

Summarizing Answer Graphs Induced by Keyword Queries

Yinghui Wu¹ Shengqi Yang¹ Mudhakar Srivatsa² Arun Iyengar² Xifeng Yan¹
¹University of California Santa Barbara ²IBM Research
{yinghui, sqyang, xyan}@cs.ucsb.edu {msrivats, aruni}@us.ibm.com

ABSTRACT

Keyword search has been popularly used to query graph data. Due to the lack of structure support, a keyword query might generate an excessive number of matches, referred to as “answer graphs”, that could include different relationships among keywords. An ignored yet important task is to group and summarize answer graphs that share similar structures and contents for better query interpretation and result understanding. This paper studies the summarization problem for the answer graphs induced by a keyword query Q . (1) A notion of *summary graph* is proposed to characterize the summarization of answer graphs. Given Q and a set of answer graphs \mathcal{G} , a summary graph preserves the relation of the keywords in Q by summarizing the paths connecting the keywords nodes in \mathcal{G} . (2) A quality metric of summary graphs, called *coverage ratio*, is developed to measure information loss of summarization. (3) Based on the metric, a set of summarization problems are formulated, which aim to find minimized summary graphs with certain coverage ratio. (a) We show that the complexity of these summarization problems ranges from PTIME to NP-complete. (b) We provide exact and heuristic summarization algorithms. (4) Using real-life and synthetic graphs, we experimentally verify the effectiveness and the efficiency of our techniques.

1. INTRODUCTION

Keyword queries have been widely used for querying graph data, such as information networks, knowledge graphs, and social networks [37]. A keyword query Q is a set of keywords $\{k_1, \dots, k_n\}$. The evaluation of Q over graphs is to extract data related with the keywords in Q [5, 37].

Various methods were developed to process keyword queries. In practice, these methods typically generate a set of graphs \mathcal{G} induced by Q . Generally speaking, (a) the keywords in Q correspond to a set of nodes in these graphs, and (b) a path connecting two nodes related with keywords k_1, k_2 in Q suggests how the keywords are connected, *i.e.*, the relationship between the keyword pair (k_1, k_2) . We refer to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy.

Proceedings of the VLDB Endowment, Vol. 6, No. 14
Copyright 2013 VLDB Endowment 2150-8097/13/14... \$ 10.00.

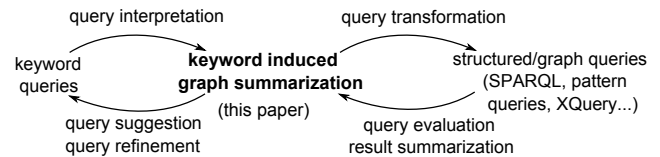


Figure 1: Keyword induced graph summarization – bridging keyword query and graph query

these graphs as *answer graphs* induced by Q . For example, (1) a host of work on keyword querying [13, 14, 17, 20, 27, 37] defines the query results as answer graphs; (2) keyword query interpretation [3, 35] transforms a keyword query into graph structured queries via the answer graphs extracted for the keyword; (3) result summarization [16, 22] generates answer graphs as *e.g.*, “snippets” for keyword query results.

Nevertheless, keyword queries usually generate a great number of answer graphs (as intermediate or final results) that are too many to inspect, due to the sheer volume of data. This calls for effective techniques to summarize answer graphs with representative structures and contents. Better still, the summarization of answer graphs can be further used for a range of important keyword search applications.

Enhance Search with Structure. It is known that there is an usability-expressivity tradeoff between keyword query and graph query [33] (as illustrated in Fig. 1). For searching graph data, keyword queries are easy to formulate; however, they might be ambiguous due to the lack of structure support. In contrast, graph queries are more accurate and selective, but difficult to describe. Query interpretation targets the trade-off by constructing graph queries, *e.g.*, SPARQL [31], to find more accurate results. Nevertheless, there may exist many interpretations as answer graphs for a single keyword query [9]. A summarization technique may generate a small set of summaries, from which graph queries can be induced. That is, a user can first submit keyword queries and then pick up the desired graph queries, thus taking advantage of both keyword query and graph query.

Improve Result Understanding and Query Refinement. Due to query ambiguity and the sheer volume of data, keyword query evaluation often generates a large number of results [16, 19]. This calls for effective result summarization, such that users may easily understand the results without checking them one by one. Moreover, users may come up with better queries that are less ambiguous, by inspecting the connection of the keywords reflected in the summary. Based on the summarization result, efficient query refinement and suggestion [23, 30] may also be proposed.

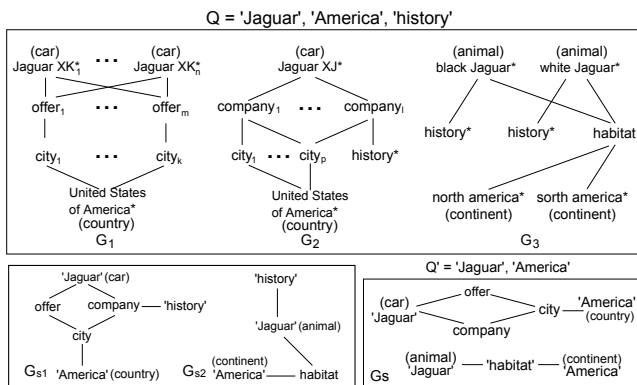


Figure 2: Keyword query over a knowledge graph

Example 1: Consider a keyword query $Q = \{ \text{Jaguar, America, history} \}$ issued over a knowledge graph. Suppose there are three graphs G_1 , G_2 and G_3 induced by the keywords in Q as *e.g.*, query results [20,27], as shown in Fig. 2. Each node in an answer graph has a type, as well as its unique id. It is either (a) a keyword node marked with '*' (*e.g.*, Jaguar XK*) which corresponds to a keyword (*e.g.*, Jaguar), or (b) a node connecting two keyword nodes.

The induced graphs for Q illustrate different relations among the same set of keywords. For example, G_1 suggests that “Jaguar” is a brand of cars with multiple offers in many cities of USA, while G_3 suggests that “Jaguar” is a kind of animals found in America. To find out the answers the users need, reasonable graph structured queries are required for more accurate searching [3]. To this end, one may construct a *summarization* over the answer graphs. Two such summaries can be constructed as G_{s1} and G_{s2} , which suggest two graph queries where “Jaguar” refers to a brand of car, and a kind of animal, respectively. Better still, by summarizing the relation between two keywords, more useful information can be provided to the users. For example, G_{s1} suggests that users may search for “offers” and “company” of “Jaguar”, as well as their locations.

Assume that the user wants to find out how “Jaguar” and “America” are related in the search results. This requires a summarization that only considers the connection between the nodes containing the keywords. Graph G_s depicts such a summarization: it shows that (1) “Jaguar” relates to “America” as a type of car produced and sold in cities of USA, or (2) it is a kind of animal living in the continents of America.

Moreover, in practice one often place a budget for summarizations [12,32]. This calls for quality metrics and techniques for concise summaries that illustrate the connection information between keywords as much as possible. \square

The above example suggests that we may summarize answer graphs \mathcal{G} induced by a keyword query Q , to help keyword query processing. We ask the following questions. (1) How to define “query-aware” summaries of \mathcal{G} in terms of Q ? (2) How to characterize the quality of a summary for Q ? (3) How to efficiently identify good summaries under a budget?

In this paper we study the above problems for summarizing keyword induced answer graphs.

(1) We formulate the concept of answer graphs for a keyword query Q (Section 2). To characterize the summarization for answer graphs, we propose a notion of *summary graph* (Section 2). Given Q and \mathcal{G} , a summary graph captures the relationship among the keywords from Q in \mathcal{G} .

(2) We introduce quality metrics for summary graphs (Section 3). One is defined as the size of a summary graph, and the other is based on *coverage ratio* α , which measures the number of keyword pairs a summary graph can *cover* by summarizing pairwise relationships in \mathcal{G} .

Based on the quality metrics, we introduce two *summarization* problems (Section 3). Given Q and \mathcal{G} , (a) the α -summarization problem is to find a minimum summary graph with a certain coverage ratio α ; we consider 1-summarization problem as its special case where $\alpha = 1$; (b) the K summarization problem is to identify K summary graphs for \mathcal{G} , where each one summarizes a subset of answer graphs in \mathcal{G} . We show that the complexity of these problems ranges from PTIME to NP-complete. For the NP-hard problems, they are also hard to approximate.

(3) We propose exact and heuristic algorithms for the summarization problems. (a) For 1-summarization, we present an exact, quadratic-time algorithm to find a minimum 1-summary (Section 4). For a given keyword query, it is to identify a set of “redundant” (resp. “equivalent”) nodes in \mathcal{G} for Q , and construct the summary by removing (resp. merging) these nodes. (b) We provide heuristic algorithms for the α -summarization (Section 4) and k summarization problems (Section 5), respectively. These algorithms greedily select and summarize answer graphs with the minimum estimated cost in terms of size and coverage.

(4) We experimentally verify the effectiveness and efficiency of our summarization techniques using both synthetic data and real-life datasets. We find that our algorithms effectively summarize the answer graphs. For example, they generate summary graphs that cover every pair of keywords with size in average 24% of the answer graphs. They also scale well with the size of the answer graphs. These effectively support summarization over answer graphs.

Related Work. We categorize related work as follows.

Graph compression and summarization. Graph summarization is to (approximately) describe graph data with small amount of information. (1) Graph compression [26] uses MDL principle to compress graphs with bounded error. However, the goal is to reduce space cost while the original graph can be restored, by using auxiliary structures as “corrections”. (2) Summarization techniques are proposed based on (a) bisimulation equivalence relation [25], or (b) relaxed bisimulation relation that preserves paths with length up to K [18,25]. Simulation based minimization [2] reduces a transition system based on simulation equivalence relation. These work preserve paths for every pair of nodes, *i.e.*, all-pair connectivity, which can be too restrictive to generate concise summaries for keyword queries. (3) Summary techniques in [34,38] enable flexible summarization over graphs with multiple node and edge attributes, while the path information is approximately preserved, controlled by additional parameters, *e.g.*, participation ratio [34].

In contrast to these work, we find concise summaries that preserve relationships among keywords, rather than all-pair connectivity [18,25] or entire original graph [26]. Moreover, in contrast to [26,34], these summaries require no auxiliary structure for preserving the relationships.

Relation discovery. Relation discovery is to extract the relations between keywords over a (single) graph [8,27,32]. The problem is studied for a single entity [32], a pair of

keywords [8], and for general keyword queries [27] which finds top ranked subgraphs in terms of *e.g.*, relevance. In contrast to these studies which focus more on searching, we propose summarization techniques over the extracted results. Moreover, users can place constraints, *e.g.*, coverage ratio, to identify summaries for a part or all keyword pairs from the query, which are not addressed in these works.

Graph clustering. A number of graph clustering approaches have also been proposed to group similar graphs [1]. As remarked earlier, these techniques are not query-aware, and may not be directly applied for summarizing query results as graphs [21]. In contrast, we propose algorithms to (1) group answer graphs in terms of a set of keywords, and (2) find best summaries for each group.

Result Summarization. Result summarization over relational databases and XML are proposed to help users understand the query results. [16] generates summaries for XML results as trees, where a snippet is produced for each result tree. This may produce snippets with similar structures that should be grouped for better understanding [21]. To address this issue, [22] clusters the query results based on the classification of their search predicates. Our work differs in that (1) we generate summaries for and as general graphs rather than trees [16], (2) we study how to summarize connections “induced” by keywords, while the main focus of [22] is to identify proper return nodes.

Application Scenarios. There have been a host of studies on processing keyword queries that generate answer graphs. Our work can be applied to these applications.

Keyword queries over graphs. Keyword search over graphs typically return graphs that contain all the keywords [37]. For example, an answer graph as a query result is represented by (1) subtrees for XML data [13, 14], or (2) subgraphs of schema-free graphs [17, 20, 27]. The summarization techniques in our work can be applied in these applications as post-processing, to provide result summarizations [16].

Query interpretation. Keyword query interpretation transforms a keyword query into graph queries, *e.g.*, XPath queries [28], SPARQL queries [31], or formal queries [35] (see [3] for a survey). Keyword query templates for SQL queries are extracted by leveraging schema from relational tables [6]. Closer to our work are [6, 35], which generate top ranked query templates [6] or conjunctive queries [35] for keywords, by summarizing data graphs and schema information. Our work differs from these work in that (1) we generate summaries over answer graphs with coverage and conciseness guarantees, and (2) no schema information is required. The conciseness and information coverage of the summaries is not discussed in [35].

Query expansion. [23] considers generating suggested keyword queries from a set of clustered query results. [30] studies the keyword query expansion that extends the original queries with “surprising words”. Neither considers structured expansions. Our work produces structural summaries that not only include keywords and their relationships, but also a set of highly related nodes and relations, which could provide good suggestions for query refinement (Section 6).

2. ANSWER GRAPHS AND SUMMARIES

In this section, we formulate the concept of answer graphs induced by keyword queries, and their summarizations.

2.1 Keyword Induced Answer Graphs

Answer graphs. Given a keyword query Q as a set of keywords $\{k_1, \dots, k_n\}$ [37], an *answer graph* induced by Q is a connected undirected graph $G = (V, E, L)$, where V is a node set, $E \subseteq V \times V$ is an edge set, and L is a labeling function which assigns, for each node v , a label $L(v)$ and a *unique* identity. In practice, the node labels may represent the type information in *e.g.*, RDF [27], or node attributes [38]. The node identity may represent a name, a property value, a *URI*, *e.g.*, “*dbpedia.org/resource/Jaguar*,” and so on. Each node $v \in V$ is either a *keyword node* that corresponds to a keyword k in Q , or an *intermediate* node on a path between keyword nodes. We denote as v_k a keyword node of k . The keyword nodes and intermediate nodes are typically specified by the process that generates the answer graphs, *e.g.*, keyword query evaluation algorithms [37]. A path connecting two keyword nodes usually suggests a relation, or “connection pattern”, as observed in *e.g.*, [8].

We shall use the following notations. (1) A path from keyword nodes v_k to v'_k is a nonempty *simple* node sequence $\{v_k, v_1, \dots, v_n, v'_k\}$, where v_i ($i \in [1, n]$) are intermediate nodes. The label of a path ρ from v_k to v'_k , denoted as $L(\rho)$, is the concatenation of all the node labels on ρ . (2) The union of a set of answer graphs $G_i = (V_i, E_i, L_i)$ is a graph $G = (V, E, L)$, where $V = \bigcup V_i$, $E = \bigcup E_i$, and each node in V has a unique node id. (3) Given a set of answer graphs \mathcal{G} , we denote as $\text{card}(\mathcal{G})$ the number of the answer graphs \mathcal{G} contains, and $|\mathcal{G}|$ the total number of its nodes and edges. Note that an answer graph does not necessarily contain keyword nodes for all the keywords in Q , as common found in *e.g.*, keyword querying [37].

Example 2: Fig. 2 illustrates a keyword query Q and a set of answer graphs $\mathcal{G} = \{G_1, G_2, G_3\}$ induced by Q . Each node in an answer graph has a label as its type (*e.g.*, *car*), and a unique string as its id (*e.g.*, *Jaguar XK1*).

Consider the answer graph G_1 . (a) The keyword nodes for the keyword *Jaguar* are Jaguar_{XK_i} ($i \in [1, n]$), and the node *United States of America* is a keyword node for *America*. (b) The nodes offer_i ($i \in [1, m]$) and city_j ($j \in [1, k]$) are the intermediate nodes connecting the keyword nodes of *Jaguar* and *America*. (c) A path from *Jaguar* to *USA* passing the nodes offer_1 and city_1 has a label $\{\text{car}, \text{offer}, \text{city}, \text{country}\}$. Note that (1) nodes with different labels (*e.g.*, Jaguar_{XK_1} labeled by “*car*” and black jaguar by “*animal*”) may correspond to the same keyword (*e.g.*, *Jaguar*), and (2) a node (*e.g.*, city_1) may appear in different answer graphs (*e.g.*, G_1 and G_2). \square

2.2 Answer Graph Summarization

Summary graph. A summary graph of \mathcal{G} for Q is an undirected graph $G_s = (V_s, E_s, L_s)$, where V_s and E_s are the node and edge set, and L_s is a labeling function. Moreover, (1) each node $v_s \in V_s$ labeled with $L_s(v_s)$ represents a node set $[v_s]$ from \mathcal{G} , such that (a) $[v_s]$ is either a keyword node set, or an intermediate node set from \mathcal{G} , and (b) the nodes v in $[v_s]$ have the same label $L(v) = L_s(v_s)$. We say v_{s_k} is a *keyword node* for a keyword k , if $[v_{s_k}]$ is a set of keyword nodes of k ; (2) For any path ρ_s between keyword nodes v_{s_1} and v_{s_2} of G_s , there exists a path ρ with the same label of ρ_s from v_1 to v_2 in the *union* of the answer graphs in \mathcal{G} , where $v_1 \in [v_{s_1}]$, $v_2 \in [v_{s_2}]$. Here the path label in G_s is similarly defined as its counterpart in an answer graph.

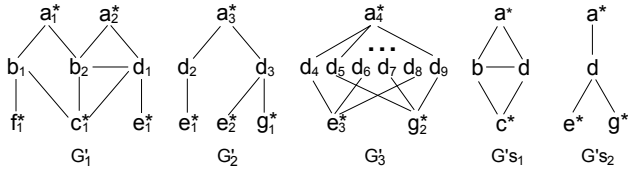


Figure 3: Answer graphs and summary graphs

Hence, a summary graph G_s never introduces “false” paths by definition: if v_{s_1} and v_{s_2} are connected via a path ρ_s in G_s , it suggests that there is a path ρ of the same label connecting two keyword nodes in $[v_{s_1}]$ and $[v_{s_2}]$, respectively, in the union of the answer graphs. It might, however, “lose” information, *i.e.*, not all the labels of the paths connecting two keyword nodes are preserved in G_s .

Example 3: Consider Q and \mathcal{G} from Fig. 2. One may verify that G_{s_1} , G_{s_2} and G_s are summary graphs of \mathcal{G} for Q . Specifically, (1) the nodes Jaguar, history and America are three keyword nodes in G_{s_1} , and the rest nodes are intermediate ones; (2) G_{s_2} contains a keyword node Jaguar which corresponds to keyword nodes {black jaguar, white jaguar} of the same label animal in \mathcal{G} . (3) For any path connecting two keyword nodes (*e.g.*, {Jaguar, offer, city, America}) in G_{s_1} , there is a path with the same label in the union of G_1 and G_2 (*e.g.*, {Jaguar_{XK1}, offer₁, city₁, United States of America}).

As another example, consider the answer graphs G'_1 , G'_2 and G'_3 induced by a keyword query $Q' = \{a, c, e, f, g\}$ in Fig. 3. Each node a_i (marked with * if it is a keyword node) in an answer graph has a label a and an id a_i , similarly for the rest nodes. One may verify the following. (1) Both G'_{s_1} and G'_{s_2} are summary graphs for the answer graph set $\{G'_1, G'_2\}$; while G'_{s_1} (resp. G'_{s_2}) only preserves the labels of the paths connecting keywords a and c (resp. a, e and g). (2) G'_{s_2} is not a summary graph for G'_3 . Although it correctly suggests the relation between keywords (a, e) and (a, g) , it contains a “false” path labeled (e, d, g) , while there is no path in G'_3 with the same label between e_3 and g_2 . □

Remarks. One can readily extend summary graphs to support directed, edge labeled answer graphs by incorporating edge directions and labels into the path label. We can also extend summary graphs for preserving path labels for each answer graph, instead of for the union of answer graphs, by reassigning node identification to answer graphs.

3. QUALITY MEASUREMENT

We next introduce two metrics to measure the quality of summary graphs, based on information coverage and summarization conciseness, respectively. We then introduce a set of summarization problems. To simplify the discussion, we assume that the union of the answer graphs contains keyword nodes for each keyword in Q .

3.1 Coverage Measurement

It is recognized that a summarization should summarize as much information as possible, *i.e.*, to maximize the information coverage [12]. In this context, a summary graph should capture the relationship among the query keywords as much as possible. To capture this, we first present a notion of *keywords coverage*.

Keywords coverage. Given a keyword pair (k_i, k_j) ($k_i, k_j \in Q$ and $k_i \neq k_j$) and answer graphs \mathcal{G} induced by Q , a summary graph G_s covers (k_i, k_j) if for any path ρ from

keyword nodes v_{k_i} to v_{k_j} in the union of the answer graphs in \mathcal{G} , there is a path ρ_s in G_s from v_{s_i} to v_{s_j} with the same label of ρ , where $v_{k_i} \in [v_{s_i}]$, $v_{k_j} \in [v_{s_j}]$. Note that the coverage of a keyword pair is “symmetric” over undirected answer graphs. Given Q and \mathcal{G} , if G_s covers a keyword pair (k_i, k_j) , it also covers (k_j, k_i) .

Coverage ratio. Given a keyword query Q and \mathcal{G} , we define the *coverage ratio* α of a summary graph G_s of \mathcal{G} as

$$\alpha = \frac{2 \cdot M}{|Q| \cdot (|Q| - 1)}$$

where M is the total number of the keyword pairs (k, k') covered by G_s . Note that there are in total $\frac{|Q||Q|-1}{2}$ pairs of keywords from Q . Thus, α measures the information coverage of G_s based on the coverage of the keywords.

We refer to as α -*summary graph* the summary graph for \mathcal{G} induced by Q with coverage ratio α . The coverage ratio measurement favors a summary graph that covers more keyword pairs, *i.e.*, with larger α .

Example 4: Consider Q and \mathcal{G} from Fig. 2. Treating G_{s_1} and G_{s_2} as a single graph G_{s_0} , one may verify that G_{s_0} is a 1-summary graph: for any keyword pair from Q and any path between the keyword nodes in \mathcal{G} , there is a path of the same label in G_{s_0} . On the other hand, G_s is a $\frac{1}{3}$ summary graph for Q : it only covers the keyword pairs (Jaguar, America). Similarly, one may verify that G'_{s_1} (resp. G'_{s_2}) in Fig. 3 is a 0.1-summary graph (resp. 0.3-summary graph), for answer graphs $\{G'_1, G'_2, G'_3\}$ and $Q = \{a, c, e, f, g\}$. □

3.2 Conciseness Measurement

A summary graph should also be concise, without introducing too much detail of answer graphs, as commonly used in information summarization [12, 32].

Summarization size. We define the size of a summary graph G_s , (denoted as $|G_s|$) as the total number of the nodes and edges it has. For example, the summary graph G_{s_1} and G_{s_2} (Fig. 2) are of size 12 and 7, respectively. The smaller a summary graph is, the more concise it is.

Putting the information coverage and conciseness measurements together, We say a summary graph G_s is a *minimum* α -summary graph, if for any other α -summary graph G'_s of \mathcal{G} for Q , $|G_s| \leq |G'_s|$.

Remarks. Bisimulation [11], graph summarization [26, 34] and query suggestion [35] also induce summarized graphs, by grouping similar nodes and edges together for entire graphs, rather than for specified keyword nodes. Moreover, (a) they may not necessarily generate concise summaries; and (b) their summary graphs may introduce “false” paths.

Example 5: Bisimulation relation [11] constraints the node equivalence via a recursively defined neighborhood label equivalence, which is an overkill for concise summaries over keyword relations. For example, the nodes b_1 and b_2 cannot be represented by a single node as in G_{s_1} via bisimulation (Fig. 3), due to different neighborhood. On the other hand, error-tolerant [26], structure-based summaries [34] and schema extraction [35] may generate summary graphs with “false paths”, such as G'_{s_2} for G'_3 . To prevent this, auxiliary structures and parameters are required. In contrast in our work, a summary graph preserves path labels for keywords without any auxiliary structures. □

3.3 Summarization Problems

Based on the quality metrics, we next introduce two summarization problems for keyword induced answer graphs. These problems are to find summary graphs with high quality, in terms of information coverage and conciseness.

Minimum α -Summarization. Given a keyword query Q and its induced answer graphs \mathcal{G} , and a user-specified coverage ratio α , the *minimum α -summarization* problem, denoted as MSUM, is to find an α -summary graph of \mathcal{G} with minimum size. Intuitively, the problem aims to find the smallest summary graph [32] which can cover the keyword pairs no less than user-specified coverage requirement.

The problem is, however, nontrivial.

Theorem 1: MSUM is NP-complete (for decision version) and APX-hard (as an optimization problem). \square

The APX-hard class consists of all problems that cannot be approximated in polynomial time within arbitrary small approximation ratio [36]. We prove the complexity result and provide a heuristic algorithm for MSUM in Section 4.

Minimum 1-summarization. We also consider the problem of finding a summary graph that covers every pair of keywords (k_i, k_j) ($k_i, k_j \in Q$ and $i \neq j$) as concise as possible, i.e., the *minimum 1-summarization problem* (denoted as PSUM). Note that PSUM is a special case of MSUM, by setting $\alpha = 1$. In contrast to MSUM, PSUM is in PTIME.

Theorem 2: Given Q and \mathcal{G} , PSUM is in $O(|Q|^2|\mathcal{G}| + |\mathcal{G}|^2)$ time, i.e., it takes $O(|Q|^2|\mathcal{G}| + |\mathcal{G}|^2)$ time to find a minimum 1-summary graph, where $|\mathcal{G}|$ is the size of \mathcal{G} . \square

We will prove the above result in Section 4.

K Summarization. In practice, users may expect a set of summary graphs instead of a single one, where each summary graph captures the keyword relationships for a set of “similar” answer graphs in terms of path labels. Indeed, as observed in text summarization (e.g., [12]), a summarization should be able to cluster a set of similar objects.

Given Q , \mathcal{G} , and an integer K , the *K summarization* problem (denoted as KSUM) is to find a summary graph set G_S , such that (1) each summary graph $G_{s_i} \in G_S$ is a 1-summary graph of a group of answer graphs $G_{p_i} \subseteq \mathcal{G}$, (2) the answer graph sets G_{p_i} form a K -partition of \mathcal{G} , i.e., $\mathcal{G} = \bigcup G_{p_i}$, and $G_{p_i} \cap G_{p_j} = \emptyset$ ($i, j \in [1, K]$, $i \neq j$); and (3) the total size of G_S , i.e., $\sum_{G_{s_i} \in G_S} |G_{s_i}|$ is minimized. The KSUM problem can also be extended to support α -summarization.

The following result tells us that the problem is hard to approximate. We will prove the result in Section 5, and provide a heuristic algorithm for KSUM.

Theorem 3: KSUM is NP-complete and APX-hard. \square

Remarks. The techniques for MSUM and KSUM can be used in a host of applications. (a) The α -summaries from MSUM can be used to suggest (structured) keyword queries [3], as well as graph (pattern) queries [7,31,35]. The intermediate nodes in the summaries also benefit reasonable query expansion [30]. (b) In practice the answer graphs can be too many to inspect. The techniques for KSUM naturally serve as post-processing for result summarizations [16]. Better still, KSUM also provides a reasonable clustering for answer graphs [12]. The generated K summaries can further be used for query expansion based on clustered results [23].

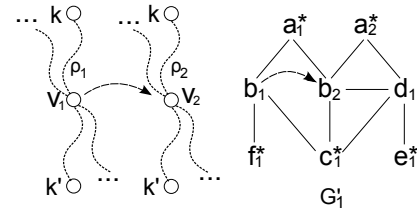


Figure 4: Dominance relation: $(v_1, v_2) \in R_{\leq}$

While determining the optimal value of α and K remain to be open issues, α can be usually set according to e.g., “budget” of comprehension [32], and K can be determined following empirical rules [24] or information theory.

4. COMPUTING α -SUMMARIZATION

In this section we investigate the α -summarization problem. We first investigate PSUM in Section 4.1, as a special case of MSUM. We then discuss MSUM in Section 4.2.

4.1 Computing 1-Summary Graphs

To show Theorem 2, we characterize the 1-summary graph with a sufficient and necessary condition. We then provide an algorithm to check the condition in polynomial time.

We first introduce the notion of dominance relation.

Dominance relation. The *dominance relation* $R_{\leq}(k, k')$ for keyword pair (k, k') over an answer graph $G = (V, E, L)$ is a binary relation over the intermediate nodes of G , such that for each node pair $(v_1, v_2) \in R_{\leq}(k, k')$, (1) $L(v_1) = L(v_2)$, and (2) for any path ρ_1 between keyword node pair v_{k_1} of k and v_{k_2} of k' passing v_1 , there is a path ρ_2 with the same label between two keyword nodes v'_{k_1} of k and v'_{k_2} of k' passing v_2 . We say v_2 *dominates* v_1 w.r.t. (k, k') ; moreover, v_1 is *equivalent* to v_2 if they dominate each other. In addition, two keyword nodes are equivalent if they have the same label, and correspond to the same keyword.

The dominance relation is as illustrated in Fig. 4. Intuitively, (1) $R_{\leq}(k, k')$ captures the nodes that are “redundant” in describing the relationship between a keyword pair (k, k') in G ; (2) moreover, if two nodes are equivalent, they play the same “role” in connecting keywords k and k' , i.e., they cannot be distinguished in terms of path labels. For example, when the keyword pair (a, c) is considered in G'_1 , the node b_1 is dominated by b_2 , as illustrated in Fig. 4.

Remarks. The relation R_{\leq} is similar to the *simulation* relation [2,15], which computes node similarity over the entire graph by neighborhood similarity. In contrast to simulation, R_{\leq} captures dominance relation induced by the paths connecting keyword nodes only, and only consider intermediate nodes. For example, the node b_1 and b_2 is not in a simulation relation in G'_1 , unless the keyword pair (a, c) is considered (Fig. 4). We shall see that this leads to effective summarizations for specified keyword pairs.

Sufficient and necessary condition. We now present the sufficient and necessary condition, which shows the connection between R_{\leq} and a 1-summary graph.

Proposition 4: Given Q and \mathcal{G} , a summary graph G_s is a minimum 1-summary graph for \mathcal{G} and Q , if and only if for each keyword pair (k, k') from Q , (a) for each intermediate node v_s in G_s , there is a node v_i in $[v_s]$, such that for any other node v_j in $[v_s]$, $(v_j, v_i) \in R_{\leq}(k, k')$; and (b) for any intermediate nodes v_{s_1} and v_{s_2} in G_s with same label and any nodes $v_1 \in [v_{s_1}]$, $v_2 \in [v_{s_2}]$, $(v_2, v_1) \notin R_{\leq}(k, k')$. \square

Proof sketch: We prove Proposition 4 as follows.

(1) We first proof by contradiction that G_s is a 1-summary graph if and only if Condition (a) holds. Assume G_s is a 1-summary graph while Condition (a) does not hold. Then there exists an intermediate node v_s , and two nodes v_i and v_j that cannot dominate each other. Thus, there must exist two paths in the union of answer graphs as $\rho = \{v_1, \dots, v_i, v_{i+1}, \dots, v_m\}$ and $\rho' = \{v'_1, \dots, v_j, v_{j+1}, \dots, v_n\}$ with different labels, for a keyword pair (k, k') . Since v_i, v_j is merged as v_s in G_s , there exists, *w.l.o.g.*, a false path in G_s as ρ'' with label $L(v_1) \dots L(v_i)L(v_{j+1}) \dots L(v_m)$, which contradicts the assumption that G_s is a 1-summary graph. Now assume Condition (a) holds while G_s is not a 1-summary graph. Then there at least exists a path from keywords k to k' that is not in G_s . Thus, there exists at least an intermediate node v_s on the path with $[v_s]$ in G_s which contains two nodes that cannot dominate each other. This contradicts the assumption that Condition (a) holds.

(2) For the summary minimization, we show that Conditions (a) and (b) together guarantee if there exists a 1-summary G'_s where $|G'_s| \leq |G_s|$, there exists a one to one function mapping each node (resp. edge) in G'_s to a node (resp. edge) in G_s , *i.e.*, $|G_s| = |G'_s|$. Hence, G_s is a minimum 1-summary graph by definition. \square

We next present an algorithm for PSUM following the sufficient and necessary condition, in polynomial time.

Algorithm. Fig. 5 shows the algorithm, denoted as pSum. It has the following two steps.

Initialization (lines 1-4). pSum first initializes an empty summary graph G_s (line 1). For each keyword pair (k, k') from Q , pSum computes a “connection” graph of (k, k') induced from \mathcal{G} (line 2-3). Let G be the union of the answer graphs in \mathcal{G} . A connection graph of (k, k') is a subgraph of G induced by (1) the keyword nodes of k and k' , and (2) the intermediate nodes on the paths between the keyword nodes of k and those of k' . Once $G_{(k, k')}$ is computed, pSum sets G_s as the union graph of G_s and $G_{(k, k')}$ (line 4).

Reducing (lines 5-7). pSum then constructs a summary graph by removing nodes and edges from G_s . It computes the dominance relation R_{\leq} by invoking a procedure DomR, which removes the nodes v as well as the edges connected to them, if they are dominated by some other nodes (line 5). It next merges the nodes in G_s that have dominate relation, *i.e.*, line 6 (as defined in 4(a)), into a set $[v_s]$, until no more nodes in G_s can be merged. For each set $[v_s]$, a new node v_s as well as its edges connected to other nodes are created. G_s is then updated with the new nodes and edges, and is returned as a minimum 1-summary graph (line 7).

Procedure DomR. Similar as the process to compute a simulation [15], DomR extends the process to undirected graphs. For each node v in G_s , DomR initializes a dominant set $[v]$, as $\{v' | L(v') = L(v)\}$ (lines 1-2). For each edge $(u, v) \in G_s$, it identifies the neighborhood set of u (resp. v) as $N(u)$ (resp. $N(v)$), and removes the nodes that are not in $N(v)$ (resp. $N(u)$) from $[u]$ (resp. $[v]$) (lines 4). Indeed, a node $u' \in [u]$ cannot dominant u if $u' \notin N(v)$, since a path connecting two keyword nodes passing edge (u, v) contains “ $L(u)L(v)$ ” in its label, while for u' , such path does not exist. The process repeats until no changes can be made (lines 3-4). R_{\leq} is then collected from the dominant sets and returned (line 5-7).

Input: A keyword query Q , an answer graph set \mathcal{G} .

Output: A minimum 1-summary graph G_s .

1. Initialize $G_s = \emptyset$;
2. **for each** keyword pair (k, k') ($k, k' \in Q, k \neq k'$) **do**
3. build $G_{(k, k')}$ as an induced connection graph of (k, k') ;
4. merge G_s with $G_{(k, k')}$;
5. $R_{\leq} := \text{DomR}(G_s)$; remove dominated nodes from G_s ;
6. merge each v_{s_1}, v_{s_2} in G_s where there is a node $v_1 \in [v_{s_1}]$ such that for $\forall v_2 \in [v_{s_2}], (v_2, v_1) \in R_{\leq}(k, k')$;
7. **return** G_s ;

Procedure DomR

Input: a graph G_s, \mathcal{G} ;

Output: the dominance relation R_{\leq} over G_s .

1. **for each** node v in G_s **do**
2. dominant set $[v] = \{v' | L(v') = L(v)\}$;
3. **while** $[v]$ is changed for some v **do**
4. **for each** edge (u, v) in G **do**
 $[u] = [u] \cap N([v]); [v] = [v] \cap N([u])$;
5. **for each** v and $v' \in [v]$ **do**
6. $R_{\leq} = R_{\leq} \cup \{(v, v')\}$;
7. **return** R_{\leq} ;

Figure 5: Algorithm pSum

Analysis. pSum correctly returns a summary graph G_s . Indeed, G_s is initialized as the union of the connection graphs, which is a summary graph (lines 2-4). Each time G_s is updated, pSum keeps the invariants that G_s remains to be a summary graph. When pSum terminates, one may verify that the sufficient and necessary condition as in Proposition 4 is satisfied. Thus, the correctness of pSum follows.

It takes $O(|Q|^2|\mathcal{G}|)$ to construct G_s as the union of the connection graphs for each keyword pairs (lines 2-4). It takes DomR in total $O(|\mathcal{G}|^2)$ time to compute R_{\leq} . To see this, observe that (a) it takes $O(|\mathcal{G}|^2)$ time to initialize the dominant sets (line 1), (b) during each iteration, once a node is removed from $[u]$, it will no longer be put back, *i.e.*, there are in total $|G_s|^2$ iterations, and (c) the checking at line 4 can be done in constant time, by looking up a dynamically maintained map recording $|[u] \setminus N(v)|$ for each edge (u, v) , leveraging the techniques in [15]. Thus, the total time complexity of pSum is in $O(|Q|^2|\mathcal{G}| + |\mathcal{G}|^2)$.

Theorem 2 follows from the above analysis.

Example 6: Recall the query Q and the answer graph set \mathcal{G} in Fig. 2. The algorithm pSum constructs a minimum 1-summary graph G_s for \mathcal{G} as follows. It initializes G_s as the union of the connection graphs for the keyword pairs in Q , which is the union graph of G_1, G_2 and G_3 . It then invokes procedure DomR, which computes dominance sets for each intermediate node in G_s , partly shown as follows ($k < p$).

Nodes in G_s	dominance sets
offer	$\{\text{offer}_i\} (i \in [1, m])$
city	$\{\text{city}_i\} (i \in [1, k]), \{\text{city}_j\} (j \in [k+1, p])$
company	$\{\text{company}_i\} (i \in [1, l-1]), \{\text{company}_l\}$

pSum then reduces G_s by removing dominated nodes and merging equivalent nodes until no change can be made. For example, (1) company_x ($x \in [1, l-1]$) are removed, as all are dominated by company_l ; (2) all the offer nodes are merged as a single node, as they dominate each other. G_s is then updated as the union of G_{s_1} and G_{s_2} (Fig. 2). \square

From Theorem 2, the result below immediately follows.

Corollary 5: *It is in $O(|S||\mathcal{G}| + |\mathcal{G}|^2)$ to find a minimum 1-summary graph of \mathcal{G} for a given keyword pair set S . \square*

Indeed, pSum can be readily adapted for specified keyword pair set S , by specifying G_s as the union of the connection graphs induced by S (line 4). The need to find 1-summary graphs for specified keyword pairs is evident in the context of *e.g.*, relation discovery [8], where users may propose specified keyword pairs to find their relationships in graph data.

4.2 Minimum α -summarization

We next investigate the MSUM problem: finding the minimum α -summarization. We first prove Theorem 1, *i.e.*, the decision problem for MSUM is NP-complete. Given Q , a set of answer graphs \mathcal{G} induced by Q , a coverage ratio α , and a size bound B , the decision problem of MSUM is to determine if there exists a α -summary graph G_s with size no more than B . Observe that MSUM is equivalent to the following problem (denoted as MSUM*): find an m -element set $S_m \subseteq S$ from a set of keyword pairs S , such that $|G_s| \leq B$, where (a) $m = \alpha \cdot \frac{|Q||Q-1|}{2}$, (b) $S = \{(k, k') | k, k' \in Q, k \neq k'\}$, and (c) G_s is the minimum 1-summary graph for \mathcal{G} and S_m . It then suffices to show MSUM* is NP-complete.

Complexity. We show that MSUM* is NP-complete as follows. (1) MSUM* is in NP, since there exists a polynomial time algorithm to compute G_s for a keyword pair set S , and determine if $|G_s| \leq B$ (Corollary 5). (2) To show the lower bound, we construct a reduction from the maximum coverage problem, a known NP-complete problem [10]. Given a set X and a set T of its subsets $\{T_1, \dots, T_n\}$, as well as integers K and N , the problem is to find a set $T' \subseteq T$ with no more than K subsets, where $|\bigcup T' \cap X| \geq N$. Given an instance of maximum coverage, we construct an instance of MSUM* as follows. (a) For each element $x_i \in X$, we construct an intermediate node v_i . (b) For each set $T_j \in T$, we introduce a keyword pair (k_{T_j}, k'_{T_j}) , and construct an answer graph G_{T_j} which consists of edges (k_{T_j}, v_i) and (v_i, k'_{T_j}) , for each v_i corresponding to $x_i \in T_j$. We set S as all such (k_{T_j}, k'_{T_j}) pairs. (c) We set $m = |T| - K$, and $B = |X| - N$. One may verify that there exists at most K subsets that covers at least N elements in X , if and only if there exists a 1-summary graph that covers at least $|S| - K$ keyword pairs, with size at most $2 * (|X| - N + m)$. Thus, MSUM* is NP-hard. Putting (1) and (2) together, MSUM* is NP-complete.

The APX-hardness can be proved by constructing an approximation ratio-preserving reduction [36] from the weighted maximum coverage problem, a known APX-hard problem, via a similar transformation as discussed above.

The above analysis completes the proof of Theorem 1.

The APX-hardness of MSUM indicates that it is unlikely to find a polynomial-time algorithm for MSUM with every fixed approximation ratio [36]. Instead, we resort to an efficient heuristic algorithm, mSum.

A greedy heuristic algorithm. Given Q and \mathcal{G} , mSum (1) dynamically maintains a set of connection graphs \mathcal{G}_C , and (2) greedily selects a keyword pair (k, k') and its connection graph G_c , such that the following “merge cost” is minimized:

$$\delta_r(\mathcal{G}_C, G_c) = |G_s(\mathcal{G}_C \cup \{G_c\})| - |G_s(\mathcal{G}_C)|$$

where $G_s(\mathcal{G}_C \cup \{G_c\})$ (resp. $G_s(\mathcal{G}_C)$) is the 1-summary graph of the answer graph set $\mathcal{G}_C \cup \{G_c\}$ (resp. (\mathcal{G}_C)). Intuitively, the

Input: A keyword query Q , a set of answer graphs \mathcal{G} , a coverage ratio α

Output: An α -summary graph G_s .

1. Initialize G_s ; Set $\mathcal{G}_C := \emptyset$;
2. **for each** pair (k, k') where $k, k' \in Q$ **do**
3. compute connection graph $G_{c(k, k')}$; $\mathcal{G}_C := \mathcal{G}_C \cup \{G_{c(k, k')}\}$;
4. **while** $G_s \neq \emptyset$ **do**
5. **for each** $G_{c(k, k')} \in \mathcal{G}_C$ with minimum merge cost **do**
6. **if** $G_s = \emptyset$ **then** $G_s := \text{pSum}((k, k'), \mathcal{G})$;
7. **else** $\text{merge}(G_s, G_{c(k, k')})$;
8. $\mathcal{G}_C := \mathcal{G}_C \setminus \{G_{c(k, k')}\}$;
9. **if** m connection graphs are merged **then break** ;
10. **for each** $G_c \in \mathcal{G}_C$ **do**
11. update merge cost of G_c ;
12. **return** G_s ;

Figure 6: Algorithm mSum

strategy always chooses a keyword pair with a connection graph that “minimally” introduces new nodes and edges to the dynamically maintained 1-summary graph.

The algorithm mSum is shown in Fig. 6. It first initializes a summary graph G_s (as empty), as well as an empty answer graph set \mathcal{G}_C to maintain the answer graphs to be selected for summarizing (line 1). For each keyword pair (k, k') , it computes the connection graph $G_{c(k, k')}$ from the union of the answer graphs in \mathcal{G} , and puts $G_{c(k, k')}$ to \mathcal{G}_C (line 2-3). This yields a set \mathcal{G}_C which contains in total $O(\frac{|Q|(|Q|-1)}{2})$ connection graphs. It then identifies a subset of connection graphs in \mathcal{G} by greedily choosing a connection graph G_c that minimizes a dynamically updated merge cost $\delta_r(\mathcal{G}_C, G_c)$, as remarked earlier (line 5). In particular, we use an efficiently estimated merge cost, instead of the accurate cost via summarizing computation (as will be discussed). Next, it either computes G_s as a 1-summary graph for $G_{c(k, k')}$ if G_s is \emptyset , by invoking pSum (line 6), or updates G_s with the newly selected G_c , by invoking a procedure merge (line 7). G_c is then removed from \mathcal{G}_C (line 8), and the merge cost of all the rest connection graphs in \mathcal{G}_C are updated according to the selected connection graphs (line 10-11). The process repeats until $m = \lceil \frac{\alpha|Q|(|Q|-1)}{2} \rceil$ pairs of keywords are covered by G_s , *i.e.*, m connection graphs are processed (line 9). The updated G_s is returned (line 12).

Procedure. The procedure merge (not shown in Fig. 6) is invoked to update G_s upon new connection graphs. It takes as input a summary graph G_s and a connection graph G_c . It also keeps track of the union of the connection graphs G_s corresponds to. It then updates G_s via the following actions: (1) it removes all the nodes in G_c that are dominated by the nodes in itself or the union graph; (2) it identifies equivalent nodes from the union graph and G_c (or have the same identification); (3) it then splits node v_s in G_s if $[v_s]$ contains two nodes that cannot dominate each other, or merge all the nodes in G_s that have dominance relation. G_s is then returned if no more nodes in G_s can be further updated.

Optimization. The merge cost (line 5) of mSum takes in total $O(|\mathcal{G}|^2)$ time. To reduce the merging time, we efficiently *estimate* the merge cost. Given \mathcal{G} , a neighborhood containment relation R_r captures the *containment* of the label sets from the neighborhood of two nodes in the union of the graphs in \mathcal{G} . Formally, R_r is a binary relation over the nodes in \mathcal{G} , such that a pair of nodes $(u, v) \in R_r$ if and

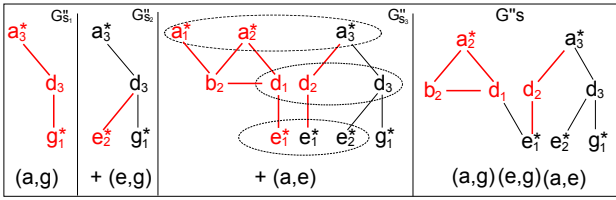


Figure 7: Computing minimum α -summary graph

only if they have the same label, and for each neighbor u' of u , there is a neighbor v' of v with the same label of u' . Denoting as $D(R_r)$ the union of the edges attached to the node u , for all $(u, v) \in R_r$, we have the following result.

Lemma 6: For a set of answer graphs \mathcal{G} and its 1-summary G_s , $|\mathcal{G}| \geq |G_s| \geq |\mathcal{G}| - |R_r(\mathcal{G})| - |D(R_r)|$. \square

To see this, observe the following. (1) $|\mathcal{G}|$ is clearly no less than $|G_s|$. (2) Denote G as the union of the answer graphs in \mathcal{G} , we have $|G_s| \geq |G| - |R_r(G)| - |D(R_r)|$, where $R_r(G)$ is the dominance relation over G , and $D(R_r)$ is similarly defined as $D(R_r)$. (3) For any $(u, v) \in R_r(G)$, (u, v) is in $R_r(\mathcal{G})$. In other words, $|R_r(G)| \leq |R_r(\mathcal{G})|$, and $|D(R_r)| \leq |D(R_r)|$. Putting these together, the result follows.

The above result tells us that $|\mathcal{G}| - |R_r(\mathcal{G})| - |D(R_r)|$ is a lower bound for G_s of \mathcal{G} . We define the merge cost $\delta_r(\mathcal{G}_C, G_C)$ as $|\mathcal{G}| - |R_r(\mathcal{G})| - |D(R_r)| - |G_s(\mathcal{G}_C)|$. Using an index structure that keeps track of the neighborhood labels of a node in \mathcal{G} , $\delta_r(\mathcal{G}_C, G_C)$ can be evaluated in $O(|\mathcal{G}|)$ time.

Analysis. The algorithm `mSum` correctly outputs an α -summary graph, by preserving the following invariants. (1) During each operation in `merge`, G_s is correctly maintained as a minimum summary graph for a selected keyword pair set. (2) Each time a new connection graph is selected, G_s is updated to a summary graph that covers one more pair of keywords, until m pairs of keywords are covered by G_s .

For complexity, (1) it takes in total $O(m \cdot |\mathcal{G}|)$ time to induce the connection graphs (line 1-3); (2) the `while` loop is conducted m times (line 4); In each loop, it takes $O(|\mathcal{G}|^2)$ time to select a G_C with minimum merge cost, and to update G_s (line 7). Thus, the total time complexity is $O(m|\mathcal{G}|^2)$. Note that in practice m is typically small.

Example 7: Recall the query $Q' = \{a, c, e, f, g\}$ and the answer graph set $\mathcal{G} = \{G_1, G_2\}$ in Fig. 3. There are in total 10 keyword pairs. To find a minimum 0.3-summary graph, `MSUM` starts with a smallest connection graph induced by e, g , (a, g) , and computes a 1-summary graph as G_{s_1}'' shown in Fig. 7. It then identifies the connection graph G_C induced by (e, g) , with least merge cost. Thus, G_{s_1} is updated to G_{s_2} by merging G_C , with one more node e_2 and edge (d_3, e_2) inserted. It then updates the merge cost, and merges the connection graph of (a, e) to G_{s_2}'' to form G_{s_3}'' , by invoking `merge`. `merge` identifies that in G_{s_3}'' (1) a_1 is dominated by a_2 , (2) the two e_1^* nodes refer to the same node. Thus, it removes a_1 and merges e_1^* , updating G_{s_3}'' to G_s'' , and returns G_s'' as a minimum 0.3-summary graph. \square

Remarks. `mSum` can be adapted to (approximately) find a summary graph with size no larger than B with a maximized α . To this end, `mSum` is invoked in $O(\log |Q|)$ times to find the coverage ratio via a binary search. At each iteration, it computes a minimum α -summary graph G_s for a fixed α . If $|G_s|$ is larger than B , it changes α to $\frac{\alpha}{2}$; and otherwise, $2 \cdot \alpha$. The process repeats until a proper α is identified.

5. COMPUTING K SUMMARIZATIONS

In this section we study how to construct K summary graphs for answer graphs, *i.e.*, the `KSUM` problem.

Complexity. We start by proving Theorem 3 (Section 2). Given Q, \mathcal{G} , an integer K and a size bound B , the decision problem of `KSUM` asks if there exists a K -partition of \mathcal{G} , such that the sum of the 1-summary graph for each partition is no more than B . The problem is in NP. To show the lower bound, we construct a reduction from the graph decomposition problem shown to be NP-hard [29]. Given a complete graph G where each edge is assigned with an integer weight, the problem is to identify K' partitions of edges, such that the sum of the maximum edge weight in each partition is no greater than a bound W . Given an instance of the problem, (a) We identify the maximum edge weight w_m in G , and construct w_m intermediate nodes $V_I = \{v_1, \dots, v_{w_m}\}$, where each intermediate node has a distinct label. (b) For each edge in G with weight w_i , we construct an answer graph with two fixed keyword nodes k_1, k_2 and edges (k_1, v_j) and (v_j, k_2) , where $v_j \in V_I$, and $j \in [1, w_i]$. (c) We set $K = K'$, and $B = W$. One may verify that if a K' -partition of edges in G has a total weight within W , then there exists a K -partition of \mathcal{G} with total summary size within $3W + 2K$, and vice versa. Thus, `KSUM` is NP-hard. This verifies that `KSUM` is NP-complete.

The APX-hardness of the K summarization problem can be shown similarly, by conducting an approximation preserving reduction from the graph decomposition problem, which is shown to be APX-hard [29]. The above analysis completes the proof of Theorem 3.

We next present a heuristic algorithm for `KSUM`. We first introduce a distance measure for answer graphs.

Graph distance metric. Given two answer graphs G_1 and G_2 , we introduce a similarity function $F(G_1, G_2)$ as $\frac{|R_r(G_{1,2})| + |D(R_r)|}{|G_1| + |G_2|}$, where $G_{1,2}$ is the union of G_1 and G_2 , and $R_r(G_{1,2})$ and $D(R_r)$ are as defined in Section 4. Intuitively, the similarity function F captures the similarity of two answer graphs, by measuring “how well” a summary graph may compress the union of the two graphs [12]. Thus a distance function $\delta(G_1, G_2)$ can be defined as $1 - F(G_1, G_2)$.

Based on the distance measure, we propose an algorithm, `kSum`, which partitions \mathcal{G} into K clusters G_P , such that the total set distance $F(G_{P_i})$ in each cluster G_{P_i} is minimized. This intuitively leads to K small summary graphs.

Algorithm. The algorithm `kSum` works similarly as a K -center clustering process [4]. It first initializes a set G_P to maintain the partition of \mathcal{G} , an answer graph set \mathcal{G}_K with randomly selected K answer graphs from \mathcal{G} as K “centers”, and a summary set \mathcal{G}_S to keep record of K 1-summary graphs, each corresponds to a cluster G_{P_i} in G_P ; in addition, the total difference θ is initialized as a large number, *e.g.*, $K|\mathcal{G}|^2$. It then iteratively refines the partition G_P as follows. (1) For each answer graph $G \in \mathcal{G}$, it selects the “center” graph G_{c_j} which minimizes $\delta(G, G_{c_j})$, *i.e.*, is the closest one to G , and extends the cluster G_{P_j} with G . (2) The updated clusters G_P forms a partition of \mathcal{G} . For each cluster $G_{P_i} \in G_P$, a new “center” graph G_{c_i}' is selected, which minimizes the sum of the distance from G_{c_i}' to all the rest graphs in G_{P_i} . The newly identified K graphs replace the original graphs in \mathcal{G}_K . (3) The overall distance $\theta = \sum_i \sum_{G \in G_{P_i}} \delta(G, G_{c_i})$ is recomputed for G_P . `kSum` re-

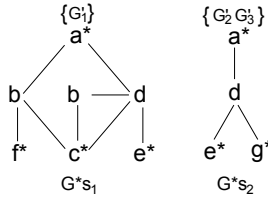


Figure 8: summary graphs for a 2-partition

peats the above process until θ converges. It then computes and returns K 1-summary graph by invoking the algorithm `pSum` for each cluster $G_{p_i} \in G_P$.

Example 8: Recall the answer graph set $\mathcal{G} = \{G'_1, G'_2, G'_3\}$ in Fig. 3. Let $K=2$. The algorithm `pSum` first selects two graphs as “center” graphs, *e.g.*, G'_1 and G'_3 , and computes the distance between the graphs. One may verify that $\delta(G'_1, G'_2) > \delta(G'_2, G'_3)$. Thus, G'_2 and G'_3 are much “closer,” and are grouped together to form a cluster. This produces a 2-partition of \mathcal{G} as $\{\{G'_1\}, \{G'_2, G'_3\}\}$. The 1-summary graphs are then computed for each cluster. `pSum` finally returns G'_{s_1} and G'_{s_2} as the minimized 2 1-summary graphs, with total size 22 (Fig. 8). \square

Analysis. The algorithm `kSum` correctly computes K 1-summary graphs for a K -partition of \mathcal{G} . It heuristically identifies K clusters with minimized total distance of each answer graph in the cluster to its “center” graph. `kSum` can also be used to compute K α -summary graphs.

For complexity, (1) it takes `kSum` $O(\mathcal{G})$ time for initialization; (2) the clustering phase takes in total $O(I \cdot K \cdot |G_m|^2)$ time, where I is the number of iterations, and G_m is the largest answer graph in \mathcal{G} ; and (3) the total time of summarization is in $O(|Q|^2 ||\mathcal{G}| + |\mathcal{G}|^2)$. In our experiments, we found that I is typically small (no more than 3).

6. EXPERIMENTAL EVALUATION

In this section, we experimentally verify the effectiveness and efficiency of the proposed algorithms.

6.1 Experimental Settings

Datasets. We use the following three real-life datasets in our tests. (1) *DBLP* (<http://dblp.uni-trier.de/xml/>), a bibliographic dataset with in total 2.47 million nodes and edges, where (a) each node has a type from in total 24 types (*e.g.*, ‘paper’, ‘book’, ‘author’), and a set of attribute values (*e.g.*, ‘network’, ‘database’, etc), and (b) each edge denotes *e.g.*, authorship or citation. (2) *DBpedia* (<http://dbpedia.org>), a knowledge graph which includes 1.2 million nodes and 16 million edges. Each node represents an entity with a type (*e.g.*, ‘animal’, ‘architectures’, ‘famous places’) from in total 122 types, with a set of attributes (*e.g.*, ‘jaguar’, ‘Ford’). (3) *YAGO* (<http://www.mpi-inf.mpg.de/yago>) is also a knowledge graph. Compared with *DBLP* and *DBpedia*, it is “sparser” (1.6 million nodes, 4.48 million edges) and much richer with diverse schemas (2595 types).

Keyword queries. We design keyword queries as follows.

(1) For *DBLP*, we select 5 common queries as shown in Table 1. The keyword queries are for searching information related with various topics or authors. For example, Q_1 is to search the mining techniques for temporal graphs.

(2) For *DBpedia* and *YAGO*, we design 6 query templates

Query	Keywords	card(\mathcal{G})	$ V , E $
Q_1	mining temporal graphs	355	(5,6)
Q_2	david parallel computing ACM	1222	(5,4)
Q_3	distributed graphs meta-data integration	563	(5,5)
Q_4	improving query uncertain database conference	1617	(9,14)
Q_5	keyword search algorithm evaluation XML conference	7635	(7,8)

Table 1: Queries for DBLP

Query	Keywords template	$ Q_T $	card(\mathcal{G})	$ V , E $
Q_{T_1}	Jaguar place	136	75	(5,7)
Q_{T_2}	united_states politician award	235	177	(6,7)
Q_{T_3}	album music genre american music awards	168	550	(11,25)
Q_{T_4}	fish bird mammal protected_area north_american	217	1351	(12,24)
Q_{T_5}	player club manager league city country	52	1231	(17,28)
Q_{T_6}	actor film award company hollywood	214	1777	(12,27)

Table 2: Queries and the answer graphs for *DBpedia*. The templates are also applied for *YAGO*.

Q_{T_1} to Q_{T_6} , each consists of *type* keywords and *value* keywords. The *type* keywords are taken from the type information in *DBpedia* (*resp.* *YAGO*), *e.g.*, *country* in Q_{T_5} , and the value keywords are from the attribute values of a node, *e.g.*, *United States* in Q_{T_2} . Each query template Q_{T_i} is then extended to a set of keyword queries (simply denoted as Q_{T_i}), by keeping all the value keywords, and by replacing some type keywords (*e.g.*, *place*) with a corresponding value (*e.g.*, *America*). Table 2 shows the query templates Q_T and the total number of its corresponding queries $|Q_T|$. For example, for Q_{T_1} , 136 keyword queries are generated for *DBpedia*. One such query is {‘Jaguar’, ‘America’}.

Answer graph generator. We generate a set of answer graphs \mathcal{G} for each keyword query, leveraging [17,20]. Specifically, (1) the keyword search algorithm in [17] is used to produce a set of trees connecting all the keywords, and (2) the trees are expanded to a graph containing all the keywords, with a bounded diameter 5, using the techniques in [20]. Table 1 and Table 2 report the average number of the generated answer graphs $\text{card}(\mathcal{G})$ and their average size, for *DBLP* and *DBpedia*, respectively. For example, for Q_{T_3} , an answer graph has 11 nodes and 25 edges (denoted as (11, 25)) on average. For *YAGO*, $\text{card}(\mathcal{G})$ ranges from 200 to 2000, with answer graph size from (5, 7) to (10, 20). On the other hand, various methods exist *e.g.*, top- k graph selection [35], to reduce possibly large answer graphs.

Implementation. We implemented the following algorithms in Java: (1) `pSum`, `mSum` and `kSum` for answer graph summarization; (2) `SNAP` [34] to compare with `pSum`, which generates a summarized graph for a single graph, by grouping nodes such that the pairwise group connectivity strength is maximized; (3) `kSumtd`, a revised `kSum` using a top-down strategy: (a) it randomly selects two answer graphs G_1 and G_2 , and constructs 2 clusters by grouping the graphs that are close to G_1 (*resp.* G_2) together; (b) it then iteratively splits the cluster with larger total inter-cluster distance to two clusters by performing (a), until K clusters are constructed, and the K summary graphs are computed.

All experiments were run on a machine with an Intel Core2 Duo 3.0GHz CPU and 4GB RAM, using Linux. Each experiment was run 5 times and the average is reported here.

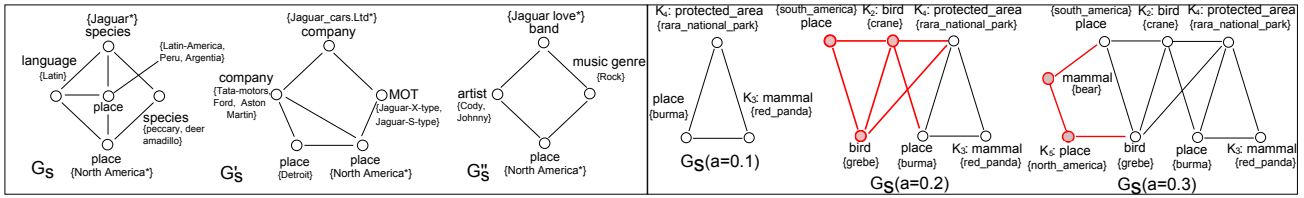


Figure 9: Case study: summarizing real-life answer graphs

6.2 Case Study: K and α -summarization

We first provide a case study using *DBpedia*. (1) Fixing $K = 10$ and $Q = \{\text{Jaguar}, \text{America}\}$, we select 3 summary graphs generated by $k\text{Sum}$, as shown in Fig. 9 (left). The summary graph suggests three types of connection patterns between Jaguar and America, where Jaguar is a type of animal, car, and a band, respectively. Each intermediate node (e.g., company) contains the entities connecting the keyword nodes, (e.g., Ford). Observe that each summary graph can also be treated as a suggested graph query for Q . (2) Fig. 9 (right) depicts three α -summary graphs for a keyword query Q from the query template Q_{T_4} . $G_s(\alpha=0.1)$ covers a single pair of keyword “protected area” and “mammal”. With the increase of α , new keywords are added to form new α -summary graphs. When $\alpha = 0.3$, we found that $G_s(\alpha=0.3)$ already covers 67% of the path labels for all keyword pairs.

6.3 Performance on Real-life Datasets

Exp-1: Effectiveness of $p\text{Sum}$. We first evaluate the effectiveness of $p\text{Sum}$. To compare the effectiveness, we define the *compression ratio* cr of a summarization algorithm as $\frac{|G_s|}{|G|}$, where $|G_s|$ and $|G|$ are the size of the summary and answer graphs. For $p\text{Sum}$, G_s refers to the 1-summary graph for G and Q . Since SNAP is not designed to summarize a set of graphs, we first union all the answer graphs in G to produce a single graph, and then use SNAP to produce a summarized graph G_s . To guarantee that SNAP preserves path information between keywords, we carefully selected parameters e.g., participation ratio [34]. We verify the effectiveness by comparing cr of $p\text{Sum}$ with that of SNAP .

Fixing the query set as in Table 1, we compared cr of $p\text{Sum}$ and SNAP over *DBLP*. Fig. 10(a) tells us the following. (a) $p\text{Sum}$ generates summary graphs much smaller than the original answer graph set. For example, cr of $p\text{Sum}$ is only 7% for Q_2 , and is on average 23%. (b) $p\text{Sum}$ generates much smaller summary graphs than SNAP . For example, for Q_2 over *DBLP*, the G_s generated by $p\text{Sum}$ reduces the size of its counterparts from SNAP by 67%. On average, $p\text{Sum}$ outperforms SNAP by 50% over all the datasets. While SNAP may guarantee path preserving via carefully set parameters, it cannot identify dominated nodes, thus produces larger G_s .

Using Q_{T_i} ($i \in [1, 6]$), we compared cr of $p\text{Sum}$ and SNAP over *DBpedia* (Fig. 10(b)) and *YAGO* (Fig. 10(c)). (1) $p\text{Sum}$ produces summaries on average 50% (resp. 80%) smaller of the answer graphs, and on average 62% (resp. 72%) smaller than their counterparts generated by SNAP over *DBpedia* (resp. *YAGO*). (2) For both algorithms, cr is highest over *DBpedia*. The reason is that *DBpedia* has more node labels than *DBLP*, and the answer graphs from *DBpedia* are denser (Table 2). Hence, fewer nodes can be removed or grouped in the answer graphs for *DBpedia*, leading to larger summaries. To further increase the compression ratio, one can resort to α -summarization with information loss.

Exp-2: Effectiveness of $m\text{Sum}$. In this set of experiments, we verify the effectiveness of $m\text{Sum}$. We compare the average size of α -summary graphs by $m\text{Sum}$ (denoted as $|G_s^\alpha|$) with that of 1-summary graphs by $p\text{Sum}$ (denoted as $|G_s|$). Using real-life datasets, we evaluated $\frac{|G_s^\alpha|}{|G_s|}$ by varying α .

Fixing the keyword query set as $\{Q_3, Q_4, Q_5\}$, we show the results over *DBLP* in Fig. 10(d). (1) $|G_s^\alpha|$ increases for larger α . Indeed, the smaller coverage ratio a summary graph has, the fewer keyword pair nodes and the paths are summarized, which usually reduce $|G_s^\alpha|$ and make it more compact. (2) The growth of $|G_s^\alpha|$ is slower for larger α . This is because new keyword pairs are more likely to have already been covered with the increment of α . Fig. 10(e) and Fig. 10(f) illustrate the results over *DBpedia* and *YAGO* using the query templates $\{Q_{T_4}, Q_{T_5}, Q_{T_6}\}$ (Table 2). The results are consistent with Fig. 10(d).

We also evaluated the *recall* merit of $m\text{Sum}$ as follows. Given a keyword query Q , we denote the recall of $m\text{Sum}$ as $\frac{|P'|}{|P|}$, where P (resp. P') is the set of path labels between the keyword nodes of k and k' in G (resp. α -summary graph by $m\text{Sum}$), for all $(k, k') \in Q$. Figures 10(g), 10(h) and 10(i) illustrate the results over the three real-life datasets. The recall increases with larger α , since more path labels are preserved in summary graphs, as expected. Moreover, we found that $m\text{Sum}$ covers on average more than 85% path labels for all keyword pairs over *DBLP*, even when $\alpha = 0.6$.

In addition, we compared the performance of $m\text{Sum}$ with an algorithm that identifies the minimum summary graph by exhaust searching. Using *DBpedia* and its query templates, and varying α from 0.1 to 1 (we used $p\text{Sum}$ when $\alpha = 1.0$), we found that $m\text{Sum}$ always identifies summary graphs with size no larger than 1.07 times of the minimum size.

Exp-3: Effectiveness of $k\text{Sum}$. We next evaluate the effectiveness of $k\text{Sum}$, by evaluating the *average compression ratio*, $cr_K = \frac{1}{K} \sum_{i=1}^K \frac{|G_{s_i}|}{|G_{p_i}|}$ for each cluster G_{p_i} and its corresponding 1-summary graph G_{s_i} .

Fixing the query set $\{Q_3, Q_4, Q_5\}$ and varying K , we tested cr_K over *DBLP*. Fig. 10(j) tells us the following. (1) For all queries, cr_K first decreases and then increases with the increase of K . This is because a too small K induces large clusters that contain many intermediate nodes that are not dominated by any node, while a too large K leads to many small clusters that “split” similar intermediate nodes. Both cases increase cr_K . (2) cr_K is always no more than 0.3, and is also smaller than its counterpart of $p\text{Sum}$ in Fig. 10(a). By using $k\text{Sum}$, each cluster G_{p_i} contains a set of similar answer graphs that can be better summarized.

The results in Fig. 10(k) and 10(l) are consistent with their counterparts in Fig. 10(a). In addition, cr_K is in general higher in *DBpedia* than its counterparts over *DBLP* and *YAGO*. This is also consistent with the observation in Exp-1.

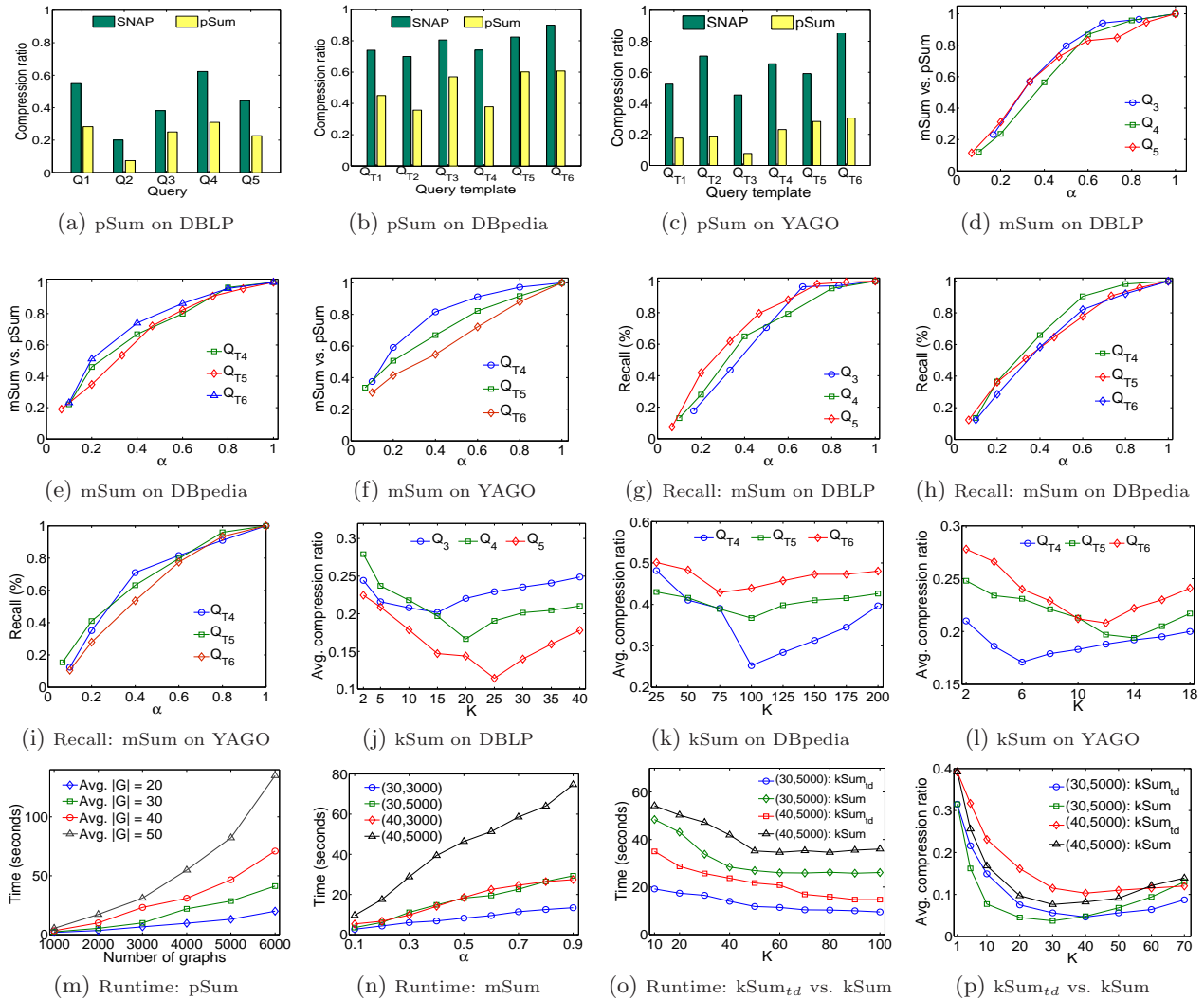


Figure 10: Performance evaluation

The space cost of the algorithms is mainly on storing answer graphs and dominance relations. In general, **pSum** takes at most 100M over *DBLP* and *YAGO*, which is less than 1% of the cost for storing the original data graphs. The space cost of **mSum** and **kSum** are similar to that of **pSum**.

Summary: effectiveness. We found the following. (1) The summarization effectively constructs summary graphs: the compression ratio of **pSum** is on average 24%, and the average compression ratio is 20% for **kSum**. Moreover, **mSum** can provide more compact summary results with some information loss. (2) Graphs with simpler schema (less types) and topology can be better summarized. In addition, our algorithms take up to several seconds over all real-life datasets.

6.4 Performance on Synthetic Dataset

We randomly generated synthetic queries using 5 keywords from a set Σ of 40 random labels. We generated \mathcal{G} with size $\text{card}(\mathcal{G})$ and average graph size $\text{Avg. } |G|$ as follows. We selected 5 labels from Σ , and randomly generated 50 path templates, each connects two keywords with the labels. We then constructed an answer graph by (a) constructing a path from a template by replacing its labels with nodes, and (b) merging two paths until it has size $\text{Avg. } |G|$.

Exp-4: Summarization efficiency. We varied $\text{card}(\mathcal{G})$ from 1000 to 6000 and $\text{Avg. } |G|$ from 20 to 50. Fig. 10(m) tells us that **pSum** takes more time over larger answer graphs, and over larger $\text{card}(\mathcal{G})$. It scales well with $\text{card}(\mathcal{G})$. Note that **pSum** seldom perform its worst case complexity.

Varying α from 0.1 to 0.9, we tested the efficiency of **mSum** where $\text{card}(\mathcal{G})$ (resp. $\text{Avg. } |G|$) varies from 3000 to 5000 (resp. 30 to 40). Fig. 10(n) shows that **mSum** scales well with α , and takes more time when $\text{card}(\mathcal{G})$ and $\text{Avg. } |G|$ increase.

Fixing $\text{card}(\mathcal{G}) = 5000$, we evaluated the efficiency of **kSum** and its baseline version **kSum_{td}**, by varying K (resp. $\text{Avg. } |G|$) from 10 to 100 (resp. 30 to 40). Figure 10(o) tells us that both algorithms take less time with the increase of K , since they take less total time over smaller clusters induced by larger K . In general, **kSum_{td}** runs faster than **kSum**, due to a faster top-down partitioning strategy.

Fixing $\text{card}(\mathcal{G}) = 5000$, we compared cr_K , *i.e.*, average compression ratio of **kSum_{td}** and **kSum**, by varying K (resp. $\text{Avg. } |G|$) from 1 to 70 (resp. 30 to 40). As shown in Fig. 10(p), cr_K first decreases, and then increases with the increasing of K , the same as Fig. 10(j) and Fig. 10(k). Although **kSum_{td}** is faster, **kSum** outperforms **kSum_{td}** with lower cr_K , due to better iterative clustering strategy.

Summary: efficiency. Our summarization algorithms are efficient, and scale well with the size of answer graphs. They take more time over random graphs than over real datasets, due to (1) larger answer graph number and size, and (2) more diversity in connection patterns. Techniques such as incremental simulation [7] may apply for dynamic and interactive summarization for large number of answer graphs.

7. CONCLUSION

In this paper we have developed summarization techniques for keyword search in graph data. By providing a succinct summary of answer graphs induced by keyword queries, these techniques can improve query interpretation and result understanding. We have proposed a new concept of summary graphs and their quality metrics. Three summarization problems were introduced to find the best summarizations with minimum size. We established the complexity of these problems, which range from PTIME to NP-complete. We proposed exact and heuristic algorithms to find the best summarizations. As experimentally verified, the proposed summarization methods effectively compute small summary graphs for capturing keyword relationships in answer graphs.

For future work, we will compare the summarization for different keyword search strategies. It is also important to consider keywords with different weights, as *e.g.*, importance or interestingness [8], as well as provable guarantees on summary quality and improved efficiency. Our work can also be extended to enhance keyword search with summary structures, so that the access to graph data becomes easier.

Acknowledgments. Yang was supported by an IBM Summer Internship. This research was also supported by the U.S. National Science Foundation under grant IIS-0954125 and by the Army Research Laboratory under cooperative agreement W911NF-09-2-0053 (NSCTA). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

8. REFERENCES

- [1] C. C. Aggarwal and H. Wang. A survey of clustering algorithms for graph data. In *Managing and Mining Graph Data*, pages 275–301. 2010.
- [2] D. Bustan and O. Grumberg. Simulation-based minimization. *TOCL*, 4(2):181–206, 2003.
- [3] S. Chakrabarti, S. Sarawagi, and S. Sudarshan. Enhancing search with structure. *IEEE Data Eng. Bull.*, 33(1):3–24, 2010.
- [4] M. Charikar, S. Guha, É. Tardos, and D. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *STOC*, pages 1–10, 1999.
- [5] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD*, 2009.
- [6] J. Fan, G. Li, and L. Zhou. Interactive sql query suggestion: Making databases user-friendly. In *ICDE*, pages 351–362, 2011.
- [7] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, 2011.
- [8] L. Fang, A. D. Sarma, C. Yu, and P. Bohannon. Rex: Explaining relationships between entity pairs. *PVLDB*, 5(3):241–252, 2011.
- [9] H. Fu and K. Anyanwu. Effectively interpreting keyword queries on rdf databases with a rear view. In *ISWC*, 2011.
- [10] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.
- [11] R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Automated Reasoning*, 2003.
- [12] J. Goldstein, V. Mittal, J. Carbonell, and M. Kantrowitz. Multi-document summarization by sentence extraction. In *NAACL-ANLP Workshop on Automatic Summarization*, pages 40–48, 2000.
- [13] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: ranked keyword search over xml documents. In *SIGMOD*, 2003.
- [14] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.
- [15] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.
- [16] Y. Huang, Z. Liu, and Y. Chen. Query biased snippet generation in xml search. In *SIGMOD*, pages 315–326, 2008.
- [17] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.
- [18] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *ICDE*, pages 129–140, 2002.
- [19] G. Koutrika, Z. M. Zadeh, and H. Garcia-Molina. Data clouds: summarizing keyword search results over structured data. In *EDBT*, pages 391–402, 2009.
- [20] G. Li, B. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, 2008.
- [21] Z. Liu and Y. Chen. Query results ready, now what? *IEEE Data Eng. Bull.*, 33(1):46–53, 2010.
- [22] Z. Liu and Y. Chen. Return specification inference and result clustering for keyword search on xml. *TODS*, 35(2):10, 2010.
- [23] Z. Liu, S. Natarajan, and Y. Chen. Query expansion based on clustered results. *PVLDB*, 4(6):350–361, 2011.
- [24] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate analysis*. 1980.
- [25] T. Milo and D. Suciu. Index structures for path expressions. In *ICDT*, 1999.
- [26] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.
- [27] K. Parthasarathy, S. Kumar, and D. Damien. Algorithm for answer graph construction for keyword queries on rdf data. In *WWW*, 2011.
- [28] D. Petkova, W. B. Croft, and Y. Diao. Refining keyword queries for xml retrieval by combining content and structure. In *ECIR*, pages 662–669, 2009.
- [29] J. Plesník. Complexity of decomposing graphs into factors with given diameters or radii. *Mathematica Slovaca*, 32(4):379–388, 1982.
- [30] N. Sarkas, N. Bansal, G. Das, and N. Koudas. Measure-driven keyword-query expansion. *PVLDB*, 2(1):121–132, 2009.
- [31] S. Shekarpour, S. Auer, A.-C. N. Ngomo, D. Gerber, S. Hellmann, and C. Stadler. Keyword-driven sparql query generation leveraging background knowledge. In *Web Intelligence*, pages 203–210, 2011.
- [32] M. Sydow, M. Pikula, R. Schenkel, and A. Siemion. Entity summarisation with limited edge budget on knowledge graphs. In *IMCSIT*, pages 513–516, 2010.
- [33] S. Tata and G. M. Lohman. Sqak: doing more with keywords. In *SIGMOD*, 2008.
- [34] Y. Tian, R. Hankins, and J. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, 2008.
- [35] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, 2009.
- [36] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.
- [37] H. Wang and C. Aggarwal. A survey of algorithms for keyword search on graph data. *Managing and Mining Graph Data*, pages 249–273, 2010.
- [38] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *ICDE*, 2010.