Diversified Top-k Graph Pattern Matching

Wenfei Fan1,2 Xin Wang1 Yinghui Wu3
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 \( u_0 \). Given \( G \) and \( Q \), it is to find those nodes in \( M(Q,G) \) that match \( u_0 \), instead of the large set \( M(Q,G) \). (2) We study two classes of functions for ranking the matches: relevance functions \( \delta_1(\cdot) \) based on, e.g., social impact, and distance functions \( \delta_2(\cdot) \) to cover diverse elements. (3) We develop two algorithms for computing top-k matches of \( u_0 \) based on \( \delta_1(\cdot) \), 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 \( \delta_1(\cdot) \) and \( \delta_2(\cdot) \). 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.

(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 \( u_0 \) as “query focus” [3], i.e., we just want those nodes in a social graph \( G \) that are matches of \( u_0 \) 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 \( u_0 \), it is to find top-k matches of \( u_0 \) in \( M(Q,G) \), ranked by a quality function. The users only need to check \( k \) matches of \( u_0 \) 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 \( u_0 \) 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),
A company issues a graph search query to find PMs who supervised both DBs and PRGs, and moreover, (1) the DB worked under the PM 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 a “*” 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. 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), (DB, DB), (PRG, PRG), (ST, ST), 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 of ST, (i.e., $[1, 4]$) 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 are returned (i.e., $[1, 4]$).

We can do better with top-$k$ matching. When $k = 2$, we find two top-ranked PM’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’s, an algorithm with the early termination property identifies the PM’s without computing the entire $M(Q, G)$.

To rank the matches PM’s of PM, one may consider the following criteria. (1) Social impact [21]. Observe that PM2 can reach more people than any other PM, i.e., PM2 has collaborated with more people. Thus PM2 has stronger social impact. (2) Social diversity [2, 35]. Consider match sets $\{PM_1, PM_2\}$ and $\{PM_3, PM_4\}$. While PM2 and PM3 worked with the same people, PM4 and PM2 are quite “dissimilar” since they covered different groups of people. Putting these together, $\{PM_1, PM_2\}$ 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 $M_o(Q, G, u_o)$, 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 $b_r()$ that measure the relevance of a match, and distance functions $b_d()$, which measure the “dissimilarity” of two matches. Based on both, we define a bi-criteria (balanced by a parameter $\lambda$) 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 $b_r()$. 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 $\lambda$. 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 search- ing [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 $O(1 − \frac{1}{k})$ 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 $(u, v)$ 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 $\Sigma$. Intuitively, the node labels denote e.g., keywords, social roles, ratings [11].

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

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

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 $f_Q(u)$ of a query node $u$ specifies a search condition: a node $v$ in $G$ can match only if $L(v) = f_Q(u)$.

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 $O(|V_Q| + |V||E_Q|)$ time [11], where $|M(Q,G)|$ is bounded by $O(|V||V_Q|)$ [18].

We denote $|V_Q| + |E_Q|$ 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 \( Q = (V_Q, E_Q, f_Q, u_Q) \), where \( u_Q \) is a query node in \( V_Q \), labeled with “\( * \)”, referred to as the output node of \( Q \), and \( V_Q, E_Q \) and \( f_Q \) are the same as above.

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

Note that \( M_u(Q,G,u_Q) \) is smaller than \( M(Q,G) \): its size is bounded by \( |V| \) as opposed to \( |V|/|V_Q| \).

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 \( M_u(Q,G,PM) = \{ PM_i \mid i \in [1,4] \} \), 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 \( M_u(Q,G,u_Q) \) 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 \( u_Q \) 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 \( u_Q \). 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 \((u_i,v_i) \in E(Q,G)(i \in [1,n]) \). 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 \( \delta_r(u,v) \) 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 \( \delta_r(u,v) \) values are preferred for relevance.

**Top-\( k \) matching problem.** We now state the top-\( k \) matching problem, denoted by \text{topKP}. Given a graph \( Q \), 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

\[
\delta_r(S) = \arg \max_{S'} \sum_{(u_o,v_i) \in S'} \delta_r(u_o,v_i).
\]

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

That is, \text{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 \( M_u(Q,G,PM) \) are shown below.

<table>
<thead>
<tr>
<th>match</th>
<th>relevant set</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM_1</td>
<td>{DB_1, PRG_1, ST_1, ST_2}</td>
</tr>
<tr>
<td>PM_2</td>
<td>{DB_2, DB_3, PRG_2, PRG_3, PRG_4, ST_3, ST_4}</td>
</tr>
<tr>
<td>PM_(i \in [3,4])</td>
<td>{DB_1, DB_3, PRG_3, PRG_4, ST_3, ST_4}</td>
</tr>
</tbody>
</table>

One may verify that \( S = \{PM_2, PM_3\} \) or \( S = \{PM_2, PM_4\} \) 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 \( \delta_r(S) = \delta_r(PM_2, PM_3) + \delta_r(PM_2, PM_4) = 8 + 6 = 14 \).

The need for studying \text{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 diversity 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 \( u_o \), (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) \( \delta_d(PM_2, PM_4) = 0 \); this suggests that \( PM_2 \) 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) \( \delta_d(PM_2, PM_2) = \frac{10}{17} \), \( \delta_d(PM_2, PM_2) = \frac{2}{7} \), \( \delta_d(PM_1, PM_1) = 1 \). 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 \( S = \{ u_1, \ldots, u_v \} \) of the output node \( u_o \), the diversification function \( F() \) is defined as

\[
F(S) = (1 - \lambda) \sum_{v_i \in S} \delta_i^*(u_o, v_i) + \frac{2 \cdot \lambda}{k - 1} \sum_{v_i \in S, \forall j \in S, i < j} \delta_i(v_i, v_j),
\]

where \( \lambda \in [0, 1] \) is a parameter set by users, \( \delta_i^*(u_o, v_i) \) is a normalized relevance function defined as \( \frac{1 - (u_o, v_i)}{\frac{1}{2}} \), 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 \( u' \) in \( G \) is called a candidate of a query node \( u' \) if \( L(u') = f_o(u') \), i.e., they share the same label. The diversity metric is scaled down with \( -\frac{\lambda}{k - 1} \), since there are \( \frac{k(k-1)}{2} \) 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 \( \lambda \) 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 \( G \), \( Q \), and a parameter \( \lambda \), when the users only care about diversity, a top-2 set is found by users, as a trade-off between the two [35].

**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 \( \{ PM2, PM3 \} \); (b) when \( \lambda = 1 \), i.e., when the users only care about diversity, a top-2 set is \( \{ PM1, PM3 \} \). Moreover, (c) when \( \frac{1}{3} < \lambda < 0.5 \), \( \{ PM1, PM2 \} \) makes a top-2 diversified match set; (d) when \( \lambda < \frac{1}{2} \), \( \{ PM2, PM3 \} \) is the best choice; and (e) when \( \lambda \geq 0.5 \), \( \{ PM1, PM3 \} \) turns out to be the best diversified match set.

### 3.4 Generalized Top-k Matching

We next generalize \( \delta_i() \) and \( \delta_k() \) 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 \( Q \) that are “relevant” to \( u \) or its descendants (denoted as \( R(u) \)) and \( R^*(u, v) \). To refer to such functions as generalized relevance functions \( \delta_i^*(u, v) \). Accordingly, the relevance function of a match set \( S \), denoted by \( \delta_i^*(S) \), is a monotonically increasing PTIME function of \( \delta_i^*(u, v) \), for each \( v \in S \).

(2) A generalized distance function \( \delta_i^*(v_1, v_2) \) 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

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

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

One may verify that \( \delta_i^*(\cdot), \delta_k^*(\cdot) \) and \( F() \) given earlier are special cases of \( \delta_i^*(\cdot), \delta_k^*(\cdot) \) and \( F^*(\cdot) \), respectively. Moreover, \( \delta_i^*(\cdot) \) and \( \delta_k^*(\cdot) \) 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:

<table>
<thead>
<tr>
<th>Ranking functions</th>
<th>Types</th>
<th>Formulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preference attachment [24]</td>
<td>relevance</td>
<td>( R(u) ) and ( R^*(u, v) )</td>
</tr>
<tr>
<td>Common neighbors [22]</td>
<td>relevance</td>
<td>( M(Q, G, R(v)) ) and ( R(v) )</td>
</tr>
<tr>
<td>Jaccard Coefficient [28]</td>
<td>relevance</td>
<td>( R(Q, G, f_o(u); v) ) and ( R(v) )</td>
</tr>
<tr>
<td>Neighborhood diversity [23]</td>
<td>distance</td>
<td>1 - ( \frac{</td>
</tr>
<tr>
<td>Distance-based diversity [36]</td>
<td>distance</td>
<td>( 1 - \frac{</td>
</tr>
</tbody>
</table>

**Generalized top-k matching.** Given a weighted graph \( G \), an integer \( k \), the generalized topKDP (resp. topKDP) problem is to find a subset \( S \subseteq M_o(Q, G, u_o) \) 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 \( S_1 \subseteq S_2 \subseteq S \) and \( x \in S \setminus S_2 \), \( f(S_1 \cup \{ x \}) \geq f(S_1) + f(S_2 \cup \{ x \}) - f(S_2) \). Note that our diversification functions are not necessarily submodular. For example, \( F() \) (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 \( F() \) contains a submodular component \( \delta_i() \).

To simplify the discussion, we present algorithms for topKDP (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 (topK) 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 \( M(Q, G, u_o) \) 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 \( k \), topK can be solved by early-termination algorithms.

These algorithms leverage a sufficient condition for early termination. For a query node \( u \), we denote as \( \text{can}(u) \) 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 an upper bound of \( \delta_i(u, v) \), respectively, i.e., \( l(u, v) \leq \delta_i(u, v) \leq h(u, v) \). Then one can easily verify the following.

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

That is, the smallest lower bound of the matches in \( S \) is no less than the largest upper bound of those in \( \text{can}(u_o) \). We use this condition to decide whether \( S \) is a top-k match set.
We will discuss the case for "non-root" patterns, (a) for each unvisited candidate of query nodes in $Q$, computes topological rank $r(v)$; (b) for each $v$, initializes $v.T$; (c) while $\text{termination} = \text{false}$ do; (d) for each $v$, initializes $\text{can}(v)$. We use $\text{propagate}$ to denote 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 $v$ is a leaf in $\text{GSCC}$ (i.e., with outdegree 0), and otherwise, (b) $r(v) = \max\{(1 + r(v')) | (v,v') \in \text{E}_{\text{SCC}}\}$. Based on these notations and Proposition 3, we provide two algorithms for $\text{topKP}$ 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 $\text{topKP}$ when $Q$ is a DAG pattern, denoted as $\text{TopKDAG}$. 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 $Q$, $\text{TopKDAG}$ dynamically maintains a vector $v.T$, which contains (1) a Boolean equation $v.bf$ of the form $X_v = f$, where $f$ is a Boolean formula that indicates whether $v$ is a match of $u$; (2) a subset $v.R$ of its relevant set $R(u,v)$; and (3) integers $v.l$ and $v.h$ to estimate the lower and upper bound of $r(v,u)$, respectively. Instead of computing $M(Q,G)$, $\text{TopKDAG}$ 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 $\text{TopKDAG}$ is shown in Fig. 2. It has two stages: initialization and propagation, given as follows.

(1) Initialization (lines 1-4). $\text{TopKDAG}$ first initializes (a) a min-heap $S$ to maintain the matches of $u_o$ ranked by $v.l$, and (b) a Boolean variable $\text{termination}$ 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 $\text{TopKDAG}$ sets $v.bf$ as $X_v = \text{true}$, $v.R = \emptyset$, and $v.l = v.h = 0$; (2) otherwise, $v.R = \emptyset$, $v.l = 0$, and $v.bf$ is set as $X_v = \bigwedge_{(u,v) \in \text{E}} \bigvee_{v' \in \text{can}(u)} X_{v'}$, 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, $\text{TopKDAG}$ 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 $S$, 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 $S$ is already a match set since each node in $S$ is a leaf. If $S$ is not empty, i.e., there exist unvisited matches (line 7), $\text{TopKDAG}$ then propagates $v.T$ to all the ancestors $v'$ of $v$ and updates $v'.T$ and $S$, by invoking procedure $\text{AcyclicProp}$ (line 8). If the condition specified in Prop. 3 holds, or if $S$ is empty, termination is set true (line 9-10). It returns $S$ as the result (line 11).

Procedure $\text{AcyclicProp}$. Given a set $S$ of matches, $\text{AcyclicProp}$ (not shown) updates $G$ and $S$ as follows. For each match $v$ of $u$ in $S$, where $X_v$ is true, and for each pattern edge $(v', u)$, it identifies all the parents $v''$ of $v'$. As matches of $u''$, and updates $v'.T$ as follows: (1) $v'.bf$ is re-evaluated with $X_v = \text{true}$; (2) if $X_v$ becomes true, then for each child $v''$ of $v'$ which $X_{v''}$ is true, $v'.R := v'.R \cup \{v''\}$; (3) if $X_v$ is true, $v'.l$ is set as $\{v'.R\}$ after $v'.R$ is updated; intuitively, only when $v$ is determined to be a match, its lower bound can be “safely” estimated by $v'.R$. (4) $v'.h$ is set to be $\{v'.R\}$ as soon as for all children $v''$ of $v'$, none of $v''.h$ is changed further; and (5) if $v.bf$ 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, $\text{AcyclicProp}$ 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 $\text{G}$ given in Fig. 1 and a DAG pattern $Q_2$ with edge set $\{\text{PM, DB}, \{\text{PM, PRG}, \{\text{PRG, DB}\}\}$, where $u_o = \text{PM}$. When $Q_2$ is issued on $G$, $\text{TopKDAG}$ identifies the top-1 match for $u_o$ as follows.

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

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PM2</td>
<td>${X_{PM2} = {X_{PRG} \lor X_{PM1}} \land X_{DB2}, 0, 0, 3}$</td>
</tr>
<tr>
<td>PM3</td>
<td>${X_{PM3} = {X_{PM2} \land X_{DB2}, 0, 2}}$</td>
</tr>
<tr>
<td>PRG2 ($j \in [3, 4]$)</td>
<td>${X_{PRG2} = X_{DB2}, 0, 1}$</td>
</tr>
<tr>
<td>DB2 ($k \in [1, 3]$)</td>
<td>${X_{DB2} = \text{true}, 0, 0, 0}$</td>
</tr>
</tbody>
</table>

(2) In the propagation stage, $\text{AcyclicProp}$ selects $S_e$ as e.g., a candidate $\{DB_2\}$ for the query node DB ranked 0 in $Q_1$. It then starts the propagation, which update the vectors as:

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PM2</td>
<td>${X_{PM2} = \text{true}, {PRG2, PRG2, DB2}, 3, 3}$</td>
</tr>
<tr>
<td>PM3</td>
<td>${X_{PM3} = \text{true}, {PRG3, PRG3, DB2}, 2, 2}$</td>
</tr>
<tr>
<td>PRG3 ($j \in [3, 4]$)</td>
<td>${X_{PRG3} = \text{true}, {DB2}, 1, 1}$</td>
</tr>
</tbody>
</table>

One can verify that PM2 is determined to be a match of PM after a single iteration. Better still, the early termination condition is satisfied: PM2.l is 3, which is already the largest relevance value. Hence, $\text{TopKDAG}$ returns PM2 directly. □

Correctness. Algorithm $\text{TopKDAG}$ correctly computes $S$ as a top-$k$ match set for $u_o$ based on $\delta(.)$. (1) It always terminates. In each while iteration (lines 5-10), a set of unvisited candidates $S_e$ is checked. $\text{TopKDAG}$ terminates
either when the termination condition is true, or when $S_i$ 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 $v \in can(u)$ with $r(u) = 0$, if $u_{SCC}$ contains a single node $u$, it assigns a vector $v.T = (X_v = true, 0, 0, 0)$ to $v$; otherwise, $v.T$ is initialized in the same way as in TopKDAG.

In the propagation step, TopK selects a set $S_v$ of candidates, such that for each node $v$ in $S_v$, 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 $v \in can(u)$ of a query node $u \in u_{SCC}$, it first verifies whether

### Procedure SccProcess

**Input**: pattern $Q = (V_p, E_p, f_p, u_o)$, graph $G = (V, E, L)$, node $u_v$, a nontrivial node $u_{SCC} \in Q_{SCC}$, and a min-heap $S$.

**Output**: Updated $(G, S)$.

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

**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 $v.o.bf$ 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 $u_o$ 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 $v_o$ 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 $v' \in can(u')$ encountered (line 5), where $u'$ is a query node, SccProcess updates $v'.T$ in the same way as in TopKDAG (line 6). If $v.o.bf$ can be evaluated to be true (line 7), (1) if $v'$ is not $v_o$, $v'$ 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 $v_i.T := v.o.R \cup V_A$ and $v_i.T := |v_i.R|$ (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 $v.o.bf$ is still false after the while loop, $v$ is not a match. Thus for each node $v'$ visited in the loop, $v.o.bf$ 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 DBSCC containing DB and PRG. It starts with $e.g.$, a set of candidates $S_v := \{ST_{24}, ST_{34}\}$. When the propagation reaches candidates of DBSCC, (parts of) their vectors are shown as below.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>$(X_{DB2} = X_{PRG}, true, 0, 0)$</td>
</tr>
<tr>
<td>PRG2</td>
<td>$(X_{DB2} = X_{PRG}, true, 0, 0)$</td>
</tr>
<tr>
<td>DB1</td>
<td>$(X_{DB1} = X_{PRG}, true, 0, 0)$</td>
</tr>
<tr>
<td>PRG1</td>
<td>$(X_{DB1} = X_{PRG}, true, 0, 0)$</td>
</tr>
</tbody>
</table>
TopK then invokes \texttt{SccProcess} to check the validity of those candidates for the query nodes in DBSCC. Assume that \texttt{SccProcess} first pushes DB3 onto stack VA (line 2). It then propagates \(X_{DB3} = \text{true} \) upwards, updates \(PRG_{bf} \) to \(X_{PRG_{bf}} = \text{true} \) and pushes \(PRG_{gt} \) onto \(VA \). Similarly, \(DB_{bf} \) and \(PRG_{bf} \) are updated to \(X_{DB_{bf}} = \text{true} \) and \(X_{PRG_{bf}} = \text{true} \) successively. When DB3 is encountered, \texttt{SccProcess} updates DB3.T to \(X_{DB3} = \text{true} \). It then updates vectors for each node in VA (line 10). After this, the vectors of the candidates for PMs are as follows (\(i \in [3,4] \)):

<table>
<thead>
<tr>
<th>(v )</th>
<th>(X_{pm} )</th>
<th>(PRG_{bf} )</th>
<th>(PRG_{gt} )</th>
<th>(DB_{bf} )</th>
<th>(DB_{gt} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM2</td>
<td>(X_{pm} = \text{true} )</td>
<td>(ST, ST, DB, DB, PRG, PRG), (6,7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM3</td>
<td>(X_{pm} = \text{true} )</td>
<td>(ST, ST, DB, DB, PRG, PRG), (6, 6)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Observe that after a single propagation, the termination condition in Proposition 3 is satisfied: \(PM_2.l = PM_3.l = 6 \), which are no less than \(PM_1.h \), i.e., 4 and \(PM_1.h \). Thus TopK returns PM2 and PM3 as top-2 matches.

**Correctness & Complexity.** \texttt{SccProcess} always reaches a fixpoint, at which it correctly finds matches of the query nodes in SCCs. Indeed, (a) \texttt{SccProcess} stops when either the termination condition is true (line 12), or \(X_v \) is updated to \(\text{true} \) (and never changed back to \(\text{false} \)) for all the (finitely many) candidates \(v \) of \(v \) in \(u_{SCC} \) (lines 7, 14); and (b) \texttt{SccProcess} changes \(X_v \) to true if \(v \) is a match. The correctness of propagation and termination condition for TopKP can be verified along the same lines as for TopKDP. For the complexity, the initialization is in \(O(|Q| + |G|) \) time, and propagation is in \(O(|V|(|V| + |E|)) \) time. Thus TopK is in \(O(|Q| + |V|(|V| + |E|)) \) time.

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:** TopKDP 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 \(d.o() \) 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 \(O(|Q| + |V|(|V| + |E|)) \) time, and (3) has a heuristic in \(O(|Q| + |V|(|V| + |E|)) \) 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 \(S \subseteq M_{\alpha}(Q, G, u_a) \) and \(F(S) \geq B \) in \(PTIME \). To show the lower bound, observe that by setting \(\alpha = 1 \), 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 \(d_u \) over the elements in \(U \), the problem is to find a \(k\)-element subset \(S \), which maximizes \(F_u(S) = f(S) + c \sum_{v \in S, v \in G} d_u(v, v) \), where \(f(S) \) is a submodular function (see Section 3). Our diversification function \(F(\cdot) \) is in the form of \(F_u(S) \), if normalized by \((1-\lambda) \). It is shown in [4] that no polynomial time algorithm can approximate \(F_u(\cdot) \) within \(\frac{1}{e} \), assuming \(P \neq NP \). 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 topKDP. (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 \(u_a \), 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 \(d_u(v, v') \) and diversity \(d_v(v, v') \) for all matches \(v, v' \in M_\lambda(Q, G, u_a) \). Next, it iteratively selects two matches \(v_1, v_2 \) to maximize \(F(v_1, v_2) = \frac{1}{2} (d_u(v_1, v_2) + d_v(v_1, v_2) + d_v(v_2, v_1)) \). It then adds (resp. removes) them to \(S \) (resp. from \(M_\lambda(Q, G, u_a) \)). This process repeats \(k \) times. If \(k \) is odd, \(|S| = k - 1 \), TopKDiv then greedily selects a match \(v \) to maximize \(F(S \cup \{v\}) \). 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 \(M_0(Q, G, u_a) = \{PM_i \mid i \in \{1,4\}\} \). Next, it then greedily selects a pair \((v_1, v_2) \) of matches that maximizes \(F(v_1, v_2) = 0.5(d_u(v_1, v_2) + d_v(v_1, v_2)) \) (lines 3-9). Then \(\{PM_1, PM_3\} \) is selected, since \(F(\cdot)(PM_1, PM_3) = 1.45 \) is maximum. Thus TopKDiv returns this pair. When \(\lambda = 0.5 \), this is a top-2 match based on \(F(\cdot) \) (see Example 6).

**Correctness & Complexity.** TopKDiv returns \(S \), which consists of \(k \) matches for \(u_a \) if \(u_a \) has at least \(k \) matches, and all matches of \(u_a \) 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 \(G' \) 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 \( M_u(Q, G, u_x) \), we construct a complete graph \( G_c \) in which each node (simply denoted as \( v \)) represents a match \( v \in M_u(Q, G, u_x) \) with a weight \( \delta_v(u_x, v) \), and each edge \((v_1, v_2)\) carries a weight \( \delta(v_1, v_2) \). Given a set of \( k \) matches \( S \subseteq M_u(Q, G, u_x) \) and the corresponding \( k \) node set \( V_c \) in \( G_c \), we define cost \( F'(V_c) = \sum_{v_1, v_2 \in S, i < j} F'(v_i, v_j) \), where \( F'(v_i, v_j) \) is given in TopKDiv (line 4). One may verify that \( F'(V_c) = (k - 1) \cdot \sum_{v \in G_c} \delta_v(u_x, v) + \sum_{v_1, v_2 \in S} \delta(v_1, v_2) = F(S) \), 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 MAXDASP [16]. Hence, it returns a set \( S \) of \( k \) matches such that \( F(S) \geq \frac{1}{2} \cdot F(S^*) \), where \( S^* \) is 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 \( M_u(Q, G, u_x) \), and the relevance and distance values (line 1). It takes in total \( O(\frac{1}{k} |V|^2) \) 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 \( \delta(\cdot) \) alone.

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

5.2 Early Termination Heuristics

Algorithm TopKDiv requires all the matches in \( M_u(Q, G) \) 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 \( u_x \). In each propagation, it collects a set \( S' \) of matches of \( u_x \) 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 \( F''(\cdot) \) re-initializes \( F(\cdot) \) by replacing \( \delta(v_i, v) \) with \( v_i \cdot L_i \), where \( L_i \) is 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 \(|E|\) to denote the size of \( G \).

5.3 Pattern generator. We also implemented a generator for graph patterns \( G = (V, E, L) \), 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 \(|E|\) to denote the size of \( G \).

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).

(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/youtube.html) 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 \(|V|\) to denote the size of \( G \).

(3) Pattern generator. We also implemented a generator for graph patterns \( G = (V, E, f, u_x) \), controlled by four parameters: \(|V|\), \(|E|\), label \( f \) from the same \( \Sigma \), and the output node \( u_x \). We denote as \(|V|, |E|\) 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.”

We defined a cycle pattern \( P \in \mathcal{P} \) as a directed cycle within a network that satisfies the condition specified on attributes.

Two such patterns on Youtube are shown in Figures 4(a) and 4(b). (a) The cyclic pattern \( P_1 \) in Fig. 4(a) is to find top-2 videos in category “music” (marked with “*” as the output node) with rating \( R \geq 2 \) (out of 5), which are related to “entertainment” videos with \( R \geq 2 \) and have been watched more than 5000 times (\( V > 5000 \)). (b) Similarly,
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 TopKopt (resp. TopKDAGopt), a naive version of TopK (resp. TopKDAG) that randomly selects $S_i$ to start propagation, rather than choosing a minimal set $S_i$ 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 TopKDAGopt (resp. TopKopt), compared to Match. We measured their effectiveness by (1) counting the number of the matches $|M(Q,G,u_i)|$ of $u_i$ inspected by them, and (2) computing a match ratio $\text{MR} = \frac{|M(Q,G,u_i)|}{|M(Q,G,u_i)|}$.

We compared MR of these algorithms over the three real life datasets: (1) TopK, TopKopt and Match on Youtube by varying $|Q|$ (Fig. 5(a)); (2) TopKDAG, TopKDAGopt and Match on Youtube by varying $|Q|$ (Fig. 5(b)); and (3) TopK, TopKopt and Match on Citation by varying $|Q|$ (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, TopKDAGopt and Match were tested on Citation for DAG patterns.

Performance for cyclic patterns. Fixing $k = 10$, we varied DAG pattern size $|Q|$ from $(4, 8)$ 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 TopKDAGopt. 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 TopKopt on Amazon. As shown in Fig. 5(c), the match ratio MR of TopK (resp. TopKopt) 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 TopKopt.

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, TopKopt and Match on Youtube by varying $|Q|$ (Fig. 5(d)), (2) TopKDAG, TopKDAGopt and Match on Citation by varying $|Q|$ (Fig. 5(e)), and (3) TopK, TopKopt 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 TopKopt always outperform Match: TopK (resp. TopKopt) takes 52% (resp. 64%) of the time of Match on average. (2) On average, TopK improves TopKopt 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. TopKDAGopt) outperforms Match by 64% (resp. 56%) on average, and (2) TopKDAG improves TopKDAGopt 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 $M_{u_i}(Q,G,u_i)$. (2) TopK and TopKopt 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 TopKopt, 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].
**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{opt} and Match. As shown in Fig. 5(g), the results tell us the following: (a) TopKDAG and TopKDAG\text{opt} 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{opt}. 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{opt} and Match. As shown in Fig. 5(h), the results are consistent with Fig. 5(g): TopK (resp. TopK\text{opt}) 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 TopKD, (2) the efficiency of TopKDiv, TopKD and TopKDGDH, as well as (3) their scalability using large synthetic data.

**Effectiveness.** Observe that (a) the MR of TopKD is always 1, as it requires $M_s(Q,G, u_s)$ to be computed, and (2) the MR of TopKDGH (resp. TopKDGDH) is the same as that of TopK (resp. TopKDAG), since they only differ in match selection strategy (see Section 5). Thus, the comparison of MRs for TopKD, TopKDGH and TopKDGDH is consistent with the results in Figures 5(a) and 5(b). Instead, we are interested in how well TopKD and TopKDGDH, 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 TopKDDiv (resp. TopKD), 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 TopKD is a heuristic. (2) However, TopKD is not bad: $F(S')$ is 77% of $F(S)$ in the worst case. Thus TopKD, on average, “approximately” finds a set $S'$ with $F(S') \geq \frac{1}{2} F(S)$ 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 TopKD for $Q_1$ and $Q_2$ of Figures 4(a) and 4(b), respectively. As also shown in Fig. 4, TopKD 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 TopKDGDH, by fixing $k = 10$ and varying $|Q|$ from (3, 2) to (7, 6). As shown in Fig. 5(j), (1) TopKDGDH takes only 42% of the time of TopKDiv on average, but (2) TopKDiv is less sensitive to $|Q|$ than TopKDGDH, 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. TopKD 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 $\lambda$. 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 TopKDGH 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 TopKD 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 TopKD. TopKDGH 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, NSFC 61133002, Guangdong Innovative Research Team Program 2011D005 and Shenzhen Peacock Program 1105100030834361 of China, and EPSRC EP/J015377/1, UK.

8. REFERENCES