Summarization for Knowledge Graph Search

Qi Song[1] F A Rezaur Rahman Chowdhury[1] Yinghui Wu[1]
Janardhan Rao Doppa[1] Xin Luna Dong[2]
[1] Washington State University [2]Google Inc.
{qsong, fchowdhu, yinghui, jana}@eecs.wsu.edu lunadong@google.com

Abstract

Knowledge graphs arise in several real-world applications. However, querying heterogeneous and large-scale knowledge graphs is very challenging under real-world resource constraints (e.g., time). In this paper, we study a novel graph summarization framework to facilitate knowledge graph search. 1) We characterize summarization of a knowledge graph in terms of a set of *summary patterns*, where each summary pattern can be seen as a graphical representation of a group of similar entities and their relationships. In contrast to conventional patterns (e.g., those defined by subgraph isomorphism), summary patterns are capable of identifying and describing approximately similar entities in a time-efficient manner. 2) We formulate the computation of graph summarization as a bi-criteria pattern mining problem. Given a knowledge graph G, the problem is to discover k diversified summary patterns within a specified representation budget (pattern size). We develop two qualitatively different algorithms to solve this problem: an *approximation* algorithm with quality guarantees and a parameterized *anytime* algorithm to trade-off speed and accuracy in a principled manner. 3) We develop fast query evaluation algorithms by leveraging the graph summarization. These algorithms efficiently compute (approximate) answers with high accuracy by only accessing a small number of summary patterns and their materialized views. Using real-world knowledge graphs, we experimentally verify the effectiveness and efficiency of our algorithms for computing summarizations; and query evaluation guided by summarization.

1. Introduction

Graphs are routinely used to represent entities and their relationships in knowledge bases [3, 7, 11]. Unlike relational data, real-world large knowledge graphs lack the support of well-defined schema and typing system. For example, public knowledge graphs such as Freebase [7] contains more than 2.4 billion diversified facts over 58 million topics.

To search knowledge graphs, a number of query processing techniques are proposed [19, 30]. Nevertheless, it is hard for the end-users to issue precise queries that will lead to meaningful answers without any prior knowledge of the underlying data graph. Importantly, query evaluation is expensive due to the ambiguity in queries and the inherent computational complexity (*e.g.,* subgraph isomorphism [39]) over large-scale knowledge graphs.

Example 1: Fig. 1 illustrates a sample knowledge graph G of footballers and clubs. Suppose a user wants to find (footballer) and his team information who got at least two awards, and was part of clubs whose managers are from the same country where the club is located. This search can be represented as a subgraph query Q as shown in Fig 1. The answer of Q refers to the set of footballer entities induced by subgraphs that are isomorphic to Q.

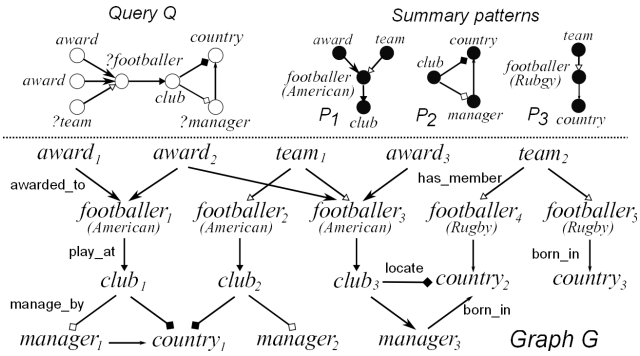


Figure 1: Knowledge graph and summary patterns.

The evaluation of Q over large G is expensive. Moreover, the ambiguous label "footballer" may refer to different types of footballers (*e.g.,American*, *Rugby*). It is hard for the users to concretely specify Q without prior knowledge of G.

Observe that the graph G can be *summarized* with three small graph patterns, shown as P_1, P_2 and P_3 in Fig. 1. Each pattern summarizes a fraction of G, by grouping the entities and their relationships of the same type. For example, P_1 illustrates that G contains a set of American footballers with their award, club, and team information; and P_3 suggests that for Rugby footballers, only their team and country are associated in G. These summaries help the users in understanding the high-level description of G without inspecting the underlying entities.

An additional advantage is that summary patterns can improve the efficiency of query evaluation. For example, Q can be correctly answered using the entities and relationships summarized in P_1 and P_2 (all the matches of Q are included in the entities summarized by P_1 and P_2), *without* visiting additional entities in G. □

This example demonstrates that we can efficiently search large-scale knowledge graphs by leveraging the summarization of the data graph. Graph summarization can help knowledge graph search in a "one-stone-three-bird" way: **1)** Summaries help the users in understanding the complex knowledge graph without exploring a large amount of data; **2)** They have the ability to suggest relevant data for query evaluation and can be directly queried through their materialized views [14, 24]; and **3)** They can facilitate data-driven query suggestions with meaningful labels and relationships.

These observations suggests us to exploit *summarization* of large-scale knowledge graphs to facilitate efficient search.

Therefore, we ask the following questions: **1)** *How can we construct summarization of a given knowledge graph to support effective search?*; and **2)** *How can we leverage the summarization to efficiently answer graph queries?*

Contributions. This paper studies a novel graph summarization framework to facilitate knowledge graph search.

1) We propose the concept of *summary pattern*, a class of graph patterns that preserve path information among entities. We characterize the support of summary patterns in a single knowledge graph. We introduce the notion of knowledge graph summarization, and provide a mathematical formulation of the top-k diversified summarization problem induced by a set of summary patterns.

2) We show that the diversified knowledge graph summarization problem is nontrivial. Despite the hardness, we develop summarization algorithms with desirable performance guarantees, which leverage effective pattern mining techniques. Specifically, we introduce two qualitatively different algorithms to compute summarization with performance guarantees: a) a polynomial time 2-approxmation algorithm; and b) a parameterized *anytime* algorithm to trade-off speed and accuracy in a principled manner.

3) We develop knowledge graph query evaluation algorithms using summarization. Given a resource bound (*e.g.,* amount of data allowed to access), the search algorithm guided by summarization selects a small number of summary patterns to answer a given graph query. Query evaluation is performed by accessing a small fraction of the knowledge graph (matches of the selected patterns). We show that our algorithm achieves feasible accuracy and response time, without relying on a static query optimization or evaluation pipeline.

4) We perform extensive experiments on three large-scale real-world knowledge graphs and synthetic graphs to demonstrate the effectiveness and efficiency of our algorithms for computing summarizations; and query evaluation guided by summarization.

To the best of our knowledge, ours is the first work to employ diversified pattern discovery for summarization and querying large-scale knowledge graphs. Summarization has potential applications beyond supporting fast graph search. For example, it can be used for query suggestion/refinement and to improve the understanding of results, as illustrated in our case studies. We believe that our framework can pave the way for promising tools for accessing, searching, and understanding complex knowledge graphs.

Related work. We categorize the related work as follows.

Graph summarization: Graph summarization has been studied to describe the data graph with a small amount of information [23, 28, 31, 34, 36, 37, 40]. These approaches can be classified as follows: **1)** Graph compression methods [23, 28, 31] aim to compress graphs within a bounded error by minimizing a information complexity measure, e.g., Minimum Description Length (MDL). There are also methods [28, 31] to reduce the space cost such that the topology of the original data graph can be approximately restored; and algorithms [23] that employ clustering and community detection to describe the data graph with frequent structures (vocabulary) including stars and cliques. However, attributes and labels of nodes/edges are not considered in these works. **2)** Summarization techniques [36, 40] attempt to construct summaries over attributed graphs, where nodes with similar attributes and relationships are grouped together in a controlled manner using parameters such as participation ratio [36]. **3)** Bisimulation relation is adopted [20] to group paths carrying same node labels up to a bounded length to construct summaries. However, the paths can be grouped only when they are pairwise similar, which can be an overkill for knowledge graph summarization. Relaxed bisimulation relation has also been studied to generate summaries over a set of answer graphs [37] to improve the understanding of results. **4)** Entity summarization [33] generates diversified answers (subgraphs) of an underlying knowledge graph for a given entity.

Our work differs from the above works in the following ways: **1)** Our methods produce lossy summaries to facilitate efficient graph query processing over large-scale knowledge graphs, instead of trying to restore the exact topology information of the data graph [28, 31]. **2)** Our goal is to summarize a single knowledge graph to faciliate efficient search, which qualitatively different from query answer summarization [33,37]. Our framework is slightly more involved, but can also be applied for diversified result summarization. **3)** Our summarization is measured by both informativeness and diversity. Therefore, our methods are more involved than MDL-based approaches. **4)** In contrast to [28, 36], our methods do not rely on auxiliary structure and parameters for preserving the entity and relationships. Importantly, diversified graph summarization is not studied in these works.

Graph clustering and mining: A number of graph clustering approaches have been proposed to group similar graphs [4]. However, these techniques are not query-aware [26], and can not be directly applied for summarizing a single knowledge graph. Frequent patterns [12] can be mined from a single graph to describe large graphs [23]. However, as we demonstrate in our experimental study, patterns defined by homomorphism are often an overkill for knowledge graph summarization. Our work differs in the following ways: 1) We define summary patterns based on path simulation, and 2) We provide fast pattern mining algorithms to generate diversified summaries for knowledge graphs.

Answering queries using views: View based query evaluation has been shown to be effective for SPARQL queries [24] and graph pattern queries [14]. However, views can be hard to derive over heterogeneous knowledge graphs. Moreover, view-based query evaluation typically requires us to find an equivalent query by accessing views defined in the same query language. Our work differs in the following ways: 1) We integrate graph summarization and view-based query evaluation in a common framework, and develop effective view selection mechanisms; 2) Our framework connects two different queries: summary patterns based on graph simulation and subgraph isomorphism; and 3) We study parameterized query evaluation algorithms that can trade-off accuracy and response time in a principled manner.

2. Knowledge Graphs and Summary Patterns

In this section, we formally define graph summarization in terms of summary patterns. We start with several notations.

Knowledge graphs. We consider *knowledge graph* G as a (directed) labeled graph (V, E, L), where V is a set of nodes, $E \subseteq V \times V$ is a set of edges. Each node $v \in V$ represents an entity with label $L(v)$ that may carry the content of v such

as type, name, and attribute values, as found in knowledge bases and property graphs [2]; and each edge $e \in E$ represents a relationship $L(e)$ between two entities.

Example 2: A sample knowledge graph G is shown in Fig. 1. Each footballer entity has label that carries type information (*e.g.,*American, Rugby), and is associated with entities such as club (club), award (award), and team (team), through corresponding relationships (*e.g.,*plays_{at}). Unlike XML and RDF graphs, there is no standard schema to describe the entities and relationships of knowledge graph G. □

We use the following notations: 1) A path ρ in a graph G is a sequence of edges $e_1, \ldots, e_n$, where $e_i=(v_i, v_{i+1})$ is an edge in G; 2) The *path label* $L(\rho)$ is defined as $L(v_1)L(e_1)\ldots L(v_n)L(e_n)L(v_{n+1})$, *i.e.,* concatenation of all the node and edge labels on the path ρ; and 3) A graph $G'=(V', E', L')$ is a node induced subgraph of $G=(V, E, L)$ if $V' \subseteq V$, and E' consists of all the edges in G with endpoints in V'. It is an edge induced subgraph if it contains $E' \subseteq E$ and all nodes that are endpoints of edges in E'.

Summary pattern. Given a knowledge graph G, a summary pattern P of G is a graph (V_P, E_P, L_P), where V_P (resp. $E_P \subseteq V_P \times V_P$) is a set of pattern nodes (resp. edges), and each node $u \in V_P$ (resp. edge $e \in E_P$) has a label $L_P(u)$ (resp. $L_P(e)$). Moreover, 1) Each node $u \in V_P$ (resp. $e \in V_E$) represents a node set $[u]$ (resp. edge set $[e]$) from G, such that all the nodes $v \in [u]$ (resp. $e' \in [e]$) have the same label, *i.e.,* $L(v)=L_P(u)$ (resp. $L(e')=L_P(e)$); and 2) For any path ρ_p in P from pattern node $[u_1]$ to $[u_2]$, and for every node $v_1 \in [u_1]$, there exists a node $v_2 \in [u_2]$, such that there exists a path ρ from v_1 to v_2 in G with path label $L(\rho_p)$, i.e., $L(\rho)=L(\rho_p)$. Note that the path label in pattern P is defined similar to its counterpart in G.

Intuitively, a summary pattern P provides an abstract graphical representation of a set of entities and their connectivity information in G. Moreover, it will never suggest "wrong" paths: each path in P corresponds to a set of paths with same labels in G.

Example 3: Fig. 1 shows three summary patterns P_1, P_2, and P_3 of the sample knowledge graph G. The entities in G summarized by P_1 are shown below.

pattern node	entities
[award]	$\{\mathsf{award}_i,\ i \in [1,3]\}$
[team]	$\{\mathsf{team}_1\}$
[footballer]	$\{\mathsf{footballer}_i,\ i \in [1,3]\}$
[club]	$\{\mathsf{club}_i,\ i \in [1,3]\}$

Indeed, for every path in P_1 (*e.g.,* ρ_p={award,footballer, club}) and for every entity with label award, there exists a path ρ (*e.g.,* $\{\mathsf{award}_1,\mathsf{footballer}_1, \mathsf{club}_1\}$) in G with the same label as ρ_p. Similarly, P_2 preserves the path labels between club, manager, and country entities; and P_3 specifies the relationships between teams, footballers, and country in G. □

A pattern matching interpretation. Given a graph pattern P and a knowledge graph G, we may want to verify if P is a summary pattern of G; and if so, identify the *maximum* amount of entities and relationships it can summarize. Fortunately, this problem is tractable.

Proposition 1: *Given a graph pattern $P=(V_P, E_P, L_P)$ and a graph $G=(V, E, L)$, it is in $O((|V_P|+|V|)(|E_P|+|E|))$ time to determine if P is a summary pattern of G.* □

We can achieve this by interpreting the semantics of summary patterns in terms of graph pattern matching based on *simulation preorder* on graphs [18].

Graph simulation preorder [18]. Given a pattern P and a graph G, we say G matches P if there exists a relation $R \subseteq V_P \times V$ that satisfies the following properties: a) for every node u in P, there exists a node v in G such that $L_P(u)=L(v)$ and $(u, v) \in R$ (*i.e., v is a node match of u*); and b) for every node pair $(u, v) \in R$ and each edge $e=(u, u')$ with label $L_P(e)$, there exists an edge $e'=(v, v')$, such that $L(e')=L_P(e)$ (*i.e., e' is an edge match of e*), and $(u', v') \in R$.

The following result that provides a sufficient and necessary condition for determining whether P is a summary pattern of G or not.

Lemma 2: *A pattern P is a summary pattern for graph G, if and only if G matches P via simulation preorder.* □

The above result provides an interpretation of a summary pattern P as follows. Given a graph G that matches P via a simulation preorder R: 1) Each pattern node u in P naturally provides a compact description of a set of nodes $[u]$ from G as match set of u specified by R; and 2) For any path ρ_p connecting two pattern nodes u_1 and u_2 in P, and every node $v_1 \in [u_1]$, we can find a path from v_1 to a node $v_2 \in [u_2]$ with the same label, as a sequence of edge matches specified by R.

Base graph. It is known that there exists a unique, maximum match relation R^* for a given pattern P and G [18] that specifies a unique, largest set of entities and relationships in G that P can summarize. Hence, the pattern matching interpretation demonstrates that the summary pattern is well-defined. We define the non-empty *base graph* G_P of P as the subgraph of G induced by all the node and edge matches specified by the unique, maximum match R^*.

Better still, we can determine if G matches P via simulation [13,18] (hence, to compute G_P) in $O((|V_P|+|V|)(|E_P|+|E|))$ time. As we will discuss in Section 4, this property is beneficial for efficiently computing the knowledge graph summarization.

We present the detailed proof of Prop. 1 in [1].

Example 4: Consider the summary pattern P_1 and knowledge graph G in Fig 1. We can verify that there exists a unique, maximum simulation preorder R^* between P_1 and G that induces the entities summarized by P_1 in Example 3. For example, for each entity $\mathsf{award}_i \in [\mathsf{award}]$ ($i \in [1,3]$), $(\mathsf{award}, \mathsf{award}_i) \in R$. Note that team_2 in G is not a match of team in P_1. Hence, team_2 cannot be summarized by P_1 since it has no children of type footballer that connects to a club node as required by P_1.

The base graph G_{P_1} of P_1 is the subgraph of G induced by all the entities summarized by P_1 (shown in Example 3) and contains 19 entities and relathionships in total. □

Knowledge graph summarization. We characterize knowledge graph summarization through summary patterns. A summarization $\mathcal{S}_G$ of a knowledge graph G is a set of summary patterns $\{P_1, \ldots, P_n\}$. Each summary pattern P_i is associated with its base graph G_{P_i}. For the graph G in Fig. 1, a summarization $\mathcal{S}_G$ consists of three summary patterns $\{P_1, P_2, P_3\}$.

Discussion. Frequent graph patterns [21] may be considered to summarize entities captured by isomorphic subgraphs. However, this requires expensive isomorphism test, and typically requires much more patterns in order to sum-

marize heterogeneous entities in knowledge graphs. For example, edge (team_1, $\mathsf{footballer}_2$) can no longer be a match of P_1 as subgraph pattern. Simulation equivalence [9, 37] and bisimulation [15] constructs equivalence relations of the nodes in G, and induces quotient graphs as summaries. These semantics require pairwise similarity which can easily be an overkill for summarizing entities such as $\mathsf{footballer}_1$, $\mathsf{footballer}_2$ and $\mathsf{footballer}_3$ that are neither bisimilar nor simulation equivalence with each other. Neighborhood based summaries [8] merge overlapped neighbors of entities. In contrast, summary pattern can merge entities which do not necessarily share neighbors (*e.g.,*club_1 and club_3).

3. Quality of Summarization

In this section, we describe ways to measure the quality of knowledge graph summarization. Specifically, we introduce a bi-criteria metric that captures both the informativeness and diversity of a summarization.

3.1 Informativeness

The interestingness of a summary pattern P can be quantified by its informativeness, which should capture (a) the total amount of information (entities and their relationships) it encodes in a knowledge graph G [29], and (b) "completeness", *i.e.,* the ratio of summarized information compared with related ones. Indeed, P can be lossy: a path in G through entities that are matches of P may not have a counterpart path in P with the same label. We characterize the informativeness of P with a notion of *relative support.*

Relative support. The relative support of a summary pattern P in a knowledge graph G, denoted as $\mathtt{Supp}(\mathtt{P}, \mathtt{G})$, is defined as $\frac{|G_P|}{|G_C|}$, where (1) G_P is the base graph of P; (2) G_C, referred as the *candidate graph* of P, is the subgraph of G induced by all the entities which have the same labels as those of the nodes in P. Here $|G_P|$ (resp. $|G_C|$) refers to the total number of nodes and edges in G_P (resp. G_C).

In contrast to conventional support in terms of the graph pattern frequency [12, 38], $\mathtt{Supp}(\mathtt{P}, \mathtt{G})$ quantifies the amount of information it can summarize normalized by all the "related" ones in G; larger is better. Moreover, it can be efficiently computed, following Lemma 1 (Section 2).

Corollary 3: *Given a summary pattern P=(V_P, E_P) and a knowledge graph G=(V, E), we can compute* $\mathtt{Supp}(\mathtt{P}, \mathtt{G})$ *in* $O(|V_P| + |V|)(|E_P| + |E|)$ *time.* □

Informativeness. Using relative support, we define the informativeness function $I(\cdot)$ of a summary pattern P as:

$$I(P) = \frac{|P|}{b_p} * \mathtt{Supp}(\mathtt{P}, \mathtt{G})$$

where $|P|$ refers to the size of a pattern P, *i.e.,* total number of nodes and edges in P, and b_p is the budget on the size of the summary patterns that will be considered to construct graph summarization [35]. Intuitively, $I(\cdot)$ favors larger summary patterns that have high relative support. Alternative metrics including total label set and density can also be considered to capture the informativeness of P.

Example 5: Consider the sample knowledge graph G and the summary patterns P_1 in Fig. 1. We can verify the following: 1) Total size of the base graph G_{P_1} is 19; and 2) The candidate graph G_C of P_1 contains 24 entities and edges in total. Hence, the relative support for P_1 is $\frac{19}{24}$. Similarly, the relative supports of P_2 and P_3 are $\frac{13}{17}$ and $\frac{10}{15}$ respectively. Let the size bound b_p be 8. The informativeness of P_1, P_2, and P_3 can be verified as $I(P_1)$=$\frac{7}{8} * \frac{19}{24}$=0.7, $I(P_2)$=$\frac{6}{8} * \frac{13}{17}$=0.57, and $I(P_3)$=$\frac{5}{8} * \frac{10}{15}$=0.42 respectively. P_1 is the most informative pattern. □

Remark. Conventional support defined by the frequency of subgraphs [12] leads to small but less informative patterns (*e.g.,* frequent edges). The minimum description length (MDL) principle has been used to enforce concise encoding with patterns that maximally compress the graph data [21]. However, this approach does not consider the informativeness in terms of the number of entities and relationships a pattern can summarize [29]. Our interestingnesss function $I(\cdot)$ combines both the MDL principle (relative support) and informativeness measure (pattern size).

3.2 Pattern Diversification

Another challenge is to avoid redundancy of information among the extracted summary patterns. Redundancy may arise due to the following: 1) common sub-patterns among summary patterns; and 2) common entities and relationships among the base graphs of the two summary patterns. It is desirable to diversify the summary patterns in a summarization [34].

In conventional pattern mining, closed patterns (frequent patterns with no super-pattern that is more frequent) [38] are commonly employed to avoid redundancy. In an analogical manner, we capture the most informative patterns [29] with a notion of closed summary patterns.

Closed summary patterns. Given a knowledge graph G and a summary pattern P, P is said to be *closed*, if $\mathtt{Supp}(\mathtt{P}) \geq \mathtt{Supp}(\mathtt{P}')$ for every summary patterns P' derived by a unit update (*i.e.,* addition/removal of a node or edge). Indeed, a closed summary pattern implies that addition or removal of entities/relationships will not result in a pattern with better support.

Distance function. To further capture the difference introduced by the entities and relationships summarized by patterns, we define a distance function D between two patterns P_i and P_j over G as follows:

$$D(P_i, P_j) = 1 - \frac{|C_{P_i} \cap C_{P_j}|}{|C_{P_i} \cup C_{P_j}|}$$

where C_{P_i} (resp. C_{P_j}) refers to the match set, *i.e.,* the set of node and edge matches of P_i (resp. P_j)in G.

The distance function D quantifies the pattern difference in terms of the difference between the sets of entities and relationships they can summarize. Other metrics, including label/type difference of the entities can also be integrated into the distance function D.

Example 6: Consider the patterns P_1 and P_2 in Fig. 1. We can verify the following: 1) P_1 is a closed summary pattern within size bound b_p=8. If b_p=10, a closed pattern P_1' can be obtained by adding edge ($\mathsf{club}, \mathsf{country}$) to P_1, with relative support $\mathtt{Supp}(\mathtt{P}_1, \mathtt{G})$=$\frac{25}{30}$=0.83; and 2) Distance between P_1 and P_2 is $D(P_1, P_2)$ = 1-$\frac{2}{30}$ = 0.94, where $|C_{P_1}|$=19, $|C_{P_2}|$ = 11, and $|C_{P_1} \cap C_{P_2}|$=2. Similarly, $D(P_1, P_3)$= 1.0, and $D(P_2, P_3)$ = 0.85. □

3.3 Diversified Graph Summarization

Good summarizations should cover diverse concepts in a single knowledge graph via highly informative patterns. We can naturally formulate this objective using a bi-criteria function F using our informativeness $I(\cdot)$ and distance $D(\cdot)$ functions. Given a summarization $\mathcal{S}_G$ for knowledge graph G as a set of summary patterns, F is defined as follows:

$$F(\mathcal{S}_G) = (1-\alpha)\sum I(P_i) + \frac{\alpha}{k-1}\sum_{P_i \neq P_j \in \mathcal{P}_\mathcal{S}} D(P_i, P_j)$$

where $\mathsf{card}(\mathcal{S}_G)$, the cardinality of $\mathcal{S}_G$, is k,, and $\alpha \geq 0$ is a tunable parameter to trade-off informativeness and diversification. We scale down the second summation (diversification) in $F(\mathcal{S}_G)$ which has $\frac{k(k-1)}{2}$ terms, to balance out the fact that the first summation (informativeness) has k terms.

We introduce the *Diversified knowledge graph summarization* problem. Given a knowledge graph G, an integer k, and a size budget b_p, we want to find a summarization $\mathcal{S}_G$ that contains at most k summary patterns, where (1) each summary pattern in $\mathcal{S}_G$ is a closed pattern; (2) size of each summary pattern bounded by b_p; and 3) the overall objective function $F(\mathcal{S}_G)$ is maximized.

Example 7: Consider the sample graph G in Fig. 1. Let α=0.5 and b_p=8. A top-2 diversified summarization $\mathcal{S}_G$ will contain two summary patterns $\{P_1, P_2\}$, with total quality score $F(\mathcal{S}_G)$= $0.5 * (0.7 + 0.57)$+$0.5 * 0.94 = 1.1$. □

However, this problem is computationally intractable. We can easily verify that it is NP-Hard to find a diversified summarization for knowledge graph G, by constructing a reduction from the maximum dispersion problem [16] that is known to be NP-Complete. In the next section, we present multiple algorithms to solve the summarization problem with desirable quality and efficiency guarantees.

4. Computing Summarization

In this section, we introduce two qualitatively different algorithms to solve the diversified knowledge graph summarization problem: 1) a 2-*approximation* algorithm; and 2) a parameterized *anytime* algorithm to trade-off speed and accuracy in a principled manner that can be interrupted upon time constraints, with the property that the quality of summarization increases monotonically with more time. Both algorithms rely on a closed summary pattern miner to generate candidate patterns.

We introduce the summary pattern miner for generating the candidate patterns in Section 4.1. Next, we describe our 2-*approximation* and *anytime* algorithms for computing summarization in Section 4.2 and Section 4.3 respectively.

4.1 Summary Pattern Miner

Given a knowledge graph G and pattern size bound b_p, the goal of miner, denoted as a procedure candidateGen, generates all the closed summary patterns within size b_p as candidates for computing graph summarization.

Procedure candidateGen**.** Our miner follows a level-wise generate-and-evaluate strategy as in the conventional pattern mining framework [12]. However, it differs in the following ways: 1) quality of the patterns is computed by adapting pattern matching to graph simulation [13]; and 2) avoids redundant computation by incrementally evaluating the quality of the newly generated patterns.

Algorithm Approx-Sum

Input: a graph G, integer k, size bound b_p
Output: summarization $\mathcal{S}_G$.

1. Initialization: $\mathcal{S}_G$:=$\emptyset$; S_P:= $\emptyset$; and t:=0;
2. $\mathcal{C}_P$:= candidateGen(G, b_p);
3. **for** t:=1 to $\lfloor \frac{k}{2} \rfloor$ **do**
4. Find a pattern pair (P, P') from $\mathcal{C}_P$ that maximizes $F'(P, P')$, where $F'(P, P')$= $(1-\alpha)(I(P) + I(P'))+\frac{\alpha}{k-1}D(P, P')$;
5. $\mathcal{S}_G$:=$\mathcal{S}_G \cup \{P, P'\}$;
6. $\mathcal{C}_P$:=$\mathcal{C}_P \setminus \{P, P'\}$;
7. **if** $|\mathcal{S}_G| < k$ **and** $\mathcal{C}_P \neq \emptyset$ **then**
8. Select a pattern $P \in \mathcal{C}_P$ that maximizes $F(\mathcal{S}_G \cup \{P\})$;
9. $\mathcal{S}_G$:=$\mathcal{S}_G \cup \{P\}$;
10. **return** $\mathcal{S}_G$;

Figure 2: Algorithm Approx-Sum

More specifically, procedure candidateGen initializes the set of closed patterns $\mathcal{C}_P$ to be returned with a set of edge patterns (level 1). It generates patterns at a next level $i(>1)$ by merging any two patterns P_{i-1} and P'_{i-1} from level $i-1$ to create a new patterns P_i whenever possible (generation); and incrementally verifies the informativeness function $I(P_i)$ of the new pattern P_i. If P_i is a closed pattern, it is added to the set $\mathcal{C}_P$. The above process is repeated until all the closed summary patterns with size bounded by b_p are discovered. At the end, candidateGen returns $\mathcal{C}_P$, the candidate set of closed summary patterns.

Incremental verification. Procedure candidateGen performs incremental pattern verification to avoid redundant computation. It keeps track of the base graphs of all the generated patterns. For each newly generated pattern P merged by two patterns P_1 and P_2, it initializes and iteratively refines the match sets of P by only accessing the matches of P_1 and P_2, following the definition of graph simulation. This process is repeated until no more edges can be removed from the match set of P. Indeed, the result below shows that summary patterns can be correctly verified without computing from scratch (see full details in [1]).

Lemma 4: *For any summary pattern P obtained by merging two patterns P_1 and P_2, the base graph G_P of P is a subgraph of the union of the base graphs of P_1 and P_2.* □

4.2 Approximate Summarization

We first show that the summarization problem is 2-approximable. as verified by the result below.

Theorem 5: *There is a 2-approximation algorithm for knowledge graph summarization problem with running time $O(N(b + |V|)(b + |E|) + \frac{k}{2}N^2)$, where N is the number of closed summary patterns.* □

Let $\mathcal{S}^*_G$ denote the optimal summarization that maximizes the quality function F. We introduce an approximation algorithm, which returns a summarization $\mathcal{S}_G$ with quality $F(\mathcal{S}_G) \geq \frac{F(\mathcal{S}^*_G)}{2}$. In a nutshell, the algorithm generates all the closed summary patterns within size bound b_p, and follows a greedy strategy by iteratively selecting patterns that maximally improve the overall summarization quality.

Algorithm. Given G, integer k, and size bound b_p, the Approx-Sum algorithm (see Fig. 2) returns a summarization with k summary patterns. It initializes a set $\mathcal{S}_G$ to store the top-k summary patterns (line 1) and invokes the mining procedure candidateGen to obtain a candidate set of closed patterns $\mathcal{C}_P$ (line 2). Next, it iteratively selects two sum-

mary patterns $\{P, P'\}$ from $\mathcal{C}_P$ that maximizes a function F' by "rounding down" the original function F, and adds that pair to $\mathcal{S}_G$ (line 4). This process is repeated $\lfloor \frac{k}{2} \rfloor$ times to get a set of top-k summary patterns $\mathcal{S}_G$ (lines 3-6). If k is odd ($|\mathcal{S}_G|$ is $k-1$), Approx-Sum selects a summary pattern that maximizes the overall quality $F(\mathcal{S}_G)$ if added to $\mathcal{S}_G$ (lines 7-9). It finally returns $\mathcal{S}_G$ (line 10) as a summarization of G.

Example 8: Consider the sample graph G in Fig. 1. Let b_q=8, k=2, and α=0.5. Approx-Sum computes a summarization $\mathcal{S}_G$ with two patterns as follows: 1) It first discovers all the closed summary patterns within size bound 8 by invoking candidateGen. This yields a set of candidate patterns $\mathcal{C}_P$=$\{P_1, P_2, P_3\}$; and 2) Approx-Sum then computes the pairwise quality scores as $F'(P_1, P_2)$= 1.1 (see Example 7), $F'(P_1, P_3)$=$0.5 * (0.7 + 0.42) + 0.5 * 1 = 1.06$, and $F'(P_2, P_3)$=$0.5 * (0.57 + 0.42) + 0.5 * 0.85 = 0.62$. Hence, (P_1, P_2) is selected and returned as the summarization $\mathcal{S}_G$ with quality 1.1. □

Analysis. Approx-Sum approximates the optimal summarization with ratio 2. Once all the closed patterns are identified, the instance of graph summarization problem can be transformed to an instance of the *maximum dispersion problem* [17]. The problem of maximum dispersion is to find, in a weighted complete graph, a subgraph of size k with maximum node and edge weights. By treating summary patterns as a node and each pattern pair as an edge, one may verify that $\mathcal{S}_G$ contains top-k closed summary patterns that maximizes $F(\mathcal{S}_G)$ if and only if the corresponding node set maximizes the total weight. The algorithm Approx-Sum simulates a 2-approximation greedy selection strategy for maximum dispersion problem [17]. Hence, it preserves the approximation ratio 2 (see full details in [1]).

Time complexity analysis. It takes $O(N(b+|V|)(b+|E|))$ time to generate and evaluate a total of N summary patterns with size bounded by b in the procedure candidateGen; and $O(\frac{k}{2}|\mathcal{C}_P|^2)$ time to compute $\mathcal{S}_G$ using the greedy strategy, where $\mathcal{C}_P$ refers to the candidate set of closed summary patterns. Hence, the overall time complexity of Approx-Sum is bounded by $O(N(b+|V|)(b+|E|)+\frac{k}{2}N^2)$, where N is the number of summary patterns generated in candidateGen.

4.3 Anytime Summarization

The main drawback of the 2-approximation algorithm is that it needs to *wait* until all the closed summary patterns are mined to compute the summarization. However, the mining process can be very expensive in terms of both time and space cost for large-scale graphs. For example, it may require $O(|\mathcal{C}_P|^2)$ space to store the pairwise distance information for all the closed patterns in $\mathcal{C}_P$. Therefore, to trade-off computational efficiency and summarization accuracy in a principled manner, we develop a parameterized *anytime* summarization algorithm.

Anytime approximation. Given a problem I and a function $\mathcal{J}$ to measure the quality of a solution, an algorithm $\mathcal{A}$ qualifies as *anytime* [5] with respect to I and $\mathcal{J}$ if the following conditions are met: a) $\mathcal{A}$ returns an answer $\mathcal{A}(I)_t$ when it is interrupted at any time t; and b) $\mathcal{J}(\mathcal{A}(I)_{t'}) \geq \mathcal{J}(\mathcal{A}(I)_t)$ for $t' \geq t$, i.e., quality of results improve with more time.

An optimal anytime algorithm $\mathcal{A}^*$ will return locally optimal answer $\mathcal{A}^*(I)_t$ at any time t based on all the data accessed upto time t. We say an anytime algorithm $\mathcal{A}$ is an *anytime ϵ-approximation* algorithm with respect to I and $\mathcal{J}$, if at *any* time t, the answer $\mathcal{A}(I)_t$ returned by $\mathcal{A}$ approximates the answer $\mathcal{A}^*(I)_t$ with a fixed approximation ratio ϵ. This property is desirable: it verifies that the answer produced by $\mathcal{A}$ has guaranteed approximation ratio to the optimal answers from $\mathcal{A}^*$, if they access the same amount of data and resource for any given time.

The main result of this section is shown below.

Theorem 6: *There is an anytime 2-approximation algorithm for knowledge graph summarization problem.* □

In a nutshell, the anytime algorithm maintains a top-k set $\mathcal{S}_G$, and keeps track of a bounded number of summary patterns that can potentially improve the summarization quality $F(\mathcal{S}_G)$. Instead of waiting for all the patterns to be generated by candidateGen, it updates the patterns once a new closed pattern is discovered, and incrementally updates $\mathcal{S}_G$ by accessing the saved pattern pairs. When interrupted, it returns the up-to-date summarization $\mathcal{S}_G$.

Below we first introduce the auxiliary structure used by the algorithm, followed by the actual algorithm.

Auxiliary structure. Our anytime algorithm maintains the following: 1) a set $\mathcal{C}_P$ to store the closed summary patterns returned by candidateGen; and 2) a set $\mathcal{L}$ of lists, one list L_i for each closed pattern $P_i \in \mathcal{C}_P$. Each list L_i maintains the top-n ($n \in [1, k-1]$) pattern pairs (P_i, P_j) in S_P that have the highest $F'(P_i, P_j)$ score, where $F'(\cdot)$ refers to the revised quality function defined in Approx-Sum (line 4). It bounds the size of the list L_i based on a tunable parameter l_p, which can be adjusted as per the available memory.

Algorithm. Given G, integer k, and two threshold b_p and l_p, AT-Sum computes a summerization $\mathcal{S}_G$ as follows (see Fig. 3). It first initializes $\mathcal{S}_G$, $\mathcal{C}_P$, $\mathcal{L}$, and a flag termination (set as false) to mark if the termination condition is satisfied (line 1). The algorithm AT-Sum then iteratively conducts the following steps.

1) Invokes candidateGen to fetch a newly generated closed pattern P. Note that the procedure candidateGen can be easily modified to return a single pattern after evaluation, instead of returning a set of patterns in a batch.

2) Updates $\mathcal{C}_P$ and the list $\mathcal{L}$ (line 4) based on pattern P. For each pattern $P_i \in \mathcal{C}_P$, it computes the quality score $F'(P_i, P)$, and updates the top-l_p list L_i of P_i by replacing the lowest scoring pair (P_i, P') with (P_i, P), if $F'(P_i, P')$<$F'(P_i, P)$.

3) Incrementally updates the top-k summary pattern set $\mathcal{S}_G$

Algorithm AT-Sum

Input: a graph G, integer k,
pattern size threshold b_p, threshold l_p, time bound t_{max};
Output: summarization $\mathcal{S}_G$.

1. Initialization: $\mathcal{S}_G$:=∅; $\mathcal{C}_P$:= ∅; termination:=false; and L:=∅;
2. **while** termination ≠ true **do**
3. Invoke candidateGen to generate a new pattern P;
4. Update $\mathcal{L}$ and $\mathcal{C}_P$ based on P;
5. Update $\mathcal{S}_G$ with top pattern pairs in $\mathcal{L}$;
6. **if** no new pattern can be generated
OR time-bound reaches t_{max} **then**
7. termination:=true;
8. **return** $\mathcal{S}_G$;

Figure 3: Algorithm AT-Sum

(line 5) as in Approx-Sum. Greedily selects top $\lfloor \frac{k}{2} \rfloor$ pairs of patterns with maximum quality $F'(\cdot)$ from the list set $\mathcal{L}$, and adds the patterns to $\mathcal{S}_G$. If $|\mathcal{S}_G| < k$, a pattern $P \in \mathcal{C}_P \setminus \mathcal{S}_G$ that maximizes the quality $F(\mathcal{S}_G \cup \{P\})$ is added to $\mathcal{S}_G$.

The above process is repeated until the termination condition is satisfied (lines 6-7): a) no new summary pattern can be discovered in candidateGen; or b) running time reaches the time-bound t_{max}. The up-to-date $\mathcal{S}_G$ is then returned.

Example 9: Using the same setting as in Example 8, AT-Sum computes $\mathcal{S}_G$ as follows. It invokes candidateGen to discover a closed summary pattern, *e.g.,* P_3, and initializes $\mathcal{C}_P$ and $\mathcal{S}_G$ with P_3. In round 2, it discovers a new pattern P_2, and updates the auxiliary structures as follows.

round	L	$\mathcal{C}_P$	$\mathcal{S}_G$
2	L_2={<(P_2, P_3),0.62} L_3={<(P_3, P_2),0.62}	$\{P_2, P_3\}$	$\{P_2, P_3\}$

In round 3, it discovers pattern P_1, and updates the top-1 entries in each list of $\mathcal{L}$. The new top element in the lists L_1, L_2, and L_3 are (P_1, P_2), (P_2, P_1) and (P_3, P_1) respectively. Hence, it replaces $\{P_2, P_3\} \in \mathcal{S}_G$ with $\{P_1, P_2\}$, and updates the auxiliary structures as follows.

round	L	$\mathcal{C}_P$	$\mathcal{S}_G$
3	L_1={<(P_1, P_2),1.1} L_2={<(P_2, P_1),1.1} L_3={<(P_3, P_1),1.06}	$\{P_1, P_2, P_3\}$	$\{P_1, P_2\}$

As all the closed patterns within size 8 are discovered, AT-Sum terminates and returns $\mathcal{S}_G$=$\{P_1, P_2\}$. □

Analysis. The algorithm AT-Sum always terminates, and correctly returns a set of summary patterns $\mathcal{S}_G$. We can show that it is an anytime 2-approximation algorithm. First, AT-Sum is an anytime algorithm: 1) It can be terminated for any time-bound t to return $\mathcal{S}_G$; and 2) the set $\mathcal{S}_G$ is incrementally updated such that the overall quality $F(\mathcal{S}_G)$ monotonically improves (line 5). Let $\mathcal{C}_{P_t}$ be the set of patterns generated upto time t, and $\mathcal{S}^*_{G_t}$ be the optimal summarization over candidate set $\mathcal{C}_{P_t}$. Importantly, for l_p=k-1, AT-Sum simulates its 2-approximation counterpart Approx-Sum by accessing $\mathcal{C}_{P_t}$, and produces a summarization $\mathcal{S}_{G_t}$ where $F(\mathcal{S}_{G_t}) \geq \frac{\mathcal{S}^*_{G_t}}{2}$ for a given time t (see details in [1]).

The time-bound t_{max} and threshold l_p provides a principled way to trade-off summarization quality and time/memory cost. The algorithm AT-Sum takes $O(l_p|\mathcal{C}_P|)$ memory, and terminates on or before time-bound t_{max}. When l_p=$k-1$ and t_{max} is sufficiently large, AT-Sum behaves as the 2-approximation algorithm with memory cost $O(|S_P|(k-1))$. As verified in Section 6, AT-Sum only processes 20% of the total summary patterns visited by Approx-Sum to generate 90% accurate summarizations.

Remark. We can easily specialize both Approx-Sum and AT-Sum algorithms to compute summaries for a set of entities/relationships of interest. To this end, we only need to constrain the miner candidateGen with labels from a specified alphabet Σ, which may include frequent labels from *e.g.,* query log analysis [6].

5. Graph Querying via Summarization

In this section, we present a knowledge graph search framework by leveraging summarizations.

Subgraph querying. We adopt the conventional semantics of querying via subgraph isomorphism, which is widely employed in knowledge graph search [19, 25, 30, 39]. A query Q is defined as a graph (V_q, E_q, L_q). Given a graph G=(V, E, L), a match of Q in G is a subgraph G'=(V', E', L') of G that is isomorphic to Q, *i.e.,* there exists a bijective function $f: V_q \rightarrow V'$ such that a) for each node $u \in V_q$, $L_q(u)$=$L'(f(u))$; and b) e=$(u, u') \in E_q$if and only if $f(e)$=$(f(u), f(u')) \in E'$ and $L_q(e)$=$L'(f(e))$.

The answer of Q in G, denoted as $Q(G)$, refers to the set of all the node and edge matches of Q in G. For example, the answer $Q(G)$ for Q in Fig. 1 contains node matches { award$_2$, award$_4$, team$_1$, footballer$_3$, club$_3$, country$_2$, and manager$_3$} in G.

Given a graph pattern query Q, a knowledge graph G and its summarization $\mathcal{S}_G$, our evaluation framework follows a *select-and-evaluate* strategy: 1) In the selection phase, it selects a small number of summary patterns from $\mathcal{S}_G$ that will provide intermediate answers for Q to minimize the evaluation cost; and 2) In the evaluation phase, it accesses the base graphs of the selected summary patterns to compute $Q(G)$, and fetches additional data from G only when necessary.

We next introduce the evaluation and selection mechanisms, in Section 5.1 and Section 5.2 respectively.

5.1 Query Evaluation using Summarization

We first introduce an algorithm to evaluate a query Q by accessing the summarization $\mathcal{S}_G$. The algorithm leverages a mapping structure from Q to the summary patterns in $\mathcal{S}_G$.

Auxiliary mapping λ. Given query Q=(V_q, E_q, L_q), and a set of summary patterns $\mathcal{S}_G$, we say a query edge $e \in E_q$ is covered by $\mathcal{S}_G$, if for all graph G with $\mathcal{S}_G$ as a summarization, there is a mapping λ that maps e to a set of edges $\lambda(e)$=$\{e_1, \ldots, e_m\}$, such that 1) each e_i is from a pattern $P_i \in \mathcal{S}_G$, and 2) $M_e \subseteq \bigcup C(e_i)$ $(i \in [1, m])$, where M_e is the set of edge matches of e in $Q(G)$, and $C(e_i)$ refers to the edges summarized by pattern edge e_i in the base graph G_{P_i}. Specifically, we say that Q is covered by $\mathcal{S}_G$, denoted by $Q \sqsubseteq \mathcal{S}_G$, if there exists a mapping λ such that for each query edge $e \in E_q$, $\lambda(e) \neq \emptyset$.

Intuitively, if $Q \sqsubseteq \mathcal{S}_G$, the subgraph query Q can be answered by accessing only the summary patterns in $\mathcal{S}_G$, without accessing G. In practice, a query Q may only be "partially" covered by $\mathcal{S}_G$, *i.e.,* only a subset of query edges in Q can be mapped to pattern edges by λ to preserve the matching result.

Example 10: Consider query Q in Fig. 1, and summarization $\mathcal{S}_G$=$\{P_1, P_2\}$. We can verify that Q is covered by $\mathcal{S}_G$. Indeed, there exists a mapping λ that maps both edges (award, footballer) in Q to (award, footballer) in P_1, edges (team, footballer) and (footballer, club) to their counterparts in P_1, and the rest of the three edges in Q to their counterparts in P_2. The match of edge *e.g.,* (footballer, club) in Q, which is the edge (footballer$_3$, club$_3$) in G, is included in the match set of edge (footballer, club) in P_1. □

Below we introduce a query evaluation algorithm that leverages the mapping λ.

Evaluation algorithm. Given a subgraph query Q, a knowledge graph G, a summarization $\mathcal{S}_G$ of G, and a mapping λ as input, algorithm Eval-Sum (see Fig. 4) performs efficient query evaluation to compute answer for Q as follows: 1) Splits Q into two subqueries Q_1 and Q_2, where Q_1 is covered by $\mathcal{S}_G$, and Q_2 is induced by the uncovered edges in Q; 2) Evaluates Q_1 by only accessing $\mathcal{S}_G$ and the corresponding base graphs; and 3) If necessary, refines the

Algorithm Eval-Sum

Input: subgraph query Q, graph G, summarization $\mathcal{S}_G$, mapping λ;
Output: query answer $Q(G)$.

1. Initialization: $Q(G):=\emptyset$; set $E_c:=\emptyset$;
2. Construct subquery Q_1 with edges covered by $\mathcal{S}_G$ via λ;
3. Construct subquery $Q_2:= Q \setminus Q_1$;
4. **for each** edge e in Q_1 **do**
5. **for each** edge $e' \in \lambda(e)$ **do**
6. $E_c:=E_c \cup C_{e'}$, where $C_{e'}$ is the edge match set of e';
7. Construct graph G_1 induced by edges in E_c;
8. $Q(G):=Q(G) \cup Q_1(G_1)$;
9. **for each** edge e in Q_2 **do**
10. Extend $Q(G)$ with candidates of e;
11. Refine matches in $Q(G)$;
12. **return** $Q(G)$;

Figure 4: Algorithm Eval-Sum

matches for Q_2 by visiting additional nodes and edges in G.

More specifically, Eval-Sum consists of the following steps.

1) Initializes the match set $Q(G)$ and a set E_c to store the edges from base graphs for query evaluation (line 1). Next, it constructs Q_1, a subquery of Q induced by all the query edges e covered by $\mathcal{S}_G$ (*i.e.,* $\lambda(e) \neq \emptyset$) (line 2), and a subquery Q_2 induced by the rest query edges (line 3).

2) Computes the answer for Q_1 by only accessing $\mathcal{S}_G$ (lines 4-8). For each edge e in Q_1, it finds all the pattern edges in the set $\lambda(e)$ and associated summary patterns, and collects the matches of these edges in their base graphs in the set E_c (lines 4-6). Next, it constructs a graph G_1 induced by the edges in E_c (line 7), computes the match set $Q_1(G_1)$ by invoking a subgraph isomorphism matching algorithm (*e.g.,* VF2 [10]), and adds the match set to $Q(G)$ (line 8). It also keeps track of the bijective mappings, where each mapping maps every edge in Q_1 to an edge in G_1.

3) Completes the evaluation by incrementally verifying the matches for edges in Q_2 only (lines 9-11). It adds the edge candidates for all the edges in Q_2, and checks if each partial mapping in step 2) can be extended to a complete bijection by verifying the edge candidates. If yes, it replaces the partial mapping with complete mapping, and adds the node and edge matches to $Q(G)$. Otherwise, it removes the matches induced by the partial mapping from $Q(G)$.

The above process is repeated until no new matches can be added to or removed from $Q(G)$. At the end, $Q(G)$ is returned as the answer of Q in G.

Example 11: Given query Q in Fig. 1, and the mapping λ in Example 10, Eval-Sum algorithm computes $Q(G)$ as follows. It first identifies the covered sub-pattern Q_1 (=Q), and merges all the summarized entities and edges from P_1 and P_2. Eval-Sum then invokes a subgraph isomorphism algorithm to find the matches $Q(G)$. As all the edges are covered, it returns $Q(G)$ by visiting at most 31 entities and edges, instead of the entire graph G. □

Analysis. To show the correctness of the Eval-Sum algorithm, it suffices to show the following: a) Eval-Sum correctly computes the matches $Q_1(G)$ for Q_1 (lines 4-8); and b) The partial mapping for Q_1 is correctly extended and evaluated for the complete query Q (lines 9-11). For (a), we can see that$Q_1(G_1)$=$Q_1(G)$. Indeed, for each edge e in Q_1, G_1 contains all the edges C_e summarized by pattern edges $e \in \lambda(e)$, and $M_e \subseteq \bigcup_{e' \in \lambda(e)} C'_e$ by definition. For (b), the match set $Q(G)$ after extension contains all the possible matches for Q. The evaluation part correctly identifies the matches for Q by employing a subgraph matching algorithm (*e.g.,* VF2).

Consider a standard subgraph query evaluation algorithm $\mathcal{A}$ (*e.g.,* VF2). We denote the time to compute the answer $Q(G)$ using $\mathcal{A}$ as a function $T(|Q|, |G|)$. It takes Eval-Sum $O(T(|Q_1|, |\mathcal{S}_G|))$ time to compute the answer for Q_1, and additional $O(T(|Q_2|, |Q_1(G) + \Delta|))$ time to complete the computation of $Q(G)$, where $|\mathcal{S}_G|$ refers to the total size of the summary patterns and their base graphs, and Δ refers to the number of additional nodes and edges in G accessed by Eval-Sum. When $Q \sqsubseteq \mathcal{S}_G$, Eval-Sum takes $O(T(|Q|, |\mathcal{S}_G|))$ time to compute $Q(G)$ by only accessing the summarization $\mathcal{S}_G$, without accessing G. Note that any query evaluation algorithm $\mathcal{A}$, along with optimization techniques, can be applied to improve the efficiency of Eval-Sum.

5.2 Selection of Summary Patterns

We next introduce efficient algorithms (realization of mapping λ) to select relevant summary patterns and reduce evaluation cost. In practice, we want to compute a mapping λ that "maximally" uses summarization to evaluate Q. Moreover, the evaluation may be constrained by resource budgets (*e.g.,* time, number of processed nodes/edges). We study pattern selection with resource budget.

Budgeted selection of summary patterns. Given a query Q, a summarization $\mathcal{S}_G$, integer n and a size budget B, the *budgeted summary pattern selection* problem is to find a set of patterns $\mathcal{P}_\mathcal{M}$=$\{P_1, \ldots, P_n\} \subseteq \mathcal{S}_G$ meeting the following conditions: 1) the number of edges in Q covered by $\mathcal{P}_\mathcal{M}$ is maximized; and 2) the total base graph size of the patterns in $\mathcal{P}_\mathcal{M}$ is bounded by B.

Intuitively, the problem is to select a set of n summary patterns that maximally cover Q, and incurs bounded computational cost. We present our main result in this section.

Proposition 7: *The budgeted summary pattern selection problem is* NP*-hard. There exists a* $(1-\frac{1}{e})$ *approximation algorithm with running time* $O(\mathsf{card}(\mathcal{S}_G)|Q|^2+|Q||\mathcal{S}_G|+|\mathcal{S}_G|^2)$ *to select the summary patterns, where* $\mathsf{card}(\mathcal{S}_G)$ *is the number of summary patterns in* $\mathcal{S}_G$*.* □

The NP-hardness of the budgeted summary pattern selection problem can be verified by constructing a reduction from the minimum set cover problem that is known to be NP-complete. As a proof for the second part of the Proposition 7, we present an approximation algorithm for the budgeted summaries selection problem.

We start with a procedure that determines the edges covered by a given summarization, by studying the coverage problem below.

Coverage problem. Given Q, a summary pattern P and a number m, the coverage problem is to determine if at least m query edges in Q are covered by P. The result below shows that the coverage problem can be solved efficiently.

Proposition 8: *Given a subgraph query* Q=(V_q, E_q, L_q) *and a summary pattern* P=(V_p, E_p, L_p)*, it takes* $O((|V_q| + |V_p|)(|E_q| + |E_p|))$ *time to find all the edges in* Q *that are covered by* P*.* □

As a proof of Proposition 8, we present a sufficient and necessary condition to determine if an edge of Q is covered by P. Consider a graph simulation preorder R between summary pattern P and the subgraph query Q as a graph, where

for a pattern edge e=(u, u') in P, a query edge e'=(v, v') in Q is a match of e if $(u, v) \in R$ and $(u', v') \in R$.

Lemma 9: *Given a query Q and a summary pattern P of graph G, an edge e' in Q is covered by P if and only if e' is a match of an edge e in P via a simulation relation.* □

Procedure Cover-Sum. Based on Lemma 9, We introduce a procedure Cover-Sum for the coverage problem. Given Q and a summary pattern P, Cover-Sum computes the simulation preorder R from P to Q. It then checks if a query edge e' in Q is a match of an edge e in P via R. If yes, e' is covered by e. It adds all the edge pairs (e', e) to λ to be useful for summary pattern selection. Cover-Sum performs a simulation matching between Q and P. Hence, its running time is $O((|V_q| + |V_p|)(|E_q| + |E_p|))$. This completes the proof of Proposition 8 (see details in [1]).

Based on Prop. 8, we next introduce a selection algorithm, denoted as Select-Sum.

Selection algorithm. Given a query Q and summarization $\mathcal{S}_G$ of graph G, Select-Sum algorithm employs a greedy selection strategy to iteratively find the "top" summary pattern that covers maximum uncovered edges in Q. We define a ranking function $r(\cdot)$ to score each summary pattern in $\mathcal{S}_G$ as follows:

$$r(P) = \frac{|E_{c_P} \setminus E_c|}{|G_P|}$$

where E_{c_P} refers to the edges in Q that are covered by P, E_c refers to the edges in Q that are already covered by the patterns in $\mathcal{P}_\mathcal{M}$, and $|G_P|$ refers to the size of the base graph of P. The score $r(P)$ for each summary pattern in $\mathcal{S}_G$ is dynamically updated in each iteration.

Select-Sum algorithm first computes the edge set E_{c_P} in Q covered by each summary pattern P, by invoking procedure Cover-Sum. After that, it iteratively performs the following steps until $\mathcal{P}_\mathcal{M}$ contains n summary patterns, or the total size of base graph is larger than B: 1) Selects the summary pattern P with highest score $r(P)$, adds it to $\mathcal{P}_\mathcal{M}$, and updates E_c with the new edges covered by P; and 2) Updates the scores of all the other patterns, i.e., $\mathcal{S}_G \setminus \mathcal{P}_\mathcal{M}$. At the end, $\mathcal{P}_\mathcal{M}$ is returned as the set of selected summary patterns. Additionally, Select-Sum also constructs the mapping λ by computing the matching from each selected pattern to Q, similar to Cover-Sum.

Analysis. The correctness and approximation ratio of Select-Sum can be verified by an approximation-preserving reduction to the budgeted maximum coverage problem [22]. Select-Sum algorithm mimics the greedy approximation algorithm in [22] (with approximation ratio $(1 - \frac{1}{e})$) for budgeted summary pattern selection, and preserves the approximation ratio $(1 - \frac{1}{e})$. For time complexity, Select-Sum takes $O(\mathsf{card}(\mathcal{S}_G)|Q|^2 + |Q||\mathcal{S}_G| + |\mathcal{S}_G|^2)$ time in total n iterations (see detailed analysis in [1]).

The above analysis completes the proof of Prop. 7.

6. Experimental Evaluation

Using real-world and synthetic knowledge graphs, we conducted three sets of experiments to evaluate the following: 1) Performance of algorithms Approx-Sum and AT-Sum to compute summarization; 2) Effectiveness of algorithm Eval-Sum for query evaluation with summarizations; and 3) Effectiveness of summary patterns, using a case study.

Experimental Setting. We used the following setting.

Datasets. We employ three real-life knowledge graphs for our experiments: 1) *DBpedia*[1] consists of 4.86M nodes and 15M edges. Each node represents an entity carrying one of the 676 labels (*e.g.,*'Settlement', 'Person', 'Building'); 2) *YAGO*[2], a sparser graph compared to *DBpedia* with 1.54M nodes and 2.37M edges, but contains more (324343) diversified labels; and 3) *Freebase* (version 14-04-14)[3], with 40.32M entities, 63.2M relationships, and 9630 labels.

Synthetic graphs. We employ *BSBM* ([4]), a benchmark dataset built around an e-commerce use-case, to generate synthetic knowledge graphs G=(V, E, L) over a set of products with different types, related vendors, consumers, and views. The generator is controlled by the number of nodes $|V|$ (up to $60M$ and edges $|E|$ (up to $152M$), and label mapping L drawn from an alphabet Σ of 3080 labels.

Query generation. To evaluate our Eval-Sum algorithm, we generated 50 subgraph queries Q=(V_q, E_q, L_q) over real-world graphs, controlled by the number of query nodes $|V_p|$ and edges $|E_p|$. We inspected meaningful query examples and benchmarks posed on the real-world knowledge graphs, and generated queries with labels drawn from their data (domain, type, and attribute values). For synthetic graphs, we generated 50 queries with labels drawn from an BSBM label alphabet of 500 labels. We consider mixed queries with sizes ranging from (4,6) to (8,14) with different structures including star, trees, and cyclic patterns.

Algorithms. We implemented all the algorithms in Java.

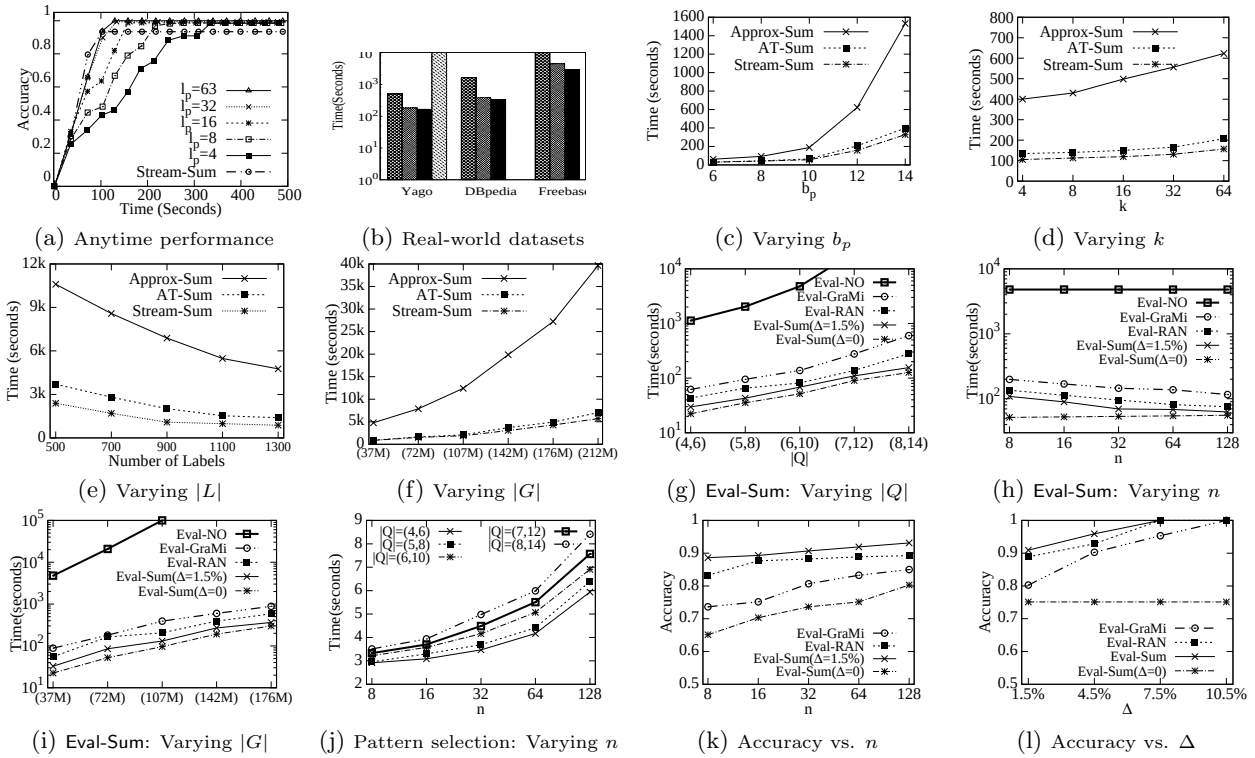
1) Algorithms Approx-Sum and AT-Sum for computing graph summarization are compared with the following baselines: a) Stream-Sum, an anytime heuristic counterpart of AT-Sum that incrementally maintains a diversified summarization $\mathcal{S}_G$ over the stream of closed patterns following [27]. Each time a new pattern P is added by candidateGen to the stream, it simply checks if there is a pattern P' in $\mathcal{S}_G$ that can be replaced by P, such that $F(\mathcal{S}_G \setminus \{P'\} \cup \{P\}) > F(\mathcal{S}_G)$; and b) GraMi, an open-source graph pattern mining tool [12] to discover frequent patterns above a given threshold as a summarization. Here the base graph is computed as the union of all the subgraphs isomorphic to the pattern in G.

2) Our Eval-Sum algorithm is compared with the following baseline algorithms: a) Eval-RND that performs random selection instead of using Select-Sum; b) Eval-GraMi, a counterpart that employs frequent graph patterns mined by GraMi; and c) Eval-NO that evaluates Q by directly employing a subgraph isomorphism algorithm VF2. We also allow a resource bound Δ to be posed on Eval-Sum, Eval-RND and Eval-GraMi, to allow them to return approximate answers by fetching at most Δ additional entities and relationships from G. When Δ=0, they only accesses summary patterns and their matches in G to evaluate the query.

We ran all our experiments on a linux machine powered by an Intel 2.4 GHz CPU with 128 GB of memory. We ran each experiment 5 times and report the averaged results.

Experimental results. We next report our findings.

[1] http://dbpedia.org
[2] http://www.mpi-inf.mpg.de/yago
[3] http://freebase-easy.cs.uni-freiburg.de/dump/
[4] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/



(a) Anytime performance (b) Real-world datasets (c) Varying b_p (d) Varying k

(e) Varying $|L|$ (f) Varying $|G|$ (g) Eval-Sum: Varying $|Q|$ (h) Eval-Sum: Varying n

(i) Eval-Sum: Varying $|G|$ (j) Pattern selection: Varying n (k) Accuracy vs. n (l) Accuracy vs. Δ

Figure 5: Performance evaluation.

Exp-1: Effectiveness of summarization.

Settings. We fixed parameter α=0.5 for diversification, k=64, the summary pattern size bound b_q=6, and l_p=0.5 for this experiment, unless otherwise specified. In addition, we set a relative support threshold of θ=0.5 for closed patterns in the pattern mining procedure candidateGen used by Approx-Sum, AT-Sum, and Stream-Sum. For all the real-life datasets, we considered labels that are not overly general (e.g., top 2% frequent labels like "thing", "place", and "person") to discover more meaningful summaries.

Anytime performance. We evaluate the impact of time constraint t and list size constraint l_p (the number of pattern pairs stored for each pattern in AT-Sum), on the anytime performance of AT-Sum and Stream-Sum. We define the *anytime accuracy* of AT-Sum as $\frac{F(\mathcal{S}_{G_t})}{F(\mathcal{S}_G)}$, where $\mathcal{S}_{G_t}$ refers to the summarization returned by AT-Sum at time t, and $\mathcal{S}_G$ refers to the summarization returned by Approx-Sum. The accuracy of Stream-Sum is defined similarly. Specifically, we monitor the summarization quality and report the "convergence" time of AT-Sum and Stream-Sum when the accuracy reaches ε=98% or does not change for 20 seconds.

Fig. 5(a) shows the anytime accuracy of AT-Sum and Stream-Sum over YAGO. Our observations are as follows: 1) The quality of summarization returned by AT-Sum increases monotonically as t and l_p increases; 2) Convergence speed of AT-Sum to near-optimal summarization improves with increasing l_p values as more pattern pairs are stored and compared. Remarkably, AT-Sum converges in 100 seconds when l_p=63; and 3) Stream-Sum converges faster than AT-Sum, but stops at accuracy 0.9 on average. These verify that AT-Sum provides flexibility to trade-off accuracy with time, and is able to converge early by processing a small number of summary patterns. Remarkably, AT-Sum converges by verifying 65 patterns instead of 300 patterns in total when l_p=63.

Efficiency of Summarization. We evaluate the efficiency of Approx-Sum, AT-Sum, Stream-Sum, and GraMi over the real-world knowledge graphs. For the two anytime algorithms AT-Sum and Stream-Sum, we report their convergence time. For GraMi, we carefully adjusted a support threshold to allow the generation of patterns with similar label set and size to those from Approx-Sum. Our observations from Fig.5(b) are as follows: 1) Approx-Sum and AT-Sum are orders of magnitude faster than GraMi. Moreover, GraMi does not run to completion within 10 hours over both DBpedia and Freebase; 2) Performance of AT-Sum is comparable to that of Stream-Sum, and AT-Sum is 5 times faster than Approx-Sum for generating summarizations of comparable accuracy; 3) In general, the flexibility of AT-Sum makes it feasible over large knowledge graphs. For example, AT-Sum takes less than 100 seconds to produce high-quality summarization with 64 summary patterns for YAGO.

Varying b_p. Using the settings in Fig. 5(a) over YAGO, we varied the pattern size threshold b_p from 6 to 14 (node # + edge#). As shown in Fig.5(c), it takes more time to compute summarization for YAGO with larger values of b_p for all the three algorithms Approx-Sum, AT-Sum, and Stream-Sum, as more candidate patterns are explored and evaluated during the search process. Remarkably, on average AT-Sum is 5 times faster than Approx-Sum to produce summarization with 90% accuracy, and its convergence time is less sensitive to increasing values of b_p when compared with Approx-Sum.

Varying k. Using the same setting over YAGO, we varied k from 8 to 128. Fig. 5(d) tells us that all the algorithms

take more time with larger k, as expected. Additionally, the convergence time of AT-Sum and Stream-Sum are less sensitive to increasing of k when compared with Approx-Sum.

Varying label size $|L|$ (Synthetic). Using synthetic knowledge graph with size (10M,27M), we evaluated the impact of the label size by varying it from 500 to 1300. As shown in Fig. 5(e), all the algorithms take less time with larger $|L|$. This is because it takes less time for verifying patterns with less matches on average, when $|L|$ is larger.

Varying $|G|$ (Synthetic). We evaluate the scalability of our algorithms with large synthetic graphs. We make the following observations from Fig. 5(f): 1) All the algorithms scale well with increasing $|G|$; 2) AT-Sum and Stream-Sum are less sensitive to increasing $|G|$ due to their speed of covergence; and 3) AT-Sum is feasible over very large graphs. Remarkably, it summarizes a knowledge graph of size $(60M, 152M)$ within 1.5 hours. In contrast, GraMi does not run to completion within 10 hours even with graphs of size $(10M, 27M)$.

Exp-2: Effectiveness of Eval-Sum. We evaluate the efficiency of Eval-Sum, and compare it with Eval-Sum (Δ=0), Eval-RND, Eval-GraMi, and Eval-NO.

Varying $|Q|$. We set $n = 64$ and card $(\mathcal{S}_G)$=500, and varied the query size $|Q|$ from $(4, 6)$ to $(8, 14)$ over YAGO. We make the following observations from Fig. 5(g): 1) By leveraging summary patterns, Eval-Sum and Eval-Sum (Δ=0) find answers in less than 60 seconds over YAGO, and improves the query evaluation efficiency over Eval-NO by 20 and 30 times respectively; 2) On average, Eval-Sum and Eval-Sum (Δ=0) are 2.5 and 4 times faster than Eval-GraMi. This demonstrates that the summary patterns captured by simulation preorder are more effective than frequent subgraph patterns for query evaluation; 3) Our summary pattern selection strategy is effective: Eval-Sum is 2-2.5 times faster than Eval-RND that employs random selection. Eval-NO does not run to completion within 10^4 seconds for queries with more than 6 nodes.

Varying n. We set $|Q|$ = (6,10), card $(\mathcal{S}_G)$=500, and varied n, the number of selected summary patterns out of $\mathcal{S}_G$ from 8 to 128. Fig. 5(h) shows that Eval-Sum takes less time with more available patterns, as more query edges will be covered. The evaluation time of Eval-Sum (Δ=0) and Eval-NO are not sensitive to increasing n. Eval-RND does not benefit from more patterns, as it selects patterns randomly.

Varying $|G|$. We evaluate the scalability of Eval-Sum with large synthetic graphs. We set $|Q|$=(6,10), card $(\mathcal{S}_G)$=500, and varied the size of synthetic graph $|G|$ from (10M,27M) to (50M,126M). We make the following observations from Fig. 5(i): 1) all algorithms take longer time for large $|G|$ as expeceted; (2) Eval-Sum and Eval-Sum (Δ=0) scale better than all the other algorithms. Eval-Sum is reasonably efficient: when $|G|$=(10M,27M), Eval-Sum takes 35 seconds.

Efficiency of pattern selection. We also evaluated the efficiency of pattern selection procedure Select-Sum. We set card $(\mathcal{S}_G)$=500, and varied n from 8 to 128, and $|Q|$ from (4,6) to (8,14). As shown in Fig. 5(j), 1) Select-Sum is very efficient: it takes less than 10 seconds in all cases; 2) Select-Sum takes more time over larger queries and n, as expected.

Accuracy. We evaluated the accuracy of the query answers produced by Eval-Sum (Δ=0), Eval-RND, and Eval-GraMi. Let $Q(G)_A$ be the set of node and edge matches returned by a query evaluation algorithm A, and $Q(G)$ the exact match set. We define the accuracy of algorithm A as the Jaccard similarity $\frac{Q(G)_A \cap Q(G)}{Q(G)_A \cup Q(G)}$. For Eval-NO, the accuracy is 1.

We perform evaluation along two dimensions: 1) n, the number of selected summary patterns; and 2) Δ, specified as the allowed fraction of entities to be visited in $|G|$ besides summarization. As shown in Fig 5(k) and 5(l), all algorithms perform better with larger n, and Eval-Sum achieves highest accuracy with Δ = 1.5%. Remarkably, Eval-Sum can get 100% accuracy with 7.5% of original graph, however, Eval-GraMi needs more data compared with Eval-Sum.

Query diversity. We evaluate the performance of Eval-Sum (Δ=1.5%) over three categories of queries over YAGO: 1) **Frequent**, where each query node is associated with most frequent labels in G; 2) **Diversified**, where the query node labels range over a diversified set of labels; and 3) **Mixed** that combines queries uniformly sampled from the two categories. The table below shows the results of Eval-Sum on these three query sets, where C (resp. C_{ISO}) refers to the total number of nodes and edges including summarization visited by Eval-Sum (resp. Eval-NO).

	Time(seconds)	Accuracy	$\frac{C}{C_{ISO}}$	$\frac{C}{\lvert G\rvert}$
Diversified	32.46	0.9292	0.2274	0.036
Frequent	43.71	0.9442	0.1583	0.057
Mixed	34.76	0.9367	0.1822	0.044

Eval-Sum takes more time for **Frequent** queries due to large candidates for frequent labels and visits more entities with different labels for **Diversified** queries. In general, it visits no more than 23% (resp. 6%) data visited by Eval-NO (resp. $|G|$) to achieve accuracies not less than 92%.

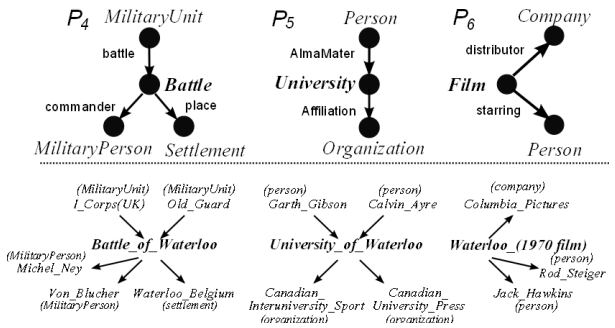


Figure 6: Real-life Summary Patterns: DBpedia.

Exp-3: Case study. To demonstrate the effectiveness of summary patterns, we show in Fig. 6 three real-life summary patterns discovered by Approx-Sum in DBpedia: 1) Summary pattern P_1 summarizes Battle entities (*e.g.,*Battle of Waterloo), and related commanders and units; 2) P_2 summarizes famous persons and their almaMaters (*e.g.,*Univerisity of Waterloo), with their organization information; and 3) P_3 identifies films (*e.g.,*Waterloo) with distributors and actor information.

Interestingly, for ambiguous keyword *e.g.,* "Waterloo", these summary patterns may 1) help user to locate reasonable matches with proper type; 2) suggest intermediate keywords as enhanced queries (*e.g.,*Military Person); and 3) can also suggest answers for *e.g.,* Précis queries [32] that find diversified facts of entities. In contrast, most top patterns from GraMi (not shown) are frequent edges (*e.g.,* (footballer, award), (footballer,country)), and stars as the union of frequent edges. These are less informative and typically redundant.

Summary. Our main experimental findings are as follows:

1) Knowledge graph summarization captured by summary patterns can be effectively and efficiently constructed. It is not very expensive to mine diversified summary patterns over large real-world graphs. For example, Approx-Sum takes 100 seconds on YAGO with size (1.54M,2.37M); 2) Our anytime summarization algorithm provides a principled way to trade-off accuracy and time. It can produce high-quality summarizations in relatively less time (*e.g.,* 180 seconds over YAGO); 3) Query evaluation leveraging summarization improves the efficiency of its naive counterpart by orders of magnitude. For example, Eval-Sum is 20 times faster than Eval-NO for YAGO. Additionally, the summary patterns based on simulation provide better improvement when compared to its frequent subgraph patterns counterpart: Eval-Sum is 2.5 times faster than Eval-GraMi; and 4) Our summary pattern selection strategy is effective: Eval-Sum outperforms Eval-RND by 1.5-2 times for 64 patterns. Also, it does not take much additional cost (up to 5% of graph size) for Eval-Sum (Δ=0) to find exact answers.

7. Conclusions

We have studied knowledge graph summarization and query evaluation problems using a class of summary patterns. We formulated a bi-criteria objective for summarization problem, and developed efficient pattern mining and summarization algorithms. We also developed efficient query evaluation algorithms by accessing a small number of diversified summary patterns and their matches. Our experimental results demonstrated the efficiency and effectiveness of our algorithms. Our case study also suggests that summary patterns can be applied for effective query suggestion.

Future work includes developing parallel and distributed algorithms for computing summarization and querying large-scale knowledge graphs using summarization; and designing concrete algorithms for query suggestion/refinement guided by summarization and conducting large-scale user studies for robust evaluation.

8. References

[1] Full version. *http://eecs.wsu.edu/∼yinghui/full.pdf*.

[2] Neo4j project. *http://neo4j.org/*.

[3] Yago2. *mpi-inf.mpg.de/yago*.

[4] C. C. Aggarwal and H. Wang. A survey of clustering algorithms for graph data. In *Managing and Mining Graph Data*, pages 275–301. 2010.

[5] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms. In *VLDB*, pages 914–925, 2007.

[6] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, pages 76–85, 2007.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.

[8] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to Web graph compression with communities. In *WSDM*, 2008.

[9] D. Bustan and O. Grumberg. Simulation-based minimization. *TOCL*, 4(2):181–206, 2003.

[10] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367–1372, 2004.

[11] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.

[12] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7):517–528, 2014.

[13] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1), 2010.

[14] W. Fan, X. Wang, and Y. Wu. Answering graph pattern queries using views. In *ICDE*, 2014.

[15] R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Automated Reasoning*, 2003.

[16] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.

[17] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations research letters*, 21(3):133–137, 1997.

[18] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

[19] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *ICDE*, pages 953–962, 2008.

[20] 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.

[21] N. S. Ketkar, L. B. Holder, and D. J. Cook. Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 71–76. ACM, 2005.

[22] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[23] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the SIAM International Conference on Data Mining (SDM), Philadelphia, PA*. SIAM, 2014.

[24] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *WWW*, 2011.

[25] W. Le, F. Li, A. Kementsietsidis, and S. Duan. Scalable keyword search on large rdf data. *TKDE*, 26(11):2774–2788, 2014.

[26] Z. Liu and Y. Chen. Query results ready, now what? *IEEE Data Eng. Bull.*, 33(1):46–53, 2010.

[27] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *SIGIR*, 2011.

[28] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

[29] F. Pennerath and A. Napoli. The model of most informative patterns and its application to knowledge extraction from graph databases. In *Machine Learning and Knowledge Discovery in Databases*, pages 205–220. 2009.

[30] B. Quilitz and U. Leser. Querying distributed RDF data sources with sparql. In *ESWC*, pages 524–538, 2008.

[31] M. Riondato, D. Garcia-Soriano, and F. Bonchi. Graph summarization with quality guarantees. In *ICDM*, pages 947–952, 2014.

[32] A. Simitsis, G. Koutrika, and Y. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal*, 17(1):117–149, 2008.

[33] M. Sydow, M. Pikula, and R. Schenkel. Diversum: Towards diversified summarisation of entities in knowledge graphs. In *ICDE Workshops*, pages 221–226, 2010.

[34] M. Sydow, M. Pikuła, and R. Schenkel. To diversify or not to diversify entity summaries on rdf knowledge graphs? In *Foundations of Intelligent Systems*. 2011.

[35] M. Sydow, M. Pikula, R. Schenkel, and A. Siemion. Entity summarisation with limited edge budget on knowledge graphs. In *IMCSIT*, 2010.

[36] Y. Tian, R. Hankins, and J. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, 2008.

[37] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan. Summarizing answer graphs induced by keyword queries. *Proceedings of the VLDB Endowment*, 6(14):1774–1785, 2013.

[38] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295, 2003.

[39] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *PVLDB*, 7(7):565–576, 2014.

[40] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *ICDE*, 2010.

9. Appendix: Proofs and Algorithms

Proof of Prop. 1. To prove Prof. 1, it suffice to show (a) Lemma 2, and (b) the condition in Lemma 2 can be checked in PTIME.

Proof of Lemma 2. We first show the **If** condition. Consider a knowledge graph G that matches P via a simulation preorder R. Denote a path ρ_p connecting two pattern nodes u_1 and u_n in P as ρ_p=$\{u_1, \ldots, u_n\}$. For every node $v_1 \in [u_1]$, there must exist a child $v_2 \in [u_2]$, such that (v_1, v_2) is an edge with the same label of edge (u_1, u_2). Hence the path (a single edge) from v_1 to v_2 has the same label with (u_1, u_2). By induction, for every node $v_1 \in [u_1]$, we can find a path from v_1 to a node $v_n \in [u_n]$ with the same label, i.e., the path consists of a sequence of edge matches, one for each pattern edge on ρ_p, following the definition of simulation preorder.

We prove **Only If** condition by contradition. Assume that P is a summary pattern of G, while a knowledge graph G does not matches P via simulation preorder. Then there exists at least a node u in P, such that no node in G either has the same label as u, or does not have a neighbor v' with the required label of a corresponding neighbor v of u. In either case, the path from u to v in P has no corresponding path in G. Hence, P is not a summary pattern. This contradicts our assumption.

The above completes the proof of Lemma 2.

Matching algorithm. It is verified that it takes $O((|V_P| + |V|)(|E_P|+|E|))$ time to determine if G matches P via simulation [13, 18]. Specifically, the algorithm initializes a match set $C(u)$ for each pattern node u in P with all the entities having the same label of u in G. It then iteratively checks, for each entity $v \in C(u)$ and each edge (u, u') in P, if there exists an edge (v, v') with the same label, where $v' \in C(u')$. If v has no such child v', it removes v from $C(u)$. The process repeats until no entity can be removed from any match set. If the match set is not empty for all the pattern nodes, P is a summary pattern.

Putting these together, Prop. 1 follows.

Proof of Corollary 3. We outline an algorithm to compute Supp(P, G) below. (1) We invoke a pattern matching algorithm [13, 18] to verify if P is a summary pattern of G or not. (2) If P is a summary pattern, we compute the total size of the node and edge matches induced by the simulation preorder R (Lemma 2) as $|G_P|$. (3) Supp(P, G) can then be computed as the ratio of $|G_P|$ to the candidate graph $|G_c|$. It is easy to verify that the algorithm takes in total $O(|V_P|+|V|)(|E_P|+|E|)$ time. In contrast, the conventional support for frequent patterns requires intractable subgraph isomorphism test [12].

Incremental evaluation in candidateGen. The process of computing $I(P)$ for every newly generated pattern P is very expensive. Therefore, candidateGen performs incremental pattern evaluation to avoid redundant computation. During the mining process, it keeps track of the base graphs of all the generated patterns. For a newly generated pattern P, it considers two cases: 1) If P consists of a single edge, it computes all the edge matches of P as its match set $C(P)$; 2) For each pattern P obtained by merging two patterns P_1 and P_2, incSum initializes $C(P)$ with $C(P_1) \cup C(P_2)$, and iteratively removes the candidates in $C(P)$ that fails the matches for summary patterns by definition: a) Removes a set of edges e=(v_1, v_2) such that v_1 or v_2 fails the match condition in the merged pattern P (*e.g.,* having no edge with the required labels), and adds those edges to an *affected edge* set AFF; and b) Iteratively verifies if each edge e' adjacent to an edge e in AFF is an invalid edge match. If yes, it removes e' from $C(P)$ and adds it to AFF. This process is repeated until no more edges can be removed from $C(P)$. $I(P)$ is computed from the final $C(P)$.

Analysis of Algorithm Approx-Sum. We can show that Approx-Sum approximates the optimal summarization with ratio 2. Once all the closed patterns are identified, the instance of graph summarization problem can be transformed to an instance of the *maximum dispersion problem* [17]. The problem of maximum dispersion is to find, in a weighted complete graph, a subgraph of size k with maximum node and edge weights. Given a closed pattern set $\mathcal{C}_P$, we construct a complete graph G_c where each node v_P represents a summary pattern $P \in \mathcal{C}_P$ with weight $I(P)$, and each edge $(v_P, v_{P'})$ carries a weight $D(P, P')$. Given a set of k summary patterns $\mathcal{S}_G$ and a corresponding set of nodes V_c in G_c, we define a cost function $F_c(V_c)$=$\sum_{v_p, v'_p \in V_c} F'(P, P')$, where $F'(P, P')$ is defined in algorithm Approx-Sum (line 4). We can easily verify that $F_c(V_c)$= $(1-\alpha)\sum I(P) + \frac{\alpha}{k-1}\sum D(P, P')$=$F(\mathcal{S}_G)$ ($P \neq P' \in \mathcal{S}_G$), where $F(\cdot)$ is the interestingness function (Section 3). Hence, $\mathcal{S}_G$ contains top-k closed summary patterns that maximizes $F(\mathcal{S}_G)$ if and only if the set V_c maximizes $F_c(V_c)$. The algorithm Approx-Sum simulates a 2-approximation greedy selection strategy for maximum dispersion problem [17]. Hence, it returns a set $\mathcal{S}_G$ of closed summary patterns that approximates optimal summarization with ratio 2.

Time complexity analysis. It takes $O(N(b + |V|)(b + |E|))$ time to generate and evaluate a total of N summary patterns with size bounded by b in the procedure candidateGen; and $O(\frac{k}{2}|\mathcal{C}_P|^2)$ time to compute $\mathcal{S}_G$ using the greedy strategy, where $\mathcal{C}_P$ refers to the candidate set of closed summary patterns. Hence, the overall time complexity of Approx-Sum is bounded by $O(N(b+|V|)(b+|E|)+\frac{k}{2}N^2)$, where N is the number of summary patterns generated in candidateGen.

Approximation of algorithm AT-Sum. The algorithm AT-Sum always terminates, and correctly returns a set of summary patterns $\mathcal{S}_G$. We can show that it is an anytime 2-approximation algorithm. First, AT-Sum is an anytime algorithm: 1) It can be terminated for any time-bound t to return $\mathcal{S}_G$; and 2) the set $\mathcal{S}_G$ is incrementally updated such that the overall quality $F(\mathcal{S}_G)$ monotonically improves (line 5). Let $\mathcal{C}_{P_t}$ be the set of patterns generated upto time t, and $\mathcal{S}^*_{G_t}$ be the optimal summarization over candidate set $\mathcal{C}_{P_t}$. Importantly, for l_p=k-1, AT-Sum simulates its 2-approximation counterpart Approx-Sum by accessing $\mathcal{C}_{P_t}$, and produces a summarization $\mathcal{S}_{G_t}$ where $F(\mathcal{S}_{G_t}) \geq \frac{\mathcal{S}^*_{G_t}}{2}$ for a given time t (see details in [1]).

AT-Sum only need to store the top $k-1$ pattern pairs in each list $L_i \in \mathcal{L}$ that maximizes $F'(\cdot)$. We can see this based on the following observations: a) AT-Sum identifies at most $\lfloor \frac{k}{2} \rfloor$ pairwise disjoint pattern pairs each time a new pattern is returned by candidateGen (line 5); and b) for each new pattern, at most $2(\lfloor \frac{k}{2} \rfloor - 1)$ pattern pairs are replaced/removed in each list of $\mathcal{L}$. Hence, to preserve the 2-approximate answer, it suffices to maintain the top $k-1$ pairs (P_i, P_j) for

each pattern P_i that maximizes $F'(\cdot)$ score.

Proof of Lemma 9. We first show the **If** condition. It suffices to show that for graph G, the match set M'_e of the edge e' in Q (via subgraph isomorphism) is contained in the match set C_e of the edge e in P (via simulation). For any edge $e'' \in M'_e$, there exists a bijection that maps e' to e''. Since the edge e' in Q matches edge e in P via graph simulation, we can verify that e'' in G also matches e in P via simulation. Hence, $e'' \in C_e$, *i.e.,* $M'_e \subseteq C_e$, which means edge e' is covered by the pattern P.

We show the **Only If** condition by contradiction. If e' is not a match for any edge e in P, then a graph G' isomorphic to the union of Q and P can be constructed, such that the match set $M'_e$is not contained in the base graph of P in G.

Analysis of Algorithm Select-Sum. The algorithm Select-Sum correctly selects a set $\mathcal{P}_\mathcal{M}$ of n summary patterns and computes a mapping λ to be used in Eval-Sum. The approximation ratio of Select-Sum can be verified by an approximation-preserving reduction to the budgeted maximum coverage problem (BMaxCover) [22]: Given a weighted set U and a collection S of subsets of U with associated cost, and a budget L, the BMaxCover problem is to find a subset $S' \subseteq S$ such that the total cost of S' is bounded by L, and the total weights of the elements in U covered by S' is maximized. The reduction treats the edge set of Q as U with unit weight, $\mathcal{S}_G$ as S, and for each summary pattern $P \in \mathcal{S}_G$, the edge set of Q covered by P as a set $S_P \in S$, with a cost defined as the base graph size $|G_{b_P}|$. Select-Sum algorithm mimics the greedy approximation algorithm in [22] (with approximation ratio $1 - \frac{1}{e}$) for budgeted summary pattern selection, and preserves the approximation ratio $1 - \frac{1}{e}$.

For time complexity, Select-Sum takes $O(\mathsf{card}(\mathcal{S}_G)|Q|^2 + |Q||\mathcal{S}_G| + |\mathcal{S}_G|^2)$ time to compute the edge set covered by each summary pattern in $\mathcal{S}_G$. In each iteration, it takes $O(\mathsf{card}(\mathcal{S}_G)|Q|)$ time to find a summary pattern with the largest $r(\cdot)$, and performs at most n iterations. Hence, the running time of Select-Sum is $O(\mathsf{card}(\mathcal{S}_G)|Q|^2 + |Q||\mathcal{S}_G| + |\mathcal{S}_G|^2)$, where $|Q|$ and $\mathsf{card}(\mathcal{S}_G)$ are often smaller than $|\mathcal{S}_G|$. This completes the proof of Prop. 7.