previous section, NLQA takes a natural language question as the input, and aims at finding *exact answers* to the question from an attributed graph. NLQA provides a more user-friendly mechanism than keyword search.

Example 15 Consider an example attributed graph shown in Figure 13a, which is essentially a knowledge-based RDF graph as discussed in Section 7. Given a natural language question



(a) Knowledge graph



(b) Phrase mapping and disambiguation (c) Query graph

Figure 13 An example of natural language question answering over knowledge graphs

"who was married to an actor that plays in Philadelphia?", NLQA directly finds "Melanie Griffin" as an answer from the graph to give user gratification.

Although easy to use, NLQA is much more challenging than keyword query due to the complexities of natural language questions. First, the *phrases* in a natural language question may be very ambiguous that correspond to various semantics in an attributed graph. For example, Figure 13b illustrates the potential mappings from a phrase in our example question to our example attributed graph. We can see that "actor" could either be a conceptual type ACTOR or indicates a BOOK instance "*An Actor Prepares*". The phrase "play in" may indicate various edge labels, such as *actIn* or *playIn*. The noun "Philadelphia" may be either a Film, a BASKETBALLTEAM or a CITY. Second, unlike keyword query, the phrases in a natural language question have strong correlation with each other. For instance, given that "actor" is mapped a type ACTOR, the verb phrase "play in" is more like to be associated with edge label *actIn* rather than *playIn*, although the latter is more similar to the phrase literally. Third, the structure of the attributed graph is usually complex and schema-less, thus making the inference over attributed graph very challenging.

This section reviews the core techniques to address these challenges. We note that there exist extensive studies on NLQA [7, 41, 102, 213]. We focus on the *factoid question answering* over attributed graphs, which is formally defined as follows.

Definition 27 (Factoid Question Answering) Given an attributed graph G and a natural language question Q, factoid question answering finds *subgraphs* in graph G, each of which matches the semantics of Q.

For example, the subgraph annotated with the red circle in Figure 13a is an answer to our example question "who was married to an actor that plays in Philadelphia?".

We broadly classify existing approaches to factoid question answering into two categories. The first category *query graph construction* approaches [3, 131, 147, 206, 242] focus on "translating" the natural language question into a *structural query*. Then, the structural query can be executed over the attributed graph to retrieve the final answers. See Section 8.1 for more details. The second category is *end-to-end question answering* approaches [29, 44, 211] that develop deep learning models that directly extract subgraphs from G without taking query graph construction as an intermediate step.

8.1 Query graph construction

The basic idea of query graph construction is to interpret the natural language question into a *query graph* that corresponds to a pattern against the data graph G. Then, the query graph can be used to retrieve all subgraphs that match the corresponding pattern as answers. For example, Figure 13c illustrates a query graph interpreted from our example natural language question. We can see that it is essentially a *logic form* that assigns variables, such as x and y, to a subgraph of G. Note that this section focuses on the techniques for query graph construction. The step of using query graph to retrieve answers can be implemented by mapping the query graph into a SPARQL query or resorting to the subgraph pattern query techniques in Section 4. We first review the approaches to general query construction in Section 8.1.1, and then discuss triple query construction in Section 8.1.2.

8.1.1 General query graph construction

In general, given a natural language question Q, existing techniques for query graph construction apply a pipeline consisting of the following three stages.

- **Phrase Mapping** extracts phrases (sequences of tokens) from question Q that are potentially mapped to the *semantic items*, i.e., vertex and edge labels, in graph G.
- **Disambiguation** chooses the most likely semantic item for each phrase from the candidate space.
- **Query Graph Construction** creates a query graph containing all the mapped and disambiguated semantic items. The query graph is used to retrieve final answers.

Figure 13 provides an example of the stages. Figure 13b illustrates the stages of phrase mapping and disambiguation, while Figure 13c shows query graph construction and result retrieval. Following the pipeline, existing works focus on developing various techniques for the first two stages, which pose tougher challenges than the last stage. The remaining of the section reviews and compares these works.

Yahya et al. [3] develop a framework, called DEANNA that comprises a full suite of the above stages. The framework uses multiple detectors for detecting phrases of different types. For noun phrase detection (such as conceptual types and entities), they use a detector that utilizes a dictionary which is constructed as part of attributed graphs, independently of NLQA task. For relation detection, it leverages REVERB [47], a tool that automatically identifies and extracts binary relationships from English sentences. After the phrases are detected, each phrase is mapped to a set of semantic items in the attributed graph. Such mapping process also relies on a dictionary that maps a phrase to vertex labels or textual patterns of edge labels. Given the mapping results, the framework formalizes disambiguation as an integer linear program (ILP). To solve ILP, it constructs a disambiguation graph that encodes all possible mappings and defines an objective function that maximizes the joint quality of the mappings over entities, types and relations. After that, once phrases are mapped to unique semantic items, they can construct query graph and retrieve the final answers. Clearly, one limitation of the DEANNA framework is that it heavily relies on the dictionaries, which are external resources of the framework.

To address the limitation, many approaches are proposed to *automatically* construct a mapping between phrases and semantic items in attributed graphs, rather than relying on external dictionaries. Zou et al. [242] propose to automatically map edge labels in attributed graphs to their natural language paraphrases. For example, edge label *actIn* in Figure 13a can be mapped to "*play in*" in natural language expressions. They utilize a triple-based dependency-parsing approach PATTY [145]. The approach considers each edge, denoted by $(\mathsf{s}, \mathsf{p}, \mathsf{o})$ in the attributed graph, where s and o are associated vertices and p is an edge label, e.g., (Antonio Banderas, *actIn*, Philadelphia). It retrieves all the sentences in a natural language corpus that mentions both s and o, and applies a dependency parser to each retrieved sentence. Then, the shortest paths between s and o in the dependency parsing result are selected as the paraphrases of relation p. Using this approach, Zou et al. [242] construct a paraphrase dictionary. During the online stage, an input question is parsed into a dependency tree where relation phrases are recognized with the help of the paraphrase dictionary. Moreover, for disambiguation, they consider the structure of attributed graphs, e.g., star-like subgraphs containing mapped edge labels.

Yang et al. [206] propose an embedding-based approach to mapping phrases to semantic items in attributed graphs. The idea is to first collect pairs of edge labels and their paraphrases as training data, and then represent both natural language phrases and semantic items into numeric vectors. Let C denote a set of n-grams of an input question that are the lexical representations of entities, P denote a set of predicates in the graph corresponding to relations extracted by PATTY, and T denote a set of types in the attributed graph. The approach considers a training triple as $w = [\mathsf{c}, \mathsf{t}, \mathsf{p}]$, where each feature ($\mathsf{c} \in C, \mathsf{t} \in T, \mathsf{p} \in$

P) is encoded in the embedding-based representation. Using the training triples, it learns the semantic mappings of $C - T$, $C - P$, and $T - P$ based on relation-predicate pairs and question pattern-predicate pairs.

Besides mapping construction, disambiguation is also challenging due to the inherent ambiguity of natural language expressions. Most of the approaches [131, 147] utilize the NLP techniques for disambiguation. In contrast, Zou et al. [242] take a lazy approach and push down the disambiguation to the query evaluation stage. They allow that phrases in input question mapped to multiple semantic items in an attributed graph, and resolve the ambiguity at the time when answers of the queries are found. To speed up the performance, they use PATTY to build a paraphrase dictionary that records the semantic equivalence between relation phrases and predicates.

He et al. [78] propose a method based on Markov Logic Networks (MLNs) [159]. They formalize the knowledge to resolve the ambiguities in the first stage as first-order logic clauses in an MLN. In the framework of MLN, all clauses will have an interactive effect and solve the problem together across all stages as a unified process. In this way, the results in each stage can be globally optimized. For example, for phrase detection, they do not use the dictionary or dependency parser which may cause a problem of low coverage. To avoid missing useful phrases, they retain all n-grams as phrase candidates, and then use predefined rules to filter. To improve phrase mapping, they apply different techniques for different types of phrases: they employ anchor, redirection and disambiguation information from Wikipedia for noun phrase mapping, and utilize the resources from PATTY for relation mapping.

To further improve the performance, some approaches propose to make use of other potentially relevant text data, outside the KB, which could supplement the available information. Savenkov et al. [158] introduce a new system, Text2KB, that enriches question answering using external text data. Text2KB utilizes Web search results, community question answering and general text document collection data, to alleviate the three challenges in NLQA.

On the other hand, to improve generalization ability, the concept of *query template* is introduced to directly model the internal structure of the question [38, 182, 183]. A query template specifies query's select or ask clause, its filter and aggregation functions, as well as the number and form of its triples, and a specific query can be instantiated by mapping phrases to the template.

8.1.2 Triple query construction

As a special case of NLQA, triple query is used to express a natural language question that is composed of an entity mention and a binary relation description, and the answer to this question would be an entity that has the relation with the entity. In practice, triple query is usually a popular question answering requirement, and thus many approaches introduce techniques to address this query form. Different from general query graph construction, these approaches do not need to explicitly generate complete semantic structures for input questions, and some latest technologies, such as convolutional neural networks (CNN) and recurrent neural networks (RNN) with Gated Recurrent Units (GRU) can be leveraged.

Bao et al. [8] treat NLQA as a machine translation procedure. They parse each input question, and answers of the span covered by each parser cell are considered the translations of that cell. The final answers can be obtained from the root cell. Yih et al. [212] propose a semantic similarity method using CNN. The method trains two semantic similarity models: one measures the similarity of entity mentions with entities in attributed graphs and the other measures the similarity of relation patterns and relations in attributed graphs. By using a general semantic similarity model to measure two pairs mentioned above, they can construct the triple query to find the answer with higher precision. Note that this method splits the question into mention-pattern pairs, but it is only applicable for simple questions. Furthermore, Denis et al. [132] propose RNN with GRU containing a nested word/character-level question encoder which allows for handling out-of-vocabulary and rare word problems while still being able to exploit word-level semantics. Using this method, they can better handle synonyms and find better matches between words in the question and entities or predicates in the graph.

8.2 End-to-end question answering

Query graph construction relies on complicated natural language processing pipelines with respect to some specific domains (e.g., domain-dependent dictionaries or NLP tools), and may encounter difficulties when adapting to new domains. To address this issue, many end-to-end question answering approaches [29, 44, 211] have been proposed recently. The basic idea is to model question answering as a *classification* problem where the pair of input question and each candidate answer is regarded as an instance in the classification problem. Based on this idea, the approaches focus on addressing the following two key problems.

1) *Feature extraction* extracts features from the natural language question. To this end, Yao et al. [211] use a dependency parser to convert a question into a question graph. Li et al. [44] utilize word embeddings (i.e., vector representation) to represent question words, while Chen et al. [29] further improve the techniques by considering *position-aware influence* of question words.
2) *Classification model* identifies the correct answers. Yao et al. [211] leverage a back-off probability estimation model to classify whether each candidate answer is appropriate for the question. Li et al. [44] introduce a deep learning-based model, the multi-column convolutional neural networks (CNNs). Chen et al. [29] propose a Positional Attention based RNN (RNN-POA) model, which models the position-aware influence of the question word for answer's attentive representations.

As this survey does not focus on deep learning, we encourage the interested readers to refer to original papers.

8.3 Summary

Natural language question answering returns answers in attributed graphs to natural language questions. The essential challenges are how to understand questions and how to bridge the gap between natural language and structural semantics of attributed graphs. In this section, we review two mainstream methods to address the challenge.

– **Query graph construction**-based approaches focus on interpreting a natural language question into a structural query graph, which is then used to retrieve answers. The main challenge is how to cope with the diversification of natural language. Existing works apply advanced NLP tools, such as relation extraction, dependency parser, and graph embedding.

- **End-to-end query answering**-based approaches directly match subgraphs as answers with respect to a natural language question. To this end, many deep learning models, such as CNN and RNN, have been applied to understand the semantics of natural language questions.

However, most of the existing works assume a fixed and pre-defined set of lexical triggers which limit their domains and scalability capability. Thus, the future research opportunities may fall into the following directions. First, it is still very challenging to develop *open-domain* question answering over attributed graphs. Thus, there may be many research topics, such as open-domain relation paraphrase learning, open-domain query construction, etc. Second, it calls for a *dialog* mechanism for NLQA to incorporate user's feedback. There may be many research issues in the mechanism, such as how to position the "errors" in initial query graphs, how to recommend correction options to refine the queries, and how to assure efficiency.

9 Conclusion

Attributed graphs are ubiquitously adopted for modeling complex structures with rich semantics in many real-world scenarios. In recent years, a great variety of queries on attributed graphs have been proposed to meet the demand for knowledge and information extraction in numerous data-intensive applications. In this survey, we have reviewed existing studies on typical attributed graph queries. First, we provided a taxonomy of typical attributed graph queries according to query inputs and outputs. Then, we presented motivating applications and summarized formulations for each category of queries. Finally, we compared different approaches proposed for each query category in terms of how their formulations affect the result quality and discussed the technical challenges of query processing.

We also pointed out several promising future research directions, some of which were common across different categories of attributed graph queries. The first one is to provide assistance to help end-users design queries effectively. In particular, structured graph queries are often expressed in the form of *declarative languages*, e.g., SQL and SPARQL, which are difficult to understand by non-expert users, and thus it is an important task to assist non-expert users to specify their queries. The second one is to devise interactive query interfaces to collect user feedback so as to improve the quality of query results incrementally. The third one is to support attributed graph queries in dynamic, streaming, and distributed settings since real-world graph data is often highly dynamic, available as a stream, or too large to fit in a single machine. The fourth one is to exploit the computation power of modern hardwares such as GPUs and FPGA for accelerating attributed graph query processing. The fifth one is to incorporate machine learning (ML) approaches into attributed graph queries. Although ML-based methods have been adopted for unstructured queries, e.g., graph embedding for similarity search and NLP techniques for NLQA, very little effort has been made to exploit the potential of ML techniques for structured queries.

We believe that this survey can serve as a guidance for end-users to choose appropriate queries in their application scenarios as well as a good starting point for new researchers who want to work in this field.

Acknowledgments This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-2-065), NSF of China (62072461, 61632016, U1911203, 61925205, U1711261), the Research Funds of Renmin University of China (18XNLG18), Huawei, and TAL Education.

References

1. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: enabling keyword search over relational databases. In: SIGMOD, p 627 (2002)
2. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68:1–68:40 (2017)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A nucleus for a Web of open data. In: ISWC, pp 722–735 (2007)
4. Azimov, R., Grigorev, S.: Context-free path querying by matrix multiplication. In: GRADES-NDA, pp 5:1–5:10 (2018)
5. Bai, Y., Wang, C., Ying, X.: Para-g: Path pattern query processing on large graphs. World Wide Web **20**(3), 515–541 (2017)
6. Balmin, A., Hristidis, V., Papakonstantinou, Y.: ObjectRank: Authority-based keyword search in databases. In: VLDB, pp 564–575 (2004)
7. Baltadzhieva, A., Chrupala, G.: Question quality in community question answering forums: a survey. SIGKDD Explor. Newsl. **17**(1), 8–13 (2015)
8. Bao, J.-W., Duan, N., Zhou, M., Zhao, T.: Knowledge-based question answering as machine translation. In: ACL, pp 967–976 (2014)
9. Bao, Z., Zeng, Y., Jagadish, H.V., Ling, T.W.: Exploratory keyword search with interactive input. In: SIGMOD, pp 871–876 (2015)
10. Barrett, C.L., Bisset, K.R., Holzer, M., Konjevod, G., Marathe, M.V., Wagner, D.: Engineering label-constrained shortest-path algorithms. In: AAIM, pp 27–37 (2008)
11. Barrett, C.L., Jacob, R., Marathe, M.V.: Formal-language-constrained path problems. SIAM J. Comput. **30**(3), 809–837 (2000)
12. Barthélemy, M.: Spatial networks. Phys. Rep. **499**(1), 1–101 (2011)
13. Beheshti, S.-M.-R., Benatallah, B., Nezhad, H.R.M., Allahbakhsh, M.: A framework and a language for on-line analytical processing on graphs. In: WISE, pp 213–227 (2012)
14. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using BANKS. In: ICDE, pp 431–440 (2002)
15. Bi, F., Chang, L., Lin, X., Qin, L., Zhang, W.: Efficient subgraph matching by postponing cartesian products. In: SIGMOD, pp 1199–1214 (2016)
16. Bi, F., Chang, L., Lin, X., Zhang, W.: An optimal and progressive approach to online search of top-k influential communities. PVLDB **11**(9), 1056–1068 (2018)
17. Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD, pp 1247–1250 (2008)
18. Bonchi, F., Gionis, A., Gullo, F., Ukkonen, A.: Distance oracles in edge-labeled graphs. In: EDBT, pp 547–558 (2014)
19. Bonifati, A., Ciucanu, R., Lemay, A.: Learning path queries on graph databases. In: EDBT, pp 109–120 (2015)
20. Bonnici, V., Giugno, R.: On the variable ordering in subgraph isomorphism algorithms. IEEE/ACM Trans. Comput. Biology Bioinform. **14**(1), 193–203 (2017)
21. Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D.E., Ferro, A.: A subgraph isomorphism algorithm and its application to biochemical data. BMC Bioinforma. **14**(S-7), S13 (2013)
22. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognit. Lett. **19**(3-4), 255–259 (1998)
23. Cai, H., Zheng, V.W., Chang, K.C.-C.: A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Trans. Knowl. Data Eng. **30**(9), 1616–1637 (2018)
24. Cao, Y., Jiang, T., Girke, T.: A maximum common substructure-based algorithm for searching and predicting drug-like compounds. Bioinformatics **24**(13), 366–374 (2008)
25. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.S.: Graph OLAP: Towards online analytical processing on graphs. In: ICDM, pp 103–112 (2008)

26. Chen, L., Liu, C., Liao, K., Li, J., Zhou, R.: Contextual community search over large social networks. In: ICDE, pp 88–99 (2019)
27. Chen, L., Liu, C., Yang, X., Wang, B., Li, J., Zhou, R.: Efficient batch processing for multiple keyword queries on graph data. In: CIKM, pp 1261–1270 (2016)
28. Chen, L., Liu, C., Zhou, R., Li, J., Yang, X., Wang, B.: Maximum co-located community search in large scale social networks. PVLDB **11**(10), 1233–1246 (2018)
29. Chen, Q., Hu, Q., Huang, J.X., He, L., An, W.: Enhancing recurrent neural networks with positional attention for question answering. In: SIGIR, pp 993–996 (2017)
30. Chen, Y., Wang, W., Liu, Z.: Keyword-based search and exploration on databases. In: ICDE, pp 1380–1383 (2011)
31. Chen, Y.-C., Zhu, W.-Y., Peng, W.-C., Lee, W.-C., Lee, S.-Y.: CIM: community-based influence maximization in social networks. ACM Trans. Intell. Syst. Technol. **5**(2), 25:1–25:31 (2014)
32. Cheng, J., Yu, J.X., Ding, B., Yu, P.S., Wang, H.: Fast graph pattern matching. In: ICDE, pp 913–922 (2008)
33. Cheng, J., Zeng, X., Yu, J.X.: Top-k graph pattern matching over large graphs. In: ICDE, pp 1033–1044 (2013)
34. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Exact and approximate algorithms for finding k-shortest paths with limited overlap. In: EDBT, pp 414–425 (2017)
35. Conte, A., Ferraro, G., Grossi, R., Marino, A., Sadakane, K., Uno, T.: Node similarity with q-grams for real-world labeled networks. In: KDD, pp 1282–1291 (2018)
36. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp 151–158 (1971)
37. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell. **26**(10), 1367–1372 (2004)
38. Cui, W., Xiao, Y., Wang, H., Song, Y., Hwang, S.-W., Wang, W.: KBQA: Learning question answering over QA corpora and knowledge bases. PVLDB **10**(5), 565–576 (2017)
39. Dey, S.C., Cuevas-Vicenttín, V., Köhler, S., Gribkoff, E., Wang, M., Ludäscher, B.: On implementing provenance-aware regular path queries with relational query engines. In: EDBT/ICDT Workshops, pp 214–223 (2013)
40. Dibbelt, J., Pajor, T., Wagner, D.: User-constrained multimodal route planning. ACM J. Exp. Algorithmics **19**(1), 3.2:1–3.2:19 (2014)
41. Diefenbach, D., López, V., Singh, K.D., Maret, P.: Core techniques of question answering systems over knowledge bases: a survey. Knowl. Inf. Syst. **55**(3), 529–569 (2018)
42. Ding, B., Wang, H., Jin, R., Han, J., Wang, Z.: Optimizing index for taxonomy keyword search. In: SIGMOD, pp 493–504 (2012)
43. Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X., Lin, X.: Finding top-k min-cost connected trees in databases. In: ICDE, pp 836–845 (2007)
44. Dong, L., Wei, F., Zhou, M., Xu, K.: Question answering over freebase with multi-column convolutional neural networks. In: ACL-IJCNLP, pp 260–269 (2015)
45. Du, B., Zhang, S., Cao, N., Tong, H.: FIRST: Fast interactive attributed subgraph matching. In: KDD, pp 1447–1456 (2017)
46. Dutta, S., Nayek, P., Bhattacharya, A.: Neighbor-aware search for approximate labeled graph matching using the chi-square statistics. In: WWW, pp 1281–1290 (2017)
47. Fader, A., Soderland, S., Etzioni, O.: Identifying relations for open information extraction. In: EMNLP, pp 1535–1545 (2011)
48. Fan, J., Li, G., Zhou, L.: Interactive SQL query suggestion: Making databases user-friendly. In: ICDE, pp 351–362 (2011)
49. Fan, Q., Wang, Z., Chan, C.-Y., Tan, K.-L.: Towards neighborhood window analytics over large-scale graphs. In: DASFAA, pp 201–217 (2016)
50. Fan, W.: Graph pattern matching revised for social network analysis. In: ICDT, pp 8–21 (2012)
51. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y.: Adding regular expressions to graph reachability and pattern queries. In: ICDE, pp 39–50 (2011)
52. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.: Graph pattern matching: From intractable to polynomial time. PVLDB **3**(1), 264–275 (2010)
53. Fan, W., Li, J., Ma, S., Wang, H., Wu, Y.: Graph homomorphism revisited for graph matching. PVLDB **3**(1), 1161–1172 (2010)
54. Fan, W., Wang, X., Wu, Y.: Performance guarantees for distributed reachability queries. PVLDB **5**(11), 1304–1315 (2012)
55. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching. PVLDB **6**(13), 1510–1521 (2013)
56. Fang, L., Sarma, A.D., Yu, C., Bohannon, P.: REX: Explaining relationships between entity pairs. PVLDB **5**(3), 241–252 (2011)

57. Fang, Y., Cheng, R., Chen, Y., Luo, S., Hu, J.: Effective and efficient attributed community search. VLDB J. **26**(6), 803–828 (2017)
58. Fang, Y., Cheng, R., Li, X., Luo, S., Hu, J.: Effective community search over large spatial graphs. PVLDB **10**(6), 709–720 (2017)
59. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. PVLDB **9**(12), 1233–1244 (2016)
60. Fang, Y., Huang, X., Qin, L., Zhang, Y., Zhang, W., Cheng, R., Lin, X.: A survey of community search over big graphs. VLDB J. **29**(1), 353–392 (2020)
61. Fang, Y., Yang, Y., Zhang, W., Lin, X., Cao, X.: Effective and efficient community search over large heterogeneous information networks. PVLDB **13**(6), 854–867 (2020)
62. Fang, Y., Lin, W., Zheng, V.W., Wu, M., Chang, K.C.-C., Li, X.: Semantic proximity search on graphs with metagraph-based learning. In: ICDE, pp 277–288 (2016)
63. Fard, A., Nisar, M.U., Miller, J.A., Ramaswamy, L.: Distributed and scalable graph pattern matching: models and algorithms. Int. J. Big Data **1**(1), 1–14 (2014)
64. Fard, A., Nisar, M.U., Ramaswamy, L., Miller, J.A., Saltz, M.: A distributed vertex-centric approach for pattern matching in massive graphs. In: BigData, pp 403–411 (2013)
65. Foggia, P., Percannella, G., Vento, M.: Graph matching and learning in pattern recognition in the last 10 years. Int. J. Pattern Recognit. Artif. Intell. **28**(1), 1450001 (2014)
66. Gallagher, B.: Matching structure and semantics: A survey on graph-based pattern matching. In: AAAI FS-06-02, pp 45–53 (2006)
67. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. **13**(1), 113–129 (2010)
68. Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: A survey. Knowl.-Based Syst. **151**, 78–94 (2018)
69. Gu, Q., Zhang, C., Sun, T., Ji, Y., Hu, Z., Qiu, X.: Path sampling based relevance search in heterogeneous networks. In: BigCom, pp 453–463 (2016)
70. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked keyword search over XML documents. In: SIGMOD, pp 16–27 (2003)
71. Guo, L., Deng, Y., Liao, K., He, Q., Sellis, T., Hu, Z.: A fast algorithm for optimally finding partially disjoint shortest paths. In: IJCAI, pp 1456–1462 (2018)
72. Han, S., Zou, L., Yu, J.X., Zhao, D.: Keyword search on RDF graphs - A query graph assembly approach. In: CIKM, pp 227–236 (2017)
73. Han, W.-S., Lee, J., Lee, J.-H.: Turbo$_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In: SIGMOD, pp 337–348 (2013)
74. Harris, S., Seaborne, A., Prud'hommeaux, E.: Sparql 1.1 query language. W3C Recommendation **21**(10), 778 (2013)
75. Hassan, M.S., Aref, W.G., Aly, A.M.: Graph indexing for shortest-path finding over dynamic sub-graphs. In: SIGMOD, pp 1183–1197 (2016)
76. He, H., Wang, H., Yang, J., Yu, P.S.: BLINKS: ranked keyword searches on graphs. In: SIGMOD, pp 305–316 (2007)
77. He, J., Bailey, J., Zhang, R.: Exploiting transitive similarity and temporal dynamics for similarity search in heterogeneous information networks. In: DASFAA, pp 141–155 (2014)
78. He, S., Liu, K., Zhang, Y., Xu, L., Zhao, J.: Question answering over linked data using first-order logic. In: EMNLP, pp 1092–1103 (2014)
79. Hellings, J.: Conjunctive context-free path queries. In: ICDT, pp 119–130 (2014)
80. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: FOCS, pp 453–462 (1995)
81. Höffner, K., Walter, S., Marx, E., Usbeck, R., Lehmann, J., Ngomo, A.-C.N.: Survey on challenges of question answering in the semantic Web. Semantic Web **8**(6), 895–920 (2017)
82. Holme, P., Saramäki, J.: Temporal networks. Phys. Rep. **519**(3), 97–125 (2012)
83. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient IR-style keyword search over relational databases. In: VLDB, pp 850–861 (2003)
84. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword search in relational databases. In: VLDB, pp 670–681 (2002)
85. Huan, J., Bandyopadhyay, D., Wang, W., Snoeyink, J., Prins, J., Tropsha, A.: Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. J. Comput. Biol. **12**(6), 657–671 (2005)
86. Huang, X., Lakshmanan, L.V.S.: Attribute-driven community search. PVLDB **10**(9), 949–960 (2017)
87. Huang, Z., Zheng, Y., Cheng, R., Sun, Y., Mamoulis, N., Li, X.: Meta structure: Computing relevance in large heterogeneous information networks. In: KDD, pp 1595–1604 (2016)

88. Hung, H.H., Bhowmick, S.S., Truong, B.Q., Choi, B., Zhou, S.: QUBLE: towards blending interactive visual subgraph search queries on large networks. VLDB J. **23**(3), 401–426 (2014)
89. Islam, M.S., Liu, C., Li, J.: Efficient answering of why-not questions in similar graph matching. IEEE Trans. Knowl. Data Eng. **27**(10), 2672–2686 (2015)
90. Jayaram, N., Khan, A., Li, C., Yan, X., Elmasri, R.: Querying knowledge graphs by example entity tuples. IEEE Trans. Knowl. Data Eng. **27**(10), 2797–2811 (2015)
91. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: KDD, pp 538–543 (2002)
92. Jeh, G., Widom, J.: Scaling personalized Web search. In: WWW, pp 271–279 (2003)
93. Jiang, M., Fu, A.W.-C., Wong, R.C.-W.: Exact top-k nearest keyword search in large networks. In: SIGMOD, pp 393–404 (2015)
94. Jin, R., Hong, H., Wang, H., Ruan, N., Xiang, Y.: Computing label-constraint reachability in graph databases. In: SIGMOD, pp 123–134 (2010)
95. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB, pp 505–516 (2005)
96. Kargar, M., An, A.: Keyword search in graphs: Finding r-cliques. PVLDB **4**(10), 681–692 (2011)
97. Kargar, M., An, A.: Efficient top-k keyword search in graphs with polynomial delay. In: ICDE, pp 1269–1272 (2012)
98. Kargar, M., An, A.: Finding top-k, r-cliques for keyword search from graphs in polynomial delay. Knowl. Inf. Syst. **43**(2), 249–280 (2015)
99. Katsarou, F., Ntarmos, N., Triantafillou, P.: Performance and scalability of indexed subgraph query processing methods. PVLDB **8**(12), 1566–1577 (2015)
100. Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., Tao, S.: Neighborhood based fast graph search in large networks. In: SIGMOD, pp 901–912 (2011)
101. Khan, A., Wu, Y., Aggarwal, C.C., Yan, X.: NeMa: Fast graph search with label similarity. PVLDB **6**(3), 181–192 (2013)
102. Khashabi, D., Khot, T., Sabharwal, A., Roth, D.: Question answering as global reasoning over semantic abstractions. In: AAAI, pp 1905–1914 (2018)
103. Koschmieder, A., Leser, U.: Regular path queries on large graphs. In: SSDBM, pp 177–194 (2012)
104. Lao, N., Cohen, W.W.: Fast query execution for retrieval models based on path-constrained random walks. In: KDD, pp 881–888 (2010)
105. Lao, N., Cohen, W.W.: Relational retrieval using a combination of path-constrained random walks. Mach. Learn. **81**(1), 53–67 (2010)
106. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: KDD, pp 467–476 (2009)
107. Le, T.N., Ling, T.W.: Survey on keyword search over XML documents. SIGMOD Rec. **45**(3), 17–28 (2016)
108. Le, W., Li, F., Kementsietsidis, A., Duan, S.: Scalable keyword search on large RDF data. IEEE Trans. Knowl. Data Eng. **26**(11), 2774–2788 (2014)
109. Lee, J., Han, W.-S., Kasperovics, R., Lee, J.-H.: An in-depth comparison of subgraph isomorphism algorithms in graph databases. PVLDB **6**(2), 133–144 (2012)
110. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD, pp 903–914 (2008)
111. Li, J., Cao, Y., Ma, S.: Relaxing graph pattern matching with explanations. In: CIKM, pp 1677–1686 (2017)
112. Li, J., Liu, C., Islam, M.S.: Keyword-based correlated network computation over large social media. In: ICDE, pp 268–279 (2014)
113. Li, J., Wang, X., Deng, K., Yang, X., Sellis, T., Yu, J.X.: Most influential community search over large social networks. In: ICDE, pp 871–882 (2017)
114. Li, R.-H., Qin, L., Ye, F., Yu, J.X., Xiao, X., Xiao, N., Zheng, Z.: Skyline community search in multi-valued networks. In: SIGMOD, pp 457–472 (2018)
115. Li, R.-H., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. PVLDB **8**(5), 509–520 (2015)
116. Li, R.-H., Qin, L., Yu, J.X., Mao, R.: Efficient and progressive group steiner tree search. In: SIGMOD, pp 91–106 (2016)
117. Li, R.-H., Su, J., Qin, L., Yu, J.X., Dai, Q.: Persistent community search in temporal networks. In: ICDE, pp 797–808 (2018)
118. Liang, J., Ajwani, D., Nicholson, P.K., Sala, A., Parthasarathy, S.: What links alice and bob?: Matching and ranking semantic patterns in heterogeneous networks. In: WWW, pp 879–889 (2016)
119. Liptchinsky, V., Satzger, B., Zabolotnyi, R., Dustdar, S.: Expressive languages for selecting groups from graph-structured data. In: WWW, pp 761–770 (2013)

120. Liu, C., Chen, C., Han, J., Yu, P.S.: GPLAG: detection of software plagiarism by program dependence graph analysis. In: KDD, pp 872–881 (2006)
121. Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: SIGMOD, pp 563–574 (2006)
122. Liu, G., Zheng, K., Wang, Y., Orgun, M.A., Liu, A., Zhao, L., Zhou, X.: Multi-constrained graph pattern matching in large-scale contextual social graphs. In: ICDE, pp 351–362 (2015)
123. Liu, Q., Zhao, M., Huang, X., Xu, J., Gao, Y.: Truss-based community search over large directed graphs. In: SIGMOD, pp 2183–2197 (2020)
124. Liu, Q., Zhu, Y., Zhao, M., Huang, X., Xu, J., Gao, Y.: VAC: vertex-centric attributed community search. In: ICDE, pp 937–948 (2020)
125. Liu, Y., Safavi, T., Dighe, A., Koutra, D.: Graph summarization methods and applications: A survey. ACM Comput. Surv. **51**(3), 62:1–62:34 (2018)
126. Liu, Z., Zheng, V.W., Zhao, Z., Li, Z., Yang, H., Wu, M., Ying, J.: Interactive paths embedding for semantic proximity search on heterogeneous graphs. In: KDD, pp 1860–1869 (2018)
127. Liu, Z., Zheng, V.W., Zhao, Z., Yang, H., Chang, K.C.-C., Wu, M., Ying, J.: Subgraph-augmented path embedding for semantic user search on heterogeneous social network. In: WWW, pp 1613–1622 (2018)
128. Liu, Z., Zheng, V.W., Zhao, Z., Zhu, F., Chang, K.C.-C., Wu, M., Ying, J.: Semantic proximity search on heterogeneous graph by proximity embedding. In: AAAI, pp 154–160 (2017)
129. Liu, Z., Zheng, V.W., Zhao, Z., Zhu, F., Chang, K.C.-C., Wu, M., Ying, J.: Distance-aware dag embedding for proximity search on heterogeneous graphs. In: AAAI, pp 2355–2362 (2018)
130. Livi, L., Rizzi, A.: The graph matching problem. Pattern Anal. Appl. **16**(3), 253–283 (2013)
131. López, V., Fernández, M., Motta, E., Stieler, N.: PowerAqua: Supporting users in querying and exploring the semantic Web. Semantic Web **3**(3), 249–265 (2012)
132. Lukovnikov, D., Fischer, A., Lehmann, J., Auer, S.: Neural network-based question answering over knowledge graphs on word and character level. In: WWW, pp 1211–1220 (2017)
133. Luo, Y., Lin, X., Wang, W., Zhou, X.: Spark: top-k keyword query in relational databases. In: SIGMOD, pp 115–126 (2007)
134. Luo, Y., Wang, W., Lin, X., Zhou, X., Wang, J., Li, K.: SPARK2: Top-k keyword query in relational databases. IEEE Trans. Knowl. Data Eng. **23**(12), 1763–1780 (2011)
135. Lyu, B., Qin, L., Lin, X., Chang, L., Yu, J.X.: Scalable supergraph search in large graph databases. In: ICDE, pp 157–168 (2016)
136. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Capturing topology in graph pattern matching. PVLDB **5**(4), 310–321 (2011)
137. Ma, T., Yu, S., Cao, J., Tian, Y., Al-Dhelaan, A., Al-Rodhaan, M.: A comparative study of subgraph matching isomorphic methods in social networks. IEEE Access **6**, 66621–66631 (2018)
138. Medeiros, C.M., Musicante, M.A., da Costa, U.S.: Efficient evaluation of context-free path queries for graph databases. In: SAC, pp 1230–1237 (2018)
139. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. SIAM J. Comput. **24**(6), 1235–1258 (1995)
140. Meng, C., Cheng, R., Maniu, S., Senellart, P., Zhang, W.: Discovering meta-paths in large heterogeneous information networks. In: WWW, pp 754–764 (2015)
141. Meng, X., Shi, C., Li, Y., Zhang, L., Wu, B.: Relevance measure in large-scale heterogeneous networks. In: APWeb, pp. 636–643 (2014)
142. Meng, Z., Shen, H.: Dissimilarity-constrained node attribute coverage diversification for novelty-enhanced top-*k* search in large attributed networks. Knowl.-Based Syst. **150**, 85–94 (2018)
143. Mondal, J., Deshpande, A.: EAGr: supporting continuous ego-centric aggregate queries over large dynamic graphs. In: SIGMOD, pp 1335–1346 (2014)
144. Mottin, D., Lissandrini, M., Velegrakis, Y., Palpanas, T.: Exemplar queries: Give me an example of what you need. PVLDB **7**(5), 365–376 (2014)
145. Nakashole, N., Weikum, G., Suchanek, F.M.: PATTY: A taxonomy of relational patterns with semantic types. In: EMNLP-CoNLL, pp 1135–1145 (2012)
146. Namaki, M.H., Wu, Y., Zhang, X.: GExp: Cost-aware graph exploration with keywords. In: SIGMOD, pp 1729–1732 (2018)
147. Napolitano, G., Usbeck, R., Ngomo, A.-C.N.: The scalable question answering over linked data (SQA) challenge 2018. In: SemWebEval, pp 69–75 (2018)
148. Nolé, M., Sartiani, C.: Regular path queries on massive graphs. In: SSDBM, pp 13:1–13:12 (2016)
149. Pacaci, A., Bonifati, A., Özsu, M.T.: Regular path query evaluation on streaming graphs. In: SIGMOD, pp 1415–1430 (2020)
150. Pande, S., Ranu, S., Bhattacharya, A.: SkyGraph: Retrieving regions of interest using skyline subgraph queries. PVLDB **10**(11), 1382–1393 (2017)

151. Peng, Y., Zhang, Y., Lin, X., Qin, L., Zhang, W.: Answering billion-scale label-constrained reachability queries within microsecond. PVLDB **13**(6), 812–825 (2020)
152. Qiao, M., Qin, L., Cheng, H., Yu, J.X., Tian, W.: Top-k nearest keyword search on large graphs. PVLDB **6**(10), 901–912 (2013)
153. Qin, L., Yu, J.X., Chang, L.: Diversifying top-k results. PVLDB **5**(11), 1124–1135 (2012)
154. Qu, Q., Zhu, F., Yan, X., Han, J., Yu, P.S., Li, H.: Efficient topological OLAP on information networks. In: DASFAA, pp 389–403 (2011)
155. Ranu, S., Hoang, M.X., Singh, A.K.: Answering top-k representative queries on graph databases. In: SIGMOD, pp 1163–1174 (2014)
156. Ren, X., Wang, J.: Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. PVLDB **8**(5), 617–628 (2015)
157. Rice, M.N., Tsotras, V.J.: Graph indexing of road networks for shortest path queries with label restrictions. PVLDB **4**(2), 69–80 (2010)
158. Savenkov, D., Agichtein, E.: When a knowledge base is not enough: Question answering over knowledge bases with external text data. In: SIGIR, pp 235–244 (2016)
159. Schoenfisch, J., Meilicke, C., von Stülpnagel, J., Ortmann, J., Stuckenschmidt, H.: Root cause analysis in it infrastructures using ontologies and abduction in markov logic networks. Inf. Syst. **74**(Part 2), 103–116 (2018)
160. Semertzidis, K., Pitoura, E.: Top-k durable graph pattern queries on temporal graphs. IEEE Trans. Knowl. Data Eng. **31**(1), 181–194 (2019)
161. Shang, H., Lin, X., Zhang, Y., Yu, J.X., Wang, W.: Connected substructure similarity search. In: SIGMOD, pp 903–914 (2010)
162. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. PVLDB **1**(1), 364–375 (2008)
163. Shang, H., Zhu, K., Lin, X., Zhang, Y., Ichise, R.: Similarity search on supergraph containment. In: ICDE, pp 637–648 (2010)
164. Shi, C., Kong, X., Huang, Y., Yu, P.S., Wu, B.: HeteSim: A general framework for relevance measure in heterogeneous networks. IEEE Trans. Knowl. Data Eng. **26**(10), 2479–2492 (2014)
165. Shi, C., Kong, X., Yu, P.S., Xie, S., Wu, B.: Relevance search in heterogeneous networks. In: EDBT, pp 180–191 (2012)
166. Shi, C., Li, Y., Zhang, J., Sun, Y., Yu, P.S.: A survey of heterogeneous information network analysis. IEEE Trans. Knowl. Data Eng. **29**(1), 17–37 (2017)
167. Shi, Y., Chan, P.-W., Zhuang, H., Gui, H., Han, J.: PReP: Path-based relevance from a probabilistic perspective in heterogeneous information networks. In: KDD, pp 425–434 (2017)
168. Shi, Y., Cheng, G., Kharlamov, E.: Keyword search over knowledge graphs via static and dynamic hub labelings. In: WWW, pp 235–245 (2020)
169. Shikha, A., Junhu, W., Md. Saiful Islam: Modular Decomposition-Based Graph Compression for Fast Reachability Detection. Data Sci. Eng. **4**(3), 193–207 (2019)
170. Sommer, C.: Shortest-path queries in static networks. ACM Comput. Surv. **46**(4), 45:1–45:31 (2014)
171. Song, C., Ge, T., Chen, C.X., Wang, J.: Event pattern matching over graph streams. PVLDB **8**(4), 413–424 (2014)
172. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW, pp 697–706 (2007)
173. Sun, Y., Han, J., Yan, X., Yu, P.S., Wu, T.: PathSim: Meta path-based top-k similarity search in heterogeneous information networks. PVLDB **4**(11), 992–1003 (2011)
174. Sun, Z., Wang, H., Wang, H., Shao, B., Li, J.: Efficient subgraph matching on billion node graphs. PVLDB **5**(9), 788–799 (2012)
175. Tabei, Y., Tsuda, K.: Kernel-based similarity search in massive graph databases with wavelet trees. In: SDM, pp 154–163 (2011)
176. Tian, Y., McEachin, R.C., Santos, C., States, D.J., Patel, J.M.: SAGA: a subgraph matching tool for biological graphs. Bioinformatics **23**(2), 232–239 (2007)
177. Tong, H., Faloutsos, C., Gallagher, B., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In: KDD, pp 737–746 (2007)
178. Tran, Q.T., Chan, C.-Y.: How to conquer why-not questions. In: SIGMOD, pp 15–26 (2010)
179. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In: ICDE, pp 405–416 (2009)
180. U, L.H., Yao, K., Mak, H.F.: PathSimExt: Revisiting pathsim in heterogeneous information networks. In: WAIM, pp 38–42 (2014)
181. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23**(1), 31–42 (1976)

182. Unger, C., Bühmann, L., Lehmann, J., Ngomo, A.-C.N., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: WWW, pp 639–648 (2012)
183. Unger, C., Cimiano, P.: Pythia: Compositional meaning construction for ontology-based question answering on the semantic Web. In: NLDB, pp 153–160 (2011)
184. Vachik, S., Dave, B.Z., Pin-Yu, C., Mohammad Al, H.: Neural-Brane: Neural Bayesian Personalized Ranking for Attributed Network Embedding. Data Sci. Eng. **4**(2), 119–131 (2019)
185. Valstar, L.D.J., Fletcher, G.H.L., Yoshida, Y.: Landmark indexing for evaluation of label-constrained reachability queries. In: SIGMOD, pp 345–358 (2017)
186. Wadhwa, S., Prasad, A., Ranu, S., Bagchi, A., Bedathur, S.: Efficiently answering regular simple path queries on large labeled networks. In: SIGMOD, pp 1463–1480 (2019)
187. Wang, C., Song, Y., Li, H., Sun, Y., Zhang, M., Han, J.: Distant meta-path similarities for text-based heterogeneous information networks. In: CIKM, pp 1629–1638 (2017)
188. Wang, C., Sun, Y., Song, Y., Han, J., Song, Y., Wang, L., Zhang, M.: RelSim: Relation similarity search in schema-rich heterogeneous information networks. In: SDM, pp 621–629 (2016)
189. Wang, H., Aggarwal, C.C.: A survey of algorithms for keyword search on graph data. In: Managing and Mining Graph Data, pp 249–273, Springer (2010)
190. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Trans. Knowl. Data Eng. **29**(12), 2724–2743 (2017)
191. Wang, X., Smalter, A.M., Huan, J., Lushington, G.H.: G-hash: towards fast kernel-based similarity search in large graph databases. In: EDBT, pp 472–480 (2009)
192. Wang, X., Wang, S., Xin, Y., Yang, Y., Li, J., Wang, X.: Distributed pregel-based provenance-aware regular path query processing on RDF knowledge graphs. World Wide Web **23**(3), 1465–1496 (2020)
193. Wang, Z., Fan, Q., Wang, H., Tan, K.-L., Agrawal, D., Abbadi, A.E.: Pagrol: Parallel graph OLAP over large-scale attributed graphs. In: ICDE, pp 496–507 (2014)
194. Wang, Z., Wang, W., Wang, C., Gu, X., Li, B., Meng, D.: Community focusing: Yet another query-dependent community detection. In: AAAI, pp 329–337 (2019)
195. Wood, P.T.: Query languages for graph databases. SIGMOD Rec. **41**(1), 50–60 (2012)
196. Wu, Y., Yang, S., Srivatsa, M., Iyengar, A., Yan, X.: Summarizing answer graphs induced by keyword queries. PVLDB **6**(14), 1774–1785 (2013)
197. Wu, Y., Yang, S., Yan, X.: Ontology-based subgraph querying. In: ICDE, pp 697–708 (2013)
198. Wu, Y., Jin, R., Li, J., Zhang, X.: Robust local community detection: On free rider effect and its elimination. PVLDB **8**(7), 798–809 (2015)
199. Xin, Y., Wang, X., Jin, D., Wang, S.: Distributed efficient provenance-aware regular path queries on large RDF graphs. In: DASFAA, pp 766–782 (2018)
200. Xin, W., Lele, C., Qiang, X., Yajun, Y., Jianxin, L., Junhu, W., Yunpeng, C.: Efficient Subgraph Matching on Large RDF Graphs Using MapReduce. Data Sci. Eng. **4**(1), 24–43 (2019)
201. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest LCAs in XML databases. In: SIGMOD, pp 537–538 (2005)
202. Yan, J., Yin, X.-C., Lin, W., Deng, C., Zha, H., Yang, X.: A short survey of recent advances in graph matching. In: ICMR, pp 167–174 (2016)
203. Yan, X., He, B., Zhu, F., Han, J.: Top-k aggregation queries over large networks. In: ICDE, pp 377–380 (2010)
204. Yan, X., Yu, P.S., Han, J.: Graph indexing: A frequent structure-based approach. In: SIGMOD, pp 335–346 (2004)
205. Yan, X., Yu, P.S., Han, J.: Substructure similarity search in graph databases. In: SIGMOD, pp 766–777 (2005)
206. Yang, M.-C., Duan, N., Zhou, M., Rim, H.-C.: Joint relational embeddings for knowledge-based question answering. In: EMNLP, pp 645–650 (2014)
207. Yang, M., Ding, B., Chaudhuri, S., Chakrabarti, K.: Finding patterns in a knowledge base using keywords to compose table answers. PVLDB **7**(14), 1809–1820 (2014)
208. Yang, S., Han, F., Wu, Y., Yan, X.: Fast top-k search in knowledge graphs. In: ICDE, pp 990–1001 (2016)
209. Yang, Y., Agrawal, D., Jagadish, H.V., Tung, A.K.H., Wu, S.: An efficient parallel keyword search engine on knowledge graphs. In: ICDE, pp 338–349 (2019)
210. Yang, Z., Fu, A.W.-C., Liu, R.: Diversified top-k subgraph querying in a large graph. In: SIGMOD, pp 1167–1182 (2016)

211. Yao, X., Durme, B.V.: Information extraction over structured data: Question answering with freebase. In: ACL, pp 956–966 (2014)
212. Yih, W.-T., He, X., Meek, C.: Semantic parsing for single-relation question answering. In: ACL, pp 643–648 (2014)
213. Yih, W.-T., Ma, H.: Question answering with knowledge base, Web and beyond. In: NAACL-HLT, pp 8–10 (2016)
214. Yin, M., Wu, B., Zeng, Z.: HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis. In: DOLAP, pp 137–144 (2012)
215. Yu, J.X., Cheng, J.: Graph reachability queries: A survey. In: Managing and Mining Graph Data, pp 181–215, Springer (2010)
216. Yu, J.X., Qin, L., Chang, L.: Keyword search in relational databases: A survey. IEEE Data Eng. Bull. **33**(1), 67–78 (2010)
217. Yu, X., Sun, Y., Norick, B., Mao, T., Han, J.: User guided entity similarity search using meta-path selection in heterogeneous information networks. In: CIKM, pp 2025–2029 (2012)
218. Yuan, Y., Lian, X., Chen, L., Yu, J.X., Wang, G., Sun, Y.: Keyword search over distributed graphs with compressed signature. IEEE Trans. Knowl. Data Eng. **29**(6), 1212–1225 (2017)
219. Yuan, Y., Wang, G., Chen, L., Wang, H.: Efficient keyword search on uncertain graph data. IEEE Trans. Knowl. Data Eng. **25**(12), 2767–2779 (2013)
220. Yuan, Y., Wang, G., Wang, H., Chen, L.: Efficient subgraph search over large uncertain graphs. PVLDB **4**(11), 876–886 (2011)
221. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: On approximating graph edit distance. PVLDB **2**(1), 25–36 (2009)
222. Zhang, J., Tang, J., Ma, C., Tong, H., Jing, Y., Li, J., Luyten, W., Moens, M.-F.: Fast and flexible top-*k* similarity search on large networks. ACM Trans. Inf. Syst. **36**(2), 13:1–13:30 (2017)
223. Zhang, M., Wang, J., Wang, W.: HeteRank: A general similarity measure in heterogeneous information networks by integrating multi-type relationships. Inf. Sci. **453**, 389–407 (2018)
224. Zhang, S., Li, S., Yang, J.: GADDI: distance index based subgraph matching in biological networks. In: EDBT, pp 192–203 (2009)
225. Zhang, S., Yang, J., Jin, W.: SAPPER: Subgraph indexing and approximate matching in large graphs. PVLDB **3**(1), 1185–1194 (2010)
226. Zhang, S., Li, J., Gao, H., Zou, Z.: A novel approach for efficient supergraph query processing on graph databases. In: EDBT, pp 204–215 (2009)
227. Zhang, W., Lin, X., Zhang, Y., Zhu, K., Zhu, G.: Efficient probabilistic supergraph search. IEEE Trans. Knowl. Data Eng. **28**(4), 965–978 (2016)
228. Zhang, X., Özsu, M.T.: Correlation constraint shortest path over large multi-relation graphs. PVLDB **12**(5), 488–501 (2019)
229. Zhang, X., Feng, Z., Wang, X., Rao, G., Wu, W.: Context-free path queries on RDF graphs. In: ISWC, pp 632–648 (2016)
230. Zhao, P., Han, J.: On graph query optimization in large networks. PVLDB **3**(1), 340–351 (2010)
231. Zhao, P., Li, X., Xin, D., Han, J.: Graph cube: on warehousing and OLAP multidimensional networks. In: SIGMOD, pp 853–864 (2011)
232. Zhao, X., Xiao, C., Lin, X., Liu, Q., Zhang, W.: A partition-based approach to structure similarity search. PVLDB **7**(3), 169–180 (2013)
233. Zheng, W., Lian, X., Zou, L., Hong, L., Zhao, D.: Online subgraph skyline analysis over knowledge graphs. IEEE Trans. Knowl. Data Eng. **28**(7), 1805–1819 (2016)
234. Zheng, W., Zou, L., Lian, X., Hong, L., Zhao, D.: Efficient subgraph skyline search over large graphs. In: CIKM, pp 1529–1538 (2014)
235. Zhou, Y., Huang, J., Li, H., Sun, H., Peng, Y., Xu, Y.: A semantic-rich similarity measure in heterogeneous information networks. Knowl.-Based Syst. **154**, 32–42 (2018)
236. Zhu, G., Lin, X., Zhu, K., Zhang, W., Yu, J.X.: TreeSpan: efficiently computing similarity all-matching. In: SIGMOD, pp 529–540 (2012)
237. Zhu, Q., Cheng, H., Huang, X.: I/O-efficient algorithms for top-k nearest keyword search in massive graphs. VLDB J. **26**(4), 563–583 (2017)
238. Zhu, Y., Qin, L., Yu, J.X., Cheng, H.: Finding top-k similar graphs in graph databases. In: EDBT, pp 456–467 (2012)
239. Zhu, Y., Zhang, Q., Qin, L., Chang, L., Yu, J.X.: Querying cohesive subgraphs by keywords. In: ICDE, pp 1324–1327 (2018)
240. Zhu, Y., Zhang, Q., Qin, L., Chang, L., Yu, J.X.: Cohesive subgraph search using keywords in large networks. IEEE Trans. Knowl. Data Eng. (2020)

241. Zou, L., Chen, L., Özsu, M.T.: Distance-join: Pattern match query in a large graph database. PVLDB **2**(1), 886–897 (2009)
242. Zou, L., Huang, R., Wang, H., Yu, J.X., He, W., Zhao, D.: Natural language question answering over RDF: a graph data driven approach. In: SIGMOD, pp 313–324 (2014)
243. Zou, L., Xu, K., Yu, J.X., Chen, L., Xiao, Y., Zhao, D.: Efficient processing of label-constraint reachability queries in large graphs. Inf. Syst. **40**, 47–66 (2014)

Affiliations

Yanhao Wang[1] · Yuchen Li[2] · Ju Fan[3] · Chang Ye[2] · Mingke Chai[3]

Yanhao Wang
yanhao.wang@helsinki.fi

Yuchen Li
yuchenli@smu.edu.sg

Chang Ye
changye.2020@phdcs.smu.edu.sg

Mingke Chai
cmk@ruc.edu.cn

[1] Department of Computer Science, University of Helsinki, Helsinki, 00560, Finland

[2] School of Information Systems, Singapore Management University, Singapore, 178902, Singapore

[3] Key Lab of Data Engineering and Knowledge Engineering (DEKE) and Information School, Renmin University of China, Beijing, 100872, China