Parallel Graph Summarization for Knowledge Search

Qi Song
Washington State University
qsong@eecs.wsu.edu

Peng Lin
Washington State University
plin1@eecs.wsu.edu

ABSTRACT

Querying heterogeneous and large-scale knowledge graphs are typically expensive. This paper studies a parallel graph summarization framework to facilitate knowledge graph search. (1) We propose a class of reduced summaries characterized by graph patterns, which are capable of summarizing entities in terms of their neighborhood similarity up to a certain hop. (2) We study a bi-criteria diversified summarization problem. Given a knowledge graph $G$, it is to discover top-$k$ diversified reduced summaries with maximized quality in terms of both informativeness and diversity. (3) We show that diversified summarization is feasible for large knowledge graphs, by developing a parallel approximation algorithm with quality guarantees. We show that the algorithm is parallel scalable, which ensures the feasibility of summarization in large graphs. Using real-world graphs, we experimentally verify the efficiency of our sequential and parallel summarization algorithms, and query evaluation guided by summarization.

1 INTRODUCTION

Knowledge graphs are routinely used to represent entities and their relationships in knowledge bases [2, 9]. Unlike relational data, real-world knowledge graphs lack the support of well-defined schema and typing system. To search knowledge graphs, a number of query processing techniques are proposed [9, 14, 18, 28]. Nevertheless, it is hard for end-users to specify precise queries that will lead to meaningful answers without prior knowledge of the underlying graph. Searching knowledge graphs is challenging due to the ambiguity in queries, the inherent computational complexity (e.g., subgraph isomorphism [9, 18]) and possible resource constraints (e.g., data allowed to be accessed, response time) [6].

Example 1: Fig. 1 illustrates a sample knowledge graph $G$ of artists and bands. Suppose a music publisher wants to find artists who are experts in two genres (genre), acted in a film, and also collaborated with a band whose manager is located in the same country as the band. This search can be represented as a graph query $Q$ [9, 14, 18, 28] as shown in Fig 1. The answer of $Q$ refers to the set of entities typed with an artist in the subgraphs of $G$ that are isomorphic to $Q$. In this example, T.McGraw is the correct answer for $Q$.

It is costly to evaluate $Q$ when $G$ is large. For example, the ambiguous label “artist” requires the inspection of all the entities having the type. It is also hard for the users to specify $Q$ without prior knowledge of $G$. Moreover, subgraph isomorphism is NP-hard.

Observe that graph $G$ can be “summarized” by three small graph patterns $P_1$, $P_2$ and $P_3$, as illustrated in Fig. 1. Each pattern abstracts a fraction of $G$, by representing a group of entities as a single node, along with their common neighboring entities in $G$. For example, $P_1$ specifies three artists J.Browne, T.McGraw and D.Yoakam in $G$ as a single node, indicating “musicians”; and $P_3$ distinguishes the artists T.Hanks and M.Ryan who are associated with only film and country (i.e., “actors”). These concise summaries help the users in understanding $G$ without a daunting inspection of low-level entities.

One can further use these patterns as “views” [4, 13] to speed up knowledge search. For example, $Q$ can be correctly answered by only referring to the entities summarized by $P_1$ and $P_3$ in $G$, as all the matches of $Q$ are contained in those entities.

The above example suggests that graph patterns can benefit knowledge search by suggesting (and can be directly queried as) interpretable “views”. Although desirable, computing summaries is nontrivial. Summaries defined by frequent subgraphs [3, 10, 11, 14] can often be an overkill for entities with similar neighbors up to a certain hop. For example, the subgraphs induced by J.Browne and T.McGraw along with their 1 hop neighbors (Fig. 1) should be summarized by $P_1$, despite that they are not isomorphic to each other. Mining summaries is also expensive when $G$ is large.

Contributions. This paper studies a new parallel graph summarization framework to compute diversified summaries for knowledge graph query evaluation.

(1) We review a class of summaries namely, $d$-summaries [21], and introduce reduced $d$-summaries. A $d$-summary is characterized by a $d$-matching relation, which encodes a lossy representation of similar entities and their $d$-hop neighbors. A reduced summary encodes a minimum representation of all equivalent summaries.
We show that $d$-summary is feasible in practice, by studying the verification and reduction problems. The verification problem is to check whether a graph pattern is a $d$-summary. The reduction problem computes a reduced counterpart of a summary. We show that both problems are tractable (Section 2) for $d$-summarization.

(2) We study a bi-criteria function to quantify the quality of summaries that integrates both informativeness and diversity measures (Section 3). Based on the quality function, we introduce the problem of diversified graph summarization. Given a knowledge graph $G$, integers $k$ and $d$, the problem is to compute a set of $k$ $d$-summaries that maximizes the bi-criteria function.

(3) The diversified summarization problem is (not surprisingly) NP-hard. Nevertheless, we show that it is feasible to discover diversified summaries in large graphs, by developing a new parallel diversified summarization algorithm (Section 4). The algorithm has the parallel scalability, a guarantee to reduce response time with the increase of processors. This ensures the feasibility of summarization in large graphs by adding processors.

(4) Using real-world knowledge bases and synthetic graphs, we experimentally verify the effectiveness and efficiency of our summarization and query-evaluation algorithms (Section 5). We found that it is feasible to compute diversified summaries over real-world knowledge graphs. Our case studies also verify that summary-based querying supports “cross-domain” querying by accessing summaries from multiple knowledge bases.

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 [11, 17, 19, 23, 25, 26]. (1) Graph compression aims to compress graphs within a bounded error by minimizing an information complexity measure [11, 17, 19], e.g., Minimum Description Length (MDL). The algorithm in [11] employs clustering and community detection to describe the data graph with predefined frequent structures (vocabulary) including stars and cliques. (2) Summarization techniques attempt to construct summaries over attributed graphs, where nodes with similar attributes are clustered in a controlled manner using parameters such as participation ratio [25]. (3) Bi-simulation relation is adopted to group paths carrying same content [26]. It summarizes the entities only when they are pairwise similar, which can be an overkill for knowledge graphs. Entity summarization [23] generates diversified answers for entity search instead of general queries.

Our work differs from these works in the following ways: (1) We introduce lossy summaries for knowledge query evaluation, rather than to compress the graphs [11, 17, 19]. (2) We discover summaries to access single graphs rather than for query answers [23, 26], and it can be applied for diversified result summarization. (3) The summaries are measured in terms of both informativeness and diversity, which is more involved than MDL-based measures [17, 19]. (4) In contrast to [17, 25], our summary model requires little parameter tuning effort. In addition, diversified summaries are not addressed in these works.

Parallel graph pattern mining. Frequent subgraph patterns can be mined from a single graph to describe large graphs [3, 11]. Redundancy-aware mining is studied for general patterns [27]. Type-based summarization is applied to facilitate keyword search in RDF graphs [14]. In contrast, we capture summaries in terms of approximate pattern matching rather than strict subgraph isomorphism [3, 14]. Bi-criteria summarization that integrates diversification and informativeness is not addressed in these works.

Parallel algorithms have been developed for pattern mining in terms of subgraph isomorphism, for transactional graph databases [1] or single graph [24]. Vertex-centric models (e.g., Pregel [16] and GraphLab [15]) are developed to parallelize sequential graph query processing. These methods can not be readily applied for diversified summarization in a single graph. In contrast to these works, (1) Our methods discover diversified graph summaries in terms of approximate pattern matching, and (2) We propose an algorithm with parallel scalability guarantee which has not been mentioned in previous works.

Querying using views: View-based query evaluation has been shown to be effective for SPARQL queries [13] and general graph pattern queries [4]. It requires equivalent query rewriting by accessing views defined in the same query language. In contrast, we show that $d$-summaries can be used as views for graph queries defined by subgraph isomorphism, rather than requiring queries to be in the same semantics. We also develop parallel summarization algorithms. These are not addressed in [4, 13].

2 KNOWLEDGE GRAPH SUMMARIZATION

2.1 Graphs and Summaries

We start with the notions of knowledge graphs and summaries.

Knowledge graphs. A knowledge graph $G$ is a directed labeled graph $(V, E, L)$ with a set of nodes $V$ and a set of edges $E \subseteq V \times V$. Each node $v \in V$ has a label $L(v)$ that may carry the content of $v$ (e.g., type, name, attribute values) as found in knowledge bases and property graphs [9]; and each edge $e \in E$ has a label $L(e)$.

We do not assume a standard schema over $G$, and our techniques will benefit from such a schema, if exists. Fig. 1 depicts a fraction of a typed knowledge graph. Each entity (e.g., Brown) has a label that carries its type (e.g., artist), and connects to other typed entities (e.g., band) via labeled relationships (e.g., collaborated).

Summaries. We review the notions of summaries in [21] below. Given a knowledge graph $G$, a $d$-summary $P$ of $G$ is a connected graph pattern $(V_p, E_p, L_p)$, where $V_p$ (resp. $E_p \subseteq V_p \times V_p$) is a set of summary nodes (resp. edges). Each node $u \in V_p$ (resp. edge $e \in E_p$) has a label $L_p(u)$ (resp. $L_p(e)$), and represents a non-empty node set $[u]$ (resp. edge set $[e]$) from $G$.

$d$-matching [21]. Given a pattern $P$ and a graph $G$, a backward (resp. forward) $d$-matching from $P$ to $G$ is a nonempty binary relation $R^d$ $\subseteq V_p \times V$ (resp. $R^d$ $\subseteq V_p \times V$), where

- $(u, v) \in R^d_1$ and $(u, v) \in R^d_0$ if $L_p(u) = L(v)$;
- $(u, v) \in R^d_1$ if $(u, v) \in R^d_{d-1}$, and for every parent $u'$ of $u$ in $P$, there exists a parent $v'$ of $v$ in $G$, such that $L_p(u', u) = L(v', v)$ (i.e., edges $(u', u)$ and $(v', v)$ have the same edge label), and $(u', v') \in R^d_{d-1}$;
(u, v) \in R_d^1 \text{ if } (u, v) \in R_d^0 \text{ and for every child } u' \text{ of } u \text{ in } P,
\text{there exists a child } v' \text{ of } v \text{ in } P \text{ such that } L_p(u, u') = L(v, v'),
\text{and } (u', v') \in R_d^1.

We define a d-match, \( R_d \), between \( P \) and \( G \) as the set of node pairs \( \{(u, v) | (u, v) \in R_d^0 \cap R_d^1 \} \). We say \( P \) is a \( d \)-summary of \( G \) (denoted as \( P \sim \sim G \)), if for every summary node \( u \) and every node \( v \in \{u\} \cup \{u\} \neq \emptyset \), \((u, v) \in R_d \). The base graph of \( P \) in \( G \), denoted as \( G_P \), refers to the subgraph of \( G \) induced by the node set \( \bigcup_{u \in V_P} \{u\} \), and the edge set \( \bigcup_{e \in E_P} \{e\} \), for each \( u \in V_P \) and \( e \in E_P \).

Intuitively, a \( d \)-summary \( P \) guarantees that \( G \) preserves the parent and child relation of each summary node up to hop \( d \). Note that a base graph can be disconnected for a connected summary. Given a knowledge graph \( G \) and an integer \( d \), a summarization \( S_G \) of \( G \) is a set of \( d \)-summaries. In practice, additional mapping structures can be used to trace the base graphs for the summaries.

Example 2: Fig. 1 illustrates a summarization of the knowledge graph \( G \), which contains three 2-summaries, \( P_1, P_2, \) and \( P_3 \). The base graph of \( P_1 \) contains the entities: \{\textit{genre}:\{country, punk\}, \{\textit{film}:\{Going Home, Four_Holidays\}, \{actor:It\}=T.McGraw, D.Yoakam, J.Browne\}, \{\textit{band}:\{The_Eagles, Husker_Du,Def_Lepppard\}\}. Note that \( P_1 \) cannot summarize T.Hanks as the latter has no path to a band as required by \( P_1 \). Similarly, \( P_2 \) summaries band Def_Lepppard and The_Eagles, and their associated company and manager in \( G \), and \( P_3 \) summaries You’ve got a Mail and Sleepless in Seattle, actors T.Hanks and M.Ryan in their countries.

Summary verification. Given a graph pattern \( P \), a knowledge graph \( G \), and an integer \( d \), the verification problem is to check if \( P \) is a \( d \)-summary of \( G \), and if so, identify the largest base graph \( G_P \) of \( P \) in \( G \). In contrast to its counterpart defined by frequent subgraphs (NP-hard), the verification of \( d \)-summaries is tractable.

Lemma 2.1. Given a summary \( P=(V_P, E_P, L_P) \), an integer \( d \), and a graph \( G=(V,E,L) \), it is in \( O(|V_P|(|V_P|+|V|)(|E_P|+|E|)) \) time to verify if \( P \) is a \( d \)-summary of \( G \).

As a proof of Lemma 2.1, we outline an algorithm, denoted as valiSum, that determines if \( P=(V_P, E_P, L_P) \) is a \( d \)-summary in polynomial time. The algorithm valiSum first initializes a match set \([u]=(\{(u, v) | (u, v) \in R_d^0 \cup R_d^1 \} | u \in V_P \}) \). It then refines the match sets as follows. (1) It first computes the forward \( d \)-similarity relation \( R_d^1 \). For each edge \((u', u) \in E_P \), it iteratively removes all the nodes \( v' \in [u'] \) if there exists no child of \( v' \) in \( G \) such that \((u, v) \in R_d^{i-1} \), for \( i \in [1,d] \). This process repeats until no change can be made to \([u]\), for each node \( u \) in \( V_P \). (2) It continues to refine the match sets derived from (1), by removing the nodes that do not satisfy the backward \( d \)-similarity relation. (3) If for every node \( u \in V_P \), \([u] \neq \emptyset \), \( P \) is verified as a \( d \)-summary. Otherwise, it determines that \( P \) is not a \( d \)-summary.

The algorithm follows the following invariants: (1) For any pair \((u, v) \in R_d^0 \cap R_d^1 \), \( u \in [u] \) when it terminates. (2) If a node \( v \) is removed from \([u]\) at any time, then \((u, v) \notin R_d \). Hence it correctly computes the largest \( R_d \) and \( G_P \). To see the complexity, observe that it takes \( O(|V_P|(|V_P|+|V|)(|E_P|+|E|)) \) time to verify forward and backward \( d \)-similarity for a single summary node in \( V_P \). Thus the total verification time is \( O(|V_P|(|V_P|+|V|)(|E_P|+|E|)) \).

3 DIVERSIFIED SUMMARIZATION

We next introduce a bi-criteria function that captures the quality of summaries in terms of both informativeness and diversity.

Informative Summaries. We are interested in informative summaries that capture more information. This can be measured by an informativeness function defined as:

\[ I(P) = \frac{|P|}{b_P} \cdot \text{supp}(P, G) \]

where \(|P|\) (the size of \( P \)) is defined as the total number of nodes and edges in \( P \), \( b_P \) is a size bound to normalize \(|P|\), which can be specified as a recognition budget (i.e., the largest summary size a user can understand) [23], and the support \text{supp}(P, G) is defined as \( |G_P| \), where \(|G_P|\) (resp. \(|G|\)) is the size (i.e., the total number of nodes and edges) in \( G_P \) (resp. \( G \)). That is, the informativeness \( I(\cdot) \) favors larger summaries that also have higher support.

Example 3: Consider the 2-summaries \( P_1, P_2 \) of the graph \( G \) (with 45 entities and edges) in Fig. 1. Let the summary size bound be \( b_P=8 \), we can verify that the size of the base graph \( |G_{P_1}| \) is 20. Hence, \text{supp}(P_1, G)=20 \text{ and the informativeness of } P_1 \text{ is } 20 \times 20=0.39. Similarly, \text{supp}(P_2, G)=20 \text{ and the informativeness of } P_2 \text{ is } 20 \times 20=0.39.

Diversified Summaries. A second challenge is to diversify the summaries in terms of the entities they summarized. We introduce a distance function for summaries.

Distance function. Given summaries \( P_1 \) and \( P_2 \), their distance is quantified by the distance function \( \text{diff} \) defined as:

\[ \text{diff}(P_1, P_2) = 1 - \frac{|V_{G_{P_1}} \cap V_{G_{P_2}}|}{|V_{G_{P_1}} \cup V_{G_{P_2}}|} \]

where \( V_{G_{P_1}} = \bigcup_{u \in V_{P_1}} \{u\} \) (resp. \( V_{G_{P_2}} = \bigcup_{u \in V_{P_2}} \{u\} \)); that is, it measures the Jaccard distance between the set of entities summarized by \( P_1 \) and \( P_2 \) in their base graphs. One can verify that \( \text{diff} \) is a metric, i.e., for any three \( d \)-summaries \( P_1, P_2, \) and \( P_3 \), \( \text{diff}(P_1, P_2) \leq \text{diff}(P_1, P_3) + \text{diff}(P_2, P_3) \). Here we quantify entity set difference as a more important factor of summary difference. Label/type difference of the entities can also be applied to quantify weighted \( V_{G_{P_1}} \) in the distance function \( \text{diff} \).

Example 4: Consider the 2-summaries \( P_1, P_3 \) of the graph \( G \) in Fig. 1. The differences are calculated as follows: \( \text{diff}(P_1, P_2)=1-\frac{1}{8}=0.86 \), as they summarize two common entities \{The_Eagles, Def_Lepppard\}. Similarly, \( \text{diff}(P_1, P_3)=1.00 \), and \( \text{diff}(P_2, P_3)=0.90 \).

Reduced summaries. While the distance function captures the difference of summaries in terms of their base graphs, informative summaries may contain redundant pattern nodes and edges that can be further reduced. This is not discussed in [21].

Example 5: Consider three summaries: \( P_1 \) (Example 1), \( P_1^2 \), and \( P_1^4 \) as illustrated in Fig. 2. We can verify that these three summaries have a same base graph in \( G \) (Fig. 1). Although \( P_1^2 \) and \( P_1^4 \) are larger than \( P_1 \), they contain “redundant” nodes \{e.g., \textit{film} and band in \( P_1^2 \), and artist in \( P_1^4 \)\} that do not contribute new information, hence should be “reduced” to a concise summary \( P_1 \).
Given two d-summaries $P_1$ and $P_2$, we say $P_1$ and $P_2$ are equivalent, denoted as $P_1 \sim P_2$, if there exists a d-matching $R_{12}$ from $P_1$ to $P_2$ (denoted as $P_1 \leq P_2$), and its inverse relation $R_{12}^{-1}$ is a d-matching from $P_2$ to $P_1$ (denoted as $P_1 \leq P_2$). The following result bridges summary equivalence and their support.

**Lemma 3.1.** For any graph $G$ and its two d-summaries $P_1$ and $P_2$, $\text{supp}(P_1,G) = \text{supp}(P_2,G)$ if $P_1 \sim P_2$.

**Proof:** Given two equivalent summaries $P_1=(V_1, E_1, |P_1|)$ and $P_2=(V_2, E_2, |P_2|)$, it suffices to show that $|G_{P_1}| = |G_{P_2}|$, where $G_{P_1}=(V_1, E_1, |P_1|)$ (resp. $G_{P_2}=(V_2, E_2, |P_2|)$) refers to the base graph of $P_1$ (resp. $P_2$). Denote as the d-matching relation from $P_1$ (resp. $P_2$) to $G$ as $R_{12}$ (resp. $R_{21}$). (1) As $P_1 \leq P_2$, for every node $u_2 \in V_2$, there exists a node $u_2 \in V_1$ such that $(u_1, u_2) \in R_{12}$. As $P_2$ is a d-summary of $G$, by definition of d-matching, any match of $u_2$ in $G$ is also a match of $u_1$. Thus, $V_2 \subseteq V_1$. Similarly, as $P_2 \leq P_1$, $V_1 \subseteq V_2$. That is, $V_1=V_2$. (2) Similarly, we can verify that $E_1=E_2$. Putting these together, $|G_{P_1}| = |G_{P_2}|$. Thus, $\text{supp}(P_1,G) = \text{supp}(P_2,G)$.

A summary $P$ is a reduced summary, if there exists no smaller summary $P'$, such that $P \sim P'$. Indeed, a reduced summary is a smallest representation of its equivalent summaries, without losing the entities it can summarize in $G$. Better still, the result below verifies that a summary $P$ can be efficiently “reduced”.

**Lemma 3.2.** Given a summary $P=(V_P, E_P, |P|)$, it is in $O((|V_P| + |E_P|)^2 + |V_P|^2)$ time to compute a reduced summary $P_r$ of $P$.

As a proof of Lemma 3.2, we provide a reduction algorithm, denoted as Reduce, as follows. Given a summary $P$, **Reduce** (1) computes a d-matching $R$ from $P$ to itself, and (2) identifies all the node pairs $(u, v)$ such that $(u, v) \in R$ and $(v, u) \in R$. The node pairs forms an equivalence relation $R' \subseteq R$. It then “merges” all the nodes in the same equivalence class to a single node $[u]$, and redirects the edges to $[u]$. This yields a reduced d-summary $P_r$ of $P$.

**Analysis.** It is easy to verify that $P_r \leq P$ and $P \leq P_r$. We now prove that $P_r$ is the smallest pattern among its equivalent counterparts, by contradiction. Assume there exists a smaller summary $P'_r$ such that $P'_r \sim P_r$. Then $P'_r \sim P_r$. Denote as the equivalence relation between $P'_r$ and $P_r$ as $R'$. Then there exists at least two distinct nodes $u, u'$ in $P_r$, and a third node $v$ in $P'_r$, such that $(u, v) \in R'$ and $(v, u') \in R'$. Thus, $u, u'$ and $v$ in $P_r$ belongs to the same equivalence class. Nevertheless, $u$ and $u'$ are not merged in $P_r$. Thus, either $P'_r$ is not equivalent to $P_r$ or $|P'_r| = |P_r|$. Either leads to a contradiction.

The above reduction procedure Reduce takes $O(|V_P| + |E_P|^2)$ time to compute the equivalence relation, and $O(|V_P|^2)$ time to perform the reduction. The total cost is hence in $O((|V_P| + |E_P|^2)^2 + |V_P|^2)$ time. Lemma 3.2 thus follows.

**Example 6:** Following Example 5, Reduce finds that the two film nodes and two band nodes belong to two equivalent classes in $P_1$, thus reduces $P_1^1$ to $P_1$. $P_2^1$ can also be reduced to $P_1$ by merging all artist nodes. Note that $P_2^1$ is a reduced summary but not equivalent to $P_1$, as artist in $P_1$ can not be matched with those in $P_2^1$.

### Diversified Summarization

We now introduce a bi-criteria function $F$ that integrates informativeness $I()$ and distance diff() functions. Given a summarization $S_G$ for a knowledge graph $G$, the function $F$ is defined as:

$$F(S_G) = (1-a) \sum_{P_i \in S_G} I(P_i) + \frac{a}{|\text{card}(S_G)|} \sum_{P_i \neq P_j \in S_G} \text{diff}(P_i, P_j)$$

where $(1)\text{card}(S_G)$ refers to the number of summaries it contains; and $(2)a \in [0, 1]$ is a tunable parameter to trade-off informativeness and diversification. We scale down the second summation (diversification) which has $\frac{\text{card}(S_G) \cdot \text{card}(S_G) - 1}{2}$ terms, to balance out the fact that the first summation has $\text{card}(S_G)$ terms.

**Example 7:** Set $b_p = 8$ and $a = 0.1$, a top-2 diversified summarization $S^\ast_G$ of $G$ (Fig. 1) is $(P_1, P_2)$, where total quality score $F(S_G) = 0.9 	imes (0.39 + 0.20) + 0.1 \times 0.86 = 0.62$.

Based on the quality metrics, we next introduce a graph summarization problem for knowledge graphs.

**Problem statement.** Given a knowledge graph $G$, integers $k$ and $d$, and a size budget $b_p$, the **diversified graph summarization problem** is to compute a set of $k$ summaries $S_G^\ast$ of $G$ such that

- each summary in $S_G^\ast$ is a reduced d-summary with size bounded by $b_p$; and
- the quality function $F(S_G)$ is maximized.

That is, it finds $k$ reduced summaries that are both informative and diversified. Although desirable, it is (not surprisingly) NP-hard. The hardness can be shown by a reduction from the maximum dispersion problem [8] (a known NP-complete problem).

Despite the hardness, we show that diversified summarization is feasible over large graphs, by providing a parallel approximation algorithm in Section 4.

### 4 PARALLEL DIVERSIFIED SUMMARIZATION

We start with a sequential mining algorithm. Let $S_G^\ast$ denote the optimal summarization that maximizes the function $F$. An $\epsilon$-approximation algorithm returns a summarization $S_G^\ast$, such that $F(S_G^\ast) \geq \frac{F(S_G^\ast)}{\epsilon}$ ($\epsilon \geq 1$). We show that diversified summarization is $2$-approximable, by presenting such an algorithm.

**Approximated Summarization.** Given a graph $G$, integers $k$ and $d$, and size budget $b_p$, the algorithm, denoted as approxDis, has the following steps. (1) It invokes a mining algorithm sumGen $(G, k, d, b_p)$ to discover a set $C_p$ of reduced d-summaries. (2) It then invokes a diversification algorithm sumDiv to compute the top-k diversified summaries $S_G^\ast$ from $C_p$.

**Auxiliary structure.** Underlying approxDis is the maintenance of a reduced summary lattice $\mathcal{P}=(V_r, E_r)$ encodes the generation of d-summaries, where each node $P_r \in V_r$ at level $i$ of $\mathcal{P}$ is a reduced d-summary with $i$ edges, and there exists an edge $e_i=(P_r, P'_{r}) \in E_r$, if $P_r$ and $P'_{r}$ are two reduced d-summaries at level $i$ and $i+1$ ($j \in [1, b_p - 1]$), and $P'_{r}$ is obtained by adding an edge to $P_r$.
Procedure sumGen. Procedure sumGen follows conventional pattern mining (e.g., level-wise) to generate summaries with \( \mathcal{P} \). The difference is that it uses a validation procedure that guarantees each group of equivalent summaries is verified only once.

For each pattern \( P \), it validates if \( P \) is a reduced \( d \)-summary in two steps. (1) It first "reduces" \( P \) to its reduced counterpart with procedure Reduce in polynomial time (Lemma 2.1). (2) Given a reduced pattern \( P' \), it checks whether there exists a reduced summary \( S' \) at level-\( |P'| \) such that \( P' \sim S' \). If so, it sets \( \text{supp}(P', G) = \text{supp}(P, G) \) without verification (Lemma 3.1). Otherwise, it invokes procedure validSum (Lemma 2.1) to verify \( P' \). It stores \( \text{supp}(P', G) \) and updates \( \mathcal{C}_P \) if \( P' \) is a valid \( d \)-summary.

Procedure sumDiv. Given a set of reduced summaries \( \mathcal{C}_P \), procedure sumDiv greedily adds a summary pair \( (P, P') \) from \( \mathcal{C}_P \) to \( \mathcal{S}_G \) that maximally improves a function \( F'(\mathcal{S}_G) \), defined as \( F'(P, P') = (1 - \alpha) (\text{cost}(P) + \text{cost}(P')) + \alpha \text{approxDis}(P, P') \). That is, \( F' \) is rounded down from the original function \( F \), which guarantees an approximation ratio. This step is repeated \( \left\lceil \frac{1}{2}k \right\rceil \) times to obtain top-\( k \) \( d \)-summaries \( \mathcal{S}_G \). If \( k \) is odd, it selects an additional summary \( P \) that maximizes \( F(\mathcal{S}_G \cup \{P\}) \) after \( \left\lceil \frac{1}{2} \right\rceil \) rounds of selection.

Analysis. The correctness of approxDis follows from the correctness of procedure sumGen, and that procedure Reduce and Validate correctly validates all reduced \( d \)-summary candidates. Following [8], approxDis simulates an approximation for maximum dispersion that guarantees approximation ratio 2.

For complexity, approxDis takes \( O(t_{s}(G, b_p)) \) (the cost of sumGen) + \( O(t_{v}(G, k)) \) (the cost of sumDiv) time. Denote as \( N \) the total verified patterns in sumGen. (1) By Lemma 2.1 and Lemma 3.2, the verification takes \( O(t_{v}(G, b_p)) = O(N + b_p|V|) \) time. (2) The diversification sumDiv takes in total \( O(t_{s}(G, k)) + O(t_{v}(G, k)) = O(N + b_p|V|) \) time. Thus, the total cost is in \( O(t_{s}(G, b_p)) + O(t_{v}(G, k)) = O(N + b_p|V| + \frac{1}{2}N^2|V|) \) time.

The sequential algorithm is expensive when \( G \) is large: both mining and diversification are two intractable processes. We can do better with parallel diversified summarization algorithm.

4.1 Parallel Approximability

We consider a partition strategy \( \mathcal{P} \) that constructs a fragmentation \( G \) of \( G \) by distributing \( G \) to \( n \) workers, where each worker \( P_i \) (\( i \in [1, n] \)) manages its local fraction (a subgraph) of \( G \), denoted as \( G_i \). To characterize the effectiveness of parallel summarization, we introduce a class of parallel approximation algorithms.

We first review parallel scalability [5, 12]. Consider a "yardstick" sequential algorithm that, given graph \( G \), integer \( d \) and size bound \( b_p \), approximately computes the summaries, e.g., algorithm approxDis. Denote the time cost of approxDis as \( t([G], b_p, k) \). A parallel algorithm \( A_p \) is parallel scalable if its running time by \( n \) processors can be expressed as

\[
T([G], b_p, k, n) = O\left(\frac{t([G], b_p, k)}{n}\right)
\]

Intuitively, it measures the speedup over a sequential algorithm by parallelization. It is a relative measure w.r.t. a yardstick sequential algorithm \( A \). A parallel scalable \( A_p \) "linearly" reduces the sequential running time of \( A \) when \( n \) increases.

Algorithm approxDis

Input: a fragmented graph \( G \), integer \( k \), size bound \( b_p \).
Output: a set of diversified reduced \( d \)-summaries.

1. /* executed at coordinator */
   set \( \mathcal{C}_P = \emptyset \); set \( \mathcal{S}_G = \emptyset \); lattice \( \mathcal{P} = \emptyset \);
   integer \( i = 1 \); flag \( \text{newP} = \text{true} \);
   \(
   \mathcal{P} = \text{Spawn}(0) ; /* initialize \mathcal{P} \text{ with single-node patterns }*/
   \)
   while \( i \leq b_p \) and \( \text{newP} = \text{true} \) do /* superstep \( i */
   2. set \( \Sigma_i = \text{Spawn}(i) ; /* returns a solution } \tilde{x} \text{ for which } \tilde{x} \leq \epsilon \times x */
   3. if \( \text{newP} \) then
   4. set \( \mathcal{C}_P_{i} = \text{ParsumGen}(\Sigma_i) ; /* parallel validation */
   5. update \( \mathcal{P} ; \mathcal{C}_P = \mathcal{C}_P \cup \mathcal{C}_P_{i} ;
   6. construct work units \( M_i \) and distribute \( M_i \) to workers;
   7. set \( \mathcal{S}_G_{i} = \text{ParsumDiv}(M_i) ; /* parallel diversification */
   8. update \( \mathcal{S}_G \) with \( \mathcal{S}_G_{i} ;
   9. return \( \mathcal{S}_G ;

Figure 3: Algorithm approxDis

Parallel approximability. Consider an optimization (e.g., maximization) problem with an optimal solution quantified by a single numerical \( x \). We say the problem is parallel \( \epsilon \)-approximable, if there exists a parallel scalable algorithm \( A_p \) w.r.t. a sequential yardstick algorithm \( A \) and returns a solution \( \tilde{x} \) for which \( \tilde{x} \leq \epsilon \times x \). We present the main result of this section below.

Theorem 4.1. There exists a parallel \( \epsilon \)-approximable algorithm w.r.t. the sequential algorithm approxDis that discovers diversified top-\( k \) summaries with time cost in \( O\left(\frac{T([G], b_p, k)}{n}\right)\).

As a proof, we develop a parallel algorithm, denoted as approxDis (illustrated in Fig. 3). It follows bulk synchronous model and runs in supersteps, where each superstep contains two steps: (1) Parallel verification (denoted as ParsumGen) that "paralyzizes" its sequential counterpart sumGen to generate and verify summaries (as in Algorithm approxDis), and (2) Parallel diversification (denoted as ParsumDiv) that "paralyzizes" its sequential counterpart sumDiv (as in Algorithm approxDis) to update \( \mathcal{S}_G \).

Below we introduce the algorithm approxDis.

4.2 Parallel Diversified Summarization

Overview. Given a fragmented graph \( G \), the algorithm approxDis, shown in Figure 3, executes at most \( b_p \) supersteps. It first invokes an operator \( \text{Spawn} \) to initialize \( \mathcal{P} \) with single node patterns (line 2). At each superstep \( i \), it performs the following.

(1) It invokes \( \text{Spawn}(i) \) to generate a set \( \Sigma_i \) of patterns of size \( i \) at a coordinator \( \Sigma_c \) (line 4). It then invokes \( \text{ParsumGen} \) to verify the patterns at the workers, in parallel (line 6), and updates \( \mathcal{P} \) with new reduced summaries if any (line 7).

(2) It then constructs work units \( M \) as pairs of summaries, and distributes \( M \) to all the workers following a load balancing strategy (to be discussed, line 8). It invokes \( \text{ParsumDiv} \) to collect the top-\( k \) diversified summary pairs \( \mathcal{S}_G \), computed locally at each worker in parallel (line 9), and update \( \mathcal{S}_G \) with the summaries that improve the rounded function \( F'(\mathcal{S}_G \cup \mathcal{P}) \) in approxDis.

The above steps repeat until \( b_p \) supersteps are executed, or no new pattern can be generated, indicated by a flag \( \text{newP} \) (line 3).

Below we present procedures ParsumGen and ParsumDiv.
Parallel verification. Upon receiving $\Sigma_i$ from Spawn($i$), procedure ParSumGen validates $\Sigma_i$ in parallel as follows.

1. At $S_c$, for each pattern $P \in \mathcal{P}_t$, ParSumGen identifies a verified summary $P_i$ in $\mathcal{P}$ such that $P$ is obtained by adding an edge $e$ to $P_i$. It then constructs a work unit $(P_i, e, j)$, which encodes a request that “computes the distances between the (LDG) scheme [22], which assigns a vertex to the partition where it is.”

2. Upon receiving a set of work units, for each work unit $(P_i, e, j)$, each worker $S_j$ performs incremental validation for $P$ as the “union” of $P_i$ and $e$, which (a) issues an on-demand fetching of $e(G_k)$, the local edge matches of $e$ from other workers $S_k$, and (b) verifies $P$ in the graph $P_i(G_j) \cup \bigcup_{k \in [1,n]} e(G_k)$ (i.e., the “union” of local matches $P_i(G_j)$ and all the edge candidates of $e$) of the entire graph $G_j$.

3. Each worker stores $P(G_j)$ for the next round of computation.

Parallel Diversification. Given the verified $d$-summaries $C_d$ so far (including $C_{d+1}$ (line 7), ParSumDiv updates diversified summaries $S_G$ in parallel, as follows.

1. At $S_c$, for each summary $P \in C_{d+1}$, paraDis constructs a work unit $wp$. The work unit $wp$ consists of (a) $P$ and the vector $P.vec_j$, and (b) a set of summaries $D_P \subseteq C_{d+1}$, as well as their bit vectors, which encodes a request that “computes the distances between $P$ and the summaries in $D_P$”. Given the work units $WP = \bigcup_{P \in C_{d+1}} wp$, it distributes $WP$ to all the workers, following a work load balancing strategy (to be discussed, line 8).

2. Upon receiving a set of work units $M_j$, for each work unit $wp \in M_j$, a worker $S_j$ computes the distances $\text{diff}(P, P')$ ($P' \in D_P$) by bit vector operations on $P.vec_j$ and $P'.vec_j$. It locally executes a top-$k$ query to find out the local top-$k$ diversified pairs $S_{G_j}$ that maximize the diversification function $F^*$, and returns $S_{G_j}$ to $S_c$.

3. The coordinator $S_c$ collects local top-$k$ pairs from all the workers, and updates $S_{G_d} = \bigcup_{j \in [1,n]} (S_{G_j})$ (line 10). It then updates $S_{G_d}$ with the new summary pairs in $S_{G_d}$.

Load balancing. We partition $G$ with linear deterministic greedy (LDG) scheme [22], which assigns a vertex to the partition where it has the most edges. This reduces skewed distribution due to high-degree edges. Algorithm paraDis further uses strategies below.

1. For each work unit $(P_i, e, j)$ (ParSumGen), $e(G)$ is evenly distributed to all workers. paraDis estimates a “runtime skewness” of $P_i(G)$ at $S_j$ as $1 - \frac{\|P_i(G)|}{|P_i(G)|}$. If the skewness is above a threshold, it evenly redistributes $P_i(G)$ to all workers.

2. For each work unit $wp$ with a set of summaries $D_P$, paraDis estimates an upper bound of the diversification cost as $|D_P||V|^2$, and assigns the cost to $wp$. It then adopts a greedy strategy following the generalized assignment problem [20] which iteratively assigns a work unit with the smallest cost to a worker with the (dynamically updated) least load. By developing an approximation preserving reduction, we can verify that this algorithm is a 2-approximation [20], and takes $O(|V||n \log n)$ time ($|V| \leq |C_P|$) for at most $|C_P|$ validated summaries at superstep $i$.

Performance Analysis. To see that paraDis is parallel scalable, it suffices to show that parallel verification and diversification are parallel scalable w.r.t. their sequential counterparts in approxDis.

Parallel scalability. Recall that approxDis incurs two parts of cost $O(t_1(G, b_p)) = O(N \cdot b_p |V||E|)$, and $O(t_2(G, k)) = O(k^2 N^2 |V|)$. In accordance, paraDis incurs the cost below.

1. At superstep $i$, each worker $S_j$ (a) receives $e(G_k)$ ($k \neq j$) in $O(\frac{|E|}{n})$ time. due to the balanced edge partition of $G$, (b) sends $e(G_j)$ to other $n-1$ workers in $O(\frac{n-1|E|}{n})$ time, which is bounded by $O(\frac{|V||E|}{n})$ time as $n \ll |V|$, (c) performs local verification in parallel, in $O(\frac{|E|}{n})$ time, and (d) returns the local matches in parallel in $O(\frac{|V||E|}{n})$ time. As there are $b_p$ supersteps, the total cost is in $O(\frac{N \cdot b_p |V||E|}{n}) = O(\frac{t_1(G, b_p)}{n})$.

2. At superstep $i$, ParSumDiv takes in total $O(\frac{|V|}{n} |M_j||D_P|^2 |V|)$ time to computes top-$k$ summary pairs, bounded by $O(\frac{k^2 |C_P|^2 |V|}{n})$, as all the summaries are from $C_P$. The parallel cost of sending top summaries to $S_c$ is in $O(\frac{b_p \cdot k}{n})$. The total parallel diversification cost is thus in $O(\frac{k^2 N^2 |V|}{n} + O(\frac{t_1(G, b_p)}{n}))$.

Putting these together, algorithm paraDis takes in total $O(\frac{k^2 N^2 |V|}{n} + O(\frac{t_1(G, b_p)}{n}))$ time, hence is parallel scalable w.r.t. its sequential counterpart approxDis.

Approximation. The quality guarantee of paraDis follows from the invariant that at any superstep $i$, the set $S_{G_d}$ is a 2-approximation of the optimal diversified summaries from the validated ones. Indeed, it suffices to identify the top-$k$ summaries from ParSumDiv. Thus, paraDis generates $S_{G_d}$ with approximation ratio 2 when terminates, and is a parallel 2-approximation algorithm.

The above analysis completes the proof of Theorem 4.1.

Remark: paraDis can be readily extended to parallelize “anytime” summarization (streamDis) [21]. To this end, it only needs to maintain a cache at the coordinator to store (a) a set of summaries, and (b) for each summary, top-$k$ most different summaries (that maximize function $\text{diff}(\cdot)$). It performs parallel verification and diversification to maintain the cached summaries at coordinator, in parallel, and returns the summarization whenever requested.

5 EXPERIMENTAL EVALUATION

Using real-life and synthetic data, we evaluated the efficiency of sequential algorithm approxDis, the scalability of parallel algorithm paraDis and the effectiveness of the summaries.

Experimental Setting. We used the following setting.
Figure 4: Sequential summarization: Performance

Datasets. We use three real-life graphs: (1) DBpedia\(^1\) contains 4.86M nodes and 15M edges and 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 with more diversified (324343) labels; and (3) Freebase (version 14-04-14)\(^3\), with 40.32M nodes, 63.2M edges, and 9630 labels.

We also use BSBM\(^4\) e-commerce benchmark to generate synthetic knowledge graphs over products with different types, related vendors, consumers, and views. The generator is controlled by the number of nodes (up to 60M), edges (up to 152M), and labels drawn from an alphabet of 3080 labels.

Algorithms. We implemented the algorithms below, all in Java:

(1) Sequential algorithm aproxDis, compared with (a) GRAMI [3] that discovers frequent subgraphs as summaries, where the base graph of a summary is induced by all its isomorphic counterparts in G; (b) stream-style algorithms heuDis and streamDis [21] extended to reduced summaries, where streamDis applies procedure sumDiv to cached (at most \(l_p\)) summaries of a stream of verified ones, and heuDis follows streamDis, but simply swaps a summary with a new one that maximize function \(F\);

(2) Parallel algorithm paraDis (including procedures ParsumGen and ParsumDiv), compared with paraDisn, its counterpart without load balancing strategy; and

(3) A query evaluation algorithm extended from [21], which accesses reduced d-summaries instead of d-summaries. It selects and refers to a small set of reduced summaries that best "cover" the query, and fetches entities from the graph only when necessary.

As reduced summaries are a special case of d-summaries, all the techniques and complexity results remain intact. We remark that one can readily plug-in any standard query evaluation algorithms for subgraph queries to the query evaluation framework.

We ran all our experiments on a Linux machine powered by an Intel 2.4 GHz CPU with 128 GB of memory. For the tests of parallel summarization, we used Amazon EC2 r4.large instances, each powered by an Intel 2.8GHz CPU and 16G of memory. We ran each experiment 5 times and report the averaged results.

We next report the details of our findings.

Exp-1: Sequential summarization. We fixed parameter \(\alpha=0.5\) in function \(F\) for diversification, \(k=64\), summary size bound \(b_p=6\), and \(d=1\). We set a support threshold \(\theta=0.005\). For GRAMI,

\[\text{Time(seconds)}\]

(a) Real-world datasets

(b) streamDis: quality vs. time(s)

Exp-2: Parallel summarization. We evaluate the parallel algorithm paraDis, compared with paraDisn. We fixed parameter \(\alpha=0.5\) for diversification, \(k=64\), the summary size bound \(b_p=6\), number of hops \(d=1\), and varied the number of workers \(n\) from 4 to 20.

We report the performance of paraDis over the real-world datasets in Fig. 5(a)- Fig. 5(c), respectively. (1) paraDis scales well with larger \(n\). The performance of paraDis is improved by 2.6 times from when \(n\) is increased from 1 to 4 and 3.3 times for \(n\) from 4 to 20. (2) The load balancing strategy improves the performance of paraDis by 4.4 times on average. (3) It is feasible to summarize large graphs. For example, paraDis takes 200 seconds for paraDis to summarize DBpedia, when \(n=20\), improving its sequential counterpart aproxDis by 13 times. (4) In all cases, the accuracy of paraDis (not shown) is almost the same as its sequential counterpart approxDis, due to its parallel approximability.

Using the same setting, we evaluated paraDis with a set of synthetic graphs. The scalability result varying \(n\) is consistent with its counterparts over real-world graphs (not shown): paraDis is 3 times faster when \(n\) is increased from 4 to 20 for \(|G|=60M, 152M|\) (60 million nodes, 152 million edges). Furthermore, Fig. 5(d) verifies that paraDis scales well with graph size \((n=20)\): it takes 371 seconds for \(|G|=10M, 22M|\) and 2.5k seconds for \(|G|=60M, 152M|\).

Exp-3: Case study. We performed two case studies to evaluate the practical application of the knowledge summaries.

Coverage of summaries. We investigate the practical number of diversified summaries needed to "cover" the entity types. We

http://dbpedia.org

http://www.mpi-inf.mpg.de/yago

http://freebase-easy.cs.uni-freiburg.de/dump/

http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/
IT Companies winning e.g., and YAGO e.g., bases [2]. We generated 20 cross-domain queries over the summaries from GRAMI, respectively, and “merges” the matches from each if they have the same URI, to form a complete answer.

We show a query and its answer in Fig. 7. The query finds Award winning IT_Companies with specified products and their parent companies (IC). While YAGO reports parent companies, and DBpedia provides products, a complete answer (Amazon.com) can be found by accessing summaries P_1 and P_3 from YAGO and DBpedia, respectively, integrating the partial answers. This suggests the application of summaries in supporting multi-source querying.

6 CONCLUSIONS
We studied reduced d-summaries and developed sequential and parallel summarization algorithms for large knowledge graphs. Our experimental results verified that our algorithms are feasible, and can significantly reduce the cost of knowledge graph query evaluation [7]. Another topic is to develop summary-based algorithms for more types of analytical queries.

REFERENCES
[22] Isabelle Stanton and Gabriel Kliot. 2012. Streaming graph partitioning for large distributed graphs. In KDD.
[27] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. 2006. Extracting redundancy-aware top-k patterns. In SIGKDD.