Diversified Keyword Expansion on Multi-labeled Graphs

Mohammad Hossein Namaki(✉), Yinghui Wu, and Xin Zhang

Washington State University, Pullman, USA
{mnamaki,yinghui,xzhang2}@eecs.wsu.edu

Abstract. Keyword search has been widely adopted to explore graph data. Due to the intrinsic ambiguity of terms, it is desirable to develop query expansion techniques to find useful and diversified information progressively in large graphs. To support exploration with keywords, we study the problem of *diversified keyword expansion* in graphs. Given a set of validated content nodes in a graph, it is to find a set of terms that maximizes the aggregated relevance of the validated nodes. Moreover, the terms should be diversified to cover different search interests. We develop a fast stream-based $(\frac{1}{2}\text{-}\epsilon)$-approximation to suggest diversified terms, which guarantees a linear scan of the terms in the content nodes up to a bounded area with small update cost. Using real-world graphs, we experimentally verify the effectiveness and efficiency of our algorithms, and their applications in knowledge base exploration.

Keywords: Keyword query expansion · Submodular maximization

1 Introduction

Keyword search (KWS) has been used to explore and understand graph data [4,18,25,26]. A keyword query Q is a set of keywords $\{t_1, \ldots, t_l\}$. Given a keyword query Q and a graph G, an answer of Q is a subgraph of G that contains a set of *content nodes*. Each content node matches to a term in Q. The subgraph can be *e.g.,* a minimal rooted tree [29], a weighted Steiner tree [5], or an r-clique [14], when Q is distinct root-based (DR), Steiner tree-based (ST), or Steiner graph-based (SG), respectively [25]. In practice, answers can be ranked by established distance-based relevance, such as shortest paths in answers [5], semantic distances [33], and normalized Web distances [8].

To cope with the ambiguity of keywords, exploratory search [1,4,19,26] exploits interactive search sessions to find useful information progressively. Each session interleaves query suggestion and query evaluation [7]. Effective graph exploration with KWS is nevertheless challenging. (1) The interactive exploration requires fast query suggestion, which aims to discover both relevant and diversified terms [6,16]. While the nodes in real-world graphs are often multi-labeled, searching for good terms is usually expensive. (2) Real-world graphs are also dynamic in that new terms are constantly inserted. Query suggestion needs to respond to such changes in a timely manner.

We consider an example from KWS-based knowledge exploration.

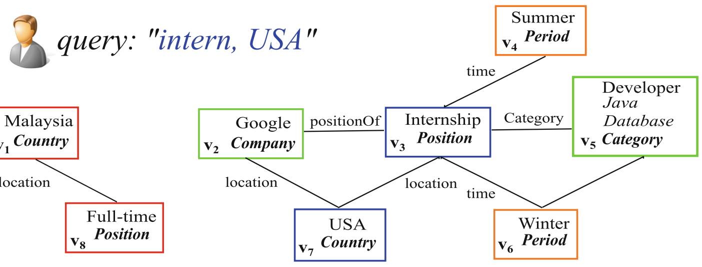


Fig. 1. Graph exploration with KWS (Color figure online)

Example 1. Figure 1 illustrates a fraction of a professional network G, such as LinkedIn knowledge graph. Each entity carries multiple terms, such as "Java, Database" that describe required skills, Developer as the title, and Category as its type. A user may search for internship opportunities with a query Q that contains terms "Intern" and "USA", and an answer $Q(G)$ that contains two entities with exactly matched labels marked in blue rectangles. While $Q(G)$ is not very informative, several terms can be suggested to help users to specify the search. (a) {Full-time, Malaysia} may be suggested due to co-occurrence of their types with the terms in Q following keyword suggestion for relational database [20], which finds job opportunities in Malaysia. This is not very relevant to the search intent as it ignores the semantic connection of entities in the information network. (b) The terms Summer and Winter are also relevant to $Q(G)$. But these are quite redundant due to that both suggest time, and lead to less informative queries.

A more desirable suggestion can be {Google, Developer}. Both terms are relevant to $Q(G)$ in distance-based semantic measures. Moreover, one covers a perspective of "USA companies" for the term "USA", and another specifies the titles for the term "internship". The user can issue a query expanded with either or both of the terms to perform diversified exploration.

The need of diversified query expansion for graphs is evident in knowledge base exploration [1], "why-not" queries [22], and may help to identify highly-correlated queries to reverse-engineering sensitive entity information [23]. However, discovering such terms can be expensive. For example, one may need to search for nearest neighbors of all validated nodes in $Q(G)$, and validate possible combinations of terms carried by these nodes to maximize diversity. Moreover, new terms can be inserted to G. For example, when a term "Tensorflow" is inserted to most relevant nodes with title "Developer" as a new required skill, a query suggestion scheme should respond fast and suggest the term.

Contribution. This paper studies diversified keyword expansion in graphs. We develop expansion algorithms with provable guarantees on term quality and time cost. We make the following contributions.
(1) Diversified KWS *Query Expansion.* Given a set V_C of validated content nodes, and a graph G, we identify a set of terms with maximized aggregated relevance to V_C, and suggest diversified search intent. We characterize the aggregated relevance of a term with minimum weighted paths from its content nodes to V_C. We also introduce a diversification measure by rewarding terms with new content nodes in V_C connected by such paths.

(2) Stream-Based Approximation. We show that the diversified query expansion problem is $(\frac{1}{2}\text{-}\epsilon)$-approximable for constant $\epsilon > 0$, and develop a stream-style query expansion algorithm. The algorithm follows the Threshold Algorithm (TA) to generate a stream of terms with aggregated relevance via bounded traversal, and maintains lists to approximate top-k terms with early termination. The algorithm takes small update cost upon discovery of terms, and can naturally cope with newly inserted terms without more update cost.

(3) KWS*-Based Graph Exploration.* We specialize the algorithm for established KWS classes DR, ST, and SG. By simple adaption, we can devise diversified graph exploratory schemes that make use of these query classes, with matching performance guarantees. We evaluate the effectiveness and efficiency of our query expansion algorithms, using real world graphs (Sect. 4). We also show that these algorithms readily lead to flexible KWS for graph exploration, as verified by our case studies in knowledge bases and recommendation networks.

Related Work. We categorize the related work as follows.

Keyword Search in Graphs. Keyword search (KWS) has been studied extensively for unstructured [29] and structured [25] data. In graphs, KWS is to find subgraphs that contain content nodes that match given keyword queries. There are several established KWS query classes. (1) Distinct root-based KWS (DR) [11,13] defines $Q(G)$ as a minimal rooted tree that contains all content nodes at its leaves. (2) KWS with Steiner trees (ST) differs from DR queries in that it aims to find a weighted Steiner tree that minimizes the total edge weights. (3) Steiner graph-based KWS (SG) finds a set of content nodes such that the sum of pairwise distances is minimized (or bounded as r-cliques [14]).

Standard KWS follows "query-response" paradigms that assume accurate queries. We study diversified query expansion that can directly support graph exploration with established DR, ST, and SG queries, and provide parameters to enable tunable query expansion and graph exploration.

Query Suggestion. Keyword query suggestion and its variants (*e.g.,* query refinement [17] and reformulation [28]) have been studied to suggest new queries that better describe search intent. Prior work mostly adopts information retrieval methods that access query logs and user feedback [7] for Web search. Such log is scarce for emerging graph search applications. Co-occurring term retrieval (CoOcc) [20] discovers relevant keywords as those that appear most frequently in the results of original query Q. Tag cloud (TagCloud) [15] discovers terms that are highly relevant to the initial results with *e.g.,* tf-idf measures. These methods are designed for relational data rather than general graphs, where the relevance are often more involved with graph topology. Subgraphs are extracted to suggest structured queries by approximate KWS [24]. These methods cannot be directly applied for graph exploration with keywords.

Mismatch [30] aims to provide explanations for XML queries with empty answers. To this end, approximate answers of a query is computed by expanding its original answers with content nodes of the same type, and new terms are suggested to replace those without match. While Mismatch aims to find good replacement of terms over typed XML data, our work specifies diversified terms to expand original queries for general, multi-labeled graphs. Little has been done on KWS expansion to support diversified exploration in multi-labeled graphs.

Exploratory Search. Exploratory methods have been studied to query graphs [4, 18, 24, 26, 30]. It involves an interactive process of exploration and refinement as knowledge is progressively acquired. Visual interfaces are developed to support interactive KWS in XML [4], with a focus on query suggestion with mismatches [30]. TriniT [26] facilitates exploratory querying in incomplete knowledge graphs. It relaxes queries by replacing *e.g.,* its predicate to semantically similar ones using rules, and retrieves the relevant answers ranked with the weight of the rules. Query-by-example [12] refers to specified substructures of a graph as examples, and identifies relevant substructures using *e.g.,* (relaxed) subgraph isomorphism to approximate user's intent.

We develop efficient query expansion for established KWS queries and term relevance measures. Prior work does not provide efficiency guarantees for query expansion. Our diversified query expansion techniques can be readily adapted to enable data exploration with established KWS models, and can also be used in combination with existing data exploration schemes.

2 Keyword Query Expansion for Graphs

We consider a multi-labeled graph $G = (V, E)$ with a node set V and a set of weighted, undirected edges $E \subseteq V \times V$. Each node $v \in V$ is associated with a set of labels (terms) $L(v) \subseteq \Sigma$ that encodes its content, where Σ is a finite set of terms. Each edge $e = (u, v) \in E$ has a weight $w(e)$. In practice, a node v (resp. edge e) may represent a tuple [29] (resp. a dependency), an entity (resp. a fact) in a knowledge graph, or a location. The edge weight may characterize semantic closeness of entities [8], fact reliability, or spatial distance.

Keyword Search in Graphs. A keyword query Q is a set of terms $\{t_1, \dots t_n\}$. Given graph $G{=}(V, E)$ and a term t_i, a match function determines a set of *content nodes* $V(t_i) \subseteq V$ that match t_i. For example, it can be a transformation [27] that finds the nodes with labels that are synonyms of t_i.

Existing matching semantics are based on either (1) lowest common ancestor (LCA), which finds a set of content nodes with common ancestor that minimizes a certain distance measure (such as root-leaf distances [11,13] or total edge cost [9]), or (2) dense subgraphs [14], which induce content nodes with small aggregated pairwise distances. We consider a general form of query answer $Q(G)$ of a keyword query Q, which is a subgraph of G that contains at least one node from $V(t_i)$ for each $t_i \in Q$. We denote the set of content nodes in $Q(G)$ as V_C.

Keyword Query Expansion. Query expansions are useful to interpret queries, understand results, and suggest new queries for graph exploration [24]. Given a keyword query Q, we define an *expansion* of Q as a set of terms Q_T not in Q.

Effective graph exploration requires to find meaningful new terms to trigger future search sessions toward desired information in graphs. The quality of such terms should be defined *relative to* query Q and validated content nodes V_C in $Q(G)$. We next introduce two relative measures that characterize term *relevance* and *diversity*, respectively. These measures have their conventional counterparts in IR-based query expansion for unstructured texts and documents.

Term Relevance. First, a suggested term should be relevant to validated content nodes V_C. A common practice in graph search quantifies the relevance between two nodes as a function dist of their (normalized) distances [8,28,33]. Distance-based measures are useful in its capacity to capture semantic distance between concepts, which is usually represented by the path connecting two entities in G. The more "closer" the two nodes are, the more relevant they are.

Given a term t and a content node $v_c \in V_C$ that matches a term $t' \in Q$, the *distance* of t to node v_c is defined as

$$\mathsf{dist}(t, v_c) = \min_{v_t \in V(t) \setminus V_C} \mathsf{dist}(v_t, v_c)$$

where $\mathsf{dist}(v_t, v_c)$ is a distance function from v_t to v_c in graph G. For example, $\mathsf{dist}(v_t, v_c)$ can be defined as $\sum_{e \in \rho} w(e)$, for a shortest path ρ between v_t and v_c in G. Here, we only consider those content nodes of term t that are not in V_C, which count for relevance from the "unseen" nodes to be explored.

Following this intuition, a term that is relevant to V_C should have content nodes with small aggregated distances. Given content nodes V_C and term t, the *term relevance* $\mathsf{rev}(t, V_C)$ of t *w.r.t.* V_C is further defined as

$$\mathsf{rev}(t, V_C) = \frac{1}{1 + \sum_{v_c \in V_C} \mathsf{dist}(t, v_c)}; \quad \mathsf{rev}(Q_T, V_C) = \sum_{t \in Q_T} \mathsf{rev}(t, V_C)$$

That is, the relevance of a term *w.r.t.* V_C is the aggregated relevance from its content nodes that are closest to the nodes in V_C. The relevance of a query expansion $Q_T = \{t_1, \ldots, t_m\}$ *w.r.t.* V_C is further aggregated from the relevance of its terms. Note that both are relative measures, which are determined by specific query, validated answers, and the observed terms. For example, among Google, Developer, and Malaysia in Example 1, Google has the highest and Malaysia has the lowest relevance since as shown in Fig. 1, Google is closest to both validated content nodes and the Malaysia is far from and thus less relevant to them.

Remarks. We are aware of other Corpus-Based based relevance such as TF-IDF, node relevance [5], distinguishability [30], and surpriseness. Such terms can be readily integrated into $\mathsf{dist}(\cdot)$.

Term Diversity. The suggested terms should also be diversified to expand as many different aspects of answers $Q(G)$ as possible. Such diversity is preferred in *e.g.,* faceted search and result diversification [6,16,31].

We shall use the following notations. (1) We consider a reasonable *partition* $\mathcal{P}(\mathcal{V_C})$ of $V_C = \{V_{C_1}, \ldots, V_{C_n}\}$, where each set $V_{C_i} \subseteq V_C$ represents a group of content nodes with similar contents that indicates an "aspect" of V_C. (2) Given a threshold r and a term t, the *cover set* of t *w.r.t.* r, denoted as $\mathsf{cov}(t, r)$, refers to the set $\{v | \mathsf{dist}(t, v) \leq r, v \in V_C\}$. Equally, $\mathsf{rev}(t, \{v\}) \geq \frac{1}{1+r}$.

We use the partition $\mathcal{P}(\mathcal{V_C})$ of V_C and cover set to characterize the diversity of a query expansion Q_T. Let $\mathcal{P}(\mathcal{V_C})$ contains n groups of content nodes. The diversity of Q_T *w.r.t.* $\mathcal{P}(\mathcal{V_C})$ and threshold r is defined as

$$\mathsf{div}(Q_T, \mathcal{P}(\mathcal{V_C}), r) = \frac{1}{n} \sum_{i=1}^{n} \sqrt{\sum_{t \in Q_T} \frac{|\mathsf{cov}(t, r) \cap V_{C_i}|}{|V_{C_i}|}}$$

Intuitively, the function $\mathsf{div}(\cdot)$ rewards the diversity in that there is more benefit to expand a query by a term that can cover content nodes from more "aspects", than its counterparts that cover fewer nodes and groups which are already covered by other terms. In practice, (1) $\mathcal{P}(\mathcal{V_C})$ can be constructed by grouping content nodes with the terms in Q they match, by node types, or by applying semantic clustering [32]; (2) the threshold r is tunable by users: larger r allows terms that are "further" from V_C to be considered for expansion.

Example 2. Consider a partition of content nodes V_C (Fig. 1) as two sets Country= $\{v_7\}$ and Position= $\{v_3\}$ based on their *node type*, and set $r = 1$. For term Winter (resp. Google), $\mathsf{cov}(\mathsf{Winter}, 1) = \{v_3\}$ (resp. $\mathsf{cov}(\mathsf{Google}, 1) = \{v_3, v_7\}$). Then, $\mathsf{div}(\{\mathsf{Summer}, \mathsf{Winter}\}) = 0.5(\sqrt{2})$; and $\mathsf{div}(\{\mathsf{Summer}, \mathsf{Google}\}) = 0.5(1+\sqrt{2})$, respectively. This indicates that $Q_T = \{\mathsf{Summer}, \mathsf{Google}\}$ is more diversified. Indeed, term Google has a larger reward if combined with Winter, since it increases the diversity of Q_T by covering the partition Country not being covered before.

Diversification. Given a query expansion Q_T, a partition $\mathcal{P}(\mathcal{V_C})$ of V_C and threshold r, we model the quality $F(Q_T)$ of Q_T as a bi-criterion objective function that combines relevance and diversity, which is defined as

$$F(Q_T, \mathcal{P}(\mathcal{V_C}), r) = \mathsf{rev}(Q_T, V_C) + \lambda * \mathsf{div}(Q_T, \mathcal{P}(\mathcal{V_C}), r)$$

When it is clear from the context, we refer to the quality function as $F(Q_T)$. We now formulate our diversified query expansion problem.

Problem Statement. Given graph G, keyword query Q, a distance threshold r and a set of validated content nodes V_C, the diversified query expansion problem is to find a set of query expansion Q_T^* with k terms, such that (1) for each node $v_c \in V_C$, there is a term $t \in Q_T^*$, such that $v_c \in \mathsf{cov}(t, r)$; and (2) Q_T^* satisfies

$$Q_T^* = \mathop{\arg\max}_{Q_T \subseteq \Sigma, |Q_T| = k} F(Q_T, \mathcal{P}(\mathcal{V}_C), r)$$

For query expansion, we define the *marginal gain* $\mathsf{mg}(t, Q_T)$ of a term t to query expansion Q_T ($t \notin Q_T$) as $F(Q_T \cup \{t\}) - F(Q_T)$. The result below shows that function $F(\cdot)$ is well defined in terms of submodularity, which is widely used to characterize the quality of sets in Web search and database applications.

Lemma 1. *The function $F(\cdot)$ is a monotone submodular function for query expansion. That is, for any two query expansion Q_T and Q'_T, if $Q_T \subseteq Q'_T$, then (1) $F(Q_T) \leq F(Q'_T)$, and (2) for any term $t \notin Q'_T$, $\mathsf{mg}(t, Q'_T) \leq \mathsf{mg}(t, Q_T)$.*

It is easy to verify Lemma 1 by definition of monotone submodularity and that the diversity reward function is defined in terms of square root function. Due to space limit, we omit the detailed proof.

The problem is NP-hard by reduction from set diversification. In practice, V_C is relatively small, while the size of terms and their content nodes are large and disk-based. This calls for fast algorithms to support query expansion for interactive data exploration, and time response for term insertions to G.

3 Diversified Query Expansion

Enumerating combinations of k terms to find Q_T^* is clearly not practical. We start with an alternative that uses an "explore-and-diversify" process. We then introduce a fast approximation algorithm that is based on stream diversification.

Explore-and-Diversify. The basic intuition is to first find all relevant terms T *w.r.t.* V_C, and diversify T to find Q_T. The algorithm capitalizes on data locality, and visits only the neighbors of the nodes up to a bounded distance.

The algorithm, denoted as Div, invokes an iterator $\mathsf{SSSP}(v)$ that performs a bounded shortest path traversal up to $N^r(v)$ (*i.e.,* the nodes with distance up to r) from v, for each content node $v \in V_C$. This conceptually constructs a node list ranked by their distance to v, following a sorted access to $N^r(v)$.

(1) For each content node $v \in V_C$, iterator $\mathsf{SSSP}(v).\mathsf{next}()$ outputs a pair $(v', \mathsf{dist}(v, v'))$, which finds the next node $v' \in N^r(v)$ following an increasing order of $\mathsf{dist}(v, v')$. For each term $t \in L(v')$, it computes term relevance $\mathsf{rev}(t, V_C)$, and $\mathsf{cov}(t, r) \cap V_C$. These are bookkept in a matrix dmap ($\mathsf{dmap}[t][v] = \mathsf{dist}(t, v)$).

(2) When no new term can be found, it collects the set of all the terms T, and invokes a standard greedy algorithm [10] to find top-k diversified terms in T. While $|Q_T| < k$, the process iteratively selects a term $t \in T \setminus Q_T$, by consulting dmap, such that t has a maximized marginal gain $\mathsf{mg}(t, Q_T)$. It then updates Q_T from iteration $i-1$ (denoted as Q_T^{i-1}) as

$$Q_T^i = Q_T^{i-1} \cup \arg\max_{t \in T} \mathsf{mg}(t, Q_T^{i-1})$$

at iteration i. It is known that this strategy provides a (1-$\frac{1}{e}$)-approximation.

The algorithm Div requires the discovery of all the terms. The greedy algorithm requires k passes over the term set T. Furthermore, it already takes $O(|V_C|(|N^r(V_C)| \log |N^r(V_C)| + |N^r(V_C)|)$ time to traverse the graph, thus it is expensive to recompute Q_T when new terms are inserted to G.

3.1 Stream-Based Diversification

We can do better by capitalizing on stream-based computation [3]. Our major idea is to treat terms as a weighted stream, and process each term at most once to find Q_T with optimality guarantee, and incur small update cost for each term.

Overview. Our stream-based algorithm, denoted as streamDiv, takes as input the graph G, query Q, partitioned content nodes $\mathcal{P}(\mathcal{V_C})$, integer k, threshold r. The algorithm interleaves two procedures: (1) *term generation*: a procedure GenTerm that follows a Threshold Algorithm (TA) style process to generate a stream $\mathcal{T}$ of terms with operator SSSP, and (2) *stream diversification*: a procedure DivTerm that dynamically constructs and maintains Q_T from $\mathcal{T}$. Each time GenTerm generates a term t, it adds t to stream $\mathcal{T}$. Upon receiving a new term t in $\mathcal{T}$, DivTerm updates Q_T, until no new terms can be added to $\mathcal{T}$.

We say query expansion Q_T is an α-approximation ($\alpha \in [0, 1]$) of the optimal Q_T^* if $F(Q_T, \mathcal{P}(\mathcal{V_C}), r) \geq \alpha * F(Q_T^*, \mathcal{P}(\mathcal{V_C}), r)$. We present our major result below.

Theorem 1. *Given a constant $\epsilon > 0$, algorithm* streamDiv

- *computes a $(\frac{1}{2} - \epsilon)$-approximation Q_T in $O(|V_C|(|N^r(V_C)| \log |N^r(V_C)| + |N^r(V_C)|)(\frac{\log k}{\epsilon}))$ time;*
- *performs a single pass of all terms in $N^r(V_C)$, and takes $O(k|V_C| + \frac{\log k}{\epsilon})$ time to process each term; and*
- *takes $O(k|V_C| + \frac{\log k}{\epsilon})$ time to update Q_T upon each term insertion to G.*

As a proof of Theorem 1, we introduce details of algorithm streamDiv. For the ease of presentation, we first present term stream diversification procedure DivTerm, with the performance guarantees in Theorem 1. We then show how procedure GenTerm optimizes the stream by only generating promising terms.

```
Procedure DivTerm($\mathcal{T}, \mathcal{P}(\mathcal{V}_\mathcal{C}), r, Q_T, F_{max}$);
1.  if sieve set $O := \emptyset$ then
2.    $O := \{(1+\epsilon)^i | F_{max} \le (1+\epsilon)^i \le k.F_{max}\}$;
3.  for each value $s \in O$ and each term $t \in \mathcal{T}$ do
4.    if $Q_T(s)$ is not defined then $Q_T(s) := \emptyset$;
5.    if $\mathsf{mg}(t, Q_T(s)) \ge \frac{\frac{s}{2} - F(Q_T(s), \mathcal{P}(\mathcal{V}_\mathcal{C}), r)}{k - |Q_T(s)|}$ and $|Q_T(s)| < k$ then
6.      $Q_T(s) := Q_T(s) \cup \{t\}$; update minsieve;
7.  if $\forall s \in O$: $|Q_T(s)| = k$ or Condition 2 then
8.    $Q_T := \arg\max_{Q_T(s)} F(Q_T(s), \mathcal{P}(\mathcal{V}_\mathcal{C}), r)$;
9.    VoteforHalt(); return $Q_T$; /* early termination */
```

Fig. 2. Procedure DivTerm

3.2 Term Diversification

Upon receiving a stream $\mathcal{T}$ of elements (from procedure GenTerm) and their relevance, procedure DivTerm constructs Q_T by dynamically diversifying the elements from $\mathcal{T}$. We first review sieve-streaming for diversification.

Sieve-Streaming Revisited [3]**.** Given a monotone submodular function $F(\cdot)$, a constant $\epsilon > 0$, and the set of elements $\mathcal{D}$, sieve-streaming finds top-k elements $\mathcal{S}$ that maximizes $F(\mathcal{S})$ as follows. If $F_{max} = \max_{e \in \mathcal{D}} F(\{e\})$ is known, it determines a set of sieve values $(1+\epsilon)^i$ (i as integer) to discretizes the range $[F_{max}, k.F_{max}]$, and use the sieve values to approximate $F(\mathcal{S}^*)$. For each sieve value s, a corresponding sieve list S_v is dynamically maintained as the top-k terms with marginal gain at least $(\frac{s}{2} - F(S_v))/(k - |S_v|)$, falling into the specific range determined by s. It is shown that selecting $\mathcal{S}$ as the top-k elements from all the sieve sets generates a $(\frac{1}{2} - \epsilon)$-approximation [3].

Overview. The procedure DivTerm, illustrated in Fig. 2, solves a submodular optimization problem over the term stream $\mathcal{T}$, following sieve-streaming. It initializes a set of sieve values O determined by maximized value $F_{max} = \arg\max_{t \in \mathcal{T}} F(t)$ (line 2). Here, $F(t) = \mathsf{rev}(t, V_C) + \lambda * \mathsf{div}(t, \mathcal{P}(\mathcal{V}_\mathcal{C}), r)$. DivTerm is activated upon seeing the maximum value F_{max} (see "Lazy sieving"). For each sieve value s, it maintains a list of top-k terms $Q_T(s)$ (lines 3–4). (1) For each term t arrived in $\mathcal{T}$, it dynamically inserts t into a sieve list $Q_T(s)$ if

$$\mathsf{mg}(t, Q_T(s)) \ge (\frac{s}{2} - F(Q_T(s), \mathcal{P}(\mathcal{V}_\mathcal{C}), r))/(k - |Q_T(s)|)$$

(2) Once there is no new term in stream $\mathcal{T}$, or all the sieve lists $Q_T(s)$ have size k, it terminates and constructs Q_T as the top-k terms from sieve lists (lines 7–9).

Analysis. Following [3], DivTerm is a $(\frac{1}{2} - \epsilon)$-approximation. (1) There exists a sieve value $s = (1+\epsilon)^i \in [F_{max}, 2k * F_{max}]$ that is closest to $F(Q_T^*)$, say,

$(1-2\epsilon)F(Q_T^*) \leq s \leq F(Q_T^*)$; and (2) each set $Q_T(s)$ is a $(\frac{1}{2}-\epsilon)$ answer for an estimation of $F(Q_T^*)$ with sieve value s. Indeed, if $\mathsf{mg}(t, Q_T)$ satisfies the condition in DivTerm (line 5), then $F(Q_T(s), \mathcal{P}(\mathcal{V}_\mathcal{C}), r) \geq \frac{s|Q_T(s)|}{2k} = \frac{s}{2}$ (when $|Q_T(s)|{=}k$). As there exists a value $s \in O$ that best estimates the optimal $F(\cdot)$, the top-k terms Q_T from sieve lists achieves approximation ratio $(\frac{1}{2}-\epsilon)$.

Optimization. The procedure DivTerm further optimizes sieve-streaming in [3] with two strategies, both interact with procedure GenTerm and makes use of an upper bound estimation of $F(t)$ for t in the "unseen" part of $\mathcal{T}$, denoted as $\overline{F}$.

Lazy Sieving. The first question is "how to find F_{max}?". When terms are inserted to $\mathcal{T}$ but F_{max} is unknown, sieve-streaming [3] eagerly maintains a dynamic number of lists for an increased value range $[F_{max}, 2k * F_{max}]$, determined by F_{max} in currently seen stream, to safely update results. Instead, we let procedure GenTerm "activates" the sieve streaming in DivTerm when it confirms the identification of F_{max}.

This is done by testing "*whether $\overline{F}$ is no larger than the smallest $F(t)$ for t in seen part of $\mathcal{T}$*" (**Condition 1**). When this happens, GenTerm sends F_{max} as the largest seen $F(t)$ to DivTerm, and activates sieve streaming. We found that such terms can be identified at quite early stage of streaming, thanks to the sorted access of $\mathsf{SSSP}(\cdot)$ and distance-based relevance measures.

Early Termination. Procedure DivTerm talks to GenTerm in turn to terminate the stream early. To this end, it keeps track of (1) the smallest seen threshold $\mathsf{minsieve} = \min_{s \in O}(\frac{s}{2} - F(Q_T(s), \mathcal{P}(\mathcal{V}_\mathcal{C}), r))/(k - |Q_T(s)|)$; and (2) an estimated upper bound of all unseen terms t and sieve values s, denoted as $\overline{\mathsf{mg}}(\mathcal{T})$, and tests "*whether* $\overline{\mathsf{mg}}(\mathcal{T}) < \mathsf{minsieve}$" (**Condition 2**). If so, it sends a signal (VoteforHalt(), line 9) back to GenTerm to stop the term stream. Indeed, this indicates that no new term can be inserted to any sieve list and can make contribution to Q_T.

The following result shows that $\overline{\mathsf{mg}}$ can be set as $\overline{F}$ for early termination.

Lemma 2. *$\overline{F}$ is a valid upper bound of* $\mathsf{mg}(t)$ *for any unseen term t in $\mathcal{T}$.*

Proof sketch: For any term t in unseen part of $\mathcal{T}$ and any sieve value $s \in O$, we can verify that $\mathsf{mg}(t, Q_T(s)) \leq \mathsf{rev}(\{t\}, V_C) + \lambda \mathsf{div}(\{t\}, \mathcal{P}(\mathcal{V}_\mathcal{C}), r) = F(\{t\}, \mathcal{P}(\mathcal{V}_\mathcal{C}), r)$. As $\overline{F}$ is no less than the largest $F(\{t\}, \mathcal{P}(\mathcal{V}_\mathcal{C}), r)$ for all unseen term t in $\mathcal{T}$, Lemma 2 follows.

Condition 1 and **Condition 2** verify that we can "activate" and "terminate" the stream diversification, by using only a simple upper bound estimation $\overline{F}$. The optimization is quite effective: they improve the efficiency of query expansion by 81%, as verified by our experimental study.

We next introduce procedure GenTerm, and show how to compute $\overline{F}$.

```
Procedure GenTerm
Input: Graph $G$, query $Q$, partition $\mathcal{P}(\mathcal{V_C})$, integer $k$, threshold $r$;
Output: a stream of terms $\mathcal{T}$.
1.    stream $\mathcal{T} := \emptyset$, $F_{max} := 0$; $F_{min}$:=0; $\overline{F} := +\infty$;
      /* activating SSSP traversal */
2.    for each $v_i \in V_C$ do create iterator SSSP($v_i$);
      /* TA-style term generation */
3.    while there exists $v_i \in V_C$ : SSSP($v_i$).peekDist() $\leq r$ do
4.      pair $<v', d>$ := SSSP($v_i$).next();
5.      for each term $t \in L(v')$ do
6.        if $\mathcal{T}$ contains $t$ then continue ; /*$t$ is already seen*/
7.        $\mathcal{T} := \mathcal{T} \cup \{t\}$; update rev($\cdot$), div($\cdot$) and $F(\cdot)$ for term $t$;
8.        $F_{max} = \max_{t \in \mathcal{T}} F(\{t\}, \mathcal{P}(\mathcal{V_C}), r)$;
9.        $F_{min} = \min_{t \in \mathcal{T}} F(\{t\}, \mathcal{P}(\mathcal{V_C}), r)$;
10.       estimate $\overline{F}$; /* estimate the best $F(\cdot)$ of unseen terms */
11.       if $\overline{F} \leq F_{min}$ (Condition 1) then
12.         DivTerm($\mathcal{T}, \mathcal{P}(\mathcal{V_C}), r, F_{max}$); /* "activate" stream */
13.       else if DivTerm is active then
14.         DivTerm($\{t\}, \mathcal{P}(\mathcal{V_C}), r, F_{max}$); /* process a new term */
```

Fig. 3. Procedure GenTerm

3.3 Term Stream Generation

Given graph G and partitioned content nodes $\mathcal{P}(\mathcal{V_C})$, procedure GenTerm generates a stream of promising terms that are likely to contribute to Q_T. The procedure follows a Threshold-Algorithm style processing.

Overview. The procedure GenTerm, illustrated in Fig. 3, invokes iterator SSSP as in algorithm Div for each content node $v \in V_C$ (line 2). The bounded traversal is controlled by operator peekDist, which reports the distance of the next node in $N^r(v)$ by a sorted access to $N^r(v)$ following non-decreasing order of distances to v. This operator can be easily implemented using standard single source shortest path algorithm *e.g.,* Dijkstra [5]. It then verifies new terms to be added to stream $\mathcal{T}$, and updates their relevance and diversity by leveraging an index such as pruned landmark labeling [2] (lines 5–7). Note that it never reprocesses a term t once it has been added to $\mathcal{T}$ earlier, as all the term information has been captured due to the visiting of its "closer" content node (line 6). It then estimates $F(\cdot)$ for unseen terms, and test whether F_{max} is identified to activate Div (line 12). While DivTerm is active, it sends a new term for processing (line 14).

Estimation of $\overline{F}$. To compute $\overline{F}$ (line 10), GenTerm estimates an upper bound of rev($\{t\}, V_C$) and div($\{t\}, \mathcal{P}(\mathcal{V_C}), r$), respectively. For rev($\{t\}, V_C$) by consulting the current status of SSSP iterators, GenTerm estimates the potential closest distance for any unseen t, and updates rev($\{t\}, V_C$) $=$ $(1 +$

$\sum_{v_i \in V_C} \mathsf{SSSP}_i.\mathsf{peekDist})^{-1}$. For $\mathsf{div}(\{t\}, \mathcal{P}(\mathcal{V}_\mathcal{C}), r)$, given each cluster $\mathcal{P}(\mathcal{V}_\mathcal{C})$, it estimates an upperbound for coverage of any unseen t as $|\{v_i \in \mathcal{P}(\mathcal{V}_\mathcal{C}) | \mathsf{SSSP}_i.\mathsf{peekDist} \leq r\}|$.

Performance Analysis. Procedure GenTerm executes $|V_C|$ number of SSSP originated from $v \in V_C$ bounded by neighbors $N^r(v)$ up to distance r that each one takes $O(|N^r(v)| \log |N^r(v)| + |N^r(v)|)$. As each node carries a small constant number of terms, time cost of updating term information of a node is in $O(1)$. Each time a term is sent to DivTerm, it takes $O(\frac{\log k}{\epsilon})$ time to update the sets $Q_T(s)$'s. Thus, the total time is in $O(|V_C|(|N^r(V_C)| \log |N^r(V_C)| + |N^r(V_C)|)(\frac{\log k}{\epsilon}))$.

Coping with Term Insertion. Given a newly added term t to a node in G, streamDiv simply takes $O(k|V_C|)$ time to compute $F(t, \mathcal{P}(\mathcal{V}_\mathcal{C}), r)$ by (1) update $\mathsf{dist}(t, v)$ for $v \in V_C$, using distance index [2]; and (2) update $\mathsf{cov}(t, r) \cap V_C$. It then invokes DivTerm only if $F(t, \mathcal{P}(\mathcal{V}_\mathcal{C}), r) \geq \mathsf{minsieve}$, and at most $O(\frac{\log k}{\epsilon})$ time to update sieve lists and update Q_T.

4 Experimental Evaluation

Using real-life graphs, we evaluate (1) the effectiveness and efficiency of our diversified query expansion algorithm, (2) the impact of factors, and (3) case studies to verify its application in knowledge base exploration.

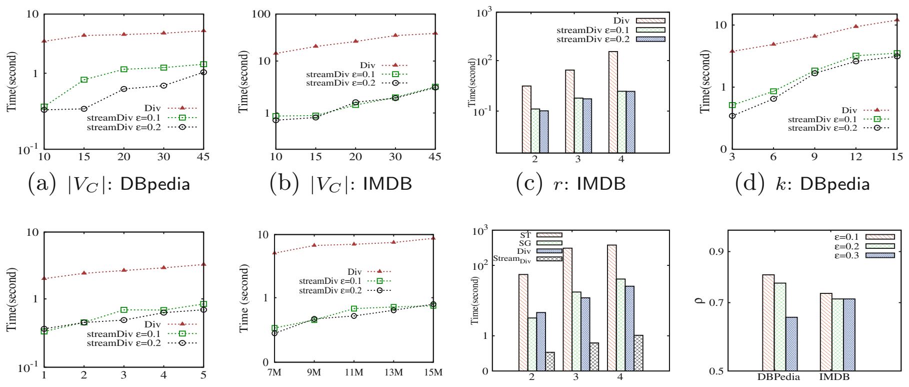
Experimental Setting. We use the following setting.

Datasets. We use the following datasets. (1) DBpedia[1], a knowledge graph that contains 4.8M multi-labeled entities, in total 1.5M terms from both entity names and their type (*e.g.,* "Obama", "Place"), and 15M edges with 670 distinct relationships. Each node carries on average 11 terms. (2) IMDB[2] is an information network including 1.6M entities of movies, TV shows, and crews. It contains 5.1M edges and 1.4M tokens from *e.g.,* genre and titles of movies, and the name of crews. Each node has on average 3 terms. We assign edge weights following backward KWS in graph databases [13], and set $\mathsf{dist}(\cdot)$ as the distance of shortest path between two nodes.

Partitioning. We created partitions of V_C grouped by the pair (t, type), where each group has the same type (*e.g.,*"Person", "Genre") and matched with the same term in query Q (See example 2).

KWS *queries.* The query generation is controlled by the size of Q ($|Q|$) and threshold r. We sample the queries using a random walk with restart [21]. To construct Q, we start from a random origin v in G and perform random walk to

[1] dbpedia.org.
[2] http://www.imdb.com.



(a) $|V_C|$: DBpedia (b) $|V_C|$: IMDB (c) r: IMDB (d) k: DBpedia

(e) $|L|$: DBpedia (f) $|G|$: DBpedia (g) $|Q|$: DBpedia (h) ρ: Quality

Fig. 4. Efficiency of query expansion

visit $N^r(v)$ multiple times. We construct Q with top keywords that have highest TF-IDF score in its neighbors, to ensure the existence of reasonable answers. We adopt established Bidirectional Search [13] for DR queries; GST-k [9] which finds top answers for ST queries; and r-cliques [14] for SG queries.

Algorithms. We implement the "explore and diversify" algorithm Div and stream-based algorithm streamDiv. We also implemented three term suggestion algorithms applicable for KWS in graphs. (1) CoOcc [20] suggests new terms to Q that are most frequently co-occurred in its answer $Q(G)$. (2) TagCloud [15] finds "search entities" as star graphs induced by a center node and its neighbors. It returns top keywords determined by TagCloud score, computed by TF – IDF over search entities relevant to Q. (3) QBE [12] takes as input query tuples (keywords) and induces a query graph with the neighborhood of the answers to find relevant triples with similar edge type. We take triples from $Q(G)$ as "examples" to QBE. The distance index of IMDB and DBpedia, constructed by pruned landmark labeling [2], takes 1.16 GB and 4.34 GB, respectively.

All the tests are conducted on a machine with an Intel 2.3 GHz processor with 64 GB memory. Each test is repeated 5 times and the average is reported. We set $|Q| = 3$, $r = 3$, $|V_C| = 30$, $\epsilon = 0.1$, and $k = 5$, unless otherwise specified.

Exp-1: Efficiency of Query Expansion. We first evaluate the efficiency of streamDiv, compared with two variants of Div with $\epsilon \in \{0.1, 0.2\}$. We use 50 queries, and evaluate the average response time per query. We evaluate the impact of 5 factors: $|V_C|$, r, k, $|L|$ (the number of terms per node), and $|G|$.

Varying $|V_C|$. Fixing other parameters by default, we varied $|V_C|$ from 10 to 45. Figure 4(a) (resp. Fig. 4(b)) shows the time required for query expansion over DBpedia for DR (resp. IMDB for SG) queries. The results tell us the following. (1) All the algorithms take more time as $|V_C|$ increases, due to that more terms are processed. On the other hand, streamDiv outperforms Div by 17 and 3 times over IMDB and DBpedia, respectively. (2) streamDiv takes longer time for smaller ϵ, due to higher sieve lists maintenance cost. The impact of ϵ is higher over DBpedia than IMDB. Indeed, streamDiv pays more overhead from stream generation and sieve list maintenance in trade for quality over graphs with more labels per node.

Varying r. We evaluate the impact of threshold r over IMDB and DR queries. Figure 4(c) verifies that query expansion takes more time as the bound r increases, since the algorithms explore more parts of the graph.

Varying k. Figure 4(d) verifies the impact of number k. Varying k from 3 to 15 over DBpedia, streamDiv takes more time over larger k. The increased cost is due to that the stream takes longer to meet the early termination criteria when larger number of terms are required. On the other hand, streamDiv visits each term at most once, and is less sensitive to the change of k compared with Div.

Varying $|L|$. Varying the number of associated terms to each node in DBpedia, we investigate the impact of $|L|$ on performance of the query expansion. Figure 4(e) verifies that the more number of labels a node carries, the more time is needed for query expansion. While Div needs to collect all terms, streamDiv visits each term once, and never process a term that has been seen earlier in the stream.

Varying $|G|$. We sample 5 versions of DBpedia with edge size varied from $7M$ to $15M$. As shown in Fig. 4(f), the query expansion algorithms scale well with larger $|G|$ and streamDiv algorithms consistently outperform Div.

Session Response Time. We also report the response time of query expansion in practical graph exploration sessions. To this end, we implemented an interface that allows end users to explore real-life graphs with streamDiv. We let end users to start with queries with $|Q| = 2$, and execute multiple search sessions. Each session allows users to validate V_C, run our query expansion algorithm, and review results via query evaluation. We compare the query evaluation time (KWS) and expansion time per session, as two major bottlenecks for data exploration.

Figure 4(g) reports the results for ST queries over DBpedia. (1) Both KWS and streamDiv take more time as queries become larger, due to higher exploration cost. (2) The query expansion takes up to 4% of KWS evaluation for ST queries; similarly for other query classes we evaluated (not shown). These results indicate that streamDiv makes query expansion no longer a major bottleneck.

Exp-2: Quality of Query Expansion. We define a relative measure of quality for streamDiv *w.r.t.* Div, denoted as ρ. Given n queries in $\mathcal{Q}$, we define $\rho = \frac{\sum_{Q \in \mathcal{Q}} \frac{F(Q_T)}{F(Q'_T)}}{n}$ where Q_T and Q'_T are the set of expanded terms returned by streamDiv and Div, respectively. For Div, ρ is constantly 1.

Figure 4(h) shows that the quality of diversified terms is tunable by ϵ. When ϵ is changed from 0.1 to 0.3, the quality of terms from streamDiv is in general

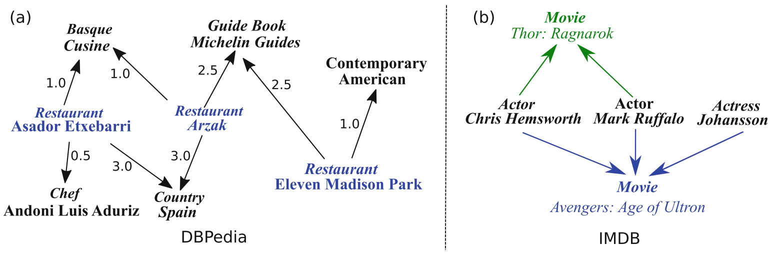


Fig. 5. Knowledge graph exploration with KWS (Color figure online)

not degrading too much *w.r.t.* Div (especially over IMDB). On the other hand, streamDiv becomes 15.48 and 2.83 times faster over IMDB and DBpedia, respectively. This verifies the tunable performance of streamDiv, which is desirable for data exploration under different time constraints.

Exp-3: Case Study. We next verify the applications of streamDiv.

Tunable Knowledge Base Exploration. We report a case in which an end user starts with a query $Q = \{\mathsf{Asador}, \mathsf{Arzak}, \mathsf{Madison\ Park}\}$ to find famous restaurants. The answer (Fig. 5(a), in blue) is partitioned by their location (Spain and U.S).

(1) Setting $\lambda = 0.1$ and $k = 2$, streamDiv suggests top-2 terms $\{\mathsf{Basque}, \mathsf{Chef}\}$. While both are relevant, they state facts only about "Spain restaurants". The terms recommend a popular local Spanish dish and chefs who are known for it.
(2) The user enlarges λ to 0.6, and streamDiv identifies a set of more diversified terms $\{\mathsf{Basque}, \mathsf{American}\}$. Indeed, one suggests facts about US restaurants by recommending contemporary American food, and the other suggests local Spanish dish served by two Spanish restaurants.

Given the same query, QBE suggests other restaurants that are reviewed by Trade Magazine, CoOcc suggests a term $\{\mathsf{Restaurant}\}$ which is not informative, and TagCloud suggests $\{\mathsf{Bar}\}$ with content nodes less relevant to the restaurants.

"why-not" Queries. streamDiv can also be used to answer "why-not" questions. Given a query Q and answer $Q(G)$, "why-not" question aims to find a new query that contains the validated nodes in $Q(G)$, and has answers relevant to a set of "why-not" entities. We asked our end users to mark "why-not" entities in their exploration sessions, and show one such case in Fig. 5(b).

Our end user posed a query $Q = \{\mathsf{Johansson}, \mathsf{Ruffalo}\}$, searching for the actors played in the movie "*Thor: Ragnarok*" she can't recall. The result (marked in blue) nevertheless misses this movie. She wonders why the entity $\{\mathsf{Thor\colon Ragnarok}\}$ is not in $Q(G)$, and adds it to V_C. The change of V_C now triggers streamDiv to explore another actor Hemsworth who played in "Thor: Ragnarok" with actor "Ruffalo", and suggests term "Hemsworth". Both the term and the entity can be suggested to user that "explains" the missing match.

5 Conclusion and Future Work

We have studied diversified query expansion for KWS in graphs. Our model maximizes the aggregate relevance to a set of validated content nodes with diversity guarantees over partitioned content nodes. We have developed a stream-style ($\frac{1}{2}$-ϵ) approximation that updates diversified terms with small update cost. We also showed that it readily copes with new insertion of terms. Our experimental results have verified the efficiency and effectiveness of diversified query expansion algorithm, and its applications in knowledge base exploration. Future topics include comparing with more graph exploration methods as well as investigating more relevancy measures and partitioning strategies.

Acknowledgment. Namaki and Wu are supported in part by NSF IIS-1633629 and Huawei Innovation Research Program (HIRP).

References

1. Achiezra, H., Golenberg, K., Kimelfeld, B., Sagiv, Y.: Exploratory keyword search on data graphs. In: SIGMOD, pp. 1163–1166 (2010)
2. Akiba, T., Iwata, Y., Yoshida, Y.: Fast exact shortest-path distance queries on large networks. In: SIGMOD, pp. 349–360 (2013)
3. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: massive data summarization on the fly. In: SIGKDD, pp. 671–680 (2014)
4. Bao, Z., Zeng, Y., Jagadish, H., Ling, T.W.: Exploratory keyword search with interactive input. In: SIGMOD, pp. 871–876 (2015)
5. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using BANKS. In: ICDE, pp. 431–440 (2002)
6. Bouchoucha, A., He, J., Nie, J.-Y.: Diversified query expansion using conceptnet. In: CIKM, pp. 1861–1864 (2013)
7. Carpineto, C., Romano, G.: A survey of automatic query expansion in information retrieval. CSUR **44**, 1 (2012)
8. De Nies, T., Beecks, C., Godin, F., De Neve, W., Stepien, G., Arndt, D., De Vocht, L., Verborgh, R., Seidl, T., Mannens, E., et al.: A distance-based approach for semantic dissimilarity in knowledge graphs. In: ICSC, pp. 254–257 (2016)
9. Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X., Lin, X.: Finding top-k min-cost connected trees in databases. In: ICDE, pp. 836–845 (2007)
10. Gollapudi, S., Sharma, A.: An axiomatic approach for result diversification. In: WWW, pp. 381–390 (2009)
11. He, H., Wang, H., Yang, J., Yu, P.S.: BLINKS: ranked keyword searches on graphs. In: SIGMOD, pp. 305–316 (2007)
12. Jayaram, N., Khan, A., Li, C., Yan, X., Elmasri, R.: Querying knowledge graphs by example entity tuples. TKDE **27**, 2797–2811 (2015)
13. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB (2005)
14. Kargar, M., An, A.: Keyword search in graphs: finding r-cliques. VLDB **4**, 681–692 (2011)

15. Koutrika, G., Zadeh, Z.M., Garcia-Molina, H.: Data clouds: summarizing keyword search results over structured data. In: EDBT, pp. 391–402 (2009)
16. Ma, H., Lyu, M.R., King, I.: Diversifying query suggestion results. In: AAAI (2010)
17. Mishra, C., Koudas, N.: Interactive query refinement. In: EDBT (2009)
18. Mottin, D., Müller, E.: Graph exploration: from users to large graphs. In: PODS, pp. 1737–1740 (2017)
19. Namaki, M.H., Wu, Y., Zhang, X.: GExp: cost-aware graph exploration with keywords. In: SIGMOD (2018)
20. Tao, Y., Yu, J.X.: Finding frequent co-occurring terms in relational keyword search. In: EDBT, pp. 839–850 (2009)
21. Tong, H., Faloutsos, C., Pan, J.-Y.: Fast random walk with restart and its applications (2006)
22. Tran, Q.T., Chan, C.-Y.: How to ConQueR why-not questions. In: SIGMOD, pp. 15–26 (2010)
23. Tran, Q.T., Chan, C.-Y., Parthasarathy, S.: Query reverse engineering. VLDB **23**, 721–746 (2014)
24. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In: ICDE, pp. 405–416 (2009)
25. Wang, H., Aggarwal, C.C.: A survey of algorithms for keyword search on graph data. In: Aggarwal, C., Wang, H. (eds.) Managing and Mining Graph Data. ADBS, vol. 40, pp. 249–273. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-6045-0_8
26. Yahya, M., Berberich, K., Ramanath, M., Weikum, G.: Exploratory querying of extended knowledge graphs. VLDB **9**, 1521–1524 (2016)
27. Yang, S., Wu, Y., Sun, H., Yan, X.: Schemaless and structureless graph querying. PVLDB **7**(7), 565–576 (2014)
28. Yao, J., Cui, B., Hua, L., Huang, Y.: Keyword query reformulation on structured data. In: ICDE, pp. 953–964 (2012)
29. Yu, J.X., Qin, L., Chang, L.: Keyword search in relational databases: a survey. IEEE Data Eng. Bull. **33**, 67–78 (2010)
30. Zeng, Y., Bao, Z., Ling, T.W., Jagadish, H., Li, G.: Breaking out of the mismatch trap. In: ICDE, pp. 940–951 (2014)
31. Zheng, B., Zhang, W., Feng, X.F.B.: A survey of faceted search. J. Web Eng. **12**, 041–064 (2013)
32. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. VLDB **2**, 718–729 (2009)
33. Zhu, G., Iglesias, C.A.: Computing semantic similarity of concepts in knowledge graphs. TKDE **29**, 72–85 (2017)