View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Edinburgh Research Explorer

Diversified Top-k Graph Pattern Matching

Wenfei Fan[1,2] Xin Wang[1] Yinghui Wu[3]
[1]University of Edinburgh [2]RCBD and SKLSDE Lab, Beihang University [3]UC Santa Barbara
{wenfei@inf, x.wang-36@sms, y.wu-18@sms}.ed.ac.uk

ABSTRACT

Graph pattern matching has been widely used in *e.g.,* social data analysis. A number of matching algorithms have been developed that, given a graph pattern Q and a graph G, compute the set $M(Q,G)$ of matches of Q in G. However, these algorithms often return an excessive number of matches, and are expensive on large real-life social graphs. Moreover, in practice many social queries are to find matches of a specific pattern node, rather than the entire $M(Q,G)$.

This paper studies top-k graph pattern matching. (1) We revise graph pattern matching defined in terms of simulation, by supporting a designated output node u_o. Given G and Q, it is to find those nodes in $M(Q,G)$ that match u_o, instead of the large set $M(Q,G)$. (2) We study two classes of functions for ranking the matches: relevance functions $\delta_r()$ based on, *e.g.,* social impact, and distance functions $\delta_d()$ to cover diverse elements. (3) We develop two algorithms for computing top-k matches of u_o based on $\delta_r()$, with *the early termination property*, *i.e.,* they find top-k matches without computing the entire $M(Q,G)$. (4) We also study *diversified top-k matching*, a bi-criteria optimization problem based on both $\delta_r()$ and $\delta_d()$. We show that its decision problem is NP-complete. Nonetheless, we provide an approximation algorithm with *performance guarantees* and a heuristic one with *the early termination property*. (5) Using real-life and synthetic data, we experimentally verify that our (diversified) top-k matching algorithms are effective, and outperform traditional matching algorithms in efficiency.

1. INTRODUCTION

Graph pattern matching is being widely used in social network analysis [5, 32], among other things. A number of algorithms have been developed for graph pattern matching that, given a graph pattern Q and a graph G, compute $M(Q,G)$, the set of matches of Q in G (*e.g.,* [11, 18]).

Social data analysis, however, introduces new challenges to graph pattern matching. Social graphs are typically large, with millions of nodes and billions of edges. This gives rise to the following problems with the matching algorithms.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy.
Proceedings of the VLDB Endowment, Vol. 6, No. 13
Copyright 2013 VLDB Endowment 2150-8097/13/13... $ 10.00.

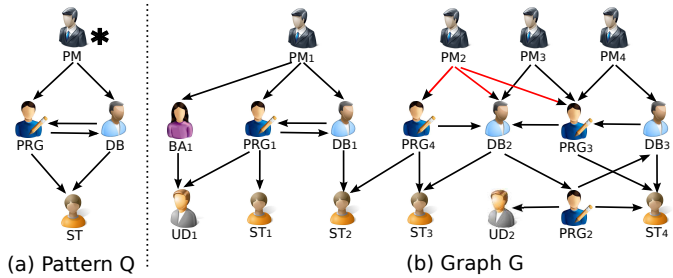


Figure 1: Querying collaboration network

(1) The matching algorithms often return an excessive number of results. Indeed, when matching is defined by subgraph isomorphism [14], $M(Q,G)$ may contain exponentially many subgraphs of G; when graph simulation is adopted, $M(Q,G)$ is a relation of size $O(|G||Q|)$ [18], which may be larger than graph G. It is a daunting task for the users to inspect such a large $M(Q,G)$ and find what they are searching for.

(2) The sheer size of social graphs makes matching costly: for matching defined by simulation, it takes $O(|G||Q|+|G|^2)$ time to compute $M(Q,G)$ [11]; for subgraph isomorphism, it is NP-complete to decide whether a match exists (cf. [29]).

(3) Social queries often need to find matches of a specific pattern (query) node u_o as "query focus" [3], *i.e.,* we just want those nodes in a social graph G that are matches of u_o in $M(Q,G)$, rather than the entire set $M(Q,G)$ of matches of Q. Indeed, this is how "graph search" (`http://www.facebook.com/about/graphsearch`) of Facebook is conducted on a big social graph with more than 1 billion users and 140 billion links(`http://newsroom.fb.com/`). The need for this is also evident in, *e.g.,* egocentric search [6] and expert recommendation [27, 31]. In fact, 15% of social queries are to find matches of specific pattern nodes [27].

These highlight the need for *top-k graph pattern matching*: given Q, G and a designated pattern node u_o, it is to find top-k matches of u_o in $M(Q,G)$, ranked by a quality function. The users only need to check k matches of u_o instead of $M(Q,G)$. Better still, if we have an algorithm with *the early termination property*, *i.e.,* it finds top-k matches of u_o *without* computing the entire $M(Q,G)$, we do not have to pay the price of full-fledged graph pattern matching.

Example 1: A fraction of a collaboration network is given as graph G in Fig. 1. Each node in G denotes a person, with attributes such as *job title*, *e.g.,* project manager (PM),

database developer (DB), programmer (PRG), business analyst (BA), user interface developer (UD) and software tester (ST). Each edge indicates a supervision relationship; *e.g.,* edge (PRG_1, ST_1) indicates that PRG_1 supervised ST_1.

A company issues a graph search query to find PMs who supervised both DBs and PRGs, and moreover, (1) the DB worked under the PRG directly or indirectly, and vice versa; and (2) both the DB and the PRG supervised an ST [21]. The requirements for the PMs are expressed as a graph pattern Q shown in Fig. 1 (a). Here PM is the "focus" of the query, *i.e.,* only the matches of PM are asked for [3]. This is indicated by labeling PM with '$*$' as the "output node" of Q.

When graph pattern matching is defined in terms of subgraph isomorphism [29], no match of Q can be identified in G. Indeed, it is too restrictive to define matches as isomorphic subgraphs [11]. Bisimulation [8] extends subgraph isomorphism with matching relations as equivalence relations, which is still unable to identify some sensible matches, *e.g.,* PM_1. Instead, we adopt simulation [18] with a *designated node*, extending graph search by supporting both matching relation and a specified "query focus". With graph simulation, $M(Q, G)$ is a binary relation on the pattern nodes in Q and their matches in G, including (PM, PM_i), (DB, DB_j), (PRG, PRG_i), (ST, ST_i) for $i \in [1, 4]$ and $j \in [1, 3]$.

Observe that $M(Q, G)$ contains most of the nodes in G as matches, which are excessive since, *e.g.,* the matches ST_i ($i \in [1, 4]$) for ST are not required. However, what the user wants are the PM matches. It is hence unnecessary and too costly to compute the entire large set $M(Q, G)$. Even for the output node PM, too many PM_i are returned ($i \in [1, 4]$).

We can do better with top-k matching. When $k = 2$, we find two top-ranked PM_i's that match PM, and return them in response to the request, instead of $M(Q, G)$. Better still, it is less costly. Indeed, while a naive algorithm for top-k matching is to first compute $M(Q, G)$ and then pick top-2 PM_i's, an algorithm with *the early termination property* identifies the PM_i's without computing the entire $M(Q, G)$.

To rank the matches PM_i's of PM, one may consider the following criteria. (1) Social impact [21]. Observe that PM_2 can reach more people than any other PM, *i.e.,* PM_2 has collaborated with more people. Thus PM_2 has stronger social impact. (2) Social diversity [2,35]. Consider match sets $\{\mathsf{PM}_1, \mathsf{PM}_2\}$ and $\{\mathsf{PM}_2, \mathsf{PM}_3\}$. While PM_2 and PM_3 worked with the same people, PM_1 and PM_2 are quite "dissimilar" since they covered *different* groups of people. Putting these together, $\{\mathsf{PM}_1, \mathsf{PM}_2\}$ makes a good candidate for top-2 matches in terms of both social impact and diversity. □

This example shows that top-k graph pattern matching may rectify the limitations of existing matching algorithms. To make practical use of it, however, several questions have to be answered. How can specific output nodes be incorporated into graph pattern matching? What quality and diversity functions should be used to rank the matches? What is the complexity of computing top-k matches based on one or both of the functions? How can we guarantee early termination by our algorithms for computing top-k matches?

Contributions. This paper answers these questions. We focus on graph pattern matching defined by graph simulation, which has proved useful in *social data analysis* [5, 32].

(1) We revise the traditional notion of graph pattern matching by designating an output node u_o (Section 2). Given Q and G, it is to compute $M_u(Q, G, u_o)$, the set of matches of u_o in G via Q. That is, the set of nodes in G that are in $M(Q, G)$ and moreover, match the output pattern node u_o.

(2) We study two classes of functions to rank matches of u_o (Section 3), namely, *relevance functions* $\delta_r()$ that measure the relevance of a match, and *distance functions* $\delta_d()$, which measure the "dissimilarity" of two matches. Based on both, we define a bi-criteria (balanced by a parameter λ) *diversification* function $F()$, which aims both to maximize social impact and to cover social elements as diverse as possible. To simplify the discussion, we focus on two simple relevance and distance functions. Nonetheless, we show that our techniques support a range of such functions used in practice.

(3) We investigate top-k graph pattern matching (Section 4). Given a graph pattern Q, a graph G and a positive integer k, it is to find k top-ranked matches of u_o based on the relevance function $\delta_r()$. We provide two algorithms for doing so, in $O(|G||Q| + |G|^2)$ time yet with *the early termination property* [10]. That is, they stop as soon as the top-k matches are found, without computing the entire $M(Q, G)$.

(4) We also study the diversified top-k graph pattern matching problem (Section 5). It is to find top-k matches of u_o based on the diversification function $F()$. We show that its decision version is NP-complete. The bi-criteria optimization problem is also hard to approximate, as suggested in [4] (see more in Section 5). Despite this, we develop an approximation algorithm in time $O(|G||Q| + |G|^2)$ with *approximation ratio* 2. We also give a heuristic algorithm in time $O(|Q||G| + |G|^2)$, with *the early termination property.*

(5) Using both real-life and synthetic data, we experimentally verify the efficiency and effectiveness of our methods (Section 6). We find that they effectively reduce excessive matches: when $k = 10$, our top-k matching methods only need to examine 40% - 45% of matches in $M(Q, G)$ on average, and our diversified top-k heuristic finds high-quality matches by inspecting 45% of the matches. Better still, our algorithms are more efficient than their traditional counterparts, improving the efficiency by 64% (resp. 48%) on average for acyclic (resp. cyclic) patterns. In addition, they scale well with $|Q|$, $|G|$ and k, and are not sensitive to the change of λ. These verify that our methods indeed remedy the limitations of traditional matching, to an extent.

These results yield a promising approach to querying social data. In the worst case, our (diversified) top-k matching algorithms have the same complexity $O(|G||Q| + |G|^2)$ as traditional matching algorithms, despite *the extra computation* introduced for ranking and diversifying matches. Better still, they have *the early termination property* and hence, perform better than the traditional algorithms in efficiency, as verified analytically and experimentally. All the proofs, algorithms and complexity analyses can be found in [1].

Related work. We categorize the related work as follows.

Top-k queries. There has been a host of work on top-k query answering for relational data, XML and graphs.

Relational databases. Top-k query answering is to retrieve top-k tuples from query result. Given a monotone scoring function and sorted lists, one for each attribute, Fagin's algorithm [9] reads attributes from the lists and constructs tuples with the attributes. It stops when k tuples are constructed from the top-ranked attributes that have been seen. It then performs random access to find missing scores. It is

optimal with high probability for some monotone scoring functions. Extending Fagin's algorithm, the threshold algorithm [10] is optimal for all monotone scoring functions, and allows early termination with approximate top-k matches. It reads all grades of a tuple once seen from the lists, and performs sorted access to tuples by predicting maximum possible grades in unseen tuples, until k tuples are found. Other top-k queries, *e.g.,* selection, join and Datalog queries, adapt and extend the methods of [9, 10] (see [19] for a survey).

We focus on top-k matching on graphs rather than relational tables. Moreover, while the prior work assumes monotone scoring functions and requires ranked lists to be provided as input, (1) we combine the query evaluation and result ranking in *a single process*, *without* requiring ranked lists as input, and (2) our relevance and diversification functions are more involved than monotone scoring functions. Nonetheless, our algorithms promise early termination, and return answers without computing the entire $M(Q, G)$.

XML and graph matching. Top-k queries have also been studied for XML, and for graph queries defined in terms of subgraph isomorphism. (1) XML keyword search (*e.g.,* [15]) is to find top-k subtrees of a document, given a set of keywords. Essentially, the prior work is to find top-ranked trees or connected subgraphs induced from a set of keywords, rather than to find matches for a general graph pattern. (2) Top-k XPath queries are to identify top matches for the nodes in a tree pattern, based on tree pattern matching. For example, [26] finds top-ranked matches for tree patterns in terms of keyword and document frequency. (3) Top-k subgraph matching is to find top-ranked subgraphs that are isomorphic to a graph pattern [14, 37, 38], ranked by, *e.g.,* the total node similarity scores [38].

Our work differs from the prior work in the following. (1) We study top-k queries defined by graph simulation [18], rather than subgraph (tree) isomorphism. Further, we consider matches of a single output node that are computed with early termination. (2) We support *result diversification*, which is not studied in the prior work mentioned above.

Result diversification. Result diversification is a bi-criteria optimization problem for balancing result relevance and diversity [4, 13], with applications in *e.g.,* social searching [2]. (1) General frameworks for query result diversification are introduced in [4, 13, 30]. A set of axioms for designing diversification systems is proposed in [13], to characterize reasonable diversification functions. A general framework for diversified top-k search is proposed in [30], which consists of three general functions that capture the termination conditions and search strategies. (2) Based on result diversification, Top-k diversity queries are to find k answers that maximize both the relevance and overall diversity, which have been studied for *e.g.,* keyword search [7, 17]. Generally speaking, the approaches to finding top-k diversified results consist of two steps: (1) a ranked list *w.r.t.* relevance score is computed; and (2) the list is re-ranked by combining diversity scores to find top-k diversified objects [30]. It is shown [13, 35] that query diversification is intractable. An $O(1 - \frac{1}{e})$ approximation is given in [17] for submodular relevance and diversity functions. Closer to our work is [4], which generalizes diversification function with a submodular weight function and a "supermodular" part of distance sum.

In contrast, (1) we study top-k diversified matches for a designated node in *graph pattern matching*. We are not aware of any prior work on this topic. (2) Our algorithms combine query evaluation and result ranking, with early termination, while the previous work assumes that the query result is *already known*, except [7] for keyword search. (3) While our diversification function is not submodular as assumed in [17] and moreover, it is nontrivial to approximate based on a recent result of [4] (see Section 5 for a detailed discussion), we provide a 2-approximation algorithm.

Pattern queries with output nodes. Several query languages allow one to specify a designated output node, notably twig queries on XML data [26]. Such nodes can also be specified with a "return" clause in XQuery [25], or a "select" clause in SPARQL. These languages are typically based on subgraph (tree) isomorphism. To reduce search effort, [33] proposes a "Seed-Finder" that identifies matches for certain pattern nodes. These nodes are, however, not specified by users. This work extends twig queries to graph pattern matching defined in terms of graph simulation, and provides algorithms for computing diversified top-k matches with early termination, which were not studied for XPath.

2. GRAPH PATTERN MATCHING

In this section, we first review data graphs, pattern graphs and graph simulation [18]. We then revise the traditional matching notion by designating an output node.

2.1 Data Graphs and Pattern Graphs

Data graphs. A *data graph* (or simply a graph) is a directed graph $G = (V, E, L)$, where (1) V is a finite set of nodes; (2) $E \subseteq V \times V$, in which (u, u') denotes an edge from node u to u'; and (3) L is a function such that for each node u in V, $L(u)$ is a label from an alphabet Σ. Intuitively, the node labels denote *e.g.,* keywords, social roles, ratings [11].

Pattern graphs. A *pattern graph* is a directed graph $Q = (V_p, E_p, f_v)$, where (1) V_p is the set of *query nodes*, (2) E_p is the set of *query edges*, and (3) f_v is a function defined on V_p such that for each node $u \in V_p$, $f_v(u)$ is a label in Σ.

Graph simulation [18]. A graph G *matches* a pattern Q if there exists a binary relation $S \subseteq V_p \times V$ such that (1) for each node $u \in V_p$, there exists a node $v \in V$ such that $(u, v) \in S$, referred to as a *match* of u; (2) for each pair $(u, v) \in S$, (a) $f_v(u) = L(v)$, and (b) for each edge (u, u') in E_p, there exists an edge (v, v') in G such that $(u', v') \in S$.

It is known that if G matches Q, then there exists a *unique maximum* relation $M(Q, G)$ [18]. If G does not match Q, $M(Q, G)$ is the empty set. This maximum relation $M(Q, G)$ is referred to as *the set of matches of Q in G*. The relation $M(Q, G)$ can be depicted as *the result graph of Q in G* [11].

Example 2: Example data graph G and pattern Q are given in Fig. 1. One may verify that G matches Q, with the unique, maximum match $M(Q, G)$ given in Example 1. The label $f_v(u)$ of a query node u specifies a search condition: a node v in G can match u only if $L(v) = f_v(u)$. □

Given G and Q, the traditional notion of graph pattern matching by simulation is to compute $M(Q, G)$. It is known that $M(Q, G)$ can be computed in $O((|V_p|+|V|)(|E_p|+|E|))$ time [11], where $|M(Q, G)|$ is bounded by $O(|V||V_p|)$ [18].

We denote $|V_p|+|E_p|$ as $|Q|$ (the size of the pattern graph), and $|V| + |E|$ as $|G|$ (the size of the data graph).

2.2 Graph Pattern Matching Revised

We extend a *pattern* to be $Q = (V_p, E_p, f_v, u_o)$, where u_o is a query node in V_p labeled with '$*$', referred to as the *output node* of Q, and V_p, E_p and f_v are the same as above.

Given a pattern Q and a graph G, we define the *matches* of Q in G to be $M_u(Q, G, u_o) = \{v \mid (u_o, v) \in M(Q, G)\}$, *i.e.,* the matches of the output node u_o in the unique maximum $M(Q, G)$. Here $M_u(Q, G, u_o) = \emptyset$ if G does not match Q.

Note that $M_u(Q, G, u_o)$ is smaller than $M(Q, G)$: its size is bounded by $|V|$ as opposed to $|V||V_p|$.

Example 3: Recall graph G and pattern Q from Example 1. The node PM is marked as the output node of Q. Then the set $M_u(Q, G, \mathsf{PM}) = \{\mathsf{PM}_i \mid i \in [1, 4]\}$, which consists of 4 nodes as opposed to 15 *node pairs* in $M(Q, G)$. □

Graph pattern matching can be readily extended to support the following: (1) patterns (resp. graphs) with multiple predicates (resp. attributes) on its nodes, *i.e.,* search conditions defined with multiple predicates; and (2) patterns with multiple output nodes. To simplify the discussion, we focus on a single designated output node in this paper, as commonly found in practice [27]. Nonetheless, the results of this work extend to patterns with multiple output nodes; the interested reader is invited to consult [1] for details.

3. RANKING PATTERN MATCHES

The result set $M_u(Q, G, u_o)$ could still be excessively large when G is large, while users are often only interested in the best k matches of the output node of Q [19]. This suggests that we define certain functions to rank the matches, and compute top-k matches for u_o based on the functions.

In this section we study two sets of ranking functions: *relevance functions* to measure the relevance of matches (Section 3.1), and *distance functions* to measure match diversity (Section 3.2). We then define a *diversification function*, which is a bi-criteria objective function combining relevance and diversity (Section 3.3). Based on these, we introduce two top-k graph pattern matching problems. To simplify the discussion, we start with a simple formulation of the ranking functions based on "social impact", and we generalize the problems to a variety of ranking functions (Section 3.4).

3.1 Relevant Matches

We start with a simple function to measure the *relevance* of the matches of u_o. It is based on a notion of *relevant sets*.

Relevant set. Given a match v of a query node u in Q, *the relevant set of* v w.r.t. u (denoted as $R_{(u,v)}$) includes all matches v' of u' for each descendant u' of u in Q, such that if u reaches u' via a path $(u, u_1, \ldots, u_n, u')$, then v reaches v' via $(v, v_1, \ldots, v_n, v')$, where $(u_i, v_i) \in M(Q, G)$ $(i \in [1, n])$.

That is, $R_{(u,v)}$ includes all matches v' to which v can reach via a path of matches. Following [11], one can verify the following, which shows that the relevant set is well-defined.

Lemma 1: *Given a pattern graph Q and a data graph G, if G matches Q, then for any match v of a query node u, there exists a unique, maximum relevant set $R_{(u,v)}$.* □

Relevance function. On a match v of u, we define the relevance function $\delta_r()$ in terms of the relevant set $R_{(u,v)}$:

$$\delta_r(u, v) = |R_{(u,v)}|.$$

That is, the relevance function favors those matches that can reach more other matches: for a match v_o of the output node u_o, the more matches v_o can reach, the bigger "impact" it may have, as observed in social network studies [20]. Thus, the matches with high $\delta_r()$ values are preferred for relevance.

Top-k matching problem. We now state the *top-k matching problem*, denoted by topKP. Given a graph G, a pattern Q with output node u_o, and a positive integer k, it is to find a subset $S \subseteq M_u(Q, G, u_o)$, such that $|S| = k$ and

$$\delta_r(S) = \operatorname*{arg\,max}_{S' \subseteq M_u(Q,G,u_o), |S'|=k} \sum_{v_i \in S'} \delta_r(u_o, v_i).$$

Abusing $\delta_r()$, we also use $\delta_r(S)$ to denote $\sum_{v_i \in S} \delta_r(u_o, v_i)$, referred to as *the relevance of S to u_o*.

That is, topKP is to identify a set of k matches of u_o that maximizes the total relevance to u_o. In other words, for all $S' \subseteq M_u(Q, G, u_o)$, if $|S'| = k$ then $\delta_r(S) \geq \delta_r(S')$.

Example 4: Recall G and Q from Fig. 1. The relevant sets of the matches in $M_u(Q, G, \mathsf{PM})$ are shown below.

match	relevant set
PM_1	$\{\mathsf{DB}_1, \mathsf{PRG}_1, \mathsf{ST}_1, \mathsf{ST}_2\}$
PM_2	$\{\mathsf{DB}_2, \mathsf{DB}_3, \mathsf{PRG}_2, \mathsf{PRG}_3, \mathsf{PRG}_4, \mathsf{ST}_2, \mathsf{ST}_3, \mathsf{ST}_4\}$
PM_i $(i \in [3, 4])$	$\{\mathsf{DB}_2, \mathsf{DB}_3, \mathsf{PRG}_2, \mathsf{PRG}_3, \mathsf{ST}_3, \mathsf{ST}_4\}$

One may verify that $S = \{\mathsf{PM}_2, \mathsf{PM}_3\}$ or $S = \{\mathsf{PM}_2, \mathsf{PM}_4\}$ is a top-2 relevant match set, *i.e., S* reaches more matches in G than any other 2-match set for PM. The total relevance $\delta_r(S) = \delta_r(\mathsf{PM}, \mathsf{PM}_2) + \delta_r(\mathsf{PM}, \mathsf{PM}_3) = 8 + 6 = 14$. □

The need for studying topKP is evident: instead of inspecting possibly large set $M_u(Q, G, u_o)$, we want to find top-k elements that are most relevant to our search.

3.2 Match Diversity

We next introduce a simple metric for result diversity [30]. As observed in [2, 35], it is important to diversify (social) search results to avoid repeated recommendations for similar elements (see Example 1), advocate elements in different groups and to cover elements with new information.

Diversity function. To characterize the diversity of a match set, we define a distance function to measure the "dissimilarity" of two matches. Given two matches v_1 and v_2 of a query node u, we define their *distance* $\delta_d(v_1, v_2)$ to be:

$$\delta_d(v_1, v_2) = 1 - \frac{|R_{(u,v_1)} \cap R_{(u,v_2)}|}{|R_{(u,v_1)} \cup R_{(u,v_2)}|}.$$

The distance function $\delta_d()$ computes the number of distinct matches that two matches of u_o may impact. The larger $\delta_d(v_1, v_2)$ is, the more dissimilar v_1 and v_2 are. It indicates the social diversity between the matches. Observe that the function constitutes a *metric*. For any matches v_1, v_2 and v_3 of u_o, (1) $\delta_d(v_1, v_2) = \delta_d(v_2, v_1)$, and (2) it satisfies the triangle inequality, *i.e.,* $\delta_d(v_1, v_2) \leq \delta_d(v_1, v_3) + \delta_d(v_3, v_2)$.

Example 5: Given G and Q in Fig. 1, we have the following: (1) $\delta_d(\mathsf{PM}_3, \mathsf{PM}_4) = 0$; this suggests that PM_3 and PM_4 have impact on exactly the same group of people in G, *i.e.,* they cannot be distinguished in terms of "social impact"; and (2) $\delta_d(\mathsf{PM}_1, \mathsf{PM}_2) = \frac{10}{11}$, $\delta_d(\mathsf{PM}_2, \mathsf{PM}_3) = \frac{1}{4}$, $\delta_d(\mathsf{PM}_1, \mathsf{PM}_3) = 1$. Thus PM_1 and PM_3 are most dissimilar to each other, as they are related to two completely different groups of people. □

3.3 Match Diversification

It is recognized that search results should be relevant, and at the same time, be as diverse as possible [13,35]. Based on $\delta_r()$ and $\delta_d()$ we next introduce a diversification function.

Diversification function. On a match set $S = \{v_0, \ldots, v_k\}$ of the output node u_o, the diversification function $F()$ is defined as

$$F(S) = (1-\lambda) \sum_{v_i \in S} \delta'_r(u_o, v_i) + \frac{2 \cdot \lambda}{k-1} \sum_{v_i \in S, v_j \in S, i<j} \delta_d(v_i, v_j),$$

where $\lambda \in [0,1]$ is a parameter set by users, $\delta'_r(u_o, v_i)$ is a normalized relevance function defined as $\frac{\delta_r(u_o, v_i)}{C_{u_o}}$, and C_{u_o} is the total number of the candidates of all those query nodes u' to which u_o can reach in Q. Here a node v' in G is called a *candidate* of a query node u' if $L(v') = f_v(u')$, *i.e.,* they share the same label. The diversity metric is scaled down with $\frac{2 \cdot \lambda}{k-1}$, since there are $\frac{k \cdot (k-1)}{2}$ numbers for the difference sum, while only k numbers for the relevance sum.

The function $F()$ is a minor revision of max-sum diversification introduced by [13]. It is a bi-criteria objective function to capture both relevance and diversity. It strikes a balance between the two with a parameter λ that is controlled by users, as a trade-off between the two [35].

Diversified top-k matching problem. Based on the function $F()$, we next state the diversified top-k matching problem, denoted by topKDP. Given G, Q with output node u_o, a positive integer k, and a parameter $\lambda \in [0,1]$, it is to find a set of k matches $S \subseteq M_u(Q, G, u_o)$ such that

$$F(S) = \underset{S' \subseteq M_u(Q,G,u_o)}{\arg\max} F(S'),$$

i.e., for all k-element sets $S' \subseteq M_u(Q, G, u_o)$, $F(S) \geq F(S')$. In contrast to topKP that is to maximize relevance only, topKDP is to find a set of k matches from $M_u(Q, G, u_o)$ such that the bi-criteria diversification function is maximized.

Example 6: Recall graph G and pattern Q from Fig. 1. One can verify that (a) when $\lambda = 0$, *i.e.,* when only relevance is considered, a top-2 set is $\{\mathsf{PM}_2, \mathsf{PM}_3\}$; and (b) when $\lambda = 1$, *i.e.,* when the users only care about diversity, a top-2 set is $\{\mathsf{PM}_1, \mathsf{PM}_3\}$. Moreover, (c) when $\frac{4}{33} < \lambda < 0.5$, $\{\mathsf{PM}_1, \mathsf{PM}_2\}$ makes a top-2 diversified match set, (d) when $\lambda \leq \frac{4}{33}$, $\{\mathsf{PM}_2, \mathsf{PM}_3\}$ is the best choice; and (e) when $\lambda \geq 0.5$, $\{\mathsf{PM}_1, \mathsf{PM}_3\}$ turns out to be the best diversified match set. □

3.4 Generalized Top-k Matching

We next generalize $\delta_r()$ and $\delta_d()$ to define generic relevance and distance functions, based on which we characterize *generalized (diversified) top-k matching* problems.

Generalized ranking functions. For a match v of a pattern node u, we use a generalized relevant set $R^*(u, v)$ to represent the set of descendants of v in G that are "relevant" to u or its descendants (denoted as $R(u)$) in Q. We denote by $M(Q, G, R(u))$ the matches of the nodes in $R(u)$.

(1) We consider a class of generic relevance functions, which are arbitrary monotonically increasing polynomial-time computable (PTIME) functions defined in terms of $R(u)$ and $R^*(u, v)$. We refer to such functions as *generalized relevance functions* $\delta^*_r(u, v)$. Accordingly, the relevance function of a match set S, denoted by $\delta^*_r(S)$, is a monotonically increasing PTIME function of $\delta^*_r(u, v)$, for each $v \in S$.

(2) A generalized distance function $\delta^*_d(v_1, v_2)$ of two matches v_1 and v_2 can be any PTIME computable function *metric* defined with $R^*(u, v_1)$ and $R^*(u, v_2)$. Given a match set S, the generalized diversification function $F^*(\cdot)$ is defined as

$$F^*(S) = (1-\lambda)\delta^*_r(S) + \frac{2 \cdot \lambda}{k-1} \sum_{v_i \in S, v_j \in S, i<j} \delta^*_d(v_i, v_j),$$

where $\lambda \in [0,1]$ is a parameter set by users.

One may verify that $\delta_r()$, $\delta_d()$ and $F()$ given earlier are special cases of $\delta^*_r()$, $\delta^*_d()$ and $F^*()$, respectively. Moreover, $\delta^*_r()$ and $\delta^*_d()$ are able to express a variety of ranking functions commonly used in *e.g.,* social/information networks [22, 24] and Web search [28], including the following:

Ranking functions	Types	Formulations
Preference attachment [24]	relevance	$\lvert R(u)\rvert * \lvert R^*(u,v)\rvert$
Common neighbors [22]	relevance	$\lvert M(Q,G,R(u)) \cap R^*(u,v)\rvert$
Jaccard Coefficient [28]	relevance	$\frac{\lvert M(Q,G,R(u)) \cap R^*(u,v)\rvert}{\lvert M(Q,G,R(u)) \cup R^*(u,v)\rvert}$
Neighborhood diversity [23]	distance	1- $\frac{\lvert R^*(u,v_1) \cap R^*(u,v_2)\rvert}{\lvert V\rvert}$
Distance-based diversity [36]	distance	$1 - \frac{1}{d(v_1,v_2)}$ ($d(v_1, v_2)$ is the distance between v_1 and v_2), or 1 if $d(v_1, v_2)$=∞.

Generalized top-k matching. Given Q with output node u_o, graph G and an integer k, the *generalized* topKP *(resp.* topKDP*) problem* is to find a subset $S \subseteq M_u(Q, G, u_o)$ of k matches, which maximizes $\delta^*_r(S)$ (resp. $F^*(S)$).

Remarks. A function $f(S)$ over a set S is called *submodular* if for any subset $S_1 \subseteq S_2 \subset S$ and $x \in S \setminus S_2$, $f(S_1 \cup \{x\})$ - $f(S_1) \geq f(S_2 \cup \{x\})$ - $f(S_2)$. Note that our diversification functions are *not* necessarily submodular. For example, $F(\cdot)$ (Section 3.3) is not submodular. Indeed, one may verify that $F(S_1 \cup \{v\})$ - $F(S_1) \leq F(S_2 \cup \{v\})$ - $F(S_2)$, although $F(\cdot)$ contains a submodular component $\delta_r(\cdot)$.

To simplify the discussion, we present algorithms for topKP (Section 3.1) and topKDP (Section 3.3). Nonetheless, we show that the algorithms can be readily extended to support generalized top-k matching stated above.

4. FINDING TOP-K MATCHES

We next develop several algorithms for solving the top-k matching problem (topKP) in quadratic time.

The first one, referred to as Match, follows a "find-all-match" strategy. Given Q with output node u_o, G and k, (1) it first finds $M(Q, G)$ with the algorithms in *e.g.,* [11,18]; (2) it then simply computes the relevance for all the matches of u_o, and selects k most relevant matches. One may verify that the algorithm is in $O((|Q| + |V|)(|V| + |E|))$ time.

This algorithm, however, always computes the entire $M(Q, G)$ and is costly for big G. We can rectify this by using "early termination" algorithms. In contrast to Match, these algorithms stop as soon as top-k matches are identified, without computing the entire $M(Q, G)$.

Proposition 2: *For given Q, G and an integer k,* topKP *can be solved by early-termination algorithms.* □

These algorithms leverage a sufficient condition for early termination. For a query node u, we denote as $\mathsf{can}(u)$ the set of all the candidates v of u, *i.e., v* has the same label as u. We use $l(u, v)$ and $h(u, v)$ to denote a *lower bound* and *upper bound* of $\delta_r(u, v)$, respectively, *i.e.,* $l(u, v) \leq \delta_r(u, v) \leq h(u, v)$. Then one can easily verify the following.

Proposition 3: *A k-element set $S \subseteq \mathsf{can}(u_o)$ is a set of top-k matches of u_o if (1) each v in S is a match of u_o, and (2)* $\min_{v \in S}(l(u_o, v)) \geq \max_{v' \in \mathsf{can}(u_o) \setminus S}(h(u_o, v'))$. □

That is, the smallest lower bound of the matches in S is no less than the largest upper bound of those in $\mathsf{can}(u_o) \setminus S$. We use this condition to decide whether S is a top-k match set.

Input: A DAG pattern $Q = (V_p, E_p, f_v, u_o)$,
data graph G, and a positive integer k.
Output: A top-k match set of u_o.

1. min-heap $\mathsf{S} := \emptyset$; termination := false;
2. **for each** $u \in V_p$ **do**
3. compute topological rank $r(u)$; initialize $\mathsf{can}(u)$;
4. **for each** $v \in \mathsf{can}(u)$ **do** initialize $v.T$;
5. **while** (termination = false) **do**
6. select a set of unvisited candidates $S_c \subseteq \mathsf{can}(u)$ of query nodes u in Q, where $r(u) = 0$;
7. **if** $S_c \neq \emptyset$ **then**
8. $\langle G, \mathsf{S}\rangle := \mathsf{AcyclicProp}(Q, S_c, G, \mathsf{S})$;
9. check the termination condition and update termination;
10. **else** termination:= true;
11. **return** S;

Figure 2: Algorithm TopKDAG

We also use the notion of ranks. For a graph G, the *strongly connected component graph* $\mathsf{G_{SCC}}$ is a DAG obtained by collapsing each strongly connected component SCC of G into a single node. We use v_{SCC} to denote the SCC node containing v and E_{SCC} as the set of edges between SCC nodes. The *topological rank* $r(v)$ of a node v in G is defined as (a) $r(v) = 0$ if v_{SCC} is a leaf in $\mathsf{G_{SCC}}$ (*i.e.,* with outdegree 0), and otherwise, (b) $r(v) = \max\{(1 + r(v')) \mid (v_{\mathsf{SCC}}, v'_{\mathsf{SCC}}) \in E_{\mathsf{SCC}}\}$.

Based on these notations and Proposition 3, we provide two algorithms for topKP as a constructive proof of Prop. 2, when Q is a DAG pattern (Section 4.1) and a cyclic pattern (Section 4.2), respectively. The detailed proofs are in [1].

4.1 Algorithm for Acyclic Patterns

We start with an algorithm for topKP when Q is a DAG pattern, denoted as TopKDAG. To simplify the discussion, we assume that the output node u_o is a "root" of Q, *i.e.,* it has no parent, and it can reach all the query nodes in Q. We will discuss the case for "non-root" u_o shortly.

We use the condition given in Proposition 3 to achieve early termination. For each candidate v of a query node u in Q, TopKDAG dynamically maintains a vector $v.T$, which contains (1) a Boolean equation $v.\mathsf{bf}$ of the form $X_v = f$, where f is a Boolean formula that indicates whether v is a match of u; (2) a subset $v.\mathsf{R}$ of its relevant set $R_{(u,v)}$ and (3) integers $v.l$ and $v.h$ to estimate the lower and upper bound of $\delta_r(u, v)$, respectively. Instead of computing $M(Q, G)$, TopKDAG computes a set of matches for some query nodes, and iteratively updates the vectors of the other candidates by "propagating" the partially evaluated results.

Algorithm. Algorithm TopKDAG is shown in Fig. 2. It has two stages: *initialization* and *propagation*, given as follows.

(1) *Initialization* (lines 1-4). TopKDAG first initializes (a) a min-heap S to maintain the matches of u_o ranked by $v.l$, and (b) a Boolean variable termination for the termination condition (line 1). For each query node u in Q, it then computes its topological rank $r(u)$, initializes data structures for all the candidates of u (lines 2-4).

The vector $v.T$ is initialized as follows (line 4). For each candidate v of u, (1) if $r(u) = 0$, then v is already a match, thus TopKDAG sets $v.\mathsf{bf}$ as $X_v = \mathsf{true}$, $v.\mathsf{R} = \emptyset$, and $v.l = v.h = 0$; (2) otherwise, $v.\mathsf{R} = \emptyset$, $v.l = 0$, and $v.\mathsf{bf}$ is set as $X_v = \bigwedge_{(u,u_i)\in E_p} (\bigvee_{v_i \in \mathsf{can}(u_i)} X_{v_i})$, for each child v_i of v. Intuitively, X_v is true iff for each child u_i of u, v has a child v_i that matches u_i. In addition, $v.h = C_u(v)$ (Section 3).

(2) Propagation (lines 5-10). In the propagation, TopKDAG (1) checks whether some candidates become matches of u_o, and (2) updates the lower and upper bounds of the candidates, until either the termination condition is satisfied, or all the matches are identified. It iteratively propagates the known matches and their relevance to evaluate Boolean equations of candidates, and adjusts their lower and upper bounds. Using a greedy heuristic, it selects a set S_c of candidates of query nodes ranked 0 (line 6), which is a *minimal set* that includes all the children of those candidates of query nodes with rank 1. Note that S_c is already a match set since each node in S_c is a leaf. If S_c is not empty, *i.e.,* there exist unvisited matches (line 7), TopKDAG then propagates $v.T$ to all the ancestors v' of v and updates $v'.T$ and S, by invoking procedure AcyclicProp (line 8). If the condition specified in Prop. 3 holds, or if S_c is empty, termination is set true (line 9-10). It returns S as the result (line 11).

Procedure AcyclicProp. Given a set S_c of matches, AcyclicProp (not shown) updates G and S as follows. For each match v of u in S_c, where X_v is true, and for each pattern edge (u', u), it identifies all the parents v' of v that are candidates of u', and updates $v'.T$ as follows: (1) $v'.\mathsf{bf}$ is re-evaluated with $X_v = \mathsf{true}$; (2) if $X_{v'}$ becomes true, then for each child v'' of v' of which $X_{v''}$ is true, $v'.\mathsf{R} := v'.\mathsf{R} \cup v''.\mathsf{R} \cup \{v''\}$; (3) if $X_{v'}$ is true, $v'.l$ is set as $|v'.\mathsf{R}|$ after $v'.\mathsf{R}$ is updated; intuitively, only when v is determined to be a match, its lower bound can be "safely" estimated by $v'.\mathsf{R}$; (4) $v'.h$ is set to be $|v'.\mathsf{R}|$ as soon as for all children v'' of v', none of $v''.h$ is changed further; and (5) if $v'.\mathsf{bf}$ no longer has Boolean variables that are not instantiated, $v.l = v.h$, *i.e.,* $R_{(u,v)}$ is determined now. If $v'.T$ is updated, v' is added in a queue for further propagation.

During the process, AcyclicProp inserts a new match v into the min-heap S if it has less than k matches, or replaces a match v'' in S with v', which is a new match of u_o and is not in S, if $v'.l > v''.h$. It then returns updated S and G.

Example 7: Consider graph G given in Fig. 1 and a DAG pattern Q_1 with edge set $\{(\mathsf{PM}, \mathsf{DB}), (\mathsf{PM}, \mathsf{PRG}), (\mathsf{PRG}, \mathsf{DB})\}$, where $u_o = \mathsf{PM}$. When Q_1 is issued on G, TopKDAG identifies the top-1 match for u_o as follows.

(1) For initialization (lines 1-4), TopKDAG sets the vectors $v.T = \langle v.\mathsf{bf}, v.\mathsf{R}, v.l, v.h\rangle$ for (parts of) candidates as follows.

v	v.T = $\langle v.\mathsf{bf}, v.\mathsf{R}, v.l, v.h\rangle$
PM_2	$\langle X_{\mathsf{PM}_2} = (X_{\mathsf{PRG}_3} \vee X_{\mathsf{PRG}_4}) \wedge X_{\mathsf{DB}_2}, \emptyset, 0, 3\rangle$
PM_3	$\langle X_{\mathsf{PM}_3} = X_{\mathsf{PRG}_3} \wedge X_{\mathsf{DB}_2}, \emptyset, 0, 2\rangle$
PRG_j $(j \in [3,4])$	$\langle X_{\mathsf{PRG}_j} = X_{\mathsf{DB}_2}, \emptyset, 0, 1\rangle$
DB_k $(k \in [1,3])$	$\langle X_{\mathsf{DB}_k} = \mathsf{true}, \emptyset, 0, 0\rangle$

(2) In the propagation stage, AcyclicProp selects S_c as *e.g.,* a candidate $\{\mathsf{DB}_2\}$ for the query node DB ranked 0 in Q_1. It then starts the propagation, which update the vectors as:

v	v.T = $\langle v.\mathsf{bf}, v.\mathsf{R}, v.l, v.h\rangle$
PM_2	$\langle X_{\mathsf{PM}_2} = \mathsf{true}, \{\mathsf{PRG}_3, \mathsf{PRG}_4, \mathsf{DB}_2\}, 3, 3\rangle$
PM_3	$\langle X_{\mathsf{PM}_3} = \mathsf{true}, \{\mathsf{PRG}_3, \mathsf{DB}_2\}, 2, 2\rangle$
PRG_j $(j \in [3,4])$	$\langle X_{\mathsf{PRG}_j} = \mathsf{true}, \{\mathsf{DB}_2\}, 1, 1\rangle$

One can verify that PM_2 is determined to be a match of PM after a single iteration. Better still, the early termination condition is satisfied: $\mathsf{PM}_2.l$ is 3, which is already the largest relevance value. Hence, TopKDAG returns PM_2 directly. □

Correctness. Algorithm TopKDAG correctly computes S as a top-k match set for u_o based on $\delta_r()$. (1) It always terminates. In each **while** iteration (lines 5-10), a set of unvisited candidates S_c is checked. TopKDAG terminates

either when the termination condition is true, or when S_c is empty, *i.e.,* all matches have been found. (2) TopKDAG returns S that consists of either top-k matches by Prop. 3, or all matches of u_o when u_o has less than k matches.

Complexity. The initialization (lines 1-4) takes $O(|Q||G|)$ time, by using an index. For each node v in G, the index records the numbers of its descendants with a same label, and efficiently estimates $v.h$ by aggregating the numbers. It takes in total $O(|V|(|V|+|E|))$ time to propagate changes and update vectors (lines 5-10). Min-heap S can be maintained in $O(|V|\log k)$ time in total (line 8). Checking early termination can be done in $O(1)$ time (line 9), by using a max-heap to record the upper bounds of those candidates of u_o. Thus TopKDAG takes $O(|Q||G| + |V|(|V| + |E|) + |V|\log k)$ time *in the worst case*, *i.e.,* $O(|Q||G| + |V|(|V| + |E|))$ as $\log k$ is often far smaller than $|Q|$ (see [1] for details).

Early termination. Algorithm TopKDAG has the early termination property. More specifically, it combines the evaluation and ranking in *a single process*, and terminates as soon as top-k matches are identified based on Proposition 3. That is, it computes top-k matches for u_o *without* computing and sorting the entire $M_u(Q, G, u_o)$. As will be verified in Section 6, while TopKDAG has the same worst-case complexity as Match, it substantially outperforms Match.

Algorithm TopKDAG can also be extended to support u_o that is not a root node. Besides the termination condition (Prop. 3), it simply checks whether there *exists* a match for all query nodes in Q. One can verify that the correctness and complexity results hold for the extended TopKDAG, as well as the early termination property (see [1] for details).

4.2 Algorithm for Cyclic Patterns

To cope with a cyclic pattern Q, we next provide an algorithm for topKP, denoted as TopK, by extending TopKDAG. Given Q, topKP first computes the strongly connected component graph Q_{SCC} of Q (Section 4.1). Treating Q_{SCC} as a DAG pattern, it then conducts initialization and bottom-up propagation along the same lines as TopKDAG. It terminates as soon as the condition of Proposition 3 is satisfied.

In contrast to TopKDAG, however, TopK has to deal with *nontrivial nodes* in Q, *i.e.,* those nodes u whose corresponding SCC node u_{SCC} contains more than one query node of Q. TopK first verifies whether a candidate matches the nontrivial node. It then employs a *fixpoint strategy* to propagate relevance changes: when a candidate v is known to be a match of u, relevance changes caused by v is propagated to the matches of those query nodes *in* u_{SCC} *only*, to adjust their vectors. The propagation proceeds until a fixpoint is reached, *i.e.,* when no vector can be updated in the propagation for all the candidates of the query nodes in this u_{SCC}.

Algorithm. Similar to TopKDAG, TopK works in two steps, *i.e.,* initialization and propagation. In the *initialization* step, (a) it defines the rank $r(u)$ of a query node u of Q to be the rank $r(u_{SCC})$ of the node u_{SCC} in Q_{SCC}, and (b) for each candidate $v \in \mathsf{can}(u)$ with $r(u) = 0$, if u_{SCC} contains a single node u, it assigns a vector $v.T = \langle X_v = \mathsf{true}, \emptyset, 0, 0\rangle$ to v; otherwise, $v.T$ is initialized in the same way as in TopKDAG.

In the *propagation* step, TopK selects a set S_c of candidates, such that for each node v in S_c, there exist an SCC node u_{SCC} ranked 0 in Q_{SCC} and a query node $u \in u_{SCC}$, where u and v have the same label. For each candidate $v \in \mathsf{can}(u)$ of a query node $u \in u_{SCC}$, it first verifies whether

Procedure SccProcess

Input: pattern $Q = (V_p, E_p, f_v, u_o)$, graph $G = (V, E, L)$, node v_c, a nontrivial node $u_{SCC} \in Q_{SCC}$, and a min-heap S.
Output: Updated $\langle G, S\rangle$.

1. stack $V_A := \emptyset$; termination := false;
2. push v_c onto V_A;
3. **while** $V_A \neq \emptyset$ and termination = false **do**
4. node $v := V_A$.pop(); $X_v :=$ true;
5. **for each** $(v', v) \in E$ and $(u', u) \in E_p$ **do**
   /*$v \in \mathsf{can}(u)$ for $u \in u_{SCC}$, and $v' \in \mathsf{can}(u')$ */
6. update $v'.T$;
7. **if** $X_{v'}$ is evaluated to true **then**
8. **if** $v' \neq v_c$ **then** V_A.push(v');
9. **else if** $v' = v_c$ **then**
10. update $v_i.T$ for each $v_i \in V_A$;
11. **if** $u' = u_o$ **then** update S;
12. check the termination condition; update termination;
13. **if** termination = true **then break** ;
14. **if** v.bf $\neq$ true **then** restore v'.bf for each visited node v';
15. **return** $\langle G, S\rangle$;

Figure 3: Procedure SccProcess

v matches u via procedure SccProcess, when u_{SCC} contains more than one query node. After validity checking of v, if v.bf is evaluated true, *i.e.,* v has children as matches of each child of u, TopK then propagates relevance as follows: (a) if u_{SCC} contains u only, the propagation process from v is the same as in procedure AcyclicProp (Section 4.1); (b) otherwise, it employs the similar strategy as procedure SccProcess (which takes v as an "entry" node v_c in its input) and propagates relevance to matches of nodes in u_{SCC} only.

Procedure SccProcess. The procedure is given in Fig. 3. It takes among the input a candidate v_c as an "entry" node. It uses a stack V_A to perform propagation, and a Boolean variable termination to indicate termination (line 1-2). Utilizing V_A, it performs a reversed depth-first traversal of G starting from v at the top of V_A (lines 3-13). For each $v' \in \mathsf{can}(u')$ encountered (line 5), where u' is a query node, SccProcess updates $v'.T$ in the same way as in TopKDAG (line 6). If v'.bf can be evaluated to be true (line 7), (1) if v' is not v_c, v' is pushed onto the stack to continue the reversed depth-first traversal (line 8). (2) otherwise (line 9), one can verify that all the nodes in stack V_A are valid matches, since they correspond to query nodes in u_{SCC}. Hence for each $v_i \in V_A$, it updates $v_i.T$ by letting $v_i.R := v_i.R \cup V_A$ and $v_i.l := |v_i.R|$ (line 10). Furthermore, if u' is the output node, it updates S with new matches (line 11), checks the termination condition (Proposition 3), and terminates if the condition holds (lines 12-13).

If v.bf is still false after the **while** loop, v is not a match. Thus for each node v' visited in the loop, v'.bf is restored to its original form saved earlier (line 14). SccProcess returns the updated vectors and S for further propagation (line 15).

Example 8: Recall graph G and pattern Q from Fig. 1. TopK finds top-2 matches for PM as follows. It first computes Q_{SCC} of Q, which has a nontrivial node DB_{SCC} containing DB and PRG. It starts with *e.g.,* a set of candidates S_c={$\mathsf{ST}_3, \mathsf{ST}_4$}. When the propagation reaches candidates of DB_{SCC}, (parts of) their vectors are shown as below.

v	v.T = $\langle v$.bf, v.R, $v.l$, $v.h\rangle$
DB_2	$\langle X_{\mathsf{DB}_2} = X_{\mathsf{PRG}_2} \wedge$ true, $\emptyset$, 0, 6$\rangle$
PRG_2	$\langle X_{\mathsf{PRG}_2} = X_{\mathsf{DB}_3} \wedge$ true, $\emptyset$, 0, 6$\rangle$
DB_3	$\langle X_{\mathsf{DB}_3} = X_{\mathsf{PRG}_3} \wedge$ true, $\emptyset$, 0, 6$\rangle$
PRG_3	$\langle X_{\mathsf{PRG}_3} = X_{\mathsf{DB}_2} \wedge$ true, $\emptyset$, 0, 6$\rangle$
PRG_4	$\langle X_{\mathsf{PRG}_4} = X_{\mathsf{DB}_2} \wedge$ (true $\vee X_{\mathsf{ST}_2}$), $\emptyset$, 0, 7$\rangle$

TopK then invokes SccProcess to check the validity of those candidates for the query nodes in $\mathsf{DB}_{\mathsf{SCC}}$. Assume that SccProcess first pushes DB_3 onto stack V_A (line 2). It then propagates X_{DB_3} = true upwards, updates PRG_2.bf to X_{PRG_2} = true and pushes PRG_2 onto V_A. Similarly, DB_2.bf and PRG_3.bf are updated to X_{DB_2} = true and X_{PRG_3} = true successively. When DB_3 is encountered, SccProcess updates $\mathsf{DB}_3.T$ to $\langle X_{\mathsf{db}_3} = \mathsf{true}, \{\mathsf{ST}_3, \mathsf{ST}_4, \mathsf{DB}_2, \mathsf{DB}_3, \mathsf{PRG}_2, \mathsf{PRG}_3\}, 6, 6\rangle$. It then updates vector for each node in V_A (line 10). After this, the vectors of the candidates for PMs are as follows ($i \in [3, 4]$):

v	v.T = $\langle v.\mathsf{bf}, v.\mathsf{R}, v.l, v.h\rangle$
PM_2	$\langle X_{\mathsf{PM}_2} = \mathsf{true}, \{\mathsf{ST}_3, \mathsf{ST}_4, \mathsf{DB}_2, \mathsf{DB}_3, \mathsf{PRG}_2, \mathsf{PRG}_3\}, 6, 7\rangle$
PM_i	$\langle X_{\mathsf{PM}_i} = \mathsf{true}, \{\mathsf{ST}_3, \mathsf{ST}_4, \mathsf{DB}_2, \mathsf{DB}_3, \mathsf{PRG}_2, \mathsf{PRG}_3\}, 6, 6\rangle$

Observe that after a single propagation, the termination condition in Proposition 3 is satisfied: $\mathsf{PM}_2.l = \mathsf{PM}_3.l = 6$, which are no less than $\mathsf{PM}_1.h$, *i.e.,* 4 and $\mathsf{PM}_4.h$. Thus TopK returns PM_2 and PM_3 as top-2 matches. □

Correctness & Complexity. SccProcess always reaches a fixpoint, at which it correctly finds matches of the query nodes in SCC nodes. Indeed, (a) SccProcess stops when either the termination condition is true (line 12), or X_v is updated to true (and never changed back to false) for all the (finitely many) candidates v of u in u_{SCC} (lines 7, 14); and (b) SccProcess changes X_v to true iff v is a match. The correctness of propagation and termination condition for TopK can be verified along the same lines as for TopKDAG. For the complexity, the initialization is in $O(|Q||G|)$ time, and propagation is in $O(|V|(|V| + |E|))$ time. Thus TopK is in $O(|Q||G| + |V|(|V| + |E|))$ time. Moreover, along the same lines as for TopKDAG, one can verify that TopK has *the early termination property*, by Proposition 3.

From the analysis above Proposition 2 follows.

Generalized top-k matching. The result below shows that our techniques can be readily applied to generalized relevance functions given in Section 3 (see [1] for a proof).

Proposition 4: TopKDAG *and* TopK *can be extended for generalized* topKP*, with the early termination property.* □

The techniques can be easily extended to patterns with multiple output nodes that are not necessarily "roots" [1].

5. FINDING DIVERSIFIED MATCHES

In this section, we investigate the diversified top-k matching problem (topKDP). In contrast to topKP that is based on $\delta_r()$ alone, the topKDP problem is intractable. The main result of this section is as follows.

Theorem 5: *The* topKDP *problem is (1)* NP*-complete (decision problem); (2)* 2*-approximable in* $O(|Q||G| + |V|(|V| + |E|))$ *time, and (3) has a heuristic in* $O(|Q||G| + |V|(|V| + |E|))$ *time, but with the early termination property.* □

We prove Theorem 5(1) as follows. The decision problem of topKDP is in NP, since one can guess a k-element set S and then check whether $S \subseteq M_u(Q, G, u_o)$ and $F(S) \geq B$ in PTIME. To show the lower bound, observe that by setting $\lambda = 1$, topKDP includes the K-diverse set problem [35] as its special case, which is known to be NP-hard [35]; hence topKDP is NP-hard. Thus, topKDP is NP-complete.

Recent results for the *max-sum diversification* [4] suggests that topKDP is, in general, nontrivial to approximate. Given a set U with a distance function δ_o over the elements in U, the problem is to find a k-element subset S, which maximizes $F_o(S) = f(S) + c \sum_{u,v \in S}(\delta_o(u, v))$, where $f(S)$ is a submodular function (see Section 3). Our diversification function $F(\cdot)$ is in the form of $F_o(S)$, if normalized by $(1-\lambda)$. It is shown in [4] that no polynomial time algorithm can approximate $F_o(\cdot)$ within $\frac{e}{e-1}$, assuming P ≠ NP. In addition, it is shown that the diversification problem for submodular functions is approximable within $(1 - \frac{1}{e})$ [17]. However, $F(\cdot)$ is *not* submodular, as remarked earlier in Section 3.

Despite the hardness, we provide two algorithms for topKP. (1) One is an approximation algorithm to compute diversified matches with approximation ratio 2, hence proving Theorem 5(2) (Section 5.1). (2) The approximation algorithm may be costly on large graphs, however. Thus we give a heuristic algorithm for topKDP with the early termination property (Section 5.2), verifying Theorem 5(3).

5.1 Approximating Diversification

We show Theorem 5 (2) by presenting an approximation algorithm, denoted by TopKDiv. In a nutshell, TopKDiv iteratively chooses a pair of matches that "maximally" introduces diversity and relevance to the selected matches, following a greedy strategy. This is done by (1) "rounding down" the diversification function $F(\cdot)$ with a revised $F'(\cdot)$, and (2) finding a solution that maximizes $F'(\cdot)$, which in turn guarantees an approximation ratio for $F(\cdot)$. This technique is commonly used for optimization problems [13, 34].

Algorithm. Given Q, G and an integer k, algorithm TopKDiv identifies a set S' of k matches of u_o, such that $F(S') \geq \frac{F(S^*)}{2}$, where S^* is an optimal set of k matches that maximizes $F(\cdot)$. That is, TopKDiv approximates topKDP with approximation ratio 2.

TopKDiv first initializes a min-heap S for top-k matches, and an integer counter i. It then computes $M(Q, G)$, the relevance $\delta'_r(u_o, v)$ and diversity $\delta_d(v, v')$ for all matches $v, v' \in M_u(Q, G, u_o)$. Next, it iteratively selects two matches $\{v_1, v_2\}$ that maximize $F'(v_1, v_2) = \frac{1-\lambda}{k-1}$ $(\delta'_r(u_o, v_1)$ $+ \delta'_r(u_o, v_2)) + \frac{2\lambda}{k-1}$ $\delta_d(v_1, v_2)$; it then adds (resp. removes) them to S (resp. from $M_u(Q, G, u_o)$). This process repeats $\frac{k}{2}$ times. If k is odd, |S| is $k - 1$; TopKDiv then greedily selects a match v to maximize $F(\mathsf{S} \cup \{v\})$ Finally, it returns S We present the details of TopKDiv in [1].

Example 9: Given graph G and pattern Q of Fig. 1, and assuming $\lambda = 0.5$, TopKDiv finds top-2 diversified matches for PM as follows. (1) It first computes $M_u(Q, G, u_o)$ $= \{\mathsf{PM}_i \mid i \in [1, 4]\}$, and the relevance and diversity of those PM nodes (lines 1-2). (2) It then greedily selects a pair (v_1, v_2) of matches that maximizes $F'(v_1, v_2)$ $= 0.5(\delta'_r(u_o, v_1) + \delta'_r(u_o, v_2)) + \delta_d(v_1, v_2)$ (lines 3-9). Then $\{\mathsf{PM}_1, \mathsf{PM}_3\}$ is selected, since $F'(\mathsf{PM}_1, \mathsf{PM}_3) = 1.45$ is maximum. Thus TopKDiv returns this pair. When $\lambda = 0.5$, this pair is a top-2 match based on $F(\cdot)$ (see Example 6). □

Correctness & Complexity. TopKDiv returns S, which consists of k matches for u_o if u_o has at least k matches, and all matches of u_o otherwise. We next show that TopKDiv approximates topKDP with ratio 2. To see this, note that an instance of TopKDiv can be transformed to an instance of the *Maximum Dispersion problem* (MAXDISP) [16]. The problem MAXDISP is to find a subgraph G'_c induced by a k-node set V_c from a weighted complete graph G_c, with

the maximum sum of node and edge weights. Given a match set $M_u(Q, G, u_o)$, we construct a complete graph G_c in which each node (simply denoted as v) represents a match $v \in M_u(Q, G, u_o)$ with a weight $\delta_r(u_o, v)$, and each edge (v_1, v_2) carries a weight $\delta_d(v_1, v_2)$. Given a set of k matches $S \subseteq M_u(Q, G, u_o)$ and the corresponding k node set V_c in G_c, we define cost $F'(V_c) = \sum_{v_i, v_j \in S, i<j} F'(v_i, v_j)$, where $F'(v_i, v_j)$ is given in TopKDiv (line 4). One may verify that $F'(V_c) = (k-1) \cdot \frac{1-\lambda}{k-1} \sum_{v_i \in S} \delta'_r(u_o, v_i) + \frac{2\lambda}{k-1} \sum_{v_1, v_2 \in S} \delta_d(v_i, v_j) = F(S)$, where $F(\cdot)$ is the diversification function (Section 3). Thus, S contains top-k matches if and only if V_c maximizes $F'(V_c)$. Note that TopKDiv simulates a greedy 2-approximation algorithm for MAXDISP [16]. Hence, it returns a set S of k matches such that $F(S) \geq \frac{1}{2} \cdot F(S^*)$, where S^* refers to the optimal top-k matches, *i.e.,*TopKDiv approximates topKDP with ratio 2.

For the complexity, it takes $O((|Q|+|V|)(|V|+|E|))$ time to compute $M_u(Q, G, u_o)$, and the relevance and distance values (line 1). It takes in total $O(\frac{k}{2}|V|^2)$ time to update S with the greedy strategy (lines 3-9). Thus, TopKDiv is in $O(|Q||G| + |V|(|V| + |E|))$ time in the worst case, since k is typically treated a small constant. Hence, despite the necessary computation for diversifying the ranks, TopKDiv does not incur substantially extra cost compared to the algorithms for computing top-k matches based on $\delta_r()$ alone.

This analysis above completes the proof of Theorem 5(2).

5.2 Early Termination Heuristics

Algorithm TopKDiv requires all the matches in $M(Q, G)$ to be computed, which may not be efficient for large graphs. To rectify this we present a heuristic algorithm for topKDP, denoted as TopKDH, with *the early termination property.*

Algorithm. Similar to TopK (Section 4), TopKDH (not shown) uses a min-heap S to maintain top-k matches; and initializes the same vector for each candidate. It updates the vectors via propagation to check the termination condition (Proposition 3). In contrast to TopK, TopKDH utilizes a greedy strategy to choose matches for u_o. In each propagation, it collects a set S' of matches of u_o with updated vectors. It then updates S as follows: (a) if $|S| + |S'| \leq k$, S = S $\cup$ S'; (b) otherwise, TopKDH iteratively replaces $v \in$ S with v' to maximize $F''(S \setminus \{v\} \cup \{v'\})$ - $F''(S)$; here $F''()$ revises $F(\cdot)$ by replacing $\delta_r(u_o, v)$ with $v.l/C_{u_o}$, and $\delta_d(v_i, v_j)$ with 1- $\frac{|v_i.R \cap v_j.R|}{|v_i.R \cup v_j.R|}$; it then removes v' from S'. Intuitively, TopKDH always selects matches that "maximally" diversifies S. These steps repeat until S' is $\emptyset$ or $|S| = k$.

Example 10: Consider graph G and pattern Q from Fig. 1. Let $\lambda = 0.1$, TopKDH finds top-2 diversified matches for PM as follows. It first selects $S_c = \{ST_3, ST_4\}$, and adjusts the vectors of the candidates. After the propagation, it selects $\{PM_2, PM_3\}$ as top-2 matches, which maximizes $F''()$ as $0.9 * \frac{13}{11} + 0.2 * \frac{1}{7} = 1.1$. Now the condition of Proposition 3 is satisfied. Hence, TopKDH returns $\{PM_2, PM_3\}$, which is indeed a top-2 pair when $\lambda = 0.1$ (see Example 6). □

Correctness & Complexity. Algorithm TopKDH differs from TopK only in that it does extra computation to select the matches. One may verify its correctness along the same lines as the argument for TopK given earlier. For the complexity, the extra computation takes $O(k|V|^2)$ time in total. Thus TopKDH is still in $O(|Q||G| + |V|(|V| + |E|))$ time.

TopKDH *terminates early*: it processes as many matches as TopK does in propagation, and it *stops as soon as* the termination condition of Proposition 3 is satisfied.

The analysis completes the proof of Theorem 5(3).

Generalized diversified top-k matching. Our diversified matching algorithms can be easily extended for generalized diversified function $F^*(\cdot)$ (Section 3.4), preserving the nice properties, *e.g.,* early termination and approximation ratio. We defer the detailed algorithms and proofs to [1].

Proposition 6: *Algorithm* TopKDH *(resp.* TopKDiv*) can be extended for generalized* topKDP*, with the early termination property (resp. preserving approximation ratio* 2*).* □

6. EXPERIMENTAL EVALUATION

We next experimentally verify the effectiveness and efficiency of our top-k graph pattern matching algorithms, using real-life and synthetic data (see [1] for more results).

Experimental setting. We used the following datasets.

(1) Real-life graphs. We used three real-life graphs.

(a) *Amazon* (`http://snap.stanford.edu/data/index.html`) is a product co-purchasing network with 548,552 nodes and 1,788,725 edges. Each node has attributes such as title, group and sales rank. An edge from product x to y indicates that people who buy x also buy y.

(b) *Citation* (`http://www.arnetminer.org/citation/`) contains 1,397,240 nodes and 3,021,489 edges, in which nodes represent papers with attributes (*e.g.,* title, authors, year and publication venue), and edges denote citations.

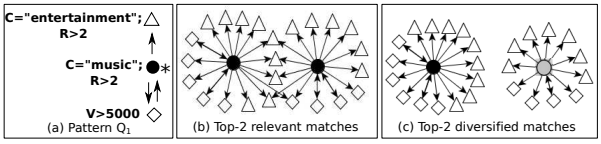
(c) *YouTube* (`http://netsg.cs.sfu.ca/youtubedata/`) is a network with 1,609,969 nodes and 4,509,826 edges. Each node is a video with attributes (*e.g.,* (A)ge, (C)ategory, (V)iews, (R)ate). An edge (x, y) indicates that the publisher of video x recommends a related video y.

(2) Synthetic data. We designed a generator to produce synthetic graphs $G = (V, E, L)$, controlled by the number of nodes $|V|$ and edges $|E|$, where L are assigned from a set of 15 labels. We generated synthetic graphs following the linkage generation models [12]: an edge was attached to the high degree nodes with higher probability (see [1] for details). We use $(|V|, |E|)$ to denote the size of G.

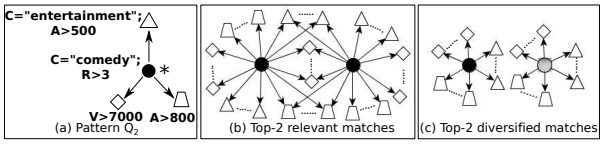
(3) Pattern generator. We also implemented a generator for graph patterns $Q = (V_p, E_p, f_v, u_o)$, controlled by four parameters: $|V_p|$, $|E_p|$, label f_v from the same Σ, and the output node u_o. We denote as $(|V_p|, |E_p|)$ the size $|Q|$ of Q. For synthetic graphs, we manually constructed a set of 9 patterns including 4 DAG patterns and 5 cyclic patterns.

For *Amazon*, we identified 10 cyclic patterns to search products with conditions specified on attributes (*e.g.,* title, category) and their connections with other products. *Citation* is a DAG, and we designed 14 DAG patterns to find papers and authors in computer science. For *Youtube*, we found 10 cyclic patterns, where each node carried search conditions for finding videos, *e.g.,* category is "music".

Two such patterns on *Youtube* are shown in Figures 4(a) and 4(b). (a) The cyclic pattern Q_1 in Fig. 4(a) is to find top-2 videos in category "music" (marked with "$*$" as the output node) with rating $R > 2$ (out of 5), which are related to "entertainment" videos with $R > 2$ and have been watched more than 5000 times ($V > 5000$). (b) Similarly,



(a) Pattern (I) and top-2 results on *Youtube*



(b) Pattern (II) and top-2 results on *Youtube*

Figure 4: Case study

the DAG pattern Q_2 in Fig. 4(b) is to identify top-2 "comedy" videos with recommendation requirements.

(4) Implementation. We implemented the following algorithms, all in Java: (1) our top-k algorithms TopKDAG for DAG patterns and TopK for cyclic patterns; (2) algorithm $\mathsf{TopK}_{\mathsf{nopt}}$ (resp. $\mathsf{TopKDAG}_{\mathsf{nopt}}$), a naive version of TopK (resp. TopKDAG) that randomly selects S_c to start propagation, rather than choosing a minimal set S_c that covers those candidates of query nodes of rank 1 (see Section 4); (3) algorithm Match for top-k matching, to compare with TopKDAG and TopK; (4) the approximation algorithm TopKDiv and heuristic algorithm TopKDH (resp. TopKDAGDH) to find diversified top-k matches for general (resp. DAG) patterns.

All the experiments were repeated 5 times on a 64bit Linux Amazon EC2 Instance with 3.75 GB of memory and 2 EC2 Compute Unit, and the average is reported here.

Experimental results. We next present our findings.

Exp-1: Effectiveness of top-k matching. We first evaluated the effectiveness of our top-k matching algorithms, *i.e.,*TopKDAG (resp. TopK) and its naive version $\mathsf{TopKDAG}_{\mathsf{nopt}}$ (resp. $\mathsf{TopK}_{\mathsf{nopt}}$), compared to Match. We measured their effectiveness by (1) counting the number of the matches $|M_u^t(Q,G,u_o)|$ of u_o inspected by them, and (2) computing a *match ratio* $\mathsf{MR} = \frac{|M_u^t(Q,G,u_o)|}{|M_u(Q,G,u_o)|}$.

We compared MR of these algorithms over the three real life datasets: (1) TopK, $\mathsf{TopK}_{\mathsf{nopt}}$ and Match on *Youtube* by varying $|Q|$ (Fig. 5(a)), (2) TopKDAG, $\mathsf{TopKDAG}_{\mathsf{nopt}}$ and Match on *Citation* by varying $|Q|$ (Fig. 5(b)), and (3) TopK, $\mathsf{TopK}_{\mathsf{nopt}}$ and Match on *Amazon* by varying k (Fig. 5(c)). The algorithms performed consistently on different datasets, and hence we do not show all the results here. Moreover, (a) Match always finds all the matches, *i.e.,* its MR = 1, and is thus not shown; and (b) *Citation* is a DAG, and thus only TopKDAG, $\mathsf{TopKDAG}_{\mathsf{nopt}}$ and Match were tested on *Citation* for DAG patterns.

Performance for cyclic patterns. Fixing $k = 10$, we varied $|Q|$ from $(4, 8)$ to $(8, 16)$ for *Youtube*. The results are reported in Fig. 5(a). Observe the following: (1) TopK and $\mathsf{TopK}_{\mathsf{nopt}}$ effectively reduce excessive matches. For instance, when $|Q| = (4, 8)$, while Match had to compute all the matches (≥ 180), TopK only inspected 88, *i.e.,* MR = 47%. On average, MR for TopK is 45%, and is 54% for $\mathsf{TopK}_{\mathsf{nopt}}$. Indeed, TopK terminates early: it finds top-k matches without computing all the matches. (2) TopK (on average) inspects 16% less matches than $\mathsf{TopK}_{\mathsf{nopt}}$ due to the greedy selection heuristics: more relevant matches are likely to be identified earlier in the propagation process (Section 4).

Performance for DAG patterns. Fixing $k = 10$, we varied DAG pattern size $|Q|$ from $(4, 6)$ to $(10, 15)$ on *Citation*. As shown in Fig. 5(b), (1) TopKDAG inspects much less matches than Match. For example, its MR is only 34% when $|Q| = (8, 12)$, and is 40% on average. (2) On average, TopKDAG examined 18% less matches than $\mathsf{TopKDAG}_{\mathsf{nopt}}$. The reduction in MR is more evident for DAG patterns than for cyclic patterns because DAG patterns are less restrictive and hence, $M(Q, G)$ tends to be larger.

Varying k. Fixing pattern size $|Q| = (4, 8)$, we varied k from 5 to 30 in 5 increments, and reported MR for TopK and $\mathsf{TopK}_{\mathsf{nopt}}$ on *Amazon*. As shown in Fig. 5(c), the match ratio MR of TopK (rsep. $\mathsf{TopK}_{\mathsf{nopt}}$) increased from 42% (resp. 46%) to 69% (resp. 77%) when k was increased from 5 to 30. Indeed, when k becomes larger, more matches have to be identified and examined, for both TopK and $\mathsf{TopK}_{\mathsf{nopt}}$.

Case study. We manually inspected top-k matches returned by our algorithms on the real-life data, and confirmed that the matches were indeed sensible in terms of their relevance. For instance, Figures 4(a) and 4(b) depict the top-2 matches (circle nodes) and graphs induced by their relevant sets *w.r.t.* patterns Q_1 and Q_2 given earlier, respectively, on *Youtube*. These were confirmed to be the top-2 matches.

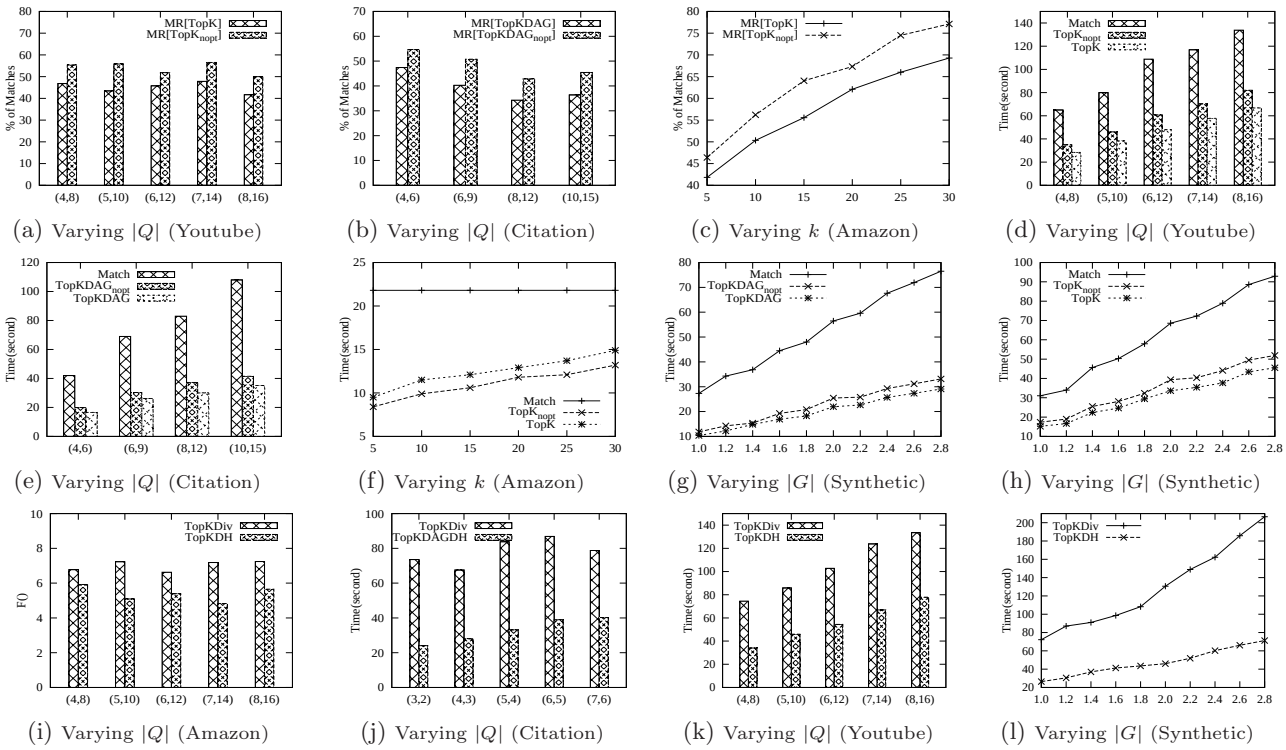
Exp-2: Efficiency and scalability of top-k matching. We next evaluated the efficiency of the algorithms. In the same settings as in Exp-1, we report the performance of (1) TopK, $\mathsf{TopK}_{\mathsf{nopt}}$ and Match on *Youtube* by varying $|Q|$ (Fig. 5(d)), (2) TopKDAG, $\mathsf{TopKDAG}_{\mathsf{nopt}}$ and Match on *Citation* by varying $|Q|$ (Fig. 5(e)), and (3) TopK, $\mathsf{TopK}_{\mathsf{nopt}}$ and Match on *Amazon* by varying k (Fig. 5(f)). We also evaluated their scalability with synthetic data.

Efficiency for cyclic patterns. The results for cyclic patterns on *Youtube* are shown in Fig. 5(d), which are consistent with Fig. 5(a): (1) TopK and $\mathsf{TopK}_{\mathsf{nopt}}$ always outperform Match: TopK (resp. $\mathsf{TopK}_{\mathsf{nopt}}$) takes 52% (resp. 64%) of the time of Match on average. (2) On average, TopK improves $\mathsf{TopK}_{\mathsf{nopt}}$ by 18%. (3) While all the algorithms take more time for larger patterns, Match is more sensitive to $|Q|$ than TopK, because Match spends 98% of its time on computing all the matches and their relevance, which heavily depend on $|Q|$.

Efficiency for acyclic patterns. As shown in Fig. 5(e), the results for DAG patterns on *Citation* are consistent with Fig. 5(d). (1) TopKDAG (resp. $\mathsf{TopKDAG}_{\mathsf{nopt}}$) outperforms Match by 64% (resp. 56%) on average, and (2) TopKDAG improves $\mathsf{TopKDAG}_{\mathsf{nopt}}$ by 16%. The improvement over Match is more evident for DAG patterns than for cyclic patterns (Fig. 5(d)) because (a) MR is smaller for DAG patterns, and (b) TopKDAG does not need fixpoint computation.

Varying k. On *Amazon*, Figure 5(f) reports the efficiency results in the same setting as in Fig. 5(c): (1) Match is insensitive to k, as it computes the entire $M_u(Q, G, u_o)$. (2) TopK and $\mathsf{TopK}_{\mathsf{nopt}}$ outperform Match, but are sensitive to the change of k. Indeed, the benefit of early termination degrades when k gets larger and more matches need to be identified. Nonetheless, k is small in practice, and TopK is less sensitive than $\mathsf{TopK}_{\mathsf{nopt}}$, as its selection strategy allows early discovery of top matches, reducing the impact of k.

In addition, we found that TopK and TopKDAG perform better for patterns with (a) smaller "height" (*i.e.,* the largest rank of the pattern node), (b) output nodes with smaller ranks, and (c) less candidates. We present the details in [1].



(a) Varying $|Q|$ (Youtube) (b) Varying $|Q|$ (Citation) (c) Varying k (Amazon) (d) Varying $|Q|$ (Youtube)

(e) Varying $|Q|$ (Citation) (f) Varying k (Amazon) (g) Varying $|G|$ (Synthetic) (h) Varying $|G|$ (Synthetic)

(i) Varying $|Q|$ (Amazon) (j) Varying $|Q|$ (Citation) (k) Varying $|Q|$ (Youtube) (l) Varying $|G|$ (Synthetic)

Figure 5: Performance evaluation

Scalability. We also evaluated the scalability of these algorithms using large synthetic datasets. Fixing $|Q| = (4, 6)$ for DAG patterns and $k = 10$, we varied $|G|$ from $(1M, 2M)$ to $(2.8M, 5.6M)$, and tested TopKDAG, TopKDAG$_{\text{nopt}}$ and Match. As shown in Fig. 5(g), the results tell us the following: (a) TopKDAG and TopKDAG$_{\text{nopt}}$ scale well with $|G|$, and better than Match; they account for only 38.1% and 43.2% of the running time of Match, respectively; and (b) TopKDAG takes 87% of the running time of TopKDAG$_{\text{nopt}}$. These are consistent with the results on real-life graphs.

Fixing k=10, we used cyclic patterns with size $|Q| = (4, 8)$, and tested the scalability of TopK, TopK$_{\text{nopt}}$ and Match. As shown in Fig. 5(h), the results are consistent with Fig. 5(g): TopK (resp. TopK$_{\text{nopt}}$) accounts for 49% (resp. 56%) of the cost of Match for cyclic patterns. A closer examination of the above results also tells us that our algorithms do much better than their worst-case complexity, due to early termination.

Exp-3: Diversified top-k matching. Finally we evaluated (1) the effectiveness of TopKDiv and TopKDH, (2) the efficiency of TopKDiv, TopKDH and TopKDAGDH, as well as (3) their scalability using large synthetic data.

Effectiveness. Observe that (a) the MR of TopKDiv is always 1, as it requires $M_u(Q, G, u_o)$ to be computed, and (2) the MR of TopKDH (resp. TopKDAGDH) is the same as that of TopK (resp. TopKDAG), since they only differ in match selection strategy (see Section 5). Thus, the comparison of MR's for TopKDiv, TopKDH and TopKDAGDH is consistent with the results in Figures 5(a) and 5(b). Instead, we are interested in how well TopKDH and TopKDAGDH, as heuristics, "approximate" the optimal diversified matches.

Fixing $\lambda = 0.5$ and $k = 10$, we tested $F(S)$ and $F(S')$ on *Amazon* by varying $|Q|$, where S (resp. S') is the set of top-k diversified matches found by TopKDiv (resp. TopKDH), and $F(\cdot)$ is the diversification function given in Section 3. As shown in Fig. 5(i), (1) $F(S) \geq F(S')$, as expected since TopKDiv has approximation ratio 2, while TopKDH is a heuristic. (2) However, TopKDH is not bad: $F(S')$ is 77% of $F(S)$ in the worst case. Thus TopKDH, on average, "approximately" finds a set S' with $F(S') \geq \frac{1}{2.6}$ of the optimal value, which is comparable to the performance of TopKDiv.

Case study. We also manually checked the top-2 diversified matches found by TopKDH for Q_1 and Q_2 of Figures 4(a) and 4(b), respectively. As also shown in Fig. 4, TopKDH correctly replaced one of the top-2 relevant matches with another (shadowed node) that made the match set diverse.

Efficiency. On *Citation*, we tested the efficiency of TopKDiv and TopKDAGDH, by fixing $k = 10$ and varying $|Q|$ from $(3, 2)$ to $(7, 6)$. As shown in Fig. 5(j), (1) TopKDAGDH takes only 42% of the time of TopKDiv on average, but (2) TopKDiv is less sensitive to $|Q|$ than TopKDAGDH, due to the tradeoff between the extra time incurred by larger Q for TopKDiv to compute $M(Q, G)$ and the reduced time for selecting diversified matches from smaller $M(Q, G)$.

Fixing $k = 10$, we evaluated the efficiency of TopKDiv vs. TopKDH on *Youtube* by using the same patterns as for TopK in Exp-2 (Fig. 5(d)). Figure 5(k) shows the results, which are consistent with Fig. 5(j) for DAG patterns on *Citation*.

We also found that both algorithms are not sensitive to the change of λ. Specifically, TopKDiv takes slightly less time when $\lambda = 0$, as it degrades to Match (see [1] for details).

Scalability. We also evaluated the scalability of TopKDiv and TopKDH, in the same setting as in Fig. 5(h). As shown in Fig. 5(l), (1) both algorithms scale well with $|G|$, and (2) The running time of TopKDH is less sensitive than that of TopKDiv. Indeed, TopKDiv spends more time on computing $M(Q, G)$ for larger G, and its running time grows faster than that of TopKDH. TopKDH seldom demonstrates its worst case complexity, due to early termination.

Summary. (1) The revised graph pattern matching effectively reduces excessive matches: TopKDAG (resp. TopK, TopKDH) only examines 40% (resp. 45%) of matches in $M(Q, G)$ on average. (2) Our early-termination algorithms outperform Match, which is based on traditional matching. Indeed, TopKDAG (resp. TopK) takes on average 36% (resp. 52%) of the time of Match for DAG (resp. cyclic) patterns. (3) Our algorithms effectively identify most relevant and diversified matches for output nodes, and scale well with k and the sizes of Q and G. (5) Our optimization technique improves the efficiency of the top-k matching algorithms by 16% (resp. 18%) for DAG (resp. cyclic) patterns.

7. CONCLUSION

We have introduced and studied the (diversified) top-k graph pattern matching problems. We have revised graph patterns by supporting a designated output node, and defined functions to measure match relevance and diversity, as well as a bi-criteria objective function based on both. We have established the complexity for these problems. In addition, we have provided algorithms for computing top-k matches based on relevance alone, and for finding diversified top-k matches, with properties such as *constant approximation ratios* and *early termination*. As verified analytically and experimentally, our methods indeed remedy the limitations of prior matching algorithms, by eliminating excessive matches and improving efficiency on big real-life social graphs. Better still, they can be employed to support a wide range of ranking functions commonly used in practice.

The work is a first step toward effective top-k matching on big social data. We are currently experimenting with various real-life graphs, ranking functions, and patterns (with multiple output nodes [1]), to fine-tune our diversification objective function. We are also exploring optimization techniques to further reduce the number of matches examined by our algorithms. The ultimate goal is to make graph pattern matching feasible on big social data. To this end, we are developing distributed top-k matching algorithms on graphs that are partitioned, distributed and possibly compressed.

Acknowledgments. Fan and Wang are supported in part by the 973 Programs 2012CB316200 and 2014CB340302, Guangdong Innovative Research Team Program 2011D005 and the Shenzhen Peacock Program 1105100030834361 of China, as well as EPSRC EP/J015377/1, UK.

8. REFERENCES

[1] Full version. *http://homepages.inf.ed.ac.uk/s0944873/topk.pdf*.
[2] O. Alonso, M. Gamon, K. Haas, and P. Pantel. Diversity and relevance in social search. In *DDR*, 2012.
[3] M. Bendersky, D. Metzler, and W. Croft. Learning concept importance using a weighted dependence model. In *WSDM*, 2010.
[4] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, pages 155–166. ACM, 2012.
[5] J. Brynielsson, J. Högberg, L. Kaati, C. Martenson, and P. Svenson. Detecting social positions using simulation. In *ASONAM*, 2010.
[6] S. Cohen, B. Kimelfeld, G. Koutrika, and J. Vondrák. On principles of egocentric person search in social networks. In *VLDS*, 2011.
[7] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. *DivQ*: Diversification for keyword search over structured databases. In *SIGIR*, 2010.
[8] A. Dovier, C. Piazza, and A. Policriti. A fast bisimulation algorithm. In *CAV*, 2001.
[9] R. Fagin. Combining fuzzy information from multiple systems. *JCSS*, 58(1):83–99, 1999.
[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.
[11] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1), 2010.
[12] S. Garg, T. Gupta, N. Carlsson, and A. Mahanti. Evolution of an online social aggregation network: an empirical study. In *IMC*, 2009.
[13] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.
[14] G. Gou and R. Chirkova. Efficient algorithms for exact ranked twig-pattern matching over graphs. In *SIGMOD*, 2008.
[15] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *SIGMOD*, 2003.
[16] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21(3):133–137, 1997.
[17] J. He, H. Tong, Q. Mei, and B. Szymanski. Gender: A generic diversified ranking algorithm. In *Advances in Neural Information Processing Systems 25*, pages 1151–1159, 2012.
[18] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.
[19] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
[20] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, 2003.
[21] T. Lappas, K. Liu, and E. Terzi. A survey of algorithms and systems for expert location in social networks. In *Social Network Data Analytics*. 2011.
[22] E. A. Leicht, P. Holme, and M. E. J. Newman. Vertex similarity in networks. *Phys. Rev. E*, 73:026120, 2006.
[23] R.-H. Li and J. X. Yu. Scalable diversified ranking on large graphs. In *ICDM*, pages 1152–1157, 2011.
[24] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.
[25] Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD*, 2007.
[26] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive processing of top-k queries in XML. In *ICDE*, 2005.
[27] M. Morris, J. Teevan, and K. Panovich. What do people ask their social networks, and why? A survey study of status message q&a behavior. In *CHI*, 2010.
[28] M. E. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):025102, 2001.
[29] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
[30] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11), 2012.
[31] R. Schenkel, T. Crecelius, M. Kacimi, S. Michel, T. Neumann, J. X. Parreira, and G. Weikum. Efficient top-k querying over social-tagging networks. In *SIGIR*, 2008.
[32] L. G. Terveen and D. W. McDonald. Social matching: A framework and research agenda. In *ACM Trans. Comput.-Hum. Interact.*, 2005.
[33] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, 2007.
[34] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.
[35] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. J. Traina, and V. J. Tsotras. On query result diversification. In *ICDE*, 2011.
[36] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. de Castro Reis, and B. A. Ribeiro-Neto. Efficient search ranking in social networks. In *CIKM*, pages 563–572, 2007.
[37] A. Wagner, T. Duc, G. Ladwig, A. Harth, and R. Studer. Top-k linked data query processing. In *ESWC*, 2012.
[38] L. Zou, L. Chen, and Y. Lu. Top-k subgraph matching query in a large graph. In *Ph.D. workshop in CIKM*, 2007.