Mining Summaries for Knowledge Graph Search

Qi Song, Yinghui Wu, Peng Lin, Luna Xin Dong, and Hui Sun

Abstract—Querying heterogeneous and large-scale knowledge graphs is expensive. This paper studies a graph summarization framework to facilitate knowledge graph search. (1) We introduce a class of *reduced summaries*. Characterized by approximate graph pattern matching, these summaries are capable of summarizing entities in terms of their neighborhood similarity up to a certain hop, using small and informative graph patterns. (2) We study a *diversified graph summarization* problem. Given a knowledge graph, it is to discover top-k summaries that maximize a bi-criteria function, characterized by both informativeness and diversity. We show that diversified summarization is feasible for large graphs, by developing both sequential and parallel summarization algorithms. (a) We show that there exists a 2-approximation algorithm to discover diversified summaries. We further develop an anytime sequential algorithm which discovers summaries under resource constraints. (b) We present a new parallel algorithm with quality guarantees. The algorithm is parallel scalable, which ensures its feasibility in distributed graphs. (3) We also develop a summary-based query evaluation scheme, which only refers to a small number of summaries. Using real-world knowledge graphs, we experimentally verify the effectiveness and efficiency of our summarization algorithms, and query processing using summaries.

Index Terms—Graph summarization, pattern mining, parallel algorithm

1 Introduction

Knowledge graphs are routinely used to represent entities and their relationships in knowledge bases [1], [2]. 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 [2], [3], [4], [5]. Nevertheless, it is hard for end-users to precise queries that will lead to meaningful answers without any prior knowledge of the underlying data graph. Querying such knowledge graphs is challenging due to the ambiguity in queries, the inherent computational complexity (e.g., subgraph isomorphism [2], [3]) and resource constraints (e.g., data allowed to be accessed, response time) [6] large knowledge graphs.

Example 1. Fig. 1 illustrates a sample knowledge graph G of artists and bands. In this example, T. McGraw is the correct answer for Q.

The evaluation of Q over large G is expensive. For example, the ambiguous label "artist" requires the inspection of all the entities having the type. Moreover, it is hard for the users to specify Q without prior knowledge of G.

Observe that the graph G can be "summarized" by three small graph patterns P_1, P_2 and P_3, as illustrated in Fig. 1. For example, P_1 specifies three artists J. Browne, T. McGraw and D. Yoakam in G as a single node artist, who are associated with their band, genre and films as 1 hop neighbors, indicating "musicians"; and (i.e., "actors"). These concise summaries help the users in understanding G without a daunting inspection of low-level entities.

We may further use these patterns as "views" [7], [8] to speed up knowledge discovery in G. For example, P_1 and P_2 can be "materialized" by the entities they summarize in G, which already contains the matches of Q. Q can then be correctly answered by accessing these entities only, without visiting an excessive number of entities in G.

The above example suggests that graph patterns can benefit knowledge search by suggesting (and can be directly queried as) highly interpretable "views". In addition, such summaries can help users in understanding complex knowledge graphs without inspecting a large amount of data, explaining facts with interpretable evidences, and suggesting meaningful queries in mining tasks.

Although desirable, computing summaries for schemaless knowledge graph is nontrivial. Conventional graph summaries defined by frequent subgraphs capture their isomorphic counterparts in a graph [4], [9], [10], [11]. This can often be an overkill for entities with similar, relevant neighbors up to a certain hop. For example, the two entities J. Browne and T. McGraw along with their relevant 1 hop neighbors in Fig. 1 *should* be summarized by a single summary P_1, despite that the two subgraphs induced by these entities are not isomorphic to each other; similarly for the entities T. Hanks and M. Ryan summarized by P_3. We ask the following questions: 1) *How to model concise and informative summaries in schema-less knowledge graphs?* 2) *How to discover the summaries in large graphs?* and moreover, 3) *How can we leverage the summaries to support fast knowledge graph search?*

Contributions. This paper studies a novel graph summarization framework to compute diversified summaries,

- *Q. Song and P. Lin are with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99163. E-mail: qsong@eecs.wsu.eduk, plin1@eecs.wsu.edu.*
- *Y. Wu is with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99163, and the Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, WA 99354. E-mail: yinghui@eecs.wsu.edu.*
- *L.X. Dong is with Amazon. Inc., Seattle, WA 98101. E-mail: lunadong@amazon.com.*
- *H. Sun is with Renmin University, Beijing 100872, China. E-mail: sun_h@ruc.edu.cn.*

Manuscript received 3 May 2017; revised 24 Jan. 2018; accepted 13 Feb. 2018. Date of publication 22 Feb. 2018; date of current version 10 Sept. 2018. (Corresponding author: Qi Song.)
Recommended for acceptance by L. Chen.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2018.2807442

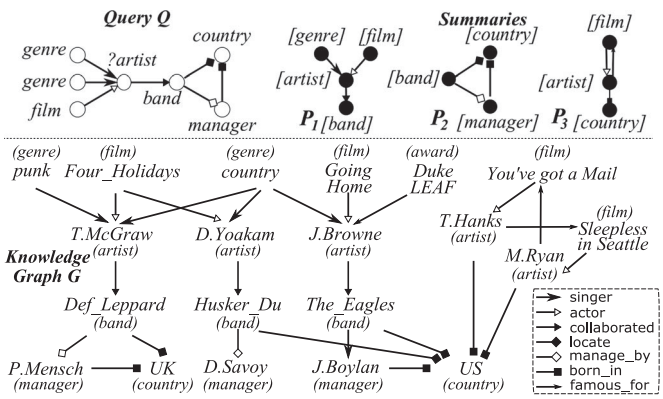


Fig. 1. Knowledge graph, summaries, and graph query.

and to evaluate knowledge graph queries with the summaries. It nontrivially extends [12] by including new summary models, complete proofs, new parallel algorithms with performannce guarantees on summarizatin qualty and scalability, and enriched experimental study for the new models and summarization in large, distributed graphs.

(1) *Extended summary models*. We introduce a new class of graph patterns, namely, *reduced d-summaries*, to summarize entities in terms of their neighborhood similarity up to a bounded hop d with minimized summary size. The new summary model refines its counterpart in [12] by capturing and removing the redundancy in summaries. We show that reduced d-summary is feasible in practice, by studying the verification and reduction problems, which checks if a graph pattern is a reduced d-summary, and reduces a summary to its reduced counterpart, respectively.

(2) *Diversified Summarization*. We extend bi-criteria functions in [12] to quantify the quality of reduced d-summaries that integrates both informativeness and diversity measures (Section 3). Based on the quality function, we introduce the problem of *diversified graph summarization*.

(3) *Parallel summarization*. We show that the diversified summarization problem remains to beNP-hard for reduced summaries. We show that diversified summarization is feasible with a single processor (Section 4), and with multiple processors (Section 5).

  (a) We first develop sequential algorithms for reduced summaries. We show that the summarization problem is 2-approximable, by developing a sequential algorithm that follows a validate-and-diversify scheme, and invokes a fast summary reduction procedure. We also extend the *anytime* mining algorithm in [12] to reduced summaries, which can be interrupted and provides "ad-hoc" summaries with desirable quality guarantees, adapting to specific resource bounds (e.g., memory, response time) (Section 4).

  (b) We develop new parallel algorithm for diversified summarization over large graphs (Section 5). 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. We have developed new parallel matching and mining operators, and load balancing strategies for reduced summary discovery. These are not addressed in [12].

(4) We further develop a query evaluation algorithm over knowledge graphs for the class of subgraph queries. The algorithm selects and refers to a small set of summaries that best "cover" the query, and fetches entities from the original knowledge graph only when necessary (Section 6).

(5) Using real-world knowledge bases and synthetic graphs, we experimentally verify the effectiveness of reduced d-summaries, and the scalability of summarization and query-evaluation algorithms (Section 7). We found the following. (a) It is feasible to compute summarizations over real-world knowledge graphs. For example, it takes 300 seconds over a knowledge graph YAGO with 3.9 million nodes and relationships, and it achieves a scalability of 3.3 times faster when the number of workers increases from 4 to 20. (b) The summaries effectively support concise, informative and diversified summarization. (c) The summarization significantly improves querying efficiency (e.g., by 40 times for YAGO). Our case studies verifies the application of reduced summaries in knowledge search, query suggestion and fact checking in knowledge graphs.

The work is the first step towards discovering and using diversified summarization to understand and search large-scale knowledge graphs. We believe that our framework suggests promising tools for accessing, searching, and understanding complex knowledge graphs.

2 Knowledge Graph Summarization

2.1 Graphs and Summaries

We start with the notions of knowledge graphs, and then introduce summaries for knowledge graphs.

Knowledge Graphs. We define a knowledge graph G as a directed labeled graph (V, E, L), where V is a set of nodes, and $E \subseteq V \times V$ 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.

We do not assume a standard schema over G, and our techniques will benefit from such a schema, if exists.

Example 2. Fig. 1 depicts a fraction of a typed knowledge graph. Each entity (e.g., J. Browne) 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).

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'.

Summaries. Given a knowledge graph G, a 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. $L_P(e)$), representing a non-empty node set $[u]$ (resp. edge set $[e]$) from G.

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 $e \in E_P$. Note that a base graph can be disconnected for a connected summary.

A summary should provide an abstract of the entities with similar neighborhoods in G. To capture this, we introduce a notion of d-matching.

d-matching. Given a graph pattern P and a graph G, a *backward* (resp. *forward*) *d-matching*from P to G is a *nonempty* binary relation $R_d^{\uparrow} \subseteq V_P \times V$ (resp. $R_d^{\downarrow} \subseteq V_P \times V$), where

- $(u, v) \in R_0^{\uparrow}$ and $(u, v) \in R_0^{\downarrow}$ if $L_P(u) = L(v)$;
- $(u, v) \in R_d^{\uparrow}$ if $(u, v) \in R_{d-1}^{\uparrow}$, 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 $(u', v') \in R_{d-1}^{\uparrow}$;
- $(u, v) \in R_d^{\downarrow}$ if $(u, v) \in R_{d-1}^{\downarrow}$, and for every child u' of u in P, there exists a child v' of v in G such that $L_P(u, u') = L(v, v')$, and $(u', v') \in R_{d-1}^{\downarrow}$.

We define a d-match R_d between P and G as the set of node pairs $\{(u, v) | (u, v) \in R_d^{\uparrow} \cap R_d^{\downarrow}\}$. We say P is a *d-summary* of G (denoted as $P \sim G$), if for every summary node u and every node $v \in [u]([u] \neq \emptyset)$, $(u, v) \in R_d$.

Intuitively, a d-summary P guarantees that for any incoming (resp. outgoing) path ρ of a summary node u with a bounded length d in P, there must exist an incoming (resp. outgoing) path of each node in $[u]$ with the same label. That is, P preserves all the neighborhood information up to length d for each summary node u in P.

We now characterize graph summarization with summaries. Given a knowledge graph G and an integer d, a *summarization* $\mathcal{S}_G$ of G is a set of d-summaries.

The matching relation can also incorporate transformation functions [5] to allow node similarity as used in knowledge graph embedding.

Example 3. 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 is induced by the entities shown in the table below (the edges are omitted).

summary node	entities
[genre]	{ country, punk }
[film]	{Going Home, Four_Holidays}
[artist]	{J. Browne, D. Yoakam, T. McGraw}
[band]	{The_Eagles, Husker_Du, Def_Leppard }

Indeed, for every path of length bounded by 2 in P_1 (e.g., ρ_p = {genre, artist, band}) and for every entity with label genre, there exists a path ρ (e.g., {country, J. Browne, The_Eagles}) in G with the same label as ρ_p. Similarly, one may verify that P_2 summaries the band Def_Leppard and The_Eagles, their associated country and manager in G, and P_3 summaries the films You've got a Mail and Sleepless in Seattle, actors T. Hanks and M. Ryan and their countries.

Note that P_1 cannot summarize T. Hanks, as the latter has no path to a band as suggested in P_1.

2.2 Summary Verification

Given a graph pattern P, a knowledge graph G and 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*.

TABLE 1
Notations

Symbol	Definition
$G = (V, E, L)$	knowledge graph G
$P = (V_P, E_P, L_P)$	summary P as a graph pattern
G_P	base graph of summary P in G
$\mathcal{S}_G$	summarization of G
$\mathsf{card}(\mathcal{S}_G)$	cardinality of $\mathcal{S}_G$
$\mathsf{supp}(P, G)$	support of summary P in G
$I(P)$	informativeness of summary P; $I(P) = \lvert P\rvert * \mathsf{supp}(P, G)$
$\mathsf{diff}(P_i, P_j)$	distance between summaries P_i and P_j
$F(\mathcal{S}_G)$	bi-criteria quality function of $\mathcal{S}_G$

Lemma 1. *Given a summary $P = (V_P, E_P, L_P)$, 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.*

Proof. As a proof of Lemma 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 $[u] = \{v | (u, v) \in R_0^{\uparrow}, v \in V\}$ for each node $u \in V_P$. It then refines the match sets as follows. (1) It computes the forward d-similarity relation $R_d^{\downarrow}$. 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 $(u, v) \in R_{i-1}^{\downarrow}$, 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. If for every node $u \in V_P$, $[u] \neq \emptyset$, P is a d-summary. Otherwise, P is not a d-summary by definition.

The algorithm has the following invariants: (1) For any pair $(u, v) \in R_d^{\uparrow} \cap R_d^{\downarrow}$, $v \in [u]$ 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. For complexity, it takes $O((|V_P| + |V|)(|E_P| + |E|))$ time to verify forward and backward d-similarity for a single summary node in V_P. Thus the total cost is in $O(|V_P|(|V_P| + |V|)(|E_P| + |E|))$. □

The notations of this paper are summarized in Table 1.

3 Diversified Summarization

We next introduce a bi-criteria function that captures the quality of knowledge graph summarization in terms of both informativeness and diversity, followed by the formulation of diversified summarization problem.

3.1 Informative Summaries

The informativeness of a summary P should capture the total amount of information (entities and their relationships) it encodes in a knowledge graph G [13]. We define the informativeness function $I(\cdot)$ of a summary P as:

$$I(P) = \frac{|P|}{b_P} * \mathsf{supp}(P, G)$$

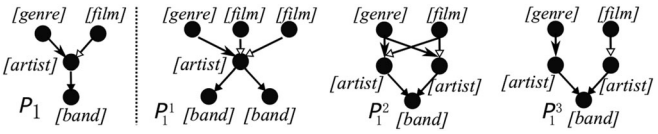


Fig. 2. Reduced summary and its "nonreduced" counterparts.

where (1) $|P|$ refers to the size of P, defined as the total number of nodes and edges in P, (2) 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) [14], and (3) the support $\mathsf{supp}(P, G)$ is defined as $\frac{|G_P|}{|G|}$, where $|G_P|$ (resp. $|G|$) refers to the size (i.e., the total number of nodes and edges) in G_P (resp. G).

Intuitively, the informativeness function $I(\cdot)$ favors larger summaries that also have higher support. Support defined by the frequency of subgraphs [10] and minimum description length [11] lead to frequent but less informative patterns, as observed in [13].

Example 4. Consider the 2-summaries P_1-P_3 of the graph G (with 45 entities and edges) in Fig. 1. Let the summary size bound $b_p = 8$, we can verify that the size of the base graph $|G_{P_1}|$ is 20. Hence, $\mathsf{supp}(P_1, G) = \frac{20}{45}$, and the informativeness of $P_1 I(P_1)$ is $\frac{7}{8} * \frac{20}{45} = 0.39$. Similarly, $I(P_2) = \frac{6}{8} * \frac{12}{45} = 0.20$, and $I(P_3) = \frac{6}{8} * \frac{11}{45} = 0.18$.

3.2 Reduced Summaries

A second challenge is to avoid redundancy among the summaries, and to characterize diversified summaries. We introduce a distance function to quantify the difference between two summaries.

Distance function. To cope with the summary redundancy due to commonly summarized entities, we define a distance function diff for two summaries P_1 and P_2 as

$$\mathsf{diff}(P_1, P_2) = 1 - \frac{|V_{G_{P_1}} \cap V_{G_{P_2}}|}{|V_{G_{P_1}} \cup V_{G_{P_2}}|}$$

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 diff is a metric, i.e., for any three d-summaries P_1, P_2 and P_3, $\mathsf{diff}(P_1, P_2) \leq \mathsf{diff}(P_1, P_3) + \mathsf{diff}(P_2, P_3)$. 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} in the distance function diff.

Example 5. Consider the 2-summaries P_1-P_3 of the graph G in Fig. 1. The differences are calculated as follows: $\mathsf{diff}(P_1, P_2) = 1 - \frac{2}{14} = 0.86$, where they summarize two common entities {The_Eagles, Def_Leppard}). Similarly, $\mathsf{diff}(P_1, P_3) = 1.00$, and $\mathsf{diff}(P_2, P_3) = 0.90$.

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.

Example 6. Consider three summaries: P_1 (Example 1), P_1^1, and P_1^2 as illustrated in Fig. 2. We can verify that these three summaries have a same base graph in G (Fig. 1). Although P_1^1 and P_1^2 are larger than P_1, they contain "redundant" nodes (e.g., film and band in P_1^1, and artist in P_1^2) 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 $P_1 \preceq P_2$), and its inverse relation R_{12}^{-1} is a d-matching from P_2 to P_1 (denoted as $P_1 \preceq P_2$). The result below bridges summary equivalence and their support.

Lemma 2. *For any graph G and its two d-summaries P_1 and P_2, $\mathsf{supp}(P_1, G) = \mathsf{supp}(P_2, G)$ if $P_1 \sim P_2$.*

Proof. Given two equivalent summaries $P_1 = (V_{P_1}, E_{P_1}, L_{P_1})$ and $P_2 = (V_{P_2}, E_{P_2}, L_{P_2})$, it suffice to show that $|G_{P_1}| = |G_{P_2}|$, where $G_{P_1} = (V_1, E_1, L)$ (resp. $G_{P_2} = (V_2, E_2, L)$) 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 $P_1 \preceq P_2$, there exists a nonempty d-matching R_{12} from P_1 to P_2. For every node $u_1 \in V_{P_1}$, there exists a node $u_2 \in V_{P_2}$ such that $(u_1, u_2) \in R_{12}$. As P_2 is a d-summary of G, for each node u_2, there is a node $v \in V_2$ such that $(u_2, v) \in R_2$. Following the definition of d-matching, we can verify that $(u_1, v) \in R_1$. That is, any match of u_2 in G is also a match of u_1. Thus, $V_2 \subseteq V_1$. Similarly, as $P_2 \preceq P_1$, we can verify that $V_1 \subseteq V_2$. That is, $V_1 = V_2$.

(2) Following the above proof, one can verify that $E_1 = E_2$. Indeed, (a) $E_1 \preceq E_2$, as E_1 and E_2 are edge matches induced by the d-matching of R_1 and R_2, respectively, and $P_1 \preceq P_2$, and (b) $E_2 \preceq E_1$. Thus, $E_1 = E_2$.

Putting these together, $|G_{P_1}| = |G_{P_2}|$. Thus, $\mathsf{supp}(P_2, G) = \mathsf{supp}(P_1, G)$ by the definition of support. □

A summary P is a *reduced summary*, if there exists no smaller summary P' obtained by removing edges from P, such that $P \sim P'$. A reduced summary is a minimal representation of its equivalent summaries. Reduced summaries are not discussed in [12]. Better still, a summary P can be efficiently "reduced", as verified by the result below.

Lemma 3. *Given a summary $P = (V_P, E_P, L_P)$, there exists an algorithm that computes a reduced summary P_r of P in $O((|V_P| + |E_P|)^2 + |V_P|^2)$ time.*

Proof. As a proof of Lemma 3, 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 new pattern P_r.

It is easy to verify that $P_r \preceq P$ and $P \preceq P_r$. 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 $P_r' \sim P$. 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 $(u, v) \in R^{*'}$, $(v, u) \in R^{*'}$, $(u', v) \in R^{*'}$, and $(v, u') \in R^{*'}$. Thus, u, u' and v in P belongs to the same equivalent class. Nevertheless, u and u' are not merged in P_r. Thus, either P_r' is not equivalent to P, or $|P_r| = |P_r'|$. Either leads to a contradiction.

The procedure Reduce takes $O(|V_P| + |E_P|)^2)$ time to compute the equivalence relation, and $O(|V_P|^2)$ to

perform the reduction. The total cost is thus in $O((|V_P| + |E_P|)^2 + |V_P|^2)$ time. Lemma 3 follows. □

Example 7. Following Example 6, the reduction process finds that the two film nodes and two band nodes belong to the same equivalent class, hence reduces P_1^1 to P_1. Similarly, P_1^2 can be reduced to P_1 by merging all artist nodes. Note that P_1^3 is a reduced summary and is not equivalent to P_1, as the artist in P_1 can not be matched with those in P_1^3.

3.3 Diversified Summarization

Good summaries should cover diverse concepts with informative summaries. We introduce a bi-criteria function F that integrates informativeness $I(\cdot)$ and distance $\mathsf{diff}(\cdot)$ functions. Given a summarization $\mathcal{S}_G$ for a knowledge graph G, the function F is defined as:

$$F(\mathcal{S}_G) = (1-\alpha)\sum_{P_i \in \mathcal{S}_G} I(P_i) + \frac{\alpha}{\mathsf{card}(\mathcal{S}_G)-1}\sum_{P_i \neq P_j \in \mathcal{S}_G} \mathsf{diff}(P_i, P_j)$$

where (1) $\mathsf{card}(\mathcal{S}_G)$ refers to the number of summaries it contains; and (2) $\alpha (\in [0,1])$ is a tunable parameter to tradeoff informativeness and diversification. Note that we scale down the second summation (diversification) which has $\frac{\mathsf{card}(\mathcal{S}_G)(\mathsf{card}(\mathcal{S}_G)-1)}{2}$ terms, to balance out the fact that the first summation (informativeness) has $\mathsf{card}(\mathcal{S}_G)$ terms.

Example 8. Set $b_p = 8$ and $\alpha = 0.1$, a top-2 diversified summarization $\mathcal{S}_G$ of G (Fig. 1) is $\{P_1, P_2\}$, with total quality score $F(\mathcal{S}_G) = 0.9 * (0.39 + 0.20) + 0.1 * 0.86 = 0.62$.

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

Diversified Graph Summarization. Given a knowledge graph G, integers k and d, and a size budget b_p, the *diversified graph summarization* problem is to compute a summarization $\mathcal{S}_G$ of G as a top k summary set, where

- each summary in $\mathcal{S}_G$ is a reduced d-summary with size bounded by b_p; and
- the quality function $F(\mathcal{S}_G)$ is maximized.

That is, the diversified graph summarization is to identify k reduced summaries that are both informative and diversified. Although desirable, the problem is (not surprisingly) NP-hard. The lower bound of the hardness can be shown by constructing a reduction from the maximum dispersion problem [15], which is known to be NP-complete.

Despite the hardness, we show that diversified summarization is feasible over large knowledge graphs, by providing both sequential and parallel diversified summarization algorithms in Section 4 and Section 5, respectively.

4 Computing Diversified Summaries

Finding the optimal set of summaries by enumerating and verifying all k-subsets of summaries is clearly not practical for large G. We develop feasible algorithms to compute diversified summaries. (1) We show that the problem is 2-approximable, by providing an approximate mining algorithm in Section 4.1. (2) We further develop a fast "anytime" algorithm that response to ad-hoc accessing to the summaries in Section 4.2.

4.1 Approximated Summarization

Let $\mathcal{S}_G^*$ denote the optimal summarization that maximizes the diversification function F. For any given graph G, an ϵ-approximation mining algorithm returns a summarization $\mathcal{S}_G$, such that $F(\mathcal{S}_G) \geq \frac{F(\mathcal{S}_G^*)}{\epsilon}$ ($\epsilon \geq 1$). We show that diversified knowledge summarization is 2-approximable, by presenting such an approximation algorithm.

Overview. 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 $\mathcal{C}_P$ of reduced d-summaries with size bounded by b_p. (2) It then invokes a diversification algorithm sumDiv to compute the top-k diversified summaries $\mathcal{S}_G$ from $\mathcal{C}_P$.

We next introduce the algorithms sumGen and sumDiv.

Algorithm sumGen. The major bottleneck is the summary mining process. Clearly, enumerating and verifying all summaries is expensive over large G. Instead, the algorithm sumGen reduces redundant verification by performing a one-time verification for all the equivalent summaries.

The algorithm extends its counterparts in [12] with a lattice for reduced summaries, and a procedure to reduce pattern candidates to their reduced counterparts.

Reduced Summary Lattice. Underlying the algorithm sumGen is the maintenance of a lattice $\mathcal{P} = (V_r, E_r)$ that encodes the generation and validation of d-summaries, where (1) V_r is a set of lattice node, and each node $P_r \in V_r$ at level j of $\mathcal{P}$ is a reduced d-summary with j edges, and (2) there exists an edge $e_t = (P_r, P_r') \in E_r$, if P_r and P_r' are two reduced d-summaries at level j and $j+1$ ($j \in [1, b_p - 1]$), and P_r' is obtained by adding an edge to P_r.

The algorithm sumGen follows a level-wise generation and validation with $\mathcal{P}$ as follows.

(1) It initializes $\mathcal{P}$ with patterns of single nodes. For each reduced summary P at level $i-1$, it invokes an operator $\mathsf{Spawn}(i, P)$ to generate new patterns, where each new pattern P' at level i is extended from P with single edge $e = (u', u)$, where either u' or u is in P.

(2) For each newly generated pattern P', it validates if P' is a reduced d-summary by invoking a procedure Validate (to be discussed); and if so, adds P' to $\mathcal{C}_P$, and update $\mathcal{P}$ with the newly validated summaries accordingly.

The above process repeats until no pattern within size bound b_p can be spawned from verified summaries. This guarantees the complete and necessary verification of all graph patterns that contributes to the top-k summaries $\mathcal{S}_G$.

Validation. We next introduce the procedure Validate. Given a graph pattern P, it validates if P is a reduced d-summary with two steps below. (1) *Pattern reduction*. The procedure Validate first "reduces" a graph pattern P to its reduced counterpart, by invoking procedure Reduce (Section 3), in polynomial time (Lemma 1). (2) *Verification*. Given a reduced pattern P' generated from Reduce, the procedure Validate validates P' as follows. (1) It first checks, at level-$|P'|$, if there exists a reduced pattern P_r such that $P_r \sim P'$. If so, it sets $\mathsf{supp}(P', G) = \mathsf{supp}(P_r, G)$, *without* verification (Lemma 2, Section 3). (2) Otherwise, it invokes the procedure valiSum (Section 2.2), which checks if P' is a d-summary. If so, it inserts P' to $\mathcal{P}$ as a validated reduced d-summary, and stores $\mathsf{supp}(P', G)$ and base graph G_P' computed by valiSum.

Algorithm sumDiv. Given a set of reduced summaries $\mathcal{C}_P$, the algorithm 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

Algorithm streamDis
Input: a graph G, thresholds b_p and l_p,
integers d and k, time bound t_{max};
Output: summarization $\mathcal{S}_G$.
1. set $\mathcal{S}_G$:=∅; $\mathcal{C}_P$:= ∅; termination:=false; list L:=∅;
2. **while** termination ≠ true **do**
/* *fetch a new summary from summary stream* */
3. summary P_t := sumGen (G, k);
4. $\mathcal{C}_P$:=$\mathcal{C}_P \cup \{P_t\}$;
5. **for each** $L_i \in \mathcal{L}$ **do**
6. Update top l_p pairs in L_i that maximizes $F'(\cdot)$;
7. Update $\mathcal{S}_G$ with top $\lfloor \frac{k}{2} \rfloor$ summary pairs in $\mathcal{L}$;
8. **if** no new summary can be generated
or running time reaches t_{max} **then**
9. termination:=true;
10. **return** $\mathcal{S}_G$;

Fig. 3. Algorithm streamDis.

$$F'(P, P') = (1-\alpha)(I(P) + I(P')) + \alpha * \mathsf{diff}(P, P')$$

That is, F' is obtained by rounding down the original function F, which guarantees an approximation ratio for F. This step is repeated $\lfloor \frac{k}{2} \rfloor$ 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 $\lfloor \frac{k}{2} \rfloor$ rounds of selection.

Analysis. Algorithm sumGen correctly generates and validates all reduced d-summaries, following from the correctness of procedures Reduce and Validate. The diversification over the maximal summaries set $\mathcal{C}_P$ can be reduced to max-sum set diversification problem [15]. sumDiv adopts the greedy strategy that simulates a 2-approximation algorithm for max-sum diversification constrained by the metric diff, hence guarantees approximation ratio 2.

The cost of approxDis is in $O(t_1(G, b_p))$ (the cost of sumGen) + $O(t_2(G, k))$ (the cost of sumDiv) time. (1) Denote as N the total number of patterns generated in sumGen. The total time cost of sumGen is in $O(t_1(G, b_p)) = O(N * b_p|V||E|)$ time. (2) It takes $O(t_2(G, k)) = O(\frac{k}{2}N^2|V|)$ time for sumDiv to find the top-k diversified summaries. Thus, the total cost ofapproxDis is in $O(t_1(G, b_p))$+$O(t_2(G, k)) = O(N * b_p|V||E| + \frac{k}{2}N^2|V|)$ time.

4.2 Anytime Diversified Summarization

The main drawback of the algorithm approxDis is that it needs to *wait* until all the summaries to be validated before the diversification. This may be infeasible under specified resource constraints (e.g., response time). A more feasible strategy is to develop an *anytime* scheme that maintains and reports summaries in an online fashion, over a "stream" of summary candidates as they are validated.

Given a problem I and a function $\mathcal{J}$ to measure the quality of a solution, an algorithm $\mathcal{A}$ is an *anytime* algorithm [16] of I *w.r.t.* $\mathcal{J}$ if: 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. To measure the quality of anytime output of $\mathcal{A}$, we introduce a notation of *anytime approximation*.

Anytime Approximation. An optimal anytime algorithm $\mathcal{A}^*$ will return locally optimal answer $\mathcal{A}^*(I)_t$ at any time t, given the fraction of the input 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.

We present the main result of this section below.

Theorem 1. *There exists an anytime 2-approximation algorithm that computes a diversified summarization, which takes (1) $O(N_t * b_p(b_p + |V|)(b_p + |E|) + \frac{k}{2}N_t^2)$ time, and (2) $O(k * N_t + |\mathcal{S}_G|)$ space, where N_t is the number of summaries it verified when interrupted, and $|\mathcal{S}_G|$ refers to the total size of summaries and their base graphs.*

As a proof of Theorem 1, we next introduce an anytime algorithm for diversified graph summarization.

Overview. The algorithm, denoted as streamDis, integrates the validation and diversification as a *single* process. (1) Instead of waiting for all the summaries to be validated, it operates on a *summary stream*, and incrementally updates $\mathcal{S}_G$ with newly validated summaries whenever possible. (2) It caches a tunable number of summary pairs in k ranked lists that can potentially improve $\mathcal{S}_G$, following the construction of Threshold algorithm [17] for top-k queries. This ensures an adaptive performance of streamDis.

The algorithm streamDis maintains the following: (1) a set $\mathcal{C}_P$ of the reduced summaries validated by sumGen; and (2) a set $\mathcal{L}$ of ranked lists, one list L_i for each summary $P_i \in \mathcal{C}_P$. Each list L_i caches the top-l_p ($n \in [1, k-1]$) summary pairs (P_i, P_j) in $\mathcal{C}_P$ that have the highest $F'(P_i, P_j)$ score, where $F'(\cdot)$ refers to the revised quality function (see Section 4.1). 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 streamDis. Given G, integer k, and two threshold b_p and l_p, the algorithm streamDis 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) t indicate if the termination condition is satisfied (line 1). It then iteratively conducts the following steps.

(1) Invokes sumGen to fetch a newly generated summary P_t. Note that the procedure sumGen can be easily modified to return a single summary after evaluation, instead of returning a set of summaries in a batch.
(2) Updates $\mathcal{C}_P$ and the list $\mathcal{L}$ (lines 5-7) based on the newly fetched summary P_t. For each summary $P_i \in \mathcal{C}_P$, it computes the quality score $F'(P_i, P_t)$, 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_t), if $F'(P_i, P') < F'(P_i, P_t)$.
(3) Incrementally updates the top-k summaries $\mathcal{S}_G$ (lines 5-6). Greedily selects top $\lfloor \frac{k}{2} \rfloor$ pairs of summaries with maximum quality $F'(\cdot)$ from the list set $\mathcal{L}$, and adds the summaries to $\mathcal{S}_G$. If $|\mathcal{S}_G| < k$, a summary $P \in \mathcal{C}_P \setminus \mathcal{S}_G$ 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 8-9): a) no new summary can be discovered in sumGen; or b) running time reaches the time-bound $t_{\max}$. The up-to-date $\mathcal{S}_G$ is then returned.

Example 9. Consider the sample graph G in Fig. 1. Let b_q = 8, k = 2, d = 2 and α = 0.1. streamDis computes a summarization $\mathcal{S}_G$ as follows. In the first round, it invokes sumGen to discover a maximal 2-summary, e.g., P_3, and initializes $\mathcal{C}_P$ and $\mathcal{S}_G$ with P_3. In round 2, it discovers a new 2-summary P_2, verifies $F'(P_2, P_3)$ as $0.9 * (0.20 + 0.18) + 0.1 * 0.90 = 0.43$, and updates L_2, L_3 and $\mathcal{S}_G$ as shown below.

round	L	$\mathcal{C}_P$	$\mathcal{S}_G$
2	$L_2 = \{<(P_2, P_3), 0.43\}$ $L_3 = \{<(P_3, P_2), 0.43\}$	$\{P_2, P_3\}$	$\{P_2, P_3\}$
3	$L_1 = \{<(P_1, P_2), 0.62\}$ $L_2 = \{<(P_2, P_1), 0.62\}$ $L_3 = \{<(P_3, P_1), 0.61\}$	$\{P_1, P_2, P_3\}$	$\{P_1, P_2\}$

In round 3, it discovers summary P_1, and updates the top-1 entries in each list of $\mathcal{L}$. It verifies the pairwise quality scores as $F'(P_1, P_2) = 0.62$ (see Example 8) and $F'(P_1, P_3) = 0.9 * (0.39 + 0.18) + 0.1 * 1.00 = 0.61$. The new top elements in the lists L_1, L_2, and L_3 are hence updated to (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.

As all the maximal summaries within size 8 are discovered, streamDis terminates and returns $\mathcal{S}_G = \{P_1, P_2\}$.

Analysis. The algorithm streamDis is an anytime 2-approximation algorithm. Let $\mathcal{C}_{P_t}$ be the set of summaries generated upto time t, and $\mathcal{S}^*_{G_t}$ be the optimal summarization over the cached summaries $\mathcal{C}_{P_t}$. When $l_p = k - 1$, streamDis simulates its 2-approximation counterpart approxDis to produce a summarization $\mathcal{S}_{G_t}$ where $F(\mathcal{S}_{G_t}) \geq \frac{F(\mathcal{S}^*_{G_t})}{2}$. Note that it suffices to store the top $k-1$ pattern pairs in each list $L_i \in \mathcal{L}$ that maximizes $F'(P_i, P_j)$ to achieve anytime 2-approximation.

5 Parallel Diversified Summarization

The sequential summarization algorithms can be expensive when the graph G is large. Nonetheless, we show that the diversified graph summarization is feasible for large-scale graphs by providing a parallel algorithm, with performance guarantees on both scalability and quality.

Parallel Diversified Summarization. Given a knowledge graph G, a partition strategy $\mathcal{P}$ constructs a fragmentation $\mathcal{G}$ 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. Given a fragmented graph $\mathcal{G}$, n workers, integers k, d and size budget b_p, the parallel diversified summarization problem is to compute the diversified summaries $\mathcal{S}_G$ over the fragmentation $\mathcal{G}$.

For NP-hard problems, a feasible parallel algorithm should demonstrate good scalability with guaranteed accuracy. We start with a new characterization of feasible parallel summarization in Section 4.1. We next introduce the parallel summarization algorithm in Section 5.2.

5.1 Parallel Approximability

To characterize the effectiveness of parallel summarization, we introduce a notion of *parallel scalable approximations*.

Parallel Scalability Revisited [18]. Consider a "yardstick" sequential algorithm that, given graph G, integer d and size bound b_p, approximately computes the summaries, e.g., the algorithm approxDis. Denote the time cost of approxDis as $t(|G|, b_p, k)$. A parallel summarization algorithm A_p is *parallel scalable*if its running time by n processors can be expressed as

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

where $O(1)$ is a "bookkeeping" time to return the results, and n, d, $k \ll |G|$.

Intuitively, parallel scalability measures speedup over a sequential algorithms 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.

Parallel Approximability. Consider an optimization (e.g., maximization) problem with an optimal solution quantified by a single numerical value x. We say the problem is *parallel ϵ-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 $\tilde{x} \leq \epsilon * x$.

We present the main result of this section below.

Theorem 2. *There exists a parallel 2-approximable algorithm* w.r.t. *the sequential algorithm* approxDis *that discovers diversified top-k summaries in time cost in* $O(\frac{T(G,b_p,k)}{n})$.

As a proof of Theorem 2, we develop a parallel summary discovery algorithm, denoted as paraDis. The algorithm paraDis follows Bulk synchronization model. It runs in supersteps, and iteratively executes two procedures in each superstep: (1) Parallel summary validation, denoted as ParsumGen, that "parallelizes" its sequential counterpart sumGen to generate and validate summaries (Section 4.1), and (2) Parallel diversification, denoted as ParsumDiv, that "parallelizes" its sequential counterpart sumDiv (Section 4.1) to update $\mathcal{S}_G$.

Performance Guarantees. Both the algorithms ParsumGen and ParsumDiv work with n processors $S_1, \ldots, S_n$ and a coordinator S_c for necessary synchronization, in parallel. We will show (in Section 5.3) that the algorithm paraDis has the following guarantees. (1) paraDis is parallel scalable. We show that (a) ParsumGen takes in total $O(\frac{t_1(G,b_p)}{n})$ time, where $t_1(G, b_p)$ is the cost of its sequential counterpart sumGen; and (b) ParsumDiv takes in total $O(\frac{t_2(G,k)}{n})$ time, where $t_2(G, k)$ is the cost of its sequential counterpart sumDiv. This ensures that paraDis is parallel scalable. (2) Algorithm paraDis is a 2-approximation. (3) Better still, paraDis demonstrates anytime approximation.

We next introduce the algorithm paraDis in Section 5.2, and provide its performance analysis in Section 5.3.

5.2 Parallel Diversified Summarization

Overview. Given a fragmented graph $\mathcal{G}$, the algorithm paraDis, shown in Fig. 4, executes at most b_p supersteps. It first invokes Spawn(0) to initialize $\mathcal{P}$ with single node patterns (line 2). At each superstep i, paraDis performs the following. (1) It invokes Spawn(i) to generate a set Σ_i of new graph patterns of size i at coordinator S_c (line 4). It then invokes ParsumGen to validate the graph patterns at the workers, in parallel (line 6), and updates $\mathcal{P}$ with newly validated reduced summaries (line 7). (2) paraDis 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 ParsumDiv to collect the top diversified summary pairs $\mathcal{S}_{G_i}$, computed locally at each worker in parallel (line 9), and update $\mathcal{S}_G$ with the summaries that improve the rounded diversification function $F'(\mathcal{S}_G)$ (Section 4.1). This process repeats until in total b_p supersteps are executed, or no new pattern can be generated as indicated by a Boolean flag newP (line 3).

Algorithm paraDis
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 newP:=true;
2. $\mathcal{P}$:=Spawn(0); /* *initialize $\mathcal{P}$ with single-node patterns* */;
3. **while** $i \leq b_p$ **and** newP **do** /* superstep i */
4. set Σ_i:=Spawn(i); **if** $\Sigma_i = \emptyset$ **then** newP:=false;
5. **if** newP **then**
6. set $\mathcal{C}_{P_i}$:= ParsumGen(Σ_i);/**parallel validation*/
7. update $\mathcal{P}$; $\mathcal{C}_P$:=$\mathcal{C}_P \cup \mathcal{C}_{P_i}$;
8. construct work units M_i and distribute M_i to workers;
9. set $\mathcal{S}_{G_i}$:=ParsumDiv(M_i); /**parallel diversification*/
10. update $\mathcal{S}_G$ with $\mathcal{S}_{G_i}$;
11. **return** $\mathcal{S}_G$;

Fig. 4. Algorithm paraDis.

Below we present ParsumGen and ParsumDiv.

Parallel Validation. Upon receiving Σ_i from Spawn(i), ParsumGen validates Σ_i in parallel as follows.

(1) At S_c, for each pattern $P \in \Sigma_i$, ParsumGen identifies a verified pattern P_r in $\mathcal{P}$ such that P is obtained by adding an edge e to P_r. It then constructs a work unit (P_r, e, j), which encodes a request that "validate if P is a d-summary with the base graph of P_r and edge matches e locally at worker S_j". It then distributes all the work units to their corresponding workers to be validated in parallel, following a workload balancing strategy.

(2) Upon receiving a set of work units, for each work unit (P_r, e, j), each worker S_i performs *incremental validation*for P as the "union" of P_r and e, which (a) issues an on-demand fetching of $e(G_k)$, the local edge matches of e from other workers S_k ($k \neq j$), and (b) verifies P in the graph $P_r(G_j) \cup \bigcup_{k\in[1,n]} e(G_k)$ (i.e., the "union" of local matches $P_r(G_j)$ of P_r and all the edge candidates of e) instead of the entire fragment G_j. Each worker stores the local matches $P(G_j)$ for the next round of computation.

For each validated d-summary P, it constructs a bit vector $P.\mathsf{lvec}_j$ with length $|G_j|$ that encodes the matches of P ($P.\mathsf{lvec}[v] = 1$ if v is a match; similarly for edge matches). It returns the set of bit vectors as a message M_j.

(3) Upon receiving all the messages M_j, S_c computes $P.\mathsf{lvec}$ by performing "OR" over $P.\mathsf{lvec}_j (i \in [1, n])$, and obtain the support of P. This completes a round of parallel validation.

Parallel Diversification. Given the verified d-summaries $\mathcal{C}_P$ so far (including $\mathcal{C}_{P_i}$) (line 7), ParsumDiv updates diversified summaries $\mathcal{S}_G$ as follows.

(1) At S_c, for each summary $P \in \mathcal{C}_{P_i}$, paraDis constructs a work unit w_P. The work unit w_P consists of (a) P and the vector $P.\mathsf{lvec}$, and (b) a set of summaries $D_P \subseteq \mathcal{C}_{P_i}$, 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 $M = \bigcup_{P\in\mathcal{C}_{P_i}} w_P$, it distributes M 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 $w_P \in M_j$, each worker S_j computes the distances $\mathsf{diff}(P, P')$ ($P' \in D_P$) by bit vector operations on $P.\mathsf{lvec}$ and $P'.\mathsf{lvec}$. It locally executes a top-k query to find out the local top-k diversified pairs $\mathcal{S}_{G_j}$ that maximize the diversification function F' (Section 4.1), and returns $\mathcal{S}_{G_j}$ to S_c.

(3) The coordinator S_c collects local top-k diversified pairs from all the workers, and updates $\mathcal{S}_{G_i}$ as $\bigcup_{j\in[1,n]}\{\mathcal{S}_{G_j}\}$ (line 10). It then updates the top-k summaries $\mathcal{S}_G$ with the new summary pairs in $\mathcal{S}_{G_i}$.

Optimization. The algorithm paraDis further reduces the parallel cost with the following optimization.

Load balancing. We partition the graph with linear deterministic greedy (LDG) scheme [19], which assign a vertex to the partition where it has the most edges. This helps us cope with the skewed distribution of workloads.

(1) For each work unit (P_r, e, j) in ParsumGen, $e(G)$ is evenly distributed in $\mathcal{P}$ with size bounded by $\frac{|G|}{n}$. ParsumGen quantifies a "runtime skewness" of $P_r(G)$ at S_j as $|1 - \frac{n*|P_r(G_j)|}{|P_r(G)|}|$. If the estimated skewness is above a threshold, it evenly redistributes $P_r(G)$ to all the workers.

(2) For each work unit w_P with a set of summaries D_P, it estimates an upper bound of the diversification cost by processing D_P as $|D_P||V|^2$, and assigns the estimated cost as a weight to each work unit. ParsumDiv then adopts a greedy strategy for a general load balancing problem [20] to iteratively assigns work units with the smallest cost to the workers with the (dynamically updated) least load. By developing an approximation-factor preserving reduction, one can verify that this algorithm is a 2-approximation [20], and is in $O(|W_j| n \log n)$ time, where $|W_j| \leq |\mathcal{C}_{P_i}|$ as there are at most $|\mathcal{C}_{P_i}|$ validated summaries at superstep i.

As verified in Section 7, the load balancing improves the performance of parallel summarization by 7.7 times on average, and remains effective for various skewness caused by "super nodes" with large degrees.

5.3 Performance Analysis

To show that paraDis is parallel scalable relative to its sequential counterpart approxDis, we only need to show that its parallel validation and diversification is parallel scalable relative to their sequential counterparts, respectively.

Parallel Scalability. Recall that the time cost of approxDis is $O(t_1(G, b_p))$ + $O(t_2(G, k))$, where $O(t_1(G, b_p))$ = $O(N * b_p|V||E|)$, and $O(t_2(G, k)) = O(\frac{k^2}{2} N^2|V|)$.

(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) conducts local validation in parallel, which is in $O(\frac{|\Sigma_i| * b_p|V||E|}{n})$ time, and (d) returns the local matches as bit vectors of length $|V|$ in parallel, in $O(\frac{|V||\Sigma_i|}{n})$ time. Taken together, the parallel cost of ParsumGen in each superstep i is $O(\frac{b_p|\Sigma_i||V||E|}{n})$. As there are b_p supersteps that validate in total N patterns ($\sum_i^{b_p} |\Sigma_i| = N$), the total cost is in $O(\frac{N*b_p|V||E|}{n}) = O(\frac{t_1(G,b_p)}{n})$.

(2) At superstep i, ParsumDiv locally computes the distances between P and each summary from work units $w_P \in M_j$ at each worker S_j, and identifies top k

summary pairs, in total $O(\frac{k^2}{2}|M_j||D_P|^2|V|)$ time. This is bounded by $O(\frac{\frac{k^2}{2}|\mathcal{C}_{P_i}|^2|V|}{n})$ time, as all the summaries are from the validated ones $\mathcal{C}_{P_i}$. The parallel cost of sending top summaries to S_c takes $O(\frac{b_p*k}{n})$ time. The total parallel diversification cost is thus in $O(\frac{\frac{k^2}{2}N^2|V|}{n}) = O(\frac{t_2(G,k)}{n})$.

Putting these together, paraDis takes in total $O(\frac{t_1(G,b_p)}{n}) + O(\frac{t_2(G,k)}{n})$ time. Thus, paraDis is parallel scalable *w.r.t.* its sequential counterpart approxDis.

Approximation. The quality guarantee of paraDis follows from the following invariant. (1) At any superstep i, the set $\mathcal{S}_G$ is a 2-approximation of the optimal diversified summaries from the validated summaries. Indeed, ParsumGen correctly validates the patterns in parallel, and it suffices to identify the top-k summaries from the local top-k diversified summary pairs from ParsumDiv. (2) When paraDis terminates, it generates a 2-approximation of the diversified summaries from all the validated ones with size bound b_p. Thus, paraDis is a parallel 2-approximation algorithm.

The above analysis completes the proof of Theorem 2.

6 Knowledge Search with Summaries

A knowledge search query is typically represented as a graph pattern $Q = (V_q, E_q, L_q)$ [2], [3], [4], [5]. Given a knowledge graph G, the answer $Q(G)$ of Q in G refers to the set of all the subgraphs of G that are isomorphic to Q. We next develop summary-based query evaluation algorithms.

"Summaries+Δ" Scheme. Given a query Q, a knowledge graph G and a summarization $\mathcal{S}_G$ of d-summaries, our query evaluation algorithm, denoted as evalSum, only refers to select d-summaries in $\mathcal{S}_G$ and their base graphs as "materialized views" [8], and fetches additional data in G only when necessary. Following its counterpart in [12], it invokes a procedure Select-Sum to select a set of reduced summaries $\mathcal{P}$, and (1) evaluates (the sub-query of) Q covered by $\mathcal{P}$, by invoking existing subgraph search algorithm (e.g., [21]), and by only accessing the base graphs of the summaries in $\mathcal{P}$. It then refines the matches for Q by visiting additional nodes and edges in G, up to a bound amount Δ. The algorithm evalSum visits no more than $B + \Delta$ nodes and edges in G. In practice, both B and Δ can be tuned to adapt to the actual resource bounds.

We next introduce the summary selection strategy.

Summary Selection. Given Q and summaries $\mathcal{S}_G$, as well as a size bound B, we want to find the summaries $\mathcal{P} \subseteq \mathcal{S}_G$, such that a maximum fraction of Q is covered by $\mathcal{P}$, with total base graph size within B. Though desirable, this problem is NP-hard. This can be verified by a reduction from the weighted set cover problem [22]. We thus resort to approximation algorithms.

We first show the following result.

Lemma 4. *A query Q is covered by a summarization $\mathcal{S}_G$, if and only if $\bigcup_{P_i \in \mathcal{S}_G} Q_{P_i} = Q$, where Q_{P_i} refers to the sub-pattern induced by the d-matching from each summary $P_i \in \mathcal{S}_G$ to Q.*

Proof. (1) **If**. Assume $\bigcup_{P_i \in \mathcal{S}_G} Q_{P_i} = Q$. For each edge $e = (u, v)$ in Q, there exists a summary $P_i \in \mathcal{S}_G$ such that e is in Q_P. Hence there exists an edge $e_p = (u_p, v_p)$ in P_i such that $(u_p, u) \in R_d$ and $(v_p, v) \in R_d$, where R_d is the d-similarity between Q and P. For any edge $e' = (u', v')$ in the answer $Q(G)$ where e is mapped to, one can verify that $(u_p, u') \in R'_d$ and $(v_p, v') \in R'_d$, where R'_d is the d-similarity between P_i and G. Hence Q is covered by $\mathcal{S}_G$ by definition. (2) **Only If**. We prove the Only If condition by contradiction. Assume Q is covered but there exists an edge e in Q not covered by any d-summary. Then there exists at least one match of e not included in any base graph, contradicting to the assumption that Q is covered. Lemma 4 thus follows. □

Selection Procedure. Based on Lemma 4, the selection procedure, denoted as Select-Sum, uses a greedy strategy to add the summaries $\mathcal{P}$ that maximally covers Q, and have small base graphs in G. To this end, it dynamically updates a rank $r(P) = \frac{|E_{Q_P} \setminus E_c|}{|G_P|}$ for the summaries in $\mathcal{S}_G$, where (1) E_{Q_P} refers to the edge set of the base graph Q_P, induced by the d-similarity between the summary P and query Q (as a graph) (Lemma 4); (2) E_c refers to the edges of Q that has been "covered", i.e., already in a base graph of a selected summary $P' \in \mathcal{P}$. In each round of selection, a summary with highest $r(P)$ is added to $\mathcal{P}$, and the ranks of the remaining summaries in $\mathcal{S}_G$ are dynamically updated. The process repeats until n patterns are selected, or the total size of the base graphs reaches B.

The selection procedure Select-Sum is efficient: it takes $O(\mathsf{card}(\mathcal{S}_G) b_q (b_q + |V_p|)(b_q + |E_p|))$ time, where b_q and $|V_q|$, $|E_q|$ are typically small. Better still, it guarantees the approximation ratio $(1 - \frac{1}{e})$ for optimal summaries under budget B, by reducing the summary selection to the budgeted maximum coverage problem [22].

7 Experimental Evaluation

Using real-world and synthetic knowledge graphs, we conducted four sets of experiments to evaluate (1) Performance of the summary mining algorithms approxDis and streamDis; (2) Scalability of the parallel summarization algorithm paraDis, (3) Effectiveness of the algorithm evalSum for query evaluation; and (4) Effectiveness of the summary model, using a case study.

Experimental Setting. We used the following setting.

Datasets. We use three real-life knowledge graphs: (1) *DBpedia*[1] consists of 4.86M nodes and 15M edges, where each entity carries 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 with more diversified (324343) labels; and (3) *Freebase*(version 14-04-14),[3] with 40.32M entities, 63.2M relationships, 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 60 M), edges (up to 152 M), and labels drawn from an alphabet of 3080 labels.

Queries. To evaluate evalSum, we generated 50 subgraph queries $Q = (V_q, E_q, L_q)$ over real-world graphs with size controlled by $(|V_p|, |E_p|)$. We inspected meaningful queries 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 500 queries with labels drawn from BSBM alphabet. We

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/

generate queries with different topologies (star, trees, and cyclic patterns) and sizes, ranging from (4,6) to (8,14).

Algorithms. We implemented the algorithms below in Java.

(1) Summarization algorithms approxDis and streamDis, compared with two baselines. (a) GRAMI, an open-source graph mining tool [10] that discovers frequent subgraphs as summaries. The base graph of a summary P refers to the subgraph of G induced by the edge matches, specified by all the subgraph isomorphism mappings from P to G. (b) heuDis, a heuristic counterpart of streamDis that incrementally maintains a diversified summarization $\mathcal{S}_G$ over the stream of summaries following [23]. Each time a new summary P is validated, it swaps out a summary P' in $\mathcal{S}_G$ if $F(\mathcal{S}_G \setminus \{P'\} \cup \{P\}) > F(\mathcal{S}_G)$. The algorithm does not guarantee 2-approximation.

(2) The parallel summarization algorithm paraDis (including procedures ParsumGen and ParsumDiv), compared with paraDisn, its counterpart without load balancing strategy.

(3) Query evaluation algorithm evalSum, compared with three variants: (a) evalRnd, a counterpart of evalSum that performs random selection instead of summary selection Select-Sum (Section 6); (b) evalGRAMI, which accesses the base graphs of the frequent subgraphs from GRAMI; and (c) evalNo that directly evaluates Q by accessing G with an optimized subgraph isomorphism algorithm in [21]. We allow a resource bound Δ to be posed on evalRnd and evalGRAMI to allow them to return approximate answers by fetching at most Δ additional data from G.

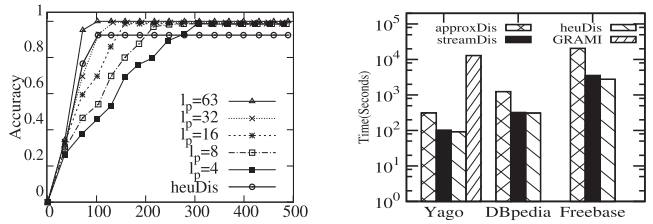
Partition Strategy. We implemented three partition strategies. (1) LDG [19], which assign a vertex to the partition where it has the most edges; (2) METIS [24], a well-regarded offline multilevel partitioning heuristic; and (3) Hashing [19], which assign a vertex based on its id and a hashing function. By default, we use LDG, unless otherwise specified.

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 2.8 GHz CPU and 16G of memory. We ran each experiment 5 times and report the averaged results.

Overview of Results. We summarize our findings below.

(1) Mining reduced summaries is feasible over large real-world graphs. (**Exp-1**). For example, the algorithm streamDis takes 90 seconds to generate summarizations with 99 percent of the quality of their counterparts from approxDis on YAGO with 3.91 million entities and relationships, and is orders of magnitude faster than GRAMI that is based on frequent subgraphs. We also found for reduced summaries, the cost of approxDis is 61 percent less on average compared with its counterpart in [12], due to that it prunes many redundant summaries that are non-reduced.

(2) It is feasible to summarize large-scale knowledge graphs in parallel (*Exp-2*). For example, It takes 55 seconds for paraDis to find diversified summaries with 20 workers over YAGO. The performance is improved by 3.3 times when the number of workers increases from 4 to 20.



(a) streamDis: quality vs. time(s) (b) Real-world datasets

Fig. 5. Sequential summarization: Performance.

(3) The summary-based search significantly improves the efficiency of query evaluation (*Exp-3*). For example, evalSum is 40 times faster than evalNo (without using summarization) over YAGO, and 2.5 times faster than evalGRAMI that access frequent subgraphs. In general, it does not take much additional cost ($\Delta \leq 5\%|G|$) to find exact answers.

(4) Our case study shows that summarization captured by d-summaries is concise, and provides a good coverage for diversified entities. Moreover, reduced summaries can be applied to query suggestion and knowledge base completion, as verified by our case study (*Exp-4*).

We next report the details of our findings.

Exp-1: Effectiveness of Summary Discovery. We fixed parameter α = 0.5 for diversification, k = 64, the summary size bound b_p = 6, $d = 1$ and l_p = k-1 for this experiment, unless otherwise specified. In addition, we set a support threshold θ = 0.005 for sumGen used by approxDis, streamDis, and heuDis. For GRAMI, we carefully adjusted its support threshold to allow the generation of patterns with similar label set and size to those from approxDis. We also excluded "overly general" (top 2 percent frequent) labels such as "Thing".

Anytime Performance. We evaluate the anytime performance of streamDis and heuDis in terms of the following. (1) We report the "anytime accuracy" of streamDis as $\frac{F(\mathcal{S}_{G_t})}{F(\mathcal{S}_G)}$, where $\mathcal{S}_{G_t}$ refers to the summaries returned by streamDis at time t, and $\mathcal{S}_G$ refers to the one returned by approxDis. The accuracy of heuDis is defined similarly. (2) We also report the "convergence" time of streamDis and heuDis when the accuracy reaches 99 percent as a satisfiable quality.

Fig. 5a shows the anytime accuracy of streamDis and heuDis over YAGO. (1) The quality of summaries from both algorithms increases as t and l_p increases. (2) streamDis convergences faster to near-optimal summarization with larger l_p, as more summary pairs are compared. Remarkably, it converges after processing 70 patterns (in less than 100 seconds) when l_p = 63. heuDis converges faster than streamDis, but stops at accuracy 0.9 on average. These results verify that streamDis reasonably trades time with accuracy, with desirable summarization quality.

Efficiency of Sequential Summarization. We report the performance of sequential summarization algorithms over real-world datasets. For the two anytime algorithms streamDis and heuDis, we report their convergence time. As shown in Fig. 5b, (1) streamDis and approxDis are both orders of magnitude faster than GRAMI. The latter does not run to completion within 10 hours over both DBpedia and Freebase; (2) Performance of streamDis is comparable to that of heuDis, and streamDis is 3-6 times faster than approxDis with comparable accuracy; (3) streamDis is

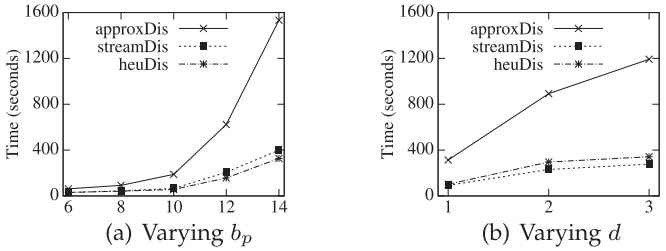


Fig. 6. The impact of bound b_p and d.

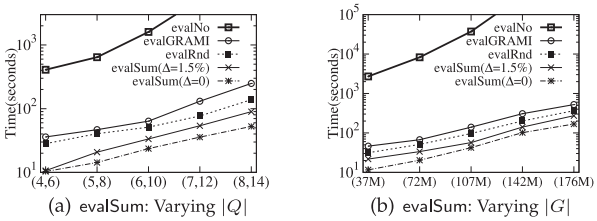


Fig. 7. Efficiency of evalSum.

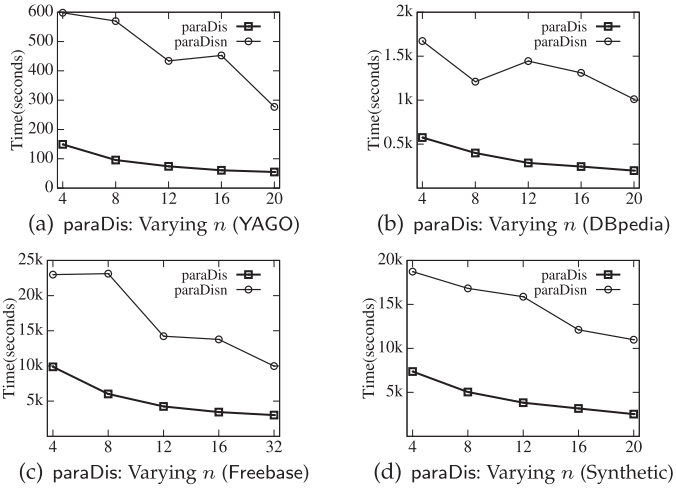


Fig. 8. Scalability of paraDis.

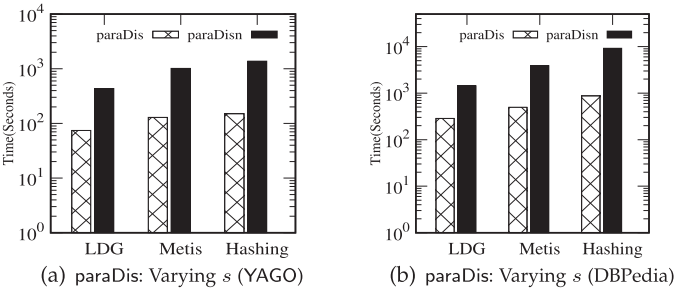


Fig. 9. Load balancing of paraDis (Varying skewness).

feasible over large knowledge graphs. For example, it takes less than 100 seconds to produce high-quality summaries by verifying only 64 summaries for YAGO.

Varying b_p. Using the settings in Fig. 5a over YAGO, we varied the summary size threshold b_p from 6 to 14 (node # + edge#). As shown in Fig. 6a, all the algorithms take more time with larger values of b_p, as more candidate patterns are examined and verified. On average, streamDis is 4 times faster than approxDis with 90 percent accuracy.

Varying d. Using the same setting over YAGO, we varied d from 1 to 3. Fig. 6b shows that all the algorithms take more time with larger d, as expected. Additionally, the convergence time of streamDis and heuDis are less sensitive to increasing of d when compared with approxDis.

We also evaluated the scalability of the sequential algorithms using larger synthetic graphs, by varying $|G|$ from $(10M, 27M)$ to $(60M, 152M)$ (not shown). The algorithms streamDis and heuDis scale well with larger $|G|$ (less than 1 hour over graphs of size $(60M, 152M)$), and are less sensitive to increasing $|G|$ due to their early convergence. In contrast, GRAMI does not run to completion in 10 hours over graphs of size $(10M, 27M)$.

Exp-2: Parallel Summarization. We next evaluate the scalability of parallel algorithm paraDis by varying the number of workers and the effectiveness of load balancing strategy by varying the skewness of partitioned graphs.

Varying n. Using the same setting for α, b_p, k and d as in Exp-1, we varied the number of workers n from 4 to 20, and report the performance of paraDis over the three real-world datasets in Figs. 8a, 8b, and 8c, respectively. We find the following. (1) paraDis scales well with n. The performance of paraDis is improved by 3.3 times when n is increased from 4 to 20. (2) The load balancing strategy improves the performance of paraDisn by 4.4 times on average. In general, it is feasible to summarize large graphs. For example, it takes 200 seconds for paraDis to summarize DBpedia, when $n = 20$.

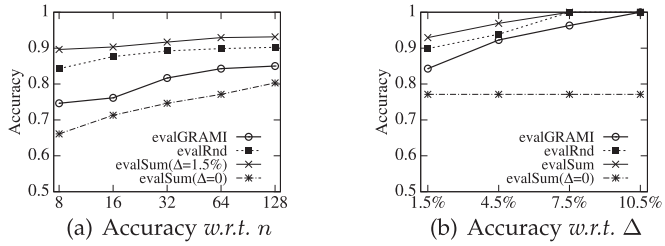
Using the same setting, we evaluated the performance of paraDis over a large synthetic graph with size $(60M, 152M)$. The result shown in Fig. 8d is consistent with its counterparts over real-world graphs.

Varying Skewness. Fixing $n = 12$, and using the same setting for α, b_p, k and d as in Exp-1, we evaluated the impact of partition strategies to paraDis, compared with its counterpart paraDisn without load balancing. We define the *skewness* as the standard deviation of the largest degrees of the border nodes from all the 12 partition, and generated partitions using LDG, METIS and Hashing over *Yago* and *BSBM*, respectively. The skewness values for these three methods are 1.9k, 2.9k and 3.3k (resp. 8.8k, 12.6k, 15.1k) for *Yago* (resp. *BSBM*). The result shown in Fig. 9 tells us the following: (1) Both algorithms take more time with more "skewed" fragments, due to larger communication cost; on the other hand, (2) paraDis outperforms paraDisn by 7.7 times on average, and is much less sensitive to the change of skewness due to the load balancing strategy.

Exp-3: Effectiveness of evalSum. We evaluate the efficiency of evalSum, and compare it with evalSum ($\Delta = 0$), evalRnd, evalGRAMI, and evalNo.

Varying $|Q|$. Fixing m (the number of selected summaries) as 64 and card ($\mathcal{S}_G$) as 500, we varied the query size $|Q|$ from $(4, 6)$ to $(8, 14)$ over YAGO. Fig. 7a tells us the following. (1) By leveraging 2-summaries, evalSum and evalSum ($\Delta = 0$) find answers in less than 60 seconds, and improves the efficiency of evalNo by 40 and 50 times respectively; evalNo does not terminate within 10^4 seconds for queries with 6 nodes. (2) On average, evalSum and evalSum ($\Delta = 0$) are 2.5 and 4 times faster than evalGRAMI, respectively. Indeed, we found that frequent subgraphs as summaries cover less answers of queries due to more strict matching semantics.

The summary selection is also effective. In all cases, it takes less than 10 seconds, and improves the response time of evalRnd (with random selection) by 2 times. We also evaluate the scalability of evalSum with synthetic graphs with different sizes. The result (Fig. 7b) shows that evalSum scales better than other algorithms.



(a) Accuracy *w.r.t.* n (b) Accuracy *w.r.t.* Δ

Fig. 10. Accuracy of evalSum.

Accuracy. We evaluated the accuracy of the query answers produced by evalSum ($\Delta = 0$), evalRnd, and evalGRAMI. 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 evalNo, the accuracy is 1. As shown in Fig. 10a and 10b, all algorithms perform better with larger n, and evalSum achieves the highest accuracy with $\Delta = 1.5\%$. Remarkably, evalSum can get 100 percent accuracy with 7.5 percent of the original graph, while evalGRAMI needs more data compared to evalSum.

Query Diversity. We also compared the performance of evalSum ($\Delta = 1.5\%$) with evalNo over three categories of queries over YAGO: (1) *Frequent*, which carries 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, where C (resp. C_{ISO}) refers to the total number of nodes and edges (including summaries) visited by evalSum (resp. evalNo).

evalSum 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 27 percent (resp. 8 percent) data visited by evalNo (resp. $|G|$) to achieve accuracies not less than 93 percent.

Exp-4: Case Study. We performed case studies to evaluate the practical application of the knowledge summaries.

Keyword Search. We first investigate how reduced summaries can support ambiguous keyword search in knowledge graphs. We sampled 50 ambiguous keywords from DBpedia (e.g., "waterloo", "Avatar"), each has on average 4 different types. We invokes approxDis to output top diversified reduced summaries with entities that matches the keywords. (1) We found that reduced summaries can distinguish ambiguous terms. For example, the top-3 summaries distinguish "waterloo" as Battle, University, and Films. These summaries suggest intermediate keywords as enhanced queries (e.g., Military Person); as well as diversified facts. (2) More diversified summarization requires less summaries to cover all possible types of keywords. For example, it takes at most 15 reduced summaries to cover all the types for each keyword when $\alpha = 0.9$. In contrast, most of the summaries from GRAMI are redundant. It cannot cover the entity types of keywords even with 64 summaries.

Cross-Domain Queries. We evaluate how the summaries can be used to support "cross-domain" querying over multiple knowledge bases [1]. We generated 20 cross-domain queries over YAGO and DBpedia. We also extended evalSum to evaluate the queries by accessing the summaries of YAGO and DBpedia, respectively, and "merges" the matches from each if they have the same URI, to form a complete answer.

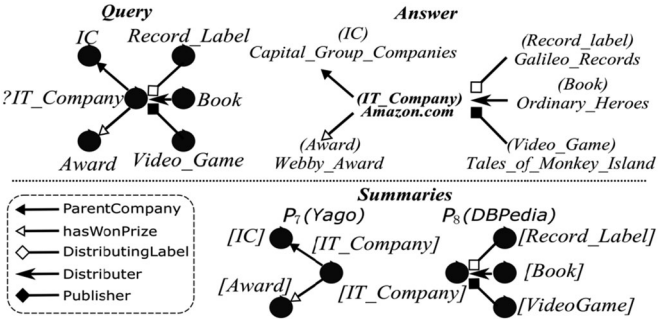


Fig. 11. Cross-domain queries over DBpedia and Freebase.

TABLE 2
Average Accuracy of Fact Checking

	YAGO				DBpedia			
Model	Acc.	Prec.	Rec.	F1.	Acc.	Prec.	Rec.	F1.
PRA	0.87	0.64	0.35	0.38	0.89	0.70	0.43	0.47
SFE	0.81	0.56	0.89	0.67	0.69	0.49	0.95	0.60
Summary	0.92	0.84	0.62	0.69	0.89	0.76	0.63	0.67

We show a query and its answer in Fig. 11. The query finds Award wining IT_Companies with specified products and their parent companies(IC). While YAGO reports the parent company, and DBpedia provides products information, evalSum reports complete answer(Amazon.com) by accessing summaries P_7 and P_8 from YAGO and DBpedia, respectively, and integrating the partial answers.

Fact Checking. We also evaluate how the summaries can be used to support fact checking. Given a knowledge base G and a new fact (an edge) $e = (u, v)$ with edge label r, it is to predict whether e belongs to a missing part of G. Two established models are (1) Path ranking (PRA) [25], which samples paths with length up to d via random walks from a set of training (true) facts, extracts path features and adopts logistic regression to train a binary classifier; and (2) SFE [26], which samples paths from subgraphs that contain the facts, and constructs enhanced path features from the subgraphs by e.g., replacing edge type with similar ones.

We developed a model (Summary) that extends PRA with reduced summaries as follows. We select 20 triple patterns, where a triple pattern $r(x, y)$ is a single-edge graph pattern with nodes having labels x and y, connected by relation r. For each pattern $r(x, y)$, we sample 80 percent of its instances as training set (true facts) and the rest 20 percent instances as testing set. Given $r(x, y)$ and its training set, we invoke streamDis to discovery reduced d-summaries with base graphs that contain the training facts. For each true fact e and each summary P, we construct a feature vector, where each entry encodes whether there exists a base graph of P that contains e. We fed the feature vectors to PRA to train a binary classifier. We report the average precision, recall and accuracy (the ratio of facts that can be predicted correctly) of these models, over the 20 triple patterns.

Interestingly, this simple extension already improves the accuracy of both PRA and SFE. As reported in Table 2, Summary achieves additional 43 percent (resp. 11 percent) gain of F1 score over DBpedia compared with PRA (resp. SFE). A closer inspection of the features of Summary shows that d-summaries can suggest subgraphs that are more discriminant to "define" true facts, compared with paths induced by random-walk [25]. The latter which may involve noise and

non-discriminant features. On the other hand, Summary has lower recall than SFE, as path features are able to "cover" more true cases than subgraphs that pose more topological constraints to identify true facts. We defer the study of summary-based fact prediction to future work (Section 9).

8 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 [9], [14], [27], [28], [29], [30]. These approaches can be classified as follows: (1) Graph compression, which aim to compress graphs within a bounded error by minimizing a information complexity measure [9], [27], [29], e.g., Minimum Description Length (MDL), or to reduce the space cost such that the topology of the original data graph can be approximately restored [27], [29]. The algorithm in [9] 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 [28]. (3) (Bi)simulation relation is adopted [31] to group paths carrying same labels up to a bounded length. Relaxed bisimulation has also been studied to generate summaries over a set of answers [30]. This work summarizes the entities only when they are pairwise similar, which can be an overkill for knowledge graphs. (4) Entity summarization [14] generates diversified answers for entity search instead of general subgraph 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 [9], [27], [29]. (2) We discovery summaries to access single graphs rather than for query answers [14], [30], and 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 [27], [29]. (4) In contrast to [27], [28], our summary model requires little parameter tuning effort. In addition, diversified summaries are not addressed in these works.

Graph Pattern Mining. Clustering approaches have been studied to group a set of similar graphs [32]. These techniques can not be applied for summarizing a single graph. Frequent subgraph patterns can be mined from a single graph to describe large graphs [9], [10].

Parallel algorithms have been developed for pattern mining in terms of subgraph isomorphism, for transactional graph databases [34] or single graph [35]. These methods can not be readily applied for diversified summarization for a single graph. We develop parallel scalable algorithms that are not addressed in prior work.

Answering Queries Using Views: View-based query evaluation has been shown to be effective for SPARQL [7] and pattern queries [8]. It typically requires equivalent query rewriting by accessing views defined in the same query language. By contrast, (1) we show that reduced summaries can be used to evaluate graph queries defined by subgraph isomorphism, which are not defined in the same language; and (2) We develop feasible summarization algorithms as view discovery process. These are not addressed in [7], [8].

Graph summaries can also be used to enhance machine learning models for fact prediction and reasoning in knowledge graphs. Notable models for this task include path-based models [25], [26], [36], recurrent neural networks [37], and reinforcement learning [38]. PRA [25] extracts features from paths around training facts via random walk with restarts to train models that validate new facts. To improve model accuracy, SFE [26] extends PRA with more expressive path features extracted from subgraphs that are induced by random walks. For example, it uses one-sided paths that do not necessarily connect two entities of a fact, and similarity features that encode paths with similar relations. DeepPath [38], which is a reinforcement learning based method, uses a policy-based agent based on knowledge graph embeddings and samples the most promising relation to extend its path. While all these models use path features, graph summaries can explicitly encode features as subgraphs they summarize, beyond paths. This indicates more discriminant features and more accurate models, as verified by our case study.

9 Conclusions

We proposed a class of reduced d-summaries, and developed sequential and parallel summarization algorithms for large knowledge graphs. We also developed query evaluation algorithm by effective summary selection. Our experimental results verified that our algorithms are feasible, and can significantly reduce the cost of knowledge graph query evaluation. One future topic is to develop summary-based algorithms for more types of analytical queries beyond subgraph queries. Another topic is to extend summaries with similarity functions as seen in graph embedding, and to study their applications in knowledge base completion supported by neural networks and reinforcement learning.

Acknowledgments

Qi Song and Yinghui Wu are supported in part by NSF IIS-1633629 and a Google Faculty Research Award.

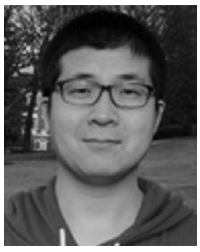
References

[1] X. Dong, et al., "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proc. 20th ACM SIGKDD Int Conf. Knowl. Discovery Data Mining*, 2014, pp. 601–610.
[2] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum, "Naga: Searching and ranking knowledge," in *Proc. IEEE 24th Int. Conf Data Eng.*, 2008, pp. 953–962.
[3] B. Quilitz and U. Leser, "Querying distributed RDF data sources with SPARQL," in *Proc. 5th Eur. Semantic Web Conf. Semantic Web: Res. Appl.*, 2008, pp. 524–538.
[4] W. Le, F. Li, A. Kementsietsidis, and S. Duan, "Scalable keyword search on large RDF data," *IEEE Trans. Knowledge Data Eng.*, vol. 26, no. 11, pp. 2774–2788, Nov. 2014.
[5] S. Yang, Y. Wu, H. Sun, and X. Yan, "Schemaless and structureless graph querying," *Proc. VLDB Endowment*, vol. 7, no. 7, pp. 565–576, 2014.
[6] W. Fan, X. Wang, and Y. Wu, "Querying big graphs within bounded resources," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 301–312.
[7] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang, "Rewriting queries on SPARQL views," in *Proc. 20th Int Conf. World Wide Web*, 2011, pp. 655–664.
[8] W. Fan, X. Wang, and Y. Wu, "Answering graph pattern queries using views," in *IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 184–195.
[9] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "VOG: Summarizing and understanding large graphs," in *Stat. Anal. Data Mining*, vol. 8, pp. 183–202, 2014.
[10] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proc. VLDB Endowment*, vol. 7, no. 7, pp. 517–528, 2014.

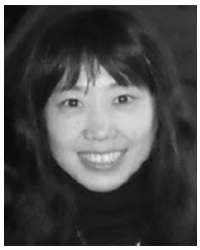
[11] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Subdue: Compression-based frequent pattern discovery in graph data," in *Proc. 1st Int. Workshop Open Source Data Mining: Frequent Pattern Mining Implementations*, 2005, pp. 71–76.
[12] Q. Song, Y. Wu, and X. L. Dong, "Mining summaries for knowledge graph search," in *IEEE 16th Int. Conf. Data Mining*, 2016, pp. 1215–1220.
[13] F. Pennerath and A. Napoli, "The model of most informative patterns and its application to knowledge extraction from graph databases," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2009, pp. 205–220.
[14] M. Sydow, M. Pikuła, and R. Schenkel, "To diversify or not to diversify entity summaries on RDF knowledge graphs?" in *Proc. Found. Intell. Syst.*, 2011, pp. 490–500.
[15] S. Gollapudi and A. Sharma, "An axiomatic approach for result diversification," in *Proc. 18th Int. Conf. World Wide Web*, 2009, pp. 381–390.
[16] B. Arai, G. Das, D. Gunopulos, and N. Koudas, "Anytime measures for top-k algorithms," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 914–925.
[17] N. Bruno and H. Wang, "The threshold algorithm: From middleware systems to the relational engine," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 4, pp. 523–537, Apr. 2007.
[18] W. Fan, X. Wang, and Y. Wu, "Distributed graph simulation: Impossibility and possibility," *Proc. VLDB Endowment*, vol. 7, no. 12, 2014.
[19] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1222–1230.
[20] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Math. Program.*, vol. 62, no. 1–3, pp. 461–474, 1993.
[21] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *Proc. VLDB Endowment*, vol. 8, no. 5, pp. 617–628, 2015.
[22] V. V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer, 2003.
[23] E. Minack, W. Siberski, and W. Nejdl, "Incremental diversification for very large sets: a streaming-based approach," in *Proc. 34th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2011, pp. 585–594.
[24] G. Karypis and V. Kumar, "Multilevel graph partitioning schemes," in *Proc. ICPP (3)*, 1995, pp. 113–122.
[25] N. Lao, T. Mitchell, and W. W. Cohen, "Random walk inference and learning in a large scale knowledge base," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2011, pp. 529–539.
[26] M. Gardner and T. M. Mitchell, "Efficient and expressive knowledge base completion using subgraph feature extraction." in *Proc. Int. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1488–1498.
[27] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 419–432.
[28] Y. Tian, R. Hankins, and J. Patel, "Efficient aggregation for graph summarization," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 567–580.
[29] M. Riondato, D. Garcia-Soriano, and F. Bonchi, "Graph summarization with quality guarantees," in *Proc. IEEE Int. Conf. Data Mining*, 2014, pp. 947–952.
[30] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan, "Summarizing answer graphs induced by keyword queries," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 1774–1785, 2013.
[31] Q. Chen, A. Lim, and K. W. Ong, "D (k)-index: An adaptive structural summary for graph-structured data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2003, pp. 134–144.
[32] C. C. Aggarwal and H. Wang, "A survey of clustering algorithms for graph data," in *Proc. Manag. Mining Graph Data*, 2010, pp. 275–301.
[33] D. Xin, H. Cheng, X. Yan, and J. Han, "Extracting redundancy-aware top-k patterns," in *Proc. 12th ACM SIGKDD Int. Conf Knowl. Discovery Data Mining*, 2006, pp. 444–453.
[34] D. J. Cook, L. B. Holder, G. Galal, and R. Maglothin, "Approaches to parallel graph-based knowledge discovery," *J. Parall. Distrib. Comput.*, vol. 61, no. 3, pp. 427–446, 2001.
[35] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: A system for distributed graph mining," in *Proc. 25th Symp Oper. Syst. Principles*, 2015, pp. 425–440.
[36] A. Neelakantan, B. Roth, and A. Mccallum, "Compositional vector space models for knowledge base completion," in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics, 7th Int. Joint Conf. Asian Federation Natural Lang. Process.*, 2015, pp. 156–166.
[37] R. Das, A. Neelakantan, D. Belanger, and A. McCallum, "Chains of reasoning over entities, relations, and text using recurrent neural networks," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics: Vol. 1*, 2017, pp. 132–141.
[38] W. Xiong, T. Hoang, and W. Y. Wang, "DeepPath: A reinforcement learning method for knowledge graph reasoning," in *Proc. 2017 Conf. Empirical Methods Natural Language Process.*, 2017, pp. 564–573.

Qi Song received the BS and MS degrees in computer science from Beihang University, in China. He is currently working toward the PhD degree in computer science at Washington State University. His research interests include distributed graph data mining and neural network based machine learning for graphs.

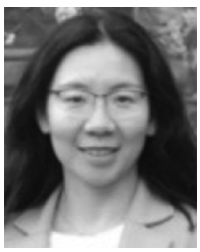
Yinghui Wu received the PhD degree from the University of Edinburgh, in 2010. He is an assistant professor in the School of EECS, Washington State University. His current research interests include big data, graph databases and network science, with applications in social and information network analytics, and network security.



Peng Lin received the BS and MS degrees in automation from Shanghai Jiao Tong University, in China. He is working toward the PhD degree at Washington State University. His current research interests include knowledge base management and data quality in graphs.



Luna Xin Dong received the PhD degree in computer science and engineering from the University of Washington. She has been a principal scientist with Amazon since July 2016, leading the efforts to build Amazon Product Graph. Prior to that, she worked at A&T LabsResearch. Her research interest includes data integration, data cleaning, and knowledge management.



Hui Sun received the BS, MS, and PhD degrees from Tsinghua University, Beijing, China, all in computer science. She is an assistant professor in the School of Information, Renmin University of China, Beijing, China, and is currently a visiting scholar in the School of EECS, Washington State University. Her current research interests include high-performance database, and graph data mining.